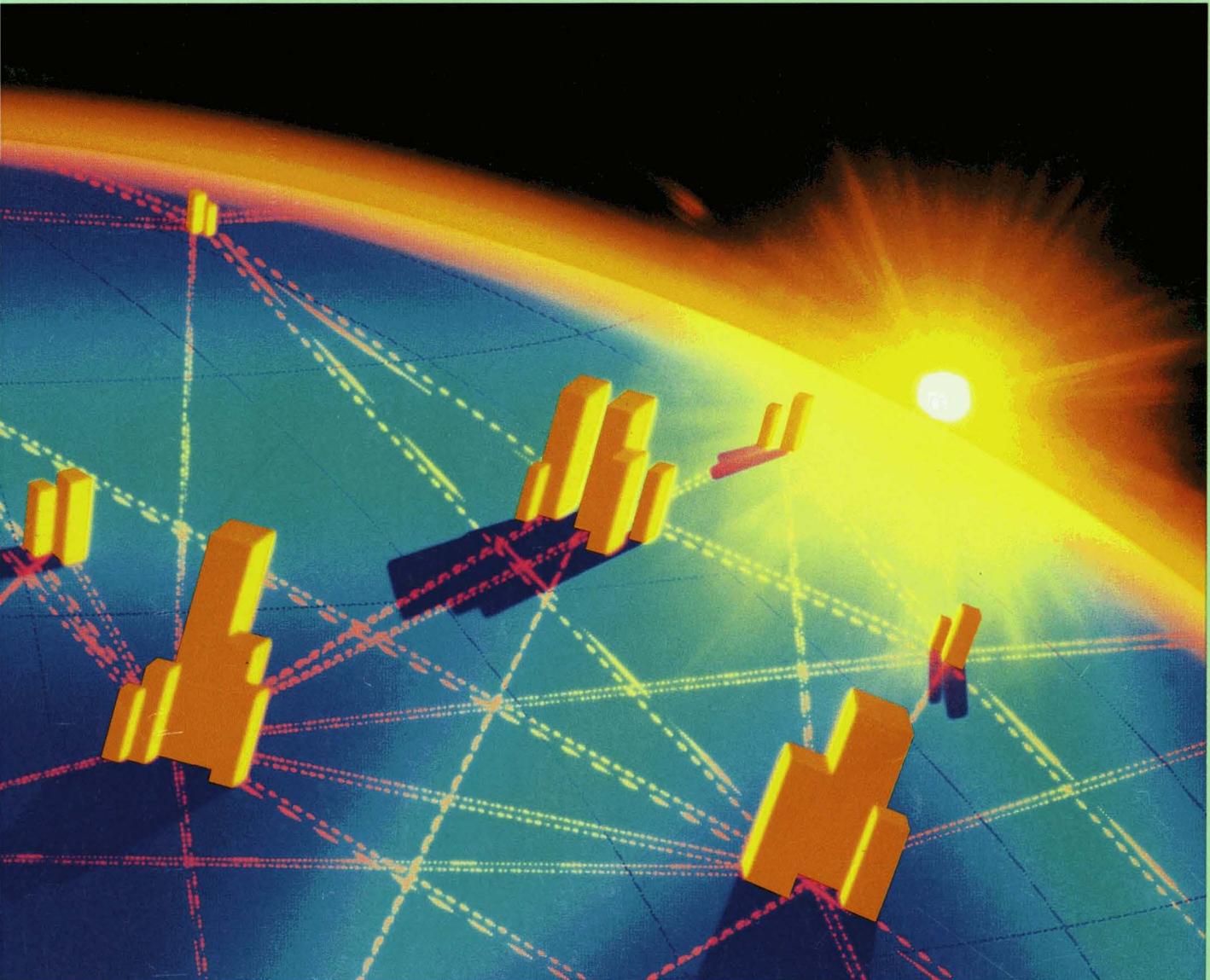


DECnet Open Networking

Digital Technical Journal

Digital Equipment Corporation



Volume 5 Number 1

Winter 1993

Editorial

Jane C. Blake, Editor
Helen L. Patterson, Associate Editor
Kathleen M. Stetson, Associate Editor

Circulation

Catherine M. Phillips, Administrator

Production

Terri Autieri, Production Editor
Anne S. Katzeff, Typographer
Peter R. Woodbury, Illustrator

Advisory Board

Samuel H. Fuller, Chairman
Richard W. Beane
Donald Z. Harbert
Richard J. Hollingsworth
Alan G. Nemeth
Jeffrey H. Rudy
Stan Smits
Michael C. Thurk
Gayn B. Winters

The *Digital Technical Journal* is a refereed journal published quarterly by Digital Equipment Corporation, 146 Main Street MLO1-3/B68, Maynard, Massachusetts 01754-2571. Subscriptions to the *Journal* are \$40.00 for four issues and must be pre-paid in U.S. funds. University and college professors and Ph.D. students in the electrical engineering and computer science fields receive complimentary subscriptions upon request. Orders, inquiries, and address changes should be sent to the *Digital Technical Journal* at the published-by address. Inquiries can also be sent electronically to DTJ@CRL.DEC.COM. Single copies and back issues are available for \$16.00 each from Digital Press of Digital Equipment Corporation, 129 Parker Street, Maynard, MA 01754.

Digital employees may send subscription orders on the ENET to RDVAX::JOURNAL or by interoffice mail to mailstop MLO1-3/B68. Orders should include badge number, site location code, and address. All employees must advise of changes of address.

Comments on the content of any paper are welcomed and may be sent to the editor at the published-by or network address.

Copyright © 1993 Digital Equipment Corporation. Copying without fee is permitted provided that such copies are made for use in educational institutions by faculty members and are not distributed for commercial advantage. Abstracting with credit of Digital Equipment Corporation's authorship is permitted. All rights reserved.

The information in the *Journal* is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in the *Journal*.

ISSN 0898-901X

Documentation Number EY-M770E-DP

The following are trademarks of Digital Equipment Corporation: ADVANTAGE-NETWORKS, Alpha AXP, the Alpha AXP logo, AXP, Bookreader, DEC, DEC 3000 AXP, DEC FDDIcontroller, DEC OSF/1 AXP, DEC LANcontroller, DEC WANcontroller, DECbridge, DECchip 21064, DECconcentrator, DEChub, DECmcc, DECnet, DECnet/SNA, DECnet-VAX, DECnet/OSI for OpenVMS, DECnet/OSI for ULTRIX, DECNIS 500/600, DECstation, DECthreads, DECUS, Digital, the Digital logo, DNA, LANbridge, LAT, OpenVMS, OpenVMS on Alpha AXP, POLYCENTER, POLYCENTER Network Manager 200, POLYCENTER Network Manager 400, POLYCENTER SNA Manager, RS232, ThinWire, TURBOchannel, ULTRIX, VAX, VMS, and VMScluster.

Advanced System Management and SOLVE: Connect for EMA are trademarks of System Center, Inc.

AppleTalk is a registered trademark of Apple Computer, Inc.

BSD is a trademark of the University of California at Berkeley.

FastPacket, StrataCom, and IPX are registered trademarks of StrataCom, Inc.

IBM and NetView are registered trademarks of International Business Machines Corporation.

Motif, OSF, and OSF/1 are registered trademarks of Open Software Foundation, Inc.

NetWare and Novell are registered trademarks of Novell, Inc.

NFS is a registered trademark of Sun Microsystems, Inc.

Prestoserve is a trademark of Legato Systems, Inc.

System V is a trademark of American Telephone and Telegraph Company.

UNIX is a registered trademark of UNIX System Laboratories, Inc.

X/Open is a trademark of X/Open Company Limited.

Book production was done by Quantic Communications, Inc.

Cover Design

●Our cover illustrates an image of the simplicity of data sharing as experienced by system users interconnected through a global network; papers in this issue describe the depth and complexity of technologies and products that make the simplicity of data exchange possible.

The cover design is by Deb Anderson of Quantic Communications, Inc.

Contents

- 10 **Foreword**
Anthony G. Lauck

DECnet Open Networking

- 12 **Overview of Digital's Open Networking**
John Harper
- 21 **The DECnet/OSI for OpenVMS Version 5.5 Implementation**
Lawrence Yetto, Dorothy Noren Millbrandt, Yanick Pouffary,
Daniel J. Ryan, Jr., and David J. Sullivan
- 34 **The ULTRIX Implementation of DECnet/OSI**
Kim A. Buxton, Edward J. Ferris, and Andrew K. Nash
- 44 **High-performance TCP/IP and UDP/IP Networking
in DEC OSF/1 for Alpha AXP**
Chran-Ham Chang, Richard Flower, John Forecast, Heather Gray,
William R. Hawe, K. K. Ramakrishnan, Ashok P. Nadkar ni,
Uttam N. Shikarpur, and Kathleen M. Wilde
- 62 **Routing Architecture**
Radia J. Perlman, Ross W. Callon, and I. Michael C. Shand
- 70 **Digital's Multiprotocol Routing Software Design**
Graham R. Cobb and Elliot C. Gerberg
- 84 **The DECNIS 500/600 Multiprotocol Bridge/Router
and Gateway**
Stewart F. Bryant and David L.A. Brash
- 99 **Frame Relay Networks**
Robert J. Roden and Deborah Tayler
- 107 **An Implementation of the OSI Upper Layers
and Applications**
David C. Robinson, Lawrence N. Friedman,
and Scott A. Wattum
- 117 **Network Management**
Mark W. Saylor, Francis Dolan, and David G. Shurtleff
- 130 **Design of the DECmcc Management Director**
Colin Strutt and James A. Swist

Editor's Introduction



Jane C. Blake
Editor

Ten years ago, a network of 200 nodes was considered very large with uncertain manageability. Today, Digital's networks accommodate 100,000 nodes in open, distributed system environments and resolve the complexities of incompatibility among multivendor systems. Ten years from today, network systems comprising a million-plus nodes will be built based upon the Digital architectures and technologies described in this issue.

John Harper provides an informative overview of advances made with each phase of the Digital Network Architecture, now in Phase V. He describes the architectural layers and distinguishes Digital's approach to network services and management from that of others in the industry. His paper offers context for those that follow.

The Phase V architecture provides the migration to open systems from previous phases of DECnet. In implementing Phase V, designers of two DECnet products for the OpenVMS and ULTRIX operating systems shared several goals: extend network access in a multivendor environment, use standard protocols, and protect customers' software investments. Larry Yetto, Dotsie Millbrandt, Yanick Pouffary, Dan Ryan, and David Sullivan describe the DECnet/OSI for OpenVMS implementation and give details of the significantly different design of Phase V network management. In their paper on DECnet/OSI for ULTRIX, Kim Buxton, Ed Ferris, and Andrew Nash stress the importance of the protocol switch tables in a multiprotocol environment. DECnet/OSI for ULTRIX incorporates OSI, TCP/IP, and X.25.

In the broadly accepted TCP/IP protocol area, Digital has developed a high-performance TCP/IP implementation that takes advantage of the full FDDI bandwidth. K.K. Ramakrishnan and members of the development team review the characteristics of the Alpha AXP workstation, OSF/1 operating system, the

protocols, and the network interface. They then detail the optimizations made for high performance.

Routing data through networks with thousands of nodes is a very difficult task. Radia Perlman, Ross Callon, and Mike Shand describe how the Phase V routing architecture addresses routing complexity. Focusing on the IS-IS protocol, they pose problems a routing protocol could experience, present alternative solutions, and explain the IS-IS approach.

The challenges in developing multiprotocol routing software for internetworking across LANs, WANs, and dial-up networks are presented in the paper by Graham Cobb and Elliot Gerberg. They highlight the importance of the stability of the routing algorithms, using the DEC WANrouter and DECNIS products as a basis for discussing alternative designs. Stewart Bryant and David Brash then focus on details of the high-performance DECNIS 500/600 bridge/router and gateway. They discuss the architecture and the algorithm for distributed forwarding that increases scalable performance. Both the hardware and the software are described.

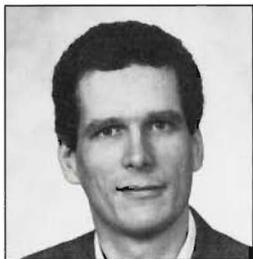
In addition to routing, the subject of data transfer of high-speed, bursty traffic using a simplified form of packet switching is described. Robert Roden and Deborah Taylor discuss frame relay networks, their unique characteristics, and the care needed in protocol selection and congestion handling.

The above discussions of data transfer and routing occur at the lower layers of the network architecture. Dave Robinson, Larry Friedman, and Scott Wattum present an overview of the upper layers and describe implementations that maximize throughput and minimize connection delays.

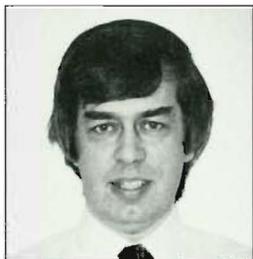
Network management is critical to the reliable function of the network. As Mark Saylor, Frank Dolan, and Dave Shurtleff tell us in their paper, Phase V management is based on a new architecture that encompasses management of the network *and* systems. They explain the decision to move management responsibility to the subsystem architecture, and also describe the entity model. The next paper elaborates on the director portion of the management architecture, called the DECmcc Management Director. Colin Strutt and Jim Swist review the design of this platform for developing management capabilities, the modularity of which allows future modules to be added dynamically.

The editors thank John Harper for his help in selecting the content of this issue.

Jane Blake



David L.A. Brash David Brash, a consultant engineer, joined Digital's Networks Engineering Group in 1985 to lead the hardware development of the MicroServer communications server (DEMSA). As the technical leader for the DECNIS 500/600 hardware platforms, David contributed to the architecture, backplane specification, module and ASIC designs and monitored correctness. He was an active member of the IEEE Futurebus+ working group. He is currently leading a group supporting Alpha design wins in Europe. David holds a B.Sc. in electrical and electronic engineering from the University of Strathclyde.



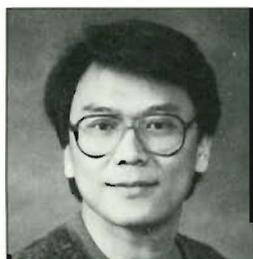
Stewart F. Bryant A consulting engineer with Networks and Communications in Reading, England, Stewart Bryant worked on the advanced development program that developed the DECNIS 600 architecture. During the last six months of the program, he was its technical leader, focusing on implementation issues. Prior to this work, Stewart was the hardware and firmware architect for the MicroServer hardware platform. He earned a Ph.D. in physics from Imperial College in 1978. He is a member of the Institute of Electrical Engineers and has been a Chartered Engineer since 1985.



Kim A. Buxton Kim Buxton is a principal software engineer in the Networks and Communications Group. During the past seven years, Kim has been working on DECnet and OSI for UNIX operating systems. She is currently the project leader of the DECnet/OSI for DEC OSF/1 AXP release. Prior to assuming the role of project leader, Kim worked on network management, session control, and transport protocols for DECnet-ULTRIX products. She has worked in the area of networks and communications since joining Digital in 1980. She earned her B.S. degree in mathematics and secondary education from the University of Lowell.



Ross W. Callon As a member of Digital's Network Architecture Group from 1988 to 1993, Ross Callon worked on routing algorithm and addressing issues. He was a primary author of the Integrated IS-IS protocol and of the guidelines for using NSAP addresses in the Internet. Previously, he was employed by Bolt Beranek and Newman as a senior scientist and helped develop the ISO CLNP protocol. Ross received a B.Sc. (1969) in mathematics from MIT and an M.Sc. (1977) in operations research from Stanford University. He is currently employed as a consulting engineer at Wellfleet Communications.



Chran-Ham Chang Chran-Ham Chang is a principal software engineer in the UNIX System Engineering Group and a member of the FAST TCP/IP project team. Since joining Digital in 1987, Chran has contributed to the development of various Ethernet and FDDI device drivers on both the ULTRIX and DEC OSF/1 AXP systems. He was also involved in the ULTRIX network performance analysis and tools design. Prior to this, Chran worked as a software specialist in Taiwan for a distributor of Digital's products. He received an M.S. in computer science from the New Jersey Institute of Technology.



Graham R. Cobb Graham Cobb is a consulting engineer in the Internet Products Engineering Group and was software project leader for the DECNIS 500/600 router development. Graham holds an M.A. in mathematics from the University of Cambridge and joined Digital as a communications software engineer in 1982. He has worked on many Digital communications products, including X.25 products and routers, and was a major contributor to the DEC WANrouter 100/500 software immediately prior to leading the DECNIS development. Most recently, Graham has been working on new-generation routing software.



Francis Dolan Frank Dolan is a consultant engineer with Digital's Telecommunication Business Group Engineering in Valbonne, France. He is currently the project manager and technical leader of the GDMO translator, a tool being developed to support the DECmcc/TeMIP OSI access module and OSI agent presentation module. Prior to this work, Frank was the architect of several Phase V DNA specifications, including DDCMP network management, OSI transport, and network routing accounting. He was also an active member of OSI management standards committees. Frank has filed one European patent application.



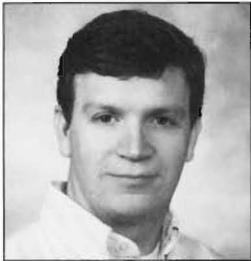
Edward J. Ferris Ed Ferris is a principal engineer in the Networks and Communications Group. During the past seven years, Ed has been working on DECnet-ULTRIX. He is currently one of the technical leaders of the DECnet/OSI for DEC OSF/1 AXP release. Ed has primarily worked at the data link and network protocol layers. He has worked on networks and communication products since joining Digital in 1982. Ed earned a B.A. in English from the University of Massachusetts and a B.S. in computer engineering from Boston University.



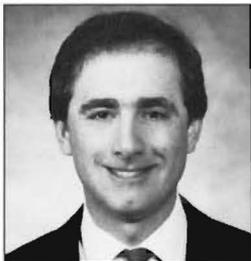
Richard Flower Richard Flower works on system performance issues in multiprocessors, networking, distributed systems, workstations, and memory hierarchies. The need for accurate time-stamping events across multiple systems led him to develop the QUIPU performance monitor. The use of this monitor led to performance improvements in networking, drivers, and RPC. Richard earned a B.S.E.E. from Stanford University (with great distinction) and a Ph.D. in computer science from MIT. Prior to joining Digital, he was a professor at the University of Illinois. Richard is a member of Phi Beta Kappa and Tau Beta Pi.



John Forecast A software consultant engineer with the Networks Engineering Advanced Development Group, John Forecast addresses network performance issues associated with the transmission of audio and video data through existing networks. John joined Digital in the United Kingdom in 1974 and moved to the United States to help design DECnet-RSX Phase 2 products, DECnet Phase IV, and DECnet implementations on ULTRIX and System V UNIX. John also worked on file servers for VMS and a prototype public key authentication system. He holds a Ph.D. from the University of Essex.



Lawrence N. Friedman Principal engineer Lawrence Friedman is a technical leader in the OSI Applications Group. He joined Digital in 1989 and is the project leader for ULTRIX FTAM V1.0 and V1.1. In addition to his project responsibilities, Larry is Digital's representative to the National Institute of Standards and Technologies (NIST) FTAM SIG and was the editor of the NIST FTAM SIG Phase 2 and Phase 3 documents from 1990 to 1992. He is currently the editor for the FTAM File Store Management International Standard Profile. Larry holds a B.A. (1978) in music from Boston University.



Elliot C. Gerberg Elliot Gerberg is a senior engineering manager in Digital's Networks Engineering Division, managing the Routing Engineering Group (USA). Since joining Digital in 1977, he has worked on numerous projects including the DEUNA, Digital's first LAN adapter; the DECserver 100, Digital's first low-cost terminal server; the SGEC, a high-performance Ethernet semiconductor interface; and various multiprotocol routers. Elliot has a B.S. in physics from SUNY and an M.S. in computer science from Boston University. He holds professional memberships with the IEEE, the ACM, and the Internet Society.



Heather Gray A principal engineer in the UNIX Software Group (USG), Heather Gray is the technical leader for networking performance on the DEC OSF/1 AXP product family. Heather's current focus is the development of IP multicast on DEC OSF/1 AXP. She has been involved with the development of Digital networking software (TCP/IP, DECnet, and OSI) since 1986. Prior to joining USG, Heather was project leader for the Internet Portal V1.2 product. She came to Digital in 1984, after working on communication and process control systems at Broken Hill Proprietary Co., Ltd. (BHP) in Australia.



John Harper As technical director of the Corporate Backbone Networks Group in NAC, John Harper directed the development of the DECnet Phase V architecture. Until last year John also chaired the ISO Committee JTCl/SC6/WG2, which deals with standards for the OSI network layer. He joined Digital in 1974 after receiving a degree in computer studies (1st class honors) from the University of Lancaster. John has ten patents (filed or issued) on computer networks and has published several conference papers on that subject. He has made numerous contributions to standards for computer networks.



William R. Hawe A senior consulting engineer, Bill Hawe manages the LAN Architecture Group. He is involved in designing architectures for new networking technologies. Bill helped design the FDDI and extended LAN architectures. While in the Corporate Research Group, he worked on the Ethernet design with Xerox and Intel and analyzed the performance of new communications technologies. Before joining Digital in 1980, Bill taught electrical engineering and networking at the University of Massachusetts, where he earned a B.S.E.E. and an M.S.E.E. He has published numerous papers and holds several patents.

Biographies



Dorothy Noren Millbrandt Dotsie Millbrandt is a principal software engineer and a co-project leader for Common Network Management. Currently she is developing management components that will work across all the DECnet/OSI platforms: OpenVMS, OSF/1, and ULTRIX. Dotsie was the project leader for the MOP component and the trace facility and has worked on OSI transport and configuration software. Prior to this work, she was a project leader and microcode developer for DSB32 and KMV11 synchronous communications controllers in the CSS Network Systems Group.



Ashok P. Nadkarni A principal software engineer in the Windows NT Systems Group, Ashok Nadkarni is working on a port of native Novell NetWare to Alpha AXP systems. Prior to this, he was a member of the NaC Advanced Development Group. He has contributed to projects dealing with IP and OSI protocol implementations, network performance improvement, a prototype of the Digital distributed time service, and mobile networking. He holds a B. Tech. in computer engineering from the Indian Institute of Technology, Bombay, and an M.S. from Rensselaer Polytechnic Institute. Ashok joined Digital in 1985.



Andrew K. Nash Andrew Nash is a principal software engineer with NaC Australia and was the project leader for the ULTRIX Phase V X.25 products. He is currently technical leader for NaC Australia and has been with the group since 1988. Since joining Digital in 1980, he has worked for Educational Services and the Customer Support Centre and has been a consultant for Software Services. Andrew received a B.Sc. (M.Sc.) from the University of Adelaide and a graduate diploma in software engineering from the University of Technology, Sydney.



Radia J. Perlman As a member of the Network Architecture Group, Radia Perlman has been designing protocols for bridges and routers since joining Digital 13 years ago. She designed the spanning tree algorithm used by all standardized forms of bridges, as well as many of the protocols in IS-IS. Radia authored the book *Interconnections: Bridges and Routers* and has more than 20 patents filed or pending in the areas of bridging, routing, and network security. She holds S.B. and S.M. degrees in mathematics and a Ph.D. in computer science, all from the Massachusetts Institute of Technology.



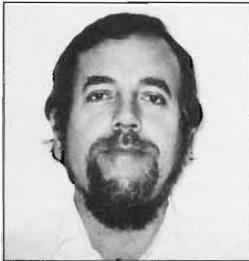
Yanick Pouffary A principal software engineer, Yanick Pouffary is currently the transport technical leader in the DECnet/OSI for OpenVMS Group. She was the principal designer and developer of OSI transport and NSP transport protocol engines. Prior to this work, she developed the presentation layer for the VTX20, a videotext terminal. Before joining Digital in 1985, Yanick worked for the CODEX Corporation on a statistical multiplexer. Yanick earned a B.S. in computer science from the University of Nice, France, and an M.S. in computer science from the State University of New York at Stony Brook.



K. K. Ramakrishnan A consulting engineer in the Distributed Systems Architecture and Performance Group, K. K. Ramakrishnan joined Digital in 1983 after completing his Ph.D. in computer science from the University of Maryland. K. K.'s research interests include performance analysis and design of algorithms for computer networks and distributed systems using queuing network models. He has published more than 30 papers on load balancing, congestion control and avoidance, algorithms for FDDI, distributed systems performance, and issues relating to network I/O. K. K. is a member of the IEEE and the ACM.



David C. Robinson David Robinson is a principal software engineer in Network Engineering Europe. He was the architect for the OSI upper layers and designed and prototyped Digital's improved upper layer implementation. He came to Digital in 1988 from the General Electric Co. (GEC) in Chelmsford, Essex, U.K., where he developed a remote procedure call and a distributed computing environment. Dave holds a B.Sc. (Eng) in computing science (1982) and a Ph.D. in management of very large distributed computing systems (1988), both from the Imperial College in London.



Robert J. Roden Robert Roden is a consulting engineer in Networks Engineering. Recently, he has been working on new transmission technologies such as frame relay and switched multimegabit data service. He has also worked on computer integrated telephony and chaired a group developing related standards. Robert joined Digital in 1986 from Racal Milgo, where he was responsible for local area networks and network management platforms. He received a B.Sc. (1971) in physics and a Ph.D. (1974) in materials science from the Imperial College in London.



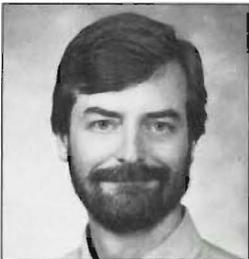
Daniel J. Ryan, Jr. A principal software engineer in the DECnet/OSI for OpenVMS Group, Dan Ryan was responsible for the configuration and installation portion of the DECnet/OSI for OpenVMS product. Recently he was the team leader for the transport development effort. Currently he is investigating DECnet/OSI and TCP/IP integration as well as DECnet/OSI critical problems. Dan has 14 years of experience in data communications and has been with Digital since 1983. He was previously employed as a systems programmer and was a free-lance consultant on computer communication solutions.



I. Michael C. Shand Consulting engineer Michael Shand of Networks Engineering is responsible for the DNA Phase V network routing layer architecture. Prior to this, he worked on the Phase V X.25 access and HDLC architectures. He represents Digital on the ISO network layer committee and was a major contributor to the standardization of the IS-IS routing protocol (ISO/IEC 10589). Mike came to Digital in 1985 from Kingston Polytechnic (U.K.). He has an M.A. (1971) in natural sciences from the University of Cambridge and a Ph.D. (1975) in surface chemistry from Kingston Polytechnic.



Uttam N. Shikarpur Uttam Shikarpur joined Digital in 1988 after receiving an M.S. in computer and systems engineering from Rensselaer Polytechnic Institute. Uttam is a senior engineer and a member of the UNIX Systems Group working on network drivers and data link issues. His current project involves writing a token ring driver for the DEC OSF/1 AXP operating system. Prior to this work, he contributed to the common agent project.



David G. Shurtleff A member of Digital's Corporate Systems Engineering Group, David Shurtleff consults in support of major systems integration projects and participates in CSE initiatives to improve engineering processes. Previously, he was a member of the EMA Architecture Group, where he worked on the specification of EMA director architectures and the development of systems management standards. David has also worked in the DECmcc strategic vendor program as a senior technical resource. Before joining Digital in 1988, David was on the packet switch development staff at BBN Communications Corporation.



Colin Strutt Colin Strutt is the DECmcc technical director in Enterprise Management Frameworks, part of the NAS Systems Management. Prior to that position, Colin was the project leader for the terminal server manager, various terminal server products, Ethernet communications server, and DECnet-1AS. He joined Digital in 1980 and is now a consulting engineer. Colin received a B.A. (honors) and a Ph.D. both in computer science from the University of Essex, U.K. He is a member of BCS and ACM. Colin has several patents pending on DECmcc technology and has published papers on integrated network management.



David J. Sullivan David Sullivan is a senior software engineer and was the technical leader of the node agent and event dispatcher components for the DECnet/OSI for OpenVMS product. David also worked as an individual contributor on the design and implementation of the session control layer. He is currently working on a development effort to allow the DECnet/OSI product to run on Digital's AXP platforms. After joining Digital in 1987, he worked in the VAX/RPC Group where he was responsible for writing tests for the pidgin compiler. David holds a B.S.C.S. (1988) from Merrimack College.



James A. Swist Jim Swist joined Digital in 1975. He is a consulting software engineer and the technical leader for open systems in the Enterprise Management Frameworks Group. Prior to this position, he was a system management architect for VMS development, technical leader and development manager for TDMS/ACMS/CDD database systems, and a consultant in software services for several large commercial TP projects. Jim earned a B.S. in electrical engineering from the Massachusetts Institute of Technology in 1970. He has one patent pending on MCC distributed dispatch.



Mark W. Saylor Mark Saylor is the manager of Digital's Enterprise Management Architecture Group. He is the author of the *EMA Entity Model* and the *Phase V DECnet Network Management Specification*. Mark was a member of the ISO and ANSI committees working on OSI system management and was the ANSI T5.4 ad hoc group leader on the structure of management information. Prior to this work, Mark was the principal designer and development supervisor for the NMCC/DECnet monitor. Mark joined Digital in 1979. He holds an M.S. in mathematics from the University of Notre Dame.



Deborah Tayler Deborah Tayler, a principal software engineer in Networks Engineering Europe, is currently responsible for the design and implementation of frame relay and point-to-point protocol functionality on multiprotocol routers. She joined Digital in 1982 and has worked on DECtalk, ALL-IN-1, and computer integrated telephony projects. Deborah received a B.Sc. (1981) in economics from University College in London and an M.Sc. (1982) in the theory and applications of computation from Loughborough University of Technology in Loughborough, Leicestershire.



Scott A. Wattum Senior software engineer Scott Wattum is a member of the OSI Applications Engineering Group. He is responsible for the design and development of OpenVMS Virtual Terminal V1.0 and is involved in the ULTRIX and OSF/1 porting efforts. Previously, Scott worked at the Colorado Springs Customer Support Center and provided network support, specializing in OSI protocols and applications. Prior to joining Digital in 1987, he was employed by the University of Alaska Computer Network in various software positions. He received a B.A. (1985) in theatre from the University of Alaska, Fairbanks.



Kathleen M. Wilde As a member of the Networks Engineering Architecture Group, Kathleen Wilde focuses on integration of new TCP/IP networking technologies into Digital's products. For the past two years, she has been prototyping high-performance network features on the OSF/1 operating system and coordinating the standards strategy for Digital's IETF participation. Previously, she was the development manager of the ULTRIX Network Group. Her responsibilities included product development of TCP/IP enhancements, FDDI, and SNMP. She has a B.S. in computer science and mathematics from Union College.



Lawrence Yetto Larry Yetto is currently a project and technical leader for the DECnet/OSI for OpenVMS Group. He joined Digital in 1981 and has held various positions in software engineering on development projects for VMS journaling, VMS utilities, and DECnet-VAX Phase IV. He also worked in the Project Services Center, Munich, and was the project leader for the OpenVMS version 5.0 field test. Prior to joining Digital, Larry worked as a systems programmer at Burroughs Corporation. He earned a B.A. in both math and computer science from the State University of New York at Potsdam.

Foreword



Anthony G. Lauck
*Corporate Consultant
Engineer and
Technical Director,
Networks Engineering*

Digital's fifth generation of computer networking products enters the market as computer networking technology enters its third decade as a practical technology. Digital's first four generations of DECnet products entered a marketplace that was oriented toward proprietary computer solutions and where networking grew slowly from a departmental function to include a functional unit of an enterprise and, eventually, an entire enterprise. With networks confined to a department or function, there was little need for heterogeneity. Engineering departments used Digital's mini-computers linked by DECnet, while corporate business applications ran on IBM mainframes accessed by SNA networks. Eventually these heterogeneous networks were linked by gateways which provided the necessary protocol conversions; but integration was never transparent—especially to the system and network managers. The number of computers in a network was limited by the scope of the department, function, or organization and by the cost of individual computers. Timesharing remained the dominant mode of computer use in these networks; there were significantly fewer computers in a network than users of the network.

When Digital began its initial architectural work on DECnet Phase V, we realized that technological and economic limitations on network size were going away. Microprocessors were making it possible for each person to have a computer. Local area networks were making it possible for each computer to be conveniently and inexpensively connected. Early experience with embedded

computers in manufacturing applications at Digital and with some of our customers convinced us that the number of computers in a network could easily exceed the number of people using the network. A few communities, such as the worldwide high-energy physics community, had built networks that extended beyond the bounds of a single enterprise. We saw that networks would need to have great scope and would need to support a great diversity of management. An architecture such as our DECnet Phase IV, which limited a single network to tens of thousands of nodes, would become too confining.

Early computer networks were homogeneous in architecture and implementation, reflecting the proprietary nature of the computer industry at the time and also the difficulty of getting heterogeneous networks to work. Digital learned the difficulties of heterogeneous networking back in the 1970s when it developed DECnet Phase II and made a network work across a range of computer systems from a single vendor. By the early 1980s there were already multiple competing network architectures, some proprietary to organizations, some viewed as proprietary to a single nation. Different enterprises and different departments of a given enterprise had chosen different computer vendors, operating systems, and network architectures. Linking these together by gateways would be too cumbersome. These factors prompted for us the vision of a common network architecture, standardized on an international scope and appropriate to Digital's role as an international corporation. Many of the papers in this issue describe our realization of this vision.

Our vision of a common networking architecture gave us the basic requirements for DECnet Phase V—a scalable network architecture that is open and standardized internationally. Like earlier generations of DECnet, this architecture would be backward compatible with its predecessor, preserving our customers' investments in applications and network infrastructure. Implementing this vision of a homogeneous network architecture based on internationally standardized protocols and backward compatibility with DECnet Phase IV proved to be a daunting task. It involved developing new networking technology, in particular new routing and addressing technology, standardizing this technology in the international community, and implementing it across a full range of products.

While Digital continued to work on its vision, networking expanded vigorously across the entire computer industry. Protocols appeared in niches: vendor based, operating system based, industry

based. Users needed connectivity between these niches, providing market pull for expansion from initial niches. The result is today's world of multiprotocol computer networks. Digital's next generation of networking products also reflects this multiprotocol reality. Host networking products support several protocol families and are constructed to isolate many of the differences between network protocols from users. Network infrastructure products such as routers and network management software support this diversity more fully, reflecting the need for the infrastructure to support all the types of network traffic. Many papers in this issue relate to our participation in this complex reality.

Computer networks have become an essential part of many organizations. These networks must be dependable and must not be bottlenecks. In its fifth generation of networking products, Digital has stressed robustness and performance. In designing Digital's router products, we placed great emphasis on robustness and network stability, particularly under conditions of traffic overload. These are not qualities that our customers will necessarily appreciate unless they have experienced their absence in an overloaded network. New applications and larger data storage mandate higher host networking

throughput. High-speed local area networks, such as FDDI, together with high-speed RISC processors, such as Alpha AXP, create the expectation of high-performance host networking. Achieving this level of performance takes more than fast hardware, however. It requires careful attention to details of protocol implementation and interaction with network interface hardware, the processor and memory system, and the operating system. Several papers in this issue describe how Digital has achieved leadership in network robustness and performance.

Networking depends on a variety of underlying communications technologies and services. This issue of the *Digital Technical Journal* concentrates on how these underlying technologies can be used to build large-scale computer networks; earlier issues described such underlying communications technologies as Ethernet and FDDI. This issue does, however, include one paper on a new wide area technology and service, Frame Relay, and how it can be used by computer networks. Many other new communications technologies and common carrier services are in the process of being integrated into Digital's family of networking products. These will be described in future issues of the *Journal*.

Overview of Digital's Open Networking

The principal element of Digital's open networking family of products is the DECnet computer network. In its latest form, DECnet supports very large networks of more than 100,000 nodes and incorporates industry standards such as OSI and TCP/IP. To meet the design goals of the Digital Network Architecture, the structure of DECnet is divided into layers with defined relationships between layers. Since its introduction in 1974, DECnet has evolved in parallel with the standards for open networking. Digital has contributed to the formation of networking standards, and the standards have, in turn, influenced the design of DECnet.

In 1974, Digital shipped the industry's first general-purpose networking product for distributed computing. The DECnet computer network was the embodiment of the vision that small systems working together could become an alternative to mainframe computing. Prior to that time, networking products had been aimed at solving some specific problem and had often been closely integrated with a particular application. In contrast, DECnet allowed any application to share data with all others. Whereas previous networking products in the industry had concentrated on connecting terminals to hosts, DECnet provided peer-to-peer networking for the first time. By doing this, it anticipated the client-server computing style that is now commonplace and established client-server computing as a viable approach.

DECnet built on work that had been done in the research community. The internet protocol, funded by the Advanced Research Projects Agency (ARPA), was of particular relevance.¹ This too was aimed at providing general-purpose distributed computing and later evolved into the well-known TCP/IP (transmission control protocol/internet protocol) protocol suite. In 1974, however, it was still a research topic.

In the same year, International Business Machines Corporation announced its Systems Network Architecture (SNA).² The comparison between SNA and DECnet is interesting because SNA was designed, not surprisingly, to support mainframe computing. It focused principally on connecting many relatively unintelligent devices, such as terminals and

remote job entry stations, into a single computer. Only after several years did SNA allow more than one mainframe to exist in the same network. Its original goal was to address the proliferation of application-specific protocols that allowed a terminal connected to the network to use one application only.

This paper presents a short history of the DECnet networking product, defining each phase of its evolution in terms of its contribution to distributed computing. It explores the development of DECnet Phase V, the current implementation, and discusses the principles of Digital's layered architecture. The paper then describes the layers of DECnet, the importance of naming services, and the role of network management.

A Short History of DECnet

The development of DECnet has proceeded by phases. Each phase has represented a major step in the evolution of the product family. The initial products, later referred to as Phase I, revealed some unexpected problems in building a range of products across different systems that would all work together. One of the consequences was the creation of a distinct Network Architecture Group. Their job was to produce detailed specifications of the protocols and interfaces to be used without constraining the implementers to build products in some particular way. At that time, software portability was practically unheard of, and each different hardware or software environment had its own completely separate implementation. Phase II of DECnet, introduced

in 1978, provided full interoperability between the different implementations, thanks to adherence to a rigorously specified architecture.

At this stage, systems still had to be directly connected to each other if they were to communicate. Phase III, which appeared in 1981, introduced the ability to route messages through any number of links and intermediate systems to reach a destination. DECnet again used a technique from the research networks, a dynamic adaptive routing algorithm, which computed the best route to a destination automatically as the physical connectivity of the network changed. Competing products at the time (such as SNA) required routes to be computed and entered manually, including backup routes for use in the event of failure of a link or a system in the network.

Phase III also included full remote management and reflected the gradual emergence of standards for computer networking by supporting X.25 packet switching networks as one means for connecting systems.³ A Phase III network could contain up to 255 nodes.

The invention of local area networks (LANs), and in particular the Ethernet, was to have a huge impact on the use of networking.⁴ For the first time it was cheap and simple to connect a system to the network. Prior to LANs, only wide area network technology was used, even when the systems were physically next to each other. DECnet Phase IV, which appeared in 1984, added support for the Ethernet and allowed networks to contain up to 64,000 nodes.

The Evolution of Open Networking

When DECnet appeared in 1974, all its networking protocols were "proprietary," that is, they had been developed by Digital and remained under Digital's control. At that time there were no standards or publicly defined network protocols. Work on standards for this purpose began during the 1970s, and in 1978 the Comité Consultatif Internationale de Télégraphique et Téléphonique (CCITT) published its Recommendation X.25.⁵ This document defined a standard way of connecting a computer to a network that would permit free communication between all attached computers. X.25 networks were typically expected to be provided by a public carrier such as a telephone company.

The appearance of this standard prompted the question, "Now that our computers can talk to each other, what are they going to say?" Simply permit-

ting them to send data to each other was of no use unless they could also understand it and make some use of it. DECnet, for example, included protocols for transferring files and for remote terminal access as well as the base protocols for transferring data.

Thus the idea of open systems interconnection (OSI) was born. OSI was the most ambitious effort in the history of standards. Its goal was to develop a complete set of standard protocols that would allow computers not only to exchange data but also to make meaningful use of it in their applications. The work was undertaken by the International Organization for Standardization (ISO). This organization has representatives from all major countries and is thus able to draw upon their extensive experience in research and commercial networking.

By 1984, when DECnet Phase IV became available, the work on OSI had made substantial progress. The architectural model had been published as an international standard, and standardization of many of the protocols was at an advanced stage.⁵ It was also becoming clear that the future of computer networks depended on the ability to communicate without regard to who was the supplier of a system. Ad hoc solutions, such as the DECnet/SNA gateway, existed for communication between different network architectures.⁶ OSI, however, held the promise of being a general solution. It was feared that the alternative to OSI would be the adoption of a vendor-specific architecture as a de facto solution, and that that architecture would inevitably be SNA. The internet family of protocols, colloquially known as TCP/IP, had not yet become the force it is today.⁷

Detailed examination of the OSI protocols showed that they formed a suitable basis for the evolution of DECnet. This was not surprising, since the ISO had incorporated Digital's basic concepts into OSI, rather than the different ideas put forth by the public network operators. A number of deficiencies were identified, but these could be remedied by contributing more of Digital's technology to the standards process. For example, all the network-layer routing protocols used in OSI were contributed by Digital. Thus the decision was made that the next phase of DECnet, Phase V, would use the OSI standards as much as possible. The existing proprietary protocols would be retained only for the purpose of backward compatibility.

During the development of the architecture and products for Phase V, another event of great significance took place. During the 1980s, TCP/IP emerged as an alternative solution for open networking. This

development was prompted by the explosion in the use of workstations based on the UNIX system style of computing. The architectural model of Phase V allowed a relatively straightforward integration of these protocols into the products, although a great deal of necessary software was written. Since OSI and TCP/IP were never designed to work together, allowing them to coexist in the same network demanded considerable creativity.⁸

Goals of DECnet Phase V

The design of DECnet Phase V had three principal goals:

- To allow networks to grow to be very large, with one million systems as a practical target
- To use standard protocols to the maximum extent possible
- To support a distributed-system mode of operation in which the systems cooperate more closely than in traditional networking

The 64,000-node size limit of Phase IV was far from posing a practical problem in 1984, but it was then foreseen that computer networks in large enterprises would approach this limit by the end of the decade. Indeed, this happened with Digital's internal network, which grew to over 100,000 nodes on Phase IV with the use of innovative management techniques. The node size limitation was imposed primarily by the size of the addresses used, which was 16 bits. Addresses in OSI networks can be as long as 20 bytes, which removes the immediate limitation. Very large networks, however, need more than large addresses to support 100,000 nodes or more. For example, the Phase IV routing algorithm has certain inherent weaknesses that start to appear for networks at the Phase IV size limit. For this reason, Phase V employs a different routing algorithm, which readily supports networks of millions of nodes.⁹ This algorithm has subsequently been adopted as the international standard for routing in OSI networks and, with modifications, for TCP/IP networks.^{10,11}

Management of very large networks also requires special attention. DECnet has always provided a high degree of automated management compared to other network architectures, but as a network increases in size, the burden of tracking the configuration increases disproportionately. Assigning addresses to nodes was a manual procedure in Phase IV, and maintaining the correspondence between node names and their addresses was

performed separately in each system. A goal for Phase V was to provide a robust, distributed naming service throughout the network. Furthermore, nodes would be allowed to generate their own addresses in a reliable and unambiguous way and to register themselves in this naming service. Thus a new system can be connected to the network without any administrative procedure, if network security policies permit.

At a more detailed level, the architecture has a set of goals that have evolved over time to include the following.

- Conceal network operation from the user. The internal operation of a large network is inevitably complex, but to the user it should appear simple.
- Support a wide range of applications.
- Support a wide range of communications facilities: LANs, wide area leased lines, X.25 networks, etc.
- Support a wide range of network topologies.
- Use standards wherever feasible rather than proprietary protocols. For cases in which standards are evolving but are not yet finished, ensure that future migration is as smooth as possible.
- Require minimum management intervention.
- Be manageable. Not all functions can be automated; for example, some depend on the organizational policy of the user. In such cases management should be as simple as possible and should not impose any particular style of management.
- Permit growth without disruption.
- Permit migration between versions. Each phase of DECnet is guaranteed to work with the next and previous phases, so that the systems in the network can be upgraded over a long period. It would be inconceivable to upgrade thousands of systems overnight.
- Be extensible to new developments in technology.
- Be highly available in the face of line or system failure or even, to the extent possible, operator error.
- Be highly distributed. The major functions of the Digital Network Architecture (DNA), such as routing and network management, are not centralized in a single system in the network. This in turn increases availability.

- Allow for security functions, such as authentication of remote users and access control.

Architectural Principles

DNA is a layered architecture. The necessary functions are divided into related and logically coherent groups called layers. The layers are built on top of one another, so that each layer makes use of services provided by the one below it. To meet the goals of DNA, particularly those relating to flexibility, the structure of a layered architecture is essential.

Figure 1 illustrates the principles of a layer in the terminology of the OSI reference model.⁵ These principles apply to any layer; in Figure 1 they are shown applied to the transport layer. Each communicating system contains its own element of the layer, called the transport entity. These entities communicate with each other through the transport protocol. This protocol is conveyed using the services of the next lower layer, in this case, the network layer. For this purpose the most important service is the one that conveys data without regard to its contents. Other services are also provided, for example, connection management services. The transport layer also provides a well-defined transport service to its user, in this case, the session layer. The detailed mechanisms and protocols of the layer are hidden from the layers above and below, so that the layer above sees only a well-defined service.

This independence of the mechanisms used permits substantial changes to be made to the mechanisms and protocols of a layer without affecting the adjacent layers. This very important property is called layer independence. It has been extensively exploited in the development of DECnet to allow

protocols to be enhanced or even completely replaced.

The principles of layered architecture were defined in a rigorous way by the OSI reference model, building on previous work such as DECnet and the TCP/IP protocol family. The original layer structure of DNA was defined in Phase I and has changed only a little since then. It corresponds to the lower layers of OSI as well as the layers of TCP/IP.

The Layers of DECnet

Figure 2 shows the layers of DECnet Phase V. The lower layers are the physical, data link, network, and transport layers. They provide a universal, reliable service for moving data from one system to another. Many different underlying means of physical communication can be used, with their associated protocols, including:

- Ethernet LANs and the equivalent standard (IEEE 802.3, ISO 8802-3)
- Token ring LANs (IEEE 802.5)
- Wide area links running over leased links at any appropriate speed
- X.25 wide area networks

The network and transport layers unify the service provided by these disparate physical networks and allow communication across any mixture of different facilities.

Protocols from different protocol suites may be used, including OSI, TCP/IP, and DECnet Phase IV, but the structure of the layers is the same in each case. This facilitates interworking in mixed-protocol networks.

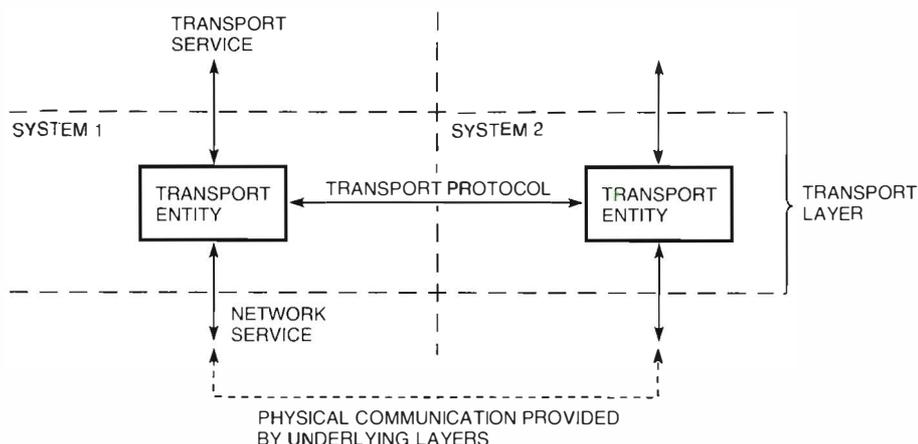


Figure 1 Elements of a Layer of DECnet Architecture

The upper layers of DECnet, the session, presentation, and application layers, make use of the reliable transport service to provide application-oriented functions, such as file transfer or electronic mail. Again, different protocol suites are supported, although in this case there are historical reasons for the different layer structures that exist.

The Physical Layer

The physical layer is concerned essentially with the electrical or other physical aspects of communication. It converts electrical or other signaling into binary data (i.e., bits) and vice versa.

In DECnet, this layer has always been viewed as the province of standards for devices such as modems and LANs. These standards may have an extremely complicated internal structure, as is the case for some of the emerging high-speed, wide area network standards, but this complexity is not visible to the layers above.

The Data Link Layer

The data link layer provides a reliable communication path between directly connected systems in the network. Its protocols can detect errors introduced by the physical layer (for example, from electrical disturbance). For media known to exhibit a high error rate, such as analog links, the data link layer also provides error-correcting mechanisms.

DECnet supports a variety of protocols in the data link layer, depending on the nature of the physical link and the need to accommodate existing technologies.

The Network Layer

The network layer provides the means to move data from one system to another, without regard to the nature of the connections between them. It finds a route through multiple systems and physical paths

as necessary for any particular pair of communicating systems. In DECnet, systems that move data through the network without being involved in the details of the communication are called routers.

A key element in this layer is the network address. Every system in the network has a unique address. Every system can communicate with every other system in the network, whether it is adjacent or located on the other side of the world. OSI provides an addressing scheme that allows every system in the world to have a unique address.¹² It may also give some hints to find a route to the system. Previous versions of DECnet (Phase IV and before) used a different addressing scheme. Phase V includes a way to map these addresses into the OSI scheme.

In addition to protocols for carrying user data between communicating systems, the network layer also contains protocols for finding routes between systems. The routing protocols used in DECnet Phase V are international standards, but the technology was developed by Digital and subsequently submitted to the relevant standards organizations.^{10,11,13}

The network layer has a complex internal structure that allows one network to use the connections provided by another. For example, some of the links in a DECnet network may be provided by a public X.25 network, which is also providing links in other private networks.

The Transport Layer

The transport layer provides a reliable end-to-end service between two communicating systems, concealing from its users the detailed way in which this is achieved. Unlike the layers below it, the transport layer is present only in the end systems communicating with each other. Thus it allows the end systems to take full responsibility for the quality of the communications. The functions of the transport layer include

- Recovery from data loss, for example, when the network layer fails to deliver a packet due to congestion
- Flow control, so that the transmitter does not send data into the network faster than the receiver can accept it
- Segmentation and reassembly of user messages, so that the necessary division of data into distinct messages sent through the network does not limit the size of messages as seen by the user

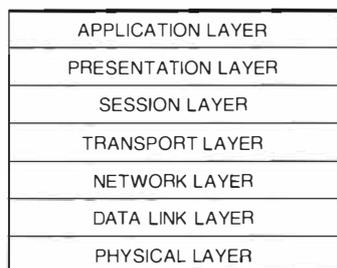


Figure 2 The Layers of DECnet

- Congestion avoidance, so that data transmitters can adjust their rate of transmission into the network in reaction to congestion indications from the network layer

DECnet supports three protocols in the transport layer: the network services protocol (NSP), defined for previous phases of DECnet; the OSI transport protocol; and TCP from the internet protocol suite.^{7,14}

Upper Layer Protocols

The OSI model defines three distinct layers above the transport layer: the session, presentation, and application layers.

- The session layer organizes the structure of message exchanges. For example, it provides half-duplex semantics and allows checkpoints to be established for recovery from system failure.
- The presentation layer deals with the existence of different data representations in different systems. It allows a mutually acceptable transfer syntax to be established which each communicating system will be able to convert to and from its internal representation.
- The application layer contains protocol elements specific to a particular application, such as file transfer. It also provides a structure that allows applications to be built that use multiple protocols in a coordinated fashion.

The DECnet Phase IV and TCP/IP protocol stacks, which are also supported by DECnet Phase V, do not have this structure. Rather, the functions of the session and presentation layers are built into the application protocols as needed.

All three protocol suites support a wide variety of applications, in addition to allowing a user the flexibility to develop custom applications. Typical applications include

- File transfer and access
- Virtual terminal
- Electronic mail
- Remote procedure calls

Naming Services

The protocols in the lower layers operate in terms of addresses which are, for practical purposes, simply bit strings. Their format is heavily constrained by the protocols, and their value is constrained by

the network topology or hardware. These addresses are not at all user friendly, nor are they intended to be. The human users of a network need access in terms of something which they can remember and which makes sense to them, which is to say a name. Computers in the network therefore need to be able to take a name and change it to an address, and vice versa for incoming traffic.

DECnet Phase IV had a very simple approach to this problem. Since it was aimed at small- to medium-sized networks, it was practical for each system to store the complete set of names and addresses. Administrative procedures, such as regular file transfers, could be used to ensure that all systems were kept up-to-date.

DECnet Phase V was designed to allow much larger networks to be built, while both OSI and TCP/IP are designed to support networks on a global scale. The administrative problems and storage requirements of the Phase IV approach make it unusable for very large networks. A further complication arises as networks span multiple organizations, since no single central site can have management responsibility for the complete set of names. Therefore, a different approach is needed.

The limitations of the Phase IV approach were recognized when this version of DECnet was in the design phase, and work was started on the Digital Distributed Name Service (DECdns). DECdns has been available as an optional component of DECnet Phase IV for some time. It provides

- Distribution: All naming information does not have to be stored at a single point in the network.
- Replication: Information can be held in more than one place, giving resilience in the face of system or network failures.
- Dynamic updating: Information can be changed at any time.
- Automatic updating: Changed or new information is automatically propagated throughout the network.
- Hierarchical naming: A name can have multiple components to reflect an administrative or other organizational structure.

The development of the DECnet and DECdns products has been closely linked, and each is designed to make maximum use of the other. When they are used together, DECnet can provide complete autoconfiguration of a new node in the

network, such that no manager or user needs explicit knowledge of the address of a node. Once a name is assigned, the node can keep the naming service up-to-date both with the initial assignment of an address and any subsequent changes. It is also possible for a DECnet system to operate without DECdns.

The TCP/IP protocol suite also includes a naming service, with similar properties to DECdns. It is called the domain name system, or DNS. At the highest level, names are assigned by a global authority to countries and to other large groupings of organizations. Within countries, they are assigned to particular organizations such as companies. These organizations can then assign names that may have further components reflecting their internal structure.

Work on a naming service for OSI has lagged behind the other protocol suites, but the most important elements have been available since 1988 in a standard generally called X.500 (after the first of a series of CCITT recommendations that define the OSI directory). The X.500 standard defines the structure of names and the protocols to be used to access the naming service, but it does not include the mechanisms required for automatic updating and maintenance of the service itself.¹⁵ Work on standards for these functions is currently at an advanced stage. Like the DNS system for TCP/IP, the X.500 standard allocates the highest level of the structure to countries and then to organizations within countries. Its design pays particular attention to the needs of electronic mail (the X.400 protocol family). In contrast to DECdns and DNS, which assign names to computer systems, the structure of X.500 names extends to the level of naming individuals within a coherent naming framework.

DECnet supports all these naming services, in conjunction with their respective protocol stacks.

Distributed Network Management

In early computer networks, management was performed "out of band." This meant that if any communication between sites was needed to keep the network running, some means other than the network (for example, the telephone) was used. It was soon realized that much of the time, the network itself provided the most effective way to communicate management information, either to investigate a problem or to modify the configuration. DECnet has included the ability to manage itself in this way since Phase III.

The most obvious requirement for such a scheme is a protocol that can carry management information through the network. Such a protocol fits naturally into the application layer, where it can make use of the services provided by the other layers.

A further requirement is a well-defined structure for the information that is to be conveyed. A network architecture is constantly evolving, and it must be possible to add new information (for example, for a new kind of data link) into the protocol.

Finally, the specific information elements, such as the fault counters to use in conjunction with a particular protocol, must be defined.

The management model and protocol used in earlier versions of DECnet were unsuitable for the needs of Phase V due to the many different protocol combinations that were to be supported. Hence, a new management model was defined. For a long time, this was called the Entity Model and was subsequently published as Digital's Enterprise Management Architecture (EMA).¹⁶ This model takes an object-oriented approach to modeling the information needed for management. It is completely flexible and is not restricted to the management of the network itself; it has since been applied to management of the computer systems themselves.

At the same time, Digital adopted an early draft of the protocol under development for OSI management, the common management information protocol (CMIP). The structure of the CMIP protocol accommodates the flexibility allowed in EMA.

The management information needed for each protocol is defined in the same architecture document as the protocol itself. The modular structure of EMA allows this to be accomplished without conflict between management information defined for different protocols. In addition to the information specific to particular protocols (such as parameters of the protocol operation or counters), there are also representations of the relationship between protocol elements, such as user to provider.

