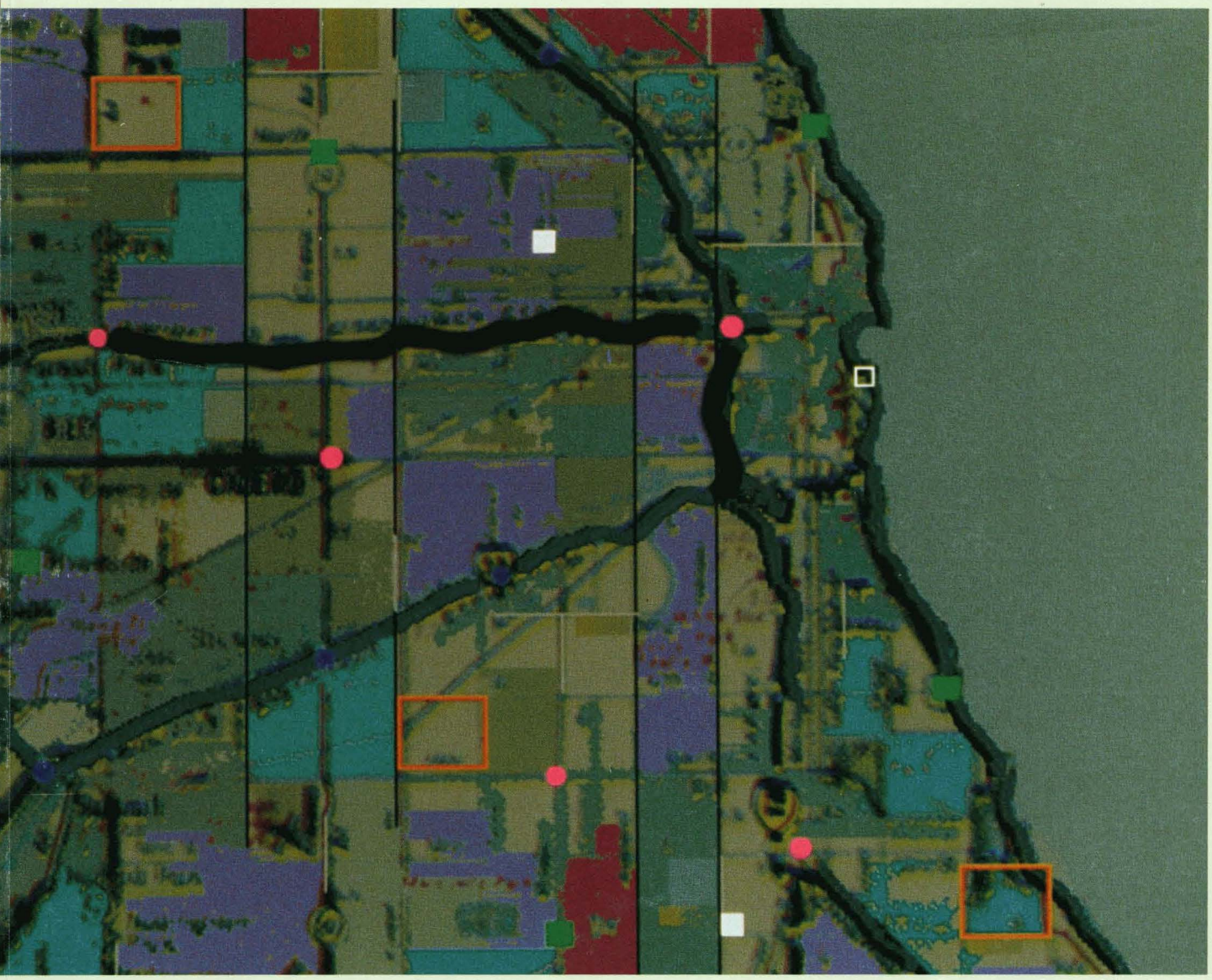
Distributed Systems

Digital Technical Journal

Digital Equipment Corporation



Number 9

June 1989

Editor
Jane C. Blake

Managing Editor
Richard W. Beane

Production Staff
Production Editor — Helen L. Patterson
Typographer — Rebecca A. Bombach
Designer — Patrick E. Conte
Illustrator — Deborah Keeley

Advisory Board
Samuel H. Fuller, Chairman
Robert M. Glorioso
John W. McCredie
Mahendra R. Patel
F. Grant Saviers
William D. Strecker
Victor A. Vyssotsky

The *Digital Technical Journal* is published by Digital Equipment Corporation, 146 Main Street, Maynard, Massachusetts 01754.

Changes of address should be sent to Digital Equipment Corporation, attention: List Maintenance, 10 Forbes Road, Northboro, MA 01532. Please include the address label with changes marked.

Comments on the content of any paper are welcomed. Write to the editor at Mail Stop MLO1-3/B68 at the published-by address. Comments can also be sent on the ENET to RDVAX::BLAKE or on the ARPANET to BLAKE%RDVAX.DEC@DECWRL.

Copyright © 1989 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. Requests for other copies for a fee may be made to Digital Press of Digital Equipment Corporation. All rights reserved.

The information in this 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 this document.

ISSN 0898-901X

Documentation Number EY-C179E-DP

The following are trademarks of Digital Equipment Corporation: ALL-IN-1, DEC/CMS, DECmail, DEC/MMS, DECnet, DECnet-DOS, DECnet-Rainbow, DECnet/SNA, DECnet-ULTRIX, DECnet-VAX, DECrouter, DECserver, DECserver 200, DECserver 500, DELNI, DEQNA, Digital, the Digital logo, EDT, LAN Bridge, LAN Traffic Monitor, MAILbus, Message Router, MicroVAX, MicroVAX II, MicroVAX 3000, PDP-11/70, RA, RA81, RD53, RD54, RSX, SA482, ThinWire, ULTRIX, ULTRIX-32m, ULTRIX-32w, VAX, VAX C, VAXcluster, VAX DEC/Test Manager, VAXELN, VAXmate, VAX Notes, VAX SPM, VAXstation 2000, VAX/VMS, VAX VTX, VAX 8200, VAX 8700, VAX 8800, VMS, VT, WPS.

BASIC is a trademark of Dartmouth College.

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

Intel is a registered trademark of Intel Corporation.

Microsoft and MS-DOS are registered trademarks of Microsoft Corporation.

SAS is a registered trademark of SAS Institute, Inc.

UNIX is a registered trademark of American Telephone and Telegraph Company.

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

Cover Design

The diverse resources of a distributed system and their physical separation are represented on our cover by a city's widespread and varied geography. Just as the areas of a city are made accessible by its highway system, so too are the distributed system's resources interconnected and made accessible by the network and its services.

The cover was designed by Barbara Grzeslo of the Graphic Design Department.

Contents

7 ***Foreword***
Butler Lampson

Distributed Systems

9 ***Development of the VAX Distributed Name Service***
Sally J. Martin, Janet M. McCann, and David R. Oran

16 ***Design and Implementation of the VAX Distributed File Service***
William G. Nichols and Joel S. Emer

29 ***Remote System Management in Network Environments***
David M. Griffin and Brad C. Johnson

37 ***The Evolution of the MAILbus***
Peter O. Mierswa

44 ***VAX/VMS Services for MS-DOS***
Alan Abrahams

51 ***The WAVE Tools Base for Protocol Testing***
Peter G. Viscarola and Jeffrey E. Watkins

61 ***Performance Evaluation of Distributed Applications and Services in the DECnet Environment***
Eugene Finkelstein and Richard A. Grawin

78 ***Measurement and Analysis Techniques for DECnet Products***
Vijay G. D'Silva and Ruei-Hsin Hsiao

87 ***Modeling and Analysis of the DECnet/SNA Gateway***
John P. Morency, Richard P. Pitkin, Ramasamy Jesuraj, and Ambrose C. Kwong

Editor's Introduction



Jane C. Blake
Editor

The network products featured in this issue of the *Digital Technical Journal* allow system resources to be shared throughout the network. Made up of different hardware systems, software systems, and many applications, the network is perceived by users as one system — a distributed system. Performance in this environment is critical; therefore this issue also includes several papers about methods and tools developed by Digital's engineers to evaluate product and system performance.

A distributed system's coherence depends largely on the ability to identify and locate resources. In the opening paper, Janet McCann, Sally Martin, and Dave Oran describe the VAX Distributed Name Service product, which provides a globally accessible directory of network resources. The authors give overviews of the architecture and implementation, and review the design decisions made to ensure that the service is fast, robust, and scales up as the network grows.

The files on the network are among its most valuable resources. Building on VAX DNS, VAX Distributed File Service gives users common access to files located on the network. Wick Nichols and Joel Emer discuss DFS design options and performance analyses, and present details of the DFS implementation.

Managing the variety of systems that reside on a network can be a complex task. However, as Dave Griffin and Brad Johnson explain, the network itself can be used to expedite certain system management activities. They relate how Remote System Manager facilitates, from a central point, software distribution, file backup, and system administrative tasks for numerous clients.

The next paper is about a set of products whose history extends back to the early 1970s. Peter Mierswa traces the development of the MAILbus family, which supports message exchange across a multivendor environment. A significant theme in the products' history is Digital's early commitment to standards for the exchange of electronic mail and messages.

Alan Abrahams then describes VAX/VMS Services for MS-DOS. This product seamlessly integrates the remote VAX/VMS file system into the MS-DOS environment. Alan discusses the file server and disk server designs, and a service that allows personal computers to be booted over the network.

The next paper is not about a network product but instead describes a reusable development tool, called WAVE, for testing sophisticated protocols. Peter Viscarola and Jeff Watkins present an overview of the script language and give an example of a WAVE-based test tool developed for DECnet routing layer protocols.

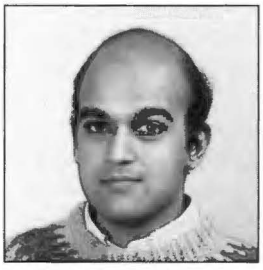
The final three papers address the methods developed for the essential work of evaluating distributed system product performance. Gene Finkelstein and Richard Grawin describe a new methodology for examining system-level performance in the DECnet environment. With case studies, they demonstrate the utility of this approach for product design and capacity planning. Next Vijay D'Silva and Ruei-Hsin Hsiao focus on the evaluation of individual product implementations. The authors outline key performance metrics and then describe the methods applied in the testing and analysis of the DFS and DNS products. In the closing paper, John Morency, Richard Pitkin, Ramasamy Jesuraj, and Ambrose Kwong describe a simulation model of the DECnet/SNA Gateway product, a model used primarily for capacity planning. The authors also discuss tools used to develop the model and the means to test its validity.

I thank Mary Ellen Lewandowski of the Networks and Communications Group for her help in preparing this issue.

Jane Blake



Alan Abrahams A principal software engineer, Alan Abrahams is a member of the Personal Computer Systems Group. He is currently a network architect responsible for integrating Microsoft's MS-NET and LANmanager into Digital's Personal Computer Systems Architecture (PCSA). Since joining Digital in 1981, he has been involved in the design of Digital's NETBIOS emulation, remote boot of MS-DOS systems, and the PRO/Communications package. Alan also represents Digital on the X/Open PC Interworking Group. He received a B.S. in computational and statistical science and a B.S. in mathematics from the University of Liverpool.



Vijay G. D'Silva Vijay D'Silva is a senior software engineer who is presently involved in the performance analysis and modeling of several networking products for the Network Performance and Conformance Engineering Group. Prior to his work on these future products, he evaluated the performance of the VAX DFS, VMS/SNA, and VAX FTAM products. Vijay joined Digital in 1986 after receiving a B.S. in electrical engineering (1984) from the Indian Institute of Technology, Bombay, India, and an M.S. in computer studies from North Carolina State University. His interests are in high-speed local area networks and computer systems modeling.



Joel S. Emer Since joining Digital in 1979, Joel Emer, a consulting engineer, has worked on VAX processor and memory subsystem architecture and performance, and systems performance analysis. He also led the distributed systems research team whose work resulted in the DFS product. Joel recently finished work on a research project at M.I.T. where he investigated issues in heterogeneous distributed computing. He holds a Ph.D. in electrical engineering from the University of Illinois, and B.S. and M.S. degrees in electrical engineering from Purdue University. Joel is a member of Tau Beta Pi, Eta Kappa Nu, ACM, and IEEE.



Eugene Finkelstein Formerly on the technical staff at AT&T Bell Laboratories and a professor at the Electrical Engineering Institute of Communications, Leningrad, Gene Finkelstein came to Digital in 1987. Currently a consultant to the Distributed System Technical Evaluation Group, he provides technical direction in the areas of testing and performance evaluation of integrated distributed systems. He is also leading the development of capacity planning guidelines for LAN-based distributed systems. Gene holds a Ph.D. and an M.S. in electrical engineering from the Electrical Engineering Institute. A senior member of IEEE, he has published four books and numerous papers in the field of communications networks.



Richard A. Grawin Richard Grawin came to Digital in 1987 from a performance analysis group at AT&T Bell Laboratories. While at Bell Laboratories, he analyzed the development of a local area circuit switch and participated in architecture and design trade-off studies. Currently, he is a principal engineer in the Distributed System Technical Evaluation Group and is the project leader responsible for the distributed systems capacity planning guidelines. Richard received a B.S. degree in mathematics from the University of Wisconsin in 1972 and an M.S. in computer science from Purdue University in 1982. He is a member of the ACM.



David M. Griffin Dave Griffin is a principal software engineer working with the VAX Distributed Name Service engineering team on a new major release of DNS. Prior to this work, he served as leader of the RSM advanced development team and project leader for RSM version 2.0. He has recently applied as co-inventor for a patent on aspects of the RSM design. Dave has also been an information systems consultant to several groups and received the Digital Information Systems Achievement Award in 1985 for his work on distributed application tools. He is a member of the ACM.



Ruei-Hsin Hsiao Ruei-Hsin Hsiao is currently leading several projects designed to evaluate the performance of products under development in the Networks and Communications Group. Since joining Digital in 1982, he has evaluated the performance of numerous Digital products, including terminal servers, DECnet products, Ethernet bridges, communication devices, and the DECnet System Services product set. Ruei-Hsin is a principal software engineer and holds a B.S. in communication engineering from the National Chiao Tung University in Taiwan, Republic of China, and an M.S. in electrical and computer engineering from the University of Massachusetts in Amherst.



Ramasamy Jesuraj A principal engineer, Ramasamy Jesuraj joined Digital's Networks and Communications (NAC) Performance Group in 1986. Currently he supervises the Mid-range System Performance Analysis Group and is working on the characterization of client/server product performance. Prior to his work in modeling and characterizing the performance of NAC products, Ramasamy was a senior engineer at Prime Computers and an assistant professor of computer science at the State University of New York. He holds a Ph.D. degree in applied mathematics from McGill University, and an M.S. degree in computer science from Concordia University, Montreal.



Brad C. Johnson Earning a B.A. (Magna Cum Laude, 1984) in computer science from Rutgers University, Brad Johnson recently received an M.S. in applied management from Lesley College. Before coming to Digital in 1986, he worked at Bell Laboratories and then at Data General where he led a project to develop a file server for IBM-compatible computers. Now a principal software engineer in the Distributed Processing Engineering Group, Brad is involved in designing a model for the next major version of RSM. He previously had project leader responsibilities for implementing RSM versions 2.0 and 2.1. Brad is a member of IEEE.



Ambrose C. Kwong Ambrose Kwong is a senior software engineer in the Network Performance and Conformance Engineering Group. Having previously worked on the development of the routing conformance test tools and the LL tester, he is currently developing DNS conformance test tools. He has also been involved in analyzing and modeling the DECnet/SNA Gateway performance, and in the performance analysis of the DECserver 100 and 200, and Ethernet terminal servers. Ambrose earned B.A. and B.S. degrees from the University of Rochester, and an M.S. from the University of Massachusetts. He is a member of IEEE, ACM, and Tau Beta Pi.



Sally J. Martin Sally Martin, a senior software engineer, joined Digital in 1980 and began work on advanced development projects related to the design of the Digital Network Architecture. She also served as a reviewer of the architecture specification. Now a member of the VAX DNS project team, Sally wrote the initial versions of the transaction agent and database and was responsible for the DNS version 1.0 background and advertisement modules. During version 1.0 field test, she directed the design and operation of the Easynet namespace, the largest namespace built to date using DNS. Sally holds a B.S. from Michigan State University.



Janet M. McCann Janet McCann has worked on the development of the VAX DNS product for over three years and is now the project leader responsible for implementing DNS version 2.0. Prior to the DNS project, she worked on the base operating system for the Professional 380 and was project leader for Pro/Synergy. Now a principal software engineer in the Networks and Communications Group, Janet came to Digital from Western Electric in 1982. She received a B.S. in mathematics from the University of New Hampshire in Durham and an M.S. in computer science from Boston University.



Peter O. Mierswa A member of the Mail Interchange Group, Peter Mierswa is chief designer responsible for the design and implementation of products that meet X.400 and X.500 international standards. In previous work, he was project leader for the DECnet-DOS family of products and headed the team that produced the first two releases of DECnet-VAXmate, DECnet-DOS, and DECnet-Rainbow. A consultant engineer, Peter previously published a paper about the DECnet-DOS system in the *Digital Technical Journal*. He received a B.S. (Summa Cum Laude) in computer sciences from S.U.N.Y. at Stony Brook, where the computer science faculty named him its best graduating student.



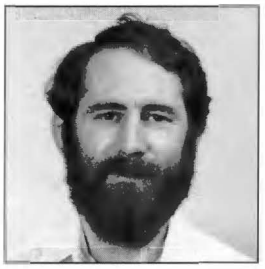
John P. Morency John Morency is a consulting engineer currently defining SNA interconnect capability in the area of on-line transaction processing. He has served as a principal technical contributor to Digital's IBM Interconnect Program for the past seven years and has also been actively involved in international standards work for on-line transaction processing. In other positions, John has provided worldwide technical support for DECnet, IBM, and X.25 products. Prior to joining Digital in 1978, John worked at IBM Corporation and the General Electric Company. He earned a B.S. degree (Magna Cum Laude) in mathematics and computer science from the University of Hew Hampshire.



William G. Nichols A consultant engineer in the Distributed Processing Engineering Group, Wick Nichols is currently working on the development of new distributed services. Prior to his work on VAX Distributed File Service, he served as project leader of a group that developed DECnet-10, and as project leader and supervisor for the DECnet-20 product. Before joining Digital in 1980, Wick was a system programmer for the Federal Reserve Bank of Boston and for the British Columbia Telephone Company. He received an A.B. from Harvard University in 1973.



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



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



Peter G. Viscarola Peter Viscarola joined Digital in 1987, first as architect and later as manager of the Networks and Communications (NAC) Conformance Engineering Group. He has represented Digital in the area of data communications conformance on a number of technical committees, including those at the Corporation for Open Systems, ANSI, and ISO. From 1979 until joining NAC, Peter ran a private consulting practice which specialized in data communications. Peter also worked at Digital in 1977, when he was hired as an instructor in Digital's Educational Services Group.



Jeffrey E. Watkins Jeff Watkins is a principal software engineer with the Networks and Communications Conformance Engineering Group. He is currently leader of the software development group that is creating conformance test tools for DECnet Phase V and OSI. He also represents Digital on the Corporation for Open Systems Achitecture Committee. Jeff is a co-designer of the WAVE tools base, as well as a major contributor to its implementation. He joined Digital in 1980 after earning a B.S. in computer science from the University of Illinois at Urbana.

Foreword



Butler W. Lampson
Corporate Consultant Engineer

A distributed system is several computers doing something together. This idea grew up naturally together with computer networking; it appears as one of the main goals in the late 1960s plans for the ARPANET. But building a distributed system turned out to be difficult, and networks have been justified instead by the value of a few very basic applications: terminal access, file transfer, and electronic mail. As workstations, personal computers, and local area networks have become widely available, however, the value of a distributed system has grown too obvious to ignore. In the last few years it has been increasingly clear that the future of computing lies here.

Customers benefit from a distributed system over a centralized one because they can

- Grow a system incrementally to meet increased demand, rather than replace it with a larger one
- Interconnect computers so their resources can be shared—especially the data they store—even when the computers are physically separate, independently managed, and different internally
- Get service more of the time, even when some parts of the system are broken
- Pay less for computing cycles in small packages

On the other hand, centralized systems provide a large number of functions and a high degree of global coherence: all the resources of the system can be accessed and managed in the same way from any part of it. These too are valuable properties. It is a major challenge to combine the virtues of both approaches.

VAX/VMS clusters offer all the benefits of a distributed system as well as the functionality and coherence of a centralized VMS system, and they have been very successful. However, clusters limit the number of machines that can be connected (a few dozen), and all the machines must run the same version of VMS and be managed as a unit. By contrast, a network such as DECnet can accommodate thousands of machines with many different architectures and operating systems, and it too has been very successful. However, a network offers few functions and little coherence. Users must be registered separately on each node, files cannot be named and accessed uniformly throughout the network, each node must be managed independently, and so forth.

A distributed system, then, is

- A set of hardware, software, and data *components*, possibly heterogeneous
- Connected by a network
- Providing a uniform set of *services* (user registration, time, files, records, printing, program execution, mail, terminals)
- With certain *global properties*: names, access, security, management and availability

The coherence that makes it a system rather than a collection of machines is a result of uniform services and global properties. The services are available in the same way to every part of the system, and the properties allow every part of the system to be viewed in the same way.

This definition of a distributed system assumes a single set of interfaces for the standard services and global properties. For example, every component of the system can be named, accessed, and managed in the same way. Further, every component that provides or consumes a service, such as file storage or printing, does so through the same interface. There still may be several implementations of the interfaces for naming, management, files, etc., and this variety allows the system to be heterogeneous. In its interfaces, however, the system is homogeneous. It is this homogeneity that makes it a *system* with predictable behavior rather than a collection of components that can communicate. If more than one interface exists for the same function, it is unavoidable that the function will work differently through the different interfaces. The system will consequently be more

complicated and less reliable. Perhaps some components will not be able to use others at all because they have no interface in common.

In reality, of course, there is no absolute distinction between a system and a collection of components. A system with more different ways of doing the same thing, whether it is naming, security, file access, or printing, is less coherent and dependable. On the other hand, it can still do a lot of useful work. The evils of heterogeneous interfaces can be mitigated by *gateways*, components that map from one interface to another. A gateway is often necessary when two systems evolve separately and later must be joined into one. An obvious example is the conjunction of IBM's System Network Architecture (SNA) and Digital's DECnet, and indeed there are several DECnet/SNA gateways.

This issue describes many of the major components that currently make up Digital's evolving distributed system product line: essential services such as files, printing, and mail, and global properties such as naming and management. Except for the naming service, these components were not developed from scratch but rather arose from previous products that were successful in standalone systems. In this way the distributed systems do not have to provide new implementations of all the functions they offer, but instead can build on existing products. In several cases these components act as gateways, for example, between VMS and existing interfaces in MS-DOS, or between DECnet and existing mail, file transfer, or communication interfaces in SNA. In each case the product is important as a tool to integrate existing systems into a larger and more effective whole.