EMA provides a clear distinction between two roles in the management of a network: the agent and the manager. The agent corresponds to the thing being managed and is part of the same system. The manager is typically elsewhere and communicates with the agent using the network and the management protocols. The manager role is taken by user interface programs. These may be simple, like the network control language (NCL), a basic command line utility appropriate for simple networks, or they may be extremely powerful.

DECmcc, for example, is a Digital product that provides the facilities appropriate to the management of networks throughout an enterprise.

If the network is being used to manage itself, the possibility exists for a kind of "deadly embrace," in which the communication path needed to fix a problem is itself unavailable due to that same problem. DECnet has been designed to minimize the likelihood and practical impact of this risk. The operation of the network layer is of vital importance in this regard. As long as a physical communication path is working, it will virtually always be able to correct a fault, even if the fault is due to a previous incorrect management operation.

The TCP/IP protocol suite also provides a management capability through the simple network management protocol (SNMP).⁷ Although both the protocol and the information model underlying it are considerably simpler than EMA, comparable facilities exist for many purposes. To the extent possible, DECnet implementations are designed to be managed through SNMP as well as through using the DECnet management protocol.

The standards for management associated with OSI protocols are still under development. Digital has made extensive contributions based on its own architecture, and the resulting standards bear a strong resemblance to EMA. Standards exist for the CMIP protocol and for the management model, but specification of the specific elements of management information needed for particular protocols have yet to be completed.

Conclusions and Future Capability

In 1974, DECnet was the first networking product to provide general-purpose, peer-to-peer communications. With the availability of Phase V, DECnet has become the first fully standards-based family of network products. It incorporates all available standards from the OSI and TCP/IP protocol suites in a way that provides the system integration and the performance traditionally associated only with proprietary network products. Achieving this migration to standards has involved a phenomenal effort, but this price has now been paid. Technology and the standards that reflect it are in a constant state of development. The future of DECnet will consist of relatively frequent and modest incremental changes that incorporate these new developments. Already major developments in areas such as naming (X.500), transaction processing, and management are finding their way into the products.

At the same time, there is an increasing need for Digital networking products to incorporate widely used, nonstandard protocols, especially for interconnection with personal computers and other desktop devices. Fortunately, the modular architecture developed for Phase V makes it relatively easy to do this in the same incremental fashion.

DECnet has changed out of all recognition from its early versions, yet it can still support the same application programs that were built in the 1970s, as well as client/server applications that are still emerging. The basic physical technology that supports networking has also undergone enormous changes, from 2,400-bit-per-second modems to Ethernet and fiber distributed data interface (FDDI), yet DECnet makes this all transparent to the user. In another 20 years we can expect these technologies to have developed as much again, or more, and we can expect too that DECnet will continue to adapt to match them.

References

1. V. Cerf and R. Kahn, "A Protocol for Packet Network Interconnection," *IEEE Transactions on Communications*, vol. COM-22 (May 1974).
2. R. Cypser, *Communications Architecture for Distributed Systems* (Reading, MA: Addison-Wesley Publishing Co., 1978).
3. *CCITT Recommendation X.25, CCITT Yellow Book*, vol. VIII.2 (Geneva: International Telecommunications Union, 1981).
4. *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).
5. *Basic Reference Model for Open Systems Interconnection*, ISO 7498:1983 (Geneva: International Organization for Standardization, 1983).
6. J. Morency, R. Pitkin, R. Jesuraj, and A. Kwong, "Modeling and Analysis of the DECnet/SNA Gateway," *Digital Technical Journal*, vol. 1, no. 9 (June 1989).
7. D. Comer, *Internetworking with TCP/IP: Principles, Protocols and Architecture* (Englewood Cliffs, NJ: Prentice-Hall, 1988).

8. G. Cobb and E. Gerberg, "Digital's Multiprotocol Routing Software Design," *Digital Technical Journal*, vol. 5, no. 1 (Winter 1993, this issue): 70-83.
9. R. Perlman, R. Callon, and M. Shand, "Routing Architecture," *Digital Technical Journal*, vol. 5, no. 1 (Winter 1993, this issue): 62-69.
10. *Information Technology: Intermediate System to Intermediate System Intra-domain Routing Information Exchange Protocol for Use in Conjunction with the Protocol for Providing the Connectionless-Mode Network Service*, ISO 10589 (Geneva: International Organization for Standardization, 1992).
11. R. Callon, *Use of OSI IS-IS for Routing in TCP/IP and Multi-Protocol Environments*, Internet Activities Board, RFC 1195 (1991).
12. *Information Processing Systems: Network Service Definition, Addendum 2: Network Layer Addressing*, ISO 8348 (Geneva: International Organization for Standardization, 1988).
13. *Information Processing Systems: End system to Intermediate System Routing Information Exchange Protocol for Use in Conjunction with the Protocol for Providing the Connectionless-Mode Network Service*, ISO 9542 (Geneva: International Organization for Standardization, 1988).
14. *Information Processing Systems: Data Communications Protocol for Providing the Connectionless-Mode Network Service*, ISO 8473 (Geneva: International Organization for Standardization, 1984).
15. CCITT IXth Plenary Assembly, "The Directory—Overview of Concepts, Models and Services," Recommendation X.500 and ISO 9594-1, *Data Communication Networks Directory: Recommendations X.500 to X.521, CCITT Blue Book*, vol. VIII.8 (Geneva: International Telecommunications Union, 1989).
16. *Enterprise Management Architecture General Description* (Maynard, MA: Digital Equipment Corporation, Order No. EK-DEMAR-GD-001, 1989).

Lawrence Yetto
Dorothy Noren Millbrandt
Yanick Pouffary
Daniel J. Ryan, Jr.
David J. Sullivan

The DECnet/OSI for OpenVMS Version 5.5 Implementation

The DECnet/OSI for OpenVMS version 5.5 product implements a functional Digital Network Architecture Phase V networking product on the OpenVMS system. This new software product ensures that all existing OpenVMS application programs utilizing published interfaces to DECnet-VAX Phase IV operate without modification over the new DECnet product. The components of DECnet/OSI for OpenVMS version 5.5 include the new interprocess communication interface. The design goals and implementation strategy were redefined for network management, the session control layer, and the transport layer. The configuration utility was structured into several files that are easy to read.

The DECnet Phase V networking software presented the DECnet-VAX development team with a major challenge. Although the Digital Network Architecture (DNA) has always corresponded to the lower layers of open systems interconnection (OSI), the Phase V architecture has substantial differences from Phase IV in many layers. For example, the session control layer now contains a global name service.¹

DECnet Phase V also added new network management requirements for all layers. In most cases, the existing Phase IV code could not be adapted to the new architecture; it had to be redesigned and rewritten. This presented the engineers with the opportunity to restructure and improve the older pieces of code that have been continually modified and enhanced since the first release of DECnet-VAX. Due to the large installed customer base, however, it also presented a huge compatibility problem. We could not simply drop the old in favor of the new; we needed to ensure that the customers' DECnet-VAX applications would continue to be supported.

This paper gives an overview of the design of the base components in the new DECnet/OSI for OpenVMS version 5.5 product. It then presents details about the internals of the network management, session control, and transport layers. Finally, the new configuration tool designed for DECnet/OSI for OpenVMS version 5.5 is discussed. Unless otherwise noted in this paper, the term DECnet/OSI for OpenVMS refers to version 5.5 of the product.

High-level Design

Numerous goals were identified during the design phase of the base components for the DECnet/OSI for OpenVMS software. Foremost among these goals was to conform to the DNA Phase V architecture and to support image-level compatibility for existing Phase IV applications. Care was also taken in the design to allow the product to be extensible to accommodate the ongoing work with industry standards.

Design Overview

The queue I/O request (\$QIO) application programming interfaces (APIs) for the VAX OSI transport service and DECnet-VAX are already defined and widely used by network applications. To ensure that existing applications would continue to work, these interfaces were modified in a compatible fashion. As a result, not all of the capabilities of Phase V could be added to the existing APIs. A new API, the interprocess communication interface (\$IPC), was developed to support all the functions defined in the Phase V session control layer. In addition, the \$IPC interface was designed to allow for future capabilities.

The \$QIO and \$IPC interfaces interpret the application's requests and communicate them to the DNA session control layer through a kernel mode system interface called session services. In the initial release of DECnet/OSI for OpenVMS, the VAX OSI

transport service joined its \$QIO interface to the stack at the network layer. The first follow-on release will fully support this API. It will be rewritten to interface directly to the common OSI transport module.

DECnet/OSI for OpenVMS implements each layer of the Phase V architecture in separate modules. These modules require a well-defined interface to communicate. This is supplied by the new interrupt-driven VAX communication interface. This interface defines the rules used by cooperating VAX communication modules to exchange information. The upper VAX communication modules consume a set of services, and the lower modules provide services. The lower VAX communication modules define the explicit messages and commands that are passed between the modules. This definition is then referred to as the lower layer's VAX communication interface. For example, the transport layer provides a service to the session control layer. Transport is the lower module, and session is the upper. The rules for how the interface works are defined by the VAX communication interface itself, but the commands and services supplied by the transport layer are defined by that layer. As a result, the interface between the session and transport

layers is referred to as the transport VAX communication interface.

To comply with the new Enterprise Management Architecture (EMA), each of the modules supplies one or more manageable entities to network management. This is accomplished by the EMA agent (EMAA) management facility. EMAA supplies both an entity interface to the individual modules and an EMAA interface to the network. This interface is discussed further in the Network Management section.

Figure 1 shows the components of the DECnet/OSI for OpenVMS product and their logical relationship to each other.

Implementation of the Modules

Each DECnet/OSI for OpenVMS base component is implemented in one of three ways. The most prominent method is through OpenVMS executive loadable images. These loadable images are all placed in the SYS\$LOADABLE_IMAGES system directory during installation and loaded as part of the NET\$STARTUP procedure, which the OpenVMS system runs during a system boot.

The two \$QIO interfaces must operate within the OpenVMS I/O subsystem. As a result, they are both coded as device drivers and loaded during

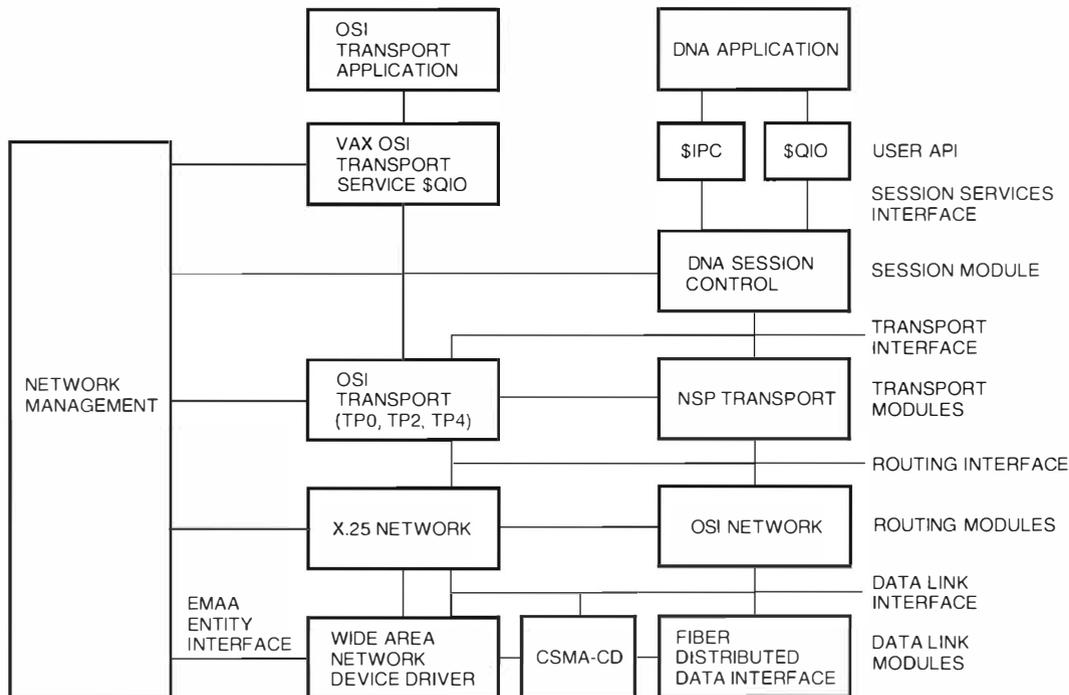


Figure 1 DECnet/OSI for OpenVMS Base Components

NET\$STARTUP by the SYSGEN utility. Once started, they can create a VAX communication interface port to the appropriate modules to process their network requests.

The third way a component can be implemented is as a standard OpenVMS image or shareable image. These images include NET\$ACP.EXE, which is started as a system process by NET\$STARTUP, and NCL.EXE, which is the utility that supplies the network control language (NCL) interface to users. Other images, such as NET\$MIRROR.EXE, are started by the network software in a separate process when a network request is received for the application.

Implementation of the Base Image

The base image, SYS\$NETWORK_SERVICES.EXE, has been present on all OpenVMS systems since version 5.4. The OpenVMS system loads this executive image early in the boot cycle. The default file shipped with OpenVMS is a stub that simply sets a system cell during initialization to indicate that the older Phase IV code is loaded. This system cell can then be interrogated through an OpenVMS system service or from a Digital Command Language (DCL) command line to determine which version of the DECnet software is loaded.

When the DECnet/OSI for OpenVMS product is installed, the base image is replaced with the Phase V version. The new image sets the system cell to indicate that Phase V is loaded. It provides a host of common services, including EMEA, to the remaining system components. It also contains the code used to implement the Phase V node agent required by EMA on each node. Each of the remaining DECnet/OSI for OpenVMS components makes use of the base image by vectoring through a system cell to the desired function.

Network Item Lists

The DECnet/OSI for OpenVMS modules pass large amounts of data between themselves. This exchange requires an efficient means to encode and move the data. Conversions are expensive operations; therefore a decision was made to use the same structure for all the interfaces within the base components. The structure chosen, a network item list, is a simple length/tag/value arrangement in which the tags are defined in a common area between sharing modules. Network item lists are very easily extended as new functions are added to the software. Since they contain no absolute

addresses, they are also position independent. This has the advantage of making it easy to copy or move them when necessary.

Network item lists are used between all VAX communication modules, by EMEA, and by the session services interface. They are also presented to user-written applications through the \$IPC interface, thus allowing the interface to be expanded as more protocols and standards are implemented in the DECnet network.

Network Management

This section discusses the DECnet/OSI for OpenVMS network management design and network management functions implemented in Phase V.

Network Management Design

The key to Phase V network management design is the EMA Entity Model, which defines the standard management structure, syntax, and interface to be used by each manageable object. The DECnet/OSI for OpenVMS EMA framework is built on this model and defines the components required for a system manager to perform actions on managed objects, both locally and across a network. The EMA framework consists of the following components.

- A director interface, through which user commands called directives are issued
- A management protocol module that carries directives to the node where the object to be managed resides
- An agent that decodes the directive into specific actions and passes that information to the managed object
- An entity, the object to be managed

For a full understanding of the DECnet/OSI for OpenVMS network management implementation, the reader should first understand the EMA model. Details on the EMA model can be found in the paper on management architecture in this issue.²

In the DECnet/OSI for OpenVMS network management design, the components and their division of function generally follow the EMA framework. There are, however, a few exceptions. Figure 2 shows the DECnet/OSI for OpenVMS components that implement the EMA model and other Phase V management functions.

The NCL utility provides the EMA director function. The NCL image processes user commands into management directives. It also displays the responses that are returned.

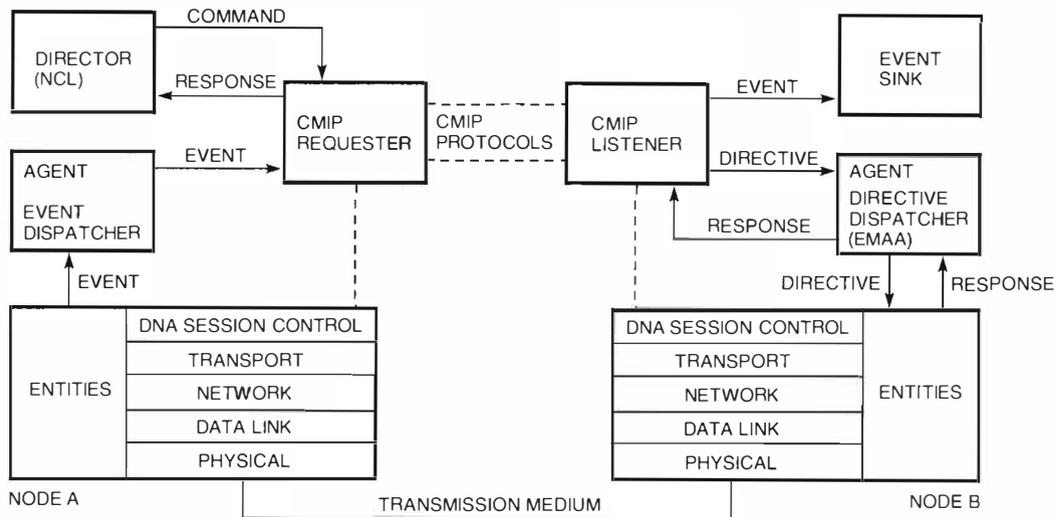


Figure 2 Network Management Components

The common management information protocol (CMIP) requester library routines provide part of the management protocol module functions. These include encoding a management directive into CMIP, transmitting it to the designated node, and receiving the response. The CMIP requester routines are implemented as part of NCL, not as a separate management protocol module.

A CMIP listener server process, CML.EXE, provides the remainder of the management protocol module function. It receives a management directive and passes it to the agent. When the agent returns a response, CML transmits the response to the originating node.

The DECnet/OSI for OpenVMS EMA agent, EMMA, accepts management directives from CML, dispatches them to the requested entity, and returns responses to CML. EMMA also extends this concept by actually performing the management directives in some cases.

Entities are not strictly a part of network management. They do, however, receive management directives from EMMA in DECnet/OSI for OpenVMS. They must be able to carry out the directives and return the results of the operation to EMMA.

In DECnet Phase V, an event is the occurrence of an architecturally defined normal or abnormal condition. Events detected by entities are posted to an event dispatcher, which passes them to a local or remote event sink. If remote, a CMIP event protocol is used. In DECnet/OSI for OpenVMS, the event dispatcher image, NET\$EVENT_DISPATCHER.EXE,

implements the event dispatching and event sink functions.

The data dictionary is a binary compilation of architecturally defined codes for all known Phase V management entities, the manageable attributes of each entity, and the actions that can be performed. It also contains information necessary to encode this information into Abstract Syntax Notation Number 1 (ASN.1), required for the CMIP protocol.

Finally, there is the maintenance operations protocol (MOP). Although MOP is not an EMA component, it is a component of DNA. It performs low-level network operations such as down-line loading and up-line dumping.

Network Management Implementation

The most visible differences between DECnet Phase IV and DECnet Phase V arise from adherence to the EMA architecture. This section discusses the replacement functions implemented in Phase V.

The NCL Utility The network control program has been replaced in Phase V with the NCL utility. NCL provides a highly structured management syntax that maps directly to the EMA specifications for each compliant entity. In an NCL command, the hierarchy of entities from the node entity to the subentity being managed must be specified. For example, the following command shows the local area network (LAN) address attribute of a routing circuit adjacency entity.

```
NCL> Show Node DEC:.zko.IIium -  
Routing Circuit lan-0 Adjacency -  
rtg$0002 LAN Address
```

The command contains the node entity name, DEC:.zko.IIium; the module entity within the node, routing; the name of the circuit subentity of routing, lan-0; the name of the adjacency subentity of circuit, rtg\$0002; and finally the attribute name.

To issue management commands from a DECnet/OSI for OpenVMS system, a user invokes the NCL utility. NCL parses commands into fragments called tokens, containing ASCII strings. It uses the data dictionary to translate these into management codes for directives, entities, and attributes. NCL then constructs a network item list from this information and invokes the CMIP requester send function.

CMIP requester functions are implemented as a set of library routines that are linked with the NCL utility. Underneath this caller interface, the CMIP routines establish a connection over DNA session control to the destination node's CMIP listener. The directive is then encoded into a CMIP message and passed to the destination.

NCL now posts the first CMIP requester receive call. More than one receive call may be needed to obtain all the response data. As soon as a partial response is available, the receive function decodes the CMIP messages into network item lists and passes them back to NCL. NCL translates these into displayable text and values and directs the output to the user's terminal or a log file. If the partial response is not complete, NCL then loops and issues another call to the CMIP requester receive function.

The CMIP requester functions are optimized for the local node case. If the destination node is specified as "0" (the local node), the CMIP requester functions interface directly to the EMAA interface, skipping the CMIP encoding, decoding, and the round trip across the network.

The CMIP Listener The CMIP listener is implemented as a server process, similar to the Phase IV network management listener. When an incoming connection request for CML is received, a process is created to run the CML image. The CML image utilizes the DNA session control interface to accept the connection and receive the CMIP encoded directive. It then uses the data dictionary to decode the message into a network item list. EMAA is then invoked to process the directive and return any required response from the entity. Once CML

has received all portions of the response from EMAA, encoded them into CMIP, and transmitted them back to the requesting node, the CML image terminates.

EMAA, the EMA Agent The management structure imposed by EMA contains common directives that must be supported by all entities. A design goal for EMAA was to provide a common management facility with support for common operations such as show or set. EMAA can perform these functions against an entity's management data structures, thereby freeing each entity from separately implementing them and simplifying the entity's code requirements. This approach was successfully implemented, though at the cost of a more complex agent implementation and a set of registration macro instructions colloquially known as the "macros from hell."

The above interface between EMAA and the entities is known as the full interface. Not all development groups' coding entities were interested in this approach; thus, EMAA also provides a basic interface. An entity specifies which interface to use during its initialization when it registers with EMAA. For an entity that uses the basic interface, EMAA simply passes the directive information to the designated entity and expects response data returned.

The choice of interface must be made by the module-level entity. If the entity uses the full interface, it must register its management structure, including all subentities and attributes, with EMAA. For these entities, EMAA processes the network item list passed by CML. It creates a data structure for each subentity instance, specifying the attributes, any values supplied, and the actions to be performed. EMAA passes this to the designated entity, which uses tables set up during initialization to call the appropriate action routine for the directive. By default, these action routines are set up as callbacks into EMAA itself, thereby allowing EMAA to perform the task. With either the basic or the full interface, a separate response is required for each subentity instance specified by a directive. EMAA calls CML iteratively through a coroutine call to pass response data back to CML.

The Event Dispatcher Phase IV event logging allowed events to be sent to a sink on one node. In Phase V, the event dispatcher supports multiple sinks that can be local or on any number of remote nodes. Event filtering can be applied on the outbound streams of events, filtering events before

they are transmitted to a sink. This provides a mechanism to direct different types of events to different sinks.

An event sink is the destination for an event message. A node can have multiple sinks, each accepting events from any number of remote nodes. Event filtering can be applied to the inbound streams of events at the event sink. An event message that passes is sent to the sink, which uses the data dictionary to format it into ASCII character strings. It is then output to the sink client, which may be a console, printer, or file.

An optimization is used when an event is generated on a node and the destination sink is on the same node. In this case, the event bypasses the outbound stream and is queued directly to the event sink. The DECnet/OSI for OpenVMS product, in the default configuration for a local node, defines one outbound stream directed to a sink on the local node and defines the console as the sink client.

An event relay provides compatibility with Phase IV nodes. This important function permits a Phase V event sink to log messages from Phase IV or Phase V DECnet systems. Event relay is a session control application that listens for DECnet Phase IV event messages. It encapsulates each Phase IV event message in a Phase V event message and posts it to the event dispatcher, using the same service that other DECnet/OSI for OpenVMS entities use to post events.

Maintenance Operations Protocol The NET\$MOP process is the DECnet/OSI for OpenVMS implementation of the DNA maintenance operations protocol. MOP uses the services of the local and wide area data link device drivers to perform low-level network operations. MOP can down-line load an operating system image to a VMScluster satellite node and respond to remote requests from a network device to down-line load or up-line dump an image. MOP also supports management directives that allow a system manager to load or boot a remote device, monitor system identification messages, perform data link loopback tests, or open a terminal I/O communications channel to a device's console program.

The primary design goal of the MOP implementation was to respond quickly and with low system overhead to remote requests from devices to down-line load an image. In some network configurations, a power failure and restoration can cause hundreds of devices to request a down-line load at the same time. The Phase IV implementation was known to have difficulty handling this, so the new

implementation of MOP was designed for multi-threaded operation. This means there is only one MOP process per node, and it processes multiple concurrent operations by creating a separate thread for each management directive, program request, or dump request received. Moreover, all management data required to service MOP requests is contained in MOP-specific management data structures, designed to be searched quickly. When a request is received, MOP can promptly ascertain whether the required information to service the request is available and make a response.

Session Control Implementation

The design of the DECnet/OSI for OpenVMS session control layer is based on goals defined by both the session control architecture and the DECnet user community. These goals include

- **Compatibility.** The DECnet-VAX product has a large customer base with major investments in DNA applications. The session control layer supports these applications without requiring a relink of the object code.
- **Performance.** Transmit and receive operations across the network must be as efficient as possible. Minimal overhead is introduced by the session control layer in making each transport protocol available to applications.
- **Extensible.** The session control layer design allows for future additions to the architecture.
- **New features.** The session control layer takes full advantage of the new naming and addressing capabilities of Phase V DNA.
- **Improved management.** The session control layer complies with EMA, allowing it to be managed from anywhere throughout the network.

Session Control Design

The session control layer is divided into several logical components, \$QIO, \$IPC, NET\$ACP, common services, and network management. \$QIO and \$IPC provide the APIs required to communicate across the network. \$QIO is fully compatible with all Phase IV DECnet-VAX applications; however, it does not allow access to the full set of features available in DECnet/OSI for OpenVMS. These new features, and any future additions, are available only through the new \$IPC interface.

The two APIs are consumers of session control services provided by the common services

component. This component provides all the network functions defined in Phase V to the APIs above it. In order to do this, the common services component makes use of both the NET\$ACP and network management portions of the session control layer.

Figure 3 shows the session layer components and their relationships to each other.

Session Control APIs

DECnet Phase IV restricted node names to six characters in length. In DECnet-VAX the \$QIO interface was the only means by which an application could make calls to the session control layer. This interface also enforced the six-character name limit. With the advent of Phase V, this restriction no longer applies. It is possible for a node running Phase V to be unreachable by a Phase IV-style six-character node name. As a consequence, the \$QIO interface was extended to allow full name representations of a node.

The \$IPC interface is a new interface that incorporates all the functions of the \$QIO interface, along with extensions made to the session control architecture. This item-list-driven interface provides a cleaner, more extensible interface and allows for easy conversion of \$QIO applications. The \$QIO interface uses a network control block (NCB) and a network function block (NFB) to hold data. This data is easily mapped to items in a network item list. Also, the function codes used by \$QIO can be easily mapped to \$IPC function codes. As new requirements arise, supported items can be added to the list without impacting the existing values.

The \$IPC interface also supplies some new features not available in \$QIO. Phase V DNA uses the Digital Distributed Name Service (DECdns) to store information about nodes and applications in a global namespace. Once an application declares

itself in the global namespace, \$IPC enables session control to maintain its address attribute. This address attribute contains all the information necessary to define where the application resides on the network. \$IPC can then be used by the client side of an application to connect to a server through a single global name, instead of using a node name and application name pair. This enables the client side of an application to communicate with its server without knowing where the server currently resides.

\$IPC supports a new means of accessing a node by its address. In Phase IV, addresses were limited to 63 areas with 1,023 nodes in each area. The address of each node could be represented with a 16-bit integer. The \$QIO interface supports a form of node name in which the 16-bit address is converted into the ASCII representation of the decimal equivalent. This is not sufficient to address all Phase V nodes, so a new function called "connect-by-address tower" is available through \$IPC. The address tower is discussed further in the Common Services Component section.

Yet another feature of \$IPC is the ability to translate a node's address into the name of the node as registered in the global namespace. In Phase IV the address-to-name translation was a management function. Furthermore, the translation was local to the node on which it was performed.

Session Control Network Management

The session control layer makes use of the full EMAA entity interface to support all entities defined by the session control architecture. These include the session control entity itself, as well as the application, transport service, port, and tower maintenance subentities. Each of these entities contains timers, flags, and other control information used by the session control layer during its operation. They also contain counters for the events generated by the session control layer.

The application subentity is of special interest. This entity is the equivalent of the Phase IV object database. It allows the system manager to register an application with session control to make it available for incoming connections. This entity is also used to control the operation of the application and select the types of connections that can be sent or received by it.

Common Services Component

The common services component is the hub for session control. It is responsible for performing

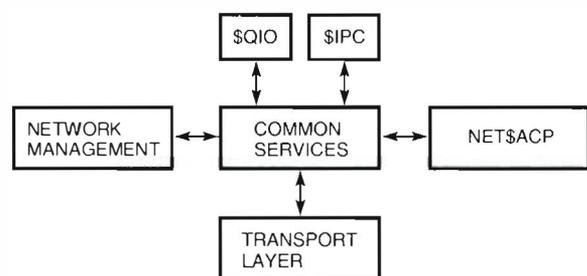


Figure 3 Session Design

tasks that are not specific to the \$IPC or \$QIO interfaces. These tasks include managing transport connections on behalf of session control users, mapping from a DECdns object name to addresses, selecting communication protocols supported by both the local and remote end systems, maintaining the protocol and address information corresponding to local objects in the namespace, and activating (or creating) processes to service incoming connect requests.

The NET\$ACP process is used to provide the common services component with process context. The NET\$ACP image itself is nothing more than a set of queues and an idle loop. When the session control layer is loaded, it creates user-mode and kernel-mode tasks. A queue is assigned for each task, and the NET\$ACP process attaches to the task when it is started. When the session component needs to execute in the context of a process and not on the interrupt stack, it builds a work queue entry, queues it to the appropriate task queue, and wakes up the NET\$ACP. The NET\$ACP finds the address of the desired routine in the work queue entry and executes it. This routine can be located anywhere that is addressable by the process, but it is usually contained within the session control loadable image. The common services component makes use of the NET\$ACP for reading files, creating network processes, and making calls to the DECdns clerk. It also makes use of the process for functions that require large amounts of memory. By performing these tasks in the NET\$ACP process, session control is able to use process virtual memory even though it is implemented as an executive loadable image.

The tower set data structure plays a key role in session control. A tower set consists of one or more towers. Each tower represents a protocol stack and is composed of three or more floors, as shown in Figure 4. The lowest floors in the tower correspond to the DNA routing, transport, and session control layers; they are used to identify protocol and associated address information to be used

at that layer. When viewed as a whole, the tower set describes a combination of protocols supported on a node. The session control layer on every DECnet/OSI for OpenVMS system not only uses this information to communicate with remote nodes, but is also responsible for building a tower set to represent that local system. Once built, this tower set is placed in the namespace as the attribute for the node.

The session control interfaces allow the user to specify a node in many ways. A node can be specified as a Phase IV-style node name, a Phase IV-style address, a DECdns full name, or a tower set. The three forms of name representations are mapped to the corresponding tower set by making calls to the DECdns clerk to obtain the node's tower set attribute. Once the tower set is in hand, it can be used to communicate with the session control layer on the remote node.

The tower set for a remote node and the tower set for the local node are used in conjunction to determine if both nodes support a common tower. If a common tower is found, session control attempts to establish a connection to the remote node using that tower. If the connection fails, the comparison continues. If another matching tower is found, the connection attempt is repeated. This continues until the connection is established or the tower sets are exhausted.

Use of DECdns

The session control layer uses DECdns objects for all global naming. These objects are used in two different ways: they can represent a node or a global application. A node object is a global representation of a node in a DECdns namespace. Each node object contains attributes that identify the location of a node. Foremost in this list of attributes is the DNA\$Towers attribute. The DNA\$Towers attribute contains the tower set for the node and is written automatically by the session control layer when DECnet/OSI for OpenVMS is configured and started. Once created, this attribute is updated by session

FLOOR N	APPLICATION-DEFINED FLOORS	
FLOOR 3	SESSION PROTOCOL	SESSION ADDRESS INFORMATION
FLOOR 2	TRANSPORT PROTOCOL	TRANSPORT ADDRESS INFORMATION
FLOOR 1	ROUTING PROTOCOL	ROUTING ADDRESS INFORMATION

Figure 4 Tower Design

control to reflect the current supported towers for the node.

When the session control layer builds the tower set for the DECdns node object, it creates a tower for each combination of supported protocols and network addresses on the node. If the node supports two transports and three network addresses, the tower set is generated with six towers. It always places the CML application protocol floor on top of the session control floor. The address information for the session control floor is then set to address the CML application. The transport address information is set to address DNA session control, and the routing information of each tower in the set is set to one of the supported network addresses for the node.

The node object DNA\$Towers attribute contains data that completely describes the node. Since session control supports node addresses and Phase IV-style node names, soft links are created in the namespace to map from a Phase V network service access point (NSAP) or a Phase IV-style node name (node synonym) to the node object. These links can then be used by the session control layer as alternate paths to the node object.

An application object is a global representation of an application. The DNA\$Towers attribute of this object contains a set of address towers used to address the application. The routing and transport floors for each tower in this set are used in the same manner as for the node object. The address information in the session floor, however, addresses the application, not CML. Once set, the information in this tower set is not maintained unless the application issues a register object call through the \$IPC interface. If this is done, session control maintains the tower in the same manner as it does for the node object.

Transport Implementation

The DECnet/OSI for OpenVMS product supports two transport protocols: the open systems interconnection transport protocol (OSI TP) and the network service protocol (NSP). Each transport protocol, or group of logically associated protocols, is bundled as a separate but equivalent VAX communication module. This approach accomplishes many goals. The more notable ones include

- Isolating each module as a pure transport engine
- Defining and enforcing a common transport user interface to all transports

- Providing extensible constructs for future transport protocols, i.e., providing a set of transport service libraries
- Eliminating previous duplication in adjacent layers (session and network routing layers)
- Providing backward compatibility with existing Phase IV transport protocol engines (NETDRIVER/NSP and VAX OSI transport service)

Transport Layer Design

A transport VAX communication module has two components, a protocol engine and the transport service libraries. The service libraries are common code between modules and are linked together with each engine to form an executive loadable image. The three elements of DECnet/OSI for OpenVMS transport, the NSP protocol engine, the OSI protocol engine, and the transport service libraries, are linked into two images. Figure 5 shows the relationship of these elements.

The specific functions provided by a transport engine depend on the protocol. The generic role of NSP and the OSI transport layer is to provide a reliable, sequential, connection-oriented service for use by a session control layer. The design provides a common transport interface to both NSP and the OSI transport layer. This enables NSP and OSI transport (class 4) to be used interchangeably as a DNA transport. As future transport protocols are developed, they can be easily added into this design.

The DECnet/OSI for OpenVMS transport design places common functions in the service libraries for use by any protocol engine that needs them. Any functions that are not specific to a protocol are performed in these libraries. Separating these functions enables new protocols to be implemented more quickly and allows operating-system-specific details to be hidden from the engines.

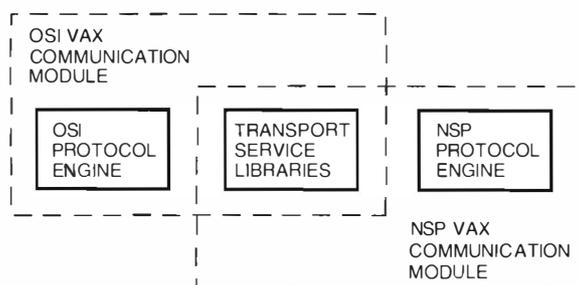


Figure 5 Logical Transport Components

The NSP transport VAX communication module operates only in the DNA stack and supports only DNA session control. Due to an essentially unchanged wire protocol, NSP is completely compatible with Phase IV implementations.

The OSI transport VAX communication module implements the International Organization for Standardization (ISO) 8073 classes 0, 2, and 4 protocols. It can operate on a pure OSI stack in a multi-vendor environment. The OSI transport is also completely compatible with the Phase IV VAX OSI transport service implementation and operates on the DNA stack supporting DNA session control.

Transport Engines The transport VAX communication modules provide a transport connection (logical link) service to the session layer. The connection management is designed to ensure that data on each logical link is handled independently from data on other logical links. Data belonging to different transport connections is never mixed, nor does a blockage of data flow on one connection prevent data from being handled on another.

The transport VAX communication modules are state table driven. Each transport engine uses a state/event matrix to determine the address of an appropriate action routine to execute for any state/event combination. As a transport connection changes state, it keeps a histogram of state transitions and events processed.

Service Libraries The following functions are common to many protocols and are implemented in the service libraries.

- Transfer of normal data and expedited data from transmit buffers to receive buffers
- Fragmentation of large messages into smaller messages for transmission and the reconstruction of the complete message from the received fragments
- Detection and recovery from loss, duplication, corruption, and misordering introduced by lower layers

The key transport service library is the data transfer library. This library gives a transport engine the capability to perform data segmentation and reassembly. Segmentation is the process of breaking a large user data message into multiple, smaller messages (segments) for transmission. Reassembly is the process of reconstructing a complete user data message from the received segments. To use the data transfer library, a protocol engine must

provide a set of action routines. These action routines hold the protocol-specific logic to be applied to the data handling process.

Network Services Phase V provides two types of network services: connectionless (CLNS) and connection-oriented (CONS). CLNS offers a datagram facility, in which each message is routed to its destination independently of any other. CONS establishes logical connections in the network layer over which transport messages are then transmitted.

Transport running over CLNS has a flexible interface. It opens an association with the CLNS layer and is then able to solicit the CLNS layer to enter a transport protocol data unit (TPDU) into the network. When admission is granted, transport sends as many TPDU's as possible at that time. Incoming TPDU's are posted to transport as they are received by the CLNS layer. Both NSP and OSI transports run over the CLNS layer.

Transport running over CONS has a more rigid interface. Once a network connection is established with the CONS layer, each transport request has to be completed by the CONS layer. Thus transport, when running over CONS, is not able to transmit all its TPDU's at once. Each transmit must be completed back to transport before the next can commence. Also, if transport is to receive incoming TPDU's, a read must be posted to the CONS layer. The OSI transport runs over the CONS layer, but the NSP protocol was designed specifically for CLNS and does not operate over CONS.

Differences between Phase IV and Phase V Transport Protocol Engines

Flow control policy is an important difference between the VAX OSI transport service and the DECnet/OSI for OpenVMS implementation. The VAX OSI transport service implements a pessimistic policy that never allocates credit representing resources it does not have. The OSI transport protocol, on the other hand, implements a more optimistic policy that takes advantage of buffering available in the pipeline and the variance in data flow on different transport connections. It makes the assumption that transport connections do not consume all allocated credit at the same time. Other enhancements to the OSI transport protocol include conformance to EMA network management, compliance with the most recent ISO specifications, and participation in DECnet/OSI for OpenVMS VMScluster Alias.

The two main differences between the Phase IV and Phase V NSP implementations are conformance to the EMA management model, and, once again, flow control. In Phase IV, NSP does not request flow control and uses an XON/XOFF mechanism. This results in large fluctuations in throughput. Phase V NSP has been enhanced to request segment flow control. This mechanism enables each side of a transport to determine when it can send data segments. Due to this difference in flow control policy, Phase V NSP throughput converges to a maximum value.

Future Direction of Transports

The DECnet/OSI for OpenVMS transport design provides a common transport user interface to all transports and places common functions in the transport service libraries. This approach provides extensibility; it allows future transports to be easily incorporated as they emerge in the industry. This common interface can also be used to provide an API that interfaces directly to a transport. DECnet/OSI for OpenVMS engineering is currently looking at providing such an API.

Configuration

Design on the new configuration tools was started by collecting user comments about the Phase IV tools and desirable features for the new tool. This data was collected from customer communication at DECUS, through internal notes files, and through internet news groups.

The first goal agreed upon was to create configuration files that are easy to read; inexperienced Phase V network managers may be required to read and understand these files. Next, the tool must be structured. The configuration is divided into several files with recognizable file names rather than one potentially unmanageable one. Each file contains the initialization commands needed to initialize one network entity. Finally, the configuration tool should be extensible. New commands, entities, or other information can easily be added to the configuration.

Configuration Design

The main configuration tool is a DCL command procedure (NET\$CONFIGURE.COM). This procedure generates NCL script files, which are executed during network start-up, to initialize the network. In general, each script file initializes one entity within DECnet/OSI for OpenVMS. It is possible, however,

for scripts to contain information for numerous entities. For example, the NSP transport initialization script contains commands to create an instance of the session control transport service provider entity, which enables the session layer to use the protocol. The procedure can extract information about the configuration by using the NET\$CONVERT_DATABASE utility to translate an existing Phase IV configuration contained in the Phase IV permanent databases. Alternatively, it can prompt the user for the information needed to allow basic operation of the node.

The first time NET\$CONFIGURE is executed, all the questions, except for the node's full name and its Phase IV address, have default choices. If the defaults are chosen, the node operates properly once the network has started. When appropriate, NET\$CONFIGURE also calls other configuration tools to configure the DECdns client and the Digital Distributed Time Service (DECdts), and to perform various network transition functions.

Once the initial configuration has been performed, customization of components is available. Subsequent execution of the NET\$CONFIGURE procedure will present the user with a menu that allows specific subsections of the configuration to be done, for example, adding or deleting MOP clients or session control applications, changing the name of the node, or controlling the use of communications devices.

General help is available while running NET\$CONFIGURE. If the user does not understand any individual query, responding with a "?" (question mark) provides a brief explanation.

The scripts created by NET\$CONFIGURE are computed. A checksum is computed by NET\$CONFIGURE for each script file, and it is stored in a database along with the answers entered for all other configuration questions. This allows the NET\$CONFIGURE procedure to detect whether a script has been modified by an outside source. If this condition is detected, NET\$CONFIGURE warns the user that user-specific changes made to the particular script may be lost.

If a user has modified the NCL scripts, NET\$CONFIGURE cannot guarantee that the information will be retained after future executions of the procedure. An attempt is made to maintain the changes across new versions. In all cases, previous scripts are renamed before the new scripts are generated. This allows the user to verify that customized change was transferred to the new script.

If not, the saved version can be used to manually replace the change.

Node Configuration NET\$CONFIGURE asks only one question that is directly related to the node entity. It asks for the node's DECdns full name and sets the node's name. Since the namespace nickname is a required component of the full name answer, it also allows the procedure to determine the namespace in which to configure DECdns.

The node synonym default is generated by using the first six characters of the last simple name of the node's full name. If the user entered the full name, USN:.Norfolk.Destroyer.Spruance.DD125, the synonym default would be DD125. The user is free to change this information as long as the response is a legal Phase IV-style name. If present, the transition tools make use of this synonym when the node is registered in the DECdns namespace.

Data Link/Routing The NET\$CONFIGURE procedure contains a table of all valid data link devices supported by DECnet/OSI for OpenVMS. When the data link/routing configuration module is called, the system configuration is scanned. Any valid devices found on the system are presented to the user for addition to the configuration. The only exceptions are asynchronous data link devices. The user must specifically request asynchronous support for these devices to be configured.

Configuration is mandatory for broadcast data link media since these devices are shareable and users other than DECnet/OSI for OpenVMS may request the device. For synchronous devices, the user has the choice to configure the device for use by DECnet/OSI for OpenVMS. If a device is configured, a choice between the Digital data communications message protocol (DDCMP) or high-level data link control (HDLC) as data link protocol must also be made.

Each data link device configured requires a name for the device and a name for the corresponding routing circuit. The defaults for these names are generated by using the protocol name, e.g., carrier sense multiple access-collision detection (CSMA-CD), HDLC, or DDCMP, along with a unit number. The user may override the default with any valid simple name. This allows the user to set the data link and routing circuit names to be more descriptive in their environment; for example, HDLC_SYNC_TO_BOSTON for a data link and CONNECTION_TO_BOSTON_DR500 for a routing circuit.

Transport/Session Control NET\$CONFIGURE supports the NSP and OSI transports. The procedure configures both transports by default, but allows the user to select only one. Commands are generated in the start-up scripts to initialize both the transports and the session control transport service provider entity instances, which allow the session control layer to use them.

If OSI transport is configured, default templates are created to allow the installation verification procedures for the OSI applications to operate successfully. The user also has the option of creating specific connection option templates for use with OSI applications.

All default session control applications, e.g., file access listener (FAL), mail, or phone, are configured in the same way as they are with the DECnet-VAX Phase IV configuration tool. The user has the option to allow access to each application through a default account or not. The only queries made by the configuration tool are about the creation of the user account for the application.

DECdts Configuration The DECdts configuration is performed by a call to the DTSS\$CONFIGURE procedure. DTSS\$CONFIGURE prompts the user to choose between universal coordinated time (UTC) or local time, which is UTC plus or minus the time-zone differential factor (TDF). If local time is chosen, then the procedure prompts for the continent and time zone on that continent to use. This information is needed to compute the TDF. The DTSS\$CONFIGURE tool creates its own NCL scripts. These scripts are not maintained by NET\$CONFIGURE, and no checksums are computed or stored for them.

Configuration To configure DECdns, the network software must be in operation so that the DECdns software may use it. The NET\$CONFIGURE procedure attempts to start the network once it has created the necessary scripts. Once the network has been started, the NET\$CONFIGURE procedure calls DNS\$CONFIGURE, passing it the node full name that was entered by the user. The full name contains the namespace nickname that the user wishes to use. DNS\$CONFIGURE then uses the DECdns advertiser to listen on the broadcast media for a name server that is advertising the same namespace nickname. If a match is made, DECdns creates an initialization NCL script with the needed instructions to configure the DECdns clerk at the next system boot. It then

tells the advertiser to configure against the same namespace.

If the namespace nickname cannot be matched, the user is given alternatives. First, a list of the current namespaces advertised on the broadcast media, along with the LOCAL: namespace is offered. LOCAL: is a special case used in lieu of the standard client-server configuration. The LOCAL namespace makes use of the client cache to store a small number of nodes locally.

If a choice is not made from the list, the user is queried to see if an attempt should be made to configure to a name server that may be located on a data link other than the broadcast media. If yes, then a valid address must be provided to the DNS\$CONFIGURE tool so that it may connect to the name server on the remote node.

If no options are chosen at this point, a final choice of creating a name server on the local node is presented. Since DECnet/OSI for OpenVMS must configure the DEC:dns clerk, if the answer is still no, the procedure returns to the original list of known namespaces and starts again.

Transition Tools Once DEC:dns is configured, the transition tools are used to create the correct namespace directory configuration. If a new namespace has been created and selected for use, the tools populate the directories with the node information from the Phase IV DECnet database found on the system. Most often, the tools simply register the node with the DEC:dns name server along with the node synonym that was provided by the user during the node configuration portion of NET\$CONFIGURE.

The transition tools also assist the user when renaming the node or changing from one namespace to another. They copy subdirectory information from the node's old DEC:dns directory to the new directory structure on the new namespace or within the same namespace, if the user only changed the node's name.

Summary

The DECnet/OSI for OpenVMS version 5.5 product implements all layers of the DNA Phase V architecture. This extends the OpenVMS system to a new degree of network access by supplying standard OSI protocols. The product also protects the large investment in network software that OpenVMS users currently hold. This is done by fully supporting the extensive selection of applications available

for Phase IV DECnet-VAX. In addition, the design of DECnet/OSI for OpenVMS is structured in a way that will ease the introduction of new standards as they come available.

Acknowledgments

Throughout the course of this project, many people have helped in the design, implementation, and documentation of the product. We would like to thank all those people for all their help. We would also like to extend a special thanks to all members of the bobsled team. Without them, this product would never have come to be.

References

1. J. Harper, "Overview of Digital's Open Networking," *Digital Technical Journal*, vol. 5, no. 1 (Winter 1993, this issue): 12-21.
2. M. Saylor, F. Dolan, and D. Shurtleff, "Network Management," *Digital Technical Journal*, vol. 5, no. 1 (Winter 1993, this issue): 117-129.

The ULTRIX Implementation of DECnet/OSI

The DECnet/OSI for ULTRIX software was developed to allow the ULTRIX operating system and ULTRIX workstation software systems to operate in a multivendor, multiprotocol network based on open standards. It operates in a complex networking environment that includes OSI, DECnet Phase IV, X.25, and TCP/IP protocols. BSD sockets and protocol switch tables provide the entry points that define interfaces for protocol modules. The DECnet/OSI for ULTRIX software incorporates Digital's Enterprise Management Architecture, which provides a framework on which to consistently manage the various components of a distributed system. The DECnet/OSI for ULTRIX software provides a set of powerful tools and a system that can be extended to include new functions as they are incorporated in the OSI standard.

DECnet/OSI for ULTRIX is an end system implementation that supports the open systems interconnection (OSI) protocol through the Digital Networking Architecture (DNA) Phase V software. This implementation provides several features and programming environments that are consistent with the UNIX system philosophy of networking. Ease of use, extensibility, and portability were key design goals during product development. Operation of DECnet/OSI for ULTRIX software in a complex networking environment provides coexistence and interaction with the transmission control protocol/internet protocol (TCP/IP), DECnet Phase IV, X.25, and multivendor OSI networks.

The paper "Overview of Digital's Open Networking" (in this issue) provides a suitable introduction to DECnet/OSI concepts.¹ For more details concerning standard Berkeley Software Distribution (BSD) networking concepts, the reader is referred to the general references listed at the end of this paper.

This paper provides an overview of DECnet/OSI for ULTRIX software. It discusses some of the design decisions made during product development, including the use of protocol switch tables. It describes the system's five communication domains, emphasizing the X.25, data link, and OSI domains. The paper continues with a discussion of application programming interfaces, interfaces into kernel modules, and a network management interface established for extensibility. It concludes

with a description of network management and network configuration.

System Overview

DECnet/OSI for ULTRIX is an end system implementation of the OSI network architecture and Digital's DNA Phase V. The DNA Phase V architecture provides a framework for incorporating OSI protocols as defined by the International Organization for Standardization (ISO) into DECnet/OSI products. DECnet/OSI for ULTRIX software is integrated into the ULTRIX kernel and layered on existing ULTRIX interfaces. This software allows the ULTRIX operating system and ULTRIX workstation software (UWS) systems to operate in a multivendor, multiprotocol network based on open standards.

The DECnet/OSI for ULTRIX software provides the following network services:

- Base networking software, which includes transport services, network layer services, X.25, and local area and wide area device driver support as described in the ISO Reference Model and DNA.²
- Network management software, incorporating the Digital Enterprise Management Architecture.
- Application programming interfaces to support user development of distributed applications.
- DECnet application software. DNA session control bridges DECnet applications such as file transfer (dcp,dls,drm), remote login (dlogin), and mail to transport layer services.

- DECdns, Digital's distributed name service, which provides a location-independent naming facility. This service is used by DNA session control to provide node name-to-address translations.³
- Digital's distributed time service, DECdts. This time synchronization service is required by many distributed applications such as DECdns to maintain a consistent time base for their operations.
- OSI applications software, including file transfer, access, and management (FTAM) and virtual terminal protocol (VTP) support.

System Goals and Development

A major goal of DECnet/OSI for ULTRIX was to support large multivendor, multiprotocol networks, including coexistence of OSI and TCP/IP on an ULTRIX UWS system. Coexistence includes the ability to share system resources and to provide a consistent set of services to users of both the OSI and internet protocols. Another goal was to provide connectivity between OSI and TCP/IP networks through the implementations of gateways and hybrid stacks.

Interoperability between DECnet/OSI and DECnet Phase IV products was required to maintain connectivity during network transition to OSI. A framework for the development of new OSI applications such as FTAM was another requirement. As in the DECnet-ULTRIX Phase IV implementation, programming and user interfaces needed to be consistent with the ULTRIX and UNIX systems environment.

Wherever possible, code was to be shared with other development projects. For this reason, software development engineers used the C programming language and aimed to produce a portable implementation. This was particularly important for the X.25 implementation, which would be used in other products. The code was structured to minimize system-specific references and dependencies. Code that interfaced directly to the BSD system was isolated in separate modules, and use of system-specific devices such as timers and buffers was hidden behind generic macros or subroutines.

In addition, the software was designed to be extensible so that future OSI protocols could be added. To achieve extensibility, interfaces were established between the various components. These include application programming interfaces, interfaces into each kernel module, and a network management interface. New protocols could be more easily added by supporting these interfaces.

DECnet/OSI for ULTRIX development began with a collection of eight distinct projects, each with its own goals, schedules, and priorities. These projects were developed across engineering organizations, and spanned three continents. They consisted of X.25, wide area device drivers, FTAM, VTP, DECdts, DECdns, OSI applications kernel (OSAK), and the DECnet/OSI base components.

Early in development, it was realized that no individual project could be successful without achieving success at a systems level for the DECnet/OSI for ULTRIX product. This realization caused a change in the way the DECnet/OSI for ULTRIX projects approached engineering development. Our focus switched to providing a common set of goals and one integrated schedule. Priorities for individual projects were reevaluated in the context of the system goals and schedule. It was critical to have a set of well-defined interfaces; any change to these interfaces could have a major system impact. Communication between all projects was essential. A significant amount of time was built into the schedule for system integration, as well as component integration.

Kernel Networking Environment

The DECnet/OSI for ULTRIX kernel implementation was designed to be consistent with other ULTRIX networking implementations such as the TCP/IP and Local Area Transport (LAT). The networking structure is based on the BSD networking subsystem.⁴

The ULTRIX networking environment allows protocol components to be insulated from each other. One important aspect of this networking system is the use of protocol switch tables. These tables contain the entry points for various protocol modules in the system, as shown in Figure 1. DECnet/OSI for ULTRIX uses these entry points to define interfaces for each protocol module. This means that there are no direct calls from one protocol component into another, an important consideration when new layers must be integrated. Moreover, one protocol module does not access another's databases. Information is accessed from a module only through the defined interface.

Insulating protocol modules from each other is advantageous for various reasons. As long as a protocol module supports a generic interface, it can act as a service provider for multiple users, which allows a system to support multiple configurations. For example, X.25 or high-level data link control (HDLC) may be configured into the kernel only

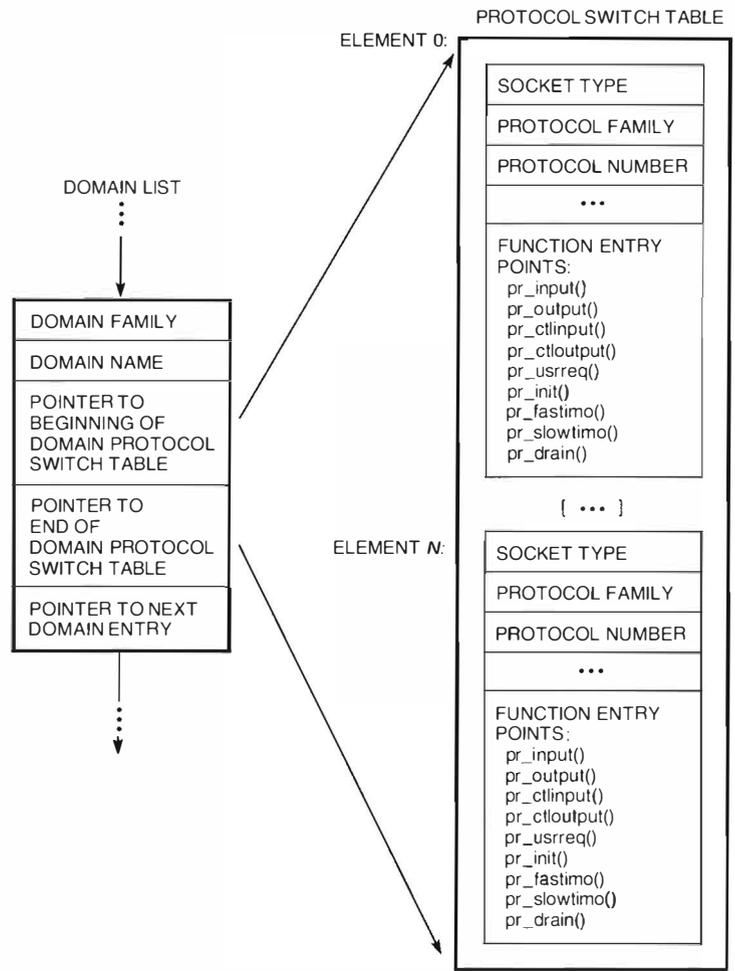


Figure 1 Domains and Protocol Switch Tables

when those services are needed. New protocol modules can be easily added. If token ring support is added as one of the broadcast devices, using the same interface as the carrier sense multiple access with collision detection (CSMA/CD) and fiber distributed data interface (FDDI) modules, little or no change will be required to the network layer.