Typically a distributed system involves a number of machines. It can be configured in many different ways. It is built on top of one or more operating systems and one or more implementations of a network architecture. All in all, there are a lot more ways for things to go wrong in a distributed system than in most centralized systems. As a result, careful modeling, performance analysis, and testing are essential to the engineering of a distributed system that works reliably and predictably. Nearly half of this issue is devoted to these topics.

The architectures, products, and engineering methods described here are significant steps toward the goal of coherent systems that encompass the full range of Digital's products. As the company's distributed systems evolve, they will do so from this foundation.

Sally J. Martin
Janet M. McCann
David R. Oran

Development of the VAX Distributed Name Service

The VAX Distributed Name Service (DNS) is a global, highly scalable directory system for computer networks. This robust name service provides fast translation of network-wide names into the values of attributes stored with the names. The DNS designers utilized such techniques as partitioning of the namespace and data replication to ensure service reliability and availability. For improved performance, DNS implements the client interface by means of a clerk which controls communication protocols and also maintains a cache.

Networks bring a large and sophisticated set of services to a diverse and growing user population. For this population to exploit the network's potential, the computing environment must be friendly, easy to navigate, and productive. One of the main impediments to achieving this environment is the increasing difficulty of identifying, locating, and accessing network resources. Much of the power of existing and future computer networks may go unused simply because users are either unaware of the facilities available to them or find the methods daunting. The difficulty can be expected to increase over time for the following reasons:

- Networks are getting very large. Thousand-node networks are now common, with million-node networks on the horizon for the world's larger public and private organizations.
- Networks are heterogeneous. More kinds of computers are participating in networks all the time — workstations, personal computers, minicomputers, supercomputers, and mainframes. Even more significant is the loose, heterogeneous composition of a large network that makes central control and management infeasible.
- Whole new classes of distributed applications are being developed. Applications such as network file systems and network-based electronic mail systems are already common.

  New applications, such as fully distributed databases, are being deployed today. Whole new models of computing in the distributed environment, such as remote procedure calls (RPCs), are being developed to make the construction of such applications easier.

To a great extent, the difficulty in accessing network resources arises from the lack of a consistent, globally accessible directory of network resources. In this paper, we describe a large-scale, production name service for DECnet networks called the VAX Distributed Name Service (DNS). We begin with a presentation of the goals of DNS, followed by a discussion of the kinds of names used. In the balance of the paper, we give an overview of the DNS architecture and implementation and describe the major issues designers addressed during product development.

The Genesis and Goals for DNS

DNS is a fundamental part of the DECnet System Services (DSS) product set and the Digital Network Architecture. DNS was under development for over five years within Digital. The project included a significant initial design phase (two years), a breadboard phase (one year), a final architectural design phase (one year), and a product implementation phase (one and a half years). This system was designed to meet a number of significant technical goals, as follows:

- Store enough names to allow the system to scale to networks of at least 100 thousand to 1 million nodes.
- Be very robust so that the name service is at least as reliable and available as the resources it is used to find.
- Be highly self-configuring to limit the amount of effort necessary to set up the service and keep it running.
- Be fast, so applications can always use the service without penalty to locate resources.
- Allow decentralized management to avoid the inefficiency of centralized management of all the resources of a large computer network.

A Global Name Service

DNS provides a global name service. A global name is one that can be used anywhere in a network, by any user, to access a resource in the network. In this section, we discuss the kinds of names such a network name service might support and place DNS in this context.

Kinds of Network Names

Figure 1 lists the kinds of names a network name service could support. A descriptive name is a construct that names a resource or object by specifying enough of its attributes to distinguish it from other objects. A descriptive name can be thought of as a query of the form "A color printer located near my office."

Descriptive names are most useful to human users who are searching the network for a resource that meets certain user-specified criteria. Descriptive name services are potentially the most powerful kind of name services, but they are the most demanding of computing resources and are difficult to distribute effectively. The Consultative Committee on International Telephony and Telegraphy (CCITT) and the International Standards Organization (ISO) are developing a standard for a descriptive name service. The standard is primarily oriented toward international electronic mail systems and is known as X.500.

A primitive name is a string of symbols that identifies a single object or resource in the network. A primitive name is similar to the name of a person or business. If one knows the name of the resource sought, the primitive name is an extremely convenient identifier to use to locate the resource or to discover more about it. A principal feature of primitive names is that they are unambiguous; no two objects or resources in a network can have the same primitive name at the same time. Because of their simplicity, name services based on primitive names are technically easier to design for speed, scalability, and robustness, which are three of the goals outlined for VAX DNS. Consequently DNS processes primitive names.

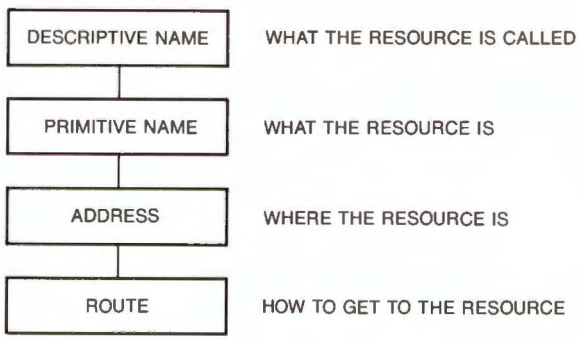


Figure 1 Kinds of Names in Network Naming Services

The third kind of name, an address, is a form of name that identifies a resource by its location in the network. For example, in DECnet Phase IV, all resources are identified by the node on which they reside. Although the node is named by a string of characters, in fact this string is a local user convenience and simply stands for the address. As a network grows, the relationship between the nodes and the resources residing on them becomes increasingly complex and arbitrary. It is unnatural to use addresses to name resources that are either replicated or that are not permanently tied to one location in the network. Reconfiguring resources as new nodes are added or old nodes removed invalidates the names they were using.

A route names a resource by enumerating the path from the accessing user to the location where the resource is located. Routes suffer all of the disadvantages of addresses as names, and other disadvantages as well. Route names are dependent on who is accessing the resource as well as where the resource is located. Different users must use different names to access the same resource. Worse, if any portion of the network is reconfigured, the path to the resource is affected and its name changes.

For the remainder of this paper, we restrict the discussion to primitive names, since those are the names processed by DNS.

Possible Structures for Network Names

Primitive names in a network name service could have a number of possible structures. The simplest of these would be a flat namespace in which the names are simply strings of symbols with no internal structure. (A namespace is the collection of names stored within DNS.) Flat names are easy for users to understand, but the assignment of unique names becomes increasingly difficult in large networks.

A second possible structure for network names is a tree. A tree is a type of graph in which each node has exactly one parent but may have any number of children. Computer file systems, such as those supported by MS-DOS, VAX/VMS, and UNIX operating systems, use a tree-structured namespace for naming files. In these systems, the tree is rooted; that is, a single node is the ancestor of all others. Trees allow a decentralized method of assigning names and give users a natural hierarchical scheme for organizing their network namespace. Trees suffer from a number

of deficiencies, however. The most serious is that a node in the graph can have only one name (because each node has a single parent).

To avoid the uniqueness limitation of trees, a third possible structure for network names is that of a directed graph. In this structure, names consist of paths from some initial node to the desired terminal node. Some computer file systems, such as UNIX, augment the basic tree with a capability called soft links. Soft links allow users to view the file system as a directed graph rather than a tree. Once a directed graph is permitted, it is possible for the graph to have cycles. Cyclic names are not terribly useful, hence most name services restrict the graph to be acyclic. Naming structures based on directed graphs have two major subtypes. The graph can be either rooted or unrooted. In a rooted graph, like a rooted tree, a distinguished node is the ancestor of all others and is globally known to all users. In an unrooted graph, no such node exists, and each user must choose a starting node in the graph from which to specify names.

DNS provides a rooted, acyclic directed graph as its naming structure. The underlying physical representation of the graph is in fact a tree, which allows simplification of the algorithms that perform distributed maintenance of the graph. The tree is then augmented with a special form of name, called a soft link, which allows the users to view the structure as a directed graph.

Architectural Overview

This section describes the architecture of DNS at a high level. Readers interested in exploring the design at greater depth may obtain the architecture specification.[1]

Properties of Names

A name is a string of characters that refers to an object or resource in the network. Names for objects are recorded in directories, which themselves have names. Some example names are as follows:

Parts.widgets.left-handed.SMOKESHIFTER
USGovernment:Treasury.Bills.CurrentSeries
Nodes.%XAA0045D4013C
Engineering.sources."all_sources.c"

A directory may contain three types of entries: object entries, child pointer entries, and soft links. An object entry consists of the object's name and a set of attributes for the object. Most prominent of these attributes is the network address where the object currently resides. Child pointer entries link the directories into a rooted tree in which there is a single path from the root directory, through a set of child directories, to the desired named object. A soft link is a form of alias, or indirect pointer, which allows a single entry to be reachable by more than one name. Soft links provide users flexibility by allowing a namespace to be viewed as a directed graph rather than as a pure tree. In addition, these links provide a graceful way to reorganize a namespace without incurring a massive invalidation of existing names. A tree of directories, starting at a root, is called a namespace. Multiple namespaces may coexist on the same network, but they are completely disjoint. Figure 2 shows an example portion of a namespace.

To ensure that the set of directories that make up a namespace in fact makes a tree at all times, the name service enforces two directory invariants:

- Each directory has exactly one parent at any moment.
- No directory is a child of any of its descendants.

The first directory invariant guarantees that the graph always forms a tree. The second invariant guarantees that the graph has no loops.

Partitioning and Replication

In a large network, it is infeasible to store all names in one central location. Besides being a single point of failure, a centralized service suffers from poor performance. First, the cost of accessing the service from distant parts of the network is high. Second, the service provider can quickly become both a processing and communication bottleneck to the entire network.

It is necessary not to allow single points of failure in the name service. The most appropriate technique for achieving this reliability is to replicate the data in multiple locations. Adding data replication to a system leads immediately to two crucial decisions that guide the architecture. The first is to determine what the unit of replication should be. One could decide to replicate each name individually; to replicate the entire database; to replicate individual nodes; or to tie replication to the partitioning in some way. The second issue is the effect of replication on data consistency. One must decide if all copies of the data should be kept tightly synchronized, as in a classic database system. If so, what kind of synchronization algorithm is appropriate. If the data is not synchronized, what consistency semantics should be supported.

For the DNS architecture, designers chose to store the namespace in a partitioned, partially replicated database. Database partitioning means that parts of

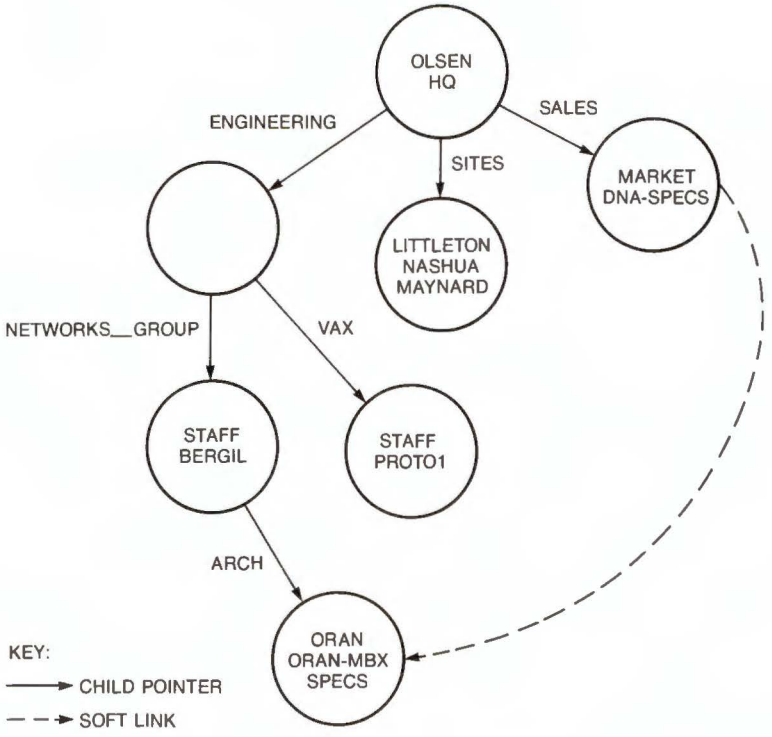


Figure 2 Example of a Namespace

the namespace are stored in different locations. By partially replicated, we mean that part of the namespace may be simultaneously stored in multiple database locations. The unit of partitioning and replication is the directory. A collection of (copies of) directories stored on a particular node is called a clearinghouse. The partitioning is accomplished by controlling which directories are stored in which clearinghouses. The replication is accomplished by storing a directory in more than one clearinghouse.

A copy of a directory stored in a particular clearinghouse is called a replica. To simplify the algorithms for name creation and general namespace maintenance, one of the replicas of a directory is designated to be the master replica for that directory. Creation of new child directories is permitted only through the master replica for the parent directory. Creation of object entries and soft link entries, in addition to any update or deletion, may be directed to another kind of replica storing the appropriate directory, called a secondary replica. A third kind of replica, the read-only replica, responds only to lookup requests and is not permitted to perform creations, updates, or deletions on behalf of clients.

The name service maintains a distributed database on behalf of its clients. However, unlike a typical distributed database, this database provides very loose consistency guarantees to allow high levels of partitioning and replication. A client may receive different answers depending on which replica of a directory is queried when updates are being propagated through the system.

Update Semantics

The exact update semantics for the namespace must be specified when the system is designed. The following must be decided:

- In the case of creating new names, what forms of mutual exclusion guarantees are provided. Can two users each try to create a resource with the same name and be allowed to succeed.
- Is each update an individual atomic action, or is some form of multioperation transaction provided as in a conventional database system.
- How are updates by multiple users at different times and locations serialized, if indeed serialization is supported at all.

DNS does not provide the usual database guarantee of mutual exclusion for name creation. An update can succeed if a single updatable replica is available. Instead of providing strong mutual exclusion, the system supports the notion of safe and unsafe names. When a user first creates a name, it is unsafe. Another user could create the same name in another

replica. While the name is unsafe, the name service will not necessarily detect this conflict. At some later time, through the operation of the background convergence algorithms, the name with the latest time-stamp becomes safe. All conflicting names disappear. Once a name is safe, no user can create the same name without first deleting the existing one. A user can request the name service to attempt to make a name safe at any time.

Updates to attributes of names are time-stamped and applied such that the one with the latest time-stamp wins. The update algorithms are designed such that all updates are idempotent and commutative; that is, multiple applications of an update have the same effect as a single application of the update, and updates applied in any order have identical results. These properties allow updates to be introduced into the system by users at multiple locations.

Convergence of Replicas

The algorithm for producing convergence among the replicas of a directory is called the skulker. Skulkers operate independently on each directory in a namespace. Successful completion of a skulk operation ensures that all replicas of a directory have absorbed all updates applied to any replica prior to the time the skulk started. The more frequently skulks are run, the more up to date are all replicas of a directory. Skulks are expensive, however, so there is a trade-off between resources (network capacity, processing, memory) and timeliness.

Clients can control the frequency of skulks either by adjusting the convergence attribute of a directory, or by initiating skulks themselves. The skulker operates as follows:

1. All updates made to the master replica and any secondary replicas since the last skulk are gathered and stored in the clearinghouse where the skulker is running.
2. Each of the gathered updates is then disseminated to all replicas of the directory.
3. Each replica is informed of the time-stamp of the latest update that all of them are guaranteed to have seen.

Since skulks are expensive, DNS provides a mechanism called propagation for rapid convergence in common error-free cases. When a user makes an update to a directory with high convergence, DNS makes one attempt to propagate the changes to all available replicas.

State Growth and Scalability

A network's growth should not be constrained by the ability of the name service to continue growing; therefore the name service must be highly scalable. In practice this means designers must avoid data storage techniques in which the amount of processing and storage grows too rapidly as the size of the namespace grows. It is essential to avoid algorithms or data structures that grow faster than linearly, and desirable for all processing and storage overhead to grow no faster than logarithmically.

By naming the clearinghouses, DNS can avoid the linear growth of state information which might occur at the root of the namespace to record the identity and location of all clearinghouses.

Clearinghouses have names so users may conveniently refer to them and so the name service can find clearinghouses by looking up their names in the namespace.

The naming of clearinghouses follows a set of rigid rules. Under these rules, a name lookup cannot fail with the error that the clearinghouse in which the directory storing the object entry could not be found. Specifically, the name service ensures that the following clearinghouse invariants are never violated during the normal operation of a namespace:

- A clearinghouse must either store the root of the namespace or store at least one directory whose name is closer to the root than the name of the directory in which the clearinghouse is registered.
- Each directory other than the root directory of the namespace must be replicated in at least one clearinghouse whose name is closer to the root than the directory's name.

The first clearinghouse invariant guarantees that the root of the namespace is reachable by starting at any clearinghouse. The second guarantees that every directory is reachable from the root without having to look up a clearinghouse which in turn would require looking up the subject directory.

Attributes Stored with a Name

Each entry in a namespace contains a set of attributes. These attributes can be used to store a variety of information that assists in locating and accessing a resource. Particularly useful information to store is the network address of a computer that contains the named resource. Also useful are various local naming and addressing components which are needed to access the resource once its location in the network is known.

Attributes are identified by strings called attribute names and are of two types. Single-valued attributes hold a single value at a time, and set-valued attributes can hold an arbitrary number of values at a time. Updates are applied to individual values of attributes, and attributes can be freely created and deleted by means of the update mechanism.

Implementation — A Client/Server Model

This section discusses the implementation of the Distributed Name Service Architecture on a VAX/VMS system.

Clients of the name service are programs that create, delete, retrieve, and modify information maintained by the name service. Clients access the name service through the client interface which is implemented by the clerk on the local node. The design of the clerk and the server are the subjects of this section.

Clerks are the components of the name service that reside on every node in the network. Clerks receive requests from clients to access the name service and determine the appropriate server to process the request. The clerk invokes the clerk-server protocol to communicate with as many servers as necessary to satisfy the request. It maintains a cache of the location of recently accessed directories and clearinghouses to improve performance.

The server itself processes requests received through the clerk-server protocol. If the server does not have the necessary information to service a request, it directs the clerk to another server that is more likely to have the information.

Design of the Clerk

The initial release of DNS was implemented on the VMS operating system. This section describes the clerk implementation.

To access the name server, a node requires the clerk, the advertisement-solicitation process, and the location of at least one name server.

The advertisement process is responsible for initializing the clerk's cache and periodically updating it. On startup, the advertiser creates a global section, which serves as the cache, and loads a configuration file into the cache. This file contains a list of namespaces and name servers to insert into the cache. The first namespace in the file is the default namespace for the node. The configuration file is the only way to initialize the cache on a wide area network (WAN) and is the management interface to the clerk.

The advertiser runs as a detached process. It listens on the Ethernet for advertisements from name servers. Advertisements contain the same type of information as the configuration file and are loaded into the clerk's cache. Advertisements provide the clerk with additional servers paired with their most recent addresses.

The clerk is written as a sharable image. It uses the cache created by the advertiser to locate servers and process client requests. In addition to the data loaded into the cache by the advertiser, the clerk adds data to the cache as the result of contacting a name server. Two types of information are cached: the names and locations of servers, and the names and locations of directories. This information enables the clerk to have up-to-date addressing data for contacting servers. Further, if the location of directories is known, the number of servers contacted can be reduced.

Since the cache is shared by multiple applications, it must be protected from both unintended updates and uncontrolled concurrent updates. The clerk prevents unintended updates by creating the cache in kernel mode, thus preventing access by less-privileged users. The clerk is a user-written system service, so it can raise its privileges and access the cache on behalf of less privileged clients. The clerk uses a simple binary semaphore to protect the cache from concurrent updates. The semaphore is stored in the cache.

The clerk also maintains a per-client DECnet logical link cache. Because of access control considerations, logical links cannot be shared by multiple clients. Since logical link establishment is expensive, a cache is needed to improve response time for successive requests to a server. The server disconnects idle links every 10 minutes. Either the server or clerk will disconnect links if resources are needed.

Design of the Server

Like the clerk, the initial release of the server was implemented for VMS systems. A server consists of the clerk, advertisement-solicitation, the transaction agent, and background processes.

The transaction agent is responsible for servicing clerk requests. The agent implements the clerk-server protocol and portions of the directory maintenance protocol. Some operations require the transaction agent to communicate with other transaction agents. This communication is effected through the clerk; therefore the transaction agent is a client of the clerk.

The background process implements the directory maintenance protocol and the update-propagation protocol. The background also communicates with other nodes through the clerk. Work is generated for the background process in three ways. First, requests from other background processes must be serviced. These are received by network connections. Second, requests are received from the transaction agent. Third, at least every 12 hours a timer goes off and the background scans the database for directories that need to be skulked.

The transaction agent and the background process have similar designs. The sections following describe the several design features these processes have in common.

Multithreading Both processes perform operations that access the disk or request information from other name servers. During this time the process is idle. To improve throughput, we needed to make use of this idle time and process multiple requests simultaneously. To accomplish this, we used asynchronous system traps (ASTs).

Both processes start running in user mode. They create the mailboxes necessary to communicate with management, issue an asynchronous read request, and wait. The user issues a start command to the management program which sends the request to the server. This causes the mailbox read to complete and to run an AST. The AST routine declares the process to be a network object and issues a read to the network. Then the routine reissues a read to the mailbox. Two threads of execution, that is, two ASTs, are now outstanding. For each new network connection, another thread is started to process its transactions.

A process known as forking is available to support asynchronous disk I/O and clerk requests. Forking allows a thread to suspend execution until some asynchronous activity completes, thus allowing other threads to run.

Database Both the transaction agent and background process were designed to be independent of the choice of data access routines. We wanted to be able to make changes if necessary in the future. After examining several techniques, we chose to use Record Management Services (RMS) indexed sequential files.

The name server requirements did not always match up with RMS features. First, RMS key sizes are a maximum of 255 bytes in length. Name server keys vary in length up to a maximum greater than 255 bytes. Therefore, we had to allow duplicates; and if a key is greater than 255 bytes, the name server performs its own sequential lookup to find the correct record.

Locking turned out to be our biggest problem. RMS provides extensive record locking capabilities for shared files. We relied on these capabilities to control concurrent access to the database by the transaction agent and background process. However, RMS does not recognize multiple threads of execution within the same process. If one thread locks a record for update and another thread attempts to access the same record on a different stream, the second request is rejected. Thus, the server restricts the operation of threads performing database updates. Once a thread locks a record for update, it cannot give up control to another thread until it unlocks the record.

Design of Management

Management of the name server is handled by two modules. Together these modules manage both the name server and the namespace. Management of the name server involves starting and stopping name servers and clearinghouses, and monitoring counters. These functions are parsed by the management control program and then sent to the management listener where the functions are performed. The two functions were separated to allow for remote management in a later release. Management of a namespace required a human interface to the clerk. The management control program provides this interface. A Network Control Program (NCP)-style command line interface was chosen.

A permanent global section is created by the management processes to maintain nonvolatile data. This includes the mapping from clearinghouse names to file names and the time-stamp generated locally. Since the transaction agent and background process also need access to this data, a shared global section is created. In addition, this section is used to maintain counters for the server. Counters are incremented by the transaction agent and background and are displayed by management commands.

Summary

Currently, DNS is in use throughout Digital and at many customer sites. Its use is expanded daily. DNS has proven to be robust in a network of many thousands of nodes. It successfully supports VAX Distributed File Service and VAX Remote System Manager, also described in this issue.

Acknowledgments

The authors wish to acknowledge the DNS development team who worked hard to release a version 1.0 product of quality. Team members include Larry Augustus, Claudia Peters, Arun Sastry, and Paul Pederson. In addition we would like to thank the DECnet-VAX group for its help, in particular, Paul Beck and Tom Harding. Thanks are also due to Wick Nichols for his careful review of this paper.

Reference

1. *DNA Naming Service Functional Specification* (Maynard: Digital Equipment Corporation, Order No. EK-DNANS-FS-001, 1988).

William G. Nichols
Joel S. Emer

Design and Implementation of the VAX Distributed File Service

The VAX Distributed File Service (DFS) provides fast, remote file access for VAX/VMS systems. DFS appears to be a local file service and thus requires no changes either to applications or to user commands. The DFS designers first examined the VMS file system to determine what type of service — disk, file, or record — would best achieve their goals. Having determined that a file service would meet the goals, they used a queuing network model to assess program response time and throughput for users in a multiple workstation environment. This model was used to assess a number of design alternatives. The resulting implementation is well integrated into the VMS operating system and consists of device drivers and a server process. This paper presents the design options, the model and the DFS implementation.

The VAX Distributed File Service (DFS) is one of the DECnet System Services (DSS) family of products. DSS products use network servers to provide services and functions that are often typical of a non-networked operating system. Use of these distributed services allows many client computers, and thus many users, to share resources and to work together effectively.

Specifically, DFS addresses two needs that arise when many users share a distributed computing environment. First, users in this environment need common access to files. Second, managing this environment can be simplified if many users' files are collected onto a common set of disks. There, the files can be managed by a common operations staff. DFS extends the local file system on a VAX/VMS system to include these common file stores.

The design of DFS was driven by three major goals. First, it should have high performance. Users should not easily perceive differences in the speed of their applications when they move files from local to remote devices. Second, the service should be transparent. User commands and applications should work in the same way whether they are accessing local or remote files. Thus, the existing programming interface and file system semantics should be preserved. When it is impossible to preserve existing semantics, the new semantic should avoid unexpected updates and other unpredictable results.

The third goal was that the service should scale up to handle large networks. Scaling encompasses three characteristics: large geographic extent, large numbers of active clients, and very large numbers of potential clients. To allow scaling, a file service has to work on either a local area network (LAN) or a wide area network (WAN). Successful scaling also implies that transparency should be unchanged as a network grows from two nodes to 100,000 nodes. Furthermore, performance should not be impaired by factors unrelated to nodes actively using the service, e.g., the number of potential user nodes, remote network partitions, or node failures.

This paper describes how engineers in the Distributed Processing Engineering Group designed and implemented DFS to meet these goals. The first section presents some options we considered in the initial design stage. The second section describes some of the performance analysis used to arrive at the current design and the third offers details of the implementation of the DFS design.

Design Options

To distribute any service, one must build software for the client system, where the application runs, and for the server system, where the service is provided. One can divide the overall service functionality between the client and server in many different ways. However, when dividing an existing service such as a file system, it is frequently best to choose an existing, stable interface rather than design a new one.

The existing VMS file system is implemented in three layers. At the lowest layer, the disk subsystem reads and writes blocks of data to a disk. At this level, the disk is viewed as a single array of fixed-

size blocks. At the middle layer, the file subsystem (FILES-11) organizes the disk blocks into files and file directories. At this level, the disk is viewed as a set of arrays of fixed-size blocks, one array for each file. At the highest level, Record Management Services (RMS) operates on structures within a file. At this level, the disk is viewed as potentially more complex arrangements of records within files.

The interfaces between these layers are well defined and thus make convenient points at which to divide the overall file system between client and server. The layers above the split remain with the client; those below are moved to the server along with the disk they control. Given the three layers in the VMS file system, the natural distributed service types are therefore a disk service, a file service, and a record service. A disk service sends disk names and disk blocks between client and server; a file service sends file names and file blocks; and a record service sends file names, record descriptions, and records. References 1 and 2 have examples of a disk service and a record service, respectively.

There are many characteristics to consider when choosing among these three types of distributed service. Based on our primary goals — performance, transparency, and scaling — we chose to implement a file service. We now outline the reasons for this decision.

Many factors govern the performance of a storage service. However, even some of the most elementary analysis shows distinctions between the three service types.

Our analysis has shown that in small networks with little sharing, the disk service has a performance advantage over a file service. In this environment the record service offers considerably less performance. These differences arise primarily because the basic operations for the three types of service are of different complexities. In fact, they impose increasing computational load at the server as one moves from the disk service through the file service to the record service. More detailed analysis of the performance contrasts, including factors such as communication load, can be found in references 3 and 4.

Distinctions between the services with respect to both performance and transparency can be seen when file sharing is considered. The VMS file system semantics demand that shared access to files and records within files be coordinated. If the server does not provide this coordination implicitly, then the clients must coordinate their access to the shared data explicitly. This coordination usually involves extra communication that can degrade performance, especially when there is significant contention for the data. Alternatively, the service might disallow shared access, thus reducing the communication overhead, but also reducing transparency.

The performance impact of sharing decreases as one successively considers a disk, file, and record service. With a disk service, many accesses require explicit coordination. For example, clients must coordinate accesses to the directories and allocation tables that make up the file structure on a disk. This coordination occurs frequently, because file structure information is shared even when no users are sharing files. Only by having a read-only file system can these costs be eliminated.

A distributed file service does not require that the clients explicitly coordinate access to the file structure on a disk. The clients send file-level requests to the server. The server then serializes the requests and handles any interlocking within itself. This implicit coordination saves coordination communication over that required by a disk service. To allow sharing of records within a file, a file service requires the clients to coordinate explicitly. Finally, a distributed record service does not require explicit coordination for accesses to either file or record structures. Thus, such a service performs well when there is significant record sharing.

The mechanisms for coordinating sharing also have an effect on the ability to scale the service. A service with implicit coordination scales very well. The scaling is primarily limited by the ability of the server to handle the active clients' load. The situation is more complex when explicit coordination is required.

There are two basic schemes for handling explicit coordination: centralized and decentralized. In centralized schemes, all clients coordinate through a single coordinator. As with implicit coordination, the ability to scale is limited by the ability to handle the active clients' requests. Centralized coordination, however, requires more communication than implicit coordination because distinct messages are required. As the network is scaled up to a WAN, performance is degraded even more than in a LAN. The higher message latencies typical in a WAN make delays resulting from these extra messages even more significant.

In decentralized schemes, the coordination information is distributed among the participants. Decentralized schemes tend to be complex, because such schemes need protocols for joining and leaving the set of active clients, for locating the coordinating information, and for rationalizing the distributed state when nodes crash or are booted. These protocols

add more messages, which impair performance, especially in a WAN. In fact, the performance of these protocols depends not only on the number of active clients, but in part on the number of potential clients as well. Thus, decentralized schemes scale less well than centralized schemes.

Actually, there is an alternative to the type of file service described above. This alternative does not keep state information about requests or open files in the server. In the event of a server crash, this absence of state information can be useful. It allows servers to recover simply and allows clients to continue processing without interruption once the server recovers. Unfortunately, without additional coordination, this absence of state information also prohibits the kind of sequencing guarantees and file sharing integrity required by the VMS system. If we add a centralized coordinator, then this alternative type is no longer stateless. In addition, it then has extra message costs over a file service with implicit coordination. If we use decentralized coordination, scaling becomes a problem again. Thus, we will not consider such a service further.

There are other factors that affect the ability of a service to scale and that show contrasts among the various service types. File security is one such factor. Security is a particular concern for larger networks in which different machines are likely to be operated by different (mutually suspicious) organizations. In each type of service, the client operating system can use network security techniques to authenticate its identity and the identities of its users.[5] Once these identities have been established, the client is responsible for associating each request with the user who made the request. Even given such authenticated requests, the different types of service provide different levels of protection for the data they hold.

A disk server can only police access to whole disks. Thus, once it allows a client access to the disk, that client has access to everything on it. This implies that a disk server trusts its clients' operating systems to protect not only the contents of files from illegitimate access, but also the file structures that define the files. Such protection might be insufficient in cases where an organization has some distrust of the operating systems of potential clients.

A file server can police access at the file level and is in complete control of the file structures on its disks. With this information, the file server can check a user's authority to access individual files. Such control is sufficient so long as one trusts the clients to properly manipulate the contents of each individual file. Finally, a record server can increase the protection provided by a file server to include both the file and the individual records within the file.

Another scaling factor to consider is the restrictions on software evolution imposed by interoperation between computers. Operating system software in a large network will necessarily evolve unevenly, and more than one operating system version will be in use at any one time. These software versions may differ along two dimensions: in the interpretation of common data structures and in the features each version offers to applications.

A distributed disk service does not support uneven evolution very well. Consider the case in which clients that are running different versions of the operating system access a single disk server. Each client may have a different expectation of the file structure on the disk or of the coordination protocol between the clients.

The problem of uneven software evolution is more tractable in file and record services. In a file service, only the server operating system reads or writes the file structure or allocation tables on the disk. Therefore the server can guarantee the integrity of its directory and file structures without regard to the client's version. A record server can protect its directory and file structures and can also guarantee integrity of the record structures within files.

Another aspect of software evolution is the evolution of the service itself. New versions of either clients or servers can introduce new functions. If a new client communicates with an old server, or vice versa, there can be a mismatch in the list of functions supported. A distributed service must be able to adapt to these mismatches.

Here the disk service has the advantage of having the simplest protocol and therefore the one least likely to be augmented over time. The advantage of simplicity extends to the file service, which is simpler than the record service.

Implementation considerations affect the ability of the three service types to fit transparently into a VMS system. The three types differ in their ability to support existing user commands and applications. The record service fits least well, since it is only available to those programs that use RMS to access files. A file service is better, because it handles any applications that either use RMS or access files using the file system interface. A disk service also supports those system programs, such as BACKUP, that sometimes view a disk as an array of blocks rather than as a set of files.

Each of the service types could meet our goals to some degree. In most cases, we found a gradient from disk service through file service to record service,

with each goal being met in greater or lesser degree as one moves along the gradient. For example performance and transparency were best in disk and file services; scaling and security were best in file and record services.

We decided to implement a file service, because it met most of our goals without seriously compromising any of them. As noted above, a file service does require that the clients cooperate with one another if they are to update files simultaneously. We noted that many applications did not require simultaneous update, so we deferred this capability to a future release. At present, shared access is disallowed to prevent unexpected updates. Our experience since releasing DFS makes us believe that we were justified in making this decision.

Performance

As part of the DFS design, a detailed analysis of several design alternatives was undertaken. This analysis allowed us to study the practicality of remote data access by assessing its performance. It also allowed us to make informed selections among the design alternatives. These selections were based on issues such as design complexity and relative performance. In this section, we first discuss the metrics used in the performance analysis and then describe the model we developed. The section concludes with examples of how performance results obtained from the model influenced our final implementation of DFS.

Performance Analysis

In the DFS analysis, we assessed performance by using a metric that reflects a user's primary perception of the performance of the system; that is, how long does it take for a job to complete. Thus, response time was our most important metric in assessing performance. At the same time, other measures were also useful, such as the utilization of the various resources. Such metrics provide a quantitative explanation of the performance, for example, by showing the location of bottlenecks.

Many performance characterizations are based on response-time or elapsed-time measurements of a single user making requests to a service. Although the performance for a single user is an important measure, it can be a misleading characterization. For a file service, the server is a shared resource that is accessed by multiple clients. The effect of queuing for the various resources at the server can have a significant effect on the performance seen by users of the server. Thus, an assessment of the performance of the system under load is needed to provide complete characterization of the system.

Finally, we also wanted to estimate how many users can be supported with satisfactory response times. For this analysis, we compared the response time given by the distributed file service to the response time offered by a local disk. Just as performance degrades as the number of users increases in a multiuser timesharing system, file service performance degrades with an increase in the number of users. This degradation results from contention for the disk and server CPU. The file service also suffers due to latency in the network. To account for these factors, we accept a response-time increase of 25 percent over that of a single workstation with a local disk.

Performance Analysis Model

For contrasting design alternatives, it is too time consuming to build many systems with different designs. Even for a given design, large multiple-client testing is very time consuming. Therefore, we developed a queuing network model to assess the performance of the file service design alternatives quickly and quantitatively. The model represents a distributed system comprising the file server and multiple single-user workstations as client machines. Since we were primarily interested in the performance of the file server, the queuing network model and the workload we describe here represent multiple clients making requests only to a single DFS server.

Some important characteristics of the DFS communication mechanism affect the model of the system. The DFS protocols are connection-oriented, with long-living connections between client and server machines. Also, these protocols generally have a strict request-response nature, so that at any time a client will usually have only one outstanding request to a server. As a result, the number of requests outstanding in the overall system (that is, requests being processed at the server or being processed or generated at the client) corresponds to the number of single-user client workstations in the system. Because of this characteristic, it is convenient to use a closed queuing network model to represent a distributed system in which multiple clients request service from a DFS server. In this model, the number of customers in the queuing network corresponds to the number of workstations.

Client requests to the server may experience queuing at each of the individual resources at the server node. The delays caused by queuing reduce the system performance as seen by the user. Therefore, we represent the server in substantial detail; each service center that represents a resource at the server also has a queue for requests that may await

service at that resource. The server resources being modeled are

- The network interface (The transmit and receive portions are each modeled separately.)
- The CPU
- The disk subsystem at the server

Since no queuing occurs at the client, the various segments of processing taking place at the client are represented as delay servers, i.e., service centers with no associated queue. A simple view of the distributed system is shown in Figure 1.

A particular system design and workload may be modeled by substituting the appropriate parameters into the model. For determining many of the system design parameters, we used a prototype file service that was developed in corporate research.[6]

The prototype had already demonstrated the feasibility of a VMS file service and some of its performance potential. This prototype itself represented only one design alternative. To estimate the performance of the other design alternatives, we used the prototype to provide parameters for our models.

All the models have the same set of service centers, but the centers are distributed differently in the various models. Therefore, we measured the time taken to execute the parts of the prototype that correspond to the service centers in our model. We could then analyze the models with real parameters to predict the performance of various design alternatives.

We constructed an artificial workload to drive our model. The workload was based on our understanding of typical user behavior and was also derived from measurements of typical timesharing environments.[7,8] A user at a client workstation was considered to invoke program images repeatedly. The programs invoked by the user require access to some number of files. The programs alternate between performing

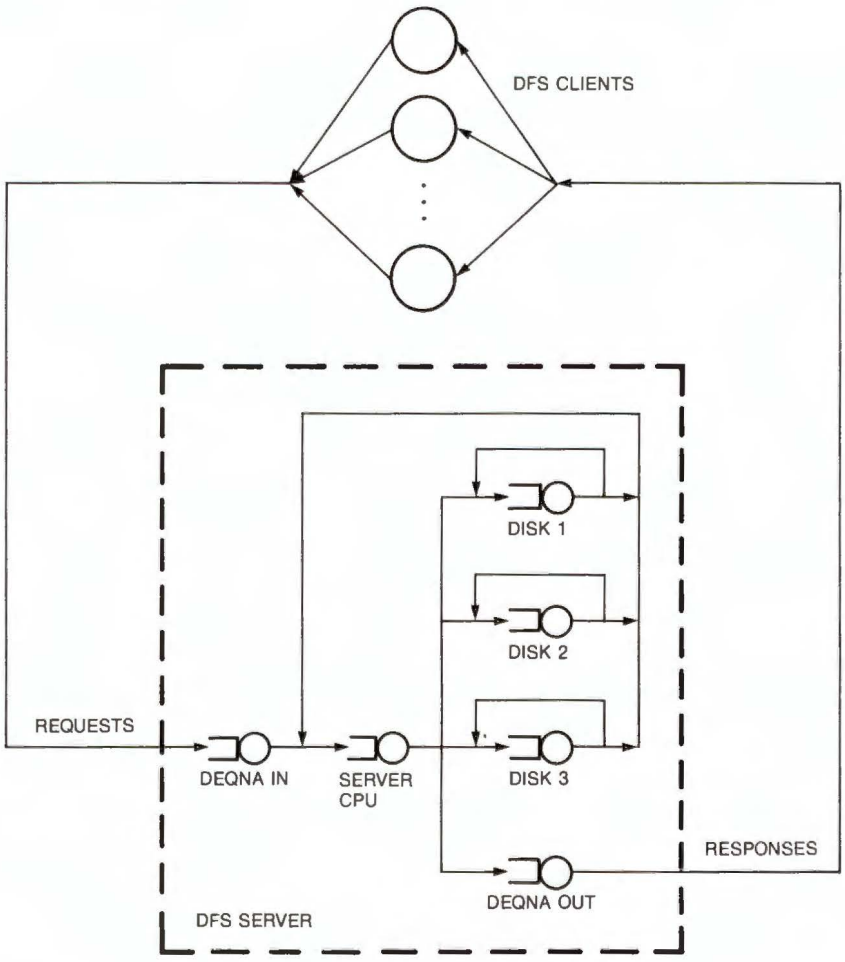


Figure 1 Simple System Queuing Model

local computation and doing system service operations for access to these files. These files are assumed to be stored at the file server.

The amount of data transferred and the processing done at the server for each request depends on the type of the request. The model distinguishes two types of request: control operations and data access operations. Control operations are operations such as open and close file which have a high computational component. On the other hand, data access operations are simple reads and writes. Data access operations usually have low computational requirements but require larger data transfers.

In between the program invocations, we assumed that the user spends a certain amount of time thinking or doing processing unrelated to our investigation. All processing at the client is represented in the model as delay servers with the appropriate branching probabilities.

To make the modeling tractable, we assumed that the service time at each service center is exponentially distributed. Since we were interested primarily in a file server's mean performance, this was an acceptable assumption.[9] Even given this assumption, the model had to distinguish control and data access operations.

Unfortunately, these two classes of operations have different service times at the various service centers. This difference means that the queuing network model does not satisfy the requirements for a product-form solution and may not be solved exactly.[10] We used a simple approximate-solution technique to solve this closed queuing network model. This technique is an extension of the basic multiclass mean value analysis technique.[11,12] The model characteristics, parameters, and the solution technique are described in greater detail in reference.[4]

Performance Results and Design Alternatives

The queuing model described above was used to extrapolate the measurements of our prototype distributed file service into predictions of the performance of various file server designs.

The performance measurements of interest were

- Program response time (time to execute a program, including remote data accesses)
- Program throughput (total number of programs executed per unit time)
- Utilization of the service centers at the server node
- Aggregate bandwidth that is used on the Ethernet

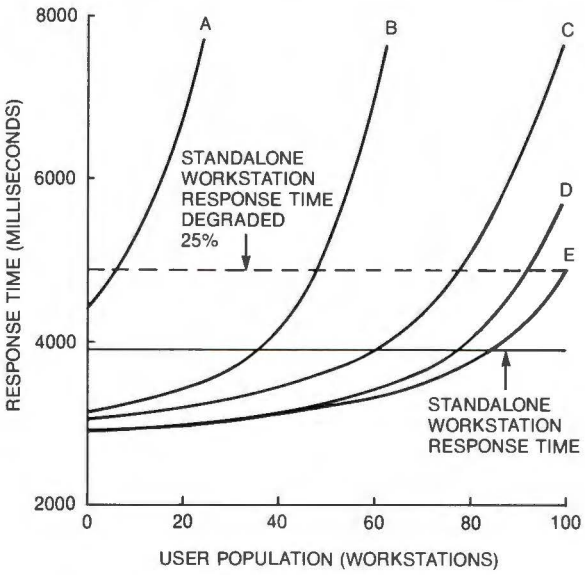


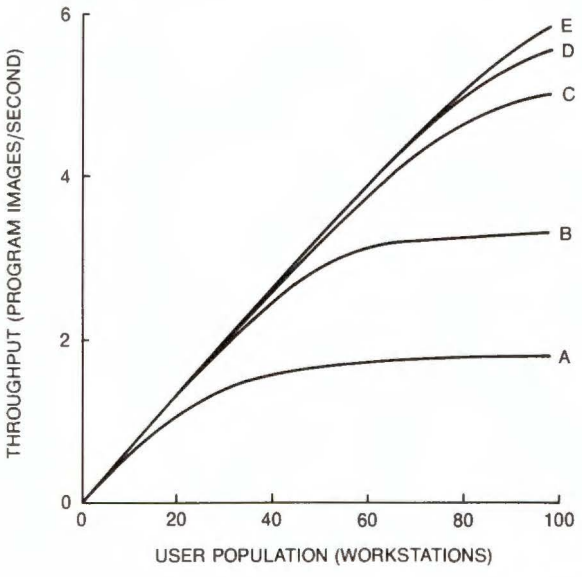
Figure 2 Program Response Time

Each performance measurement was calculated as a function of the number of client workstations.

We first considered the performance of a hypothetical configuration that we called the baseline system. The baseline system was a naive server implementation. The server in the baseline configuration consisted of a single MicroVAX II CPU as the server processor, a DEQNA interconnect as the network interface to the Ethernet, and three RD53 disk drives. The clients were single-user MicroVAX II workstations. We chose this configuration because at the time it was representative of a moderately priced system.

The implementation of the server in the baseline configuration assumed that (1) the processing of requests at the server was done in a normal user process, (2) file blocks were not cached or prefetched, and (3) each request or response message was copied exactly once by the server processor. These copies were in host memory between server message buffers and network packet buffers.

Examination of the analysis of the baseline system shows the type of results the model can generate. Curve A in Figure 2 shows the user response time as



KEY:

A - BASELINE CONFIGURATION

B - DATA CACHE INCLUDED

C - SAME AS B, PLUS KERNEL-BASED SERVICE

D - SAME AS C, PLUS FILE-OPEN CACHING

E - SAME AS D, PLUS GATHER-TRANSMIT

Figure 3 Program Throughput

a function of the number of workstations in the baseline configuration of a file service. The program throughput for the baseline configuration is shown as curve A in Figure 3. The flattening of the curve indicates that some resource is being saturated. Finally, Figure 4 shows the utilization of the various service centers at the file server node. This figure shows that the disk subsystem is the bottleneck and is being saturated. As a matter of interest, if the Ethernet controller utilizations are converted into throughput, the Ethernet throughput for the baseline configuration reaches 1.4 megabits (Mb) per second.

To estimate capacities, we compare the baseline response times to that of a standalone workstation. The response time of a standalone workstation is indicated by the horizontal solid line in Figure 2. A standalone workstation response time degraded by 25 percent is indicated by the horizontal dashed line in Figure 2. The intersection of the dashed line and the baseline system response time curve indicates the number of users that can be supported with the higher average response time. Applying this criterion to the baseline configuration, the model indicates that the file service can support approximately 10 users.

The baseline system did not exhibit satisfactory performance. We predicted that a naively implemented server would provide a maximum throughput of 1.8 program images per second. Moreover, the disks would saturate with just over 40 user workstations. This baseline system also would not support many users with an acceptable response time. In fact, the operating point would be beyond the knee of the curve, i.e., the bottleneck resource was becoming saturated. Thus, this configuration would not allow for expansion or variance in the user behavior; nor would it support the transient demands that users might place on the system.