Modularity is another advantage. Complexity can be reduced and problems can be isolated more easily when interfaces between each protocol module are carefully defined. For example, defining a network management interface for each protocol removes the requirement for network management to access protocol module databases directly. Network management code does not need to understand the internal organization of a module or the locking strategies that may be required to access the data.

To make use of the protocol switch table entry points, some minor enhancements were required. An extension was made to the control output interface to allow requests from kernel-level protocol modules and network management. The interface was further extended to allow protocol modules to use a port option to identify themselves as clients of the service provider, to acquire information from the service provider, or to modify the service provider's behavior. Network management uses a different option passed through the control output interface to manage kernel entities.

The control input interface was also enhanced. This interface provides two arguments: a request and a pointer to one or more arguments to be interpreted as a function of the request. Originally, this routine was used to notify IP of events, where each event had its own unique request value. To allow

DECnet/OSI protocols to use this interface without adding several new request values, a general-purpose request was introduced. This request is used by a service provider to interrupt one or more of its clients to inform them of a change in service. As part of the argument list, the service provider passes a value indicating the exact nature of the event being communicated. As an example, the network layer uses this mechanism to inform the transport layer modules of a change to the set of network addresses. Similarly, X.25 uses this interface to provide status about specific network connections.

The ULTRIX/BSD networking system organizes protocols into communication domains. The purpose of a communication domain is to group together common properties necessary for process-to-process communication. As an example, the X.25 domain was designed to provide a full set of X.25 services that can be selected by client protocols. It includes the socket and protocol switch table interfaces necessary for user-level and kernel-level clients, X.25 accounting, profile loading, and trace utilities.

The components of DECnet/OSI for ULTRIX may be combined in different ways depending on the configuration requirements of individual customers. A multiple domain approach was chosen to allow the various products and their development to be separated from one another. For example, network management software was placed in a separate domain to allow the X.25 and wide area network device driver (WANDD) products to be managed without installing DECnet/OSI for ULTRIX. Similarly, the OSI domain protocols may operate without the X.25 or WANDD products configured into the system.

Five domains were established:

1. The DECnet domain (AF_DECnet) is retained to provide backward compatibility to existing DECnet-ULTRIX Phase IV applications.
2. The data link domain (AF_DLI) contains all the data link protocols, including Logical Link Control (ISO 8802-2), CSMA/CD, FDDI, and HDLC. For DECnet/OSI for ULTRIX, the AF_DLI domain provides access to the drivers for kernel modules as well as user applications.
3. The X.25 domain (AF_X25) contains the protocols necessary to access X.25 networks.
4. The OSI domain (AF_OSI) contains the higher-level DECnet/OSI protocols, i.e., DNA session

control, network services protocol (NSP), OSI transport, DNA Phase V routing.

5. The network management domain (AF_NETMAN) contains all the network management functions. These functions can be used to manage any DNA networking product.

Data Link Domain

Under DECnet-ULTRIX Phase IV, the routing protocol module accessed the drivers directly. In the OSI implementation, data link interface (DLI) modules interface to the device drivers and act as service providers to network layer clients such as routing. This decision was made to minimize specific DECnet/OSI support needed in the ULTRIX operating system device drivers. This allows changes to be made more easily, and it provides a central location for common data link protocol code as well as network management code.

The AF_DLI domain provides a common interface to broadcast data links such as CSMA/CD and FDDI. Modules implementing new broadcast data link technologies can be added at any time by conforming to the DLI interface. DLI provides support for ISO 802.2 class I, type 1 functions; these may be used by any broadcast module. Other 802.2 classes are handled by passing frames directly to the client module.

The point-to-point protocols consist of HDLC and the Digital data communications message protocol (DDCMP). ULTRIX relies on the DDCMP support provided by hardware devices. However, a DDCMP software module exists to interface these devices to network management. HDLC, on the other hand, is entirely implemented as a software module operating over a device driver. Similar interfaces are provided by each protocol.

X.25 Domain

To ensure consistency with the goals and requirements of DECnet/OSI for ULTRIX, several design alternatives were considered for integrating X.25 into ULTRIX, including porting a previous Digital implementation of X.25, the VAX Packet Switch Interconnect. These alternatives were rejected because they were not consistent with the DECnet/OSI for ULTRIX implementation and BSD networking in general. A new version of X.25 was implemented in the C language using the protocol switch table infrastructure. This approach provided enough flexibility to allow the ULTRIX X.25 code to be easily ported to other product environments such as the WANrouter 250.

The X.25 components of DECnet/OSI for ULTRIX are provided as part of a wider X.25 strategy that can support multiple protocol suites, such as DECnet/OSI, TCP/IP, and International Business Machine Corporation's Systems Network Architecture (SNA). Under DECnet/OSI for ULTRIX, X.25 is used in two configurations. It provides the connection oriented network services (CONS) support to the OSI transport layer (ISO 8208, ISO 8878), and it can be used as a subnetwork for the connectionless network service (CLNS) layer. When used with TCP/IP networks, X.25 can be used as a subnetwork for the IP (Request for Comment [RFC] 877).

The interface to X.25 services was designed to be accessed by other kernel components. The protocol switch table was used to implement this interface. Components such as OSI connectionless network protocol and OSI transport make direct use of the kernel protocol switch interface with no intervening software layer.

Access by user-level applications to X.25 occurs through the BSD socket interface. The processing requirements of the socket layer and the kernel layer provided by the protocol switch are considerably different. To reduce the complexity of the kernel interface, an X.25 socket converter module was provided. The socket converter module manages issues such as queuing data at the socket interface and converting between protocol switch table routines and socket-layer calls. The converter module is treated as a client of the kernel interface.

Direct access to the X.25 kernel interface from IP was not possible due to TCP/IP development constraints. Instead, an IP device converter was supplied with ULTRIX X.25. This X.25-IP interface module appears as a device driver to IP. Furthermore, IP can be configured to use X.25 without requiring changes to the TCP/IP software. The pseudo-driver establishes an X.25 call when data is sent to the X.25 device. After the IP data has been transmitted, the X.25 connection is maintained to reduce the overhead and cost of X.25 call setup when the next IP data packet is sent. Configuration of the X.25 IP device is performed using standard ifconfig management commands.

OSI Domain

The AF_OSI domain contains the routing module, the transport modules, and DNA session control. The routing module is an end system implementation that adheres to the *Digital Network Architecture (Phase V) Network Routing Layer*

Functional Specification, version 3.0.0. It provides support for the ISO Connectionless Network Service (ISO 8473), End System to Intermediate System Routing Exchange Protocol (ISO 9542), and Phase IV routing. "Ping," a network loopback function specified in *Amendment X: Addition of an Echo Function to ISO 8473* and in RFC 1139, is provided as a diagnostic tool to test network access to a node.

Routing can be configured to operate over the data link entities previously mentioned as well as X.25. As an end system, DECnet/OSI for ULTRIX does not route protocol data units (PDUs). It can, however, operate over multiple circuits simultaneously, which allows load balancing across circuits and network redundancy. Phase V routing is capable of autoconfiguring to one or more network addresses.⁵

OSI transport (ISO 8072, ISO 8073) and NSP are the two transport modules supported. Both can be configured to operate over CLNS. However, only OSI transport can be configured to operate over CONS/X.25. OSI transport class 4 is supported over CLNS, and classes 0, 2, and 4 are supported over CONS/X.25. OSI transport also provides a connectionless transport service (CLTS) to its users. CLTS is a datagram service that operates over CLNS.

OSI transport supports two client interfaces and NSP supports one. Both support an interface to DNA session control supplied by the protocol switch table entry points. OSI user applications directly access OSI transport through X/Open transport interface (XTI).⁶ XTI specifies a transport service interface that is independent of the transport provider. On the ULTRIX implementation, XTI is a library interface implemented using the socket layer. It is discussed in more detail later in the section Application Programming Interfaces.

OSI transport can have multiple clients, and it identifies each client by an address called the transport selector. When OSI transport processes an incoming connect request, it uses the selector to determine which client should receive notification of the request.

The DNA session control protocol engine was implemented as part of NSP for the DECnet-ULTRIX Phase IV release. It is now implemented as a separate entity to allow operation over multiple transports (NSP and OSI transport). This modification created a subtle problem. DNA session control resides between the transport layers and the socket layer. However, both transport modules and DNA

session control need access to the socket. DNA session control needs access when performing connection control, and the transport modules need access when appending transmit or receive buffers to the socket queues. Since the socket is actually open to DNA session control, a mechanism was created to relay the socket pointer to the transport modules. This information is passed through the control output interface as part of the port option.

Application Programming Interfaces

To ease the transition of applications from Phase IV to DECnet/OSI, the Phase IV socket interface and programming library were retained. Applications using these interfaces will continue to work. This allows programmers time to modify their applications to use the new interfaces and the capabilities provided with DECnet/OSI for ULTRIX.

New application programming interfaces (APIs) were developed. These APIs include a DNA Phase V session control programming library, an X.25 programming library, an X.25 socket interface, and an XTI interface. They allow programmers to write network applications that use DECnet/OSI capabilities.

DNA Session Control Library

Through the use of the DNA Phase V session control library and DECdns, applications can provide location-independent services to the network. DNA session control stores information about an application and its services in an object in the DECdns namespace. Client applications can access these services by referencing the object name without knowing the current location of the service.

DNA Phase V session control applications also have the option of operating over various transport services and network services. The library gives the application programmer the flexibility of specifying the particular combination of services to be used. As an alternative, the library can determine the possible combinations of protocols that are supported on both the local and remote systems. This is done by accessing the addressing information stored in DECdns for each of these systems. If any combinations of protocols exist, DNA session control tries each of them in succession until a connection is established.

The DNA Phase V session control programming library is designed to be extensible. Instead of using a calling sequence with numerous parameters, one parameter is passed on all calls. This parameter is

an extensible data structure that consists of both input and output arguments. It allows new arguments to be added by appending fields to the end of the data structure.

The library is designed to support multithreaded application development. If a threads programming interface is supported on the ULTRIX operating system, programmers are able to write applications that have multiple control paths executing in parallel. This is useful in writing a network server application that frequently needs to handle requests from multiple clients. A single server application can process requests in parallel instead of creating additional processes to service each request. Multithreaded support in the library was accomplished by removing the use of static and global data by the library. Information is returned in dynamically allocated memory, which the applications are responsible for freeing.

X.25 Interfaces

Two programming interfaces are provided for the X.25 component. A socket interface is provided for full access to X.25 features in a manner compatible with BSD UNIX. This allows applications to make use of a direct socket interface to both TCP/IP and X.25.

An X.25 programming library was created to provide a portable programming interface that could be used for access to X.25 across current and future Digital implementations. The format of calls to the X.25 library was constructed on lines more compatible with the interface defined in the DNA X.25 access architecture than that available through the socket interface.

XTI Library

The XTI library has been extended to provide a framework for developing OSI applications. XTI provides a transport-independent programming interface that is standard across UNIX operating systems. On ULTRIX, XTI was implemented to provide a portable interface for writing TCP/IP applications. In DECnet/OSI for ULTRIX, the implementation was extended to provide support for OSI transport, including both connection oriented transport service (COTS) and CLTS. In addition to supporting the standard XTI calls, service routines were implemented. These routines provide a mechanism to build and access addressing information needed within XTI. The addressing information consists of transport selectors, network addresses, and internet ports.

Support for the Internet RFC 1006 specification was also added to the XTI library.⁷ This specification allows OSI applications to run over the TCP/IP protocol suite. RFC 1006 defines a mechanism for OSI transport class 0 (TP0) messages to be mapped across a TCP connection. OSI applications can be written to communicate over either TCP/IP networks or OSI networks, using the same API.

An RFC 1006 daemon was implemented to work in conjunction with the XTI library to handle incoming connection establishment. To allow multiple OSI applications to bind to the same RFC 1006 TCP port, a simple protocol exchanges file descriptors and a few basic messages between the XTI library and the daemon, using UNIX domain sockets. RFC 1006 specifies that a TCP connection be completed and a TP0 connect request be received before an OSI application server can be selected to process the incoming connect. The daemon hides the TCP connection and effectively blocks the OSI application server until the TP0 connect request occurs.

Network Management

DECnet/OSI network management is completely different from the management provided for DECnet Phase IV. It is based on the Enterprise Management Architecture (EMA), which provides a framework to consistently manage the various components making up a distributed system.⁸ DECnet/OSI for ULTRIX network management consists of a director, an event logger, an agent access module, and an agent for each manageable protocol entity. Figure 2 shows the network management environment.

The director, network control language (NCL), provides the user interface that allows network management commands to be entered. NCL

encodes the network management commands using the common management information protocol (CMIP). The encoded directives are passed to the common management listener (CML). CML, in turn, passes the directives to the appropriate agent in a form the agent can understand. On the ULTRIX implementation, when the connection between NCL and CML is local, a pipe is used. When NCL needs to connect to a remote CML, an OSI network connection is established.

The event logger (EVL) takes event messages generated by agents and sends them to either a local sink or a remote event sink. A local sink is a process that is executing locally, but a remote event sink is executing on a system elsewhere in the network. In the latter case, the CMIP protocol is used to convey the event message. Events are typically displayed on the console or in a file.

The DECnet/OSI for ULTRIX network management implementation is designed to be modular and extensible. The data dictionary, a key component, describes all the management attributes of each entity. The data dictionary is a dynamically extensible database and is used by all network management applications. NCL uses the data dictionary to parse command lines and display output. CML uses the data dictionary to decode/encode CMIP protocol messages from/to NCL, and EVL uses it to display an event locally. Information about new attributes or entire entities can be added to the data dictionary without modifying the network management applications. Thus layered products can easily add support for new manageable objects.

The network management environment in DECnet/OSI for ULTRIX is essentially a message passing scheme, as shown in Figure 2. Like the data dictionary, it was designed to be extensible and

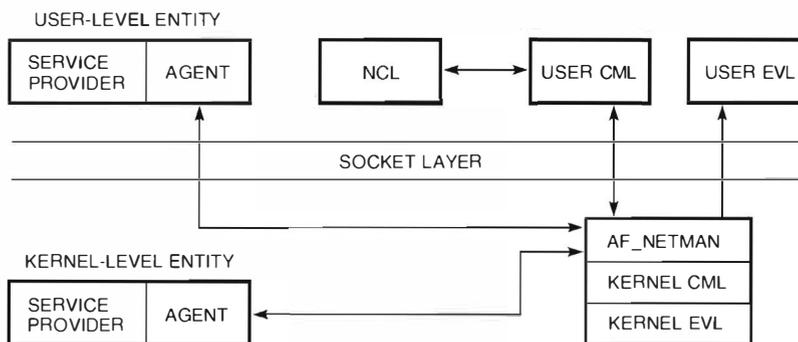


Figure 2 Network Management

generic. All manageable, DNA-architected entities use this environment. At the core is a switch, kernel CML. Kernel CML passes messages between user CML and any DNA entity. User CML and kernel CML communicate through the socket layer. User-level agents, in turn, communicate with CML using the socket-layer interface, and kernel-level agents communicate with CML through the control output routine for the entity.

User-level agents can send multiple responses to a single request, but kernel-level agents can send only one response per request. Because user-level agents reside in process space and are separated by the socket layer, their transactions can be asynchronous. Transactions of kernel-level agents, on the other hand, must be synchronous. When called, they must process the request and return a single response. Whenever multiple responses are to be returned, as in a wild-card operation, the agent relies on being invoked again by kernel CML for each of the response messages. This programming precludes the possibility of exhausting system buffers while conveying information about a large number of subentities. Kernel CML stops requesting additional responses from a kernel entity when it detects that the socket receive queue is full. Once there is more room on the queue, it resumes the wild-card operation.

The network management environment provides a core set of routines as an aid to processing and building the syntax for each message. It also provides routines that assist in wild-card processing. Agents that make use of these routines need not be aware of the physical structure of each message. This has several benefits. It provides a common set of code that is not duplicated from entity to entity. If there is a problem, it is corrected in one location instead of several. Also, it makes the implementation more portable. The message passing scheme uses the local operating system's network buffers. When changing from one operating system to another, the buffering needs to change only in the common code and not in each of the agents.

Entities may need to originate event messages bound for the EVL. The mechanism providing this support is basically the same as the message passing scheme previously described. A kernel EVL switch receives event messages from either a user-level or kernel-level agent and passes the event up to its counterpart through the socket layer. With this mechanism, however, messages flow in only one direction, from the entity to the event logger.

In DECnet/OSI, some significant architectural changes were made to the maintenance operations protocol (MOP). As in Phase IV, the current implementation supports down-line loading and up-line dumping over FDDI and CSMA/CD devices. These functions are now performed by using the MOP version 4.0 protocol over ISO 8802-2 or MOP version 3.0 over Ethernet. As part of implementing the new protocol, support for down-line loading CMIP scripts was added. These are used by remote systems such as DECnet/OSI routers to perform network management initialization. Client information is kept in a MOP-specific database. By keeping entity-specific information modular and distinct, the DECnet/OSI for ULTRIX MOP implementation is consistent with EMA. This contrasts with the DECnet-ULTRIX Phase IV implementation, which stores MOP client information in the DECnet nodes database.

Applications Supported

The DECnet Phase IV applications continue to be provided with the DECnet/OSI for ULTRIX product. These include the file transfer utility, dcp, the remote terminal utility, dlogin, and the mail utility. These DECnet applications have been modified to use the DECnet/OSI for ULTRIX programming interface and to take advantage of the new DNA Phase V capabilities. They can accept DECdns full names for node names and run over both the NSP and OSI transport. The DECnet-internet gateway is also provided as part of the product. The gateway provides bidirectional network access between DECnet and internet systems. It allows DECnet and TCP/IP users to communicate through their respective file transfer, remote login, and mail facilities.

New OSI applications were written to provide similar capabilities to the DECnet applications. They allow users to access files and terminal emulation in a multivendor environment. These OSI applications include FTAM, VTP, and X.29 terminal support. Just as the DECnet-internet gateway is provided, OSI applications provide their own gateways to link OSI and internet.⁹

ULTRIX X.25 includes X.29 terminal support. A packet assembler/disassembler (PAD) provides outgoing access. Thus PAD allows terminal emulation for X.25 connections to remote hosts in much the same way that the VTP does in a full OSI stack. For incoming X.29 calls, a UNIX daemon creates an X.29 login process or activates an application based on X.29.

Installation and Configuration

DECnet/OSI for ULTRIX networking software allows the use of OSI addressing and access to global naming services. It provides new network management utilities and the ability to configure a network stack in many different ways. For example, in configuring X.25, many attributes can be set to allow conformance to many public and private packet-switched data networks. The new capabilities add a degree of complexity to the process of configuring the networking software. To simplify this process, configuration was separated from installation. Installation occurs when files are moved from the distribution media to the target system. Configuration is the process of providing information to make the networking subsystem operational.

The ULTRIX DECnet/OSI and X.25 setup utilities provide two modes of configuration, basic and advanced. The DECnet/OSI for ULTRIX setup basic configuration process asks a limited number of questions and is designed for the user who will be installing DECnet/OSI for ULTRIX on a workstation connected to a local area network. The advanced configuration process and X.25 setup utility provide more configuration choices for the network manager who will be installing DECnet/OSI for ULTRIX in a server configuration, or who will require more detailed network configurations.

X.25 and wide area network device driver setup utilities supply a mechanism for configuring TCP/IP or DECnet/OSI for ULTRIX to run over X.25 or synchronous data links. For a more unified approach to configuring an OSI stack, these setup utilities are integrated with the DECnet/OSI for ULTRIX setup advanced process. These setup utilities add a logical abstraction above the EMA, which helps to reduce complexity. For each manageable entity on the system, NCL scripts are generated through default assumptions and responses to configuration questions.

Network configuration is accomplished with shell scripts and network management scripts. These mechanisms initialize manageable entities. At system start-up, the decnetstartup script is executed from within rc.local. This invokes the various NCL scripts to configure the networking software. One or more NCL scripts can be modified independently of the configuration utilities to change attributes of the manageable entities. As an alternative, the setup utilities can be rerun to modify the scripts. In addition, responses to configuration

questions are stored in a file to provide default answers to simplify subsequent reconfiguration.

Summary

The design of DECnet/OSI for ULTRIX was a challenging endeavor that resulted in a rich set of capabilities and a system on which to build new functions. It operates in a complex networking environment that includes OSI, DECnet Phase IV, X.25, and TCP/IP protocols. DECnet/OSI for ULTRIX software allows OSI applications to function in TCP/IP networks. RFC 1006 supports the operation of OSI applications using TCP/IP connections, and RFC 877 allows TCP/IP to be configured over X.25. In addition, a set of gateways allows intercommunication between DECnet/OSI and TCP/IP networks.

The DECnet/OSI for ULTRIX system was also designed to be extended to include new functions as they are incorporated into the OSI standards. New protocol components can be added and used without changing existing components or network management. In addition, the software was designed to be portable. The DECnet/OSI for ULTRIX software has been ported to the DEC OSF/1 AXP operating system, and DECnet/OSI version 1.0 for DEC OSF/1 AXP was released in March 1993.

DECnet/OSI for ULTRIX demonstrates Digital's continuing commitment to provide the OSI protocol on platforms based on open systems. The ULTRIX system was the first end system to include products that followed the DNA OSI strategy. These systems can interoperate with either DECnet Phase IV systems or other OSI systems. As with DECnet Phase IV, DECnet/OSI for ULTRIX continues to provide a set of components consistent with the UNIX philosophy of networking.

Acknowledgments

The authors would like to thank the people, past and present, who contributed to the design and development of the DECnet/OSI for ULTRIX product. Special thanks go to members of the following teams for their dedication and hard work: DECnet-ULTRIX, ULTRIX FTAM, ULTRIX VT, OSAK, DECdns, DECdts, ULTRIX X.25, and ULTRIX Wide Area Device Drivers.

References

1. J. Harper, "Overview of Digital's Open Networking," *Digital Technical Journal*, vol. 5, no. 1 (Winter 1993, this issue): 12-20.

2. *Information Processing Systems—Open Systems Interconnection—Basic Reference Model*, ISO 7498 (New York: American National Standards Institute, 1984).
3. S. Martin, J. McCann, and D. Oran, "Development of the VAX Distributed Name Service," *Digital Technical Journal*, vol. 1, no. 9 (June 1989): 9-15.
4. S. Leffler, W. Joy, and R. Fabry, "4.2BSD Networking Implementation Notes," (Berkeley, CA: University of California Technical Report, 1983).
5. R. Perlman, R. Callon, and M. Shand, "Routing Architecture," *Digital Technical Journal*, vol. 5, no. 1 (Winter 1993, this issue): 62-69.
6. X/Open Company, Ltd., *X/Open Portability Guide, Networking Services* (Englewood Cliffs, NJ: Prentice-Hall, 1988).
7. M. Rose and D. Cass, "Request for Comments: RFC 1006, ISO Transport Services on Top of the TCP, Version 3," May 1987.
8. M. Saylor, F. Dolan, and D. Shurtleff, "Network Management," *Digital Technical Journal*, vol. 5, no. 1 (Winter 1993, this issue): 117-129.
9. D. Robinson, L. Friedman, and S. Wattum, "An Implementation of the OSI Upper Layers and Applications," *Digital Technical Journal*, vol. 5, no. 1 (Winter 1993, this issue): 107-116.

General References

D. Comer, *Internetworking with TCP/IP: Principles, Protocols and Architecture* (Englewood Cliffs, NJ: Prentice-Hall, 1988).

S. Leffler, M. McKusick, M. Karels, and J. Quarterman, *The Design and Implementation of the 4.3 BSD UNIX Operating System* (Reading, MA: Addison-Wesley Publishing Company, May 1989).

S. Leffler, W. Joy, and R. Fabry, "A 4.2BSD Interprocess Communication Primer," (Berkeley, CA: University of California Technical Report, 1983).

*Chran-Ham Chang
Richard Flower
John Forecast
Heather Gray
William R. Howe
K. K. Ramakrishnan
Ashok P. Nadkarni
Uttam N. Shikarpur
Kathleen M. Wilde*

High-performance TCP/IP and UDP/IP Networking in DEC OSF/1 for Alpha AXP

The combination of the Alpha AXP workstations, the DEC FDDIcontroller/TURBOchannel network interface, the DEC OSF/1 operating system, and a streamlined implementation of the TCP/IP and UDP/IP delivers to user applications almost the full FDDI bandwidth of 100 Mb/s. This combination eliminates the network I/O bottleneck for distributed systems. The TCP/IP implementation includes extensions to TCP such as support for large transport windows for higher performance. This is particularly desirable for higher-speed networks and/or large delay networks. The DEC FDDIcontroller/TURBOchannel network interface delivers full bandwidth to the system using DMA, and it supports the patented point-to-point, full-duplex FDDI mode. Measurement results show UDP performance is comparable to TCP. Unlike typical BSD-derived systems, the UDP receive throughput to user applications is also maintained at high load.

We have seen significant increases in the bandwidth available for computer communication networks in the recent past. Commercially available local area networks (LANs) operate at 100 megabits per second (Mb/s), and research networks are running at greater than 1 gigabit per second (Gb/s). Processor speeds have also seen dramatic increases at the same time. The ultimate throughput delivered to the user application, however, has not increased as rapidly. This has led researchers to say that network I/O at the end system is the next bottleneck.¹

One reason that network I/O to the application has not scaled up as rapidly as communication link bandwidth or CPU processing speeds is that memory bandwidth has not scaled up as rapidly even though memory costs have fallen dramatically. Network I/O involves operations that are memory intensive due to data movement and error checking. Scaling up memory bandwidth, by making memory either wider or faster, is expensive. The result has been an increased focus on the design and implementation of higher-performance network interfaces, the re-examination of the implementation of network I/O, and the considera-

tion of alternative network protocols to achieve higher performance.^{2,3,4}

This paper describes the work we did to remove the end system network I/O bottleneck for current commercially available high-speed data links, such as the fiber distributed data interface (FDDI).^{5,6} We used the conventional internet protocol suite of transmission control protocol/internet protocol (TCP/IP) and the user datagram protocol/internet protocol (UDP/IP) on Alpha AXP hardware and software platforms.^{7,8,9} The specific hardware platform was the DEC 3000 AXP Model 500 workstation with the DEC FDDIcontroller/TURBOchannel adapter (DEFTA). The software platform was the DEC OSF/1 operating system version 1.2 using the TCP and UDP transport protocols. The combination of the Alpha AXP workstations, the DEFTA adapter, the DEC OSF/1 operating system, and a streamlined implementation of the TCP/IP and UDP/IP delivers to user applications essentially the full FDDI bandwidth of 100 Mb/s.

While the DEC FDDIcontroller/TURBOchannel network interface is lower cost than previous FDDI controllers, it also delivers full bandwidth to the system using direct memory access (DMA). In

addition, it supports the patented point-to-point, full-duplex FDDI mode. This allows a link to be used with 100 Mb/s in each direction simultaneously, which increases throughput in some cases and reduces latency compared to the standard FDDI ring mode.

Incremental work for data movement and checksums has been optimized to take advantage of the Alpha AXP workstation architecture, including 64-bit support, wider cache lines, and the coherence of cache blocks with DMA. Included in the TCP/IP implementation are extensions to TCP recently recommended by the Internet Engineering Task Force (IETF), such as support for large transport windows for higher performance.¹⁰ This is particularly desirable for high-speed networks and/or large delay networks.

We feel that good overload behavior is also important. Workstations as well as hosts acting as servers see substantial load due to network I/O. Typical implementations of UDP/IP in systems based on the UNIX operating system are prone to degradation in throughput delivered to the application as the received load of traffic to the system increases beyond its capacity. Even when transmitting UDP/IP packets from a peer transmitter with similar capabilities, the receiver experiences considerable packet loss. In some cases, systems reach receive "livelock," a situation in which a station is only involved in processing interrupts for received packets or only partially processing received packets without making forward progress in delivering packets to the user application.¹¹ Changes to the implementation of UDP/IP and algorithms incorporated in the DEFTA device driver remove this type of congestion loss at the end system under heavy receive load. These changes also eliminate unfairness in allocation of processing resources, which results in starvation (e.g., starving the transmit path of resources).

The next section of this paper discusses the characteristics of the Alpha AXP workstations, the DEC OSF/1 operating system, and the two primary transport protocols in the internet protocol suite, TCP and UDP. We provide an overview of the implementation of network I/O in a typical UNIX system using the Berkeley Software Distribution (BSD) to motivate several of the implementation enhancements described in the paper.¹²

The section on Performance Enhancements and Measurements Results then describes the specific implementation enhancements incorporated in

the DEC OSF/1 operating system version 1.2 to improve the performance of TCP and UDP. This section also provides measurement results for TCP and UDP with DEC 3000 AXP workstations running DEC OSF/1 version 1.2 in a few different configurations. Also included are measurements with TCP and UDP with Digital's patented full-duplex mode for FDDI, which can potentially increase throughput and reduce latency in FDDI LANs with point-to-point links (which can also be used in switched FDDI LANs). A few implementation ideas currently under study are also presented in the section on Experimental Work.

System Characteristics

The project to improve the implementation of Digital's TCP/IP and UDP/IP (the internet protocol suite) networking was targeted on the DEC 3000 AXP Model 500 workstation, running the DEC OSF/1 operating system version 1.2. Since we were interested in achieving the highest performance possible on a commercially available data link, we chose FDDI, and used the DEC FDDIcontroller/TURBOchannel adapter (DEFTA) to communicate between the Alpha AXP workstations. In this section, we describe the features of the workstations, relevant characteristics of FDDI, the internet protocol suite, and the DEC OSF/1 operating system itself, relative to the networking implementation. The architectural features of the Alpha AXP workstations as well as the DEC FDDIcontroller/TURBOchannel adapter are shown in Figure 1.

The Alpha AXP System

The Alpha AXP workstation, DEC 3000 AXP Model 500 was chosen for our research. The system is built around Digital's 21064 64-bit, reduced instruction set computer (RISC) microprocessor.

Digital's 21064 Microprocessor The DECchip 21064 CPU chip is a RISC microprocessor that is fully pipelined and capable of issuing two instructions per clock cycle.^{13,14} The DECchip 21064 microprocessor can execute up to 400 million operations per second. The chip includes

- An 8-kb direct-mapped instruction cache with a 32-byte line size
- An 8-kb direct-mapped data cache with a 32-byte line size
- Two associated translation buffers
- A four-entry (32-byte-per-entry) write buffer

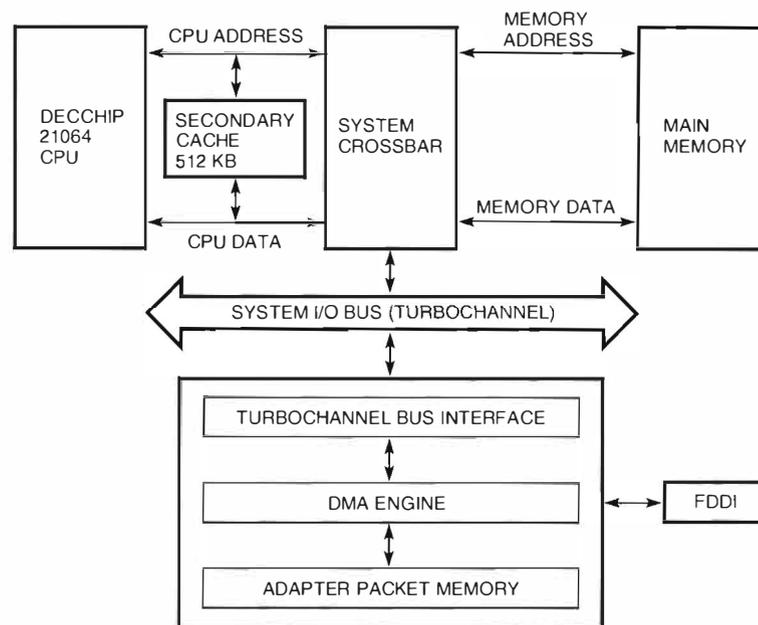


Figure 1 The Alpha AXP Workstation—CPU, Memory Subsystem, and the FDDI controller/TURBOchannel Adapter

- A pipelined 64-bit integer execution unit with a 32-entry register file
- A pipelined floating-point unit with an additional 32 registers

The DEC 3000 AXP Model 500 Workstation The DEC 3000 AXP Model 500 workstation is built around the DECchip 21064 microprocessor running at 150 megahertz (MHz).¹⁵ In addition to the on-chip caches, there is an on-board second-level cache of 512 kilobytes (kB). Main memory can be from 32 MB to 256 MB (1 GB with 16 MB dynamic random-access memories [DRAMs]). The memory bus is 256 bits plus error-correcting code (ECC) wide and has a bandwidth of 114 MB/s. Standard on the system is also a 10-Mb/s Ethernet interface (LANCE). For connection to external peripherals there is an on-board small computer systems interface (SCSI)-2 interface and six TURBOchannel slots with a maximum I/O throughput of 100 MB/s. One of the TURBOchannel slots is occupied by the graphics adapter.

The system uses the second-level cache to help minimize the performance penalty of misses and write throughs in the two relatively smaller primary caches in the DECchip 21064 processor. The second-level cache is a direct-mapped, write-back cache with a block size of 32 bytes, chosen to match the block size of the primary caches. The cache

block allocation policy allocates on both read misses and write misses. Hardware keeps the cache coherent on DMAs; DMA reads probe the second-level cache, and DMA writes update the second-level cache, while invalidating the primary data cache. More details of the DEC 3000 AXP Model 500 AXP workstation may be obtained from "The Design of the DEC 3000 AXP Systems, Two High-performance Workstations."¹⁵

DEC OSF/1 Operating System

DEC OSF/1 operating system version 1.2 for Alpha AXP systems is an implementation of the Open Software Foundation (OSF) OSF/1 version 1.0 and version 1.1 technology. The operating system is a 64-bit kernel architecture based on Carnegie-Mellon University's Mach version 2.5 kernel. Components from 4.3 BSD are included, in addition to UNIX System Laboratories System V interface compatibility.

Digital's version of OSF/1 offers both reliability and high performance. The standard TCP/IP and UDP/IP networking software, interfaces, and protocols remain the same to ensure full multivendor interoperability. The software has been tuned and new enhancements have been added that improve performance. The interfaces between the user application and the internet protocols include both

the BSD socket interface and the X/Open Transport Interface.¹² The internet implementation conditionally conforms to RFC 1122 and RFC 1123.^{16,17} Some of the networking utilities included are Telnet; file transfer protocol (FTP); the Berkeley "r" utilities (rlogin, rcp, etc.); serial line internet protocol (SLIP) with optional compression; Local Area Transport (LAT); screend, which is a filter for controlling network access to systems when DEC OSF/1 is used as a gateway; and prestoserve, a file system accelerator that uses nonvolatile RAM to improve Network File System (NFS) server response time. The implementation also provides a STREAMS interface, the transport layer interface, and allows for STREAMS (SVID2) and sockets to coexist at the data link layer. There is support for STREAMS drivers to socket protocol stacks and support for BSD drivers to STREAMS protocol stacks via the data link provider interface.

The OSF/1 Network Protocol Implementation

The overall performance of network I/O of a workstation depends on a variety of components: the processor speed, the memory subsystem, the host bus characteristics, the network interface and finally, and probably the most important, software structuring of the network I/O functions. To understand the ways in which each of these aspects influences performance, it is helpful to understand the structuring of the software for network I/O and the characteristics of the computer system (processor, memory, system bus). We focus here on the structuring of the end system networking code related to the internet protocol suite in the DEC OSF/1 operating system, following the design of the networking code (4.3 BSD-Reno) in the Berkeley UNIX distribution.^{8,9,12}

A user process typically interfaces to the network through the socket layer. The protocol modules for UDP, TCP (transport layers) and IP (network layer) are below the socket layer in the kernel of the operating system. Data is passed between user processes and the protocol modules through socket buffers. On message transmission, the data is typically moved by the host processor from user space to kernel memory for the protocol layers to packetize and deliver to the data link device driver for transmission. The boundary crossing from user to kernel memory space is usually needed in a general-purpose operating system for protection purposes. Figure 2 shows where the incremental overhead for

packet processing, based on packet size, occurs in a typical BSD 4.3 distribution.

The kernel memory is organized as buffers of various types. These are called mbufs. They are the primary means for carrying data (and protocol headers) through the protocol layers. The protocol modules organize the data into a packet, compute its checksum, and pass the packet (which is a set of mbufs chained together by pointers) to the data link driver for transmission. From these kernel mbufs, the data has to be moved to the buffers on the adapter across the system bus. Once the adapter has a copy of the header and data, it may return an indication of transmit completion to the host. This allows the device driver to release the kernel mbufs to be reused by the higher layers for transmitting or for receiving packets (if buffers are shared between transmit and receive).

While receiving packets, the adapter moves the received data into the host's kernel mbufs using DMA. The adapter then interrupts the host processor, indicating the reception of the packet. The data link driver then executes a filter function to enable posting the packet to the appropriate protocol processing queue. The data remains in the same kernel mbufs during protocol processing. Buffer pointers are manipulated to pass references to the data between the elements processing each of the protocol layers. Finally, on identifying the user process of the received message, the data is moved from the kernel mbufs to the user's address space.

Another important incremental operation performed in the host is that of computing the checksum of the data on receive or transmit. Every byte of the packet data has to be examined by the processor for errors, adding overhead in both CPU processing and memory bandwidth. One desirable characteristic of doing the checksum after the data is in memory is that it provides end-to-end protection for the data between the two communicating end systems. Because data movement and checksum operations are frequently performed and exercise components of the system architecture (memory) that are difficult to speed up significantly, we looked at these in detail as candidates for optimization.

The Internet Protocol Suite: TCP/IP and UDP/IP

The protocols targeted for our efforts were TCP/IP and UDP/IP, part of what is conventionally known as the internet protocol suite.^{7,9}

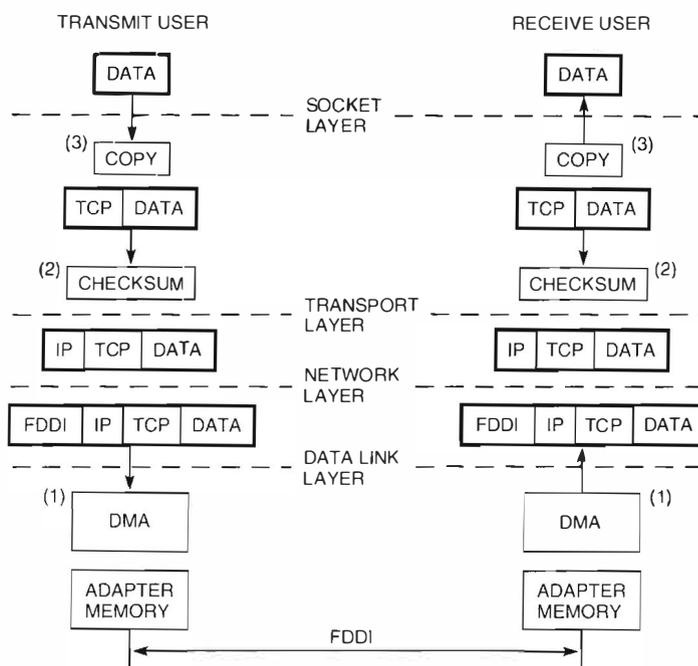


Figure 2 The incremental data operations occur in three places: (1) when the data is moved using DMA between the kernel and the network adapter memory, (2) when a checksum is computed for the data, and (3) when the data is copied between the user process and the kernel.

TCP is a reliable, connection-oriented, end-to-end transport protocol that provides flow-controlled data transfer. A TCP connection contains a sequenced stream of data octets exchanged between two peers. TCP achieves reliability through positive acknowledgment and retransmission. It achieves flow control and promotes efficient movement of data through a sliding window scheme. The sliding window scheme allows the transmission of multiple packets while awaiting the receipt of an acknowledgment. The number of bytes that can be transmitted prior to receiving an acknowledgment is constrained by the offered window on the TCP connection. The window indicates how much buffering the receiver has available for the TCP connection (the receiver exercises the flow control). This window size also reflects how much data a sender should be prepared to buffer if retransmission of data is required. The size of the offered window can vary over the life of a connection. As with BSD systems, DEC OSF/1 currently maintains a one-to-one correspondence between window size and buffer size allocated at the socket layer in the end systems for the TCP connection. An erroneous choice of window size, such as one too

small, or one leading to nonbalanced sender and receiver buffer sizes, can result in unnecessary blocking and subsequent inefficient use of available bandwidth.

TCP divides a stream of data into segments for transmission. The maximum segment size (MSS) is negotiated at the time of connection establishment. In the case of connections within the local network, TCP negotiates an MSS based on the maximum transmission unit (MTU) size of the underlying media. (For IP over FDDI the MTU is constrained to 4,352 octets based on the recommendation in RFC 1390.¹⁸) TCP calculates the MSS to offer, by subtracting from this MTU, the number of octets required for the most common IP and TCP header sizes.

The implementation of TCP/IP in DEC OSF/1 follows the 4.3 BSD-Reno implementation of TCP. Included is the use of dynamic round-trip time measurements by TCP, which maintains a timer per connection and uses adaptive time-outs for setting retransmission timers. The implementation includes slow start for reacting to congestive loss and optimizations such as header prediction and delayed acknowledgments important for network performance.¹⁹ DEC OSF/1 version 1.2 also includes

recent extensions to TCP for accommodating higher-speed networks.⁹ TCP's performance may depend upon the window size used by the two peer entities of the TCP connection. The product of the transfer rate (bandwidth) and the round-trip delay measures the window size that is needed to maximize throughput on a connection.

In the TCP specification RFC 793, the TCP header contains a 16-bit window size field which is the receive window size reported to the sender.⁹ Since the field is only 16 bits, the largest window size that is supported is 64K bytes. Enhancing the original specification, RFC 1323 defines a new TCP option, window scale, to allow for larger windows.¹⁰ This option contains a scale value that is used to increase the window size value found in the TCP header.

The window scale option is often recommended to improve throughput for networks with high bandwidth and/or large delays (networks with large bandwidth-delay products). However, it also can lead to higher throughput on LANs such as an FDDI token ring. Increased throughput was observed with window sizes larger than 64K bytes on an FDDI network.

The TCP window scale extension maps the 16-bit window size field to a 32-bit value. It then uses the TCP window scale option value to bit-shift this value, resulting in a new maximum receive window size value. The extension allows for windows of up to 1 gigabyte (GB). To facilitate backward compatibility with existing implementations, both peers must offer the window scale option to enable window scaling in either direction. Window scale is automatically turned on if the receive socket buffer size is greater than 64K bytes. A user program can set a larger socket buffer size via the `setsockopt()` system call. Based on the socket buffer size, the kernel implementation can determine the appropriate window scale factor.

Similar to the choice of large window sizes, the use of large TCP segments, i.e., those approaching the size of the negotiated MSS, could give better performance than smaller segments. For a given amount of data, fewer segments are needed (and therefore fewer packets). Hence the total cost of protocol processing overhead at the end system is less than with smaller segments.

The internet protocol suite also supports the user datagram protocol or UDP. UDP performance is important because it is the underlying protocol for network services such as the NFS. UDP is a connection-less, message-oriented transport layer

protocol that does not provide reliable delivery or flow control. The receive socket buffer size for UDP limits the amount of data that may be received and buffered before it is copied to the user's address space. Since there is no flow control, the UDP receiver may have to discard the packet if it receives a large burst of messages and there is no socket buffer space.

If the receiver is fast enough to allow the user application to consume the data, the loss rate is very low. However, most BSD-derived systems today experience heavy packet loss for UDP even when the receiving processor is the same speed as the transmitter. Furthermore, since UDP has no flow control, there is no mechanism to assure that all transmitted data will be received when the transmitter is faster than the receiver. We describe our implementation of UDP to avoid this behavior, so that packet loss is minimized.

Data Link Characteristics: FDDI

FDDI is a 100 Mb/s LAN standard that is being deployed commercially. It uses a timed-token access method and allows up to 500 stations to be connected with a total fiber length of 200 kilometers. It allows for both synchronous and asynchronous traffic simultaneously and provides a bound for the access time to the channel for both these classes of traffic.

The timed-token access method ensures that all stations on the ring agree to a target token rotation time (TTRT) and limit their transmissions to this target.²⁰ With asynchronous mode (the most widely used mode in the industry at present), a node can transmit only if the actual token rotation time (TRT) is less than the target.

The basic algorithm is that each station on the ring measures the time since it last received the token. The time interval between two successive receptions of the token is called the TRT. On a token arrival, if a station wants to transmit, it computes a token holding time (THT) as: $THT = TTRT - TRT$. The TTRT is agreed to by all the stations on the ring at the last time that the ring was initialized (typically happens when stations enter or leave the ring) and is the minimum of the requested values by the stations on the ring. If THT is positive, the station can transmit for this interval. At the end of transmission, the station releases the token. If a station does not use the entire THT allowed, other stations on the ring can use the remaining time by using the same algorithm.

A number of papers relating to FDDI have appeared in the literature, and the reader is encouraged to refer to "Performance Analysis of FDDI Token Ring Networks: Effect of Parameters and Guidelines for Setting TTRT," for more details.²¹

Network Adapter Characteristics

The DEC FDDIcontroller/TURBOchannel adapter, DEFTA, is designed to be a high-performance adapter capable of meeting the full FDDI bandwidth. It provides DMA capability both in the receive and transmit directions. It performs scatter-gather on transmit. The adapter has 1 MB of packet buffering. By default, half the memory is used for receive buffering; one-fourth of the memory is allocated for transmit buffering; and the remaining memory is allocated for miscellaneous functions, including buffering for FDDI's station management (SMT). The memory itself is not partitioned, and the adapter uses only as much memory as necessary for the packets. It avoids internal fragmentation and does not waste any memory.

The receive and transmit DMA operations are handled by state machines, and no processor is involved in data movement. The DMA engine is based on the model reported by Wenzel.²² The main concept of this model is that of circular queues addressed by producer and consumer indices. These indices are used by the driver and the adapter for synchronization between themselves; they indicate to each other the availability of buffers. For example, for receiving packets into the kernel memory, the device driver produces empty buffers. By writing the producer index, it indicates to the adapter the address of the last buffer produced and placed in the circular queue for receiving. The adapter consumes the empty buffer for receiving an incoming packet and updates the consumer index to indicate to the driver the last buffer that it has consumed in the circular queue. The adapter is capable of full-duplex FDDI operation. Finally, FDDI's SMT processing is performed by a processor on board the adapter, with the adapter's receive and transmit state machines maintaining separate queues for SMT requests and responses.

To obtain high performance, communication adapters also try to minimize the amount of overhead involved in transferring the data. To improve performance, the DEFTA FDDI port interface (interface between the hardware and the operating system's device driver) makes efficient use of host

memory data structures, minimizes overhead I/O related to the port interface, and minimizes interrupts to the host system.

The Port Architecture contains several unique features that optimize adapter/host system performance. These features include the elimination of much of the control and status information transferred between the host and adapter; the organization of data in host memory in such a way as to provide efficient access by the adapter and the host; and the use of an interrupt mechanism, which eliminates unnecessary interrupts to the host.

The design also optimizes performance through careful organization of data in host memory. Other than the data buffers, the only areas of host memory that are shared by the host and the adapter are the queues of buffer descriptors and the area in which the adapter writes the consumer indices. The adapter only reads the buffer descriptors; it never writes to this area of host memory. Thus the impact on host performance of the adapter writing to an area in memory, which may be in cache memory, is eliminated. On the other hand, the area in host memory where the adapter writes its consumer indices is only written by the adapter and only read by the host. Both the receive data consumer index and transmit data consumer index are written to the same longword in host memory, thus possibly eliminating an extra read by the host of information that is not in cache memory. Furthermore, the producer and consumer indices are maintained in different sections of memory (different cache lines) to avoid thrashing in the cache when the host and the adapter access these indices.

The device driver is also designed to achieve high performance. It avoids several of the problems associated with overload behavior observed in the past.²³ We describe some of these enhancements in the next section.

Performance Enhancements and Measurements Results

We describe in this section the various performance enhancements included in the DEC OSF/1 operating system version 1.2 for Alpha AXP systems. In particular, we describe the optimizations for data movement and checksum validation, the implementation details to provide good overload behavior within the device driver, the TCP enhancements for high bandwidth-delay product networks, and the UDP implementation enhancements.

We also present measurement results showing the effectiveness of the enhancements. In most cases the measurement environment consisted of two Alpha AXP workstations (DEC 3000 AXP Model 500) on a private FDDI token ring, with a DEC FDDI concentrator. The tests run were similar to the well-known `ttcp` test suite, with the primary change being the use of the slightly more efficient send and receive system calls instead of read and write system calls. We call this tool `inett` within Digital. The throughputs obtained were at the user application level, measured by sending at least 10,000 user messages of different sizes. With UDP, these are sent as distinct messages. With TCP, algorithms used by TCP may concatenate multiple messages into a single packet. Time was measured using the system clock with system calls for resource usage. We also monitored CPU utilization with these system calls, and made approximate (often only for relative comparison) conclusions on the usage of resources with a particular implementation alternative.

Optimizations for `bcopy()` and `in_checksum()` Routines

In TCP/UDP/IP protocol implementations, every byte of data generally must pass through the `bcopy()` and `in_checksum()` routines, when there is no assistance provided in the network interfaces. There are some exceptions: the NFS implementations on DEC OSF/1 avoid the `bcopy()` on transmit by passing a pointer to the buffer cache entry directly to the network device driver, and UDP may be configured not to compute a checksum on the data. Digital's implementations turn on the UDP checksum by default. Even with the above exceptions, it is important that the `bcopy()` and `in_checksum()` routines operate as efficiently as possible.

To write efficient Alpha AXP code for these routines, we used the following guidelines:

- Operate on data in the largest units possible
- Try to maintain concurrent operation of as many independent processor units (CPU, memory reads, write buffers) as possible
- Keep to a minimum the number of scoreboard-ing delays that arise because the data is not yet available from the memory subsystem
- Wherever possible, try to make use of the Alpha AXP chip's capability for dual issue of instructions

For network I/O, the `bcopy()` routine is called to transfer data between kernel mbuf data structures and user-supplied buffers to `read()/write()/send()/recv()` calls.

The `bcopy()` routine was written in assembler. This routine always attempts to transfer data in the largest units possible consistent with the alignment of the supplied buffers. For the optimal case, this would be one quadword (64 bits) at a time. The routine uses a simple load/store/decrement count loop that iterates across the data buffer as

```
ldq    t1, 0(a0) ;get next quadword
        ;(64 bits)
addq   a0, 8     ;move on source pointer
stq    t1, 0(a1) ;store quadword
addq   a1, 8     ;move on pointer
subq   t2, 8     ;reduce byte count
bne    t2, 1b   ;loop till done
```

Several attempts were made to improve the performance of this simple loop. One design involved unrolling the loop further to perform 64 bytes of copying at a time, while reading ahead on the second cache line. Another involved operating on four cache lines at once, based on concerns that a second quadword read of a cache line may incur the same number of clock delays as the first cache miss, if the second read is performed too soon after the first read. However, neither of these approaches produced a copy routine that was faster than the simple loop described above.

The TCP/UDP/IP suite defines a 16-bit one's complement checksum (`in_checksum()`), which can be performed by adding up each 16-bit element and adding in any carries. Messages must (optional for UDP) have the checksum validated on transmission and reception.

As with `bcopy()`, performance can be improved by operating on the largest units possible (i.e., quadwords). The Alpha AXP architecture does not include a carry bit, so we have to check if a carry has occurred. Because of the nature of the one's complement addition algorithm, it is not necessary to add the carry in at each stage; we just accumulate the carries and add them all in at the end. By operating on two cache lines at a time, we may start the next computation while the carry computation is under way, accumulate all the carries together, then add them all into the result (with another check for carry) at the end of processing the two cache lines. This results in four cycles per quadword with the addition of some end-of-loop computation to process the accumulated

carries. Interleaving the checksum computation across two cache lines also allows for some dual-issue effects that allow us to absorb the extra end-of-loop computation.

DEFTA Device Driver Enhancements

Preliminary measurements performed with the DEC FDDIcontroller/TURBOchannel adapter (DEFTA) and the OSF/1 device driver combination on DEC 3000 AXP Model 500 workstations indicated that we were able to receive the full FDDI bandwidth and deliver these packets in memory to the data link user. Although we show in this paper that the DEC OSF/1 for Alpha AXP system is able to also deliver the data to the user application, we ensure that the solutions provided by the driver are general enough to perform well even on a significantly slower machine. When executing on such a slow system, resources at the higher protocol layers (buffering, processing) may be inadequate to receive packets arriving at the maximum FDDI bandwidth, and the device driver has to deal with the overload. One of the primary contributions of the DEFTA device driver is that it avoids receive livelocks under very heavy receive load.

First, the queues associated with the different protocols are increased to a much larger value (512) instead of the typical size of 50 entries. This allows us to ride out transient overloads. Second, to manage extended overload periods, the driver uses the capabilities in the adapter to efficiently manage receive interrupts. The driver ensures that packets are dropped in the adapter when the host is starved of resources to receive subsequent packets. This minimizes wasted work by the host processor. The device driver also tends to trade off memory for computing resources. The driver allocates page-size mbufs (8K bytes) so that we minimize the overhead of memory allocation, particularly for large messages.

For transmitting packets, the driver takes advantage of the DEFTA's ability to gather data from different pieces of memory to be transmitted as a single packet. Up to 255 mbufs in a chain (although typically the chain is small, less than 5) may be transmitted as a packet. In the unusual case that a chain of mbufs is even longer than 255, we copy the last set of mbufs into a single large page-size mbuf, and then hand the packet to the device for transmission. This enables applications to have considerable flexibility, without resulting in extraneous data

movement operations to place data in contiguous memory locations.

In addition, the driver implements a policy to achieve transmit fairness. Although the operating system's scheduling provides fairness at a higher level, the policies within the driver allow for progress on transmits even under very heavy receive overload. Although the Alpha AXP systems are capable of receiving the full FDDI bandwidth, the enhanced transmit fairness may still be a benefit under bursty receive loads during which timely transmission is still desirable. In addition, as transmission links become faster, this feature will be valuable.

Wherever possible, all secondary activities—excluding the transmit and receive paths—have been implemented using threads. Scheduling secondary activity at a lower priority does not impact the latency of transmit and receive paths.

Improvements to the TCP/IP Protocol and Implementation

The initial TCP window size is set to a default or to the modified value set by the application through socket options. TCP in BSD 4.3 performed a rounding of the socket buffer, and hence the offered window size, to some multiple of the maximum segment size (MSS). The implementation in BSD 4.3 performed a rounding down to the nearest multiple of the MSS. The MSS value is adjusted, when it is greater than the page size, to a factor of the page size.

When using a socket buffer size of 16K bytes, the rounding down to a multiple of the MSS on FDDI results in the number of TCP segments outstanding never exceeding three. Depending on the application message size and influenced by one or more of both the silly window syndrome avoidance algorithms and the delayed acknowledgment mechanism, throughput penalties can be incurred.^{16,24}

Our choice in this area was to perform a rounding up of the socket buffer, and hence window size. This enabled existing applications to maintain performance regardless of changes to the buffering performed by the underlying protocol. For example, applications coded before the rounding of the buffer was implemented may have specified a buffer size at some power of 2. We believe it also allows better performance when interoperating with other vendors' systems and provides behavior

that is more consistent to the user (they get at least as much buffering as they request).

A buffer size of 4K bytes has long been obsolete for TCP connections over FDDI. Digital chose to increase this buffer to 16K bytes for ULTRIX support of FDDI. With a socket buffer of 16K bytes, even when rounding up is applied, the amount of data is limited to 17,248 octets per round-trip time. We found that the throughput over FDDI is limited by the window size. This is due to the effects of scheduling data packet processing and acknowledgments (ACKs), the interactions with window flow control, and FDDI's token access protocol (described below).^{23,25}

With memory costs decreasing considerably, we no longer consider the 16K byte default to be an appropriate trade-off between memory and throughput. Based on measurements for different values of the window size, we feel that the default window size of 32K bytes is reasonable. Increasing the window size from 16K bytes to 32K bytes results in an increase of the peak throughput over FDDI from approximately 40 Mb/s to approximately 75 Mb/s. However, increasing the window size beyond 32K bytes allowed us to increase the throughput even further, which led us to the incorporation of the TCP window scale extension.

Window Scale Extensions for TCP The implementation of TCP in DEC OSF/1 version 1.2 is based on the BSD 4.3 Reno distribution. In addition, we incorporated the TCP window scale extensions based on the model proposed in RFC 1323.¹⁰ Our work followed the implementation placed in the public domain by Thomas Skibo of the University of Illinois.