Fortunately, however, a number of design alternatives can be considered to alleviate the saturation of the bottleneck resource and thus improve performance. The performance enhancements we considered, and which are described in more detail below, were as follows:

- Save frequently reused information (caching)
- Avoid context switches and dispatches across interfaces or protection boundaries that require large amounts of CPU time
- Save data copies

Because the disk was shown to be the bottleneck, an obvious step to improve system performance was to consider in-memory caching of data at the server. Caching data blocks has the dual benefit of saving disk and CPU utilization. Disk utilization is reduced because cache hits do not require disk access. Similarly, the large amount of CPU utilization imposed by a disk access system service is saved on cache hits.

For caching file data, we assumed a 90 percent cache hit rate. Other implementations of caching have predicted such cache hit rates for write-behind caches.[8] We assumed that the cache hit rate is independent of the number of users. Therefore, there may be a need to adjust the cache size to achieve the assumed hit rate.

The response-time results with caching are shown as curve B in Figure 2. We see that the response time is considerably reduced. There are two reasons for the substantial improvement in response time: the reduction in the disk service time, and the improvement in the CPU service time. Utilization plots show that caching results in a considerable reduction in disk subsystem utilization. Furthermore, the bottleneck switches from the disk to the CPU.

With the bottleneck now the CPU, we looked at ways to reduce CPU overhead. Significant CPU time was used to pass requests for service at the server between the kernel and a server process. Thus, to further improve performance, we considered moving

the processing of some requests into the kernel. For this alternative, we assumed that all data access requests were handled entirely in the kernel. Because of the complexity of performing file-open operations in the kernel, however, control operations on the file service were still sent to a server process.

Curve C in Figure 2 shows the effect of including a kernel-based request service and caching. This enhancement shows a significant improvement over the baseline configuration with caching alone; the resource utilizations are much better balanced. With this design the server can support a substantial number of users. Curve C in Figure 3 shows the effect of CPU saturation on the program throughput for the file service.

Even with kernel-based request handling, the file service is still hampered by CPU saturation. This high CPU utilization is largely due to the control operations performed at the server. Therefore, we considered the benefits of caching file-open operations at the server. Note that if a server process is caching file-opens, the ability of other users on the server node to access files may be affected. Thus, a file service with file-open caching may need to be located on a dedicated server node.

Although we did not have any quantitative analysis of the potential efficiency of such a caching strategy, we conservatively estimated a 50 percent hit rate. We believed file-open caching could be this effective for two reasons. There is a high probability that multiple users will use the same set of files, such as system files. There is also the probability that users tend to exhibit a certain amount of locality in the files that they access over short periods of time.

In curve D in Figure 2, we see that file-open caching does indeed significantly improve the performance of the file service. At low loads, a major reduction in response time is achieved by the reduced probability of a disk visit for file-opens. At higher loads, the substantial improvement results from the reduction in the service demand caused by file-opens at the CPU, which is the bottleneck resource.

The design enhancements just described addressed performance losses that were fixed on a per-request basis. However, with the CPU the bottleneck, examination of the components of the CPU time showed significant incremental costs associated with data movement. One such cost is the time for copying information from one memory location to another. Some copying is done to gather requests and responses for transmission as a network message. Often there is more copying to append network message headers and segment the data into individual packets for transmission on the network.

The time spent copying data can be reduced by various schemes. These include passing descriptors for segmented components of a message, or taking advantage of the ability of the network interface hardware to gather together (using gather-transmit) the segmented components of a packet.

The response times achieved when a gather-transmit interface is used are shown in curve E in Figure 2. This alternative reduces the CPU utilization (by eliminating data copies) and as a consequence reduces the response time.

The DFS implementation includes data caching and kernel service for read and write requests. DFS did not include open-file caching so that applications coresident with the server can access the files. However, to save CPU time on file-opens, DFS does cache certain information to make file-opens faster. This is described in more detail in the next section. Finally, current implementation constraints forced us to defer using gather-transmit.

The DFS Implementation

In the section Design Options we gave a brief description of the three layers of the standard VAX/VMS file system: RMS, the FILES-11 file subsystem, and the

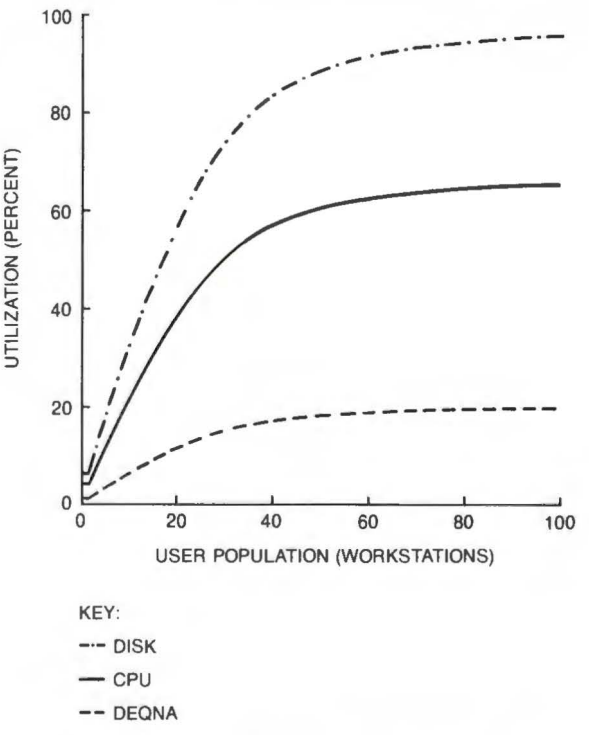


Figure 4 Server Resource Utilization with Baseline File Server

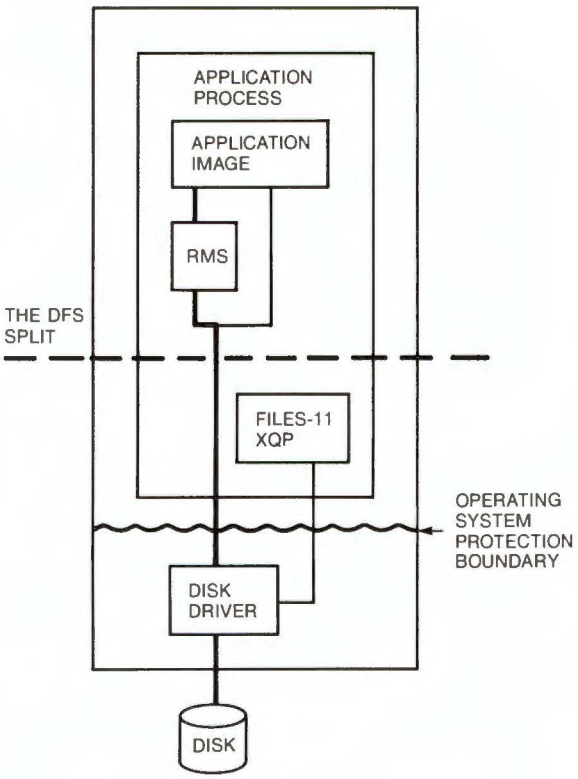


Figure 5 Standard VMS File Access

disk subsystem. In this overview we first add some detail about the interface between RMS and the file subsystem. We then show how DFS fits into that interface. To show this, we follow the flow of control from the application on the client CPU through the two parts of the DFS server and then back to the application.

Most applications call RMS when they need to access a file. RMS runs as part of the application process, as if it were a subroutine package. RMS manages records within files, and relies on the file subsystem to manage the files and to read or write blocks of file data. An application can also bypass RMS to use the file subsystem directly. Since RMS is essentially a subroutine package, the file subsystem sees no difference between RMS requests and direct application requests.

Application calls for file subsystem operations use a special system service, called a QIO, to cross the operating system protection boundary. When an application issues a QIO request, the VMS system collects all necessary information about the request into a packet. Then, if the system finds that the requested device is busy, it can queue the packet and come back to it later when the device is available. As processing goes along, intermediate status is stored in the packet. When the request is complete, the system returns the status from the packet to the application.

QIO requests for a file-structured device are sent to the disk driver. The disk driver separates requests for reads and writes from requests for control operations (such as open, close, extend file, and truncate file). For read and write requests, the driver translates file block addresses to disk block addresses and then sends them directly to the disk. The driver sends control operation requests to the FILES-11 XQP, which executes them. The XQP is a subroutine library that runs as part of the application's process, but has no direct contact with the application or with RMS.

Figure 5 shows a simplified block diagram of the standard VMS file access control paths. Modules below the protection boundary (the wavy line) run in operating system context while modules above it run in process context. The application image, RMS, and the FILES-11 XQP all run as part of the application's process; thus in the diagram, they are contained in the process's box. Two lines of control descend from the application image, a thick line to show the normal path through RMS and a thin line to show the optional path direct to the disk driver. The disk driver calls the XQP for control operations.

DFS provides a new VMS device driver package, just as one would for a new type of file-structured device such as a disk. This driver, called the DFS client device, is installed on the DFS client system. Its device name is DFSC.

The DFS client device forwards any requests it receives to its counterpart on the DFS server system, as shown in Figure 6. In the diagram, the XQP on the client and RMS on the server are shown with dashed-line borders. This indicates that these normal parts of a VMS process are not used in the DFS flow of control.

When an application issues a QIO to the DFS client device, DFS translates the QIO arguments into a DFS request and sends the message to the DFS server. The server then unpacks the arguments into a QIO again and issues the QIO on the server system. As on a single system, reads and writes go directly to the disk driver; all other QIOs go to the FILES-11 XQP in the DFS server process. When the XQP or disk driver responds to a request, DFS forwards the response back to the client system. There DFS translates the response into the QIO format and returns it to the application. Thus the application can operate exactly as it did when its file system was local.

Setting Up a Connection

Now let us examine the operation of DFS a little more closely. We start with a system manager who is

about to make a disk's file system available for access from other systems. The disk is mounted and available on a VMS system that has the DFS server installed and started.

The system manager tells the DFS server to add an access point. An access point is a directory on a disk volume and a global name for that directory. The server records the local and global names for the access point and then registers the global name with the VAX Distributed Name Service (DNS).[13] The registration also includes the network address of the node on which the DFS server is running. DNS makes the access point name available to all other nodes in the network.

A user on a client system who wishes to access the files in the access point mounts the access point on the client system. In response to the mount command, DFS creates a new device on the client system associated with that access point.

The DFS mount utility asks DNS for the network address associated with the access point. The utility then makes a DECnet connection to the DFS server on the indicated node. The DFS client passes the access point name to the DFS server, so the server knows which of possibly many access points the client is mounting.

Assuming no errors, the access point now appears as a VMS device called DFSC*n*, where *n* is a sequence number. This device appears to the system and to all applications on the system to be a normal file structured device, just like a disk driver. When an application issues a QIO to the DFSC device driver, the driver builds a request message and sends it to the DFS server associated with that DFSC device. DFS uses a custom communication layer to handle requests and responses. We describe this layer in the section DFS Communication.

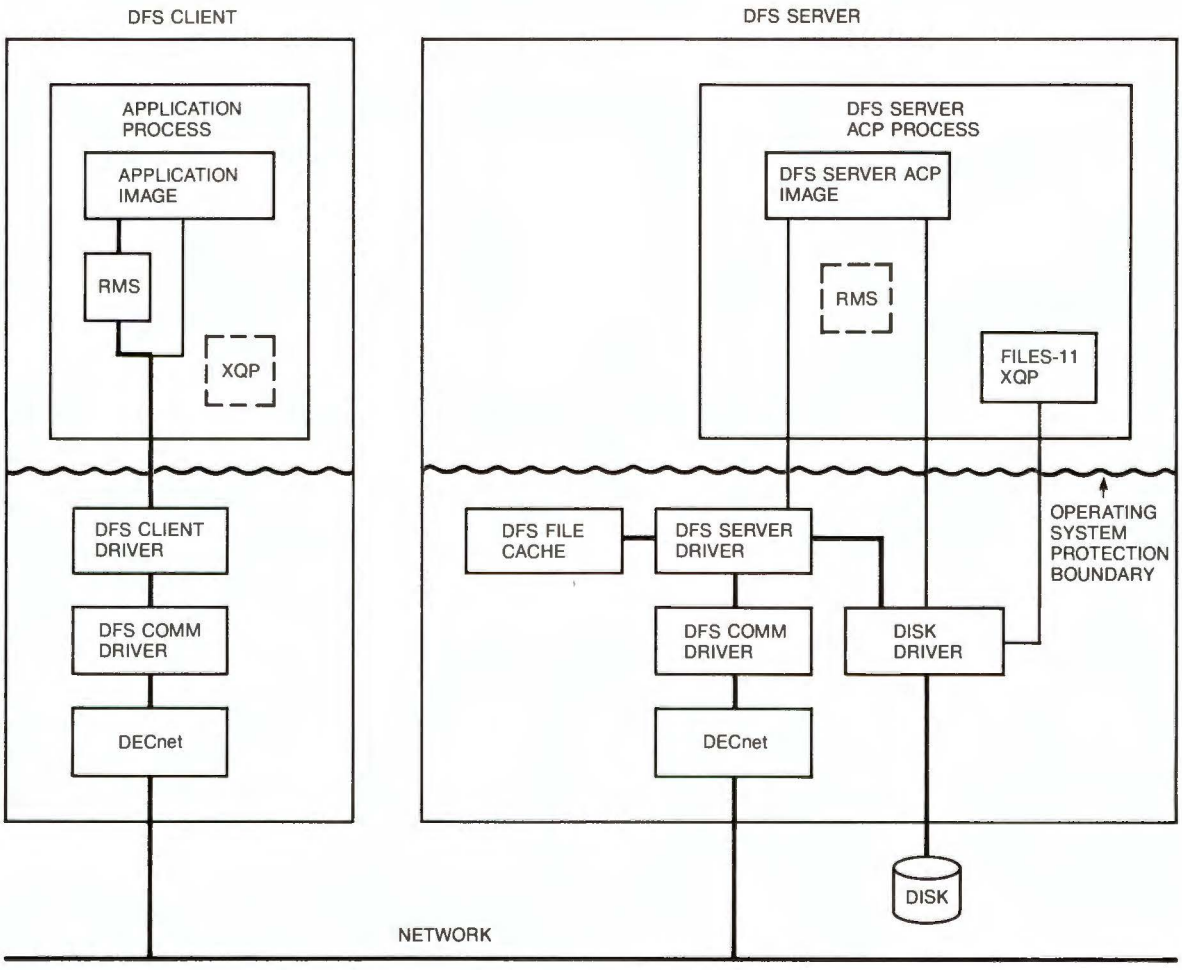


Figure 6 DFS File Access

Failures

Networks exacerbate existing failure modes in a file system and introduce new ones. It is quite possible for an operating system to lose communication with a disk, whether the disk is local or remote. If a local disk goes off-line, the VMS system leaves the files open and waits for the disk to come back on-line unless the operator tells it to stop waiting. This is safe, because users cannot modify the disk while it is off-line.

In a network system, however, it is possible to lose communication between a server and a subset of its clients. In this case, some clients can modify or even delete files during the time that other clients find the disk to be off-line. Therefore, the DFS behavior cannot be the same as the local behavior.

DFS takes a conservative approach to this problem. If the communication with the server fails for any reason, the DFS client driver reports a disk off-line error to every application that has any files open on that connection. The DFS client does not attempt to keep the files open, since they could be modified by other clients.

If an application tries to open a file after the connection has broken, the DFSC driver tries to connect to the server again. If the connection attempt fails or if the server does not respond in one minute, the driver reports a disk off-line error again. Eventually, one of these open attempts succeeds, and DFS carries on as usual.

If the DFS server detects that a client has become unreachable, it closes all files currently open for that client. This is necessary in order to match the client's behavior in this case.

The DFS Server

The DFS server is implemented in two parts: as a driver in the kernel and as a server process in the form of an ancillary control process (ACP). Incoming request messages go through the DECnet driver (NETDRIVER), then to the DFS communication driver, and finally to the DFS server driver, while still at fork (interrupt) level. (See Figure 6.) The DFS server driver handles read and write requests itself and forwards all control operations to the server process.

This design was influenced by our prototype and modeling work, which had shown that server performance was significantly improved by implementing data access operations in the kernel. We also found that implementing the control operations in the kernel offered insufficient benefit to justify the significant implementation investment required.

The DFS Server Driver The server driver receives all incoming requests. It sends the control operation requests to the server process and handles the data access requests itself. Our modeling showed that data caching increased performance significantly. Therefore, the DFS server driver implements a data cache.

If an incoming request is a read operation, the driver first checks its cache to see if the requested file blocks are available without accessing the disk. If so, the driver sends the data to the client immediately and dismisses the request. If not, the driver allocates the least recently used buffer from the cache to hold the data to be read from disk and forwards the request to the disk driver.

If the request is a write operation, the DFS driver checks to see if the requested file blocks are already in the cache. If so, it copies the new data from the request message into those cache blocks. If not, it allocates the least recently used cache block and copies the data there. The cache is write-through: the DFS driver always writes data to disk before responding to the client.

To make a disk access request from the kernel, the driver must translate the virtual block numbers from the request into the logical block numbers the disk driver expects. (Virtual blocks are addressed by offset into a file; logical blocks are addressed by offset into a disk. Both names refer to the same 512-byte blocks of data.) The DFS driver then builds a VMS internal I/O request packet and forwards it directly to the disk driver. When the disk driver calls the DFS driver back to signal I/O completion, the DFS driver builds a DFS response message and returns it to the client.

As long as a file is open, the cached data is valid because DFS does not allow write-sharing. Once the file is closed, however, other applications on the server system can modify the file and thus invalidate the cache. When the server process opens a file, it calls the DFS driver to check the revision number saved with the cache against the revision number stored in the file header on disk. If the two numbers are the same, then the cache is valid and can be used in the new session. If the numbers are different, then the disk copy has been modified and the driver marks the cached data invalid.

The DFS Server Process If the request from the client is for a control operation, the DFS server driver puts the request on a work queue and wakes the DFS server process.

There is only one DFS server process on any one computer. This process is multithreaded, so it can handle many clients at once. The alternatives to a single process are one server process per open file or one per client user. The use of a single process

reduces resource requirements. A single process uses less memory than several processes, and under load, the number of context switches can be reduced. Moreover, request latency is reduced because a single server process is always available and thus the client never has to wait for a process creation.

The single multithreaded server did not, however, lend itself completely to the transparency desired. The goal was to provide exactly the same security semantics as the VMS system provides to a local user. However, a single multithreaded server has difficulty representing multiple users with different identities. The server must run with privileges sufficient to allow it access to any file a user might legitimately access. Yet the server must somehow check protection on each file on behalf of the client user.

Exact VMS security semantics are difficult to guarantee if protection checks and file-opens are done separately. Therefore the DFS server impersonates a local user for each client user. When an application opens a file, the DFS client sends the user's name to the server. The server already knows the client node name as a result of the DECnet connection. The server uses the system proxy file to map this node::username pair into a local user name. The server then reads this local user's authorization data from the system authorization files and builds a "persona block" for the client user.

Before the DFS server process issues a QIO on behalf of a client, it copies the authorization data from the client user's persona block into its own process control block. Thus the server impersonates the user for protection checking purposes and uses the standard VMS protection checking mechanisms. Thus, the security semantics will be those that would have been applied had the local user run the application.

Our model showed that the time spent opening files had a noticeable effect on overall performance. Therefore, the DFS server caches these persona blocks. This caching reduces the number of times the server incurs the overhead associated with reading the three authorization files. DFS frequently purges persona blocks so that authorization changes will become effective in a timely manner.

DFS Communication

DFS is layered on a general request-response matching and session maintenance service called RR. This formal separation between RR and the other components of DFS makes the implementation easier and less prone to faults.

The resulting module is extensible and fully isolates the rest of DFS from its underlying transport. RR has a number of characteristics that make it particularly appropriate for DFS. We describe these characteristics in this section.

RR supports sessions to allow each request to be associated with a particular context. DFS uses a separate session for each open file. The sessions provide convenient handles for directing the requests and responses to the appropriate server thread and client process.

Each RR session can have multiple requests outstanding simultaneously. DFS uses this feature to support asynchronous I/O. Furthermore, the responses need not come in the same order that the requests are made. DFS uses these out-of-order responses to take full advantage of cache hits on read requests. Using the cache, the server can respond to any request as soon as it has the data. Of course, the DFS client reports the I/O completions to the application in the same order as the disk driver would.

The RR protocol is layered on DECnet. We found that the performance improvements to be gained by using a custom, LAN-only protocol were small. The gain was not sufficient to offset the advantages of using an established protocol. These advantages included the ability to scale the system by using WANs and to take advantage of the considerable experience our customers have in designing, installing and managing DECnet networks.

RR maps all RR sessions between a given pair of nodes onto a single underlying virtual circuit. RR's application interface gives no indication of the state or semantics of the underlying virtual circuits. Therefore the mapping from RR sessions to virtual circuits can be changed if that becomes desirable.

There are a number of trade-offs to be made when a single virtual circuit is used between each pair of nodes. On the positive side, a single circuit uses less memory and CPU time than does a circuit per open file. The single circuit also requires fewer messages to open a file, since the circuit itself only has to be opened for the first file. Using a single circuit also saves CPU time and message exchanges required to test for broken circuits. If a remote node becomes unavailable, we know that it will be unavailable for all files. There is no point in testing it anew for each open file.

There is also a negative side to multiplexing RR sessions on a single virtual circuit. The flow control in that single circuit is imposed unfairly. If one session sends too many messages for the network to handle, the virtual circuit and thus all sessions on it will be stopped until the congestion clears. This has not as yet proved to be a problem in practice, possibly because the request-response nature of DFS traffic has some implied flow control.

RR notifies its caller when the virtual circuit underlying a session fails (because of a failure in either the communication channel or the remote node). Again the session provides a convenient handle for directing this notification.

Finally, to conserve resources RR shuts down unused virtual circuits. If no sessions are open on a circuit (no files are open on it), then RR closes the circuit. Closing the circuit removes all related state information from the server side, but not from the client side. Thus the server is free to support more clients, and the client has enough information to reconnect the virtual circuit when an application opens another file.

Conclusion

DFS makes remote disks appear local to both users and applications. It supports this illusion by offering quick access to files, by using the same interfaces as the familiar local file system, and by being faithful to VMS file system semantics.

DFS can scale up to very large networks, because it is layered on DECnet and because only one client and one server are involved in any exchange. No other nodes need be advised or consulted, and thus network size is irrelevant to the DFS algorithms. Layering DFS on DECnet also gives flexibility, builds on customers' experience with DECnet, and allows easy integration into installed networks.

DFS is one of the initial DSS products that improves the way the VAX/VMS system is used in networks. DFS now works with DNS to make access point names global and will work with other products in the DSS family as they emerge. Together these products will simplify distributed access to network resources.

Acknowledgments

A large number of people contributed to DFS. The authors would especially like to acknowledge the significant contributions of Dennis Phillips, Danny Cobb, Julie Corenzwit, Pete Caswell, Carrie O'Connell, and Henk Tinkelenberg of the DFS development team. K.K. Ramakrishnan's modeling efforts provided invaluable guidance to the development effort. Finally, we would like to acknowledge the Eastern Research Lab research team, whose prototype was a model for the DFS design.

References

1. N. Kronenberg, H. Levy, and W. Strecker, "VAXclusters: A Closely-coupled Distributed System," *ACM Transactions on Computer Systems,* vol. 4 no. 2 (May 1986): 130–146.

2. *DNA Data Access Protocol (DAP) Functional Specification, Version 5.6.0* (Maynard: Digital Equipment Corporation, Order No. AA-K177A-TK, 1980).

3. R. Weber and W. Nichols, "Multiple Processor Coordinated File Access in VMS Systems," *Building Blocks for Storage Systems — Tutorial Notes for the Ninth IEEE Symposium on Mass Storage Systems* (October 1988): 209–218.

4. K. Ramakrishnan and J. Emer, "Performance Analysis of Mass Storage Service Alternatives for Distributed Systems," *IEEE Transactions on Software Engineering*, vol. 15 no. 2 (1989).

5. A. Birrell, "Secure Communication Using RPC," *ACM Transactions on Computer Systems,* vol. 3 no. 1 (February 1985): 1–14.

6. K. Ramakrishnan and J. Emer, "A Model of File Server Performance for a Heterogeneous Distributed System," *Proceedings of the ACM SIGCOMM '86 Symposium* (August 1986): 338–347.

7. R. Jain and R. Turner, "Workload Characterization Using Image Accounting," *Proceedings of the Computer Performance Evaluation Users Group 18th Meeting* (October 1982): 111–120.

8. J. Ousterhout, et al., "A Trace-driven Analysis of the UNIX 4.2 BSD File System," *Proceedings of the 10th ACM Symposium on Operating Systems Principles* (December 1985).

9. E. Lazowska, J. Zahorjan, D. Cheriton, and W. Zwaenepoel, "File Access Performance of Diskless Workstations," *ACM Transactions on Computer Systems,* vol. 4 no. 3 (August 1986).

10. F. Baskett, M. Chandy, R. Muntz, and F. Palacios, "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers," *Journal of the Association for Computing Machinery,* vol. 22 no. 2 (April 1975): 248–260.

11. S. Bruell and G. Balbo, "Computational Algorithms for Closed Queuing Networks," *Operating and Programming Systems Series,* (New York: Elsevier North-Holland, 1980).

12. M. Reiser, "A Queuing Network Analysis of Computer Communications Networks with Window Flow Control," *IEEE Transactions on Communications,* COM-27 no. 8 (August 1979): 1199–1209.

13. S. Martin, J. McCann, and D. Oran, "Development of the VAX Distributed Name Service," *Digital Technical Journal* (June 1989, this issue): 9–15.

David M. Griffin
Brad C. Johnson

Remote System Management in Network Environments

The introduction of the DECnet System Services product set opened new possibilities for the management of systems. One of the products in this set is the Remote System Manager (RSM) software, which allows managers and operations staff to support numbers of systems from a central point on the network. Based on a client-server model, RSM is designed to provide such useful functions as software distribution, file backup and restoration, and client administration. Each of these services uses common RSM facilities and tools that exist in the operating system environment — either VMS or ULTRIX.

The proliferation of computer systems, in particular workstations and personal computers, has encouraged the rapid growth of networks and has consequently magnified the importance of system management. The network itself presents an opportunity for managing these systems. The Remote System Manager (RSM) software, a member of the DECnet System Services (DSS) product set, uses this opportunity to provide centralized management of common system management activities.

This paper presents an overview of system management in a network environment and the effect of DSS in this environment. We then describe how RSM in particular addresses the problems and challenges presented by a network environment.

Distributed System Management

System management is generally defined as a set of tasks that a human must perform because the computer cannot do them. Many of these tasks involve resource management, for example, managing user directories in relation to the availability of disk drives, and information management, such as informing the system of a new user's identity and rights.

Networks create a need for a new perspective on system management. The system manager is now faced with managing dozens or even hundreds of systems. Many of these systems are small workstations, but all of them use the same software found on larger VAX computers, and all of them generate similar system management requirements. Further, workstations provide an opportunity to tailor the software environment to the job the user must perform. For example, a particular computer-aided-design software package can be tailored to the needs of a circuit board designer. Therefore the system manager's task is complicated by the need to manage multiple systems running different versions of applications, and in many cases, different operating systems.

System management of one system or a thousand is best approached as a combination of strategic resource planning and tactical operations.

The DECnet System Services product set allows resources that would normally be confined to a single system or VAXcluster system to be shared by multiple systems in a network. The management implications of shared systems are profound: networked resources can now be managed in a manner similar to managing a single system, thus reducing the overall system management load. The deployment of network resources should be planned accordingly.

- Expensive output devices, such as laser printers, plotters, etc., can be placed close to the users or in areas away from the secured main computer room. Distributed queues allow multiple systems to share a single printer or a central site to print reports on remote printers.

- Workstation mass storage is often limited because the cost per megabyte for workstation disks is greater than the costs of large, mass storage devices (such as Digital's SA482 storage array). VAX Distributed File Service (DFS) permits high-performance access to data residing on secure, cost-effective media and allows the media to be managed as a single disk volume.[1]

- Data that would normally have to be replicated between computer systems can now be shared using DFS. Data sharing reduces mass storage demands across the distributed system, and the

management of mass storage can be concentrated to a smaller number of systems.

- Local data can be backed up over the network with RSM to systems with high-speed tape drives or large-capacity disk drives. This procedure can eliminate the need for local backup devices, such as cartridge tape drives, and can reduce the hardware costs for backup across the distributed system.
- Network installation of operating systems and applications can eliminate the need for local load devices, such as floppy disks or cartridge tape drives. Further, these installations eliminate the need for local management expertise.

When considering how to manage a number of workstations in a local area network (LAN), one should consider first the option of a local area VAXcluster (LAVc) system. LAVc provides a tightly integrated management environment. Users of the cluster share operating system software, applications, file systems, and single security domain.[2] Management of a VAXcluster system is essentially the same as managing a single timesharing system, which makes it very attractive in a number of environments.

The strengths of the local area VAXcluster can also be its weaknesses. In a shared software environment, all workstations in a cluster must run the same versions and upgrade at the same time. In a common security environment, a privileged user on one workstation is privileged on all of them.

LAVc is also bound to the LAN topology, although extended LANs allow LAVc to operate beyond the default Ethernet topology.[3] When organizations span a larger geographic area, however, a wide area network (WAN) may be necessary. The VAXcluster communication protocols do not operate in a WAN topology. The DSS products are optimized to run in a LAN, and because they are layered on DECnet software, they all have wide area capabilities as well. Table 1 compares the DSS and LAVc environments.

Table 1 Comparison of DSS and LAVc Environments

DECnet System Services	Local Area VAXcluster
System software type and version can vary from system to system	System software type and version is common to all users
Multiple security and privilege domain	Single security and privilege domains
Available on local and wide area networks	Available on local area networks only

Definition of Remote System Management Tasks

When the RSM engineering team examined system usage in networks, it found that organizations were usually composed of distinct groups of users who share a common environment. For example, all members of a group would use the same version of an application or utility. These computer users generally did not want to perform system management tasks. As a result many tasks were done irregularly or not at all, including data backup tasks. Because data is an asset that must be protected, many system management groups were made responsible for maintaining the workstations. To meet this responsibility, system managers often found themselves literally running from one system to the next, installing software from floppy disks or backing up files to tape cartridges.

We also examined the variety of tasks a system manager performs and the time it takes to perform them. These tasks include

- Software installation of operating systems and applications (for example, Digital layered products)
- File backup to tape or disk
- Administrative functions, including user account maintenance, quota modification, and security access control and privilege maintenance
- System tuning and general maintenance (for example, queues maintenance)
- System monitoring

We decided to provide a product that supported the tasks of software installation and file system backup because they consumed a large portion of the system manager's time. In addition, the tasks could be structured so that differences between systems could be handled by the RSM software and hidden from the system manager.

Significant Design Decisions and RSM Overview

Like the other DSS products — DFS, Distributed Name Service (DNS), and Distributed Queue Service (DQS) — RSM is a new product. No other Digital layered product can be considered a predecessor. Furthermore, few products in the market directly compete with the RSM functionality. Therefore a number of fundamental product decisions had to be made, without the benefit of comparison, to design a usable and salable application.

The following are the RSM goals that reflect these decisions:

- The overall goal was to provide useful functions that would allow for the efficient management of systems in a distributed environment; each type of function would be provided by a separate service.
- The services of RSM would provide centralized control.
- The services of RSM would use a server-to-client model. (The terms client and server with reference to RSM are used in a somewhat unique way. In RSM a client is a system that receives management services; a server provides management services.)
- All RSM services would use tools that exist in the operating system environment in which they execute. (We would not reinvent or redevelop these tools.)
- Each service would use common RSM facilities.
- Each service would develop or use unique functionality that would help that service be more useful.
- Each service would provide client support in both the VAX/VMS and ULTRIX environments, as applicable.

Before describing the design goals in more detail, we should note that a fundamental goal was common to all of the DSS products. Because these were introductory products, we chose not to perfect the first release of the product. Rather, our goal was to bring each product to market in a timely fashion so we could learn more about the market and customer requirements from firsthand experience.

As stated previously, the overall goal of RSM was to provide management functions that are useful in a distributed environment. A useful function is characterized as one that is commonly used in managing a single system and that can be applied to many systems within a network. Currently, RSM is composed of four loosely coupled functions or services:

- Basic operation services, BOS
- Software distribution services, SDS
- Backup and restore services, BRS
- Client administrative services, CAS

In line with our design decisions, these services use common RSM facilities (BOS). Each provides separate functionality, and each has unique characteristics that are important to each service. The SDS, BOS, and BRS services are described in the section Components of the RSM Software.

RSM provides centralized management in a server-to-client model; that is, there is one server, called the RSM management server, for some number of RSM clients. The management server controls the actions on the clients. In fact, there are three types of RSM servers:

- Management server
- Library server
- Target server

A management server provides the RSM user interface and the mechanism for connecting to the available RSM services. A management server also contains a library and a target server. The library server is specific to SDS and is used for maintaining the software which SDS can distribute. The target server is specific to BRS and is used for maintaining backup save-sets. The library and target servers are described in the sections Software Distribution Services, and Backup and Restore Services, respectively.

Any system on the network can be an RSM client, including the management server. The only restrictions are that the client must be able to communicate to the server using DECnet, and the client must run either VAX/VMS or ULTRIX software. In the RSM model, client systems are passive; that is, they do not initiate any actions without explicit direction from an RSM management server.

Another characteristic of the RSM model is that each client system being serviced bears most of the required processing overhead. This reduces the risk of the management server becoming a CPU bottleneck and increases the ability of the server to service more client systems. That is, distributed processing in a network context provides true parallelism — multiple, autonomous systems work to complete tasks simultaneously. The RSM model attempts to capitalize on this aspect of the network by letting the client that is being serviced do most of the work.

To support this server-to-client model, a command execution agent (CEA) module exists on every RSM client. This module is part of the RSM client software. The purpose of this module is to accept management directives from one of the services and execute them on the client system. The management server is needed to initiate the operation, possibly supply some additional information (such as an application name for an SDS installation), and to update databases and log files once the task has completed.

Another goal was to reduce development time by taking advantage of stable technologies. Designers chose to use existing facilities or tools whenever possible for RSM services. Although this strategy places limitations on the services, we considered the strategy

worthwhile to meet our time-to-market goals. The use of existing facilities allowed us more time to address other basic distributed application issues such as logging, recovery, synchronization, and data access. Furthermore, this removed the need for the RSM manager to learn new tools — the tools that were currently used to manage a system locally would be used to manage systems remotely.

Specifically, RSM uses the following facilities and tools:

- VMS batch queues – for scheduling asynchronous RSM operations on the management server
- VMS BACKUP – for both software distribution and file system backup/restore when RSM interacts with VMS clients
- The VMSINSTAL software installation utility – to install applications on VMS clients
- The VAXELN operating system – to provide a memory-resident environment for installing the VMS operating system over a network connection
- VAX DNS – to store network-wide information about RSM clients and RSM servers
- The ULTRIX tar, dump, and restore utilities – for file system backup/restore for ULTRIX clients
- The ULTRIX setld utility – for ULTRIX application installation

One aspect of RSM that designers found difficult to quantify is the size of its management domain, that is, the number of clients a single RSM server can effectively service. Because of the dynamic characteristics of the operations that RSM performs, determination of this client-to-server ratio depends on the customer's particular environment. RSM operations, such as software installation and backup and restore, often involve the transfer of large amounts of data over the network. Thus the amount of contention for network bandwidth in the customer's environment is the primary factor in determining how many systems can be supported by an RSM server. RSM version 2.0 recommends a client-to-server ratio of 45 to 1. However, experience has shown that some environments can operate at a 150 to 1 ratio, whereas other environments may require a 10 to 1 ratio.

Components of the RSM Software

Basic Operational Services

To the RSM user, BOS provides a means of maintaining information about clients and RSM servers. Internally, BOS provides the basic user-interface services: the command-line interface, the menu interface, indirect command file execution, and miscellaneous routines shared by the other components.

RSM takes advantage of the namespace provided by DNS to store information about RSM clients and servers.[4] DNS also provides RSM with a powerful abstraction called groups, which makes the management of large numbers of systems more tractable. Groups in DNS and RSM are collections of object names. In RSM, groups can contain names of RSM clients, RSM servers, and other groups, thus forming a basis for hierarchical group naming schemes. In almost all cases in which a client or server name can be specified in RSM, a group name can be substituted. RSM expands the group to the unique set of names in the group (permitting group memberships to overlap) and filters out objects that would not make sense to the command. (For example, server names are omitted when the context implies client names.) RSM uses the group facility as a shorthand, or macroexpansion, facility; separate commands are internally generated from the user's external command so that the operation for each member of the group can fail or succeed on its own merit.

Software Distribution Services

SDS provides system managers the functionality to install applications and operating systems, from a centralized server, onto another system in the network. SDS supports two of our earlier mentioned goals in that it performs a primary system management function, and it also scales well in a network environment. Installation of an application can consume one to two hours of a system manager's time. An operating system installation can take as long as eight hours. As discussed below, the RSM manager must invest a comparable amount of time to install the initial copy of the software in the SDS library server, but all subsequent installations require only a fraction of that time.

The server is responsible for initiating and tracking requests, and the clients are the recipients of these requests. SDS can be further broken down into three components: the management server, the library server, and the client. The management server provides the SDS user interface and is the storage place for SDS files (such as the SDS databases, SDS executables, and log files). The second component, the library server, is a file hierarchy for storing copies of software for subsequent network installations. Before software can be installed using SDS, a copy of the software must exist in the SDS library. The library server is a passive system and is merely a repository for software. All processes that control the copying of files in or out of the library server are initiated on the management server. By default, the

management server is also a library server. However, the RSM manager has the option of installing the RSM server software on other VAX/VMS systems to configure additional SDS library servers. This option allows the RSM manager to determine which system is best suited for the centralized initiation of requests and which system is best suited for storing copies of software. Finally, the third component of SDS is the client. To supply management services to another system on the network (an RSM client), the RSM client software must be installed on that client system. To do this, the system manager can use SDS itself. Once the RSM server kit has been completely installed, the RSM manager can use the SDS component to initiate the installation of the RSM client kit on the appropriate client systems.

SDS Concepts One of the primary functions of SDS is to duplicate the installation of software on many systems in the network, exactly as the installation was performed on the original (source) system.

To take advantage of cloning, a master copy of some software must exist in the SDS library. First, the RSM system manager must create a copy of the software. For an operating system, this means using the standard VMS or ULTRIX installation procedures to create a VMS or ULTRIX system. For a layered product, this means using an RSM procedure called TRIAL_INSTALL, which will invoke either VMSINSTAL or setld for VMS and ULTRIX systems respectively. Then this copy must be copied into a SDS library server. Once there, it is available to be cloned onto other RSM client systems.

Creating this copy takes as long as a normal installation of the application, mainly because the RSM manager is actually installing the application on the designated system. However, once the application has been copied into an SDS library server, subsequent installations only require the RSM manager to initiate the appropriate SDS installation request on the management server. Therefore, although the RSM manager is required to install the software once before copying it into an SDS library, all subsequent installations only require the manager to initiate the task. SDS assumes the responsibility of installing the software on one or more clients and reporting the status back to the manager.

SDS expands on this basic capability by employing groups and application sets. The support for groups is provided by BOS, and the support for application sets is provided by SDS. Similar to a group, application set members are either other application sets or a single application name. By combining both groups and application sets, the RSM manager can enter a single command that represents the installation of many applications on many clients. Figure 1 shows an example of how these set mechanisms are used. One SDS command initiates an installation of all of the applications that are members of the application set Languages to all of the RSM client systems that are members of the group Area1.

SDS Process Flow All SDS commands can be classified as either synchronous or asynchronous. The synchronous commands are specific actions on the SDS databases (such as create, modify, and view), whereas the asynchronous commands allow the RSM manager to copy software into an SDS library or initiate installation requests onto RSM client systems. To explain the process flow of a typical SDS request, we present an overview of both synchronous and asynchronous requests in this section.

All synchronous requests are handled directly on the management server by the SDS user interface. The user interface performs syntax checking, supplies any appropriate default values and token interpretations, and then executes the specified request.

All (successful) asynchronous requests go through the same data path as synchronous requests. Additionally, after an asynchronous request is processed by the user interface, a client-system-specific work

```
DISTRIBUTE> Install Application Languages Area1

The application set Languages          The group Area1 contains:
contains:

C                                                  Node_1
FORTRAN                                            Node_2
.                                                  .
.                                                  .
.                                                  .
PASCAL                                             Node_n
```

Figure 1 Using Set Mechanisms

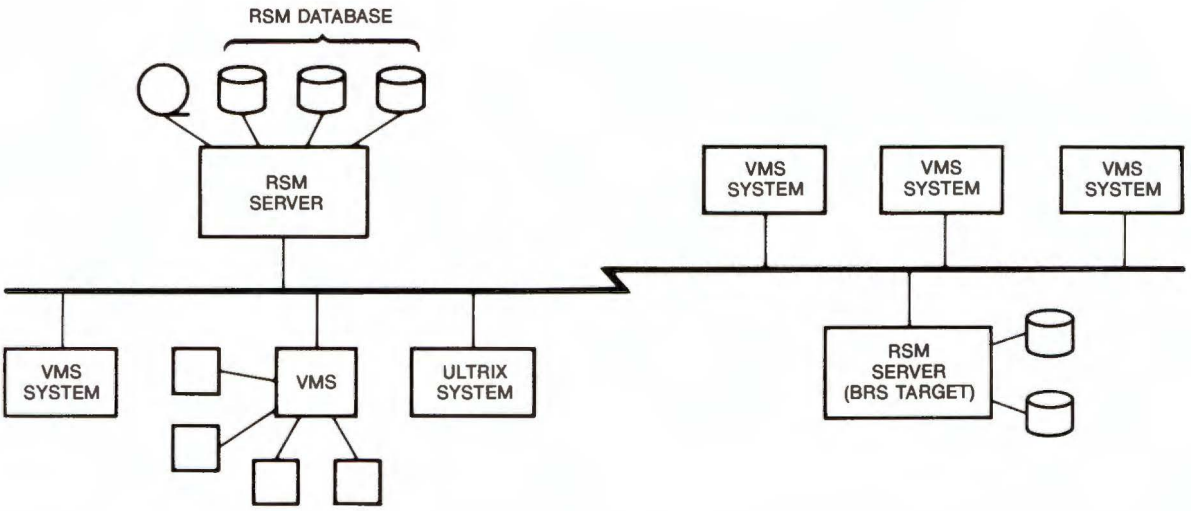


Figure 2 RSM Configured in a Wide Area Network

file is created. If there is no background job currently working on other requests for the specified RSM client system, then a VMS batch job is created to handle the work file. Otherwise the work file is updated and the currently running job handles this request. It is the responsibility of this batch job to execute the asynchronous request. If the request is to install an operating system, the job uses the DECnet Ethernet boot facility to initiate the installation on the RSM client system. For all other asynchronous requests, the job contacts the RSM client software and passes down the appropriate control information; the client system then executes all appropriate actions. In all instances, once the request has been completed, it contacts the server to report its status along with status information.

Backup and Restore Services

Ensuring the integrity and recoverability of data is a prime responsibility of system managers. Data created by computer system users is an asset for a company and should be protected as such. Since data is often modified on a daily basis, protection of that data requires that the data be continually duplicated and stored outside the system. In the past this process has typically been done by backing up disks to local tape drives or floppy disk drives.

BRS provides the ability to backup and restore files (file systems) over a DECnet network for both VMS and ULTRIX client systems. The BRS software uses the backup utility on VMS clients and either the dump or tar utilities (depending on the type of backup being performed) for ULTRIX clients. A scheduling and tracking system located on the RSM management server supports the actual process of backing up the files.

The design for the BRS subsystem is shown in Figure 2. BRS has six components: the user interface, the database server, the backup scheduler, the backup initiator, the restore initiator, and the command execution agents (CEA).

The user interface is activated by the BOS subsystem and parses DCL commands issued by the user. Commands are available to add, modify, and delete schedule entries; display schedule entries, history records, and scheduler status; and initiate ad hoc backup or restore operations.

The user interface communicates with the database server which manages two RMS-indexed sequential files: the schedule file and the history file.

The schedule file contains basic information — the date a backup should occur and what data is to be backed up. Entries in the schedule database are organized by client name and a user-specified "backup type." This organization allows the manager to name and schedule different backups for any one client. (For example, one backup entry could be constructed to back up critical project files located on a system on a daily basis; whereas another entry would back up noncritical files over weekends.)

The history file contains an entry for each backup operation attempted by BRS. Within the history record are recorded the status of the backup and pointers

to the appropriate log, listing, and save-sets (files where backup data is actually stored). History records are used by RSM when performing a restore operation. By specifying a particular history entry, RSM knows from where to retrieve the save-set and where to restore it.

The BRS scheduler is a process that resides on an RSM management server. The scheduler periodically scans the schedule file and determines if any client system needs to be backed up. When the scheduler determines that a backup should be scheduled, it creates a history record for the backup attempt. It then submits a batch job on the target server which will actually control the backup process.

A batch job on the target server runs the backup initiator program. The backup initiator secures resources for the backup save-sets, which can reside on a local disk or be directed to magnetic tape. It then initiates a connection to the RSM CEA on the client system. The initiator instructs the CEA to run the appropriate backup utility and supplies it with the list of files/directories to be backed up. The save-set data and logging information generated by the backup utility are passed to the CEA which multiplexes the information over the DECnet connection to the backup initiator process on the target server. The initiator, in turn, unpackages the data and writes it to the appropriate files or devices. BRS uses checksums to assure data integrity over the network connection.

Once the backup is completed, the associated history record is updated to reflect a successful (or failed) backup. If specified in the schedule record, mail is sent to the appropriate parties indicating the status of the backup.