The TCP window scale extension maps the 16-bit window size to a 32-bit value. The TCP window scale option occupies 3 bytes and contains the type of option (window scale), the length of the option (3 bytes), and the "shift-count." The window scale value is a power of 2 encoded logarithmically. The shift-count is the number of bits that the receive window value is right-shifted before transmission. For example, a window shift-count of 3 and a window size of 16K would inform the sender that the receive window size was 128K bytes. The shift-count value for window scale is limited to 14. This allows for windows of $(2^{16} + 2^{14}) = 2^{30} = 1$ GB. To facilitate backward compatibility with existing implementations, both peers must offer the win-

ow scale option to enable window scaling in either direction.

The window scale option is sent only at connection initialization time in an <SYN> segment. Therefore the window scale value is fixed when the connection is opened. Since the window scale option is negotiated at initialization time, only a bit-shift to the window is added to the established path processing and has little effect on the overall cost of processing a segment.

Changes made to the OSF/1 TCP implementation for using the window scale option include the addition of the send window shift-count field and receive window shift-count field to the TCP control block. TCP processing was modified: the receive window shift-count value was computed based on the receive socket buffer size, and the window scale option is sent with the receive window shift-count. A modification at connection initialization time allows the received shift-count value to be stored in the send window shift-count, if TCP receives an <SYN> segment containing a window scale option. The receive window shift-count field is assigned to the window scale option that is sent on the <SYN, ACK> segment. When the TCP enters established state for the connection, window scale is turned on if both sides have sent <SYN> segments with window scale. For every incoming segment, the window field in the TCP header is left-shifted by the send window shift-count. For every outgoing segment, the window field in the TCP header is right-shifted by the receive window shift-count.

Measurement Results with TCP with Alpha AXP Workstations We used the inett tool to measure the throughput with TCP on the DEC OSF/1 operating system between two DEC 3000 AXP Model 500 workstations on a private FDDI ring. We observed that as the window size increased from 32K bytes to 150K bytes, the throughput generally increased for message sizes greater than 3,072 bytes. For example, for a user message size of 8,192 bytes, the throughput with a window size of 32K bytes was 72.6 Mb/s and increased to 78.3 Mb/s for a window size of 64K bytes. The TCP throughput rose to 94.5 Mb/s for a window size of 150K bytes. For window sizes beyond 150K bytes, we did not see a substantial, consistent improvement in throughput between the two user applications in this environment.

We believe that window scale is required to achieve higher throughputs—even in a limited

FDDI token ring of two stations—based on the interactions that occur between the token holding time, the scheduling of activities in the operating system, and the behavior of TCP. The default value for TTRT is set to 8 milliseconds,²¹ The end system is able to transmit packets at essentially the full FDDI bandwidth of 100 Mb/s, thus potentially consuming about 350 microseconds (including CPU and network interface times) to transmit a maximum-sized FDDI TCP segment of 4,312 bytes. During the 8 milliseconds, the source is able to complete the entire protocol processing of about 23 to 24 segments (approximately 100K bytes).

Further overlap of user data and protocol processing of packets can occur while the data link is transmitting and the sink is generating acknowledgments, if there is adequate socket buffer space in the source system. Thus, with the additional window of approximately 20K bytes to 30K bytes, the source system is able to pre-process enough segments and provide them to the adapter. The adapter may begin transmitting when the token is returned to the sender (after it receives a set of acknowledgments), while the source CPU is processing the acknowledgments and packetizing additional user data. With up to 150K bytes of socket buffer (and hence window), there is maximal overlap in processing between the CPU, the adapter, and the FDDI token ring, which results in higher throughput. This also explains why no further increases in the window size resulted in any significant increase in throughput.

Figure 3 shows the throughput with TCP between two DEC 3000 AXP Model 500 workstations on an isolated FDDI token ring for different message sizes for socket buffer sizes of 32K, 64K, and 150K bytes. For 150K bytes of socket buffer, the peak throughput achieved was 94.5 Mb/s. For all message sizes, we believe that the CPU was not fully utilized. Application message sizes that are slightly larger than the maximum transmission unit size traditionally display some small throughput degradation due to additional overhead incurred for segmentation and the subsequent extra packet processing. We do not see this in Figure 3 because the CPU is not saturated (e.g., approximately 60 percent utilized at message sizes of 8K bytes), and therefore the overhead for segmentation does not result in lower throughput.

So too, application message sizes that are larger than the discrete memory buffer sizes provided by the memory allocator should incur small amounts

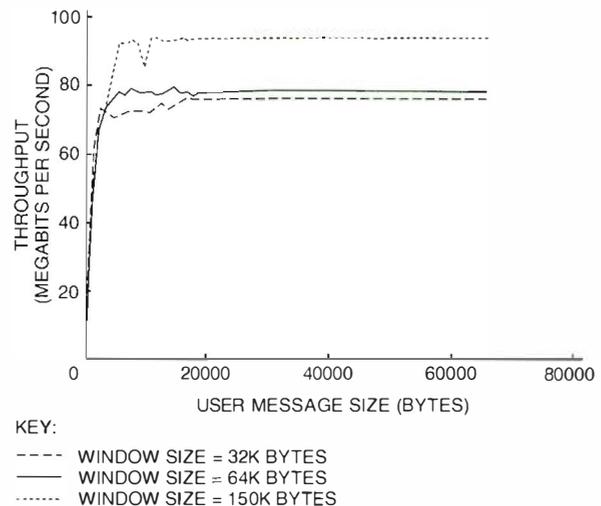


Figure 3 TCP Throughput as a Function of Window Size: Two DEC 3000 AXP Model 500 Workstations on an Isolated FDDI Ring

of extra overhead due to the necessity of chaining such buffers. Figure 3 also shows that the throughput degradation in this case is small.

Improvements to the UDP/IP Protocol Implementation and Measurement Results

UDP is a connection-less, message-oriented transport, with no assurances of reliable delivery. It also does not provide flow control. Unlike TCP, the UDP transmitter does not buffer user data. Therefore user messages are transmitted directly as packets on the FDDI. When user messages are larger than the MTU size of the data link (4,352 bytes), IP fragments the data into multiple packets. To provide data integrity, UDP uses the one's complement checksum for both data as well as the UDP header.

In our experience, the receive throughput to applications using UDP/IP with BSD-derived systems is quite poor due to many reasons, including the lack of flow control. Looking at the receive path of incoming data for UDP, we see that packets (potentially fragments) of a UDP message generate a high-priority interrupt on the receiver, and the packet is placed on the network layer (IP) queue by the device driver. The priority is reduced, and a new thread is executed that processes the packet at the IP layer. Subsequently, fragments are reassembled and placed in the receiver's socket buffer. There is a

finite IP queue and also a finite amount of socket buffer space. If space does not exist in either of these queues, packets are dropped. Provided space exists, the user process is then woken up to copy the data from the kernel to the user's space. If the receiver is fast enough to allow the user application to consume the data, the loss rate is low. However, as a result of the way processing is scheduled in UNIX-like systems, receivers experience substantial loss. CPU and memory cycles are consumed by UDP checksums, which we enable by default for OSF/1. This overhead in addition to the overhead for data movement contributes to the receiver's loss rate.

Table 1 shows the receive throughput and message loss rate with the original UDP implementation of OSF/1 for different message sizes. We modified the way in which processing is performed for UDP in the receiver in DEC OSF/1 version 1.2. We reorder the processing steps for UDP to avoid the detrimental effects of priority-driven scheduling, wasted work, and the resulting excessive packet loss. Not only do we save CPU cycles in processing, we also speed up the user application's ability to consume data, particularly as we go to larger message sizes. Table 1 gives the receive throughput and message loss rate with DEC OSF/1 version 1.2 incorporating the changes in UDP processing we have implemented.

UDP throughput was measured between user applications transmitting and receiving different size messages. Figure 4 shows the throughput at the transmitter, which is over 96 Mb/s for all message sizes over 6,200 bytes and achieves 97.56 Mb/s for the message size of 8K bytes used by NFS. During these measurements, the transmitting CPU was still not saturated and the FDDI link was perceived to be the bottleneck. Therefore, to stress the source system further, we used two FDDI adapters in the

system to transmit to two different receivers on different rings. Figure 4 also shows the aggregate transmit throughput of a single DEC 3000 AXP Model 500 workstation transmitting over two FDDI rings simultaneously to two different sinks. The source system is capable of transmitting significantly over the FDDI bandwidth of 100 Mb/s. For the typical NFS message size of 8,192 bytes, the aggregate transmit throughput was over 149 Mb/s. The throughput of the two streams for the different message sizes, indicates that, for the most part, their individual throughputs were similar. This showed that the resources in the transmitter were being divided fairly between the two applications.

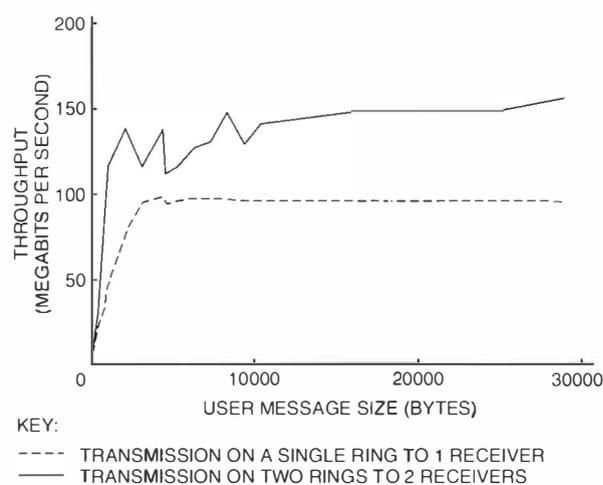


Figure 4 UDP Transmit Throughput: Single DEC 3000 AXP Model 500 Workstation Transmitting as Fast as Possible to Single Ring and Receiver and Two Receivers on Different Rings

Table 1 UDP Receive Characteristics with Peer Transmitter Transmitting at Maximum Rate

Message Size (bytes)	UDP Receive Before Changes		UDP Receive After Changes	
	Throughput (Mb/s)	Message Loss Rate	Throughput (Mb/s)	Message Loss Rate
128	0.086	98.8%	0.64	83.1%
512	0.354	98.5%	15.14	35.15%
1024	0.394	99.16%	23.77	46.86%
4096	9.5	90.26%	96.91	1.08%
8192	NA*	NA*	97.01	0.56%

* NA: Benchmark did not finish because of significant packet loss in that experiment.

Measurements of TCP/IP and UDP/IP with FDDI Full-duplex Mode

Earlier we observed that the behavior of TCP in particular depended on the characteristics of the timed-token nature of FDDI. One of the modes of operation of FDDI that we believe will become popular with the deployment of switches and the use of point-to-point FDDI is that of full-duplex FDDI. Digital's full-duplex FDDI technology, which is being licensed to other vendors, provides the ability to send and receive simultaneously, resulting in significantly higher aggregate bandwidth to the station (200 Mb/s). More important, we see this technology reducing latency for point-to-point connections. There is no token rotating on the ring, and the station does not await receipt of the token to begin transmission. A station has no restrictions based on the token-holding time, and therefore it is not constrained as to when it can transmit on the data link. The DEC FDDIcontroller/TURBOchannel adapter (DEFTA) provides the capability of full-duplex operation. We interconnected two DEC 3000 AXP Model 500 workstations on a point-to-point link using the DEFTAs and repeated several of the measurements reported above.

One of the characteristics observed was that the maximum throughput with TCP/IP between the two Alpha AXP workstations, even when using the default 32K bytes window size, reached 94.47 Mb/s. Figure 5 shows the behavior of TCP throughput

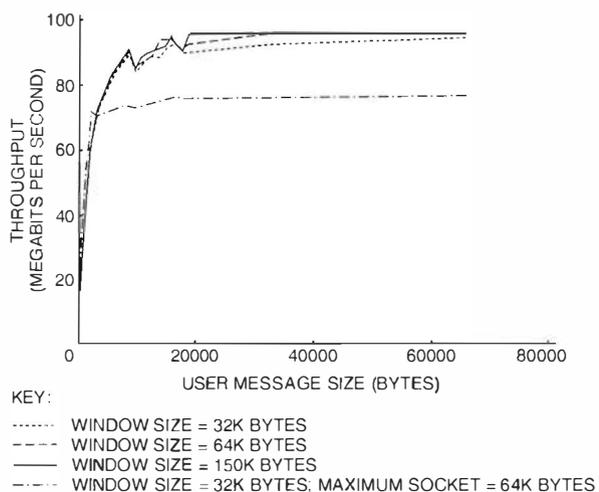


Figure 5 TCP Throughput as a Function of Window Size: Two DEC 3000 AXP Model 500 Workstations with Full-duplex FDDI

with full-duplex FDDI operation for different window sizes of 32K, 64K, and 150K bytes (when window scale is used). The throughput is relatively insensitive to the variation in the window size. For all these measurements, however, we maintained the value of the maximum socket buffer size to be 150K bytes. When using a smaller value of the maximum socket buffer size (64K bytes), the throughput drops to 76 Mb/s (for a window size of 32K bytes) as shown in Figure 5.

Although we removed one of the causes of limiting the throughput (token-holding times), full-duplex operation still exhibits limitations due to scheduling the ACK and data packet processing and the resulting lack of parallelism in the different components in the overall pipe (the two CPUs of the stations, the adapters, and the data link) with small socket buffers. Increasing the maximum socket buffer allows for the parallelism of the work involved to provide data to the protocol modules on the transmitter.

Observing the UDP/IP throughput between the DEC 3000 AXP Model 500 workstations, we found a slight increase in the transmit throughput over the normal FDDI mode. For example, the UDP transmit throughput for 8K messages was 97.93 Mb/s as compared to 97.56 Mb/s using a single ring in normal FDDI mode. This improvement is due to the absence of small delays for token rotation through the stations as a result of using the full-duplex FDDI mode.

Experimental Work

We have continued to work on further enhancing the implementation of TCP and UDP for DEC OSF/1 for Alpha AXP. We describe some of the experimental work in this section.

Experiments to Enhance the Transmit and Receive Paths for TCP/IP

The `bcopy()` and `in_checksum()` routine optimizations minimize the incremental overhead for packet processing based on packet sizes. The protocol processing routines (e.g., TCP and IP) also minimize the fixed per-packet processing costs.

All TCP output goes through a single routine, `tcp_output()`, which often follows the TCP pseudocode in RFC 793 very closely.⁹ A significant portion of its implementation is weighed down by code that is useful only during connection start-up and shutdown, flow control, congestion, retransmissions and persistence, processing out-of-band data, and so on. Although the actual code

that handles these cases is not executed every time, the checks for these special cases are made on every pass through the routine and can be a non-trivial overhead.

Rather than check each case separately, the TCP/IP code was modified to maintain a bit mask. Each bit in the mask is associated with a special condition (e.g., retransmit, congestion, connection shutdown, etc.). The bit is set whenever the corresponding condition occurs (e.g., retransmit time-out) and reset when the condition goes away. If the bit mask is 0, the TCP/IP code executes straightline code with minimal tests or branches, thus optimizing the common case. Otherwise, it simply calls the original routine, `tcp_output`, to handle the special conditions. Since the conditions occur rarely, setting and resetting the bits incurs less overhead than performing the tests explicitly every time a packet is transmitted. Similar ideas have been suggested by Van Jacobson.²⁶

Additional efficiency is achieved by precomputing packet fields that are common across all packets transmitted on a single connection. For example, instead of computing the header checksum every time, it is partially precomputed and incrementally updated with only the fields that differ on a packet-by-packet basis.

Another example is the data link header computation. The original path involved a common routine for all devices, which queues the packet to the appropriate driver, incurs the overhead of multiplexing multiple protocols, looking up address resolution protocol (ARP) tables, determining the data link formats, and then building the header. For TCP, once the connection is established, the data link header rarely changes for the duration of the connection. Hence at connection setup time, the data link header is prebuilt and remembered in the TCP protocol control block. When a packet is transmitted, the data link header is prefixed to the IP header, and the packet is directly queued to the appropriate interface driver. This avoids the overhead associated with the common routine. Network topology changes (e.g., link failures) may require the data link header to be changed. This is handled through retransmission time-outs. Whenever a retransmit time-out occurs, the prebuilt header is discarded and rebuilt the next time a packet has to be sent.

Some parameters are passed from TCP to IP through fields in the mbufs. Combining the layers eliminates the overhead of passing parameters and validating them. Passing parameters is a nontrivial

cost, since in the original implementation, some data was passed as fields in the mbuf structure. Because these were formatted in network byte order, building and extracting them incurred overhead. Moreover, the IP layer does not have to perform checks for special cases that are not applicable to the TCP connection. For example, no fragmentation check is needed since the code for TCP has already taken care to build a packet within the allowed size limits.

In a similar fashion to the transmit path, a common-case fast path code was implemented for the receive side. This mimics the most frequently executed portions of the TCP/IP input routines, and relegates special cases and errors to the original code. Special cases include fragmented packets, presence of IP options, and noncontiguous packet headers. Combining error checking across TCP and IP also eliminates additional overhead. For example, length checks can be used to detect the presence of options that can be passed to the original general case path.

These fast path optimizations were implemented in an experimental version of the OSF/1 operating system. TCP measurements on the experimental version of OSF/1 running on two systems communicating over a private FDDI ring indicate that, when both the input and output fast path segments are enabled on the two systems, throughput is improved for almost all message sizes.

Experiments to Enhance UDP/IP Processing

An enhancement for UDP/IP processing with which we experimented was to combine the data copying and checksum operations. This has been attempted in the past.²⁷ The primary motivation is to reduce memory bandwidth utilization and perform the checksums while the data is in the processor during the data movement. To allow us to do this, we introduce a new UDP protocol-specific socket option that allows users to take advantage of this optimization. When a user application posts a receive buffer after enabling this socket option, we invoke a combined copy and checksum routine on receiving a packet for that user. In the infrequent case when the checksum fails, we restore the user I/O structure and zero the user buffer so that inappropriate data is not left in a user's buffer. Preliminary performance measurements indicate significant reduction in CPU utilization for UDP receives when using this socket option.

Experiments to Eliminate the Data Copy from User to Kernel Space

As observed earlier, data movement operations add significant overhead on the end system. One method to reduce the cost of data movement for a send operation, prototyped on an experimental version of the OSF/1 operating system, is to replace the data copy from user space to the kernel socket buffer by a new virtual memory page remap function. Instead of copying the data from physical pages in the user map to physical pages in the kernel map, the physical pages associated with the user virtual address range in the user map are remapped to kernel virtual addresses. The pages associated with the new kernel virtual addresses are then masqueraded through the network as mbufs. Preliminary results indicate that a virtual memory mapping technique can be used on the OSF/1 operating system to significantly reduce the overhead associated with the transmission of messages.

The underlying design of the remap operation affects application semantics and performance. The semantics of the application are affected by which underlying page remap operation is selected. Performance may also be affected by the implementation of the page map operation and how well certain TCP/IP configuration variables are tuned to match the processor architecture and the network adapter capabilities.

Two types of remap operations were prototyped: page steal and page borrow. The page steal operation, as the name implies, steals the pages from the user virtual address space and gives the pages to the kernel. The user virtual addresses are then mapped to demand-zero pages on the next page reference. In the page steal operation, the user ends up with demand zero pages. On the other hand, in the borrow page operation, the same physical pages are given back to the user. If the user accesses a page that the kernel was still using, the user process either "sleeps," waiting for that page to become available or (depending upon the implementation) receives a copy of the page. For the page borrow operation, the user buffer size must be greater than the socket buffer size, and the user buffer must be referenced in a round-robin fashion to ensure that the application does not sleep or receive copies of the page.

Both the page steal and the page borrow operations change the semantics of the send() system calls, and some knowledge of these new semantics of the send system calls needs to be reflected in the

application. The application's buffer allocation and usage is dependent upon how the underlying remap operation is implemented. An important consideration is the impact on the application programming interface. In particular, the extent to which the semantics of the send system calls (e.g., alignment requirements for the user message buffer) need to change to support the remap operations is an area that is currently under study.

The page remap feature has not yet been incorporated in the DEC OSF/1 version 1.2 product. Inclusion of this feature in the product is expected to reduce CPU utilization. While page remapping does reduce the cost of processing a packet, the design issues outlined above impact applications. To achieve performance benefits and application portability across multiple heterogeneous open systems, future work continues in this area. In addition, integrated hardware solutions to reduce the cost of the copy operation are also under investigation.

The performance numbers presented in this paper did not include the improvements described in this section on experimental work. We anticipate that the overall performance would see substantial improvement with the inclusion of these changes.

Conclusions

Increases in communication link speeds and the dramatic increases in processor speeds have increased the potential for widespread use of distributed computing. The typical throughput delivered to applications, however, has not increased as dramatically. One of the primary causes has been that network I/O is intensive on memory bandwidth, and the increases in memory bandwidths have only been modest. We described in this paper an effort using the new Alpha AXP workstations and the DEC OSF/1 operating system for communication over FDDI to remove this I/O bottleneck from the end system.

We described the characteristics of the DEC 3000 AXP Model 500 workstation which uses Digital's Alpha AXP 64-bit RISC microprocessor. With the use of wider access to memory and the use of multilevel caches, which are coherent with DMA, the memory subsystem provides the needed bandwidth for applications to achieve substantial throughput while performing network I/O.

We described the implementation of the internet protocol suite, TCP/IP and UDP/IP, on the DEC OSF/1 operating system. One of the primary characteristics of the design is the need for data movement

across the kernel-user address space boundary. In addition, both TCP and UDP use checksums for the data. Both these operations introduce increasing overhead with the user message size and comprise a significant part of the total processing cost. We described the optimizations performed to make these operations efficient by taking advantage of the wider cache lines for the systems and the use of 64-bit operations.

We incorporated several optimizations to the implementation of TCP in the DEC OSF/1 operating system. One of the first was to increase the default socket buffer size (and hence the window size) used by TCP from the earlier, more conservative 4K bytes to 32K bytes. With this, the throughput of a TCP connection over FDDI between two Alpha AXP workstations reached 76.6 Mb/s. By increasing the window size even further, we found that the throughput increases essentially to the full FDDI bandwidth. To increase the window size beyond 64K bytes requires the use of recent extensions to TCP using the window scale option. The window scale option, which is set up at the connection initialization time, allows the two end systems to use much larger windows. We showed that, when using a window size of 150K bytes, the peak throughput of the TCP connection increases to 94.5 Mb/s.

We also improved the performance of UDP through implementation optimizations. Typical BSD-derived systems experience substantial loss at the receiver when two peer systems communicate using UDP. Through simple modifications in the processing for UDP and reordering the processing steps, we improved the delivered throughput to the receiving application substantially. The UDP receive throughput at the application achieved was 9756 Mb/s for the typical NFS message size of 8K bytes. Even at this throughput, we found that the CPU of the transmitter was not saturated. When a transmitter was allowed to transmit over two different rings (thus removing the communication link as the bottleneck) to two receivers, a single Alpha AXP workstation (DEC 3000 AXP Model 500) is able to transmit an aggregate throughput of more than 149 Mb/s for a message size of 8K bytes.

We also described throughput measurements with the FDDI full-duplex mode between two Alpha AXP workstations. With full-duplex mode there are no latencies which are associated with token rotation, lost token recovery, or limitations on the amount of data transmitted at a time as imposed by the FDDI timed-token protocol. As a result, with

full-duplex mode there are performance improvements. With TCP, we achieve a throughput of 94.5 Mb/s even with the default socket buffer of 32K bytes. This is smaller than the buffer size needed in token passing mode to achieve the same level of throughput. Since the link becomes the bottleneck at this point, there is no substantial increase in throughput achieved with the use of window scaling when FDDI is being used in full-duplex mode. An increase in peak transmit throughput with UDP is also seen when using FDDI in full-duplex mode.

Finally, a few implementation ideas currently under study were presented.

Acknowledgments

This project could not have been successful without the help and support of a number of other individuals. Craig Smelser, Steve Jenkins, and Kent Ferson were extremely supportive of this project and ensured that the important ideas were incorporated into the OSF V1.2 product. Tim Hoskins helped tremendously by providing many hours of assistance in reviewing ideas and the code before it went into the product. In addition, we thank the engineers who ported DEC OSF/1 to Alpha AXP in order to provide a stable base for our work. The DEFTA product development group led by Bruce Thompson and Tom Cassa not only provided us with a nice adapter, but also helped by giving us as many prototype adapters as we needed on very short notice. We would like to thank Gary Lorenz in particular for his help with the DEFTA adapters.

References

1. D. Clark and D. Tennenhouse, "Architectural Considerations for a New Generation of Protocols," *Proceedings of the Symposium on Communications Architectures and Protocols, ACM SIGCOMM 1990, ACM Computer Communications Review*, vol. 20, no. 4 (September 1990).
2. J. Lumley, "A High-Throughput Network Interface to a RISC Workstation," *Proceedings of the IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems*, Tucson, AZ (February 17-19, 1992).
3. P. Druschel and L. Peterson, "High-performance Cross-domain Data Transfer," Technical Report TR93-5, Department of Computer

- Science (Tucson, AZ: University of Arizona, March 1993).
4. G. Chesson, "XTP/PE Overview," *Proceedings of the 13th Conference on Local Computer Networks*, Minneapolis, MN (October 1988).
5. *FDDI Media Access Control*, American National Standard, ANSI X3.139-1987.
6. *FDDI Physical Layer Protocol*, American National Standard, ANSI X3.148-1988.
7. J. Postel, "User Datagram Protocol," RFC 768, SRI Network Information Center, Menlo Park, CA (August 1980).
8. J. Postel, "Internet Protocol," RFC 791, SRI Network Information Center, Menlo Park, CA (September 1981).
9. J. Postel, "Transmission Control Protocol," RFC 793, SRI Network Information Center, Menlo Park, CA (September 1981).
10. V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance," RFC 1323, Internet Engineering Task Force (February 1991).
11. K. Ramakrishnan, "Performance Considerations in Designing Network Interfaces," *IEEE Journal on Selected Areas in Communications, Special Issue on High Speed Computer/Network Interfaces*, vol. 11, no. 2 (February 1993).
12. S. Leffler, M. McKusick, M. Karels, and J. Quarterman, *The Design and Implementation of the 4.3 BSD UNIX Operating System* (Reading, MA: Addison-Wesley Publishing Company, May 1989).
13. R. Sites, ed., *Alpha Architecture Reference Manual* (Burlington, MA: Digital Press, 1992).
14. D. Dobberpuhl et al., "A 200-MHz 64-bit Dual-issue CMOS Microprocessor," *Digital Technical Journal*, vol. 4, no. 4 (Special Issue 1992): 35-50.
15. T. Dutton, D. Eiref, H. Kurth, J. Reisert, and R. Stewart, "The Design of the DEC 3000 AXP Systems, Two High-performance Workstations," *Digital Technical Journal*, vol. 4, no. 4 (Special Issue 1992): 66-81.
16. R. Braden, "Requirements For Internet Hosts—Communication Layers," RFC 1122, Internet Engineering Task Force (October 1989).
17. R. Braden, "Requirements For Internet Hosts—Application and Support," RFC 1123, Internet Engineering Task Force (October 1989).
18. D. Katz, "Transmission of IP and ARP over FDDI Networks," RFC 1390, Internet Engineering Task Force (January 1993).
19. V. Jacobson, "Congestion Avoidance and Control," *Proceedings of the Symposium on Communications Architectures and Protocols, ACM SIGCOMM 1988, ACM Computer Communications Review*, vol. 18, no. 4 (August 1988).
20. R. Grow, "A Timed Token Protocol for Local Area Networks," Presented at Electro/82, Token Access Protocols, Paper 17/3, May 1982.
21. R. Jain, "Performance Analysis of FDDI Token Ring Networks: Effect of Parameters and Guidelines for Setting TTRT," *Proceedings of the Symposium on Communications Architectures and Protocols, ACM SIGCOMM 1990, ACM Computer Communications Review*, vol. 20, no. 4 (September 1990).
22. M. Wenzel, "CSR Architecture (DMA Architecture)," IEEE P1212 Working Group Part III-A, Draft 1.3, May 15, 1990.
23. K. Ramakrishnan, "Scheduling Issues for Interfacing to High Speed Networks," *Proceedings of Globecom '92 IEEE Global Telecommunications Conference, Session 18.04*, Orlando, FL (December 6-9, 1992).
24. D. Clark, "Window and Acknowledgment Strategy in TCP," RFC 813, SRI Network Information Center, Menlo Park, CA (July 1982).
25. L. Zhang, S. Shenker, and D. D. Clark, "Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic," *Proceedings of the Symposium on Communications Architectures and Protocols, ACM SIGCOMM 1991, ACM Computer Communication Review*, vol. 21, no. 4 (September 1991).

26. V. Jacobson, "Efficient Protocol Implementation," *ACM SIGCOMM 1990 Tutorial on Protocols for High-Speed Networks, Part B* (September 1990).
27. C. Partridge and S. Pink, "A Faster UDP," Swedish Institute of Computer Science Technical Report (August 1991).

General References

E. Cooper, O. Menzilcioglu, R. Sansom, and F. Bitz, "Host Interface Design for ATM LANs," *Proceedings of the 16th Conference on Local Computer Networks*, Minneapolis, MN (October 1991).

B. Davie, "A Host-Network Interface Architecture for ATM," *Proceedings of the Symposium on Communications Architectures and Protocols, ACM SIGCOMM 1991, ACM Computer Communication Review*, vol. 21, no. 4 (September 1991).

H. Kanakia and D. Cheriton, "The VMP Network Adapter Board (NAB): High Performance Network Communication for Multiprocessors," *Proceedings*

of the Symposium on Communications Architectures and Protocols, ACM SIGCOMM 1988, ACM Computer Communication Review, vol. 18, no. 4 (August 1988).

M. Nielsen, "TURBOchannel," *Proceedings of 36th IEEE Computer Society International Conference, COMPCON 1991*, February 1991.

P. Steenkiste, "Analysis of the Nectar Communication Processor," *Proceedings of the IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems*, Tucson, AZ (February 17-19, 1992).

C. Traw, S. Brendan, and J. M. Smith, "A High-Performance Host Interface for ATM Networks," *Proceedings of the Symposium on Communications Architectures and Protocols, ACM SIGCOMM 1991, ACM Computer Communication Review*, vol. 21, no. 4 (September 1991).

TURBOchannel Developer's Kit, Version 2 (Maynard, MA: Digital Equipment Corporation, September 1990).

Routing Architecture

Digital developed the intermediate system-to-intermediate system (IS-IS) intra-domain routing information exchange protocol for the DECnet Phase V network layer architecture. This protocol, which has been adopted by the International Organization for Standardization, is based on a link state routing algorithm. The benefits derived from the IS-IS protocol include a self-stabilizing method for reliable link state packet distribution, a hierarchical network structure to support larger networks, protocols for efficiently utilizing local area networks, and simultaneous support for multiple network layer protocols.

The network layer architecture has three basic components. The first concerns the transmission of data packets from one end system (a host) to a remote end system, regardless of whether or not these packets are sent by way of routers. The main features of this component are packet formats and addressing. Standards for these features are defined in the connectionless network layer protocol (CLNP), adopted by the International Organization for Standardization (ISO), and in the internet protocol (IP), the equivalent standard in the transmission control protocol/internet protocol (TCP/IP) suite.^{1,2}

The second component relates to handshaking between neighbors (i.e., directly connected systems) and mapping network layer addresses to data link layer addresses. The ISO protocol that performs this function is the end system-to-intermediate system (ES-IS) protocol.³ The address resolution and internet control message protocols provide most of the same functionality in the TCP/IP protocol suite.^{4,5}

The third component of the network layer architecture pertains to routing. The routing protocol developed for Digital's DECnet Phase V network architecture and adopted by the ISO is the intermediate system-to-intermediate system (IS-IS) intra-domain routing information exchange protocol.⁶

The architecture for DECnet Phase V allows support of many network layer protocols, i.e., CLNP, IP, Novell NetWare, and AppleTalk.⁷ Each network layer suite has its own protocols for the first two components of the network layer architecture. DECnet Phase V support for a particular network layer suite implies support for such protocols. Consequently, end systems that implement an existing network layer protocol need not be modified to

operate with DECnet Phase V routers (i.e., intermediate systems). This paper briefly discusses data packet formats, types of routing control packets, and neighbor handshaking protocols and then focuses on the third component of the network layer architecture, concentrating on the IS-IS routing protocol.

Support for any network protocol suite can be added easily to the IS-IS routing protocol. DECnet Phase V routing products currently support the DECnet Phase IV, CLNP/DECnet Phase V, and the IP protocols. Support for the Novell NetWare, XNS, and AppleTalk protocols is under investigation.

Data Packet Formats

A network layer data packet carries data, usually generated by higher-layer protocols, between host systems. The purpose of the network routing layer is to correctly deliver data packets to their destinations. To accomplish this task, additional pieces of information are required; these are carried in the header of the data packet. The most important function of the header is addressing. Each data packet must uniquely identify the source and destination addresses for the packet. Other important functions include: checksumming, to ensure that transmission errors are detected; fragmentation and reassembly, to allow the transmission of large packets over links that can support only smaller packets; error reporting, to notify someone should an error occur; security, to identify special security requirements of packets; quality of service maintenance, to ensure that the correct level of service is provided; and congestion notification, to notify the source and destination should congestion occur along the path of a data packet.

The DECnet Phase IV architecture uses a proprietary packet format for data exchange. The DECnet Phase V architecture continues to support this format to allow compatibility with existing Phase IV systems. However, DECnet Phase V uses the ISO CLNP standard for communication between DECnet and open systems interconnection (OSI) systems. Use of this standard protocol also permits DECnet Phase V systems to communicate with other vendors' end systems that implement the ISO standard. In addition, communication using IP is possible with systems that implement the TCP/IP suite.

DECnet Phase IV employs a 16-bit network layer addressing scheme. When using the CLNP, the addresses, known as network service access point (NSAP) addresses, vary in length up to 20 octets. Defining a common mapping procedure allows a DECnet Phase IV address to be expressed as an equivalent ISO NSAP address. Similarly, an ISO NSAP address thus derived, and therefore Phase IV compatible, may be converted back to the original Phase IV address. Converting the source and destination addresses and the packet formats enables any DECnet Phase IV packet to be translated into a CLNP packet and back again. Therefore, two DECnet Phase IV systems can communicate over a portion of a network that supports only the CLNP. Similarly, two DECnet/OSI (or even pure OSI) systems can communicate over a portion of the network that supports DECnet Phase IV, provided that the addresses chosen are Phase IV compatible.

Overview of Routing Control Packets

The IS-IS protocol uses three basic types of packets:

1. Hello Packet. The protocol uses Hello packets to keep track of neighbors. Routers determine the identity of neighbors and periodically check the status of the link to that neighbor by exchanging Hello packets.
2. Link State Packet. Link State Packets (LSPs) list, for each neighbor of the node issuing the LSP, the ID of that neighbor and the cost of the link to it. This list includes both router neighbors and end-system neighbors. The cost of the link is assigned by the network manager to reflect the desirability of using that link. A number of factors determine the cost, including throughput capacity and the monetary cost associated with using the link.
3. Sequence Number Packet. Sequence Number Packets (SNPs) are used to ensure that neighbor-

ing routers have the same notion of what is the most recent LSP from every other router. There are two types of SNPs: the Complete Sequence Number Packet (CSNP) and the Partial Sequence Number Packet (PSNP).

The CSNP lists all LSPs present in the issuing router's LSP database, together with their sequence numbers, and is used to synchronize LSP databases. The CSNP is transmitted upon link start-up on point-to-point links and periodically on a local area network (LAN). This use of the CSNP to ensure LSP database consistency of all routers on the LAN is described in more detail in the section Efficient Use of LANs.

The PSNP lists only a few LSPs and is used to explicitly acknowledge or request one or more LSPs.

Neighbor Handshaking Protocols

The architecture for DECnet Phase V uses the ES-IS protocol to enable routers and end systems on a LAN to learn about each other's presence. Every end system periodically multicasts an End System Hello protocol data unit (PDU) to the multicast address "All Intermediate Systems." This PDU contains the end system's NSAP address and permits the receiving routers to create an entry that maps the NSAP address to the corresponding data link address from which the PDU was received. The routers use this information to deliver data PDUs to the end systems and also to communicate the existence of the end systems to other routers by means of the routing protocols.

In a similar manner, all routers periodically multicast an Intermediate System Hello to the multicast address "All End Systems." This data permits the end systems to determine the data link addresses of all routers on the LAN. In the absence of other information, an end system will transmit any data PDUs destined for another system to one of the routers it has discovered. However, the router to which the data PDU is sent may not be the best path. Indeed, direct transmission of the data PDU to the destination system may be possible, if the source and destination systems are on the same LAN. In such cases, the router concerned sends a Redirect PDU back to the source end system. The Redirect contains the data link address to use for this NSAP address, which the end system can then use for subsequent transmissions.

The ES-IS protocol replaces the proprietary DECnet Phase IV initialization protocol for use

between the DECnet and OSI systems. However, operation of the DECnet Phase IV protocol is still necessary to enable handshaking between DECnet Phase IV and DECnet Phase V systems. To avoid confusion, the Phase IV initialization messages transmitted by Phase V systems have a version number that is acceptable to only Phase IV systems. Such messages are ignored by other Phase V systems.

Routing Protocols, with Emphasis on the IS-IS Protocol

Routing protocols are used to calculate the path, i.e., the route, that a data packet will take through a network. Typically, a routing protocol dynamically adjusts to network problems, such as failed links or routers, to ensure that the network continues to operate in a robust manner. Use of dynamic routing protocols also eases installation and configuration, because routes are calculated by means of the algorithm, not the user.

The two main types of dynamic routing protocols are distance vector and link state. Many routing protocols are based on distance vector routing, for example, DECnet Phase III, DECnet Phase IV, and the routing information protocol (RIP).⁸ In a distance vector protocol, each router is responsible for keeping track of and informing its neighbors about its distance (i.e., total cost) to each destination. The router computes its distance to each destination based on its neighbors' distances to each destination. The only information a router has to know a priori is its own ID and the cost of its links to each neighbor.

Consider the distance vector routing example shown in Figure 1. Suppose a router R with five ports is configured with costs c_1 , c_2 , c_3 , c_4 , and c_5 for each of the ports, respectively. Further suppose that the neighbor on port 1 informs R that it is d_1 from some destination D, the neighbor on port 2 informs R that it is d_2 from D, and so forth. R can then figure out its own distance to destination D. If the destination is R itself, then R's distance to D is 0. Otherwise, R's distance to D is the minimum value of $c_i + d_p$, for $i = 1$ through 5. If R receives a packet addressed to destination D, R should forward the packet through the port with minimum total cost to D.

Because of their slower convergence rate, distance vector protocols generally provide lower performance than link state protocols. Distance vector protocols adapt to changes in topology less quickly than link state protocols, and until the protocol

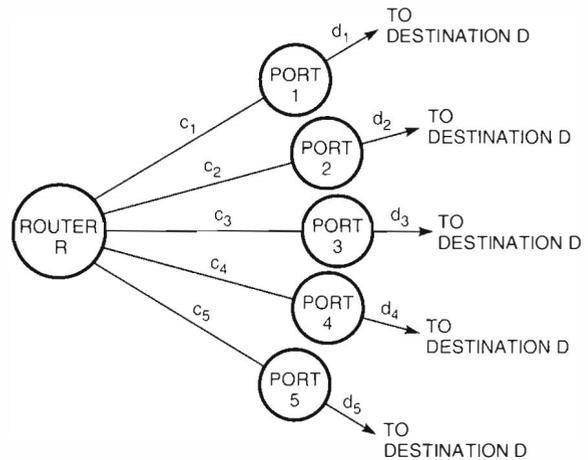


Figure 1 Distance Vector Routing

adapts to such a change, routing can be disrupted. The main reason for this convergence problem stems from incorrect information. When changes such as link failures occur in the network, the information that each node transmits to its neighbors is only that node's current impression of the distance to each destination, which may be incorrect information. Consequently, the distance vector algorithm may take several iterations to converge to the correct routes.

The first deployed link state routing protocol was developed by Bolt Beranek and Newman (BBN) for the Advanced Research Projects Agency Network (ARPANET).^{9,10} In link state routing, each router determines its local status and then constructs an LSP, defined earlier in the section Overview of Routing Control Packets. This LSP is transmitted (or "flooded") to all the other routers, which are responsible for storing the most recently generated LSP from each router.¹¹ (If the large size of the network makes it impractical for the LSP database to contain information for every other router, the network can be made hierarchical, as described in the Hierarchy section.) All routers (or all routers in an area, when hierarchical routing is used) then compute routes based on a complete topology. Figure 2 illustrates an example of link state routing, with a router R determining the state of its neighbors and then broadcasting this information by means of Hello Neighbor messages.

Link state algorithms respond rapidly and consistently to changes in networks, as compared with distance vector algorithms. Once the LSPs have been distributed, each router can calculate routes

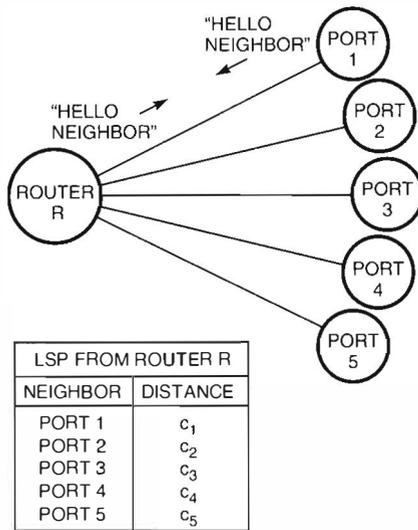


Figure 2 Link State Routing

without further reference to the other routers. The results are more stable routing and lower consumption of link bandwidth and router CPU. Therefore, the design of the IS-IS routing algorithm was based on the original BBN link state routing algorithm, which used an algorithm known as the shortest path first (SPF) to calculate the routes.¹²

The IS-IS protocol corrected many deficiencies and added extra functionality.

1. The IS-IS protocol provides a more stable method for reliably distributing LSPs. The ARPANET method was an early algorithm that used excessive overhead and was unstable in rare circumstances. The IS-IS protocol design uses a self-stabilizing protocol for LSP distribution that requires much less bandwidth.
2. The IS-IS protocol can be used in a hierarchical manner to support larger networks.
3. The ARPANET method assumed all connections were point-to-point links. Many nodes can be connected with a LAN. Modeling a LAN as a fully connected set of nodes attached with point-to-point links would be extremely inefficient. The IS-IS routing protocol incorporates protocols for efficiently utilizing LANs.
4. Given that a router has limited memory, the network can grow beyond a size that the router can support. If the router failed simply because its LSP database overflowed the available space, network management could not be used to

reconfigure the router. If the router continued to operate and based the routing on an incomplete database, loops might form and adversely affect routes that traverse that router. The IS-IS protocol has mechanisms that enable overloaded routers to remain reachable for network management.

5. Certain control packets can get very large. The IS-IS protocol has mechanisms for ensuring that fragments of a control packet can be dealt with independently rather than required to be fully reassembled first.
6. The IS-IS routing protocol can support many network layer protocols simultaneously. This support is known as Integrated IS-IS.¹³

Hierarchy

As a network grows, several factors may overload the routing protocol: the LSP database may become too large to fit into memory; computing routes may require too much CPU; the task of keeping the LSP databases up-to-date may consume too much bandwidth; or the network may be unstable because link changes are frequent. To deal with these factors, the IS-IS protocol allows the network to be partitioned into areas. Within an area, the level 1 routers keep track of all the nodes and links. Level 2 routers keep track of the location of the areas but are not concerned with the detail inside the areas. A level 2 router can also act as a level 1 router in one area.

To use the IS-IS protocol in a hierarchical way, it is convenient for the network layer addresses to be topologically hierarchical. Figure 3 illustrates the structure of an IS-IS address. All nodes in a particular area have the same value for the area address field of their address. A level 1 router looks at the area address portion of the destination address in a packet. If this field matches the router's area, the router assigns the packet a path based on the ID portion of the address. Otherwise, the router routes the packet toward a level 2 router, which directs the packet to the correct area.

The IS-IS protocol treats the last octet of the address as a selector, which is used only for demultiplexing multiple network users within the



Figure 3 IS-IS Address Structure

destination system. The selector field can therefore be ignored with respect to IS-IS routing.

In general, the area address itself is hierarchically subdivided. This structure is useful for address administration and for routing between routing domains, for example, different corporations, which may be interconnected by means of a public network. However, from the point of view of IS-IS operation, the entire area address is a single identifier for the area.

In a network of global dimensions, possibly comprising millions of addresses, the ability to use hierarchical addressing is essential to help provide some of the topological information. This addressing scheme is analogous to the use of country codes in international telephone numbers, which allows calls to be routed to other countries without complete knowledge of the internal structure of all the telephone systems in the world.

Efficient Use of LANs

All routers connected to a LAN are neighbors. If the routing protocol was simply to consider all pairs of nodes on the LAN as neighbors, then each router on the LAN would issue an LSP listing every node on the LAN. In addition, the LSP distribution would be inefficient if each router had to transmit every LSP to all other routers on the LAN and then receive acknowledgments from all these same routers.

The IS-IS protocol dramatically reduces the required size of the LSP database by considering the LAN as a pseudonode. Each router then claims to have one link to the pseudonode, rather than a link to every other router on the LAN. Only the pseudonode claims to have links to all the end systems on the LAN.

This approach requires that an LSP be transmitted for the pseudonode itself, and thus some router on the LAN has to take on the responsibility for transmitting the packet for the pseudonode. The router with the numerically highest priority (or, in the event of a tie, the highest data link address) is elected the designated router (DR). The DR gives a name to the LAN by appending an octet to its own ID.

For example, assume a LAN has 5 routers and 100 end systems, as shown in Figure 4. Let R5 be the elected DR. R5 might name the LAN R5.17. In that case, R1, R2, R3, R4, and R5 each issue an LSP listing the neighbor R5.17. R5 will issue a second LSP, from source R5.17, listing R1, R2, R3, R4, R5, and all the end systems (E1 through E100) as neighbors.

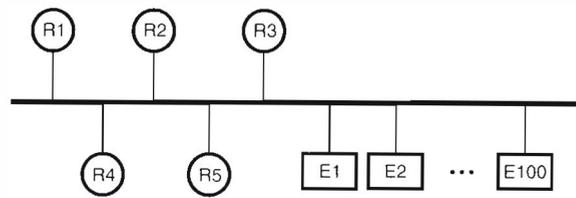


Figure 4 Local Area Network

The IS-IS protocol also contains special features to allow efficient distribution of LSPs on the LAN. IS-IS does not require explicit acknowledgments to LSPs on the LAN. Instead, a router that has an LSP to forward to the LAN simply multicasts the LSP to the other routers. A router that receives an LSP on the LAN will not multicast the same LSP on the LAN. Theoretically, if no packets get lost, only a single router would issue an LSP on the LAN.

However, packets do get lost, so the detection of lost LSPs is important. IS-IS detects lost LSPs by having the DR periodically broadcast a summary of the LSP database in a CSNP. Based on the CSNP, a receiving router can determine whether it has missed an LSP (in which case it will explicitly request the LSP from the DR), or it has a more recent LSP than the DR has (in which case the receiving router will multicast the LSP on the LAN to the other routers).

Database Overload

An implementation of a router typically has a finite amount of storage for the LSP database. Therefore, the router could receive an LSP and not be able to store it. The space may be inadequate for two reasons. First, the network may experience a static overload, i.e., the network may have become so large that the router cannot store the LSP database. Second, an ordering of events can temporarily make the LSP database larger than necessary, causing a temporary overload. For example, the DR on a large LAN may fail. The DR's previous pseudonode LSP is still in the other routers' databases. The new DR on the LAN will give the LAN a new ID and attempt to purge the previous pseudonode LSP. However, until the purge is complete, other routers will have to temporarily store twice as much information about that LAN.

Without considering this storage problem, a router implementation might employ any of the following strategies: the router might fail and recover only with operator intervention; the router might fail and reboot; or the router might ignore the

temporary overload and perform routing in the best way possible.

Each of these possible strategies is undesirable. If a router fails and needs human intervention to recover, routing will be disrupted longer than necessary if the problem is only temporary. Crashing and automatically rebooting is desirable if the overload is very short-lived (so the overload condition is corrected before the router has rebooted). Otherwise, this strategy can cause long-term instability, since after rebooting, the router starts to exchange routing information with neighbors, only to eventually overload and fail again. Routing based on an incomplete LSP database can be dangerous and can cause widespread misrouting and/or routing loops.

IS-IS solves the storage problem by requiring a router that cannot store its LSP database to set an overload flag in its own LSP. Other routers then treat that router as an end system and route *to* that router but not *through* that router. Thus, the overloaded router is available through network management. If the router has not needed to refuse an LSP from a neighbor for a period of a minute (or as configured by network management), the router will clear the flag in its LSP. Thus, if the problem is temporary, the network will recover without human intervention. An important feature of this solution is that changing the flag does not change the size of the LSP database and hence does not lead to oscillation of the overloaded condition.

Limiting the Size of Routing Control Packets

Some IS-IS packets (specifically, LSPs and CSNPs) may become too large to be transmitted as single packets. Consequently, the packets may split into several packets for transmission.

An LSP can become very large if a router has many neighbors. However, this situation is rarely an issue, except for the pseudonode LSP for a LAN. The IS-IS protocol avoids such large LSPs, which would need to be fragmented for transmission across each link and then reassembled at each router. The protocol has the LSP source break the LSP into individual fragments, each with its own unique ID and sequence number. The ID of the LSP is no longer simply the ID of the router issuing the LSP but has an additional octet appended to the router's (or pseudonode's) ID indicating the fragment number. Each fragment is independently flooded to the other routers. Only in the route computation is any connection made between the fragments of a router's LSP.

A CSNP can become large as well, since it includes the range of source addresses of LSPs to which it refers. If the range indicates x through y , then all LSPs with source IDs between x and y will be included and only those LSPs. Absence of an LSP that lies within the range implies that the issuing router has no knowledge of that LSP. Therefore, the IS-IS protocol can take action based on a CSNP fragment without waiting for all fragments. If a CSNP fragment is lost, then a lost LSP in that fragment's source address range might not be detected until the next time a CSNP fragment listing the ID of the lost LSP is transmitted.

Support of Multiple Protocols with IS-IS

Extending the IS-IS protocol to support multiple protocol suites is relatively straightforward. The OSI version of the IS-IS protocol supports routing for OSI CLNP, which also implies support for DECnet Phase V (since Phase V user data packets are identical to CLNP packets at the network layer). DECnet Phase V routing extends IS-IS to allow support for DECnet Phase V and for Phase IV-Phase V interoperability. Also, Digital worked on the Internet Engineering Task Force (IETF) to define the extension to IS-IS for support of IP.¹⁵

To understand how the OSI IS-IS protocol can be extended to support multiple protocol suites, consider what the IS-IS protocol provides. For example, consider a level 1 router within an area. The IS-IS routing protocol allows this router to know the identity and up/down status of the other routers and links in the area and which routers in the area are level 2 routers. IS-IS calculates routes to all other routers in the area. IS-IS also provides a number of important background functions, such as allowing information to be reliably broadcast between the routers in the area and allowing up/down status to be periodically checked. In addition, IS-IS allows each router to know which OSI addresses are reachable by means of each other router. (At level 1, the router would list the NSAPs of all its end-system neighbors; at level 2, the router would list all the areas and address prefixes it can reach.) IS-IS therefore already has most of the information needed to calculate routes for additional routing protocols.

To add routing support for another protocol suite such as IP, the IS-IS protocol is updated to announce the addresses that are reachable by means of that protocol suite. For example, to add IP support to IS-IS, a new field is defined in the LSPs to

announce IP addresses, expressed in ordered pairs of the form (IP address, subnet mask). This allows IP addresses and OSI (i.e., DECnet Phase V) addresses to be assigned independently, while still allowing most of the overhead functions required by a routing protocol, such as checking link status and propagating the information, to be performed only once for all supported protocol suites.

If all routers support a particular protocol, the data packets for that protocol can be transmitted in native mode, i.e., no additional header is required. If some routers do not support a particular protocol, then the packet must be encapsulated in a network layer header for a network layer protocol that all the IS-IS routers do support. In DECnet Phase V, all the routers support both IP and CLNP, so these two protocols are transmitted in native mode. However, if support for another protocol is added, for instance AppleTalk support, then the routers that have AppleTalk neighbors need to be able to parse AppleTalk packets. However, other routers will not need to be modified. To facilitate knowing when to encapsulate, IS-IS routers announce which protocols they support in their IS-IS packets. Also, routers that support the AppleTalk protocol and have AppleTalk neighbors list in their LSPs that they can reach certain AppleTalk destinations.

The IS-IS packets are encoded such that a router can ignore information pertaining to protocol suites that the router does not support but can correctly interpret the rest of the IS-IS packet. Assume that R1 and R2 are the only two routers in an area that support the AppleTalk protocol. R1 and R2 therefore announce in their LSPs which AppleTalk destinations they can reach. R1 and R2 use a format for including AppleTalk information in IS-IS LSPs that other routers in the same area can forward but will otherwise ignore. Assume R2 receives an AppleTalk packet for forwarding with destination D3, reachable through R1. Then R2 encapsulates the packet as data inside a CLNP (or IP) packet with destination R1. When R1 receives the packet, it removes the CLNP header and forwards the packet to D3. If R1 and R2 are adjacent, or if all the routers along the path from R2 to R1 support the AppleTalk protocol, then encapsulation of AppleTalk packets inside CLNP packets would not be necessary. Thus, encapsulation occurs automatically only when needed for transmission through routers that do not support the protocol of the data packet to be forwarded.

Using one integrated routing protocol to route packets from multiple protocol suites has significant advantages over using a separate routing protocol for each suite. Probably the most important advantage is that only one routing protocol needs to be managed and configured. A single coordinated routing protocol can respond to network problems, such as link failures, in an efficient manner, improves bandwidth utilization, and minimizes the CPU and memory requirements in routers. Also, integrated routing allows automatic encapsulation and eliminates the need for manual configuration of where and when to encapsulate.

Summary

IS-IS is a powerful and robust routing protocol. Many aspects are innovative and have been copied by other routing protocols. When looked at as a whole, the algorithms may appear complex, but when examined individually, the designated router election, the LSP propagation, and the LSP database overload procedure, for example, are all quite simple. IS-IS provides efficient routing for a variety of protocol suites, currently including DECnet Phase IV, CLNP/DECnet Phase V, and IP.

References

1. *Information Processing Systems, Data Communications: Protocol for Providing the Connectionless-Mode Network Service*, ISO 8473 (Geneva: International Organization for Standardization, 1988).
2. J. Postel, "Internet Protocol," Internet Engineering Task Force RFC 791 (September 1981).
3. *Information Processing Systems, Telecommunications and Information Exchange between Systems: End System to Intermediate System Routing Exchange Protocol for Use in Conjunction with the Protocol for Providing the Connectionless-Mode Network Service (ISO 8473)*, ISO 9542 (Geneva: International Organization for Standardization, 1988).
4. D. Plummer, "Ethernet Address Resolution Protocol," Internet Engineering Task Force RFC 826 (November 1982).
5. J. Postel, "Internet Control Message Protocol," Internet Engineering Task Force RFC 792 (September 1981).

6. *Information Technology, Telecommunications and Information Exchange between Systems: Intermediate System to Intermediate System Intra-Domain Routing Exchange Protocol for Use in Conjunction with the Protocol for Providing the Connectionless-Mode Network Service (ISO 8473)*, ISO/IEC 10589 (Geneva: International Organization for Standardization/International Electrotechnical Commission, 1992).
7. G. Sidhu, R. Andrews, and A. Oppenheimer, *Inside AppleTalk, Second Edition* (Reading, MA: Addison-Wesley, 1990).
8. C. Hedrick, "Routing Information Protocol," Internet Engineering Task Force RFC 1058 (June 1988).
9. J. McQuillan, I. Richer, and E. Rosen, "ARPANET Routing Algorithm Improvements, First Semiannual Technical Report," *BBN Report 3803* (April 1978).
10. E. Rosen et al., "ARPANET Routing Algorithm Improvements, Volume 1," BBN Report 4473 (August 1980).
11. R. Perlman, *Interconnections: Bridges and Routers* (Reading, MA: Addison-Wesley, 1992).
12. E. Dijkstra, "A Note on Two Problems in Connection with Graphs," *Numerical Mathematics*, vol. 1 (1959): 269-271.
13. R. Callon, "Use of OSI IS-IS for Routing in TCP/IP and Dual Environments," Internet Engineering Task Force RFC 1195 (December 1990).

Digital's Multiprotocol Routing Software Design

The implementation of Digital's multiprotocol routing strategy required addressing various technical design issues, principally the stability of the distributed routing algorithms, network management, performance, and interactions between routing and bridging. Developers of Digital's DEC WANrouter and DECNIS products enhanced real-time kernel software, implemented performance-centered protocol software, and used high-coverage, high-quality testing and simulation methods to solve problems related to these issues. In particular, a packet management strategy ensured that queuing requirements were met to guarantee the stability of the routing algorithms. Also, network management costs were minimized by down-line loading software, using a menu-driven configuration program, and careful monitoring. Router performance was optimized by maximizing the packet forwarding rate while minimizing the transit delay.

Digital's implementation of multiprotocol routing software enables internetworking across complex topologies including local and wide area networks (LANs and WANs) and dial-up networks. Evolving from Digital's successful tradition in DECnet Phase IV networks, the implementation of multiprotocol routing currently supports numerous protocol and packet types including

- DECnet Phase IV
- Transmission control protocol/internet protocol (TCP/IP)
- Novell NetWare internetwork packet exchange (IPX) protocol
- AppleTalk protocol suite
- OSI CLNS, the open systems interconnection protocol for providing the connectionless-mode network service
- X.25, the packet switching standard specified by the Comité Consultatif Internationale de Télégraphique et Téléphonique (CCITT)

Additional extensions for Digital's DECnet Phase V and ADVANTAGE-NETWORKS architecture requirements are also supported by Digital's multiprotocol routers.^{1,2} Many of these routers incorporate bridging technology, thus providing integrated bridging

routers. This paper describes the most significant technical problems encountered and the solutions implemented when many internetworking operations are integrated into Digital's multiprotocol router system designs.

Digital's Router Product Overview

Digital's multiprotocol router products comprise two types: (1) access routers, which allow access to WAN services from branch offices for large LAN and WAN integration networks, and (2) backbone routers, which provide high-speed packet switching services for the network backbone of multiple types of high-speed media. Backbone sites offer a backbone network that often consolidates high-speed WAN lines, e.g., T1, T3, and SMDS. For high-speed local sites, backbone routers provide high-speed switching for many LAN ports and types, i.e., Ethernet, fiber distributed data interface (FDDI), and token ring. This section briefly discusses some of Digital's access routers—the DEC WANrouter 500, DEC WANrouter 250, and DEC WANrouter 90 products—and backbone routers—the DECNIS 500 and DECNIS 600 products.

The DEC WANrouter 500 is one of Digital's access routers and has been available in the marketplace since 1986. Originally a DECnet Phase IV-only router, this router has been upgraded and now

offers multiprotocol routing that includes DECnet Phase IV, TCP/IP, and OSI. Additional support exists in this access router for common WAN services such as X.25 and frame relay. The DEC WANrouter 500 is a fixed-port configuration router offering one T1 WAN port and one Ethernet LAN port. This configuration permits branch office LANs to interconnect to backbone routers over relatively high-speed T1 lines. The DEC WANrouter 500 has an important place in router industry history as it was the first router ever to support the integrated intermediate system-to-intermediate system (Integrated IS-IS) routing algorithm.³

The DEC WANrouter 250, another of Digital's access routers, is significant due to its high density of WAN ports and its support for asynchronous WAN data link protocols. These two major features combine with the multiprotocol routing software to provide a router for the newly emerging computer networking needs of mobile computers. The increasing use of personal computers, including mobile laptop computers, has led to the development of new techniques for networking such remote computers. The DEC WANrouter 250 provides eight WAN ports with dial-in access for the internetworking of such remote and mobile computers.

The introduction of LAN hub technology has produced a need for new small router products for these platforms. Digital's DEChub 90 Ethernet backplane product set includes the DEC WANrouter 90 access router shown in Figure 1. One feature of the DEChub 90 technology is that this router can be configured to reside within the hub itself or as a standalone module. In addition, this router is completely self-contained and extremely small (i.e., similar in size to a VHS videocassette). Many WAN access services, such as X.25 network access,

are provided for the DEChub 90 with the DEC WANrouter 90 router.

The DECNIS 500 and DECNIS 600 (see Figure 2) bridging and routing products are Digital's highest performing and most flexible platforms. These backbone routing systems offer the power and interfaces necessary to meet the bridging and routing requirements of complex, high-speed networks, e.g., Ethernet, FDDI, T1/E1, and T3/SMDS.⁴

Router Software Development Methods

Software development for routing systems requires real-time kernel software, performance-centered protocol software development implementation, and high-coverage, high-quality testing and simulation methods. This section briefly describes some key techniques used in these development areas for the DEC WANrouter and DECNIS engineering programs.

Kernel Software

Digital has developed and refined different kernels with common interfaces to address the real-time software design environments required for their routers. A common router interface model has

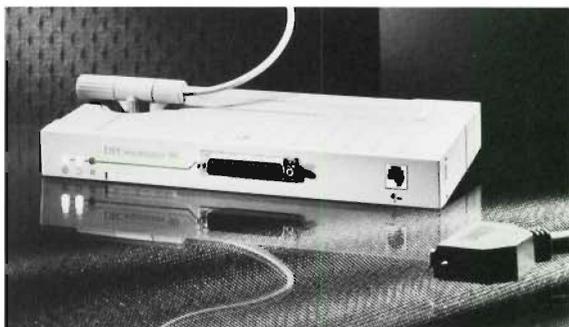


Figure 1 DEC WANrouter 90 Access Router



Figure 2 DECNIS 600 Backbone Router

permitted different kernels to be turned to specific platforms as required. In some cases, a common portable kernel was developed that permitted quick retargeting of the total router software in support of short time-to-market development needs.

Software Implementation

The following techniques were used in the development of the DEC WANrouter and DECNIS router software:

1. Implementing software directly from proprietary or standards-based architecture specifications
2. Licensing software from suppliers, e.g., external corporate software providers and government-funded university software projects
3. Importation of software from other implementations, i.e., host sources such as the ULTRIX, Open Software Foundation(OSF), and OpenVMS systems

Digital has developed special-purpose, high-performance implementations of the Integrated IS-IS routing protocol. In addition, specific software kernels provide control and extensions for the special features required. Engineers enhanced the real-time software kernels with software interfaces commonly found in public domain software (e.g., the Berkeley Software Development [BSD] UNIX socket model and system services). The inclusion of such interfaces has accelerated the addition of new software from external sources.

Common router software has been developed for use across Digital's many internetworking platforms. The majority of this routing software, which is independent of the underlying hardware, has been developed to support the evolving standards of portability. For each platform, the performance-intensive and hardware-specific code have been customized to maximize the design center for each instance of a router product architecture.

Router Software Design Issues

Many technical problems had to be resolved when building Digital's multiprotocol routers. The following sections describe the most significant issues and how they were addressed in the DECNIS 600 backbone router, as an example of router software design. These issues were

1. Stability of the distributed routing algorithms
2. Network management

3. Performance

4. Interactions between routing and bridging

Memory size and usage and congestion control are also key issues. However, this paper does not describe them in depth. Briefly, the amount of memory available is a major constraint on any router implementer. Usually, memory is largely consumed by code and by the databases the router must maintain to calculate the best route. In the case of routers that also perform connection-oriented functions (e.g., X.25 gateways and terminal servers), significant amounts of memory may be taken up by the per-connection state and counter information.

Since it is essential for routers in the network to agree on the best route to a destination, all such routers must be able to handle the route database for that network. Digital's router designs have an automatic shutdown mechanism that takes effect should a router run out of memory in which to store routing information. This mechanism prevents routing loops.

To control OSI congestion, the router must determine whether or not a packet experienced congestion by calculating the average transmission queue length over time. This calculation must be performed in an efficient real-time manner. Thus, for the DEC WANrouter and DECNIS products, Digital designed and implemented algorithms specific to the particular queue structures and hardware design.

Stability of the Distributed Routing Algorithms

Distributed routing algorithm stability was the most important issue considered in the design of Digital's router systems. A system design must guarantee successful results in support of routing control protocols even when the router is operating under a high load.

Whatever protocol is used, dynamic routing requires that all nodes that make decisions on how to forward data should agree on the correct path. Otherwise, data packets will be discarded (e.g., if sent to a node that does not know how to reach the destination) or may loop (e.g., if two routers each believe the other is the correct next node on the path to the ultimate destination, then the packet will loop between the two routers).

If network configurations never changed, and lines and routers never got overloaded, then guaranteeing successful results would be easy.

Unfortunately, actual networks are complex. In practice, for each protocol, the correct path agreement is reached using an algorithm distributed between multiple independent routers and operating on ever-changing data.

The distributed algorithm must converge rapidly so that when network conditions change, the new route is agreed upon quickly. However, the algorithm must also be stable. When changes occur at a fast rate or when the algorithm is trying to complete or has just completed, the algorithm must still converge to a consistent state between all the routers involved. In this way, the network remains useful. In addition, while the network is changing, a router or a line may suddenly be presented with an excessive load of packets to forward (e.g., because a routing loop occurred transiently). This situation must not be allowed to disturb the stability of the routing algorithm.

The stability of a well-designed routing algorithm is directly related to how well the algorithm meets the following main requirements:

- **Line speed.** The effective speed of lines between routers (allowing for error correction by the data link protocol or the modem) must be high enough to allow the routers to rapidly exchange routing control information. The maximum bandwidth required for routing control traffic can be calculated from the size of the network.⁵ In a network of 4,000 end nodes, 100 level 1 routers, and 400 level 2 routers, approximately one Link State Packet (LSP) will be received every second. This LSP may contain 1,500 bytes, which would use a line bandwidth of 12,000 bits per second. This aspect of stability is under the control of the network designer; line speeds and network size must be continuously monitored and related.
- **Processing power.** The router CPU must be fast enough to forward routing updates to neighboring routers with minimum delay and must be able to recalculate the forwarding database quickly. Of course, this requirement relates only to that portion of the CPU time available for routing functions. A router that is also doing another job (e.g., acting as a file server) will have less CPU power available, unless routing is given priority over the other functions. Consequently, most networks now use dedicated routers instead of attempting to have routing tasks share the CPU with other functions.

- **Queuing.** The most important stability factor is to make sure that the systems are self-stabilizing. As the problem gets worse, progress to the solution should not become slower. For example, as the network configuration changes more rapidly, the calculation of the best route must not get slower. To meet this requirement, the routers must be careful about queuing data and routing control messages internally so that excessive or unusual data forwarding loads do not affect the processing of routing control messages. Otherwise, when a network problem overloads a router, the routing algorithm may never converge to fix the problem.

Figure 3 illustrates a case where an incorrectly designed router (one that gives priority to data forwarding over routing control message reception and processing) could cause a permanent routing loop and thus isolate a portion of the network. In this example, node A is sending a large amount of data to node F over the high-speed T1 line. The lower-speed (64 kilobit-per-second [kb/s]) line is available as a backup line. Because the backup line runs at only 64 kb/s, node C need only be a low-power router. For example, a router rated at 128 packets per second would be sufficient because a fully saturated full-duplex 64-kb/s line with 128-byte packets handles 128 packets per second.

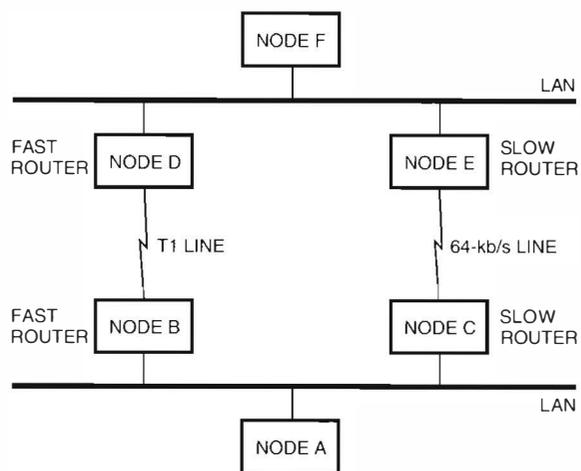


Figure 3 Network Instability

Consider what happens if the T1 line fails. Router B notices immediately and begins to forward data to router C. Initially, however, router C still believes the best route to node F is over the T1 line and so forwards the data back to router B. B resends the data to C and so on; a routing loop has been created. This problem is common during routing transitions. The loop will be broken as soon as router C runs the decision process and updates its routing tables. However, if router C is incorrectly designed and gives priority to forwarding data, then the unexpectedly large amount of data will "swamp" the router and prevent it from running the decision process.

In addition, since router C is only a low-speed router, it will be forced to discard many data packets. Eventually, the transport connections between node A and node F will fail, because packets are not being delivered (presumably causing the applications to fail). This situation will reduce the number of packets being introduced into the loop. However, each packet can go around the loop many times, thus generating a high load. In this example, if nodes are set up such that a packet can travel the loop 64 times (a common value), then introducing only two packets into the loop per second will continue to swamp router C. Any node on the LAN might be sending those packets to discover when access to the remote LAN is restored. The effect is a long-lived routing loop that isolates the whole LAN, even though there was supposed to be a backup link available.

- Memory usage. Activities less important than routing should not consume the memory necessary for routing control processes to carry out their function. Even in a dedicated router, some lesser activities will be in progress. For example, network management and accounting are important activities, but they are not as critical as maintaining network stability—without a stable network, network management and accounting will fail. Therefore, other activities should not starve the routing control processes of memory. Consequently, traditional memory pools are not an appropriate way to allocate critical memory within the router; routing memory usage must be preallocated.

The remainder of this section describes the impact of the requirements on processing power, queuing, and memory allocation on the design of the DEC WANrouter and DECNIS products.

Requirements on Processing Power

The Digital Network Architecture (DNA) routing architecture requires that routing updates be propagated within 1 second of arriving and that the forwarding database calculation take no more than 5 seconds.⁵ The forwarding database calculation is CPU-intensive, but the time is proportional to the number of links reported in LSPs. To meet the DNA requirement, various measurements were made for each product to determine the number of links the decision process could handle per second. This information indicates, for each product, the maximum number of links allowed in the network. Note that this number does not directly limit the number of nodes permitted in the network; a large network with an efficient connection strategy may have fewer links than a small network in which every node is connected directly to every other.

The update process latency requirement means that the CPU time must be fairly allocated between the decision process and the update process. If the update process was required to wait until the decision process had completed, then the delays on forwarding LSPs would be too large (i.e., 6 seconds).

We considered three possible solutions.

1. Process priorities. Give the update process a strictly higher priority than the decision process so that the database can be updated as required. The main issues to resolve are synchronizing access to the shared LSP database and allowing the decision process to complete, if a faulty router generates LSPs at an excessive rate.
2. Timeslicing. As in a traditional timesharing system, allow both processes to run simultaneously, thus sharing the CPU. This solution also requires synchronizing access to the LSP database.
3. Voluntary preemption. The decision process periodically checks to see if the update process is required and, if so, dispatches to it. This check can occur at time intervals frequent enough to meet the latency requirements and at times convenient to the decision process so that no synchronization problems occur.

To avoid the synchronization problems, Digital's DECNIS 600 software developers chose the third solution for two reasons.

1. Synchronization issues often cause problems that are serious and difficult to debug in complex systems. By avoiding these issues entirely, we simplified the software and increased its reliability.
2. The addition of synchronization mechanisms for parallel tasks can decrease the performance of the total system (for example by causing excessive rescheduling operations). Using voluntary preemption allowed a very efficient solution that still met the architectural requirements.

Requirements on Queuing

Queuing constraints ensure that high loads do not cause routing control information to be discarded. Initially, separating the data for forwarding from routing control messages might appear to be the logical solution to preserving routing control information. However, this solution works only if the router can process all the routing control messages without getting behind.

Many practical routers, including the DEC WANrouter products, do not have a CPU that is fast enough to guarantee such processing performance. Digital's routers can guarantee to meet the timing requirements on the decision and update processes (even under worst-case loads), but if that load is combined with a flood of End-node Hello messages, Router Hello messages, and other control traffic, then some of those messages have to be discarded or queued for later processing. Since there might be 1,000 or more nodes on the LAN, the worst situation would be if all these nodes were to decide to send Hello messages at the same time.

Careful software design means that the routers can meet the network stability requirements and still not lose connectivity to end nodes on the LAN. For the DEC WANrouter software, Digital designed and implemented a packet management policy that differentiates between routing packet types to meet their respective processing requirements for network stability. The following list summarizes the classes of packet types:

- Data
- End-node Hello messages
- Router Hello messages

- Link State Packets and their acknowledgments, Sequence Number Packets (SNPs) and Complete Sequence Number Packets (CSNPs)

The parameters controlling the minimum and maximum numbers of packets to be used for each differentiated type are carefully calculated based on their architected behavior and the network configurations supported by each product. For example, a router's architected design center for supporting a given maximum number of adjacent routers on an attached LAN will affect the policy selected for managing the Router Hello message queues and packet buffers. Such mechanisms are implemented to guarantee that, for network stability, forwarding performance, and network convergence, the minimum levels of forward progress per packet type are met.

This packet management policy uses both buffer pools and queuing to implement the required policies. Inbound traffic is placed on queues that are serviced using variants of round-robin algorithms. These algorithms give different weightings to each queue to ensure that progress is made for every packet type, although at different rates.⁶ For example, for every data packet processed, the router may process 5 ISPs, 5 End-node Hellos, and 10 Router Hellos. The actual weightings used are selected when the software is designed and depend on the performance characteristics and expected network configuration of each product.

Some alternatives that were considered are

- Alternative buffer pools. A completely separate pool can be used for each of the different types of packets. The disadvantage is that in small configurations or ones that are not under heavy stress, the pool of buffers available for forwarding is limited unnecessarily.
- Strict priorities. Setting strict priorities for processing different types of routing control messages is undesirable, because a flood of one type of routing control message could cause another type to be ignored for a long time. In such a case, it is better to process some of each type of message than to give one type absolute priority.

In the DECNIS routers, several queues exist at the boundaries between the different DECNIS processors.⁴ Digital designed a mechanism for these queues similar to that described for the DEC WANrouter products. When the network interface

cards, i.e., linecards, receive a packet destined to be passed to the management processor card (MPC), they analyze the packet and tell the MPC whether it is data, routing control, bridging control, or system control (which includes linecard responses to commands from the MPC). Thus, queues analogous to those described for the DEC WANrouters are used at all the interfaces within the system. For example, the assistance processor on the MPC recognizes the different types of messages and queues them on separate internal queues.

Requirements on Memory Allocation

Routers must have sufficient buffer space to handle the routing control messages. Consequently, all of Digital's router products guarantee this memory allocation. To preserve these buffers, the DECNIS MPC implements buffer swapping between layers, as illustrated in Figure 4. The data link layer must never be starved of buffers; otherwise, packets regarded as important by routing may be discarded without ever being received. To ensure that an adequate number of buffers is available to the data link layer, the MPC gives the data link a certain number of buffers and maintains that number. Every time a buffer is passed from the data link layer to the routing layer, another buffer is swapped back in return. If routing currently has no free buffers, it selects a less important packet to discard (freeing up the buffer containing the packet). In this way, the data link layer always has buffers available.

In the DECNIS linecard buffers, the arrangements are similar to those just described, but the details differ. The linecards and the MPC perform buffer swapping among themselves.⁴

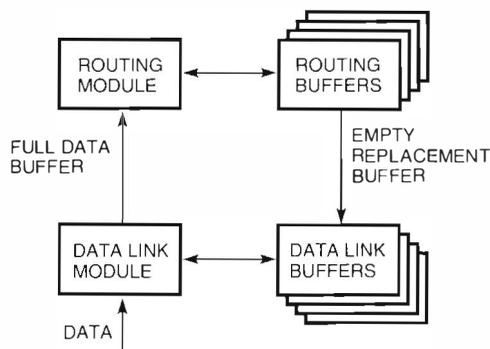


Figure 4 Buffer Swapping between Routing Module and Data Link Module

Network Management

Some of the highest costs involved in running a network are those related to obtaining and maintaining trained and experienced network managers and operators. Minimizing these costs requires routers that can be easily and efficiently managed. The major network management issues are

- Installation/loading. How are software updates distributed and installed? How long does the router take to load after a power failure?
- Configuration. How is the software told about changes to the lines or the network parameters? Does the network require a reboot to change information?
- Monitoring. How does the manager get immediate reports of problems and unexpected changes, and long-term reports of traffic patterns and usage for network planning?
- Control. How can the manager shut down a line or even a whole router?
- Problem solving. What tools are available to detect the problem and then to investigate and correct the problem?

In all networks, though, a remote management capability is essential. Skilled network management staff may not be available at all sites (e.g., a small branch office). In fact, some sites may have no staff at all (e.g., a lights-out computing center).

Installation and Loading

All DEC WANrouter and DECNIS products update their software by down-line loading new software over the network. In the case of the DECNIS, the software is stored in nonvolatile memory and so does not need to be reloaded on each boot. However, the DEC WANrouter products down-line load the software each time they are booted.

Digital considered two other alternatives.

- Read-only memory (ROM). This means of distribution has the disadvantage of being expensive to modify and difficult to replace remotely.
- Floppy disk or other interface on the router. This mechanism increases cost and reduces reliability. Loading from a floppy disk may also be slower than loading over a network. Again, remote updating may not be possible, and physical

security issues (e.g., preventing unauthorized users from supplying uncontrolled router software) may be introduced.

For the DECNIS product, Digital chose to use nonvolatile memory, e.g., flash random-access memory (RAM), for fast and reliable loading combined with backup down-line load operation when software updates are required. The down-line load can be from a DECnet system using the maintenance operations protocol (MOP) or from a TCP/IP host using the boot protocol (BOOTP) and the trivial file transfer protocol (TFTP).¹ The down-line load provides an easy way to update software when required; the software can be installed on a load host using any of the standard software distribution mechanisms (e.g., CD-ROM, magnetic tape, or the network).

Configuration

Configuring a router is notoriously difficult. Therefore, Digital developed a tool to assist the network manager with configuration. Each of the DEC WANrouter and DECNIS products comes with a configuration program. This menu-driven program leads the network manager through a series of forms to define the information needed to configure the router or to modify an existing configuration. On-line help is available, and steps may be retraced. Consequently, the network manager has no need to learn the network control language (NCL).

Digital used formal human factors testing during the design and development of the configurators to ensure that these tools were of high quality. Human interface testing continued through the router's customer field trials and provided additional feedback on our configurators' ease of use.

One thing that Digital did not originally anticipate is that users now tend to see the configurators as the user interface for the product. The configurator is often a customer's main means of interacting with the router and thus is an essential part of the product. Once people have used the configurator, they no longer regard it as an optional feature.

Monitoring

Digital's routers are fully manageable using Phase V network management. They all respond to NCL commands and can be managed using the DECMcc program, Digital's Enterprise Management Architecture (EMA)-compliant director. Therefore,

DECMcc added-value functional modules are available for performance analysis and historical data recording. The DECMcc design enables these functions to work without changing the router design.

Many users, however, are now investing in management stations that use the simple network management protocol (SNMP). Thus, for monitoring purposes, Digital already implements basic read-only SNMP management, which is being enhanced over time to add more information.

Control

Whether managed by the NCL or the DECMcc director, access is controlled using passwords. In addition, Digital is focused on offering full SNMP management for the router products. As well as providing the standard public management information, Digital is defining private management information to allow unique features of the routers to be controlled. We designed the internal management interfaces of the routers to allow us to write modules that are manageable from both the SNMP and the common management information protocol (CMIP), with minimal effort and duplication.

Problem Solving

One of the most time-consuming, and hence expensive, parts of a network manager's job is problem solving. Fortunately, many of the tools and techniques used for this task were required for debugging and testing router implementations and thus already exist.

Building initially on debugging and testing experience, and later on discussions with users, Digital has produced problem-solving guides for each DEC WANrouter and DECNIS product. These guides take the user through a step-by-step description of how to isolate and fix a problem. We have conducted human factors testing on these guides and have investigated different modes of making this information available. The DECNIS guide is currently available in hard copy and also in an on-line Bookreader form that allows moving through the flow to be automated using hot spots. Digital is currently evaluating Hypertext technology to further improve the usability. One main tool for problem solving is the common trace facility (CTF), a software tool that causes the router to record and display packets that are sent and received. Analysis routines automatically format the packets. Having

the CTF is comparable to having a built-in line or LAN analyzer. The CTF is the main diagnostic tool used by Digital's service engineers when investigating a problem and also by the development engineers when debugging software.

Digital's routers also include diagnostic and maintenance facilities, which include loopback testing over all interfaces and low-level, limited, remote management directly at the data link layer. The remote management capabilities allow monitoring of counters from an adjacent node and also allow an adjacent node to force a reboot if a suitable password is supplied. This latter operation is referred to as a MOP boot (previously known as a MOP trigger in DECnet Phase IV).¹

A MOP boot command may be the final attempt by a network manager to fix a problem with a router without having to go physically on site. For that reason, the command must be recognized and acted upon regardless of what else may be happening in the router. In the DECNIS routers, the MOP boot command is recognized by the linecards. In the DEC WANrouter, the MOP boot command is specially actioned by the lower layers of the software to make sure it is honored even if the higher layers have failed in some way or if the system is under an enormous load.

We also support the "TCP/IP ping" utility (more formally, ICMP Echo) and the similar "OSI ping" utility. These tools are commonly used for diagnosing reachability problems.

Router Performance

Today's large-scale computer data networks rely on bridge router components for the networks' total level of performance and quality of service. As such, data network designers and network managers must be knowledgeable about their chosen router platform's performance characteristics. This section of the paper discusses the performance aspects of Digital's routers.

Performance Metrics

In support of developing common metrics across the internetworking router industry, the Internet Engineering Task Force (IETF) has set up a Benchmarking Methodology Working Group, which has developed definitions for router performance.⁷ Three key metrics defined by this group provide the background for our discussion of Digital's router software design.

- Throughput—the maximum (forwarding) rate at which none of the offered frames (packets) are dropped by the device (i.e., packets per second)
- Frame loss rate—the percent of frames (packets) that should have been forwarded by the network device (router) while under a constant load but which were not forwarded due to lack of resources (i.e., percent packets lost)
- Latency—for store-and-forward devices (i.e., routers), the time interval beginning when the last bit of the input frame reaches the input port and ending when the first bit of the output frame is seen on the output port (i.e., units of time)

In the design of Digital's router software and systems, a balance has been targeted with maximizing the packet throughput forwarding rates while minimizing the packet latency. Some vendors mistakenly compare loss-free throughput rates with forwarding rates that have high loss rates. Such comparisons must be studied carefully, because they do not compare route performance measures of equal impact to the total network. To reiterate, the throughput forwarding rate occurs only at the point when the frame loss rate is zero percent. Digital's routers target throughput designs which, as much as possible, run at "wire speed" with zero frame loss rates. Regardless of the throughput value quoted, router comparison should reference common packet loss rates because network applications need to retransmit any packets that are lost by the routers.

In general, the throughput, loss-free forwarding rate is the optimum value for discussions of router forwarding performance. The other critical value is the stability of the router under heavy overload. A "receive livelock" condition occurs when the offered load, i.e., input packets received for subsequent forwarding by a given router, reaches the point where the delivered throughput, i.e., packets actually forwarded, decreases to zero.^{8,9} Real-time systems, such as routers, have the potential to livelock under traffic loads above their throughput peaks. However, it is extremely important that routing implementations avoid such responses to post-throughput saturation. In the case of Digital's routers, in all architectures and products, the routers do not livelock but remain stable even when the applied input load to a router exceeds the peak throughput forwarding packet rate. This key

performance measure of router devices remains an underlying design characteristic of all Digital DECNIS and DEC WANrouter network devices.

Packet Throughput/Forwarding Rate

Digital's routing platforms offer a range of throughput measures. For each platform, the throughput is the most often quoted value used to characterize the router's aggregate capabilities. In the case of the DECNIS 600, an aggregate throughput of 80,000 packets per second is offered.¹⁰ In smaller routers, the WAN line interface rates (i.e., 64 kb/s and T1) are often the limiting factor for the aggregate throughput. The software in all cases is optimized for the given router platforms mix of WAN and LAN interfaces.

Since the forwarding rate is the most important performance metric for a router, Digital carefully optimized the designs of its multiprotocol routers to allow data forwarding to occur as fast as possible. On the DEC WANrouter products, we handle all the forwarding on a central CPU with little hardware assistance. In the DECNIS products, forwarding and filtering operations are handled by linecards. A hardware assist for the performance-critical forwarding function's address lookup is used on DECNIS routers in support of requirements for very high-speed packet switching.⁴ On each linecard, a streamlined software kernel has been developed along with all the required software. The linecard software kernel and modules were carefully constructed to have the minimum number of instructions and the lowest number of execution cycles necessary to perform the high-speed forwarding and filtering operations. On the DECNIS MPC, the software kernel is also fully capable of the routing forwarding operations. However, this kernel is mainly required to provide the software processing for the remaining non-performance-intensive operations of the router's software (i.e., the processing of updates to the router topology database and the network management commands/received packets). This partitioning of processing of received packets in the DECNIS router system permits such routers, and the networks that they comprise, to remain highly stable when traffic overloads occur.

For the DEC WANrouter software, the forwarding operation has no hardware assist. Software lookup assist algorithms have been researched and implemented to help meet the performance-intensive requirement. As in the microcoded DECNIS linecard

adapter software, the software is highly tuned for performance. To minimize the additional maintenance overhead associated with highly tuned software, the amount of such code is kept to a minimum. The DEC WANrouter software design is an example of how Digital carefully balanced product performance requirements and product development and maintenance costs to meet the required price/performance goals for its router product family.

Packet Latency (Transit Delay)

The next most frequently specified performance requirement is packet latency or packet transit delay. For bridge/router devices, this measurement clearly depends on software and hardware timings. However, the definition of latency utilized corresponds directly to the chosen system's design.

The previously quoted IETF definition for store-and-forward devices can be further refined to accommodate differing device designs. The IETF working group clarifies the difference between a "store-and-forward device" and a "bit-forwarding device" internal design model for a router. The latter design model is often referred to as a "cut-through" design and requires a different definition than previously listed for store-and-forward devices. The definition of latency used for this cut-through model is the time interval starting when the end of the first bit of the input frame reaches the input port and ending when the start of the first bit of the output frame is seen on the output port.⁷

The issue that distinguishes the two models is whether or not processing starts prior to the packet being completely received. However, another key point is whether or not the packet received can be sent out for transmission prior to complete reception. When reception, forwarding, and transmission can occur in parallel, the design is referred to as cut-through. For Digital's router designs, the DECNIS does process reception and forwarding in parallel prior to a packet being completely received. However, the DECNIS does not start transmission until a packet is completely received. Thus, the DECNIS latency model uses the original store-and-forward definition of the IETF.

In the case of the DEC WANrouter software, the model and definition used is again store-and-forward. The factors that control the packet latency in the DEC WANrouter design are as follows:

1. Receiving the packet. The packet must be completely received.
2. Performing the forwarding operation. This factor includes packet verification, analyzing the packet, performing any required address lookup, performing any required packet modifications, and queuing the packet for transmission on the destination interface.
3. Congestion queuing. If the destination interface is not idle, the packet will have to be queued before transmission. Some transit delay measurements use only uncongested media interfaces connected to the router. However, latency measurements must be made to measure the potential latency delays due to congestion at the router output interface. The packet latency due to queue occupation delays is also included here. Congestion avoidance algorithms have been implemented to minimize this congestion delay.
4. Transmitting the packet. This factor is usually dominated by the time taken to clock the bits of the packet out of the interface but also includes media access times, i.e., delays due to another node already using a common connection.

We now examine how the DEC WANrouter and DECNIS routers separately minimize the transit delay.

The DEC WANrouters minimize the packet reception and transmission portions by allowing hardware to perform these functions using direct memory access (DMA). Because these systems have only a single processor, the forwarding delay is minimized by the same fast-path optimizations used to improve the forwarding rate.

On the other hand, the optimizations for the DECNIS routers are slightly different for the various linecards. The DEC WANcontroller 622 card has no DMA, and the linecard on-board processor is involved in receiving each byte of the packet. We parse the header as soon as there is enough information to do so. For example, the data link packet type field is decoded before the network address bytes have been received, and the network address lookup is initiated as soon as the address has been received (i.e., before the data has been received). The address lookup is then performed by the address recognition engine hardware without further involvement from the software.

The DEC WANcontroller 618 card and the DEC LANcontroller 601 and 602 cards all receive packets

one segment at a time. Internally, these cards use small fixed-size buffers that are linked together as necessary to store a whole packet. Again, they perform the analysis and forwarding lookup as soon as the data is available (i.e., when the first segment is received).

Thus, for a large packet, the entire forwarding decision will have been made before the last byte has been received. However, note that until the last byte has been received, it is not known whether the cyclic redundancy check (CRC) is correct or the packet has been corrupted. So the packet is not actually passed to the destination linecard until that check has been completed. As discussed before, this design is still store-and-forward, rather than cut-through. The DECNIS design goals were easily met without using cut-through; however, Digital has used the cut-through design on a number of LAN host-based adapters.

When a packet is to be transmitted, certain changes must be made in the data. For example, the IP and OSI protocols require that time-to-live fields and, in some cases, other options be modified. Bridged packets may need address bits modified or conversion between Ethernet and IEEE 802 forms. As with reception, all DEC WANcontrollers perform these operations as the data is transmitted. All cards have hardware assistance for recalculating header checksums and CRCs.

These features are designed to reduce the forwarding delay as much as possible, so that the transit delay is mainly controlled by the time it takes to receive and send the packet. The type of architecture that best describes the DECNIS design is a data-flow, which blends traditional store-and-forward designs with newer cut-through designs. This data-flow architecture processes packets in a distributed manner (i.e., linecards process packets) without transmitting packets prior to complete reception validation of these packets. This design limits the forwarding of packets that are found to be in error, whereas the similar full cut-through design would propagate invalid packets.

Interaction between Routing and Bridging

Designing a combined router and bridge product is complicated by the relationship between the routing and bridging functions.¹¹ A received packet must be subjected to either the bridge forwarding or the routing forwarding process (or maybe both).

Several designs are possible and are illustrated in Figure 5.

- (a) Protocol split. In this design, some protocols are bridged, e.g., Local Area Transport (LAT), and others are routed, e.g., TCP/IP. The bridging and routing functions are completely separate; they merely share line interfaces. Every packet received is passed to either routing (if intended for a protocol that is being routed) or bridging.
- (b) Integrated with one interface. In this design, the routing function is modeled as being layered on top of the bridging function. Theoretically, packets are subjected to the bridging process and then, if they are addressed to the router, subjected to the routing process. In this form of the model, the router uses a single logical interface seemingly connected to a private LAN contained within the bridge/router.
- (c) Integrated with multiple interfaces. This design is similar to the integrated design with one interface, but the router uses all the available interfaces and logically connects to the same extended LAN multiple times.

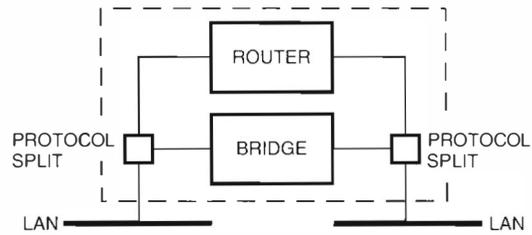
Each design model has advantages and disadvantages, and we considered all three models for the design of the DECNIS routers. The protocol-splitting model has the advantage of simplicity. The major disadvantage is that any particular protocol must be either bridged or routed. The integrated models have the disadvantage of requiring specific management to prevent a routed protocol from also being bridged. In most cases, a protocol is being routed specifically to avoid the problems associated with bridging. The model with one interface also has the disadvantage that the network manager may get confused at attempting to work out which interface is being used for routing. We chose the protocol-splitting model because of its effectiveness and ease of use.

Special Considerations of the DECNIS Architecture

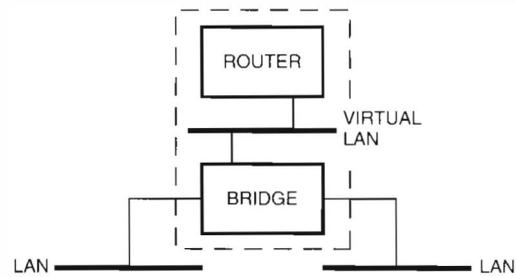
We have discussed special features of the DECNIS system architecture. Now we present some additional DECNIS software design issues.

Control and Management of Linecards

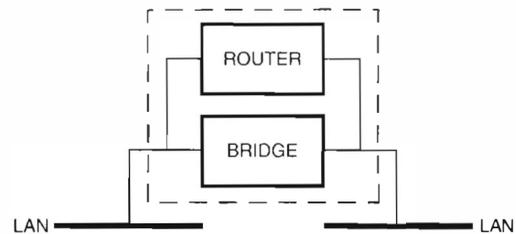
Each linecard is a separate software environment and must be managed and controlled by the management processor. The main tasks required are



(a) Protocol split. Some protocols are passed to the bridging functions, others to the routing functions.



(b) Integrated with one interface. The routing function uses a single LAN address and a single logical interface to the extended LAN.



(c) Integrated with multiple interfaces. The routing function uses all interfaces to attach to the extended LAN multiple times.

Figure 5 Bridge/Routing Design

- "Watchdog" polling. In a standalone network server product, it is necessary to guard against the software getting caught in an infinite loop and hence not responding to management and control messages. The management processor is protected by a hardware watchdog timer, but the linecards do not have a timer. To protect the linecard software, we designed the management processor software to poll each linecard every

400 milliseconds (ms). If there is no response, we reset the card.

- Counters. The network interface cards handle data forwarding and therefore must maintain the required counters (e.g., the number of data bytes received). However, to avoid requiring the linecard to maintain 64-bit counters (which costs memory and requires 64-bit arithmetic), the management processor maintains the full counters and polls the linecards frequently enough to guarantee that the on-card counters do not wrap. Each counter is sized to support the design of the management processor polling every 400 ms.
- Control. When a data link protocol or a routing protocol is started or stopped on an interface, the management processor receives the network management command and issues appropriate control messages to the network interface card.

Distributed Forwarding

Each linecard normally handles the forwarding of bridged and routed data without involving the management processor. This design requires a different approach to meeting the stability requirements from that described for the DEC WANrouter devices.

For example, the DEC WANrouter products discard data packets to meet the routing stability requirements. This discard is limited by the packet management mechanisms to guarantee a minimum level of forwarding performance for the other routing packets, even under worst-case conditions such as those caused by network topology changes. The DECNIS routers do not normally have to discard packets, because the network interface cards can continue to forward data while the management processor handles the routing protocol operations. In addition, correctly designed linecard software guarantees that control traffic is passed to the MPC, even in cases where the software is also passing large amounts of data traffic to the MPC.

Conclusion

This paper describes the complex nature of the design decisions required in the development of Digital's multiprotocol router systems and software. The issues and solutions discussed show how many conflicting technical requirements can be addressed. One example of such a conflict is related to the design goals for the performance of Digital's

multiprotocol routers. While on one hand achieving extremely high system throughput (i.e., the DECNIS 600 router supports a forwarding throughput rate of over 80,000 packets per second), the DECNIS 600 design also addresses the equally critical metric of router stability (i.e., the DECNIS 600 product remains stable under extreme network loads).¹⁰ This balancing of requirements is key to justifying Digital's approach toward router product engineering. As summarized in his recent book on computer systems performance analysis, Raj Jain states that

The performance of a network ... is measured by the speed (throughput and delay), accuracy (error rate) and availability of the packets sent.¹²

Routers that can forward packets but cannot remain stable under heavy loads, or meet the requirements for bursty packet rates as required by many of the newer network applications (e.g., packet-based videoconferencing systems such as Digital's DECspin product), will fail to satisfy customers.¹³ As such, Digital provides a well-tuned, optimized total network solution with DECNIS 600 routers and DECspin products. This synergy of Digital's network applications and network infrastructure components is the ultimate justification for the multiprotocol router design decisions outlined in this paper.

Acknowledgments

Many engineers in Australia, England, Ireland, and the United States participated in the design and implementation of the Digital's multiprotocol routers. We wish to thank all of them.

References

1. *DECnet Digital Network Architecture (Phase V) General Description* (Maynard, MA: Digital Equipment Corporation, Order No. EK-DNAPV-GD-001, 1987).
2. J. Martin and J. Leben, *DECnet Phase V* (Englewood Cliffs, NJ: Prentice-Hall, Inc., 1992).
3. R. Perlman, R. Callon, and M. Shand, "Routing Architecture," *Digital Technical Journal*, vol. 5, no. 1 (Winter 1993, this issue): 62-69.
4. S. Bryant and D. Brash, "The DECNIS 500/600 Multiprotocol Bridge Router and Gateway," *Digital Technical Journal*, vol. 5, no. 1 (Winter 1993, this issue): 84-98.

5. *DECnet Digital Network Architecture (Phase V) Network Routing Layer Functional Specification* (Maynard, MA: Digital Equipment Corporation, Order No. EK-DNA03-FS001, 1991).
6. E. Coffman, Jr., and P. Denning, *Operating Systems Theory* (Englewood Cliffs, NJ: Prentice-Hall, Inc., 1973): 169.
7. S. Bradner, "Benchmarking Terminology for Network Interconnection Devices," Internet Engineering Task Force RFC 1242 (July 1991).
8. K. Ramakrishnan and W. Hawe, "The Workstation on the Network: Performance Considerations for the Communications Interface," *IEEE Computer Society Technical Committee on Operating Systems*, vol. 3, no. 3 (Fall 1989): 29-32.
9. K. Ramakrishnan, "Scheduling Issues for Interfacing for High Speed Networks," *Proceedings of Globecom '92, IEEE Global Telecommunications Conference*, Session 18.04, Orlando, FL (December 1992): 622-626.
10. S. Bradner, "Interop Fall 1992 Router Performance Study," technical presentation, Harvard University, 1992.
11. W. Hawe, M. Kempf, and A. Kirby, "The Extended Local Area Network Architecture and LANBridge 100," *Digital Technical Journal*, vol. 1, no. 3 (September 1986): 54-72.
12. R. Jain, *The Art of Computer Systems Performance Analysis*, ISBN 0-471-50336-3 (New York: John Wiley & Sons, 1991): 23.
13. R. Palmer and L. Palmer, "DECspin: Networked Multimedia Conferencing for the Desktop," *Digital Technical Journal*, vol. 5, no. 2 (Spring 1993, forthcoming).

The DECNIS 500/600 Multiprotocol Bridge/Router and Gateway

The DECNIS 500/600 high-performance multiprotocol bridge/router and gateway are described. The issues affecting the design of routers with this class of performance are outlined, along with a description of the architecture and implementation. The system described uses a distributed forwarding algorithm and a distributed buffer management algorithm executed on plug-in linecards to achieve scalable performance. An overview of the currently available linecards is provided, along with performance results achieved during system test.

The DEC Network Integration Server 500 and 600 (DECNIS 500/600) products are general-purpose communications servers integrating multiprotocol routing, bridging, and gateway functions over an evolving set of local and wide area interfaces. The product family is designed to be flexible, offering a wide range of performance and functionality.

The basic system consists of a Futurebus+ based backplane, a management processor card (MPC), and a packet random-access memory (PRAM) card with a centralized address recognition engine (ARE) for forwarding routed and bridged traffic. Network interface cards or linecards are added to provide network attachment. The DECNIS 500 provides two linecard slots, and the DECNIS 600 provides seven linecard slots. The applications run from local memory on the MPC and linecards. PRAM is used to buffer packets in transit or destined to the system, itself.

The system was developed around distributed forwarding on the linecards to maximize performance. Software provides forwarding on the linecard for internet protocol (IP), DECnet, and open systems interconnection (OSI) traffic using integrated IS-IS (intermediate system to intermediate system) routing, along with bridging functionality for other traffic. The management processor controls the system, including loading and dumping of the linecards, administering the routing and bridging databases, generating routing and bridging control traffic, and network management. X.25 functionality, both for routing data and as an X.25 gateway, and routing for AppleTalk and IPX are supported on the management processor. Performance measurements on a system config-

ured with 14 Ethernets have demonstrated a forwarding performance of 80,000 packets per second as a router or a bridge.

This paper discusses the issues involved in the design of a fast bridge/router. It presents the processing considerations that led us to design the distributed forwarding system used in the DECNIS 500/600 products. The paper then details the hardware and software design and concludes with a performance summary.

Fast Bridge/Router Design Issues

There are a number of conflicting constraints on the design of a bridge/router. It must simultaneously forward packets, participate in the process of maintaining a global view of the network topology, and at all times be responsive to network management. This requires a sophisticated hardware and/or software design capable of striking the correct balance between the demands imposed by these constraints.

The need to make optimum use of the transmission technology is emphasized by the high link tariffs in Europe and the throughput demands of modern high-performance computing equipment. Therefore, the router designer must find methods of forwarding packets in the minimum number of CPU instructions in order to use modern transmission technology to best advantage. In addition to high performance, low system latency is required. The applications that run across networks are often held up pending the transfer of data. As CPU performance increases, the effects of network delay play an increasingly significant role in determining the overall application performance.

Another aspect of forwarding that requires attention is data integrity. Many protocols used in the local area network (LAN) have no data protection other than that provided by the data link checksum. Thus careful attention must be paid to the design of the data paths to minimize the periods when the data is unprotected. The normal technique in bridging is to leave the checksum intact from input to output. However, more advanced techniques are needed, as this simple approach is not possible when translating between dissimilar LAN types.

Two particular operations that constrain the performance of the forwarding process are packet parsing and address lookup. In a multiprotocol router, a variety of address formats need to be validated and looked up in the forwarding table. The most powerful address format in popular use is the OSI NSAP (network service access point), but this is the most complex to parse, with up to 20 octets to be analyzed as a longest-match sequence extracted from padding fields. In a bridge, supporting the rapid learning of media access control (MAC) addresses is another requirement. To provide consistently high performance, these processes benefit from hardware assistance.

Although the purpose of the network is the transmission of data packets, the most critical packets are the network control packets. These packets are used to determine topological information and to communicate it to the other network components. If a data packet is lost, the transport service retransmits the packet at a small inconvenience to the application. However, if an excessive number of network control packets are lost, the apparent topology, and hence the apparent optimum paths, frequently change, leading to the formation of routing loops and the generation of further control packets describing the new paths. This increased traffic exacerbates the network congestion. Taken to the extreme, a positive feedback loop occurs, in which the only traffic flowing is messages trying to bring the network back to stability.

As a result, two requirements are placed on the router. First, the router must be able to identify and process the network control packets under all overload conditions, even at the expense of data traffic. Second, the router must be able to process these packets quickly enough to enable the network to converge on a consistent view of the network topology.

As networks grow to global scale, the possibility emerges that an underperforming router in one

part of the world could cause incorrect network operation in a different geographical region. A bridge/router must therefore be designed to process all network control traffic, and not export its local congestion problems to other parts of the network: a "good citizenship" constraint. To achieve this, the router needs to provide processing and filtering of the received traffic at line rates, in order to extract the network control traffic from the data traffic under worst-case conditions. In some cases, careful software design can accomplish this; however, as line speeds increase, hardware support may be required. Once the control traffic has been extracted, adequate processing power must be provided to ensure that the network converges quickly. This requires a suitable task scheduling scheme.

Another requirement of a bridge/router is that it remain manageable under all circumstances. If the router is being overloaded by a malfunctioning node in the network, the only way to relieve the situation is to shut down the circuit causing the overload. To do this, it must be able to extract and process the network management packets despite the overload situation. Cobb and Gerberg give more information on routing issues.¹

Architecture

To address the requirements of a high-performance multiprotocol bridge/router with the technology currently available, we split the functional requirements into two sets: those best handled in a distributed fashion and those best handled centrally.

The data link and forwarding functions represent the highest processing load and operate in sufficiently local context that they can be distributed to a processor associated with a line or a group of lines. The processing requirements associated with these functions scale linearly with both line speed and number of lines attached to the system. Some aspects of these per-line functions, such as link initialization and processing of exception packets, require information that is available only centrally or need a sophisticated processing environment. However, these may be decoupled from the critical processing path and moved to the central processing function.

In contrast to the lower-level functions, the management of the system and the calculation of the forwarding database are best handled as a centralized function, since these processes operate in the context of the bridge/router as a whole. The

processor workload is proportional to the size of the network and not the speed of the links. Network protocols are designed to reduce the amount of this type of processing, both to minimize control traffic bandwidth and to permit the construction of relatively simple low-performance routers in some parts of the network.

These processing considerations led us to design the DECNIS 500/600 as a set of per-line forwarding processors, communicating on a peer-to-peer basis to forward the normal packets that comprise the majority of the network traffic, plus a central management processor. Although this processor behaves, in essence, like a normal monoprocessing bridge/router, its involvement in forwarding is limited to unusual types of packet.

Having split the functionality between the peer-to-peer forwarding processors and the management processor, we designed a buffer and control system to efficiently couple these processors together. The DECNIS 500/600 products use a central PRAM of 256-byte buffers, shared among the linecards. Ownership of buffers is passed from one linecard to another by a swap, which exchanges a full buffer for an empty one. This algorithm improved both the fairness of buffer allocation and the performance of the buffer ownership transfer mechanism. Fractional buffers much smaller than the maximum packet sizes were used, even though this makes the system more complicated. The consequential economy of memory, however, made this an attractive proposition.

Analysis of the forwarding function indicated that to achieve the levels of performance we

required, we would need hardware assistance in parsing and looking up network addresses. Considerations of economy of hardware cost, board area, and bus bandwidth led us to a single ARE shared among all linecards. This address parser has sufficient performance to support a DECNIS 600 server fully populated with linecards that support each link with a bandwidth of up to 2×10 megabits per second. Above this speed, local address caches are required.

Distributed Forwarding

In understanding the distributed forwarding process used on the DECNIS 500/600, it is convenient to first consider the forwarding of routing packets, and then to extend this description to the processing of other packet types. In the routing forwarding process, as shown in Figure 1, the incoming packets are made up of three components: the data link header, the routing header, and the packet body.

The receive process (RXP) terminates the data link layer, stripping the data link header from the packet. The routing header is parsed and copied into PRAM unmodified. Any required changes are made when the packet is subsequently transmitted. The information needed for this is placed in a data structure called a packet descriptor, which is written into space left at the front of the first packet buffer. The packet body is copied into the packet buffer, continuing in other packet buffers if required.

The destination network address is copied to the ARE, which is also given instructions on which address type needs to be parsed. The RXP is now free to start processing another incoming packet.

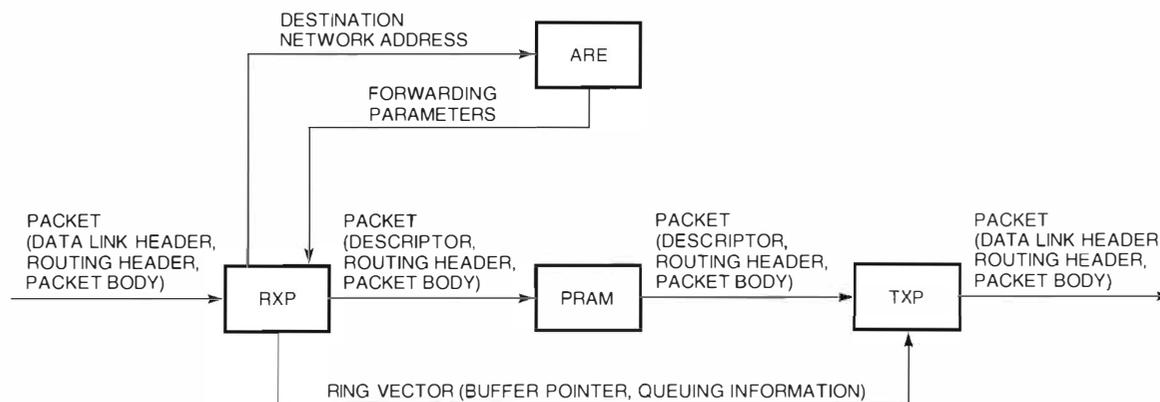


Figure 1 Distributed Forwarding

When the address lookup process has completed, the RXP is able to read from the ARE the forwarding parameters needed to complete the processing of the packet. These parameters contain information about the output port and channel to use, the destination data link address for the next hop, and any translation information. The RXP combines this information with some information saved from parsing the packet to build the packet descriptor in PRAM.

The RXP builds a set of ring vectors for the packet, one for each buffer used. Each ring vector contains a pointer to the PRAM buffer used, plus some additional information used to decide on which queue the buffer should be stored and to determine its relative importance to the system. During congestion, this information is used by the linecards to discard the least important packets first. These ring vectors are then exchanged with the transmit process (TXP) on the output linecard, which queues them ready for transmission. Before the TXP starts to process a packet for transmission, it reads the descriptor from the first PRAM buffer. From the information in the descriptor, the TXP is able to build the data link header, determine the routing header translation requirements, and locate a number of fields in the header (such as the OSI segmentation and quality of service fields) without having to reparse the header. The TXP builds the data link header, reads the routing header from PRAM, makes the appropriate modifications, and then completes the packet by reading the packet body from PRAM.

Since the transmit packet construction follows the packet transmission order byte for byte, implementations can be built without further intermediate transmission buffering. Linecards need only provide sufficient transmit buffering to cover the local latency requirements. In one instance, a linecard has significantly less than a full packet buffer. This small buffering requirement implies reduced system latency and makes available a number of different implementation styles.

If the RXP discovers a faulty packet, a packet with an option that requires system context to process, or a packet that is addressed to this system (including certain multicast packets), it queues that packet to the management processor in exactly the same way that it would have queued a packet for transmission by a TXP. The MPC contains a full-function monoproccessor router that is able to handle these exception cases. Similarly, the MPC sends packets by presenting them to the appropriate TXP in exactly the same format as a receiver.

The bridge forwarding process operates in a fashion similar to the routing forwarding process, except that the data link header is preserved from input port to output port, and only the data link header is parsed.

Buffer System

A schematic description of the DECNIS 500/600 buffer system is shown in Figure 2. The RXPs have only sufficient buffering to cope with the latencies that must be sustained in their various stages of packet processing. All long-term storage of packets takes place while the packet is owned by the TXP. When an RXP has finished processing a packet, it swaps the PRAM buffers containing the packet for the same number of empty buffers owned by the TXP that transmits the packet. Only if the TXP is able to replace these buffers with empty buffers does the transfer of ownership take place. If the swap cannot complete due to lack of free buffers, the RXP reuses these buffers for another packet. In this way, no transmitter is able to accumulate buffers and thereby prevent a receiver from receiving packets intended for other output ports.

The design of an efficient buffer transfer scheme is an important aspect of a high-performance multi-processor router. We solved this problem by using a set of single writer/single reader rings, with one ring associated with each pair-wise exchange of buffers that can take place in the system. Thus each TXP has associated with it one ring for each of the RXPs in the system (including its own), plus one for the management processor. When an RXP has a buffer to swap, it reads the next transfer location in its ring corresponding to the destination TXP. If it finds a free buffer, it exchanges that buffer with the one to be sent, keeping the free buffer as a replacement. The transferred information consists of a pointer to the buffer, its ownership status, and some information to indicate the type of information in the buffer. This structure is known as a ring vector. A single-bit semaphore is used to indicate transfer of ownership of a ring vector.

The buffer transfer scheme schematic shown in Figure 2 illustrates how this works. Each transmit port (TXa or TXb) has a ring dedicated to each of the receivers in the system (RXa and RXb). RXa swaps ring vectors to the "a" rings on TXa and TXb, and RXb swaps ring vectors to the "b" rings on TXa and TXb.

During buffer transfer, the TXP runs a scavenger process, scanning all its rings for new buffers, queuing these buffers in the transmit queues (TXQs)

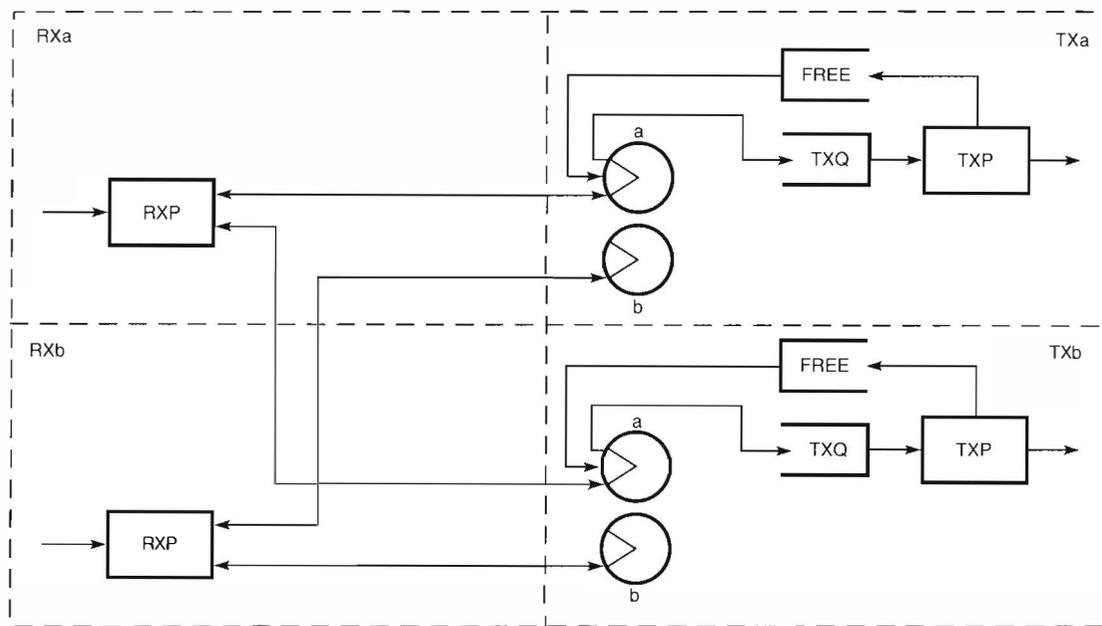


Figure 2 Movement of Buffer Ownership

specified by the ring vector, and replacing the entries in the ring from the local free list. The buffer type information enables the transmit linecard to quickly determine the importance of the buffer. Thus if the linecard runs short of buffers due to congestion, it is able to discard less important packets in preference to those packets required to preserve the stability of the network.