A similar process is used to restore files. However, by using information gleaned from the user-specified history record, the user interface bypasses the scheduler process and submits a batch job for the restore initiator on the target server. The restore initiator then obtains access to the save-set data by directly accessing the files or requesting that a particular tape be mounted. The initiator then contacts the appropriate client system's CEA and constructs a restore command which is executed on the client system.

Implementation Techniques

RSM was written primarily in the VAX C language. The development team used DEC/CMS (Code Management System) and DEC/MMS (Module Management System) software as the configuration management tools. The CMS libraries supporting the project were stored on a large VAXcluster system shared by other DSS development projects.

Project engineers performed a large portion of the development process (editing, compiling, and unit testing) on their individual workstations. DFS was used to provide access to the CMS libraries and other development resources on the workstations. When a developer was satisfied with a particular module, it was checked into the CMS library.

A regularly scheduled batch job on the VAXcluster system ran each evening and rebuilt the RSM software and produced an installation kit. Logs of various parts of the build procedure were mailed to the appropriate developers. If a kit was successfully created, the build procedure would automatically hand the kit over to the automated test suite which would install the kit on the RSM test server. A series of tests was then coordinated using the DTM (VAX DEC/Test Manager) product. The tests were performed on the kit, and the results were ready for review when the developers arrived the next morning.

The test environment for RSM was somewhat unique. RSM supports several releases of two operating systems. In the SDS initial system load feature, RSM has a number of hardware dependencies that require "matrix testing." In matrix testing the same tests are performed on different combinations of hardware and operating system software to assure that changes or new development has not introduced problems not covered by the unit testing. Once again, the DFS product was pressed into service, allowing the test suite to be shared between different servers.

Many RSM operations can take a significant amount of time to complete. For example, installing an operating system can take 30 to 40 minutes. To allow the nightly test suite to check many of the database aspects of the product in a relatively short amount of time, run-time switches are employed to avoid the time-consuming parts of the operations.

Summary

One of the most important aspects of a growing network is system management, which includes management of the network itself and the resources available on the network. The DECnet System Services product set allows these resources to be effectively shared and controlled by using the network. Complementing the DFS and the DQS products in a distributed environment, RSM provides centralized management of common system management activities.

In designing and developing RSM, we learned a great deal about providing services in a distributed environment and the numerous ways in which networks and systems can be managed. We believe we

have designed management services that are useful in a growing and heterogeneous network environment. The availability and flexibility of these services allow the customer to determine the most effective way to use a particular service.

Networks are growing in size as well as in the variety of systems they support. The ability for distributed products to operate effectively in a network environment comprising hundreds or thousands of systems will be a requirement instead of a desirable feature. Moreover, network products will have to provide services for many types of systems as more organizations adopt multivendor hardware solutions. These and other factors serve as guides to the process of building future distributed system management products.

Acknowledgments

The authors would like to acknowledge the engineers and writers who have contributed to the RSM projects: Barbara Benton, Diana Bergman, Terri Buckley, Jon Campbell, Alex Chen, Ron Jansen, Jim Lemmon, Mary Ellen Lewandowski, Bill Parke, Mike Reilly, Rich Rosenbaum, Dexter Sealy, Jim Shelhamer, Owen Tallman, Peter Wiitanen, and Roger Zee.

A special note of thanks to the internal MIS community and Digital's EasyNet users who pioneered many of the techniques and tools used within RSM and who provided extensive feedback and testing of the product.

References

1. W. Nichols and J. Emer, "Design and Implementation of the VAX Distributed File Service," *Digital Technical Journal* (June 1989, this issue): 16–28.
2. M. Fox and J. Ywoskus, "Local Area VAXcluster Systems," *Digital Technical Journal* (September 1987): 56–68.
3. W. Hawe, M. Kempf, and A. Kirby, "The Extended Local Area Network Architecture and LANBridge 100," *Digital Technical Journal* (September 1986): 54–72.
4. S. Martin, J. McCann, and D. Oran, "Development of the VAX Distributed Name Service," *Digital Technical Journal* (June 1989, this issue): 9–15.

Peter O. Mierswa

The Evolution of the MAILbus

The MAILbus product family provides facilities for creating, transmitting, receiving, and managing messages in a multiple vendor network environment. Messages can consist of combinations of text, data, and arbitrary files and can be exchanged among people or applications in various computing environments, including the DECnet network, X.400-conformant messaging systems, IBM Professional Office Systems, and IBM Systems Network Architecture Distribution Services systems. This paper presents the development history of the MAILbus product family and the design decisions that have made possible the provision of the MAILbus communication services in a changing heterogeneous network environment.

The MAILbus family includes a number of products that provide both basic message handling and directory services. Together these services offer a single distributed electronic mail and messaging application across a broad range of hardware and software systems.

With an electronic mail presentation application, such as ALL-IN-1 software, the MAILbus product family can act as a messaging backbone that allows the Digital end user to exchange messages and documents with users in other environments. Further, MAILbus products support the exchange of messages and documents between the non-MAILbus environments across the MAILbus backbone.

The basic message handling and directory services are provided by the VAX Message Router product. Included in the Message Router are a DECnet network-wide, store-and-forward message transfer service and a descriptive directory service. The message transfer service is an application-independent mechanism that reliably relays messages from an application on one computer system in a network to another, without requiring a direct end-to-end connection during the relay. This transfer is accomplished by recording the message in reliable storage at points along its path. This service supports both electronic mail and nonmail applications. A programming interface is also available that enables customers to write Message Router applications. The descriptive directory service provides access to directory entries, such as those that describe computer system users, even when given possibly incomplete descriptive information. The information contained in such entries includes the computer system users' names, organizations, locations, and electronic mail addresses.

As shown in Figure 1, the Message Router gateways offer access to numerous messaging applications. The VAX Message Router VMSmail Gateway provides Message Router applications with access to the VMS Mail utility.

The VAX Message Router X.400 Gateway provides access from Message Router applications to applications running on computers that conform to the Consultative Committee on International Telephony and Telegraphy (CCITT) X.400 Recommendations for message handling systems.

The VAX Message Router/S Gateway provides access from Message Router applications to IBM Systems Network Architecture (SNA) Distribution Services (SNADS). The VAX Message Router/P Gateway provides access from Message Router applications to the IBM Professional Office System (PROFS) within an SNA environment.

The VAX Message Router Telex Gateway provides access from Message Router applications to public Telex networks.

The ULTRIX Mail Connection provides access from Message Router applications to all ULTRIX mail domains.

The MAILbus product is a single message-handling system which supports a feature-rich message exchange for multiple applications across a large number of different hardware and software systems in very large networks. This ability is the result of a number of design decisions including

- The separation of the system into message-transfer and user-agent layers
- The encoding of messages in a standard but extensible manner
- An exposed interface between the user-agent layer and the transfer layer
- Comprehensive management services
- A network-wide directory service

Some of these characteristics were selected as a direct result of national and international standards development, and some are the result of Digital design and product work. Much of Digital's messaging system product development work has been influenced by the company's experience with messaging systems used in its own business environment.

Early Digital Messaging Projects and Products

A number of years before the early standards bodies published their first reports, the corporate operations groups that supported Digital's senior managers and administrators began to consider the problem of mail and messaging systems. The group wanted to improve communications between senior managers.

In 1974 a centralized, single-system message switch was implemented at Digital's headquarters in Maynard, Massachusetts. Originally called DECnet, the system was renamed the Corporate Message Switch (CMS). Initially messages could be delivered between 81 hardcopy terminals in New England; Chicago, Illinois; and Kanata, Canada. Over time, the system grew to support over 300 terminals in the U.S., Canada, and Europe.

In 1979 the service changed to a new, redesigned switch called the Record Communications System (RCS). RCS grew to support multiple switches and over 500 terminals in the U.S., Canada, Europe, and the Far East. Daily traffic reached a peak in 1983.

In January 1978 the Digital Telecommunications Group installed a pilot, single-system, personal mailbox electronic mail system (EMS). The system ran on a PDP-11/70 system using an application developed by the CCA (Computer Corporation of America), called COMET. The COMET application was very different from the services provided by CMS and RCS. The RCS service provided only paper copies of messages sent. The COMET application gave users the capability to compose, edit, send, read, forward, answer, file, retrieve, and delete messages. The pilot began with 40 users and grew to support 650 within a year. The pilot program was concluded with complete support from Digital's senior management committee to proceed to develop a production system.

The most serious problem facing the Digital internal production system that supplied electronic mail services was the anticipated growth in user accounts. It was recognized that the COMET system could not meet this need as it was. Proposals from CCA were considered for changes in the system. However, at the same time as the EMS pilot system was being introduced, another group within Digital, the Laboratory Data Products Group, had developed and put into production an electronic mail system for its own use. After analysis of the two projects, it was concluded that the internally developed product could be expanded to become the corporate production system. In 1979 a multinode EMS system, built from the LDP project, was introduced. By 1981 the system was supporting 10,000 subscribers on 15 nodes in the U.S. and Europe.

The CMS, RCS, and EMS systems were developed and implemented to support the management and

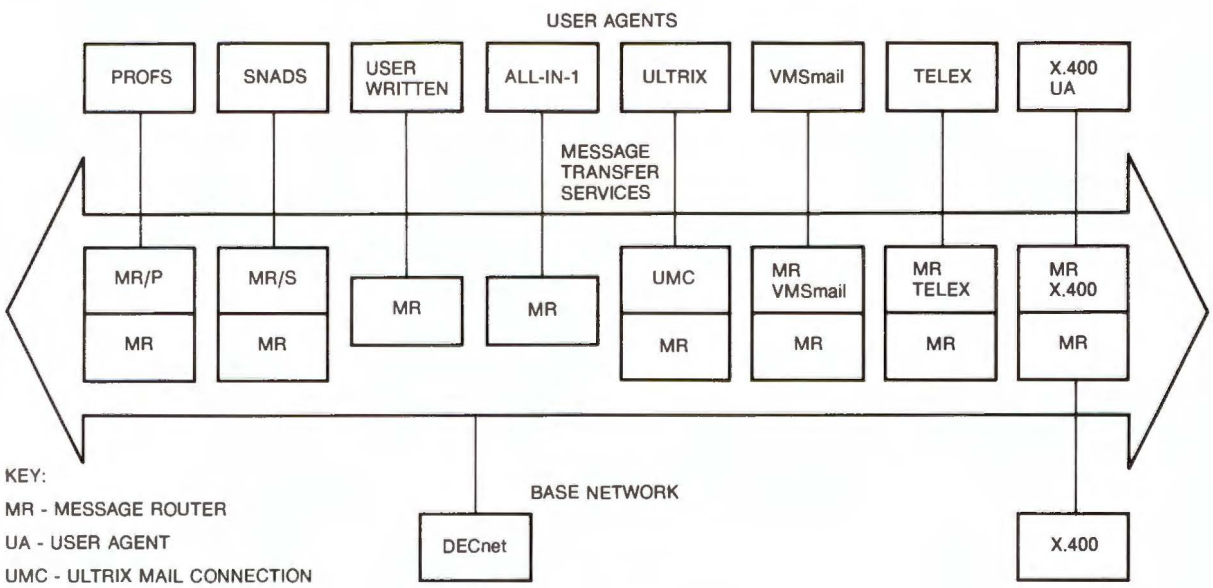


Figure 1 Overview of MAILbus Connections

administration of the corporation. As such, strict requirements governed reliability and performance, and the services were generally restricted to a small number of senior managers within the corporation. However, all employees had a need for fast, effective communication. In the late 1970s, the Digital internal computer network, based on DECnet software, was informally growing to include more and more engineering facilities. Electronic mail systems, which were based on the VMS Mail protocol and provided end-to-end text message delivery, were being introduced by engineers to meet their own communication needs. Although these mail systems and the network on which they ran were managed in an ad hoc manner, they provided a vital service. By 1982 these systems were supporting 20,000 users on more than 1,000 systems. In 1981 the EMS system and the VMS Mail systems were linked with a gateway.

By 1981 it had become painfully clear that there were serious growth problems in Digital's internal electronic mail systems. As more and more managers saw the benefits of electronic mail, the demand for EMS accounts increased. With more users participating, each user spent more time and more resources using the systems. The message traffic, as a result, was increasing rapidly. In addition to message traffic, the EMS system maintained a central directory node which updated each EMS node whenever details changed in subscriber or distribution lists. With the growth in EMS usage, this directory traffic was contributing substantially to the network traffic problem. With the introduction of the gateway between the VMS Mail systems and the EMS and RCS systems, requests for accounts and traffic growth increased unpredictably.

The problem was now a complex one. Different internal organizations had varying requirements and varying partial solutions, yet all had legitimate business reasons for wanting to communicate with each other. Traffic and use were growing enormously, and new requirements were recognized for electronic mail services, such as transfer of compound documents. A solution was needed.

Development of the Message Handling System Model

During the 1970s a product development group within Digital had been following the computer industry's standards development for handling and transmitting electronic messages. Two standards bodies were of particular interest: the International Federation of Information Processing (IFIP) and the U.S. National Bureau of Standards (NBS).

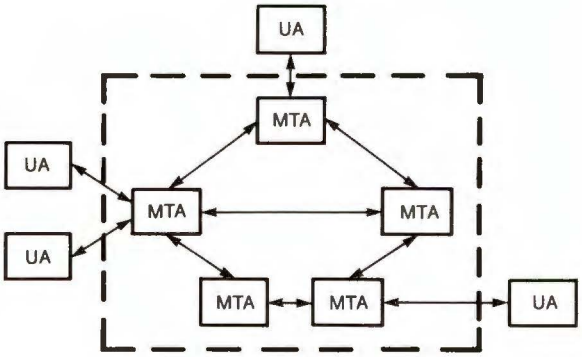


Figure 2 Message Handling System Model

The IFIP working group 6.5 produced a model for a computer-based mail environment. The model described a computer-based mail environment as one implemented using two logically distinct subsystems: message transfer agents (MTAs) and user agents (UAs). (See Figure 2.) Message transfer agents would cooperate to accept, transfer, and deliver messages on behalf of the user agents. All messages would have a well-defined format. User agents would interact with the end user to create and submit new messages and accept and display received messages.

At about the same time, the NBS was working to produce a replacement for RFC 733, the then current standard for text messages on the ARPA network. A draft report was produced in 1980 and the final report published as Federal Information Processing Standard (FIPS) 98, "Message Format for Computer-Based Message Systems," in March 1983. This report specified that messages consist of two parts, the envelope and the content. These terms are taken directly from the postal system. As in the postal system, the envelope bears only the information required to deliver the message to its intended recipient. The content of the message is the information being conveyed to the recipient. (See Figure 3.) In addition to specifying the split of messages into two parts, the NBS report specified an encoding technique for the message. In the type-length-value (TLV) encoding specified, each data item is preceded by a code that specifies its type and a field that gives its length. Data items can contain simple data of a number of different data types or data items constructed from more than one simple item. Thus users can construct complex messages containing multiple documents which include different types of data or even other messages.

Together, these early standards specified a model in which the MTAs could be built without any

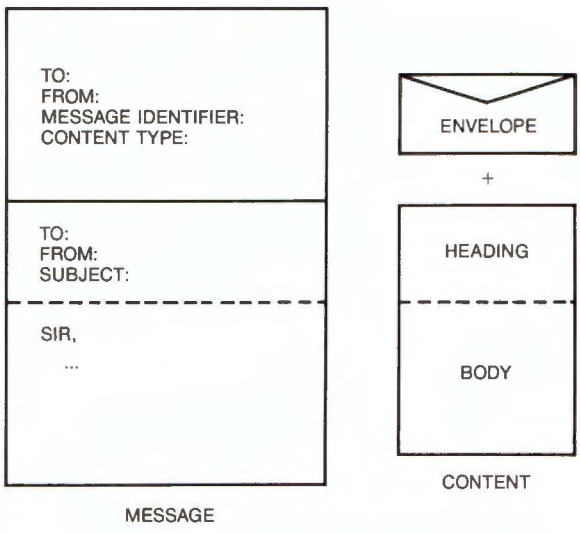


Figure 3 Basic Message Structure

knowledge or regard for the meaning or format of message contents. Since the content of a message was encoded with TLV encoding, the MTA had only to ensure that it passed on the content exactly as it received it. All the information required to deliver the message is contained in the envelope. It was now possible to build a single, message-transfer system to support any number of different applications. In addition, the use of a TLV encoding for the message made it possible to extend over time the MTA and UA services without disrupting service by including new coded items to support new services. If an old product were to encounter a new coded item, the product could use the item's length to skip easily to the next item in the message.

The First Message Router Products

In 1982 Digital recognized the need to conform to international standards and, along with a number of other vendors, committed to support NBS-formatted messages. Later in the year, the development group that had been following the work of the various standards bodies completed and shipped the first standard-conformant messaging system product — DECmail with the multinode option. This product provided the DECmail user agent with a message transfer system that conformed to the NBS message format. This first release would later be called the Message Router.

The release of this product coincided with the realization by the Digital internal operations group that the existing EMS, RCS, and VMS Mail systems were not solving the corporation's communication problems. After careful study it became clear that a product set based upon the emerging industry standards would be the ideal solution to the problems of Digital's internal users.

In 1982 the Digital Telecommunications Group began a pilot project to implement a corporate-wide, message handling system based upon the DECmail multinode option software. In 1983 the VAX Message Router was shipped as a separate product, and a VMSmail Gateway was added. At this time, it was clear that the pilot project was successful and full implementation within Digital was begun. Soon, full office automation services were added to the electronic mail service with the release of ALL-IN-1 version 2. This product was the beginning of a common approach within Digital between product engineering and internal services for the delivery of electronic messaging services.

Message Router Transfer Service

The Message Router Transfer Service provides the message transfer agent function of the message handling systems model. Its function is to transfer a message from an originating user agent to a recipient user agent. The Message Router provides a store-and-forward delivery system in which the message is written to reliable storage devices at each step along its path before the previous system discards its copy of the message. This technique provides both a high degree of reliability and is not significantly affected by temporary failures of any system or communications path.

To support both different mail user agents for different environments and message transfer services for nonmail applications, the message format chosen was the NBS format. In this format the message is split into two parts — the envelope and the content. In a manner similar to a postal service, the Message Router uses the information on the envelope to deliver the message but does not look at the content. In this way, the Message Router can act as the store-and-forward transfer service for any application.

The envelope contains data items that support a number of services. The NBS encoding used is the TLV form noted above. Using this encoding scheme, an application can include only those data items that suit its needs. Further, the scheme easily supports the addition of new services over time, since a data item can easily be skipped by a message handler if its type is not known.

Both the definition of the basic services and message format were specified by the early standards. However, the early standards did not address either the mechanism for addressing and delivering messages, or the communications method by which MTAs exchange messages.

DECnet task-to-task communication was chosen as the transport mechanism for MTA-to-MTA message transfer. This choice would clearly provide interconnection over a wide variety of communications media and would require only a single service interface.

The choice of an addressing and delivery mechanism was a much more difficult problem. The address must be meaningful to the end users, and also contain information sufficient to route the message from source to destination in both small and large networks. Moreover, the address must be a means for messages to be exchanged with other electronic mail systems that use their own various addressing schemes.

The address format and routing algorithm chosen is quite simple, but it is powerful enough to meet the requirements. An address is composed of a number of terms. All terms are syntactically equivalent. Each MTA retains a local database indexed by term name. The database can contain the following entries:

- Mailbox name – The name associated with a user agent at the local node
- Replacement terms – Names replaced by zero or more other names
- Node names – Names of remote systems containing other MTAs

The algorithm used by Message Router when processing a recipient's address is as follows:

1. Look up the next term in the local database.
2. If this term matches a locally resident user agent, deliver the message. Processing is complete.
3. If this term matches a replacement term, make the specified change to the address and go back to step 1. Otherwise go to step 4.
4. If this term matches a node name term, pass the message to the MTA at that node. Remove the current term from the address before the message is passed on. Processing is now complete.
5. Report that the message could not be delivered.

When the Message Router was first released, most DECnet networks were quite small, and this algorithm was very successful. However, our own experience with our internal network showed us that in large networks, static routing databases were difficult to manage. To overcome this problem, Message Router was updated to allow a message to be forwarded in step 4 if the term name matched any DECnet node name.

This updated algorithm can be used with very low management cost by storing only local user agent names in the transfer service database. Message addresses then contain only node names and user agent names, such as "John Smith at A1 at Bostn1." In this case the local transfer service database contains only one entry, A1, the name of the mailbox associated with the local user agent (ALL-IN-1).

Additional management can provide additional services if required. These services include routing tables to simplify addressing and management in large domains, and alias names for nodes and user agents.

Note that the transfer service uses up terms in the address one at a time until a user agent name is found. Thus it is possible to include terms in the address that the transfer service never sees because these terms follow the user agent name. For example, in the address "John Smith at A1 at BOSTON," BOSTON is the name of a node, A1 is the name of a user agent, and John Smith is a term not seen by the transfer service. However, the term John Smith can be used by the user agent for its own purposes, in this case to identify the particular user of the user agent.

Programmer's Interface

The original Message Router product, the DECmail User Agent, the VMSmail Message Router Gateway, and ALL-IN-1 software provided a full range of electronic mail services as specified by the FIPS 98 report. However, two factors led to unnecessarily high engineering costs to initially develop and debug code and to perform interworking tests. First, the contents of the encoded message can become complex; second, each of these early products was developed separately.

Designers recognized that the same problems would be experienced in the future development of user agents, gateways to other electronic mail systems, and nonelectronic mail applications. As a result, a formal library of interface routines was implemented for the 1985 release of the VAX Message Router product. The Message Router interface routines (MRIF) acted as the exposed interface between the MTA and the UA and provided the following services: assembling and disassembling messages, posting messages to the MTA, and fetching messages from the MTA. A new product, the Message Router Programmer's Kit, provided these routines to customers to allow them to build their own mail user agents, gateways to other mail systems, and nonmail applications.

MRIF allows applications to specify the data items in the envelope required by the transfer service to deliver the message to the proper recipient. Further, applications can specify in the content the data items that support the services of mail user agents or nonmail applications.

For nonmail applications, the message content is arbitrary, and MRIF allows such contents to be assembled and disassembled.

For mail user agents, MRIF supports the specification of data items that provide electronic mail services such as multiple recipients for a single message, multiple message parts of various formats, and notification to the sender of receipt or nonreceipt of a message.

CCITT X.400

In 1980 the CCITT began a four-year study of message handling systems. They were strongly influenced by the NBS and IFIP work but had an additional goal: the specification of a complete message handling system which included the interworking between the message handling system and public data networks. In 1984 CCITT published the X.400 series of Recommendations. The X.400 Recommendations included the model of messaging systems in which the services of the message transfer agent are distinct from the services of the user agent, as originally described by IFIP working group 6.5. The Recommendations also included a definition of message encoding very similar to the NBS standard. In addition, the X.400 Recommendations included the definition of the means by which MTAs communicate and the method by which messages are addressed and delivered world-wide.

Message Router X.400 Gateway

Following the publication of the CCITT 1984 X.400 Recommendations, we recognized that our support of the NBS standard in hopes of a world-wide multivendor messaging network were too early. Few vendors other than Digital implemented the NBS standard. However, our goal of supporting a world-wide messaging system remained clear. We decided to lead the way by building the first X.400-conformant messaging system.