Through judicious optimization of the ring vector encodings, we were able to condense this ring swap transaction into a single longword read followed by a single longword write for each buffer swap, for all unicast traffic. For multicast traffic, a second longword was required. To reduce the amount of bus traffic and the processor time associated with the scavenge process, the random-access memory (RAM) that holds the rings is physically located on the transmit linecard. Hardware is used to watch the rings for activity and report this to the TXP.

Analysis of the traffic patterns indicated that considerable economies in PRAM could be made if we fragmented long packets over a number of buffers. We achieved a satisfactory compromise between the processing overhead associated with buffer management and memory efficiency through the use of 256-byte buffers. With this buffer size, a large fraction of the packets are contained within a single buffer. When a linecard is driven into output congestion, it is no longer certain that a complete set of

packet buffers will be swapped. We therefore had to introduce a simple protocol to ensure that a packet was queued for transmission only if it had been fully transferred to the transmitting linecard. To cope with dissimilar swap and scavenge process speeds, we had to stage the transfer of buffers. Thus, the TXPs collect a complete set of buffers from an RXP before queuing the packet for transmission; this process is called binning. In this way, a partial transfer due to congestion or a slow receiver does not block the progress of other ports in the system.

Bridging needs a mechanism to support the distribution of flooded and multicast packets to multiple output ports. In some distributed systems, this function is handled by replicating the packet via a copy process. In other systems, the packet is handled by a central multicast service. The use of a central multicaster gives rise to synchronization issues when a destination address moves from the unknown to the learned state. Replication by the linecards is not practical in this architecture since the linecards do not hold a local copy of the buffer after it has been copied to PRAM. We therefore use a system in which multicast buffers are loaned to all the transmit linecards. A "scoreboard" of outstanding loans is used to record the state of each multicast buffer. When a buffer is returned from all its borrowers, it is

added to the multicast free queue and made available for reuse. The loan process and the return process are similar to the normal swap and scavenge process, but the ring vector is extended slightly to include the information needed for rapid dereferencing.

Centralized Resources

Three central resources are used in the DECNIS 500/600 products: MPC, PRAM, and ARE. Centralizing these resources reduced both the cost and the complexity of the system. There are two ways of building a distributed processing router. In one method, the router consists of a federation of full-function routers, each a separate network node. An alternative method is to employ a partially centralized design in which only one processor is the router in the traditional sense. The central processor is the focus for network management, calculating the forwarding table and being a central repository for the context of the router, and the peripheral processors undertake the majority of the forwarding work. An analysis of the cost and complexity both from a system and a network perspective led us to choose the latter approach. Thus the MPC provides all the software functionality necessary to bind the collection of forwarding agents located on the linecards together to form a router. To the rest of the network, the system appears indistinguishable from a traditionally designed router. The processing capability and memory requirements of the MPC are those associated with a typical medium-performance multiprotocol bridge/router.

We had a choice of three locations for the PRAM: distributed among the receiving linecards, distributed among the transmitting linecards, or located centrally. Locating the buffering at the receiver would have meant providing the maximum required transmitter buffering for each transmitter at every receiver. Locating the long-term packet buffering at the transmitters would have meant staging the processing of the packets by storing them at the receiver until the transmit port was determined and then transferring them to the appropriate transmitting linecard. This would have increased the system latency, the receiver complexity, and its workload. An analysis of the bus traffic indicated that for a router of this class, there would be adequate bus bandwidth to support the use of a centrally located, single shared packet buffer memory. With this approach, however, every packet

crosses the bus twice, rather than once as in the other approaches. Nevertheless, we chose to base the system around a single packet memory, and win the consequential economies in both linecard cost and board area.

An analysis of the processing power needed to parse and look up a network address led us to conclude that the linecards would need some form of assistance if the processing power associated with each line was to be constrained to a reasonably cost-effective level. This assistance is provided by the ARE. Some advanced development work on the design of hardware search engines showed that it was possible to design a single address parser powerful enough to be shared among all the linecards. This search engine was adaptable enough to parse the complex structure of an OSI NSAP, with its two right-justified padded fields and its longest-match semantics. In addition, the engine was able to cope with the other routing protocol address formats and the learning requirements of bridging. By centralizing the forwarding database, we also avoided the processing and bus overhead associated with maintaining several distributed forwarding databases and reduced the cost and board area requirements of the linecards.

The bus bandwidth and lookup rate needed to support multiple fiber distributed data interface (FDDI) linecards would place an excessive burden on the system. For FDDI, therefore, we equip the central lookup engine with a linecard-resident address cache.

DECNIS 500/600 Hardware Design

There are three primary systems in the DECNIS 500/600: the backplane, together with its interface circuitry, the system core functions contained in the MPC and the PRAM, and the various linecards. In this section, we describe the hardware design of each of these.

Backplane and Interface Logic

The DECNIS 500/600 backplanes are based on the Futurebus+ standard using 2.1-volt (V) terminated backplane transceiver logic (BTL).^{2,3} Although all current cards use 32-bit data and address paths, the DECNIS 600 backplane has been designed to support 64-bit operation as well.

Common to all current modules except the PRAM card, the basic backplane interface consists of two applications specific integrated circuits (ASICs), BTL transceivers, and a selection of local memory

transfers, e.g., a summary register of write activity to the RVMEM.

The data-path device can be driven from a single processor port (port T) for use in simpler, low-speed linecards. In addition, the architecture supports two data-path devices (primary and secondary) served by a common controller connected to the local bus of the primary device. Each data-path device adopts a different node identifier in the backplane address space.

Dedicated lines on the backplane are provided for power status, temperature sensing, and other system requirements.

Processor and Memory Modules

The MPC has two processors, a main processor and a uniprocessor version of the common backplane interface. The main processor, a VAX device, is in overall command of the system and provides all the management and forwarding services found in a monoprocessor router. The 16-bit, processor-based backplane interface frees the main processor from time-critical backplane-associated tasks.

A block diagram of the memory module is shown in Figure 4. Separate dynamic RAM (DRAM) arrays are used for data buffering and the forwarding database associated with the ARE. Ring structures in static memory are used to allow the linecards to post requests and read responses from the ARE, which is based on the TRIE system originally developed for text retrieval.^{4,5}

An ASIC was developed for the ARE; it was extended to include some of the other module control logic, e.g., PRAM refresh control and the synchronous portion of the Futurebus+ backplane interface.

Network Interface Cards—Linecards

The DECNIS 500/600 products currently offer synchronous communications, Ethernet, and FDDI adapters, all using variants of the standard backplane interface.

Two synchronous communication adapters are available: a two-line device operating at up to 2.048 megabits per second, and a higher fan-out device supporting up to eight lines at a reduced line rate of 128 kilobits per second. All lines are full duplex with modem control. The lower-speed adapter uses a uniprocessor architecture to drive three industry-standard serial communications controllers (SCCs). The data and clocks for the channels, along with an extra channel for multiplexed modem control, are

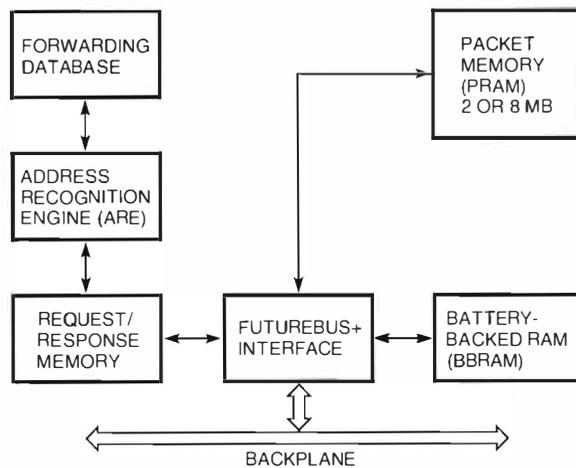


Figure 4 PRAM and ARE Module Block Diagram

connected to a remote distribution panel using a 2-meter umbilical cord. Panels are available to support eight lines using the RS232, EIA422, or V.35 electrical interface. A four-line multistandard variant allows mixed electrical interfaces from a single adapter at a reduced fan-out. The multistandard panel uses a 50-pin cable common to other communication products from Digital.

The two-line device uses a four-processor interface as shown in Figures 3 and 5. The SCC is an ASIC device designed specifically for the data-flow style of processing adopted in the system architecture. It is closely coupled to the data-path ASIC and processors for optimal throughput. The hardware design has minimal dependency between the transmit and receive tasks, recognizing the limited coupling required by acknowledged data link protocols such as high-level data link control (HDLC). State information is exchanged between processors using a small dual-ported RAM in the SCC. In addition, each SCC and associated processors operate as a separate entity, resulting in consistent performance when forwarding both on and off the module. Two 50-pin multistandard interfaces (EIA422 and V.35 only) are provided on the module handle.

Several Ethernet adapters are available. A single-port thick-wire adapter uses a dual-processor architecture (primary R and T ports in Figure 3), along with a discrete implementation, to interface the Ethernet and its associated buffer (tank) memory. This design was reengineered to put the tank memory interface (TMI) into an ASIC, resulting in a dual-port (full implementation of the interface shown in Figure 3 plus two Ethernet interfaces) adapter

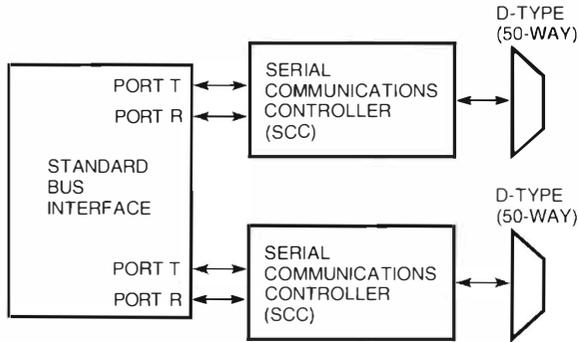


Figure 5 DEC WANcontroller 622 Block Diagram

derivative. This adapter is available in two variants supporting thick-wire, and ThinWire wiring schemes.

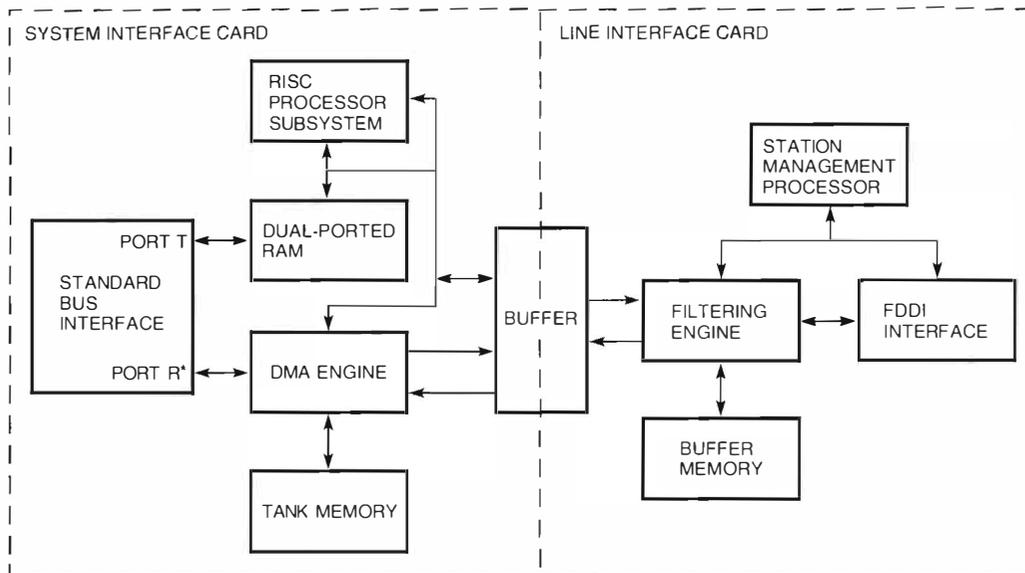
As shown in Figure 6, the FDDI adapter (DEC FDDIcontroller 621) is a two-card option designed to cope with the high filtering and forwarding rates associated with FDDI. The hardware includes a filtering engine closely coupled to the FDDI chip set, a synchronous interconnect between the two cards, and a multichannel DMA engine for data transfer through the device. The DMA engine maintains tank memory under reduced instruction set computing (RISC) processor control, and can be set

up and monitored with minimal processor overhead. Data is transferred to or from buffers in PRAM to the tank memory, where complete packets are kept in contiguous address space. A second DMA channel transfers complete packets in a single burst to or from the buffer memory on the line interface card.

Traffic processing between buffer memory and the ring is done in hardware. A third DMA path is used to prefetch and then burst transfer packet header information from tank memory into the RISC processor subsystem for packet processing. The DMA engine, which includes tank memory arbitration, can queue multiple commands and operate all DMA channels in parallel. The 32-bit RISC subsystem provides the linecard processing, communicating with the bus interface processor using dual-ported RAM. Modular connectivity is offered for different physical media. The module also supports dual-attach and optical-bypass options.

DECNIS 500/600 Software Design

This section describes the software design of the DECNIS 500/600. The structure of the management processor software is first described. The structure of the linecard receiver and transmitter is then discussed, followed by details on how we expanded the design to forward multicast packets.



*DMA interface replaces second processor.

Figure 6 DEC FDDIcontroller 621 Block Diagram

Management Processor Software

The DECNIS 500/600 MPC software structure, as shown in Figure 7, consists of a full-function bridge/router and X.25 gateway, together with the software necessary to adapt it to the DECNIS 500/600 environment. The control code module, which includes the routing, bridging, network management, and X.25 modules, is an extended version of Digital's WANrouter 500 software. These extensions were necessary to provide configuration information and forwarding table updates to the DECNIS 500/600 environment module. This module hides the distributed forwarding functionality from the control module. Thus the control module is provided with an identical environment on both the MicroServer and DECNIS 500/600 platforms.

The major component of the DECNIS 500/600 environment module contains the data link initialization code, the code to control the linecards, and the code to transform the forwarding table updates into the data structures used by the ARE. A second component of the environment module contains the swap and scavenge functions necessary to communicate with the linecards. Because of the real-time constraints associated with swap and scavenge, this function is split between the management processor on the MPC and an assist processor.

The control code module was designed as a full-function router; thus we are able to introduce new functionality to the platform in stages. If a new protocol type is to be included, it can be initially executed in the management processor with the linecards providing a framing or data link service. At a later point, the forwarding components can be moved to the linecards to provide enhanced performance. The management processor software is described in more detail elsewhere in this issue.¹

Linecard Reception

The linecard receiving processes are shown in Figure 8. The receiver runs four processes: the main receive process (RXP), the receive buffer system ARE process (RXBA), the receive buffer system descriptor process (RXBD), and the swap process.

The main receive process, RXP, polls the line communications controller until a packet starts to become available. The RXP then takes a pointer to a free PRAM buffer from the free queue and parses the data link header and the routing header, copying the packet into the buffer byte-by-byte as it does the parse. From the data link header, the RXP is able to determine whether the packet should be routed or bridged. Once this distinction has been made, the routing destination address or the destination

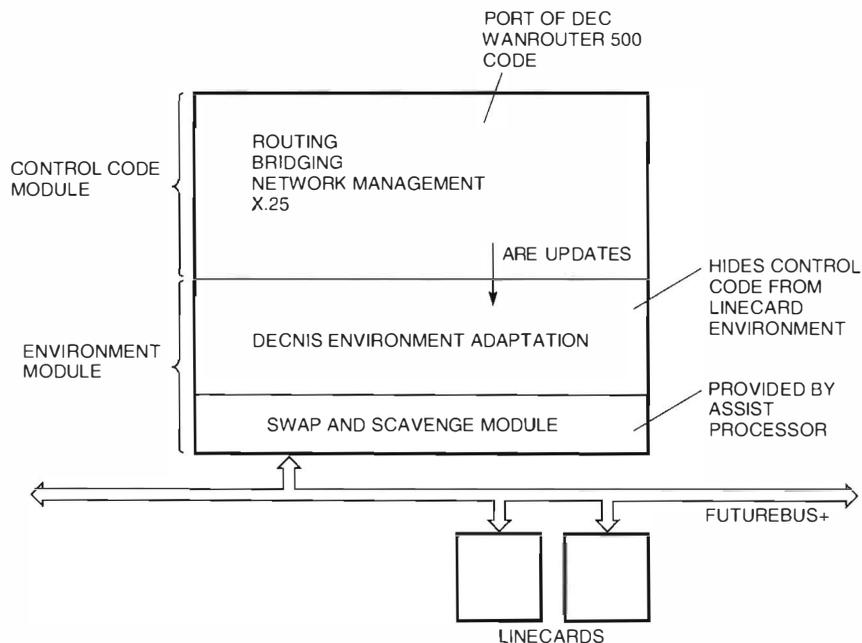


Figure 7 MPC Software Structure

process, the transmit buffers system select process (TXBS), the main transmit process (TXP), and the TXB release process. These are shown in Figure 10.

The scavenge rings process scans the swap rings for new buffers to be queued for transmission, replacing them with free buffers. Buffers are queued in reassembly bins (one per destination ring) so that only complete packets are queued in the holding queues. The process tries to replenish the destination rings from the port-specific return queues, but failing this it uses the free list. The primary use of the port-specific return queues is in multicasting (see the section Linecard Multicasting).

The scavenge bins process scans the reassembly bins for complete packets and transfers them to the holding queues. Since different protocols have different traffic characteristics, the packets are queued by protocol type.

The TXBS process dequeues the packets from these holding queues round-robin by protocol type. This prevents protocols with an effective congestion control algorithm from being pushed into congestion backoff by protocol types with no effective congestion control. It also allows both bridged and routed protocols to make progress despite overload. The scavenge bins and TXBS

processes between them execute the DECbit congestion control and packet aging functions. By assuming that queuing time in the receiver is minimal, we are able to simplify the algorithms by executing them in the transmit path. New algorithms had to be designed to execute these functions in this architecture.

The TXP process transmits the packet selected by TXBS. TXP reads in the descriptor, prepending the data link header and transmitting the modified routing header. When transmitting a protocol that uses explicit acknowledgments, like HDLC, the transmitted packet is transferred to the pending acknowledgment queue to wait for acknowledgment from the remote end. Before transmitting each packet, the transmitter checks the current acknowledgment state indicated by the receiver. If necessary, the transmitter either moves acknowledged packets from the pending acknowledgment queue to the packet release queue, or, if it receives an indication that retransmission is required, moves them back to the transmit packet queue.

The TXB release process takes packets from the prerelease queue and separates them into a series of queues used by the swap process. Simple

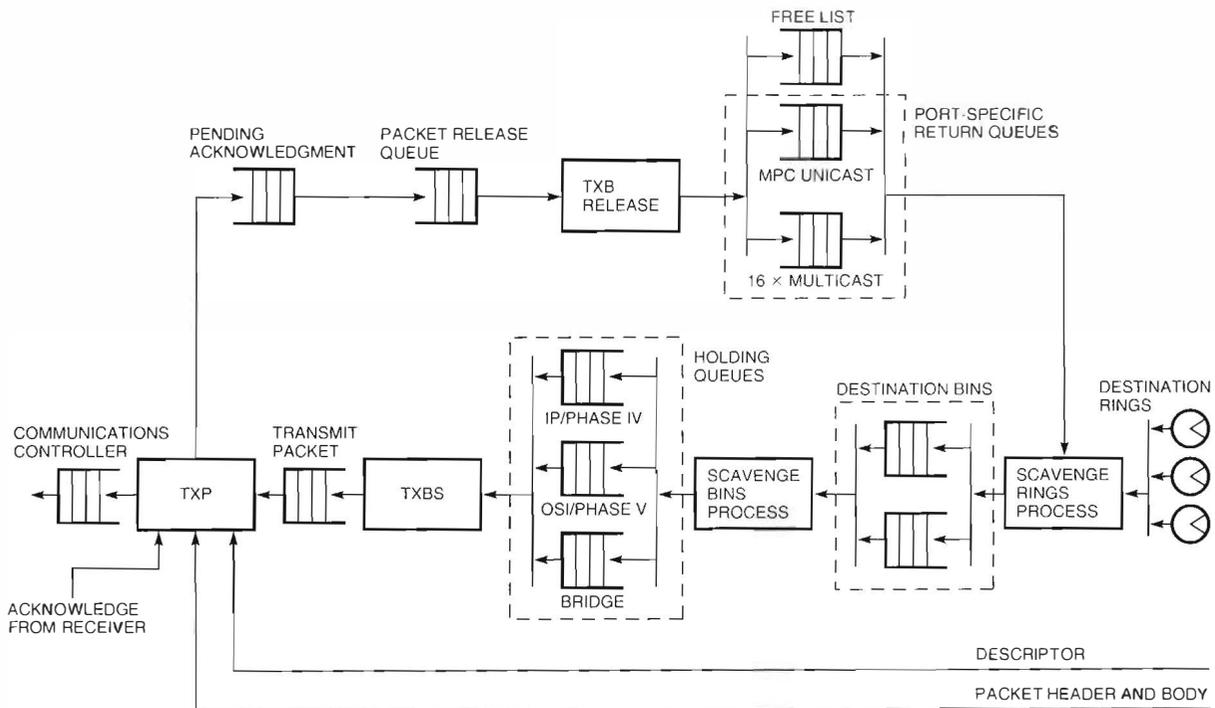


Figure 10 Linecard Transmit Processes

unicast packets have their buffers returned to the transmitter free pool. The multicast packets have their buffers placed on the port-specific queue for the source linecard, ready for return to their originating receiver. Packets intended for return to the management processor are also queued separately.

Linecard Multicasting

A bridge multicast or flooded buffer must be transmitted by a number of linecards. This is achieved by swapping a special type of ring vector, indicating that the buffer is only on loan to the transmitting linecard and must be returned to its owner upon completion. In addition to the normal packet type, fragmentation, and buffer identification information, the ring vector contains local referencing information indicating where it is stored on the multicast heap. The receiver keeps a record of which multicast buffers are on loan to which transmitters. The scavenge process notes in which ring it found the ring vector. After transmission, the TXB release process places the ring vector on the corresponding port-specific return queue. These ring vectors are then preferentially returned to their owner via the swap process. As the receiver gets these buffers back, it checks them off against a scoreboard of issued buffers. When a buffer is received from all destination linecards to which it was loaned, the buffer is moved back on the free list. For this to work successfully, some buffers must be set aside specifically for use by the multicast process.

Debugging the System

Extensive simulation was performed during system development. A model based on VHDL (a hardware description language used for simulation and logic synthesis) was built to simulate the queues, processes, bus accesses, and bus latency for the fast forwarding paths. Models were developed for the different styles of linecards, and many different traffic scenarios (packet size, packet type, packet rates) were simulated to verify the original thinking and architectural assumptions. In addition, simulation was performed on the software to measure code correctness and execution times. Gate arrays and modules were both functionally simulated and timing verified; analog modeling techniques were used to verify signal integrity of the backplane and selected etches.

The linecard processors used have a serial port and masked ROM embedded in the device. The internal ROM was programmed with a simple boot and console procedure. Provisions for a debug console via a ribbon cable to the module were developed, allowing a terminal connection to be made from the management processor to any linecard processor. Each processor on a module is software selectable from the console, which allows limited access functions to peek and poke memory maps, set break points, and step through the code. The system was enhanced by developing a breakout box and workstation environment that could connect to multiple linecards, offering multiple windows to different modules in parallel. The code executed under this regime ran at full speed. The environment allowed remote access, which proved useful between the two main module development sites in England and Ireland when problems required close cooperation between the two groups.

Performance

Performance measurements have been made on the DECNIS 500/600 products for DECnet Phase IV, DECnet Phase V (OSI), IP, and bridged traffic. For a detailed description of the measurement methodology and a comparison between the performance of the DECNIS 500/600 and competing bridge/routers, the reader is referred to independent test results compiled by Bradner.⁶

A summary of the LAN performance measured by Bradner and the WAN performance measured by ourselves is shown in Tables 1, 2, and 3. Table 1 shows the Ethernet-to-Ethernet forwarding throughput for minimum-sized packets. These measurements show the maximum forwarding performance with no packet loss. The use of a no-loss figure for comparison between different designs is important because this represents the maximum throughput usable by a network application. If the applications attempt to run at more than the loss-free rate, the packet loss causes the transport protocols to back off to the loss-free operating point. The Ethernet-to-Ethernet figures indicate the near linear scalability of performance with number of lines. Ethernet forwarding performances of this magnitude are well in excess of those required to operate on any practical LAN. The correctness software ensures the reception of any routing packets for a significant period after these rates are exceeded.

Table 1 64-byte Ethernet-to-Ethernet Packet Throughput

Protocol	Number of Ports		
	1	4	6
Bridge	13,950	48,211	80,045
IP	13,362	51,960	79,452
DECnet	9,330	34,164	53,746
OSI	6,652	25,891	38,837

Table 2 FDDI-to-FDDI Throughput

	Packet Size	
	64 Byte	2048 Byte
Throughput	16%	76%
Maximum pps*	56,869	4,352
Bandwidth		85.5 Mb/s

Note: pps = packets per second

Table 3 WAN-to-WAN Performance for Routed Traffic

NPDU Size	Measured Percentage Line Utilization		
	DECnet Phase IV	DECnet Phase V (OSI)	IP
46	96%	95%	93%
128	99%	99%	98%
512	100%	100%	100%
1450	100%	100%	100%

Note: NPDU = network packet data unit

Measurements also indicated that the unidirectional and bidirectional forwarding performances are substantially the same, which is not the case for all router designs. This is of more than academic significance. Poorly designed Ethernet subsystems do not provide adequate transmit processing power under conditions of receive overload. Such subsystems suffer from a condition known as "live-lock." In this condition, the receiver uses up all the processing cycles, thus preventing the transmitter from attempting the transmission that would force a collision on the Ethernet and thereby restore fair operation.

The FDDI forwarding performance is shown in Table 2. These measurements were also taken at the zero-loss operating point and indicate industry-leading performance results.

The performance of the WANcontroller 622 running at 2 megabits per second is shown in Table 3. These measurements were taken using HDLC (with acknowledgments) as the data link, with a packet overhead of +19 octets for Phase IV and +6 octets for OSI and IP. These results indicate that the lines were running close to saturation.

Acknowledgments

The DECNIS 500/600 project has involved a great number of people located around the world. The authors wish to recognize everyone's contribution to the largest project undertaken by the Reading and Galway network engineering groups. Special thanks are extended to Mick Seaman for his leadership and guidance throughout the advanced development and early implementation phases of this project.

References

1. G. Cobb and E. Gerberg, "Digital's Multiprotocol Routing Software Design," *Digital Technical Journal*, vol. 5, no. 1 (Winter 1993, this issue): 70-83.
2. *Futurebus+ Logical Layer Specification*, IEEE Standard 896.1-1991 (New York: The Institute of Electrical and Electronics Engineers, 1992).
3. *Futurebus+ Physical Layer and Profile Specifications*, IEEE Standard 896.2-1991 (New York: The Institute of Electrical and Electronics Engineers, 1992).
4. E. Fredkin, "TRIE Memory," *Communications of the ACM*, vol. 3 (1960): 490-499.
5. D. Knuth, *The Art of Computer Programming, Sorting and Searching*, vol. 3 (Reading, MA: Addison-Wesley Publishing Co., Inc., 1973): 481-490.
6. S. Bradner, "Testing the Devices," *Proceedings of Fall Interop 1992*. Available on the Internet by anonymous FTP from hsdndev.harvard.edu in /pub/ndtl.

Frame Relay Networks

Frame relay networks reduce the cost of transmission lines and equipment and improve network performance and response time. Designed for transmission lines with a low error rate, frame relay networks provide minimal internal checking, and consequently, error detection and recovery is implemented in the attached user systems. The Frame Relay Bearer Service was developed specifically as a data service to handle high-volume, bursty traffic by means of high-speed packet transmission, minimal network delay, and efficient use of network bandwidth. The frame protocol supports the data transfer phase of the Service; the frame relay header and the local management interface are sources of congestion avoidance mechanisms. Current implementations include the StrataCom IPX FastPacket digital networking system, which provides the frame relay network, and Digital's DECNIS 500/600 and DEC WANrouter 100/500 software for attaching user equipment.

Today's communications networks are built using high-speed digital trunks that inherently provide high throughput, minimal delay, and a very low error rate. Such transmission networks supply highly reliable service without the overhead of error control functions. Frame relay is a packet-mode transmission network service that exploits these network characteristics by minimizing the amount of error detection and recovery performed inside the network.

This paper explains the nature of the Frame Relay Bearer Service (FRBS) and provides details of the interface defined for attaching user equipment. The implications for higher-layer protocols in the user equipment are also considered.

Following this tutorial, the paper introduces some current implementations. As an example of equipment used to construct a frame relay network, the technology deployed by the StrataCom integrated packet exchange (IPX) FastPacket range of equipment is described. Access to a frame relay network is typically via a router, as is illustrated in the discussion of two Digital products:

- The DECNIS V2.1 software, i.e., network integration server, for either the DECNIS 500 or the DECNIS 600 hardware units (abbreviated as DECNIS 500/600)

- The DEC WANrouter V1.0 software for either the DEMSB or the DEMSA hardware units (subsequently referred to as the WANrouter 100/500)

The paper concludes with a brief discussion of activities related to the further development of frame relay technology.

The Frame Relay Bearer Service

The FRBS was developed specifically as a data service to handle high-volume, bursty traffic. The service was designed to provide high-speed packet transmission, minimal network delay, and efficient use of network bandwidth.¹ Local area network (LAN)-to-LAN wide area internetworking is a typical application.

The packet-based frame relay technology uses a combination of features from existing standards for X.25 packet switching and time division multiplexed (TDM) circuit switching.² Frame relay provides an X.25-like statistical interface but with lower functionality (in terms of error correction and flow control) and hence higher throughput, because most processing requirements have been removed. At the same time, frame relay has the higher speed and lower delay qualities of TDM circuit switching without the need for dedicated full-time devices and circuits and wasted time slots

when no data is being transmitted. The fact that the FRBS need not provide error detection/correction and flow control relies on the existence of intelligent end-user devices, the use of controlling protocol layers (CPLs), and high-speed and reliable communication systems. Access to the FRBS is via a frame relay interface defined between data circuit-terminating equipment (DCE) on the network side and data terminal equipment (DTE) on the user side. A typical frame relay configuration is shown in Figure 1.

In 1990, four vendors—StrataCom, Digital Equipment Corporation, Cisco Systems, and Northern Telecom—collaborated on developing a specification called the Frame Relay Specification with Extensions.³ This document, edited by StrataCom, introduced a local management interface (LMI) to provide control procedures for permanent virtual circuits (PVCs). The LMI was structured into a basic, mandatory part and a number of optional extensions. It focused on PVCs for frame relay point-to-point connections rather than on switched virtual connections (SVCs), because SVCs are not well suited for LAN interconnection.

Subsequently, standards have emerged in this area that adopt the basic form of the LMI, without the optional extensions, as an annex for PVC control procedures. These standards do differ, however, in some respects. First, the recent standards have specified primary rate access (PRA) for the physical interface rather than Comité Consultatif

International de Télégraphique et Téléphonique (CCITT) Recommendation V.35 for wideband electrical signaling, which was adopted in the original joint document.⁴ Second, the standards include signaling support for SVCs. The frame relay service is being standardized by both the American National Standards Institute (ANSI) committee, ANSI T1S1, and the CCITT.

Frame Protocol

ANSI used the earlier work as a basis for developing the frame protocol to support the data transfer phase of the FRBS.⁵ This protocol operates at the lowest sublayer of the data link layer of the International Organization for Standardization/Open Systems Interconnection (ISO/OSI) seven-layer reference model. The protocol is based on a core subset of link access protocol D (LAP-D), which is used in the Integrated Services Digital Network (ISDN). The frame protocol specifies the following characteristics of the frame relay protocol data unit (PDU) or frame:

- Frame delimiting, alignment, and transparency, provided by high-level data link control (HDLC) flags and zero-bit insertion/extraction.
- Framed integrity verification, provided by a frame check sequence (FCS). The FCS is generated using the standard 16-bit CCITT cyclic redundancy check (CRC) polynomial.

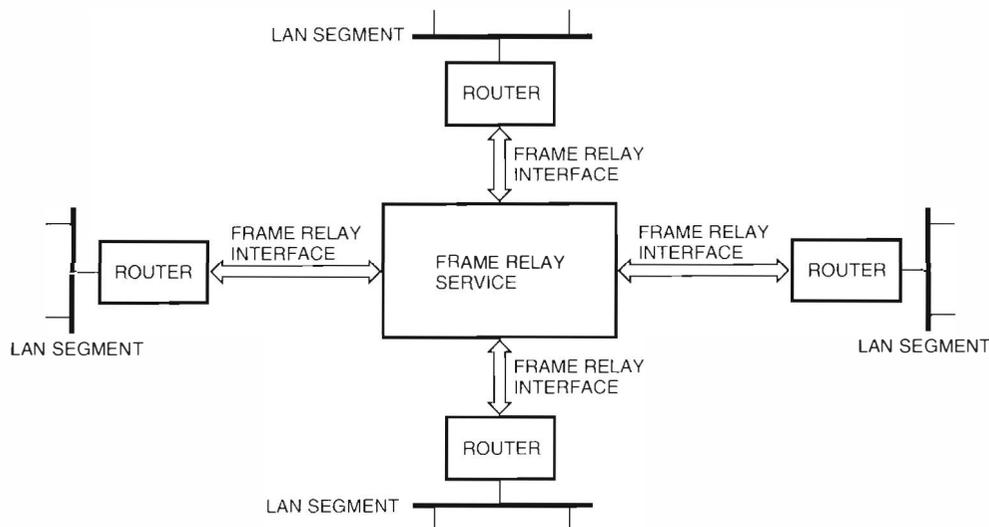


Figure 1 Typical Frame Relay Configuration

- Frame relay addressing, using headers of 2, 3, or 4 octets in length. Figure 2 shows the frame relay header formats. An extended address (E/A) bit is reserved in each octet to indicate whether or not the octet is the last one in the header.

Most of the header represents the data link connection identifier (DLCI), which identifies the frame's virtual circuit. The header may also contain a DLCI or control indicator (D/C) to indicate whether the remaining six bits are to be interpreted as lower DLCI bits or as control bits. For alignment with LAP-D, the header also contains a bit to discriminate between commands and responses (C/R). This bit is not used for supporting frame relay access.

The DLCI influences the routing of the frame to the desired destination. The DLCI is also used to multiplex PVCs onto the physical link and enables each endpoint to communicate with multiple destinations by means of a single network access. DLCIs may have either global or local significance in the network. In the global case, the scope of the DLCI extends throughout the network such that a particular DLCI always identifies the same destination, thus making the frame relay network look more like a LAN. In the local case, the scope of the DLCI is limited to the particular interface. When local DLCIs are used, the same DLCI can be reused at another interface to represent a different connection.

- Congestion control and avoidance information. The frame relay header also contains the forward explicit congestion notification (FECN) bit, the

backward explicit congestion notification (BECN) bit, and the discard eligibility (DE) indicator, which are discussed in the Congestion Avoidance section.

Permanent Virtual Circuit Control Procedures

Frame relay PVCs provide point-to-point connections between users. Although the PVCs are set up for long periods of time, they can still be considered virtual connections because network resources (i.e., buffers and bandwidth) are not consumed unless data is being transferred.

For interface management purposes, the frame relay interface includes control procedures based on the LMI definition contained in the original multivendor specification. These procedures use messages carried over a separate PVC identified by an in-channel signaling DLCI. The management messages are transferred across the interface using data link unnumbered information frames, as defined in CCITT Recommendation Q.922.⁶ The messages use a format similar to that defined in CCITT Recommendation Q.931 for ISDN signaling in support of call control and feature invocation.⁷ Each message is formed from a set of standardized information elements defining the message type and associated parameters. The control procedures perform three main functions:

- Link integrity verification initiated by the user device and maintained on a continuous basis. This function allows each entity to be confident that the other is operational and that the physical link is intact.

DLCI (6 HIGH-ORDER BITS)			C/R	E/A = 0
DLCI (4 LOW-ORDER BITS)	FECN	BECN	DE	E/A = 1
DLCI (6 HIGH-ORDER BITS)			C/R	E/A = 0
DLCI (4 BITS)	FECN	BECN	DE	E/A = 0
DLCI (6 LOW-ORDER BITS)			D/C	E/A = 1
DLCI (6 HIGH-ORDER BITS)			C/R	E/A = 0
DLCI (4 BITS)	FECN	BECN	DE	E/A = 0
DLCI (7 BITS)				E/A = 0
DLCI OR CONTROL (6 LOW-ORDER BITS)			D/C	E/A = 1

Figure 2 Frame Relay Header Formats

- When requested by the user, full status network report providing details of all PVCs. The user would normally request such a report at start-up and then periodically.
- Notification by the network of changes in individual PVC status, including the addition of a PVC and a change in PVC state (active/inactive).

The management protocol is defined in Annex D of ANSI T1.617, with equivalent functionality also defined in CCITT Recommendation Q.933, Annex A.⁸⁹

Effect on Higher-level Protocols

Frame relay provides a multiplexed PVC interface and, with regard to routing software, can be modeled as a set of point-to-point links. However, the characteristics of the frame relay service differ from normal point-to-point links. The major differences are as follows:

- Round-trip delay across a frame relay network is normally longer than the delay across a dedicated point-to-point link.
- PVC throughput can be as high as 2 megabits per second (Mb/s), whereas many existing leased lines operate at lower rates.
- A single frame relay interface can have multiple virtual connections (each one going to a different destination) as compared to the traditional point-to-point link, which supports a single connection.

Given the specific characteristics just described, a frame relay interface may have many more packets in transit than a conventional point-to-point link. Consequently, an acknowledged data link protocol whose procedures include retransmission of data frames is of limited use in this environment. For a large number of virtual connections, the memory required to store the data frames pending acknowledgment would be prohibitive. In addition, if frames are being discarded due to congestion in the frame relay subnetwork, the retransmission policy would increase, rather than recover from, this congestion. Instead, an unacknowledged data link layer should be used.

Using an unacknowledged data link protocol has implications for the routing layer operating over frame relay. In particular, the data link can no longer be considered reliable, and the routing protocol must accommodate this characteristic.

Congestion Avoidance

When a frame relay network becomes congested, network devices have no option but to drop frames once their buffers become full. With an unacknowledged data link layer, the user device will not be informed if a data frame is lost. This lack of explicit signaling when operating over frame relay networks places a requirement on the higher protocol layers in the end-system equipment. The OSI transport layer protocol demonstrates how to deal with this type of characteristic. The destination end system's transport implementation detects data loss and requests the source to retransmit the frame. The implementation reduces the source's credit to one, thus closing the source's transmit window and, in effect, reducing traffic through the congested path.

Frame relay networks are prone to congestion. Consider the scenario shown in Figure 3. Note that the committed information rate (CIR) represents minimum guaranteed throughput. In the configuration shown, the network device can support two PVCs: one running at 64 kilobits per second (kb/s) and the other at 128 kb/s. With no back pressure applied across the frame relay interface, in the worse case, the network device will become congested. The router can send frames into the network or a particular PVC at 1 Mb/s that will then be forwarded at a much slower rate. Once the network device's buffers are full, it will discard frames. As a result, routing and bridging control messages may be lost, thus causing the routing topology to become unstable. Since this, in turn, will likely lead to looping packets, a network meltdown could result.

In addition, if data frames are lost, the higher-layer protocols in the end system (e.g., the OSI transport layer) discover this situation and retransmit the lost frames. Repeated transmission of the

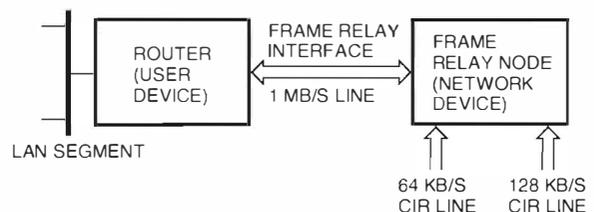


Figure 3 Example Configuration of Frame Relay Interface Rate and Permanent Virtual Circuit Throughput

same data causes the effective end-to-end throughput to drop well below the minimum guaranteed throughput.

The frame relay header has several mechanisms that can be used to apply the appropriate back pressure to prevent congestion.

- The FECN bit is set by the network when a frame experiences congestion as it traverses the network. In OSI and DECnet Phase V environments, this bit can be mapped onto the congestion-experienced bit in the header of the network layer PDU. This PDU, when subsequently delivered to the destination, allows the destination to discover that the path is congested and to notify the source transport to decrease its window and thus place less demand on the network. Standardization work is currently under way to add similar support to the transmission control protocol/internet protocol (TCP/IP).
- The BECN bit is set by the network when a frame traverses a congested virtual circuit in the opposite direction. This indicator is not perfect, because there is no guarantee that traffic will be generated in this direction on the virtual circuit. A source that detects it is transmitting on a congested path is expected to reduce its offered load.
- The DE bit, if set, indicates that during congestion the frame should be the first discarded. The procedures for deciding to set this bit are not clearly defined. This bit could be set by (1) the entry node of the network, e.g., when the input offered load is too high, or (2) the source user equipment, e.g., to discriminate data frames from the more important routing control messages.

Other methods can be used to avoid the consequences of congestion and hence frame loss. The LMI defined in the multivendor frame relay specification contained an optional extension that included a threshold notification bit in the PVC status information element of one of the messages. The threshold notification bit provided a means of allowing a network device to asynchronously inform a user device that a particular PVC connection was congested. The user device could then stop transmitting data on the connection until the network device informed it that the congestion was alleviated.

Since the loss of routing control messages can cause network instability, an alternative approach is to adopt manual configuration. Static network configurations use reachable addresses to provide routing information such that the transmission of routing control traffic is not required. Consequently, the routing behavior is independent of the performance of the network.

In addition, the user device could implement rate-based transmission to ensure that virtual circuits are not congested. However, a means of notifying the user device of the CIR of a virtual circuit was included only as an optional extension in the LMI specification, and use of such a method would destroy one of the major benefits of frame relay, i.e., the capability to allocate bandwidth on demand.

In practice, network devices have limited internal buffering to store frames; this is reflected in the CIR assigned to PVCs. Consequently, data loss occurs if user devices consistently transmit data on a PVC faster than its associated CIR. Adequate procedures and CPLs that cope with congested situations have yet to be developed and standardized. As a result, such situations may lead to unfairness in a multi-vendor environment where those users who support congestion avoidance will lose bandwidth to those who do not.

Products

Below we describe examples of frame relay products: the StrataCom IPX FastPacket equipment, which provides the frame relay network; and Digital's DECNIS 500/600 and WANrouter 100/500, which support the frame relay service by accessing the interface as user equipment.

The StrataCom IPX FastPacket Product Family

The StrataCom IPX FastPacket product family can be used to build networks that support both circuit-mode voice and data as well as frame relay. Within the network, the StrataCom IPX FastPacket nodes communicate using a technique based on cell switching, which involves the transmission of small, fixed-length cells. Additional, high-level functions provide services on top of the basic transmission network. StrataCom uses a hardware-based switching technique resulting in very high-speed switching (100,000 to 1,000,000 cells per second). With such high throughputs and low delays, these networks have been used for carrying voice, video, and data traffic.

The StrataCom IPX FastPacket network is configured by network management to provide the required virtual circuits between users. The StrataCom cell switching mechanism adopts a single-cell format for the transmission of all types of information, with each cell containing addressing information. Routing tables within the network nodes use this addressing information to forward the traffic along the desired virtual circuit. Since in any particular connection the path used for the sequence of cells is always the same, cell ordering is maintained. Intelligent interfaces at the edge of the network provide the functions required for specific services such as voice and data.

Figure 4 illustrates the concept employed by StrataCom of building service-specific functions on top of a common cell switching technology. The figure shows examples of various types of external interfaces.

For the frame relay interface, StrataCom supports the optional features defined to address congestion. The IPX FastPacket node provides the optional explicit congestion indicators defined in the frame header, which are set based on averaging queues that build up in the IPX FastPacket nodes in the network. Support is also provided for the optional threshold notification feature defined as

part of the LMI; the actual threshold values, together with buffer configuration, can be configured by the network manager.

Frame Relay Support in Digital's Family of Multiprotocol Routers

Digital has provided frame relay support in its family of multiprotocol routers that employ the OSI intermediate system-to-intermediate system (IS-IS) routing protocol. Frame relay user device functionality is implemented in the DECNIS V2.1 software for either the DECNIS 500 or the DECNIS 600 hardware units, and in the DEC WANrouter V1.0 software for either the DEMSB or the DEMSA hardware units.

Part of the development of the frame relay support involved cooperating with StrataCom to produce a working frame relay specification. In particular, extensions were added to the LMI to provide appropriate congestion control procedures. Digital's software supports the Frame Relay Specification with Extensions, Revision 1.0, written by StrataCom and the relevant ANSI T1S1 standards.^{3,1,5,8} The software has been tested and is known to be compatible with the StrataCom IPX FastPacket 16/32 equipment with Frame Relay Interface Card Software.

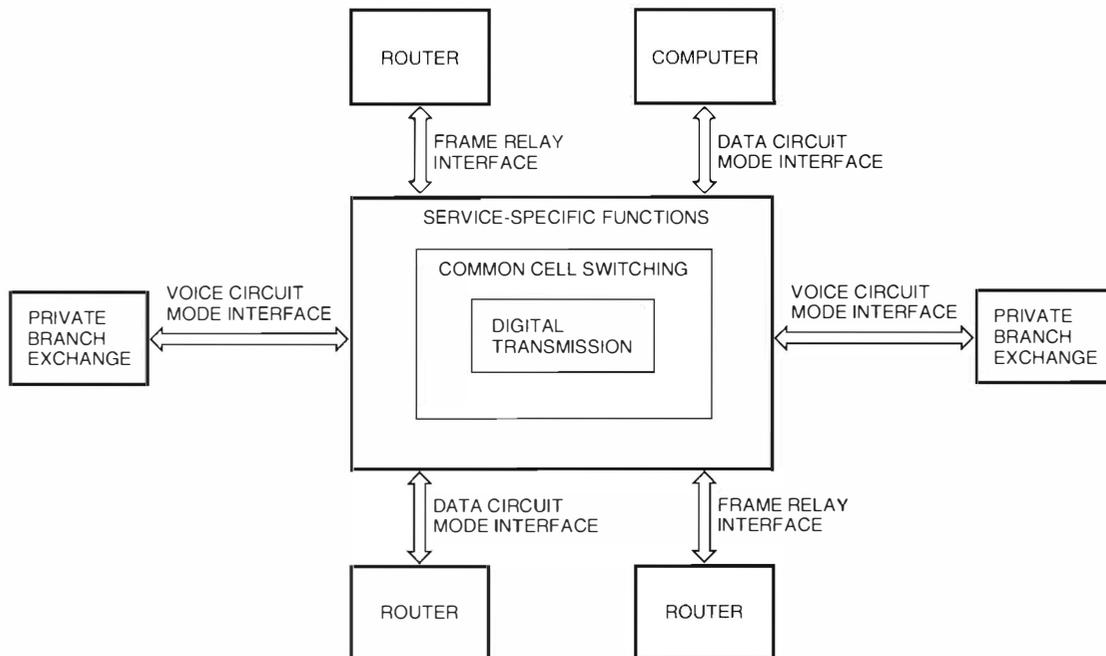


Figure 4 Sample StrataCom Network Configuration

The DECNIS and WANrouter implementations use the point-to-point protocol (PPP) for the transmission of multiprotocol datagrams over point-to-point links. PPP is defined in Requests for Comment (RFCs) 1331 and 1332, with bridging extensions specified in RFC 1220; support for DECnet Phase IV is defined in RFC 1376 and for OSI in RFC 1377.¹⁰⁻¹⁴ Congestion avoidance procedures include support for both the threshold notification signal in the LMI (when available) and the FECN. The threshold notification signal causes the end system to modify its rate of data transmission. Receipt of a frame with the FECN bit set causes the equivalent bit in the network layer PDU header to be set, which in turn causes the end systems to reduce their offered traffic. The BECN and DE bits are never set or examined.

Related Activities

Various committees are involved in activities related to the frame relay technology. These activities include standards work, specifications, and efforts to address technical issues such as interoperability.

Standards

The overall frame relay network architecture is defined in *ANSI T1.606, Frame Relay Bearer Service—Architectural Framework and Service Description*.¹ Access is provided by the frame relay interface, which is defined in various ANSI standards for both permanent and switched virtual circuits. *ANSI T1.618, DSSI—Core Aspects of Frame Protocol for Use with Frame Relay Bearer Service* contains a definition of the protocol for exchanging

frames across the interface, as well as annexes concerned with local management (e.g., notification of PVC status).⁵ Although all implementations to date have focused on a PVC-based interface, SVC access is defined in *ANSI T1.617, DSSI—Signaling Specification for Frame Relay Bearer Service*.⁸ Each of these T1S1 standards has an equivalent CCITT recommendation, as shown in Table 1.

Other Current Activities

The Internet Engineering Task Force (IETF) is developing specifications for RFCs related to the frame relay technology. A specification called Multiprotocol Interconnect over Frame Relay defines an encapsulation mechanism for supporting multiple protocols over frame relay networks. To allow use of the simple network management protocol (SNMP), an experimental management information base (MIB) for frame relay DTEs is also under development.

To promote the frame relay technology, a Frame Relay Forum has been set up in both North America and Europe. A technical committee has been established to address issues related to the technology in terms of its interoperability and evolution in multi-vendor environments. This committee actively participates with the standards bodies and develops implementation agreements and interoperability test procedures. Work continues to define a network-to-network control interface, multicasting capabilities, multiple protocol encapsulation, and interworking with other technologies, such as

Table 1 Current Status of Frame Relay Standardization

Standard	ANSI	Status	CCITT	Status	Remarks
Architecture and SVC Description	T1.606	Standard	I.233	Standard	Replaces I.222
Congestion Management Principles	Addendum to T1.606	Standard	I.370	Standard	
Data Transfer — Core Aspects	T1.618	Standard	Q.922 (Annex A corresponds to T1.618)	Standard	Most important frame relay standard
Access Signaling for SVCs	T1.617	Standard	Q.933	Standard	
Management Procedures for PVCs	Included in T1.617 Annex D	Standard	Included in Q.933 Annex A	Standard	Concepts accepted in CCITT

the switched multimegabit data service (SMDS) defined by Bell Communications Research, Inc.¹⁵

The cell switching adopted by StrataCom within their network is expected to change over time to conform with emerging CCITT recommendations for broadband ISDN.¹⁶ These recommendations cover asynchronous transfer mode (ATM), which defines a standard cell structure and ATM adaptation layers (AALs) for particular higher-level functions.

Summary

Frame relay is a simplified form of packet-mode switching that, at least in theory, provides access to high bandwidth on demand, direct connectivity to all other points in the network, and consumption of only the bandwidth actually used. Thus, to the customer, the frame relay technology offers a reduction in the cost of transmission lines and equipment and improved performance and response time.

Routers connected to a frame relay network can consider the multiplexed, PVC interface as a set of point-to-point links. The special characteristics of a frame relay network require that special care be taken in selecting the data link protocols and in handling congestion.

Acknowledgments

The authors thank StrataCom, Inc. for providing significant input on cell switching technology and its use in their IPX FastPacket equipment. The authors would also like to acknowledge Cliff Didcock of the Computer Integrated Telephony Development Group who consulted in Digital's initial frame relay implementation.

References

1. *ANSI T1.606: Frame Relay Bearer Service—Architectural Framework and Service Description* (New York: American National Standards Institute, Inc., 1990).
2. *CCITT Recommendation X.25: Interface between Data Terminal Equipment (DTE) and Data Circuit-terminating Equipment (DCE) for Terminals Operating in the Packet Mode and Connected to Public Data Networks by Dedicated Circuit* (Geneva: International Telecommunications Union, 1988).
3. *Frame Relay Specification with Extensions, Revision 1.0*, Cisco Systems, Digital Equipment Corporation, Northern Telecom, Inc., and StrataCom, Inc. (September 1990).
4. *CCITT Recommendation V.35: Data Transmission at 48 Kilobits per Second Using 60-108 kHz Group Band Circuits* (Geneva: International Telecommunications Union, 1976).
5. *ANSI T1.618: DSS1—Core Aspects of Frame Protocol for Use with Frame Relay Bearer Service* (New York: American National Standards Institute, Inc., 1990).
6. *CCITT Recommendation Q.922: ISDN User-Network Interface Layer 3 Specification for Basic Call Control* (Geneva: International Telecommunications Union, 1991).
7. *CCITT Recommendation Q.931: ISDN Data Link Layer Specification for Frame Mode Bearer Services* (Geneva: International Telecommunications Union, 1991).
8. *ANSI T1.617: DSS1—Signaling Specification for Frame Relay Bearer Service* (New York: American National Standards Institute, Inc., 1990).
9. *CCITT Draft Recommendation Q.933: ISDN Signalling Specification for Frame Mode Bearer Services* (Geneva: International Telecommunications Union, 1991).
10. *Point-to-Point Protocol for the Transmission of Multi-protocol Datagrams over Point-to-Point Links*, Internet Engineering Task Force RFC 1331 (May 1992).
11. *The PPP Internet Protocol Control Protocol (IPCP)*, Internet Engineering Task Force RFC 1332 (May 1992).
12. *Point-to-Point Protocol Extensions for Bridging*, Internet Engineering Task Force RFC 1220 (April 1991).
13. *PPP DECnet Phase IV Control Protocol (DNCP)*, Internet Engineering Task Force RFC 1376 (November 1992).
14. *PPP OSI Network Layer Control Protocol (OSI NLCP)*, Internet Engineering Task Force RFC 1377 (November 1992).
15. *Bellcore TR-TSV-000772, Generic System Requirements in Support of Switched Multi-Megabit Data Service*, Bell Communications Research, Inc. (May 1991).
16. *CCITT Draft Recommendation I.121: Broadband Aspects of ISDN* (Geneva: International Telecommunications Union, 1991).

An Implementation of the OSI Upper Layers and Applications

Above the transport layer, the open systems interconnection (OSI) basic reference model describes several application standards supported by a common upper layer protocol stack. Digital's high-performance implementation of the upper layers of the protocol stack concentrates on maximizing data throughput while minimizing connection establishment delay. An additional benefit derived from the implementation is that, for normal data exchanges, the delivery delay is also minimized. The implementation features of Digital's two OSI applications—file transfer, access, and management (FTAM) and virtual terminal (VT)—include the use of common code to facilitate portability and efficient buffer management to improve performance.

The open systems interconnection (OSI) basic reference model defined in the International Organization for Standardization standard ISO 7498-1 specifies a layered protocol model consisting of seven layers.¹ By convention, the first four layers—physical, data link, network, and transport—are referred to as the lower layers.² These layers provide a basic communication service by reliably transferring unstructured user data through one or more networks. The remaining layers—session, presentation, and application—build on the lower layers to provide services that structure data exchanges and maintain information in data exchanges to support distributed applications. These three layers are known collectively as the upper layers.

This paper first gives an overview of the OSI upper layers and of two application standards—file transfer, access, and management (FTAM) and virtual terminal (VT). The discussion that follows concentrates on the features of Digital's implementation of the upper layers and the two applications, with emphasis on novel implementation approaches.

Summary of OSI Upper Layer Standards

The application-independent parts of the OSI upper layers are defined in the following standards:

- ISO 8326 and ISO 8327—Session Connection Oriented Service and Protocol
- ISO 8822 and ISO 8823—Presentation Connection Oriented Service and Protocol

- ISO 8824—Abstract Syntax Notation One (ASN.1)
- ISO 8825—Basic Encoding Rules (BER)
- ISO 8649 and ISO 8650—Association Control Service Element (ACSE)

This section gives an overview of the services defined in these standards. The later sections File Transfer, Access, and Management Implementation and Virtual Terminal Implementation discuss two application-specific standards.

Session Layer

The transport layer service facilitates the exchange of unstructured bytes (i.e., octets) of data. However, exchanges between components of a distributed application are often structured. The function of the session layer is to standardize some of the common exchanges by supplying services that add structure to the transport layer exchanges.

The session-connection-oriented service has the three phases typical of all connection-oriented services: connection establishment, data transfer, and connection release. All structuring of the data exchanges occurs in the data transfer phase and is accomplished by using either tokens or synchronization. Hence, the connection establishment and release phases are not discussed further in this paper.

Tokens are used to control which peer session user of a session connection is permitted to invoke a particular service or group of services. The session layer also provides services to exchange

tokens between peer session users. There are four types of tokens.

1. Data, for controlling half-duplex data exchanges
2. Release, for controlling which session user can initiate the release of a session connection
3. Synchronize-minor, for controlling the issuing of the minor synchronization service
4. Major/Activity, for controlling the issuing of major synchronization and activity services

For example, when the data token has been negotiated on a session connection, session data can be sent only by the end that currently has the token. Exchanging the data token between the session users provides a half-duplex data service.

The data transfer phase provides synchronization by allowing session users to insert major and minor synchronization points into the data being transmitted. Optionally, each direction of flow can have its own set of synchronization points.

Figure 1 illustrates a data exchange structured as a single dialog unit. A dialog unit begins at a major synchronization point and terminates either at a new major synchronization point or by the release of the session connection. Further structure is possible within the dialog unit by inserting minor synchronization points.

The session synchronization services allow applications to insert synchronization points into their data exchanges. These points are application specific. The session service also provides a resynchronization service to allow a session user to request its peer to resynchronize to an earlier synchronization point, for example, to a previous point in a file transfer.

Activities provide an additional structuring service. An activity represents a logical piece of work. At any moment in time, there is at most one activity per session connection. However, several activities can exist during the lifetime of a session connection, and an activity can span session connections. The synchronization services can be used with activities services.

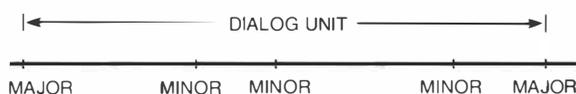


Figure 1 Data Exchange Structured as a Dialog Unit

Presentation Layer

Different computer architectures and compilers use different internal representations (i.e., concrete syntax) for data values. Therefore, conversion between representations is necessary when communicating between dissimilar architectures. The intent of the presentation layer is to allow communicating peers to negotiate the data representation to be used on a presentation connection.

The presentation standards, ISO 8822 and ISO 8823, distinguish between abstract syntax and transfer syntax. Abstract syntax is the definition of a data type independent of its representation. Typically, data types are defined using the ASN.1 standard, ISO 8824, which was developed for this purpose. ASN.1 has a number of primitive data types, including INTEGER, REAL, and BOOLEAN, as well as a collection of constructed data types, including SET and SEQUENCE OF. These primitive and constructed data types can be used to define the abstract syntax of complex data types such as application protocol data units.

A transfer syntax is the external communication representation of an abstract syntax. Values from the abstract syntax are encoded according to the rules defined in the transfer syntax. A common way to define a transfer syntax is in terms of encoding rules. For example, these rules may indicate how an INTEGER value is represented or how to encode a SEQUENCE OF data type. A widely used transfer syntax is the basic encoding rules specification, ISO 8825.

An abstract syntax can be encoded using different transfer syntaxes, of which there are many. The role of the presentation layer is to negotiate the set of abstract syntaxes to be used on a particular presentation connection and to select a compatible transfer syntax for each of these abstract syntaxes. This process ensures that both peers agree on the data representation to be used in data exchanges.

Application Layer

The application layer supports distributed interactive processing, that is, the communication aspects of distributed applications such as FTAM (defined by ISO 8571), directory service (defined by ISO 9594), and VT (defined by ISO 9040 and ISO 9041). Unlike for the session and presentation layers, numerous application layer protocols and services exist—at least as many as there are distributed applications.

The application layer structure specified in ISO 9545 defines a model for combining these

protocols in the same system. The functions for a particular application are grouped together to form an application service element (ASE). FTAM, VT, and directory service are examples of ASEs and are the basic building blocks of the application layer. One or more ASEs are combined to form an application entity (AE). An AE represents a set of communication resources and can be thought of as a program on a disk. An invocation of an AE (i.e., execution of the program) can contain one or more instances of an ASE with one or more application associations, i.e., application layer connections. The AE specification also defines the rules for interaction between ASEs operating over the same association as well as interactions between associations.

An ASE required by all applications is called the association control service element (ACSE). The ACSE, defined by ISO 8649 and ISO 8650, is the service and protocol required to establish an application association. Therefore, an AE always contains at least the ACSE.

An application association is mapped onto a presentation connection; no other application association can share this presentation connection. In this way, applications gain access to the presentation and session data phase services.

New OSI Upper Layer Implementation

Digital's implementation of the OSI upper layers, namely OSAK, includes session, presentation, and ACSE services. Users of OSAK can thus establish application associations and use session and presentation services during the data transfer phase.

Aims

In 1988, when Digital decided to produce a new version of OSAK, three aims were considered paramount: high performance, maintainability, and portability.

Performance High performance of the OSI upper layers is essential to producing competitive OSI products. Because all OSI applications use these upper layers, the performance of OSAK affects these applications. Therefore, OSAK aims to maximize data throughput and to minimize connection establishment delays. This improved performance is achieved by maximizing the use of the communication pipe and minimizing the local processing requirements. The process involves

1. Amalgamating upper layer state tables. The services provided by the presentation and session layers are similar. Also, connection establishment and release in the ACSE is basically the same as in the other two upper layers. Therefore, the three state tables can be combined into a single state table, thus improving performance by reducing the overhead. This amalgamation eliminates the need to manage links between state tables, requires all predicates to be tested in only one place, and generates only one state transition or action per inbound event.
2. Treating the presentation service P-DATA as a special case. The presentation service P-DATA is the most frequently used service, and hence, its performance has the greatest impact on data throughput. By fast-laning the processing of the P-DATA service, the normal overheads associated with the combined state table processing are avoided.
3. Good buffer management. The new application programming interface (API) to OSAK enables efficient use of buffers. We eliminated all copying of user data within OSAK by taking advantage of user buffers. On an outbound service, an OSAK user is requested to leave space at the start of the user data. If there is sufficient space, we add the OSI upper layer protocol control information (PCI) to the user buffer. This buffer is then sent to the transport provider. Otherwise, we allocate an OSAK-specific buffer using a user-supplied memory allocation routine.

Before receiving an inbound service, the user must pass at least one user buffer to OSAK. This buffer is used to receive the inbound transport event (both user data and upper layer PCI). The upper layer PCI is decoded before the user buffers are returned. In addition to being extremely efficient, this approach has the advantage of allowing OSAK users to exert inbound flow control; if OSAK is not given any buffers, no transport events will be received. Also, this buffering scheme simplifies resource management in OSAK. As OSAK does not have any of its own resources, they all come from OSAK users. One OSAK user cannot interfere with the operation of another OSAK user by consuming all OSAK resources.
4. Parsing only the upper layer headers. The presentation layer standards model the mapping

between concrete (internal) and transfer (external) representation of data values. In particular, the presentation state tables contain predicates to verify that all user data is from a current presentation context. Since the best place for encoding and decoding is in the application itself, OSAK does not implement these predicates. Rather, OSAK assumes that its users have correctly encoded their own protocol and will detect any problems when decoding.

5. Trading memory for performance. All encoding and decoding of upper layer PCI is done with in-line code. More compact coding is possible using subroutines but at the cost of performance.
6. Minimizing parameter checking. Most parameters are pointers to user buffers. To check the validity of all pointers is time-consuming and, consequently, costly. Therefore, OSAK assumes that the pointers do indeed point to the user's memory.

Maintainability The code for the new version of OSAK is easier to maintain than the previous code. As stated earlier in this section, a major step in improving the maintainability was the use of amalgamated state tables. A single state table eliminates links between tables, reduces the amount of maintenance required, and thus simplifies the code. In addition, using a single table makes it easier to serialize events. With multiple state tables, an inbound transport event can trigger a conflicting state change in the session state table at the same time a user request is changing the presentation state table. Using a single state table for a particular connection ensures that only one event (i.e., either a user or a transport event) is active in the state table at any given time.

The state tables are written in M4 macroprocessor notation. Thus, the OSAK state table definition is similar to an OSI protocol specification; this improves readability. Macros are also used extensively to handle common buffer manipulation and the encode and decode functions. Although macros are preferred over subroutines to improve performance, macros can be converted, at the expense of slower performance, should a more compact version of OSAK be required.

Portability The new version of OSAK is designed to facilitate portability of applications using both the OSAK API and OSAK itself. The new OSAK API is designed to be common across all platforms

and thus assists porting applications between platforms. The only major difference between the versions for the ULTRIX and the OpenVMS operating systems is the way events are signaled. The ULTRIX implementation supports both a polling model and an event-driven or blocking model. With the polling model, the OSAK user repeatedly calls OSAK routines to test for completion of an event; the routines used are `osak_collect_pb()` or `osak_get_event()`. In the blocking model, the OSAK user blocks awaiting the event, with the `osak_select()` routine.

These three routines are available to OpenVMS applications. In addition, the OpenVMS implementation supports event notification by asynchronous system traps (ASTs).

Also, the OSAK API is similar to XAP, the X/Open API to the OSI upper layers. To support OSAK on multiple platforms, as far as possible, OSAK code is common to all platforms. The main differences are the interface to the transport layer and the OpenVMS support for ASTs. Over 90 percent of the code is common to the ULTRIX and the OpenVMS versions.

Performance Measurements

Two performance metrics, throughput and connection establishment delay, were measured between two DECstation 3100 workstations connected by a lightly loaded Ethernet communications network. The DECstation machines were running ULTRIX V4.2 with DECnet-ULTRIX V5.1. OSAK accessed OSI transport through the X/Open transport interface (XTI) in nonblocking mode.

For throughput measurements, two programs were used: an initiator and a responder. The initiator

1. Establishes an association.
2. Reads the system time.
3. Transmits 2,000 buffers of data as quickly as possible. These user buffers contain sufficient space for the upper layer headers. When a send request fails due to flow control, the sender waits using the ULTRIX system call `select(2)` until the flow control is removed. The sender then collects the user buffers with the `osak_collect_pb()` routine before continuing with the send loop.
4. Reads the system time and calculates the time required to transmit the 2,000 buffers.
5. Releases the association.

The responder

1. Accepts an association request
2. Loops, waiting for a transport event using the ULTRIX system call `select(2)`, and then collects the data using the `osak_get_event()` routine until all 2,000 buffers have been received
3. Responds to the request to release the association

Table 1 records the throughput measurements for various buffer sizes ranging from 10 to 16,000 (16K) octets per buffer.

The data presented in Table 1 indicates that for small buffers, the throughput is poor. This low performance is due to the system overhead associated with processing a send request, independent of the amount of data to be transmitted. However, the throughput rapidly improves until the buffer size reaches 4K octets. From this size on, the throughput measurement is almost flat at between 507K and 528K octets per second. The variation is due to fragmentation in the lower layers. The number of send requests flow controlled represents the number of times a send request was delayed because of flow control by the transport service in the course of transmitting the 2,000 buffers.

We profiled the initiator and the responder. For buffers ranging in size from 10 to 16K octets, the initiator spent more than 90 percent of the time in transport. For the responder, the percent of time spent in transport varied between 60 percent for 10-octet buffers and 92 percent for 8K-octet buffers. The remaining time was spent primarily in `select(2)`, waiting for and processing the next

inbound event. Also, for the small buffers, a significant amount of time is consumed by initializing the user parameter block before returning it to the user.

We also used the throughput program to measure the connection establishment time. The program read the system time before and after the association establishment phase; the average connection establishment time was 0.08 seconds. In addition, tests on the new OpenVMS implementation indicate that throughput improved two to three fold as compared to the OSAK code in the previously existing OpenVMS implementations.

Both the throughput and profile data indicate that the transport performance dominates the performance of OSAK. Therefore, OSAK has met its design goal of reducing the overhead of the OSI upper layers to a very low level. Meeting this goal was necessary because poor OSAK performance would impact all OSI applications supported by OSAK. While further reductions in overhead are possible, such savings would be at the expense of OSI upper layer functionality.

File Transfer, Access, and Management Implementation

This section presents a summary of the ISO FTAM standard and details of Digital's implementation of this standard.

Summary of the ISO FTAM Standard

ISO 8571 File Transfer, Access, and Management (FTAM) is a five-part standard consisting of a general introduction, a definition of the virtual file store, the file service, the file protocol definitions, and the protocol implementation conformance statement proforma. The FTAM standard defines an ASE for transferring files and defines a framework for file access and file management.

Initiator and Responder FTAM service and protocol actions are based on a client-server model. In the FTAM standard, the client is referred to as the initiator, and the server is referred to as the responder.

The initiator is responsible for starting file service activity and controls the protocol actions that take place during the dialog (or FTAM association) between two FTAM applications. For example, the initiator has to request that an FTAM association be established, that a file be opened on a remote system, and that a file be read from a remote system.

The responder passively reacts to the requests of the peer initiator. The responder is responsible for

Table 1 Throughput Measurements for Digital's OSI Upper Layer Implementation

Buffer Size (Octets)	Throughput (Kilooctets/s)	Number of Send Requests Flow Controlled
10	6.60	2
100	56.80	4
512	216.00	35
1,024	266.60	794
2,048	372.60	862
4,096	453.70	1,151
6,000	507.00	1,217
8,124	528.80	596
8,125	507.10	651
10,000	527.20	751
13,000	522.20	1,101
16,000	505.27	1,279

managing the virtual file store and mapping any virtual file attributes into local file attributes.

Virtual File Store Many architectures and implementations of file systems exist, and storing and accessing data can differ from one system to another. Therefore, a mechanism is needed to describe files and their attributes independent of any particular architecture or implementation. The mechanism used in the FTAM is called the virtual file store. The FTAM virtual file store model consists of file attributes, activity attributes, file access structure, and document types.

File attributes describe the properties of the file, which include the size and the date of creation. FTAM file attributes also define the types of actions that can be performed on a file. Read access or create access are examples of file actions.

Activity attributes are properties of the file, which are in effect for only the duration of the FTAM association. Examples of activity attributes are current access request, current initiator identity, and current concurrency control. Current access request conveys the access control applied to the file, e.g., read or write access. Current initiator identity conveys the name of the initiator accessing the virtual file store. Current concurrency control conveys the status of the locks applied by the initiator.

The FTAM file access structure is hierarchical and produces an ordered tree that consists of one or more nodes. This file access structure is defined in ASN.1 and can be used to convey the structure of a wide variety of files.

In the FTAM virtual file store model, document types specify the semantics of a file's contents. The FTAM standard defines four document types.

- FTAM-1, unstructured text files
- FTAM-2, sequential text files
- FTAM-3, unstructured binary files
- FTAM-4, sequential binary files

The virtual file store model provides a framework for defining many different file types, including those not supported by the standardized document types. The U.S. National Institute of Standards and Technologies (NIST) has used the virtual file store model to define document types to support various file types, such as indexed files.

FTAM File Service The FTAM file service is a functional base for remote file operations. Functionality defined by the FTAM file service is broken down

into subsets of related services. The subsets of functionality are called functional units. Functional units are used by the FTAM protocol to convey a user's requirements. For example, the standard defines the read functional unit, which allows an implementation to read whole files, and the file access unit, which allows an implementation to access records in the file.

In addition, the FTAM standard defines the following classes of file service: transfer, management, transfer and management, access, and unconstrained. Each service class is composed of a set of functional units. For example, an FTAM implementation that supports the transfer service class will be able to either read or write files.

New FTAM Standard Work Modifications to the FTAM standard are in progress in the ISO. The most important modification is the file store management addendum, which specifies how wild cards, file directories, and references (links) to files are to be handled in an OSI environment. The addendum also specifies how to manipulate groups of files. In the current version of the standard, only one file can be selected at a time.

Digital's FTAM Implementation

Digital's FTAM products, available for the OpenVMS and ULTRIX operating systems, support FTAM applications in both the role of initiator and the role of responder. The initiator applications allow users to copy, delete, rename, list, and append files. In the OpenVMS version, the initiator applications are integrated into the Digital Command Language (DCL) so that the user can continue to use the COPY, DELETE, DIRECTORY, and RENAME commands. Where the FTAM service and protocol is used to support these commands, the additional qualifier /APPLICATION=FTAM is required. In the ULTRIX version, the same functionality is provided using the set of commands ocp, orm, ols, ocat, and omv. These commands have the same semantics as the corresponding ULTRIX commands cp, rm, ls, cat, and mv, respectively, and are similar to the set of DECnet file transfer utilities of dcp, drm, dls, and dcat. (Note that the set does not include dmv.)

The responder applications allow users to create, read, write, delete, and rename files. File access, i.e., the location of specific records in a file, is also supported by the responder applications. The OpenVMS responder application supports file locking and recoverable file transfer.

Digital's initiator and responder applications support the following FTAM document types:

- FTAM-1
- FTAM-2
- FTAM-3
- NBS-9, FTAM file directory

Programmatic Interface The FTAM API is common across all platforms and shares a "look and feel" with the OSAK API. The FTAM API allows access to all FTAM services and parameters through the use of a single parameter block and five library calls.

- `osif_assign_port()`
- `osif_deassign_port()`
- `osif_getevent()`
- `osif_send()`
- `osif_give_buffers()`

The FTAM API can be used to create either initiator or responder applications.

Protocol Gateways Digital's FTAM products support two protocol gateways: an FTAM/file transfer protocol (FTAM/FTP) gateway is available on the ULTRIX version, and an FTAM/data access protocol (FTAM/DAP) gateway is available on the OpenVMS version. The FTAM/FTP gateway supports bidirectional protocol translation. Files on internet hosts can be accessed through the gateway using FTAM; files on OSI hosts can be accessed through the gateway by using FTP.

Implementation Features Portability, maintainability, and performance were the major goals of the FTAM implementation. To achieve these goals we

1. Created a common code base. The code is implemented using the C programming language. The FTAM protocol machine and the initiator and responder application programs are implemented such that a large amount of the code can be used across multiple platforms. These modules are referred to as common code modules. Any system-specific code, which represents 90 percent of the code, is placed in system-specific modules. All other modules are common to both the ULTRIX and the OpenVMS versions.

2. Hid interface dependencies from FTAM. To aid in the porting of code to different platforms, the FTAM implementation makes no direct calls to system-specific interfaces.
3. Provided good buffer management. The FTAM implementation uses the same buffer management model as OSAK, described earlier in the section New OSI Upper Layer Implementation.

Virtual Terminal Implementation

Digital also implemented the OSI virtual terminal application standards. Details of the standards and features of the implementation follow.

Summary of the VT Standards

ISO 9040 and ISO 9041 are the two international standards that define the OSI virtual terminal. ISO 9040 is concerned primarily with specifying a model for a virtual terminal basic class service; ISO 9041 defines the protocol to be used.

OSI virtual terminals are divided into five classes, based on functionality.⁴

1. Basic—data consisting of rectangular arrays of characters
2. Forms—data consisting of characters arranged in fields of variable size and shape, with the manipulation of content controllable for each field
3. Text—data representing document structures as covered by the Office Document Architecture standards (ISO 8613 series)
4. Image—data representing images composed of arrays of dots, i.e., pixels
5. Graphics—data representing computer graphics elements, such as lines and circles

To date, most of the work within the ISO has concentrated on the basic terminal class, i.e., basic class virtual terminal (BCVT). An OSI virtual terminal implementation provides a mechanism that allows a user to interactively access another OSI system, when not directly connected to it. Since a variety of systems and terminals exist that are not necessarily compatible with each other, the ISO VT protocol provides a means by which dissimilar terminals and systems may interact.

An example of a dissimilar terminal and system interacting by means of a VT would be the action of deleting a typed character. Some systems expect the terminal user to enter the <delete> character

as an indication of the intent to delete, whereas other systems may expect the user to enter a <backspace> character. VT resolves these differences by translating the local action into a virtual action. The action in our example becomes the virtual actions of decrementing the current cursor position and erasing the character at the current location. A cooperating implementation would then translate these virtual actions into an appropriate local action.

The VT protocol is very powerful in the respect that the protocol definition provides many options and features that allow the support of complex terminal models. During association establishment, cooperating implementations agree on the subset of the protocol and the terminal model to be used. The protocol subset and terminal model are referred to as the profile. In addition, VT provides two modes of operation: asynchronous (A-mode), which may be thought of as full-duplex operation, and synchronous (S-mode), which may be thought of as half-duplex operation.

The ISO base standards define two basic profiles, one for each mode. Additional profiles have also been defined (or are being prepared) by the regional OSI workshops. Currently, the OpenVMS and ULTRIX implementations of the VT protocol both support the following profiles:

1. TELNET-1988, which mimics the basic functionality found in the transmission control protocol/internet protocol teletype network (TCP/IP TELNET) environment
2. Transparent, which allows the sending and receiving of uninterpreted data
3. A-mode-default, which provides basic A-mode functionality

Digital's VT Implementation

Digital's VT implementation provides both initiator and responder capabilities. In addition to describing the features of the implementation, this section compares the VT protocol with other network terminal protocols.

Initiator and Responder The VT implementation for both the ULTRIX and the OpenVMS systems provides the capability to act as either an initiator (a terminal implementation) or a responder (a host implementation). The initiator is responsible for establishing an association with the responder based on information provided by the user, such as

the desired profile. The responder is responsible for accepting the peer association request and for creating an interactive context for the remote peer user.

On the OpenVMS system, the VT protocol initiator is invoked by the DCL command SET HOST/VTP; on the ULTRIX system, the VT protocol initiator is invoked using the ologin command.

Implementation Features The VT implementation uses the OSAK interface outlined earlier in the paper. The goals of the VT implementation were to provide a highly portable, very efficient, and easily extensible code.

To achieve the goal of portability, the implementation was divided into two major components: interface to the OSI environment and the non-OSI interfaces (e.g., to terminals). The OSI component is completely portable to multiple platforms. The non-OSI component is platform specific and must be rewritten for each unique platform. The interface between these components consists of six basic functions, which must be supported on all platforms.

- Attach/detach—to attach and detach the non-OSI environment
- Open/close—to open or close a specific connection into the non-OSI environment
- Read/write—to read or write data between the OSI and the non-OSI environments

Because each function is simple and clearly defined, the amount of platform-specific code required for implementation is minimal. For example, the read function on the ULTRIX implementation is only 10 lines of code. The implementation is therefore highly extensible to different platforms.

Performance of the VT protocol implementation is enhanced by using preallocated buffer pools. This approach to buffer management eliminates the overhead of dynamically allocating buffers.

Our VT protocol implementation not only implements the ISO VT protocol but also provides a gateway to and from other terminal protocol environments. We provide gateways to TELNET and to the Local Area Transport (LAT) on both the OpenVMS and the ULTRIX versions. In addition, we have a VT/command terminal (VT/CTERM) gateway on the ULTRIX version.

Comparison of the VT Protocol with Other Network Terminal Protocols Most comparisons with network terminal protocols deal with echo

response time, that is, how long it takes for a character to echo to a display after being typed at the keyboard. VT, like TELNET and CTERM, can operate in two different echo modes: remote, where the echo is achieved by means of the remote host; and local, where the echo is accomplished through the local host. A number of factors contribute to response time in a remote echo situation, including protocol overhead and line speed. TELNET has little protocol overhead; in fact, for most situations, transferring normal data requires no additional overhead. VT protocol overhead is approximately 30 to 1 for a typical A-mode profile, that is, 30 octets are required to carry 1 octet of user data. VT overhead may seem excessive when compared with TELNET. However, the VT protocol provides many additional capabilities that TELNET does not, such as the ability to accurately model different terminal environments. Additionally, the 30 octets of overhead does not increase significantly when larger amounts of user data are transferred.

The largest gains for the VT are in the area of S-mode profiles. S-mode profiles enable most character echoing to be done locally. By using an appropriate S-mode profile, the VT implementation can provide sophisticated local terminal operations. Thus, it is possible to edit an entire screen of text and then to transmit it all at once to the remote host. The ability to process large amounts of terminal input as batch jobs has many advantages, including reduced network bandwidth requirements, reduced CPU requirements of the remote host (since the remote host is no longer involved in character echo), and increased user satisfaction (since users experience no network delays for character echo).

Summary

Goals common to the OSAK, FTAM, and VT protocol projects included good performance and portability of implementation. Performance is especially important for OSAK, because it supports all other OSI applications. Maximizing the use of common code and reducing system dependencies in the three projects significantly reduced the engineering effort to port an implementation from one platform to another. This savings in human resources is necessary, given the growing set of hardware and operating platforms supported by Digital. Equally important is the integration of OSI applications with their non-OSI counterparts, for example, the ocp and ologin functions and the protocol gateways.

Acknowledgments

The authors would like to thank their colleagues for reviewing previous drafts of this paper. In particular, we would like to thank Chris Gunner and Nick Emery, who were instrumental in revising the OSAK API, and the OSAK team, who converted the advanced development code into the product.

References

1. J. Harper, "Overview of Digital's Open Networking," *Digital Technical Journal*, vol. 5, no. 1 (Winter 1993, this issue): 12-20.
2. L. Yetto et al., "The DECnet/OSI for OpenVMS Version 5.5 Implementation," *Digital Technical Journal*, vol. 5, no. 1 (Winter 1993, this issue): 21-33.
3. P. Lawrence and C. Makemson, "Guide to ISO Virtual Terminal Standards," Information Technology Standards Unit (UK), Department of Trade and Industry (March 1988).

General References

Information Processing Systems, Open Systems Interconnection, Part 1: Basic Reference Model (International Organization for Standardization, reference no. ISO 7498-1, 1984).

Information Technology, Open Systems Interconnection: Connection Oriented Session Service Definition (International Organization for Standardization, reference no. ISO 8326, 1987).

Information Technology, Open Systems Interconnection: Connection Oriented Session Protocol Definition (International Organization for Standardization, reference no. ISO 8327, 1987).

Information Processing Systems, Open Systems Interconnection, File Transfer, Access, and Management: Part 1, General Introduction; Part 2, Virtual File Store; Part 3, File Service Definition; Part 4, File Protocol Specification; and Part 5, Protocol Implementation Conformance Statement Proforma (International Organization for Standardization, reference no. ISO 8571, 1988).

Information Processing Systems, Open Systems Interconnection: Service Definition for the Association Control Service Element (International Organization for Standardization, reference no. ISO 8649, 1988).

Information Processing Systems, Open Systems Interconnection: Protocol Specification for the Association Control Service Element (International Organization for Standardization, reference no. ISO 8650, 1988).

Information Processing Systems, Open Systems Interconnection: Connection Oriented Presentation Service Definition (International Organization for Standardization, reference no. ISO 8822, 1988).

Information Processing Systems, Open Systems Interconnection: Connection Oriented Presentation Protocol Specification (International Organization for Standardization, reference no. ISO 8823, 1988).

Information Processing Systems, Open Systems Interconnection: Specification of Abstract Syntax Notation One (ASN.1) (International Organization for Standardization, reference no. ISO 8824, 1987).

Information Processing Systems, Open Systems Interconnection: Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1) (International Organization for Standardization, reference no. ISO 8825, 1987).

Information Technology, Open Systems Interconnection: Virtual Terminal Basic Class Service (International Organization for Standardization, reference no. ISO 9040, 1990).

Information Technology, Open Systems Interconnection: Virtual Terminal Basic Class Protocol (International Organization for Standardization, reference no. ISO 9041, 1990).

Information Processing Systems, Open Systems Interconnection: Application Layer Structure (International Organization for Standardization, reference no. ISO 9545, 1989).

Network Management

DECnet/OSI Phase V incorporates a new network management architecture based on Digital's Enterprise Management Architecture (EMA). The EMA entity model was developed to manage all entities in a consistent manner; structuring any manageable component regardless of its internal complexity. The DNA CMIP management protocol was developed in conjunction with the model to express the basic concepts in the entity model. Phase V network management is extensible; the Phase V management architecture transparently assimilates new devices and technologies. Phase V was designed to be an open architecture. Management of DECnet/OSI Phase V components is effective in a multivendor network.

Network management has been an integral part of DECnet since 1976 when Phase II was developed.¹ Even at that early stage of the DECnet architecture, an effective management capability was recognized as an essential part of an organized approach to networking. Now in DECnet Phase V, the DECnet network management architecture has undergone a major revision based on Digital's Enterprise Management Architecture (EMA). This paper gives an overview of some of the key features and functions of EMA and of DECnet Phase V network management. See the "Overview of Digital's Open Networking" paper in this issue for an overview of the guiding principles, background, and architecture of DECnet Phase V.²

Our initial work on Phase V indicated that changes were needed in the network management architecture to support the broad range of networking functions planned for Phase V. First, network managers would have to be able to manage all the Phase V components in a consistent manner. A method was needed to build Phase V management components that would give the same general look and feel and the same modeling approach to all components.

Second, Phase V network management would have to be extensible. The Phase V network architecture was being designed to allow the use of multiple modules that would provide the same or similar services at each layer and to simultaneously support multiple-layer protocols in a network. Therefore, we designed the Phase V management architecture to transparently assimilate new devices and technologies. Our management archi-

ture had to become as extensible as the network architecture.

Finally, since Phase V was designed to be an open architecture, management of Phase V components would have to be effective in a multivendor network. Our design had to ensure that the ability to provide effective management of network components was independent of the vendors supplying them.

The individual management mechanisms used in Phase IV could have been extended to accommodate all the changes planned for Phase V. However, we felt it was time to revisit the basic network management architecture to see if we could find a unified approach that would provide a superior solution.

Enterprise Management Architecture

We began our Phase V development project by examining in detail the requirements for a new network management architecture. Our goal was to design an open architecture that allowed for consistent management of an extensible array of network components in a multivendor environment. As we identified the specific requirements that would have to be addressed to meet this goal, we realized that we had the opportunity to develop an architecture that went beyond management of Phase V networks. We realized that we could provide an architecture for the management of both networks and systems. The architecture eventually became known as the Enterprise Management Architecture or EMA.

Early in the project, we recognized that the conceptual separation of manageable components from the software that manages them was a fundamental design principle. EMA therefore distinguished *entities*, the basic components of the network that had to be managed, from *directors*, the software systems and accompanying applications used by managers to manage the components, as shown in Figure 1.

Formally, an entity was further split into a service element, a managed object, and an agent. The service element is the portion of the entity that performs the primary function of the entity, e.g., a data link layer protocol module whose primary purpose is communication with a peer protocol module on another machine. The managed object encapsulates the software that implements the functions supported by the entity for its own management. For example, it responds to management requests for the current values of state variables or to requests for the values of certain configuration variables to be set to new values. The agent is the software that provides the interface between the director and the managed object. The agent encodes and decodes protocol messages it exchanges with the director and passes requests to and receives responses from the managed object.

Informally, we generally equate the managed object and the entity because the managed object defines what the manager can monitor and control in the entity.

A director was modeled as a layered software system that provides a management-specific environment to management applications. A director was split into a framework, a management information repository (MIR), and separate configurable software modules called management modules. The director kernel provides common routines useful for the layered software modules, including

services such as dispatch (location-transparent exchange of management requests and responses with entities), encoding/decoding, data access, data dictionary access, and event management. Taken together, the director kernel and the agent provide a framework for managed objects and management applications to interact. The framework provides an application programming interface (API) to managed object and management module developers. The MIR contains data about particular entities as well as information about the structure and other properties of entity classes, which the director software also knows.

Management modules were distinguished as presentation, function, or access modules. Presentation modules implement user or software access to the director management modules that is device independent and style dependent. Function modules provide value-added management functions that are partially or completely entity independent, such as network fault diagnosis, event or alarm handling, or historical data recording. Access modules provide a consistent interface to the basic management functions performed by entities. In addition, they include one portion that maps operations on entities into the appropriate protocol primitives and another portion that implements the protocol engine for the relevant management protocol. Figure 2 shows the components of a director and an entity.

Although users can conveniently interact with systems through graphical user interfaces (GUIs), sophisticated users wished to preserve a command line interface (CLI) they could use to specify complex management requests quickly. Therefore, we

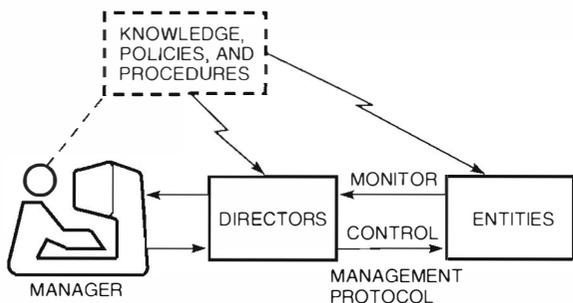


Figure 1 The Basic Entity/Director Split

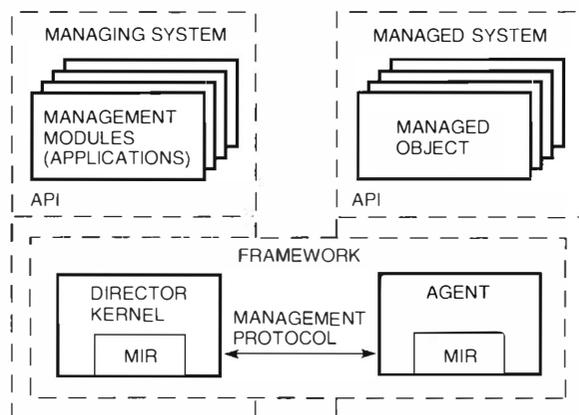


Figure 2 A Framework View of EMA

developed a single, extensible command language that would allow human operators or software programs to communicate requests to management modules and (ultimately) entities in a consistent fashion. This work developed into the network control language (NCL). An NCL command specifies an entity, an operation to be performed by the entity, a list of arguments (if any), and a list of qualifiers (for specifying users, passwords, paths, filtering values, etc.).

Digital's DECmcc Management Director is an implementation of an EMA director.³ The DECmcc product provides a platform for the development of new management capabilities and offers specific Phase V management capabilities as well as a number of generic network management tools. The DECmcc director supports both GUI and NCL CLI user interfaces.

Entity Model

To manage all entities in a consistent manner, we required a single, consistent method for structuring any manageable component (regardless of its internal complexity) and for describing its management properties: the operations that it can perform, the variables it makes available for its management, the critical occurrences it can report to managers, etc. The EMA entity model was developed to answer these needs. The structure of a manageable component in this model is shown in Figure 3. Essentially, the entity model defines techniques for specifying an object-oriented view of an entity. Each entity has the following properties:

- A position within an entity hierarchy. To ease management of networks with large numbers

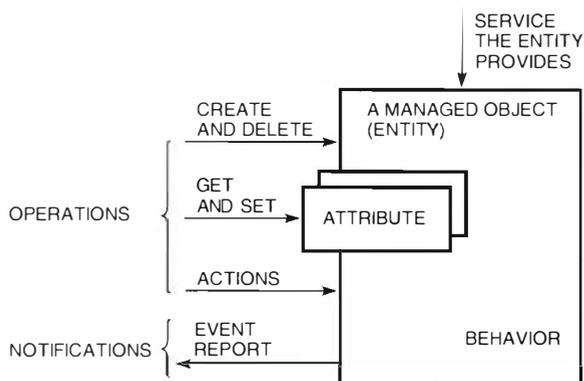


Figure 3 Structure of a Managed Object

of complex components, entity classes are organized into logical structures that reflect the relationship of their corresponding components; individual entities are named in terms of that structure. The name of the top-level entity in each structure is globally unique, and it is referred to as a global entity. All its child entities, however, have names that are unique only within the context of their level in the structure. Therefore, they are referred to as local entities.

- A hierarchically structured name. An individual entity's local name is constructed by concatenating its class name to its instance identifier. The class name is a keyword that uniquely identifies the class (object type) of an entity. The instance identifier is the value of an identifying attribute used for naming instances of the entity's class, for which each instance of the class has a unique value.

A target entity's globally unique name is constructed by concatenating its local name (a <class name, instance identifier> pair) to the local names of each of its ancestors in turn, beginning with the containing global entity and ending with the target entity's immediate parent. The construction of an entity's name and the containment hierarchy are shown in Figure 4.

- A collection of internal state variables, called attributes, that can be read and/or modified as a result of management operations. Attributes have names unique within the context of the entity. Attributes have a type that defines the values the attribute can have.
- A collection of operations that can be performed by the entity. Operations allow managers to read attributes, modify attributes, and perform actions supported by the entity. Actions are entity-specific operations that result in changes of state in the entity or cause the entity to perform an operation that has a defined effect.
- A collection of events that can be reported asynchronously by the entity. An event is some normal or abnormal condition within an entity, usually the result of a state transition observed by its service element or its agent. Event reports are sent asynchronously to the manager; they indicate the type of (entity-specific) event that occurred and may also contain arguments that

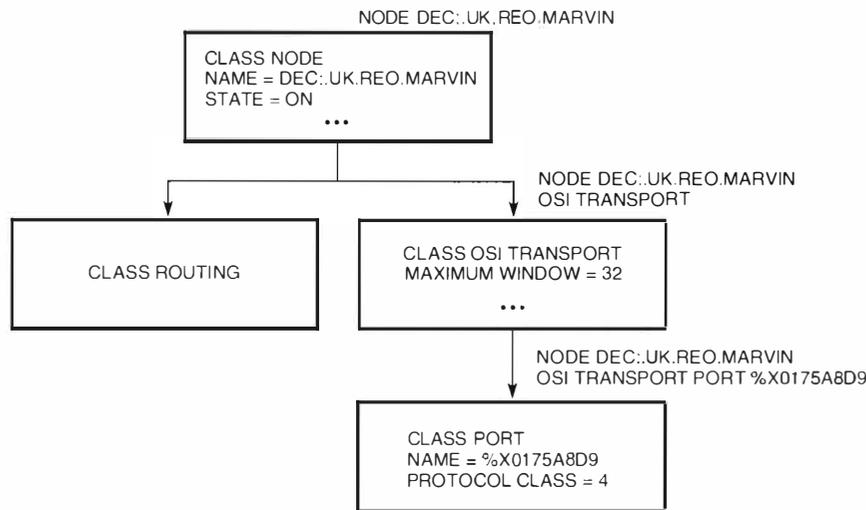


Figure 4 Managed Object Naming Hierarchy

further describe or qualify the event. For example, arguments could indicate the number of times the event occurred before a report was sent to announce that a threshold was reached, or give the old and new states in an event that reports a state transition.

- A specification of the behavior of the entity in relationship to the functions that the entity's service element provides. This is usually specified as some abstract state machine, through pseudocode, or as a set of preconditions, postconditions, and invariants.

The entity model provides specific requirements and recommendations about the way entities can be modeled in terms of these properties. These restrictions, placed on entity class definitions for purposes of both internal and global consistency, take several forms: (1) restrictions on the types and ranges of attributes that can be used for various purposes (e.g., as identifying or counter attributes); (2) constraints on operations (e.g., examine operations can have no side effects on the value of attributes whose values they report); or (3) restrictions on events (e.g., all events and event reports must have an associated time stamp and unique identifier).

Readers familiar with open systems interconnection (OSI) management will find the entity model very similar to OSI's structure of management information (SMI) standard.^{4,5} This is no coincidence. During the early development of Phase V and the entity model, we recognized the need for an open management architecture. Portions of the technol-

ogy were therefore contributed to ISO/IEC JTC 1 SC21/WG4, a working group of the International Organization for Standardization (ISO) that is responsible for efforts to define standards for OSI management. Although some details of OSI SMI and the corresponding EMA features diverged slightly from each other during their evolution, the EMA entity model and OSI SMI are still compatible. At this writing, work is under way to align certain parts of the EMA entity model with the final international standard (IS) versions of OSI SMI.

Entities

The EMA entity model describes how to specify the management of an architected subsystem. However, for Phase V, we chose to make the management specification of a subsystem a part of the subsystem's specification. As described in the Modules section, that may have been the most important decision made in the network management architecture.

As the entities for DECnet/OSI Phase V were defined, a collection of folklore grew on how typical design issues could or should be solved. As with any folklore, these guidelines were passed from one architect to another, either verbally, or as selected portions of the management specifications were copied from one subsystem to another. This folklore is continually changing, as new and better solutions are found. Much of the folklore has already been described.⁶ Some other guidelines are described below.

The Network Management Specification describes the central structure of Phase V network management, and in particular defines the node entity class.⁷ In the following sections, we describe the properties of the node entity class and, as a representative example, the OSI transport module entity class.

Node Entity Class

A single computer system in the DECnet/OSI network is called a node. The bounds of that system depend on the system's architecture; a personal computer (PC), a single-processor workstation, a multiprocessor mainframe, a diskless system, even a VAXcluster system can be considered a single node. Nodes are modeled by the node entity class.

A node entity has only a few functions in management.

- A node is a global entity that is the parent for many subsystems and provides an agent for all of them.
- A node has an identity, a name, and an address that allow it to be managed remotely.
- A node plays a major role in system initialization and start-up.

Identity

The following attributes identify a node:

- An address, the application layer address(es) of the node's agent
- A name, a DECdns fullname as defined by the DECnet/OSI distributed name server⁸
- A synonym, a Phase IV-style node name for backward compatibility
- A spatially unique identifier (ID), a 48-bit quantity used as an Institute of Electrical and Electronics Engineers (IEEE) 802 local area network (LAN) or Ethernet address
- A space- and time-unique value

A node's address is the application layer address(es) of the node's agent. The DECnet/OSI network supports multiple protocols at any of the seven layers, and the agent can operate over multiple protocol stacks. Each protocol has its own addressing conventions. Thus a node's address is actually a set of protocol towers. Each tower defines a sequence of protocols, each with its associated addressing information. A protocol tower

provides all the information needed by a director to connect to the node's agent and to issue management directives to the node or any of its children.

Users and network managers rarely refer to nodes by their addresses. First, it is difficult to remember the addresses and second, moving the node from one place to another in the network generally changes its address. Thus each node has a name, a DECdns fullname. The node knows its name and address. Each node's name is stored as a DECdns entry, and one of the entry's DECdns attributes holds the node's address. Thus, any director can look up the node's name in the DECdns and the address associated with it, and then use any one of the towers to connect to the node's agent.

To ensure backward compatibility with DECnet Phase IV, a node also has an attribute called its synonym, which is a six-character, Phase IV-style node name. If a node has a synonym name, that name is entered in a special directory in the DECdns name space as a soft link to the node's Phase V name. A soft link is a form of alias or indirect pointer, from one name to another, that allows an entry to be reached by more than one name.

Each network layer address of the node (a node can have more than one) is encoded in a standard way as a soft link to the node's name. This allows a manager (or director) to translate a node address into the equivalent node name, making many diagnostic problems much simpler.

DECnet/OSI includes many features that allow most nodes to autoconfigure their addresses. Network layer addresses consist of an area address and a 48-bit ID. This ID can be obtained from an ID read-only memory (ROM) chip on many devices (for example, each Digital 802.3 LAN device has one). End nodes detect area addresses from messages sent by the routers adjacent to the end node. Higher-level addresses used by management are architecturally defined constants.

Managers and users choose the name and synonym of a node. The manager uses the rename action to tell the node its name. Rename is an example of a situation in which an action is more appropriate than a set operation. Renaming a node is a fairly complicated operation. Not only is the name attribute changed, but also the information is stored in the DECdns name space. Although the operation can fail in many ways, actions allow errors to be reported to the manager with enough detail on what went wrong to allow corrective action to be taken. This is not easily done with a set operation.

One of the more difficult configuration problems to track down occurs when two nodes in a network have either the same name or the same address. DECnet/OSI has several management features to prevent this from occurring or to detect the situation when it does occur.

First, each node has a spatially unique 48-bit ID, i.e., no two nodes in the enterprise have the same ID at the same time. The ID is usually derived from an ID ROM chip in a LAN adapter. Special manufacturing procedures ensure that no two ID ROMs hold the same ID. Nodes with multiple ID ROMs, for example a router with two Ethernet interfaces, choose one with a simple algorithm. Nodes without an ID ROM must be assigned an ID when the system is first booted, and that ID must come from the locally administered IDs. However, an ID is not always tied to the same node. Hardware devices can be removed from one machine and inserted in another. Indeed, this is a common diagnostic procedure.

Second, each node has a space- and time-unique value provided by the unique identifier (UID) service. UIDs combine a spatially unique ID with a time stamp in such a way that no two generated UIDs will ever have the same value.⁹ The UID is stored in nonvolatile storage (if the node has some), so the UID remains constant across system reboots. Nodes without nonvolatile storage will generate a new UID on every reboot.

Third, a change in the name, address, ID, or UID attributes is reported by the node as an event, which aids in detecting duplicate node names and addresses. Two nodes can end up with the same name when the disk where a node stores its system image, name, address, and UID is copied, and then the copy is booted on another machine. When the disk is booted on the second machine, that machine would have a different ID ROM. The node would detect that its ID is different, and thus an event would be generated. The event would not prevent the duplicate node from booting, but it would allow the manager to detect that a duplicate node may be on the network.

Start-up

A node is responsible for system start-up. We model start-up through four states.

- **Dead**, when the node is down and requires manual intervention to start.
- **Booting**, when the node is in the initial stages of software start-up. The booting process is highly system specific and may be initiated by hard-

ware, by software, by a power failure, or by a manager's console request. Booting loads a system image, starts it running, and brings it to a known state. The system image can be loaded from a disk or equivalent storage, or it can be loaded over the network using the maintenance operations protocol (MOP) down-line load protocol.¹⁰ MOP is layered directly over the data link protocols. In Digital's communications devices, MOP is generally implemented in the hardware or firmware and does not require a working operating system.

- **Off**, when the node is initializing itself and its internal configuration. When booting completes, the node changes to the off state. This transition is called the "big bang." In the first instant after the big bang, the node has at least the following things available, as shown in Figure 5:
 - A working clock and time service used to time stamp events.
 - A UID generator used to give entities and events a unique identifier.
 - The node entity (and possibly some of the node's child entities) together with its agent (which includes both the directive dispatcher and event logging).
 - An initialization script, a series of management commands to configure the system. This can be in the form of a text NCL command file (described later in the section on NCL), or it can be a compiled script, one that has been encoded as a series of common management information protocol (CMP) requests. MOP can be used to down-line load an initialization script.
 - An initialization director, which reads the script and invokes the directives in the order given. Errors and other output may be displayed on a console (if the system has one) and/or reported as events.
- **On**, when the node has "completed" initialization to the extent that it can be managed remotely. Somewhere in the initialization script (probably near the end), the node is enabled, which changes its state to on, i.e., it can be managed remotely.

Modules

A node has many subsystems, called modules in DECnet/OSI. Each module may or may not be configured within any particular node. Within the

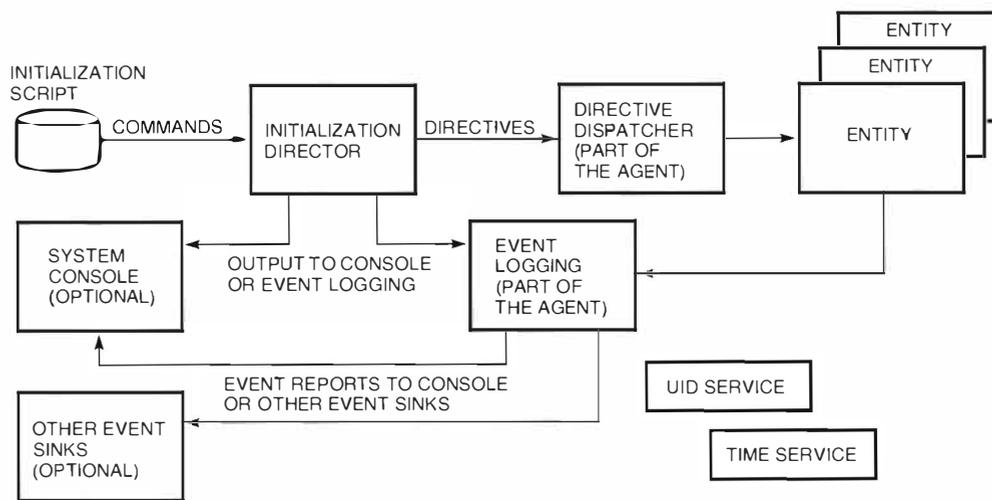


Figure 5 The Node at the "Big Bang"

modules are the various subsystems that make up DECnet/OSI. A node never has more than one instance of a module contained within it. A general-purpose node allows the manager to flexibly configure a node to serve a particular purpose by creating and deleting the appropriate modules.

In the DECnet/OSI Phase V network, the specification of the management of each module is an integral part of the architecture of the subsystem. Moving responsibility for the management of a subsystem from a central network management architecture to the subsystem architecture has made the specifications clearer and more complete. In Phase IV, a great deal of effort was spent coordinating the subsystem specifications and the network management specification. Placing responsibility in one person's hands made writing an internally consistent subsystem much easier. Besides, the sheer size of DECnet/OSI Phase V management would have made it impossible for a single person to design the management of the whole system.

The development of the OSI management standards in ISO/CCITT (Comité Consultatif International de Télégraphique et Téléphonique) has been done in a similar way and for the same reasons. ISO/IEC JTC1 SC21/WG4 is the group that has developed the OSI management information model, management specification language, and guidelines for module developers. While SC21/WG4 has itself also developed the management of specific subsystems (e.g., for event forwarding and logging), typically, the job of doing this has been left to other

groups more expert in particular areas. For example, Working Groups 1, 2, and 4 of ISO/IEC JTC1 SC6 have developed management standards for the ISO data link, network, and transport layers, based on Digital's contributions derived from the DECnet/OSI Phase V work in these areas.

In DECnet/OSI, the transport, network, and data link subsystems were among the first to have the EMA concepts applied to their management. Others quickly followed and, presently, more than 50 modules have been specified, with others being added as new subsystems are designed. Not surprisingly, during the early days considerable interaction took place between the architects responsible for the central network management architecture and those responsible for developing the management of specific subsystems. The EMA evolved and was refined based on the experiences of the many subsystem architects using it.

In almost all cases, modules contain one or more entities, each representing some management aspect of the subsystem. These entities in turn may contain other entities (subentities). This nesting can occur to an arbitrary depth, reflecting the management complexity of the subsystem. Note that modules themselves are entities, albeit with the restriction that a node never has more than one instance of a module contained within it. An entity is formally described using Digital's Management Specification Language (MSL).¹¹

We next consider in more detail the structure and contents of the DECnet/OSI Phase V OSI

transport module. Complete descriptions of this and other Phase V subsystems can be found in the Digital Network Architecture (Phase V) Documentation Kits.^{12,13,14,15}

OSI Transport Module

In DECnet/OSI Phase V, the OSI transport module contains port, template, local network service access point (NSAP) address, and manufacturing automation protocol (MAP) entities. A local NSAP entity contains remote NSAP entities. The containment hierarchy is shown in Figure 6.

The OSI transport module has characteristic attributes. A manager can change the configuration of the module by modifying its characteristic attributes. This is done for several reasons, including

- To limit the maximum permissible number of active transport connections at any one time
- To control the maximum credit window that may be granted on an individual transport connection
- To control the maximum number of transport connections that can be multiplexed on any single network connection, when the OSI transport protocol is operating over the connection-mode network service

Modification of these attributes is needed only if the manager requires anything other than a standard configuration; working default values are defined for all characteristic attributes.

Status attributes show the current operating state of the module, e.g., the number of transport connections currently active. Status attributes cannot be modified directly by a manager. To start the

operation of the OSI transport module, the manager uses the enable action. If successful, the state attribute changes from off to on.

In the DECnet/OSI Phase V architecture, a port entity represents the interface between layers, making visible to a manager how one layer (a client) is using the services of a lower layer. Ports are not created by a manager; they are created when a client of the service requests use of the service (by "opening a port"). The exact information held in a port entity varies for each subsystem. In general, a port entity contains attributes that identify the client and the service being used, and how that service is being used (e.g., as usage counters). The port entity is an example of how the EMA evolved through feedback from the subsystem architects. Before being adopted as a general mechanism in the overall management architecture, the concept was first developed and used in subsystem architectures.

In the case of the OSI transport module, the port entity also corresponds to the local end of a transport connection (TC), and it provides a window to the status information associated with the TC. For example, the OSI transport port status attributes give

- The name of the user of the OSI transport service
- Local and remote NSAP addresses and transport selectors
- The protocol class being operated on the TC

In addition, a port entity has counter attributes that record the total number of times something of interest occurred on the TC. For example, there are counters recording the number of octets and protocol data units (PDUs) sent and received. A management station can poll these and determine usage

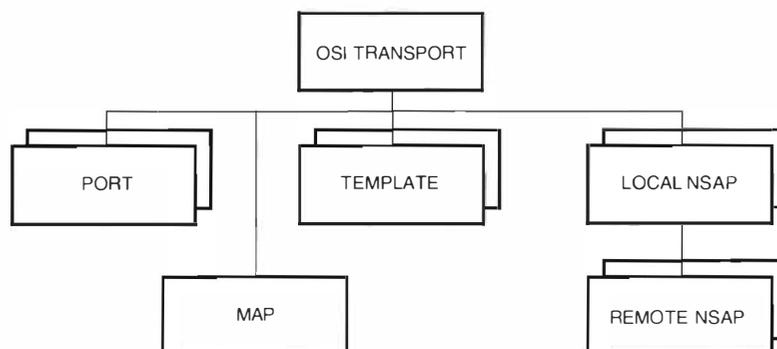


Figure 6 Containment Hierarchy for OSI Transport Module

over time. A port entity also maintains counters for both duplicated transport PDUs (TPDUs) detected and retransmitted TPDUs. Taken with the usage counters, these can be used to calculate error ratios and rates on the TC.

When a client opens a port onto a service, the client can then use the service interface to select options such as which features to use or which profiles. Maximum flexibility, however, also poses a problem. In many cases, a client has little or no knowledge or understanding of the service options available in an underlying layer. Further, it would be unrealistic to expect all clients of a service (or, ultimately, an end user) to acquire this in-depth knowledge.

One alternative was to provide default values for all the service options. However, a single set of default values satisfies only a single subset of uses. Instead we adopted the template, which is an entity that represents a set of related option values. A manager can create as many templates as required for different sets of related option values. A client needs to be configured only with the single name of the template to use, not the details of every service option. The OSI management standards groups have adopted the template concept in the form of their initial value managed object (IVMO).

A template in the OSI transport module is a collection of characteristic attributes used to supply default values for certain parameters that influence the operation of a TC. When a port is opened to the OSI transport service, a template name may be specified by the client. The characteristic attributes in the template are then used as default values for TC parameters not supplied by the user, including, for example,

- The value of the window timer
- The set of classes of protocol that may be negotiated for use on a TC
- The use of checksums that might be negotiated for a TC that operates the class 4 protocol, a variant of the OSI transport protocol defined in ISO 8073

A default template is automatically created and used if no template is specified when a port is opened.

There is one local NSAP entity for each NSAP address used by the OSI transport. A local NSAP entity is automatically created when an NSAP address used by the OSI transport is added to the network routing subsystem (the adjacent lower layer).

The remote NSAP entity is a subentity of a local NSAP entity. Each remote NSAP entity maintains counter attributes resulting from interactions between the superior local NSAP and a remote transport service provider. Events are defined for the remote NSAP entity, to provide immediate notification to the manager of error conditions. For example,

- A checksum failure event occurs whenever checksum validation fails when performed on a received TPDUs
- An invalid TPDUs received event occurs whenever a TPDUs received from the remote NSAP is in violation of the transport protocol

Consider this second example. Whenever an invalid TPDUs received event is generated, a counter is incremented. Thus, even if the manager has configured event logging to filter out these events, an indication that they are happening remains, prompting the manager to change the filtering criteria. The event contains a number of arguments as well. All events identify the generating entity and the time the event occurred. The invalid TPDUs received event also has arguments that give

- A reason code, indicating in what way the TPDUs was invalid, as specified in the ISO 8073 standard¹⁶
- The part of the TPDUs header that was invalid
- A specific Digital Network Architecture (DNA) error code, which was added to qualify the ISO 8073 reason code and to help customers diagnose problems

The MAP places a number of requirements upon implementations of the OSI transport protocol beyond simple conformance to ISO 8073. The MAP entity contains the additional management needed to meet these extra requirements. The MAP entity is optional; implementations with no business requirement to support MAP would not provide the MAP entity.

Supporting Mechanisms

Network management in DECnet/OSI is built on a number of supporting services. Wherever possible, management uses the services of the network to manage the network. This approach minimizes the number of special mechanisms we had to define specifically for network management. Some key services used by network management include

- Session control
- DECdns name service
- Digital's distributed time service (DECdts)
- A unique identifier service (UID)

A few services were developed specifically to support network management. Most had existed in earlier phases of DNA.