If we were to build a new Message Router that conformed to the X.400 Recommendations, the cost would be high, the time to market would be long, and some compatibility problems could arise with the software our customers were currently using. In addition, since there were as yet no other X.400 implementations, it was not clear that such a cost was justified. However, since the messaging services specified by X.400 are almost identical to the messaging services specified by the earlier standards plus some extensions, we decided to build a translating gateway between Message Router applications and X.400-conformant applications. The gateway need only reformat the service elements and map addresses between the two systems. Using this approach, it was possible for a small team of engineers to complete the first release of the Message Router X.400 Gateway in a very short time.

Today's Message Router

With the release of the first Message Router X.400 Gateway, the benefit of interconnecting all the diverse computer systems at a customer's site using electronic messaging became even more apparent. At this same time, Digital's own internal DECnet network had grown to tens of thousands of nodes. It was clear to the Message Router engineering team that we had to stop thinking of the Message Router as a product installed on network nodes and start thinking of it as a distributed network application—the MAILbus.

The services of the VAX Message Router version 2 product were excellent in small homogeneous networks, but suffered from problems in large mixed system networks. The two most serious problems were management and addressing complexities. These problems have been solved by the services offered by the current releases of the MAILbus products.

Management Service

The first problem encountered in a large heterogeneous network is the installation and configuration of numerous components which together provide the message handling, directory, and gateway services required. Each component offers a number of configuration options to support the various configurations within each customer site. To make the installer's job as easy as possible, all MAILbus components are shipped with a default configuration suitable for most customers. Following installation, the configuration procedure asks a few simple questions and performs the functions necessary to make the component both operational at the local node and a part of the distributed application.

On those occasions when problems occur, such as running out of disk space, the problem must be detected as soon as possible and the solution must be easy to implement. This is an especially difficult problem for the transfer service. Since it runs in the background, problems in the transfer service often are not detected until long after they occur. In a large network, a small problem in the transfer service on one system can become a large problem quite quickly. To solve this problem, we chose to retain the extensive error-logging facilities of each component and to add to each component a service that posts an exception report to a central node whenever a potential problem or serious error occurs. Each exception report includes text describing the problem, gives details of the problem, and refers the reader to a

document with detailed instructions for solving the problem. In a large network, groups of nodes can each send exception reports to their own central nodes so that the management of the entire network can be split up into smaller management domains.

Regularly encountered is the problem of managing the data required to control each end user's use of the messaging system. A typical MAILbus backbone would contain a number of MTAs, each responsible for a number of end users, and a number of gateways, each able to service any user at any MTA. We therefore had to examine how we could make the gateway services available globally, retain some form of access control, and allow the minimum management overhead. Any technique that required a manager to enter management data once at the MTA and once at each gateway was too costly for the manager and was prone to errors and inconsistencies. We decided to provide a network-wide end user directory in which the information required by the gateways was entered once and then made available automatically and globally. This directory service could also supply lookup services for end users who needed information about other users in the network.

Directory Service

The Message Router Directory Service within the MAILbus provides the services required by both the gateway and the end users. Included are the addressing and management information necessary for the gateway and a descriptive directory service for end users through their user agents. Entries in the directory service contain a number of different attributes about each user, such as name, job title, work location, and electronic mail addresses.

One of the most important features of the MAILbus—the ability to exchange messages freely between different messaging systems — is made possible by the storage of multiple mail addresses for each user in the directory service. For example, an IBM PROFS user can address a message to an ULTRIX mail user as if that person were another IBM PROFS user. When this message arrives at the VAX Message Router/P Gateway, the gateway looks up the IBM PROFS address in the directory service and finds that the user is actually an ULTRIX user and forwards the message through the ULTRIX mail connection gateway to be delivered to the ULTRIX user.

User agents, such as ALL-IN-1, use the same service to find the mail addresses of message recipients. Searches in the directory can be made on Boolean expressions formed on values of the attributes. For example, one can easily look for the manager in the sales department whose first name is Joe, or for all the engineers who work in a particular building. In addition, multiple values can be entered for many attributes, allowing searches on different values for the same end user to succeed, such as the use of maiden name, nickname, or easily misspelled names.

The design team members responsible for the directory service was faced with a difficult choice. They understood the requirements well and were committed to support the use of international standards, but the directory standards work at the time was in a very early stage and was changing often. The decision was made to use as much of the emerging standards work as possible, but not to compromise meeting the product requirements. As a result, the Message Router Directory Service provides storage and search services that are very similar to the CCITT X.500 directory service Recommendations, but use a different distribution and replication mechanism.

The distribution and replication mechanism had to provide good search performance, robustness, support for large numbers of users and network nodes, and ease of management. The solution that was chosen implements a single database stored across all the nodes in the network. Each node stores locally all the information associated with local users. A small number of nodes in the network are designated as world search nodes. World search nodes hold a copy of all objects from all other nodes.

Searches for local users involve only a local database search and are quite fast. Network-wide searches need only involve a single world search node — either the local node if it is a world search node, or a single remote node. If one world search node is unavailable, any other will do. Management of the database involves simply updating local users' database entries. The directory automatically sends updates to all world search nodes when they occur.

Conclusion

The support of international standards for the exchange of electronic mail and messages provides the assurance that implementations can interwork. However, the successful implementation of a standard conformant product on a large scale requires careful design and planning to ensure that problems of management and scale are properly handled.

General References

D. Whitten, "The Evolution of a Corporate Electronic Mail Network," *Proceedings of the International Communications Association 38th Annual Conference and Exposition* (May 1985).

CCITT VIIIth Plenary Assembly, "Data Communications Networks, Message Handling Systems, Recommendations X.400 to X.430," *CCITT Red Book*, Vol. VIII (Geneva: CCITT, 1985).

Alan Abrahams

VAX/VMS Services for MS-DOS

VAX/VMS Services for MS-DOS is a distributed application which provides file and disk services to personal computers from a VAX/VMS system. Using a server-based model, the product's heterogeneous file system seamlessly integrates the remote file system into the MS-DOS environment. The file server allows simultaneous access to shared data on the VAX system. The disk server provides a logical block service that maps MS-DOS disk access requests into a VMS container file. One of the challenges for product developers was the design of a remote boot service, which allows any personal computer to be booted over the network by means of the disk server.

VAX/VMS Services for MS-DOS allows personal computers (PCs) to share resources located on one or more VAX/VMS systems. The product uses a client/server model to access the resources on the VAX system. The system providing the resources is referred to as the server; the PC requesting the resources is referred to as the client. The two classes of server, which this paper describes, are the file server and disk server.

The file server provides both file and printer services. File services allow access to the VMS file system from the PC; printer services allow access to VMS printer queues from the PC. The disk server provides access to a section of a VMS disk that has been formatted as an MS-DOS disk drive. This disk section is called a virtual disk file.

The primary goals of the PC integration project were as follows:

- Integrate PCs into the VAX/VMS environment in a manner that allows the resources offered by the server to appear as local MS-DOS disk drives and printers on the PC. PC applications can therefore utilize the VAX file system, virtual disk drives, and printers attached to a VAX system as if these resources were available on the PC itself.
- Utilize DECnet software to provide both local area network (LAN) and wide area network (WAN) access to the server. Thus any PC can access any server on the network without the use of a gateway. Most competitive products are either restricted to LAN access or require the use of a gateway on a local server to access a remote server over a WAN. The elimination of the gateway significantly improves performance over a WAN and simplifies connection management since all nodes on the network are visible to the PC.

One of the fundamental decisions made toward achieving these goals was to utilize the Microsoft Networks/OpenNET (MS-NET) Architecture. This architecture, described in the following section, provides a high level of integration with MS-DOS. Further, it supports a generic transport interface that allows an original equipment manufacturer (OEM) to customize MS-NET for different target network environments.

Architecture

The Microsoft Networks/OpenNET Architecture defines

- The application programming interface (API) used by the MS-NET Redirector to communicate with the network transport supplied by the OEM
- The communication protocol used by the Redirector to communicate with an MS-NET server
- The management API used by the Redirector for maintaining connections to remote resources

Figure 1 is a block diagram of the file and print redirection subsystem on the client.

An application executing on a PC makes MS-DOS system calls to access files and printers. The system calls are passed by MS-DOS to the Redirector and transmitted across the network to the file server. The server performs the requested function and returns a response to the Redirector which in turn completes the system call by returning a response to the application.

Described below are the steps taken to connect to the file server and to access the service.

1. To access a resource on a server, an application calls the Redirector API to request a connection

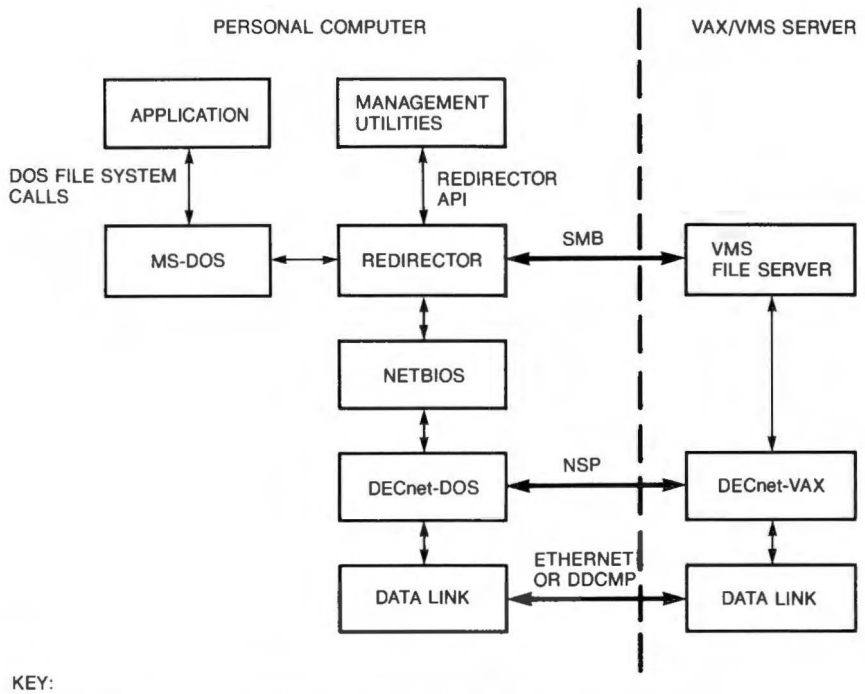


Figure 1 File Server Architecture

to the server controlling the resource. Typically the call is done by means of the user interface (UI) utility, USE.

2. The Redirector calls the network basic input-output system (NETBIOS) to create a virtual circuit to the server. NETBIOS is a standard API used on PCs for interfacing to communications networks.

   Digital supplies a NETBIOS mapping to DECnet-DOS software. This mapping allows network applications written to the NETBIOS interface to perform nontransparent task-to-task communication through DECnet to any other DECnet node in the network.

3. On successful completion of this call, the Redirector creates a disk or printer device which is redirected to the remote network resource.

4. An application may access the resource by making the standard DOS system calls that specify the redirected disk drive or printer.

5. The Redirector uses the server message block (SMB) protocol to export the local file and print system calls to the server. SMB is a request/response protocol; therefore, all requests are initiated by the client, and each request receives a single response.

File redirection exports file system access requests to a server. The server then maps the DOS file system directly to the native file system on the server. In the VMS system, this file system is ODS2.

In addition to file and print redirection, the basic MS-NET architecture has been extended to include disk redirection to a virtual disk. These architectural extensions are proprietary to Digital and were provided to facilitate the remote boot feature described later in this paper.

Disk redirection exports the logical block I/O requests to the server. In this model, the file system continues to run on the client while the disk server maps the client's disk I/O requests to a section of a VMS disk that has an MS-DOS structure. On a VMS disk, the MS-DOS virtual disk is mapped to a 512-byte, fixed-length sequential file, referred to as the virtual disk file.

Figure 2 is a block diagram of the disk redirection subsystem on the client. In disk redirection, a block device driver emulates between four and eight logical

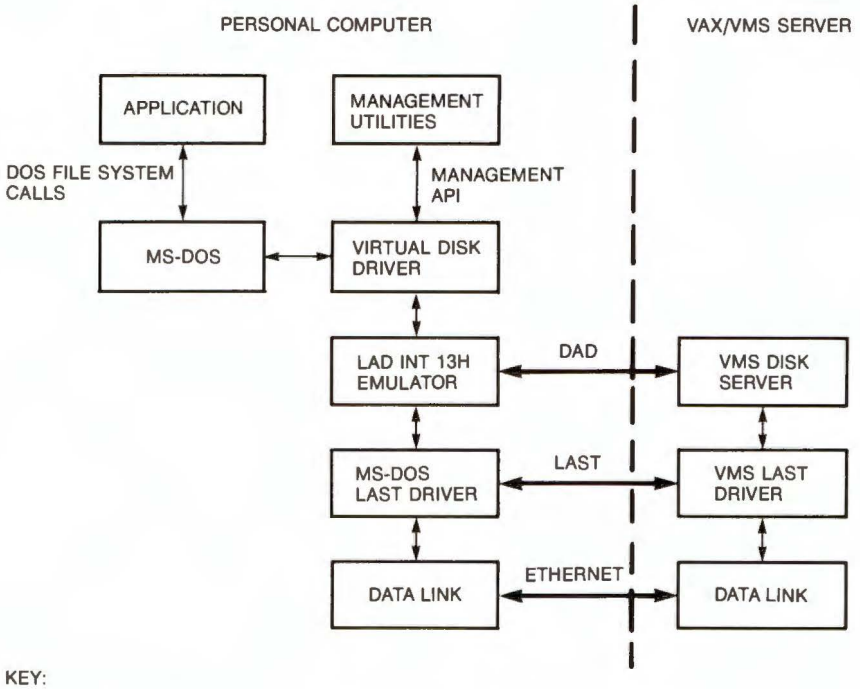


Figure 2 Disk Server Architecture

disk drives (user configurable). The driver may be accessed by the standard MS-DOS file system calls or by calls directly to the block device driver interface. Extended functions in the block device driver interface are provided for managing the connections to a virtual disk.

The block device driver communicates with the virtual drive by making BIOS INT 13H calls to the local area disk (LAD) handler. The LAD INT 13H handler performs the actual communication with the disk server. LAD uses the local area system transport (LAST) to communicate by means of a virtual circuit to the disk server.

File Server

The highest level of integration between an MS-DOS client and the VMS system is provided by the file server. Integration is highest because both systems share a common file and print system.

The file server was designed to run as a single detached process (rather than to run one process per client). This design reduces the VMS scheduling overhead and resource consumption when large numbers of PCs are connected to a server. The file server process is normally run as part of the system startup procedure and declares itself as DECnet object 64.

All incoming connections to the file server are passed by the VMS system to the file server process. The file server runs in user mode and is multithreaded. All I/O functions performed by the server and most system calls are nonblocking. Thus multiple clients can be supported without any one client blocking requests made by others.

The sections following present details of the file server design, that is, mapping, access to services, security, and byte range locking.

File System Mapping

To make the VMS file system appear to the client as an MS-DOS file system, we needed to define a mapping between the two systems. This mapping must be symmetric because it determines both how an MS-DOS file is represented to VMS and how a VMS file is represented to MS-DOS. In choosing a file mapping, the following main problems had to be overcome:

- MS-DOS files have no system-defined internal format, except that known to the application which maintains the file. The record management

services (RMS) file type that most closely corresponds to an MS-DOS file is stream format.

- The character set allowed for MS-DOS file names is much richer than that allowed by the VMS system. Thus some mechanism had to be found to represent an MS-DOS file name in the restricted character set allowed by the VMS file system.
- MS-DOS file attributes have no direct counterpart on the VMS system. These are the system, hidden, and archive attributes.

The following general rules were chosen:

- An MS-DOS file can be represented on VMS as either an RMS stream file or optionally an RMS 512-byte, fixed-length sequential file. The default representation is a stream file; the sequential representation was introduced to facilitate data sharing between applications, such as WPS, which run both on the PC and on the VMS system. The stream representation was chosen as the default because (1) it is the RMS file type that most closely corresponds to an MS-DOS file, and (2) stream file format allows ASCII text files created on an MS-DOS system to be edited using VMS standard EDT and TPU text editors.
- MS-DOS file names are mapped to VMS file names by converting characters that are illegal in a VMS file name to __XX, where XX is the two-digit hexadecimal code for the illegal character. If two underscores occur together, the second underscore is also converted using this algorithm.
- VMS files with names or extensions that are larger than those handled by MS-DOS are invisible to the client. That is, an MS-DOS directory listing shows only those files that have names and extensions compatible with the MS-DOS file system.
- Those MS-DOS file attributes that do not correspond to VMS file attributes, i.e., hidden, system, and archive, are mapped to an application access control list entry (ACE) stored in the file's access control list (ACL).
- VMS sequential, relative, and indexed files are converted to byte streams when read by MS-DOS. The file server performs this conversion according to the record attributes associated with the file. Thus MS-DOS may read any VMS file organization. However, in general, only sequential variable files with ASCII text contain data that can be used by a PC. Access to sequential, relative, and indexed files (with the exception of the 512-byte, fixed-length sequential file type) is limited to read only.

Access to Services

Whenever a client connects to a file server, the client specifies a service name — an ASCII text string of 1 to 25 characters. A service is an access point to the resource being offered by the server. A list of available services is stored in the file server's service database. In addition, the file server uses the user authorization file (UAF) as a secondary service database to allow users to connect to their personal accounts.

Services entered in the service database have an associated set of attributes, which are listed in Table 1.

Table 1 Service Attributes

Attribute	Description
Root directory	Top-level directory that a client can access through the service
Service type	System, application, or common; specifies type of security applied, that is, the level of access to the service
RMS protection	Default protection mask used when files are created
File type	Stream, or 512-byte fixed-length sequential file
Queue	Name of the VMS print queue to be used
Form	Name of a form entered as a prefix on print jobs

Access to services within the service database is controlled by access control lists. Specific users may be granted the use of a service, or a service may be made generally available. When a service is created, the service type attribute must be specified. The file server uses this attribute to determine how access control to the files within the service is performed.

Security

MS-DOS has no inherent security scheme for file access. Thus to protect files stored on a server from unauthorized PC access, we had to define a scheme that authenticates users when they connect to services on a file server. One approach considered was to pass the authentication information to the server in the DECnet connect message. However, DECnet-VAX software does not pass this information to objects, such as the file server, which service their own inbound connections.

The fallback approach was to extend the use of the SMB TREE connect message to pass the authentication information to the server when the Redirector requests a service connection. This message

already contained a field for passing password information. Therefore, we could extend the message by encoding the username in the service name field. The composite service name is formed by concatenating the actual service name with the username, using the percent sign (%) as a separator. When no username is found in the service name field of the TREE connect message, the file server determines if a guest account has been specified by the system manager. The server then uses the guest account to determine access privileges.

Once the file server has authenticated the username/password pair against the UAF, it extracts the associated access privileges from the UAF and rights-list databases. The server then uses this information together with the access control information in the file server service database to determine access privileges to files in a service.

Before the scheme was complete, it was necessary to solve one more problem which related to the conflict between system manager and user privileges. One use of the file server is to store application software in a central service. This software must be accessible by many users who have diverse access privileges. These privileges may often conflict with those of the system manager who loaded the application software to the server and who is thus the owner of the files.

To overcome this problem, the file server uses two types of security: share-level and user-level security. The algorithm applied is based on the service type attribute specified when the service is created.

Share-level security is applied to system and application service types. These services typically contain system and application software loaded by the system manager which must be accessible to many users who have diverse access privileges. Access to these services is controlled by two identifiers referred to as PCFS$READ and PCFS$UPDATE. The PCFS$READ identifier grants read access, and the PCFS$UPDATE identifier grants write access. These identifiers are attached to the root directory of the service as default ACE entries. They are propagated to all files created within the service. The server automatically gives either the PCFS$READ or the PCFS$UPDATE identifier to a user. The determination as to which identifier is assigned is based on the access control list defined for the user in the service database.

User-level security allows users access to the common service type and to user accounts from the UAF. This level is the standard VMS security algorithm based on the identity of the user accessing the file.

The process by which the server determines access to a file is described as follows:

- When a user connects to a service, the file server authenticates the user. Every service connection request must explicitly specify a username and password that corresponds to a UAF entry, or the system manager must have given the file server a default account to use when no explicit authentication information is given.
- On successfully authenticating a user, the file server extracts the user's default privileges, user identification code (UIC), and rights-list entries from the UAF and rights-list databases.
- If the user is allowed only read access to the service, the server gives the user the PCFS$READ identifier. If the user is allowed write access, the server gives the PCFS$UPDATE identifier.

When the user accesses a file in the service, the file server uses the $CHKPRO system service to determine whether the user has sufficient privileges to perform the requested operation.

Byte Range Locking

The MS-NET architecture allows files stored on a server to be simultaneously accessed by multiple PCs. Updates to files are coordinated by the MS-DOS byte range locking calls. This approach contrasts with that of RMS, which coordinates shared file updates using record locking. The following example illustrates how two programs on different PCs might attempt to lock the same byte range.

Program A locks bytes 20 to 50 in a file, and program B locks bytes 0 to 21. Both programs are attempting to lock bytes 20 and 21 in the file, which produces a lock conflict. The lock manager must be able to detect this conflict and reject program B's attempt. For the VMS lock manager to represent these locks, the resource being locked must be identified by a unique name. In the example above, the VMS lock manager would not be able to detect an overlap of lock ranges if the byte range was encoded into the resource name.

Since byte range locks cannot be represented by the VMS lock manager, the file server maintains its own lock database and arbitrates access to shared files.

In VAXcluster configurations, there may be multiple file servers offering a rooted directory tree on a shared drive. Therefore an interlock mechanism must be provided to coordinate access from the different file servers. The file server uses the VMS lock manager to coordinate access to these files by taking out a private exclusive lock on a file when an application opens the file. If the lock is granted, the file server becomes the lock master. If the lock is *not* granted, the file server determines the lock

master and reroutes all access requests for the file over a DECnet virtual circuit to the lock master. In this way, only one file server controls access to the file.

Disk Server

The disk server provides low-level redirection at the physical device level. Primarily this server assists in the remote boot of PCs, which is described below in the section Remote Boot. On the VMS system, the disk server is implemented as two device drivers and a control process that coordinates cluster-wide access to virtual disks. Virtual disks may be shared by many users reading data, but virtual disks are limited to exclusive access by a single user who reads and writes data.

Device Drivers

The two device drivers are LASTDRIVER, which provides the communications transport, and LADDRIVER, which implements the local area disk protocol. The LAST transport is specifically designed to operate on a LAN, such as Ethernet, where the physical transmission medium is highly reliable. LAST protocol packets received by the Ethernet driver on the VAX system are passed directly to LASTDRIVER using the ALTSTART interface. LASTDRIVER uses a callback mechanism to pass data to LADDRIVER, which uses the ALTSTART interface to pass I/O requests directly to the disk driver.