- DNA CMIP
- Event logging
- MOP down-line load protocol
- Application loopback

In the following sections, we describe DNA CMIP and event logging.

Digital Network Architecture Common Management Information Protocol

The entity model describes what an entity can do. Those concepts must be expressed in the management protocol. DNA CMIP, the management protocol for DECnet/OSI Phase V, is an evolution of the Phase IV management protocol (called NICE). The two protocols are remarkably similar. Both include the set, show (also called get), and event report operations. The main differences between the two protocols are in the following areas.

- Treatment of other operations. In NICE, each operation required a new kind of message; in CMIP, a general extension mechanism, the action, is provided.
- Naming. NICE supported a limited number of entity classes (eight) and provided a rudimentary naming hierarchy based on the notion of "qualifying attributes." CMIP supports hierarchical entity names and is essentially unlimited in the number of entities with which it can deal. Similarly, CMIP is much more extensible in naming attributes, attribute groups, and event reports.
- Encoding. CMIP uses ISO Abstract Syntax Notation 1 (ASN.1), a standard tag, length, value (TLV) encoding of attributes and arguments, and NICE used a private TLV encoding.

DNA CMIP is not quite the same as the IS version of OSI CMIP, although it was based on the second draft proposal of the CMIP standard. There are two reasons for this.

- First and foremost was timing. DNA CMIP was developed before the OSI CMIP was standardized. The inevitable changes to the standard led to many minor differences in the protocols. Still, because the concepts in the EMA entity model and OSI's SMI are aligned, the DNA and OSI CMIP protocols are fundamentally the same. The authors are currently migrating DNA CMIP to OSI IS CMIP. The change will be transparent to any user.

- Second, DNA CMIP operates over a DNA protocol stack, not a pure ISO stack. This allows directors on Phase IV systems to manage Phase V systems.

DNA CMIP can be viewed as two separate protocols. One protocol, management information control exchange (MICE), is used by a director to invoke a directive (get, set, action, etc.) on an entity (or entities). The other protocol, management event notification (MEN), is used by an entity (or entities) to report events to a director. The two protocols operate over separate connections for important reasons.

- The times at which the associations are connected differ. A MEN association is brought up when an entity wishes to report an event, and is thus controlled by the agent. A MICE association, however, is brought up when a director (or manager) wishes to invoke an operation on an entity, and is thus controlled by the director. Attempting to share control of association establishment was not worth the complexity.
- Whenever an association is shared by two different users, the problem of allocating resources fairly to the two users must be addressed. Since transport connections deal with this issue between connections, the addition of a multiplexing protocol at the application level (with an attendant flow control mechanism) was again considered to be too complex. Transport connections are not (or should not be) expensive.

Event Logging

The entity emits an event report to the manager when an event occurs in an entity. The event logging module provides a service that transmits event reports from the reporting entities to one or more sink applications, which are considered to be a certain kind of director in EMA. Event logging in Phase V is based on concepts similar to those provided by Phase IV. Because the principal use of event logging is for reporting faults, event logging does not

guarantee delivery of event reports to the sink application. Figure 7 shows the event logging architecture.¹⁷

When an event occurs within an entity (E) in a source node, the entity invokes the PostEvent service provided by the event dispatcher (a part of the node's agent). When posting an event, the entity supplies its name, the type of the event, all the arguments related to the event, a time stamp of when the event occurred, and a UID assigned to the event. UIDs ensure that each event can be uniquely identified, so that if a sink application receives more than one copy of an event report, it can detect the duplication. Time stamps allow the event reports to be ordered in time (an important step in determining causality). A time service (DEC:ts) is used to synchronize clocks across the network. It provides a consistent view of time for correlating observations. An important feature for management is the inclusion of an inaccuracy bound on the time stamp.

The PostEvent service formats an event report and places it in an event queue (Q). Event queues are limited in the amount of memory they use; thus they limit the number of events that can be held in the queue. Because events can be placed in the queue at a rate faster than the queue server (S) can process them, the queue can fill, and any new events placed in the queue will be lost. The events lost event is recorded as a pseudo-event in the queue (it appears as an event report from the entity holding the queue). The events lost event carries an argument that records the number of events that were lost in a row.

The queue server for the event dispatcher compares each event report against a filter (F) associated with an outbound stream. The filter lists

a collection of entities and events that are either passed through the filter or blocked by the filter. Event reports passing through the filter are placed in an event queue within the outbound stream. Each outbound stream's queue server sends events to a corresponding inbound stream in the sink application. Multiple outbound streams can be set up by the manager, allowing events to be sent to many sink applications. Outbound streams are modeled as entities in their own right, and standard management operations (create, get, set) are used to configure them.

Each inbound stream in a sink application has an event receiver (R). Inbound streams are generally created when a connection request is received from an outbound stream. Events received by the receiver are compared against a sink filter and queued to the sink application. Thus the events from multiple inbound streams are merged.

The protocol used between the outbound stream and the inbound stream is the CMIP MEN protocol, which operates over a connection (using either the DECnet transport layer protocol or OSI transport). The use of a connection lowers the probability that an event report will be lost, since the connection handles acknowledgments and retransmissions. It does not guarantee delivery, however, and events may still be lost due to failures of the sink application or the source node.

Conclusions

Our approach to Phase V management worked well. Defining the EMA entity model first provided a framework of consistency among all the architectures. Developing a management protocol (CMIP) expressing the basic concepts in the entity model

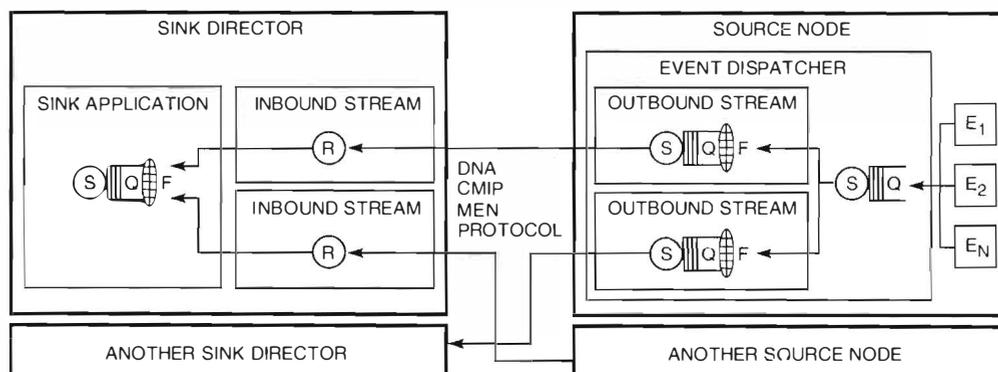


Figure 7 Event Logging

in conjunction with the model placed the protocol in a position to meet the needs of the model. Giving responsibility for defining the management of a subsystem to the architects of that subsystem made each subsystem more complete and coherent. As problems were found in the model based on lessons learned during the specification of entities, any needed changes to the entity model were applied to correct those problems.

However, some things did not go as well. The number of entities, attributes, and operations in Phase V was beyond anyone's expectations. This reflects the overall complexity and feature-richness of Phase V over Phase IV as well as the increased control that the manager is given. This burden is eased somewhat by the use of intelligent defaults, autoconfiguration, and self-management. Still, simplifying the management of a Phase V network is an important area for continual improvement.

The biggest success of EMA/Phase V management is its general applicability. EMA is being applied to more than the traditional network management areas. Systems, networks, and applications are all managed by EMA.

References

1. N. LaPelle, M. Seger, and M. Saylor, "The Evolution of Network Management Products," *Digital Technical Journal*, vol. 1, no. 3 (September 1986): 117-128.
2. J. Harper, "Overview of Digital's Open Networking," *Digital Technical Journal*, vol. 5, no. 1 (Winter 1993, this issue): 12-20.
3. C. Strutt and J. Swist, "Design of the DECmcc Management Director," *Digital Technical Journal*, vol. 5, no. 1 (Winter 1993, this issue): 130-142.
4. *OSI Management Information Services—Structure of Management Information—Part 1: Management Information Model*, ISO/IEC DIS 10165-1 (Geneva: International Organization for Standardization/International Electrotechnical Commission, 1990).
5. *OSI Management Information Services—Structure of Management Information—Part 4: Guidelines for the Definition of Managed Objects*, ISO/IEC DIS 10165-4 (Geneva: International Organization for Standardization/International Electrotechnical Commission, 1992).
6. M. Saylor, "Guidelines for Structuring Manageable Entities," *Integrated Network Management I*, B. Meandzija and J. Westcott (eds.), (Amsterdam: Elsevier Science Publishers, 1989): 169-183.
7. *DNA Network Management Functional Specification, V5.0.0* (Maynard, MA: Digital Equipment Corporation, Order No. EK-DNA02-FS-001, 1991).
8. *DNA Naming Service Functional Specification, V2.0.0* (Maynard, MA: Digital Equipment Corporation, Order No. EK-DNANS-FS-002, 1991).
9. *DNA Unique Identifier Functional Specification, V1.0.0* (Maynard, MA: Digital Equipment Corporation, Order No. EK-DNA1-FS-001, 1992).
10. *DNA Maintenance Operations Protocol Functional Specification, V4.0.0* (Maynard, MA: Digital Equipment Corporation, Order No. EK-DNA11-FS-001, 1992).
11. *DECmcc System Reference Manual*, 2 volumes (Maynard, MA: Digital Equipment Corporation, Order Nos. AA-PD5LC-TE, AA-PE55C-TE, 1992).
12. *Digital Network Architecture (Phase V) Documentation Kit No. 1* (Maynard, MA: Digital Equipment Corporation, Order No. EK-DNAP1-DK-001, forthcoming 1993).
13. *Digital Network Architecture (Phase V) Documentation Kit No. 2* (Maynard, MA: Digital Equipment Corporation, Order No. EK-DNAP2-DK-001, 1993).
14. *Digital Network Architecture (Phase V) Documentation Kit No. 3* (Maynard, MA: Digital Equipment Corporation, Order No. EK-DNAP3-DK-001, forthcoming 1993).
15. *Digital Network Architecture (Phase V) Documentation Kit No. 4* (Maynard, MA: Digital Equipment Corporation, Order No. EK-DNAP4-DK-001, 1993).
16. *Information Technology—Telecommunications and Information Exchange Between Systems—Connection Oriented Transport Protocol Specification*, ISO/IEC 8073 (Geneva: International Organization for Standardization/International Electrotechnical Commission, 1989).

17. *DNA Event Logging Functional Specification, V1.0.0* (Maynard, MA: Digital Equipment Corporation, Order No. EK-DNA09-FS-001, 1992).

General References

EMA Entity Model (Maynard, MA: Digital Equipment Corporation, Order No. AA-PV7KA-TE, 1991).

M. Saylor, "Managing DECnet Phase V: The Entity Model," *IEEE Networks* (March 1988): 30-36.

S. Martin, J. McCann, and D. Oran, "Development of the VAX Distributed Name Service," *Digital Technical Journal*, vol. 1, no. 9 (June 1989): 9-15.

C. Strutt and D. Shurtleff, "Architecture for an Integrated, Extensible Enterprise Management System," *Integrated Network Management I*, B. Meandzija and J. Westcott (eds.), (Amsterdam: Elsevier Science Publishers, 1989): 61-72.

DNA Network Command Language Functional Specification, V1.0.0 (Maynard, MA: Digital Equipment Corporation, Order No. EK-DNA05-FS-001, 1991).

L. Fehskens, "An Architectural Strategy for Enterprise Network Management," *Integrated Network Management I*, B. Meandzija and J. Westcott (eds.), (Amsterdam: Elsevier Science Publishers, 1989): 41-60.

Design of the DECMCC Management Director

The DECMCC product family represents a significant achievement in the development of enterprise management capabilities. DECMCC embodies the director portion of Digital's Enterprise Management Architecture (EMA) and is both a platform for the development of new management capabilities and a vehicle for aiding customers to manage their computing and communications environments. Initially, the DECMCC director was intended to facilitate sophisticated management of evolving networks. In addition to network management, DECMCC has been adapted to the needs of system, applications, data, environment, and telecommunications management. The first implementations contained the DECMCC kernel, a developer's toolkit, and various management modules.

Development of the DECMCC director has been a multiyear effort involving many groups within Digital. When the DECMCC design was initiated in 1987, there was no equivalent management software in the industry. Most companies, Digital included, provided one or more independent, focused products. Each of these dealt with managing a specific set of components such as a single vendor's local area network (LAN) bridges or providing a specific management application such as equipment inventory.

Digital's network management capabilities within DECnet Phase IV were reaching their limit, and the incorporation of newer communications technologies in a seamless way was becoming increasingly difficult. As part of the DECnet Phase V development, work was started to rationalize management of distributed systems. This effort led to the formal definition of such concepts as the director/entity relationship, the entity model, and the common management information protocol (CMIP).^{1,2,3,4} These ideas formed the basis for management in Phase V and were Digital's contributions to the open systems interconnection (OSI) management model from the International Organization for Standardization (ISO).

The original vision of network management in Phase V included the concept of two management directors. The first, a sophisticated director referred to as the management control center (MCC), would handle the more complex, yet user-oriented, man-

agement tasks. The second, a simple command line director referred to as network control language, would address the needs of more experienced managers who prefer a command line environment.⁵

Conceived primarily as a DECnet management director, the DECMCC product evolved to address the broader problems associated with managing a complete computing and communications environment. This evolution is not yet finished and arguably will never finish as network environments continue to change.

Since the development of DECMCC in 1987, the simple network management protocol (SNMP) has become widely implemented. DECMCC has adapted to handle SNMP as well. In addition, the DECMCC product, once a tool for the VAX VMS architecture, is now implemented on multiple platforms, such as the ULTRIX and UNIX System V Release 4 operating systems.

In this paper, we look at the development of the DECMCC director. We start by discussing our initial design ideas taken in the perspective of the industry at the time. We then describe the initial implementation of DECMCC. We also present the effects of the changing industry and how DECMCC has adapted over time. We conclude with some of the opportunities for future work.

Historical Perspective

Digital's first network management capability was delivered in 1978 as part of the release of DECnet

Phase II software. Much of the DECnet product was then manageable, both configuring the software for installation as well as the operational aspects. The main program used to perform management was the network control program (NCP). At that time management mostly consisted of looking at information and then changing it as needed. DECnet Phase II, however, could perform sophisticated diagnostic loopback tests, both nonintrusive as well as intrusive, to diagnose connectivity problems at various layers of the protocol stack.

Management formed a significant part of the DECnet Phase III and DECnet Phase IV networking products. Each major release contained many changes to manage the new functionality. However, the DECnet management structure in place in the 1970s was becoming more difficult to adapt to the requirements of the mid-1980s. For example, support was added for X.25 during Phase III and for Ethernet during Phase IV. These releases required quite different management approaches than the one used for Phase II. With the advent of the significant changes to DECnet Phase V to include support for the OSI protocol stack, another management approach was needed.

Thus in conjunction with Phase V network development, an effort was started to provide a new architectural approach to management of Phase V. One of the key requirements was to provide the Phase V management needs in a way that would extend their adaptability to the future. This work was referred to as distributed systems management because it addresses management of the computing environment as well as management of the communications that DECnet comprises. Most of the initial work in distributed systems management concerned itself with the aspects that applied to DECnet and the changes needed to provide manageability of DECnet in Phase V. The primary underlying concepts were articulated.

- Directors are management programs used by human managers to effect management. Entities represent managed components to directors through software referred to as agents.⁶
- The entity model is the underlying model for managed entities defined in terms of an object-based approach.^{1,3,7}
- The formal specification for the classes of entities is defined in terms of Module-2+ like specifications and is called management definition language (MD).⁵

- A command language, network control language (NCL), was formally defined to be unambiguous even with new entities and their definitions; an associated primitive director of the same name, part of every Phase V package, replaces the NCP of previous phases.⁵
- A management protocol called the common management information protocol (CMIP) was used to communicate between directors and entities.^{4,8,9}

CMIP was named common and presumed to handle the common aspects of management across a wide variety of management applications. Some developers suggested the possible need for a small number of specialized management information protocols (SMIPs)—perhaps one for each of the management functional areas (configuration, performance, fault, security, and accounting). However, CMIP proved to be sufficiently expressive and powerful to support management applications covering the management functional areas.

At the time the distributed systems management work was initiated, Digital's networking and communications product line was expanding to encompass more than the DECnet networking hardware and software. Along with each product came its own management software, some of which was tailored along the lines of the DECnet standard NCP. In addition, the Network Management Development Group was building some fairly sophisticated management applications that went far beyond the capabilities of NCP in DECnet. The developers necessarily took a different approach to management.

Thus, by the late 1980s Digital had developed a number of distinct management products. Many of these employed private protocols, for example

- NCP for managing DECnet, based on a command line user interface
- NMCC/DECnet monitor, a wide-area DECnet monitoring tool, based on a graphical user interface
- NMCC/ETHERnim, an Ethernet monitoring/inventory test program, based on a graphical user interface
- RBMS, Remote Bridge Monitoring Software for managing Digital's bridge family, based on a command line user interface similar to NCP

- TSM, Terminal Server Manager for managing Digital's terminal server family, based on a command line user interface similar to that used in the terminal servers
- LTM, LAN traffic monitor for understanding the traffic usage and patterns of Ethernet segments, based on a graphical user interface

Other manufacturers also provided management software capable of managing their devices. Some vendors provided particular management applications that were not tied to any specific network device. These applications performed a single function, such as maintaining an inventory of equipment on behalf of a manager.

The plethora of management capabilities from many vendors created many choices for end users. At the same time, the diverse applications were perceived as carrying significant drawbacks. Each application provided its own user interface. Each had its own database for storing management information. Each dealt with different management information. In addition, each tool provided its own, often rudimentary, independent management application.

End users viewed these many products as creating a series of problems: (1) A manager needed multiple management terminals, one per product. (2) Separate training was required to use each product. (3) Confusion occurred when the user switched between multiple products. (4) Different information was available from each product, or worse, the same information was available in a different form. (5) There was no ability to share information between products. (6) It became difficult to diagnose problems that spanned multiple technologies. Other aspects of the system management perspective in 1986 have been described.¹⁰

At that time, standards for network management had not progressed very far; SNMP did not yet exist. In fact, agreement on the overall concepts had only begun within the OSI management committees.

It is with this background, then, that the design of DECMcc as a management director was undertaken.

Opportunities

Of all the situations that existed in customer networks in the mid-1980s, probably the most important was the realization that networks no longer consisted of equipment from a single vendor. In addition, different technologies were commonly used to improve a given customer's network. With each technology came its own management proto-

col, along with its own management structure. As networks became larger, more than one network manager was typically needed.

The opportunity existed to provide complete, integrated network management that could be adapted to the changing needs of management. Our product goals were

- To provide a consistent, integrated user interface, permitting management of any component in the enterprise to be performed in a style that does not depend on the specific component
- To provide integration of the management data (contained in the components as seen by the director) and management information (as constructed by the director using the management data)
- To provide a consistent, extensible means of storing management information and of allowing it to be accessed conveniently by multiple independent management applications
- To provide an application programming interface (API) to support management applications

Obviously, an approach necessary to solve these nontrivial problems was not to be a small undertaking; an architected approach was appropriate.⁶

Design Approach

The solution to the problems outlined was seen to be a distributed applications environment, tailored to the specific needs of management. Quite quickly, the idea of defining a modular and extensible environment was selected.

Management capabilities could be added in a straightforward fashion based on an applications kernel, which could either be replicated as needed around a network, or considered as multiple, cooperating kernels supporting a distributed management environment. Hence a kernel with modules that can be added dynamically, much as applications are added to an operating system, is fundamental to the design of DECMcc.

The next consideration concerned the composition of the modules themselves. One approach to the support of multiple technologies had one module access each different sort of component to be managed. Since a number of management application functions were desirable, one might have a module for each such function. Also one might have a module for each form of user interface to

accommodate the different user interface styles, such as command line or windowing.

Thus, we arrived at the concept of distinguishing form, function, and access. Furthermore, we defined management modules based on presentation modules (PMs) for user interface, function modules (FMs) for management functions, and access modules (AMs) for accessing each distinct technology. The DECmcc director structure is shown in Figure 1.

We observed that the EMA entity model, defined initially to meet the needs of management of entities, provided generalized structuring concepts that would be appropriate for the director environment as well. Indeed, choosing the same model to handle the needs of the director removed the need for a translation between the entity environment and the director environment for EMA entities, which has proved to be advantageous for the implementations. Hence the following entity model concepts were also used in the director.

- An object-oriented approach—encapsulating objects (entities) and their operations
- A class structure—defining attributes, operations, and events for each class and specifying management information using a management specification language

As we studied the needs for stored management information in the director, we identified four different sorts of information, distinguished by the storage needs, nature of the contents, and the access patterns.

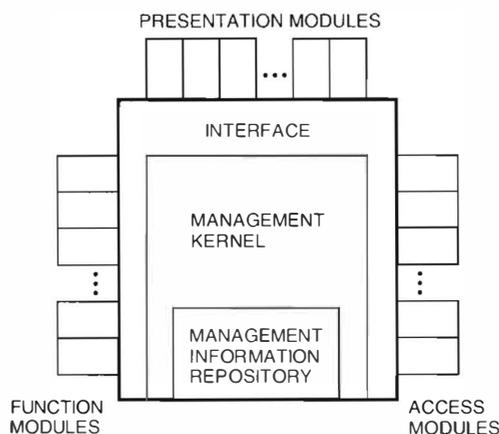


Figure 1 DECmcc Director Structure

1. Class data—the dictionary of all management operations, attributes, notifications, and their related definitions categorized by class, updated infrequently, but read often
2. Instance data—the configuration information, stored in a global naming service, changing often, but read from many places simultaneously
3. Historical data—information about specific entity instances stored over time, written incrementally and read sporadically according to the needs of applications using such data
4. Miscellaneous data—other data needed for specific modules, such as tariff information or the definition of rules specifying alarm conditions

The complete logical information store was termed the management information repository (MIR).

The kernel defines an execution environment that is suitable for management modules and supports the MIR. This was initially implemented in terms of technology provided completely within the director kernel. Many of the kernel services, however, were subsequently replaced with distributed systems services, including multithread support, naming/directory service, time service, and remote procedure call (RPC).

It is, perhaps, interesting to note that the decision to use a multithreaded approach in DECmcc was not unanimous. The alternate approach proposed an asynchronous message-passing scheme. Although the decision to use a multithreaded environment has proved to be implementable, we did not appreciate how the performance of the multithreading implementations would affect the ability to support the needs of application environments such as DECmcc.

Invoking Module Services

As we looked at how management modules would call each other, we chose a fairly straightforward approach. User interactions with a PM would cause the PM to invoke an FM, the FM to then invoke the appropriate AM, and the AM to communicate with the desired entity. The response would then be transmitted through the AM, FM, and PM, with the result presented to the user. Thus the simple procedure call paradigm between modules, as shown in Figure 2, supported the needs of applications geared toward monitoring and control operations.

However, one must consider the increase in the total number of management modules over time,

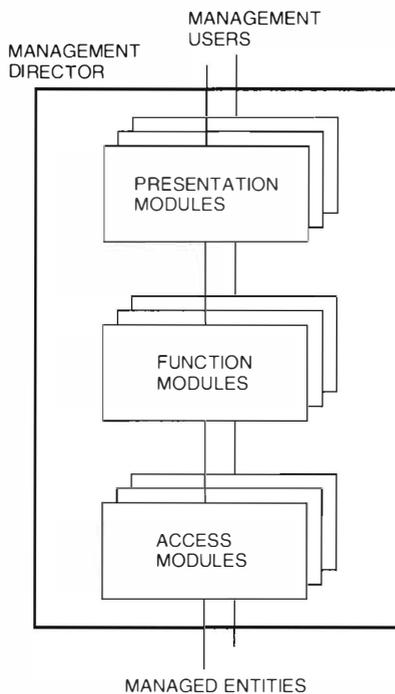


Figure 2 Management Module Calling Hierarchy

and the even greater increase in the total number of available management services (defined by specific operations on classes of entities). Thus, it became clear that the intermodule procedure calls could not use named procedures, as administering the names of ever-increasing numbers of procedures would be a burden. Instead we chose an approach whereby modules invoked each other's services by referring to the operations and the objects, using a service invocation procedure known as "mcc_call." We defined the interfaces provided by the management modules entirely in terms of operations on objects—an object-oriented approach—but this approach did not require the use of object-oriented languages or databases.

We further observed that one could decompose a management application into a number of smaller, potentially reusable services. Hence FMs could invoke other FMs in performing their services much in the same way that applications on UNIX systems pipe results from one component to another. Given the generally extensible nature of DECmcc and the supporting mcc_call structure, this led to the concept of generic applications. Being run-time driven from the class dictionary, these applications could work over a wide range of managed objects and

perform the same service for each of them without a priori knowledge of the objects. For example, one might have an FM that provides performance-related services, turning error counters (obtained directly from the managed objects) into error rates (by simply polling for two counter values, subtracting one from the other, and dividing by the time interval between polls). A different FM might provide alarm services by notifying users of particular (user-specifiable) conditions, such as when a particular counter exceeds a defined threshold.

Of course, managers are often more interested in error rates exceeding a given threshold. The same alarms FM could be primed to look for an error rate; the request would be passed on to the performance FM, which in turn would calculate the rate by looking at successive polls of the error counter. The alarms FM does not need to be aware whether the data it needs comes from the performance FM or directly from the managed object via the appropriate AM. The disposition of the methods among modules is hidden by the service invocation mechanism.

Furthermore, the alarms FM tracks the number of times a user is notified of a problem, and this counter is available as management data. One might then want to determine the rate of user notifications (using exactly the same generic performance FM as before), and use the same alarms FM to notify a different user when the rate of notifications exceeds a defined threshold. This threshold might indicate that one manager is being overloaded. Thus, in this scenario we have a number of modules involved in a calling hierarchy, with the same modules appearing more than once. Figure 3 shows the reuse of software using generic function modules in DECmcc.

Management Specification Language

The entity model's management definition language, originally intended for the specification of management agents, was modified and applied to the director environment. Director-oriented information was added to the management specification, such as user interface tags for automatically generated forms and menus. This information was named the management specification language (MSL). An MSL compiler was defined to convert MSL to an on-line form, available as metadata through an on-line dictionary, the MIR class data. With the management specification information available to management modules, modules could

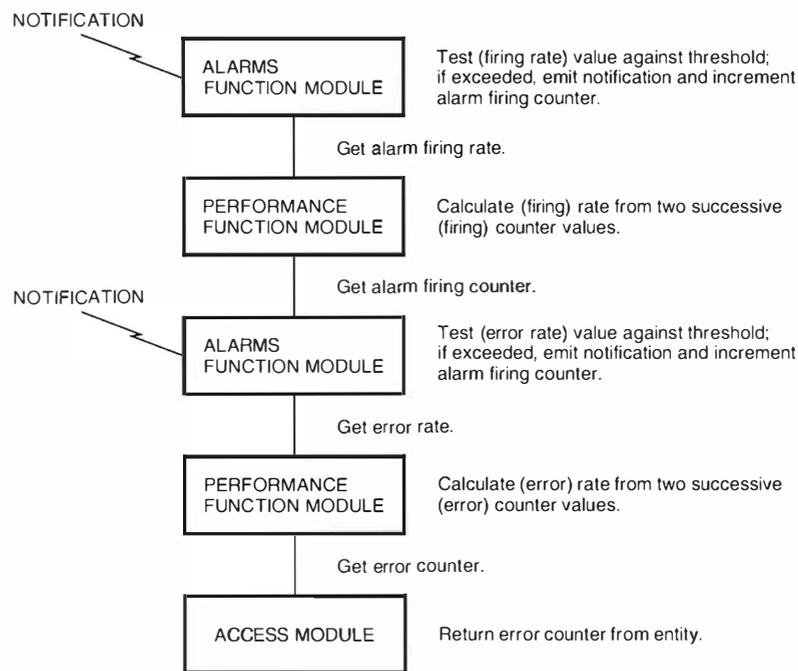


Figure 3 Data/Control Flow for Multiple FMs

adapt their behavior as new modules were added; this is especially important for generic modules. Thus the same MSL that was used to help the entity agent developers was also useful for the management director to drive the extensible management modules.¹¹

This dictionary information spurred the definition and development of the generic management modules. The generic PMs provide an extensible user interface that is capable of adapting as new managed objects or applications are added. The generic FMs provide consistent functions over a broad set of managed objects. Finally, the generic AMs support extensible management protocols, allowing the dynamic addition of new sorts of managed objects.

The design of the DECMCC director led to a number of possibilities in the type and application of the different sorts of modules. Initially AMs were conceived as being one per management protocol, which usually translated to one AM per type of device (such as bridge, terminal server, DECnet node). Since the advent of standard protocols, such as SNMP from the Internet community and CMIP for OSI management, AMs are now more typically generic and extensible.^{8,9,12} A single AM covers many different types of device with one protocol.¹³

For FMs, we originally envisioned two sorts of modules: the generic FM providing the same function over a wide variety of managed objects, and a specific FM providing a set of functions for a single class of managed object. Today, we believe one may have two different sorts of generic FM: one that is specific to a technology (such as network management related), and another, truly generic, which is completely independent of the technology being managed (such as an alarms FM).

For PMs, we recognized the need to handle device-specific aspects as well as user interface style-specific aspects. Normally one would have generic PMs provide user interface capabilities over a broad variety of managed objects and applications. However, to support the specific needs of generic FMs, specific PMs might be used to provide the appropriate user interface. PMs that are specific to an FM are less useful since they do not provide a consistent user interface "look and feel."

During the design of the DECMCC director, a number of smaller, but nonetheless important, design decisions were made. The concept of management domains was defined as a general container mechanism for entities, which could include domains themselves. Domains therefore provide a flexible, user-specifiable organizational structure for both

visual representation at the user interface, as well as a means to organize the stored management information and associated background processing.¹⁴ The need to provide a consistent approach to the naming of objects within the director was established. This was initially based on Digital's distributed name service, DECdns, providing globally unique names and network-wide access to those names.¹⁵ Finally, the concept of time, including the scheduling of operations as well as scope of interest for information retrieval, was included in the `mcc_call` API. The time concept allows management applications to be developed that can operate on historically stored information as easily as they can on data retrieved directly from the network.¹⁶

A more detailed report on the design of DECmcc has been published.¹⁷

Some other aspects of the DECmcc program, while not part of the technical design, had a major part to play in its evolution. First was the need to provide published, open definitions of the DECmcc API, based on existing standards. This allows other vendors and end users to develop their own management capabilities to add to DECmcc. Second was the establishment of a strategic vendor program within Digital to work with other vendors, particularly those that provided network technologies that complemented Digital's own offerings, to help them develop to the DECmcc platform. Finally a design center program was instituted whereby the design of DECmcc would be validated, as it evolved, against the needs of some major customers to ensure that it continued to address the management problems of those customers.

Broadening the Scope

Since DECmcc was designed to be able to manage anything that could be described by the entity model, and since the entity model is a general object-oriented framework, it follows that it is feasible to extend DECmcc to classes of managed object and applications beyond the traditional network-oriented view of nodes, hosts, bridges, routers, etc. Some of the new classes of managed objects and new applications that we have seen developed using DECmcc include

1. Management of applications such as transaction processors and databases
2. Applications in traditional system management, such as user management, disk backup, software installation, configuration maintenance, and performance monitoring

3. Management of objects in the telecommunications field, such as PBX machines, multiplexers, and switches¹⁸

4. Management of noncomputer hardware, such as air conditioners and building-environment controls

Note that the implementation of these extensions generally involves a relatively small investment, at which point the power of existing generic applications is automatically provided. For example, in the easiest case, a new object that is manageable through SNMP need only have its management information base (MIB) translated to MSL and loaded into the DECmcc dictionary, at which point it is accessible by the existing SNMP AM as well as the standard generic applications.

In other cases, such as the air conditioning example, it is only necessary to code an AM that communicates to the air conditioning controller through its private protocol. Functions such as alarms, notifications, historical data recording, and graphing are automatically provided by existing FMs and PMS upon recognition of the new object class.

In complex cases, object-specific FMs are written to perform such tasks as software installation and disk backup control. Yet even in these cases, all these functions are automatically accessible through the generic PMS.

The potential for interdisciplinary applications is now becoming possible by the normalization of the interfaces to objects traditionally handled by totally separate applications. For example, given the extensions described above, it is possible to write an application that activates an emergency disk backup and switches telephone trunk traffic to another building if an air conditioning failure occurs. In fact, depending on how the various objects are defined, it may even be possible to create such an application simply by writing a single alarm rule.

Evolution to Open Systems

With recent industry trends toward open systems environments, as well as the realization that almost any enterprise now comprises multiple hardware and software platforms from multiple vendors, it was clear that DECmcc had to evolve to this new world. Among the requirements to be met were not only the management of objects existing on various platforms, but also the execution of the director itself on different hardware and operating system platforms.

These requirements dictated two basic design goals:

1. Portability of the director kernel itself to environments other than VAX VMS
2. Portability of plug-in management modules to a DECmcc director running on any supported platform, and in particular, source compatibility to the greatest extent possible with the considerable suite of management modules that existed when the porting effort started

Many of the fundamental requirements for portability had already been met. All existing management modules were coded to the API defined in the *DECmcc System Reference Manual* (SRM), and the SRM had little code that was inherently specific to VAX or VMS.¹⁹ In fact, only the documented SRM routines were used to access DECmcc services, as well as many other common operating system services such as data storage and thread control. Consequently, the kernel implementation team had the flexibility to implement these services differently on various platforms without impacting management module source code. This was particularly true with the all-important `mcc_call` service, which provided the API for intermodule communication in a platform-independent context such that a wide variety of interprocess or intraprocess communications mechanisms could be chosen for the underlying implementation.

In the initial porting effort, which was from VAX VMS to RISC (reduced instruction set computer) and VAX ULTRIX, some of the more important changes in underlying implementations were

1. The MIR was implemented over the `ndbm` hash database manager. An earlier version of the MIR was also implemented over ULTRIX SQL, which provided some large-capacity database features at the expense of significant performance.
2. The operating system time interfaces were migrated to the distributed time service of the Open Software Foundation distributed computing environment (OSF DCE).
3. The multithreading services were migrated to the `DECTHREADS` component of the DCE.
4. The intermodule communication mechanisms (`mcc_call`) were implemented using RPC technology, with management modules running as independent RPC server processes. This allowed run-time extensibility without requiring the operating system to support a merged image activation function, a feature of the VMS implementation.
5. Through the use of various wrapper routines in the DECmcc development toolkit, we were able to allow the management module developer to code entry points to the management modules without distinction to whether they were being run in an image merge or an independent process context.

Despite these major changes, 85 percent of the kernel code is in fact platform independent, and we are maintaining a single source pool for DECmcc regardless of the number of platforms. To minimize the operating-system-dependent code we must maintain and to provide backward compatibility, we are also porting to VMS a number of the above technologies such as those built on DCE.

At the present time we continue to broaden our open systems focus by additional ports to UNIX System V, OpenVMS on Alpha AXP, OSF/1 on Alpha AXP, as well as other operating systems.

Implementation

In late 1990 and early 1991, Digital delivered the first two versions of DECmcc. Version 1.0 was written to allow other vendors to start building their management modules; version 1.1 added some components for network managers. Both releases ran on VAX VMS systems, either workstations or hosts.

In the middle of 1992, Digital released version 1.2 of DECmcc, which added significant capabilities and runs on RISC ULTRIX. Later in 1992, Digital delivered POLYCENTER SNA Manager. In conjunction with DECmcc and the SOLVE:Connect for EMA, a product from System Center, Inc., it allows bidirectional management between IBM SNA hosts and DECmcc systems.²⁰

In early 1993, Digital released version 1.3 of DECmcc under the new product family name of POLYCENTER, with the POLYCENTER Framework, which is the basis for POLYCENTER Network Manager 200 and POLYCENTER Network Manager 400. This new version adds ways to provide simpler, yet powerful, integration of management capabilities; uses an OSF/Motif graphical user interface; and provides additional development tools. These versions contain the DECmcc kernel, a corresponding developer's toolkit, and a series of management modules, which are outlined in Table 1. The SRM

Table 1 DECmcc Director Management Modules

Presentation Modules	Definitions
Forms and Command Line PM	Provides a command line user interface based on the NCL definition, together with a full-screen mode for video terminal devices. This PM also executes DECmcc command scripts.
Iconic Map PM	Provides an iconographic display based on OSF/Motif. It supports all the capabilities of the command line, but with a more usable graphical representation of the network and pull-down menu support. This PM also provides on-line graphing of management information. In addition, this PM can launch management applications that are not strictly part of the DECmcc environment, to provide a visual integration for the manager.
Notification PM	Provides an interactive management display of event or alarm firing conditions based on OSF/Motif. Flexible filtering of information is used to minimize the information displayed to the manager, but the manager can search for and display information using various criteria such as severity level, managed object, and data and time.
Function Modules	Definitions
Registration FM	Provides a means for registering entities with the director and for maintaining reference information on behalf of the entities.
Domain FM	Maintains the definitions of the various management domains, their membership, and their relationships.
Historian FM	Enables the capture and storage of user-specified management attributes from any entity in the network. Retrieval of the stored information by management modules is provided directly by the <code>mcc_call</code> API.
Exporter FM	Allows extraction of user-specified on-line or stored management information into a relational database for processing by SQL-based information management tools, such as reports.
Alarms FM	Permits managers to specify, through rules, the set of conditions about the network in which they are interested. When the alarms FM detects a condition (the rule fires), various notification techniques may be employed. These include invoking a command script, sending mail, calling a manager using an electronic beeper, or modifying an icon on the iconic map display.
Performance Analyzer FM	Calculates statistics for DECnet, transmission control protocol/internet protocol (TCP/IP), and LAN bridges, based on error and traffic utilization or other information.
Diagnostic Assistant FM	Helps the manager diagnose faults in a TCP/IP network, based on some of the more frequently occurring TCP/IP network problems.
Autoconfiguration FMs	Determine automatically the configuration and topology of specific portions of the network. Included are FMs to determine the configuration and topology of DECnet Phase IV networks, IP subnetworks, fiber distributed data interface (FDDI) ring maps, and LAN bridge spanning trees.
Access Modules	Definitions
SNMP AM	Provides access to objects that implement the SNMP protocol. It is a generic AM in the sense that it can adapt to new object definitions using information in the DECmcc dictionary. New MIB definitions are provided in a standard form and translated by a MIB translation utility into the DECmcc dictionary.
DECnet Phase IV AM	Provides access to the DECnet Phase IV implementations, be they hosts or servers such as routers. This AM implements the network information and control exchange (NICE) protocol.
DECnet/OSI Phase V AM	Provides access to the DECnet/OSI Phase V implementations, hosts, and servers. It implements the CMIP protocol used in Phase V.

Table 1 DECMcc Director Management Modules (continued)

Access Modules	Definitions
Bridge AM	Supports Digital's family of LAN bridges, including the LANbridge 100, LANbridge 150 and LANbridge 200, and the DECbridge family. It implements the RBMS protocol, which is used by the original management product of the same name.
FDDI AM	Supports Digital's FDDI DECconcentrator products and other devices that support the standard station management protocol (SMT).
Terminal Server AM	Supports Digital's family of terminal servers, implementing management through the maintenance operations protocol (MOP).
Ethernet Station AM	Supports all Ethernet and IEEE 802.3 stations that implement either, or both, the Digital MOP protocol or the IEEE 802.2 XID and TEST messages.
Circuit AM	Uses the services of other AMs to provide management of the network circuits that connect systems together, based on DECnet nodes, TCP/IP hosts, or network management forum definitions. Such circuits might be simple point-to-point or could represent complex multichannel circuits.
SNA AM and Agent PM	Permit bidirectional management of the SNA environment and the DECMcc management environment through a component that resides on an SNA host (either IBM's NetView or System Center's Advanced System Management).
Data Collector AM	Provides a means to allow other software, such as applications, to send events into DECMcc so they may be processed and analyzed along with events from devices or applications that have access modules.
Script AM	Allows invocation of existing or custom shell scripts or command procedures from DECMcc, and information to be returned from the scripts into DECMcc for processing and analysis by other modules.

provided the API definitions for management modules, as provided by the kernel. Figure 4 shows a sample screen from DECMcc being used to manage a portion of a network.

Since the DECMcc kernel is indifferent to the specific type of any management module, it is quite convenient to package different modules together, providing for a flexible packaging scheme. Each DECMcc can therefore be tailored to include the set of modules appropriate for managing the environment in which it is situated. In addition, modules from other vendors can be integrated by the customer without involvement from Digital.

As new management modules are added, the powerful generic capabilities of DECMcc allow many existing functions to be used without change. When an AM is added for a new class of resource, or when an existing generic AM is enhanced by adding new supporting definitions in the dictionary, one can immediately perform the following functions.

- Identify specific resource instances uniquely
- Make the resources known to all DECMcc directors in the network
- Represent the resources on an iconic display in one or more management domains
- Examine management attributes from these resources
- Modify management attributes in these resources
- Apply management actions to these resources
- Display event information from these resources
- Create alarm rules that can be triggered on particular conditions (polled or unsolicited) about these resources
- Have the relevant icons change color when the alarms fire
- Store, periodically, management data or information about these resources in the DECMcc historical data store, or export the information to a relational database
- View the stored historical data
- Process the relational data using standard information management tools, for example, to provide management reports

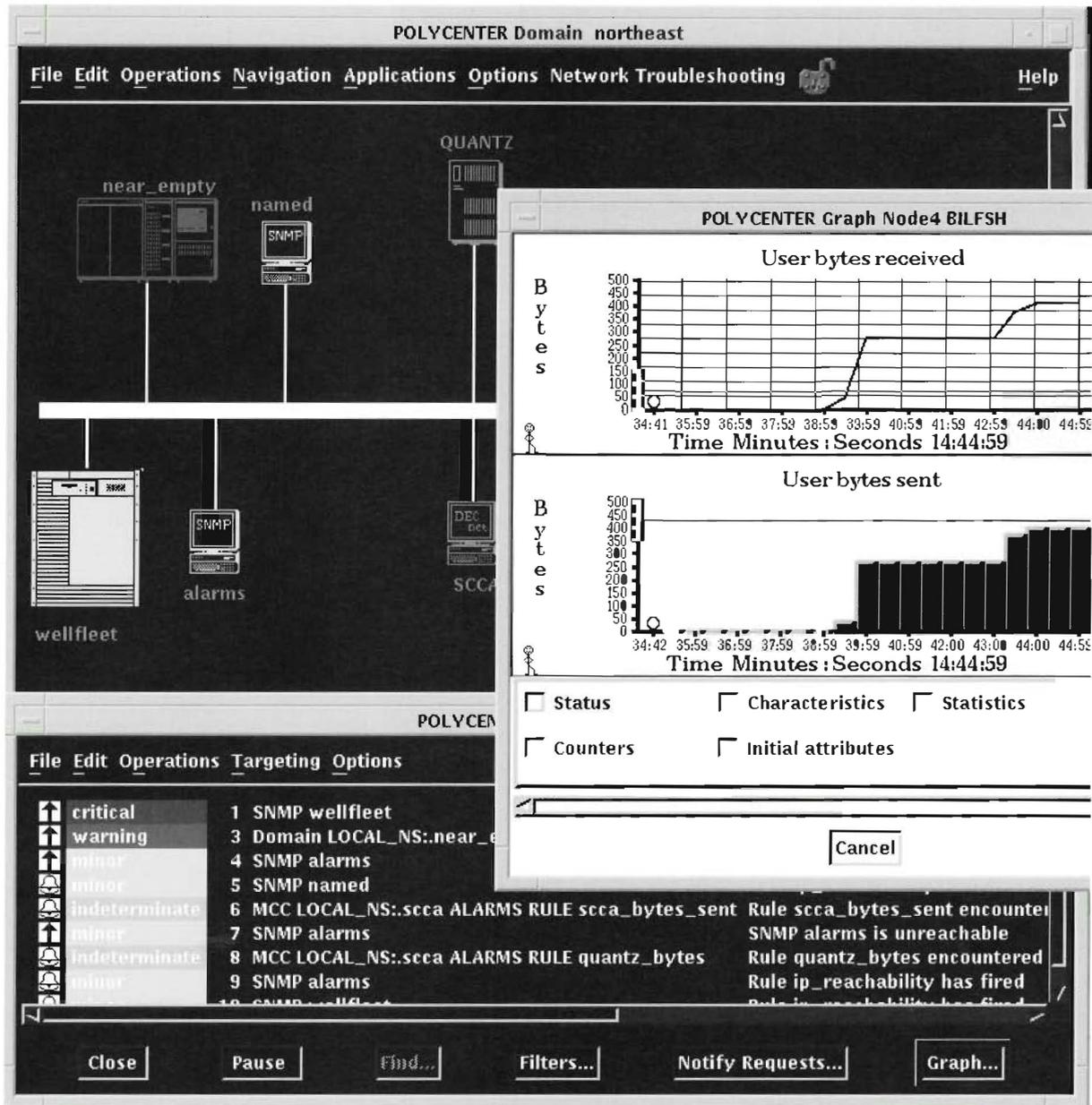


Figure 4 Screen Display of DECmcc Version 1.3

Future Work

Of course, work on a major software system such as the DECmcc director is never complete. There are many areas of opportunity for additional development. For example, DECmcc can be ported to other industry platforms (both hardware and software). New objects can be managed, not only in network management but also in system management, application management, data management, envi-

ronment management, telecommunications management, and so on. Commensurate with each of these general areas are technology-specific applications. In addition, further technology-independent generic applications can be developed. A recent paper describes how DECmcc can be considered as a distributed application and some additional work to make use of the DECmcc concepts in a distributed environment.²¹

DECMCC is not the only management director in the industry. Thus interoperability between DECMCC and other management systems is another area of opportunity. DECMCC already has links to other management systems, not the least being to manage IBM SNA systems.

Recent advances in object-oriented technology can be incorporated to enhance the object orientation of DECMCC.

Finally, new standard industry management protocols, new managed objects, and management framework innovations are always becoming available. DECMCC will be taking all of these evolutions in its stride. The distributed management environment (DME), still under development by OSF, promises to bring yet more technology to which DECMCC will adapt readily.

Summary

This paper has explained aspects of the design of DECMCC in the context of the state of the industry at the time. DECMCC has been a large undertaking, but we have been able to build and ship significant, consistent, integrated, and yet extensible, management capabilities covering a broad range of managed objects. The ability for DECMCC to adapt to the changing management environments underscores the benefit of adopting an architected approach to implementation.

Acknowledgments

The authors would like to acknowledge the work of the many people in the groups, past and present, responsible for bringing the ideas presented in this paper into practical reality in the DECMCC product set. Also, the detailed comments of two anonymous reviewers were very helpful.

References

1. M. Saylor, "Managing DECnet Phase V: The Entity Model," *IEEE Networks* (March 1988): 30-36.
2. M. Saylor, F. Dolan, and D. Shurtleff, "Network Management," *Digital Technical Journal*, vol. 5, no. 1 (Winter 1993, this issue): 117-129.
3. *EMA Entity Model* (Maynard, MA: Digital Equipment Corporation, Order No. AA-PV7KA-TE, January 1993).

4. *DNA (Phase V) Common Management Information Protocol Functional Specification* (Maynard, MA: Digital Equipment Corporation, Order No. EK-DNA01-FS-001, July 1991).
5. *DNA (Phase V) Network Control Language Functional Specification* (Maynard, MA: Digital Equipment Corporation, Order No. EK-DNA05-FS-001, July 1991).
6. L. Fehskens, "An Architectural Strategy for Enterprise Management," *IFIP Proceedings of the First Symposium on Integrated Network Management* (May 1989): 41-60.
7. M. Saylor, "Guidelines for Structuring Manageable Entities," *IFIP Proceedings of the First Symposium on Integrated Network Management* (May 1989): 169-183.
8. *Information Technology: Open Systems Interconnection: Common Management Information Service Definition*, ISO/IEC 9595 (Geneva: International Organization for Standardization/International Electrotechnical Commission, 1990).
9. *Information Technology: Open Systems Interconnection: Common Management Information Protocol Specification, Part 1*, ISO/IEC 9596-1 (Geneva: International Organization for Standardization/International Electrotechnical Commission, 1990).
10. N. La Pelle, M. Seger, and M. Saylor, "The Evolution of Network Management Products," *Digital Technical Journal*, vol. 1, no. 3 (September 1986): 117-128.
11. D. Shurtleff and C. Strutt, "Extensibility of an Enterprise Management Director," *Network Management and Control*, A. Kershenbaum, M. Malek, and M. Wall (eds.) (New York: Plenum Press, 1990): 129-141.
12. J. Case, M. Fedor, M. Schoffstall, and J. Davin, "A Simple Network Management Protocol (SNMP)," RFC 1157 (May 1990).
13. G. Stone, "Integrated Management Technologies," *AT&T UNIX Systems Management Symposium*, Spring 1991.

14. C. Strutt, "Dealing with Scale in an Enterprise Management Director," *IFIP Proceedings of the Second Symposium on Integrated Network Management* (April 1991): 577-593.
15. S. Martin, J. McCann, and D. Oran, "Development of the VAX Distributed Name Service," *Digital Technical Journal*, vol. 1, no. 9 (June 1989): 9-15.
16. A. Shvartsman, "An Historical Object Base in an Enterprise Management Director," *IFIP Proceedings of the Third Symposium on Integrated Network Management* (April 1993): 123-134.
17. C. Strutt and D. Shurtleff, "Architecture for an Integrated, Extensible Enterprise Management Director," *IFIP Proceedings of the First Symposium on Integrated Network Management* (May 1989): 61-72.
18. J. Borden, "Digital's Telecommunications Network Management Program," *Network Operations and Management* (New York: The Institute of Electrical and Electronics Engineers, 1992): 102-111.
19. *DECmcc System Reference Manual*, 2 volumes (Maynard, MA: Digital Equipment Corporation, Order No. AA-PD5LC-TE, AA-PE55C-TE, April 1992).
20. J. Fernandez and K. Winkler, "Modeling SNA Networks using the Structure of Management Information," *IEEE Communications* (May 1993).
21. C. Strutt, "Distribution in an Enterprise Management Director," *IFIP Proceedings of the Third Symposium on Integrated Network Management* (April 1993): 223-234.

Recent Digital U.S. Patents

The following patents were recently issued to Digital Equipment Corporation. Titles and names supplied to us by the U.S. Patent and Trademark Office are reproduced exactly as they appear on the original published patent.

5,117,352	L. H. Falek	Mechanism for Fail-Over Notification
5,119,043	R. W. Brown, M. D. Leis, and E. C. Simmons	Auto-Centered Phase-Locked Loop
5,119,402	S. A. Ginzburg and J. M. Rieger	Method and Apparatus for Transmission of Local Area Network Signals over Unshielded Twisted Pairs
5,119,465	M. L. Jack and R. T. Gumbel	System for Selectively Converting Plurality of Source Data Structures through Corresponding Source Intermediate Structures, and Target Intermediate Structures into Selected Target Structure
5,119,483	W. C. Madden, D. E. Sanders, G. M. Uhler, and W. R. Wheeler	Application of State Silos for Recovery from Memory Management Exceptions
5,119,484	T. F. Fox	Selections between Alternate Control Word and Current Instruction Generated Control Word for ALU in Respond to ALU Output and Current Instruction
5,120,603	P. H. Schmidt	Magneto-Optic Recording Medium with Oriented Langmuir-Blodgett Protective Layer
5,121,085	R. W. Brown	Dual-Charge-Pump Bandwidth-Switched Phase-Locked-Loop
5,121,260	G. J. Asakawa, R. Y. Noguchi, and J. Rinaldis	Read Channel Optimization System
5,121,382	H. S. Yang, M. W. Carrafiello, W. Hawe, and R. W. Graham	Station-to-Station Full Duplex Communication in a Communications Network
5,123,091	B. E. Newman	Data Processing System and Method for Packetizing Data from Peripherals.
5,123,306	N. S. Saunders and D. J. Moretti	Pin Pulling Tool
5,125,083	D. B. Fite, T. Fossum, R. C. Hetherington, J. E. Murray, Jr., and D. A. Webb	Method and Apparatus for Resolving a Variable Number of Potential Memory Access Conflicts in a Pipelined Computer System
5,125,086	F. L. Perazzoli, Jr.	Virtual Memory Paging Apparatus with Variable Size In-Page Clusters
5,126,964	J. H. Zurawski	High Performance Bit-Sliced Multiplier Circuit
5,127,006	K. Subramanian and M. A. Billmers	Fault Diagnostic System
5,136,700	C. P. Thacker	Apparatus and Method for Reducing Interference in Two-Level Cache Memories
5,150,197	W. R. Hambrgen	Die Attach Structure and Method
5,150,360	R. J. Perlman, W. R. Hawe, and A. G. Lauck	Utilization of Redundant Links in Bridges Networks
5,161,193	B. T. Lampson, W. R. Hawe, A. Gupta, and B. A. Spinney	Pipelined Cryptography Processor and Method for its Use in Communication Networks
5,179,577	N. Ilyadis	Dynamic Threshold Data Receiver for Local Area Networks
5,185,537	T. Creedon, J. Nolan, and E. O'Neill	Gate Efficient Digital Glitch Filter for Multiple Input Applications
5,193,151	R. Jain	Delay-Based Congestion Avoidance in Computer Networks
5,195,181	S. Bryant and M. Seaman	Message Processing System Having Separate Message Receiving and Transmitting Processors with Message Processing Being Distributed Between the Separate Processors

Referees, April 1992 to December 1992

The editors acknowledge and thank the referees who have participated in a peer review of the papers submitted for publication in the Digital Technical Journal. The referees' detailed reports have helped ensure that papers published in the journal offer relevant and informative discussions of computer technologies and products. The referees are computer science and engineering professionals from academia and industry, including Digital's consulting engineers.

Anant Agarwal, Massachusetts Institute of Technology
Brian Allison, Digital
Paul Beck, Digital
Lisa Bender, Digital
Brian Bershad, Carnegie-Mellon University
Dileep Bhandarkar, Digital
Meyer Billmers, Digital
Verell Boalen, Digital
Scott Bradner, Harvard University
Bevin Brett, Digital
Preston Briggs, Rice University
Dean Brock, University of North Carolina
Mark R. Brown, Digital
Randal E. Bryant, Carnegie-Mellon University
Lyman Chapin, Bolt, Beranek and Newman
John DiMarco, University of Toronto
James Duckworth, Worcester Polytechnic Institute
Hugh Durdan, Digital
Philip Enslow, Georgia Institute of Technology
Deborah Estrin, University of Southern California
Len Fehskens, Digital
David Fenwick, Digital
David Fite, Digital
John Forecast, Digital
Tryggve Fossum, Digital
Mark S. Fox, University of Toronto
Rodney Gamache, Digital
Rick Gillett, Digital
Michael Greenwald, Stanford University
Stephen Greenwood, Digital
James Grochmal, Digital
Robert Hagens, University of Wisconsin
Alf Hansen, Sintef
Steve Hardcastle-Kille, Isode

John Hauser, University of California
Bill Herrick, Digital
Hai Huang, Digital
Raj Jain, Digital
Ashok Joshi, Digital
Alberto Leon-Garcia, University of Toronto
Jeff Kalb, Maspar Computer Corporation
Kim Kappel, Georgia Institute of Technology
Paul Kinzelman, Digital
James Kirkley, Digital
Jeffery Kuskin, Stanford University
Paul Kyzivat, Digital
Mike Leary, Digital
Ian Leslie, University of Cambridge
Tom Levergood, Digital
David Lomet, Digital
Frank McCabe, Digital
John McDermott, Digital
Paul McJones, Digital
William Michalson, Worcester Polytechnic Institute
Peter Mierswa, Digital
Charles Mitchell, Digital
David Mitton, Digital
Fanya Montalvo, Digital
J. Eliot Moss, University of Massachusetts
Trevor Mudge, University of Michigan
Bill Noyce, Digital
Dave Patterson, University of California
Larry Peterson, University of Arizona
David Piscitello, Bellcore
George Polyzos, University of California
Brian Porter, Digital
James J. Quinn, Digital
Farshad Rafii, Babson College
Hemant Rotithor, Worcester Polytechnic Institute
Paul Rubinfeld, Digital
Peter Savage, Digital
Michael Schroeder, Digital
Will Sherwood, Digital
Robert Simcoe, Digital
Richard Sites, Digital
Richard Stockdale, Digital
David Stone, Digital
Joseph Tardo, Digital
Bob Taylor, Digital
Mike Uhler, Digital
Jake VanNoy, Digital
Wolf-Dietrich Weber, Stanford University
Kathrin Winkler, Digital

digital™



ISSN 0898-901X