When a virtual disk file is mounted, the file is opened by a detached process (LAD$KERNAL) and handed off to LADDRIVER. Each virtual disk file is fully mapped, so there are no window turns. LADDRIVER maintains a write-through cache in nonpaged dynamic memory. High hit rates on this cache decrease the amount of disk I/O performed and improve the response time for servicing MS-DOS block I/O requests. To further minimize the amount of disk I/O, the LAD transaction size can be controlled from the client. (LAD transaction size determines the multiblocking of read and write requests.)

For high availability of virtual disk services in a VAXcluster environment, the disk server software may be run on all nodes in the cluster. A virtual disk service can be mounted on multiple nodes in the cluster. To interlock the mount requests, the LAD$KERNAL programs on each cluster node use the VMS lock manager. If write access to the virtual disk service is permitted, the first mount request for the virtual disk results in an exclusive lock being granted to the node issuing the request. Subsequent mount requests from other nodes result in the lock being queued. The mount requests from these other nodes go into a pending state. If the lock is released by the first node, the lock is granted to one of the other nodes and the disk is automatically remounted. This technique allows automatic re-offering of the disk service to occur when the current server node becomes unavailable. In the case of read access to the virtual disk, all nodes may concurrently access the virtual disk file; thus the service may be simultaneously mounted on each server node.

Disk Service Connections

When the client system connects to a virtual disk service, the location of the virtual disk service is determined dynamically by a solicit-response protocol. This protocol is part of the LAST transport. The client node sends a solicit message and gathers responses from all nodes offering the service. These responses are filtered according to any selection criteria that were given by the client and the rating (highest) of the service making the connection. The system manager assigns each service a rating when it is mounted. This rating predisposes clients to specific servers when multiple servers are offering the same service.

Remote Boot

One of the most challenging aspects of the PC integration program was the provision of a remote boot capability for MS-DOS. In organizations where there are large numbers of PCs, remote boot allows the system manager to maintain the end user's PC environment from a centralized boot service. Our principal goal, then, in developing remote boot was to provide the ability to boot any vendor's PC over the network. Since each PC vendor distributes its own variant of MS-DOS, it followed that if we achieved this goal, then no modifications to the operating system would be required.

Our initial implementation was designed to support the VAXmate product and PCs equipped with the Digital Ethernet PC Network Adapter (DEPCA). In both cases the maintenance operation protocol (MOP) boot code exists in read-only memory (ROM). The PC triggers the remote boot process by issuing a MOP system boot request which is serviced by a VAX boot server process. The boot server downloads a network loader program to the PC. The network loader is an Intel assembler program encapsulated in an RSX task image wrapper. The loader's function is to load DOS from the network. The network loader is a permanently resident part which contains the random-access memory (RAM) data link, the LAST transport, the LAD software, and a transient control program to drive the load process.

Since MS-DOS can only be booted from drives A or C, we had to provide a low-level disk emulation by means of INT 13H that would emulate drive A during

the boot process. This emulation is the function of the LAD/LAST software in the network loader.

Once the network loader is resident in the PC, it must relocate its permanent part into an area that will not be overwritten by MS-DOS while the PC is running. On a VAXmate system, the loader achieves this relocation by copying the code into the 128-kilobit (Kb) network RAM. On a PC, the size of physical memory is reduced by the size of the permanently resident code (approximately 17Kb), and then the code is relocated to high memory just below the 640Kb boundary. The network loader must next find the name of the boot disk service for the PC. This name is determined by converting the 48-bit Ethernet hardware address (which is guaranteed to be unique) to its ASCII representation, e.g., 08-00-2B-08-41-3C. This name is then used to connect to a virtual disk which has been formatted as an MS-DOS bootable floppy disk.

The network loader reads the DECnet-DOS database from which it gets the data link configuration parameters. If no database exists, the system defaults are used.

Finally, the loader reads the boot block from the virtual disk and transfers control to the bootstrap loader contained in the boot block. The bootstrap loader uses the BIOS INT 13H calls to read MS-DOS from the boot device. These calls are intercepted by the LAD INT 13H handler. Thus access to the virtual disk is transparent to the bootstrap loader. The last step in the process is to restore the real drive A, since at this point drive A is mapped to the virtual disk.

Performance

This section describes several techniques implemented to enhance the performance of both the file and disk server. These techniques were required to provide service response times similar to those seen when an application is running from a local hard disk on the PC.

One of the major differences between the MS-DOS and VMS systems is the manner in which file header information is stored on both systems. DOS stores the header information in the directory, whereas the VMS system stores this information in the file header. Thus DOS has access to this information as soon as the directory is read into memory. On the VMS system, however, additional I/O must be performed to read the file headers. Thus a DOS directory command generates high overhead on a VMS system. The file server therefore had to minimize system overhead when enumerating directory entries and also make directory operations nonblocking to other PCs using the file server. To accomplish this, the file server makes QIO calls to XQP rather than using the $PARSE and $SEARCH functions in RMS. This technique also allows the file server to obtain the file header information used by the $CHKPRO system service when determining the user's access rights to a file.

A second technique which improves file access speed is the provision of a read-ahead cache in the file server. Many PC applications process files sequentially; the file server automatically detects this access mode and uses a double-buffered prefetch algorithm to read data from the file before it has been requested by the client. This prefetch results in fewer I/O requests since the file server prefetches the maximum amount of data that can be returned to the client in one message (8Kb). The algorithm also increases data throughput speeds since the next read request from the client can normally be satisfied with data already in memory.

The disk server is not cognizant of the file system on the virtual disk and attempts to reduce disk I/O by using a write-through cache. Like the file server, the disk server prefetches data into the cache. Data is purged from the cache based on a least-recently-used algorithm. Since the disk server is implemented as a VMS device driver and the cache is stored in nonpaged dynamic memory, the response to client I/O requests is fast.

Overall client performance can be improved by judicious use of the file and disk server in a hybrid combination. In many instances applications can be shared for read-only access by multiple clients through the disk server. Moreover, the data on which the application is operating can be shared for read-write access through the file server. This partitioning can also result in higher numbers of PCs being supported by the server, since the disk server overhead is much lower than the file server.

References

DECnet-DOS Programmers Reference Manual (Maynard: Digital Equipment Corporation, Order No. AA-EB46C-TV, April 1988).

IBM PC Network, PC Adapter Manual (Boca Raton: IBM Corporation, Reference No. 6322505, September 1984).

IBM Personal Computer Seminar Proceedings, vol. 2, no. 8 (October 1984).

IBM Token-Ring Network, PC Adapter Manual (Research Triangle Park: IBM Corporation, Reference No. 69X7830, June 1986).

P. Mierswa, D. Mitton, and M. Spence, "The DECnet-DOS System," *Digital Technical Journal* (September 1986): 108–116.

Peter G. Viscarola
Jeffrey E. Watkins

The WAVE Tools Base for Protocol Testing

One goal of data communications protocol testing is to ensure that the protocols that are implemented adhere to the relevant standards. Reaching this goal is often time-consuming and difficult. To help simplify this particular type of testing, called protocol conformance testing, Digital's Networks and Communications (NAC) Conformance Engineering Group has developed the WAVE tools base. This unique tool forms a reusable platform that can be used at almost any protocol layer. The WAVE tools base implements a programming language specifically designed to make developing protocol tests and prototypes easy. Examples of the WAVE design features presented here are derived from the group's experiences in developing a conformance test suite for the DECnet Phase V network routing layer.

The WAVE tools base and the WAVE testing language were designed and developed to help solve the unique problems inherent in data communications protocol testing. The reusable framework provided by WAVE allows tool developers to build protocol tests quickly.

Testing Sophisticated Protocols

Developers of sophisticated distributed applications face the problem of how to test their protocol implementation to ensure it is operating in accordance with applicable standards. One method of testing employed by developers runs an implementation against a copy of itself. This method does not test the correctness of the protocol itself, however, and may result in an implementation that communicates with itself but not with implementations developed elsewhere. Most developers recognize the pitfalls of this type of testing. As a result, they often create crude test programs which allow them to exercise complete control of the messages sent.

Test programs are essentially debugging aids which are written quickly and are often discarded when development is completed. The developer does not spend time adding features to the test program that would be considered essential in a product quality tool. Test programs display the following characteristics:

- A user interface that suits the developer at the time the tool is created. This interface is not necessarily easy to use, and there is no consistency in the style of interface presented by different tools.
- Suitability for only one particular purpose. These tools usually cannot be easily modified to perform other testing functions or to accommodate changes required by a new version of the protocol specification.
- A tendency to fail when used in environments that the developer did not explicitly foresee.
- A paucity of documentation and support.

We saw a definite need in the software development community for a testing tool that embodied the product qualities of ease of use, extensibility, robustness, and documentation and support. There was also a need within our own group, NAC Conformance Engineering, for such a tool. Our group is responsible for developing conformance testing methodologies and tools to test Digital's networking products. We saw a large number of products that required testing, and we needed a quick way to build quality conformance test tools for these products.

From our past experience in developing test tools, we knew that over half of the overall development time is spent creating the test tool environment. This environment consists of a language in which to express test cases (scripts), a compiler for this language, and an interpreter to execute compiled test cases. If we could create a reusable environment, the development time required for each individual tool would be significantly reduced.

The WAVE tools base is the reusable environment that we created. WAVE is not a tool itself, but a framework on which to build a test tool for a specific protocol.

Goals of the WAVE Design

When we began to consider the functional requirements for our reusable test tool platform, we quickly arrived at the following minimum requirements:

- The script language needed to be simple to learn and easy to use, and to appear familiar to even entry-level software engineers. This would reduce the "lead time" necessary for engineers to learn the language and would help the language become easily accepted by the software development community.
- The language had to provide direct access, by field name, to any field within a protocol data unit (PDU). Fields could not be restricted as to length or starting position within a PDU, since fields do not necessarily start on an even byte boundary. Also, the language had to allow the developer to define both the starting location and the length of a field within a PDU dynamically during script execution. Together, these requirements would allow the script language to support all common data communications protocols between ISO layers 2 and 4, and even many protocols at layers 5 through 7.
- The tester system needed to support easy access to, and control of, multiple separate timers. These timers would be used to measure time intervals between message exchanges, or to time the execution of an individual test. The minimum required timer granularity was 10 milliseconds (ms). It was necessary that timers count up (i.e., measure elapsed time) or count down from a predetermined value (as in waiting for an interval to pass before proceeding to another action). Another need was the ability to pause timers while they were running and later to continue their execution. Finally, and perhaps most important, it was highly desirable to be able to update timers in real time. This feature would allow the value in a timer to be examined and perhaps changed at any time, even while the timer was running.
- The tester system had to allow portability of test scripts among many different platforms. We anticipated a need to have the tester system run on several different operating systems, including VAX/VMS, ULTRIX, and perhaps even MS-DOS. Test scripts that are operating-system-independent can therefore be run on any of the various platforms without change.
- The tester system had to support the automatic logging of PDUs exchanged between it and the implementation being tested. This logging facility was required both for debugging of test scripts and for providing details of the observed protocol behavior to the implementors.
- Other requirements included support for typical integer arithmetic and logical operations, and a flexible means to interface a test script to the underlying protocol service.

Our initial inclination was to see if we could find an existing language for protocol testing that would satisfy our requirements. We therefore began to search through the literature (including journal articles, technical papers, and ISO standards). Much to our surprise, no satisfactory language seemed to exist.[1]

We next considered meeting our requirements by developing a clever set of macros, or subroutines, in a general-purpose programming language. We attempted some exploratory work in this area and were unsuccessful each time. The very attributes of general-purpose programming languages made them more difficult to learn and use than we needed. We found it extremely cumbersome to use most of these languages to describe PDU fields of arbitrary length. Even the BLISS language, which has a powerful fielding facility, allows only fields of up to one "fullword" in length. (This machine-dependent value is 32 bits on a VAX system.) Fields longer than this must be referred to by pointers to their starting addresses. References to these fields for the purposes of comparison or assignment require the user to know and supply the field length, which was certainly not desirable.

When our attempts to use an existing programming language failed, we allowed ourselves the freedom to design our own language for use as part of the tools platform. We wanted to achieve portability of test scripts among operating systems while retaining some amount of run-time efficiency. Taking a cue from the old BASIC "compilers," we developed a compiler that read a script source file and created an operating-system-independent binary object file

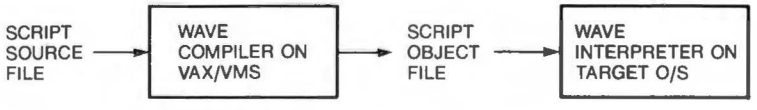


Figure 1 Basic WAVE Script Flow

containing pseudo code. The script object file could then be executed by a WAVE interpreter running on the target operating system. The flow is depicted in Figure 1.

This interpretive approach also allowed us to support fields easily within PDUs with starting locations and/or lengths that were determined (or changed) dynamically during execution.[2] The interpreter could easily keep track of the changing field definitions and use the appropriate one when the field was referenced. More details on how we implemented fields appears in the section Storage Declaration and Field Handling.

To provide an interface to the underlying protocol layer, and to accomplish those things that might prove difficult or time-consuming to do in WAVE script language, we developed the concept of "external routines." These subroutines are written in any language supported by the target operating system. The files containing the external routines are compiled on the target operating system. The resulting object files are then linked into an executable image (for example, on VMS the image is linked "/sharable"). The ability to develop many different sets of external routines (along with, of course, the generality of the WAVE language) makes WAVE a reusable tools platform and not just a tool for use with one protocol. External routines provide the mechanism by which a tool developer "customizes" WAVE with an interface to the underlying protocol service. (See Figure 2.) Furthermore, by implementing the necessary set of external routines on each target system (using whatever system-dependent services are required) a single script object file can be executed on any of those systems without modification.

To allow the validation of test results and to serve as a permanent record of an implementation's test responses, we saw the need for a log of the PDUs exchanged during a test. This capability was included in the design of the WAVE interpreter. The interpreter automatically writes records containing PDUs to a log file during test execution. These records are time-stamped and marked to indicate the source from which the PDU was received. If an implementation fails a particular test, the PDU log often provides useful information for determining the reason for failure.

Overview of the WAVE Script Language

To meet our design requirements, we needed a language that was simple to learn, easy to use, and familiar looking to most programmers. The language also needed to have very powerful field-handling capabilities.

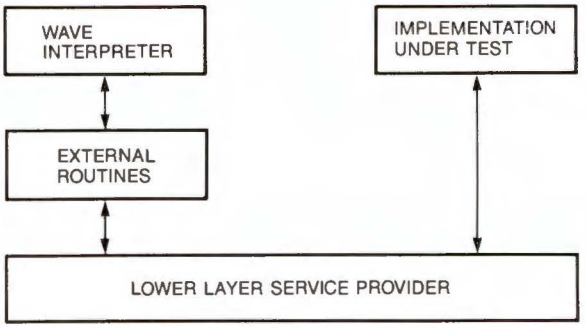


Figure 2 WAVE External Routines

To achieve these goals we decided to implement a minimum number of simple control structures that would be familiar to all programmers. We started by designing a block-structured if-then-else statement of the form:

```
if CONDITION then
    <statements>
else
    <statements>
endif
```

To this we added an indefinite loop capability, as follows:

```
loop
    <statements>
endloop
```

For completeness, we also included a simple "goto" statement. The only other control structure we included was a rudimentary subroutine capability modeled (again) on that provided by BASIC.

This facility was implemented as follows:

```
        gosub LABEL
        .
        .
        .
LABEL:  <statements>
        return
```

Assignment, logical, and arithmetic statements likewise appear in familiar forms. These features as well as those described above are best illustrated by a sample program segment:

```
MAX_WAIT_TIMER = MAX_WAIT * 100
loop
    gosub CHECK_ESHELLO
    if not GOT_ESHELLO then
        exitloop
    else
        if (RBUFF.V$F_PROTID eql 130) and
           (RBUFF.V$F_TYPE eql 2) then
            GOT_ESHELLO = TRUE
        endif
    endif
endloop
```

All these simple and familiar language elements help make WAVE simple to learn and easy to use. They also provide all program control abilities necessary to write sophisticated test scripts.

One facility not often found in simple programming languages is macro support. In WAVE, we implemented an extremely powerful keyword macro facility, including a full complement of conditional and lexical operations. Macros can be used by a sophisticated programmer to combine the elements of the WAVE language, and perhaps the calls to external routines, to create complex new facilities. In practical application, we have often used the macro facility to hide the details of external routine calling from the script developer. The use of macros allows test scripts to remain independent of the number and order of external routine parameters. Subsequent changes to the calling sequence of an external routine do not require modification of the test script; only the macro need be updated. Macros also ease external routine calls by providing appropriate defaults for external routine parameters.

Storage Declaration and Field Handling

We designed an extremely simple sort of storage mechanism for variables and PDUs. Since we were developing a language specific to protocol testing, we could not foresee any need for floating point arithmetic. (Only integer operations are supported.) Hence we developed only two types of storage declarations, as shown in the following examples:

```
variable DEVICE (4)
buffer   RBUFF  (1600)
```

The "variable" statement reserves 4 octets of storage and assigns to it the name DEVICE. Variables may be used as they are in most programming languages: as counters, holders of intermediate results, or for control information. The "buffer" statement functions identically to the "variable" statement, setting aside 1600 octets of storage for RBUFF. The only difference between variables and buffers is that buffers may be used with fields.

The heart of the WAVE system consists of the field manipulation capabilities that are built into the language. WAVE fields are used to address particular segments of a WAVE buffer. Buffers are typically used as storage locations for protocol messages. Any field may be used with any buffer (i.e., fields are not unique to a single buffer).

The field facility allows any contiguous set of bits within a PDU to be accessed, irrespective of its length or alignment. Fields are used to refer to locations within particular buffers by concatenating the field name to the buffer name, separating the two by a dot. For example,

```
RBUFF.V$F_PROTID
```

refers to the field V$F_PROTID within buffer RBUFF.

Fields are declared using the "field" statement. The name of the field is provided, along with the offset in octets and/or bits from the beginning of a buffer to the start of the field. The length of the field in octets and/or bits is also provided. The syntax for this statement is illustrated by the following example taken from the definition of fields for the ISO 9542 protocol:

```
field V$F_TYPE      (ROC$G_BASE+4,0 : 0,5)
field V$F_TYPE_RESV (ROC$G_BASE+4,5 : 0,3)
```

The field V$F_TYPE contains the PDU-type identifier. This field starts at the octet offset defined by the expression ROC$G_BASE+4 and bit offset 0. The field is 0 octets, 5 bits long. The field that follows it, V$F_TYPE_RESV, contains reserved bits that make up the remainder of the octet. As alluded to in the example, any of the four parameters of a field declaration can contain expressions.

Support for dynamic field declaration was implemented through a few variations on the field declaration scheme just described. First, field declarations may include variables (either alone or as part of an expression) as shown in the following:

```
field DATA (4:DATA_LENGTH)
```

Field statements may also include implicit references to other fields in the same buffer that is being referenced. This is accomplished as follows:

```
field V$F_E_SRCLEN  (ROC$G_V_SOURCE_POS : 1)
field V$F_E_SRCADDR (ROC$G_V_SOURCE_POS+1 :
                     .V$F_E_SRCLEN)
```

In the above example, the field named V$F_E_SRCADDR starts at the octet defined by the expression, and extends for the number of octets contained in the field V$F_E_SRCLEN. This facility was developed specifically for defining interdependent fields within a PDU. Such fields are common at all protocol levels.

Embedded field references were the last types of field support that we implemented. Embedded field references allow a user to refer to any bit within a buffer simply by describing its location, as with a field declaration. For example,

```
HOLDING = RBUFF.(ROC$G_BASE+5 : 2)
```

Here, the two octets of RBUFF starting at the offset defined by the expression are assigned to the variable HOLDING.

Timers

By choosing to implement an interpreter, we easily achieved our goal of implementing real-time timers (i.e., timers that can be interrogated or modified even while they are running). Timers are declared as shown in the following example:

```
timer T1
timer MAX_WAIT_TIMER
```

WAVE timers have a fixed size of four octets. The operation of timers is controlled via the WAVE timeoper statement. This statement allows timers to be started (either counting up or counting down), stopped, or waited on. The value in the timers is updated dynamically and may be used in any statement in which a variable is legal.

Literals and Translation Types

Another unique set of features of the WAVE language comprises the facilities provided for the specification and printing of literal data. Of course, standard ASCII text enclosed in quotation marks is supported as in most every programming language. However, when one thinks of PDUs, one often thinks of the data contained in a particular field of the PDU in a particular form. For example, the data field of a PDU is usually expressed as a series of hex bytes. The length field, however, makes more sense as a single byte expressed in the decimal radix. A pair of flag bits might most easily be understood when expressed in binary. WAVE allows you to have your choice in these matters.

Literals are expressed within single quotation marks. They are preceded by a translation-type specifier which indicates the radix and the unit of size of each of the elements of the literal. Individual elements of the literal string are separated by commas. An example will make the concept clear.

```
MESSAGE.V$F_LENGTH = %db '18'
MESSAGE.V$F_E_SRCADDR = %hb '49,01,AA,BB,CC'
MESSAGE.V$F_TYPERESV3 = %bb '111'
```

In the buffer MESSAGE, the field V$F_LENGTH is set equal to the decimal (byte) value 18. That this literal is expressed in the decimal radix, and is one byte in length is indicated by the translation type "db." Likewise, the V$F_E_SRCADDR field is filled with the hex bytes 49,01,AA,BB,CC. The field V$F_TYPERESV3 is set equal to the binary (byte) value 111. In each of these examples, the radix and the unit of size of each literal element is denoted by the translation type.

WAVE supports binary, octal, decimal, and hexadecimal radices. Sizes of 1 byte, 1 word, and 1 longword per literal element are supported.

We also used these same translation types for output display translation. Therefore, when the following statements

```
MESSAGE.V$F_LENGTH = %db '18'
put( terminal ) MESSAGE.V$F_LENGTH:db
```

are executed, the value 18 is displayed. By just changing the translation-type to hexadecimal, as follows:

```
put( terminal ) MESSAGE.V$F_LENGTH:hb
```

the output becomes 12.

The consistent use of a translation-type specifier and the ability to assign or display values of PDU fields in any common radix and in any common unit, make field usage easy and less prone to error.

A WAVE-based Tool for the DECnet Routing Layer

In this section we provide an example of an actual WAVE-based tool. We show how the aspects of WAVE are exploited in creating a tool. The example we chose is a conformance test tool for the DECnet Phase V network routing layer protocols.

Our example focuses on the testing of the ISO 9542 protocol. This protocol is one of several used in the routing layer (ISO layer 3) of DECnet Phase V implementations. We further narrow our focus by considering only that portion of the routing tester used for testing on local area networks (LANs).

The developer of a typical WAVE-based tool creates the following components:

- External routines for accessing the underlying protocol layer
- External routines to extend the capabilities of the WAVE language
- Macros for calling external routines
- Field definitions for protocol messages
- External routine error return codes

The tool developer is required to supply only the first component, external routines for layer access. All other components are optional. The test tool we developed for routing contains all of these optional components. Their inclusion in the tool considerably eases development of test scripts for the routing protocols.

External Routines for Layer Access

The routing tester uses the services of the data link layer to send routing layer PDUs to the implementation under test (IUT). In Figure 3 we see the configuration

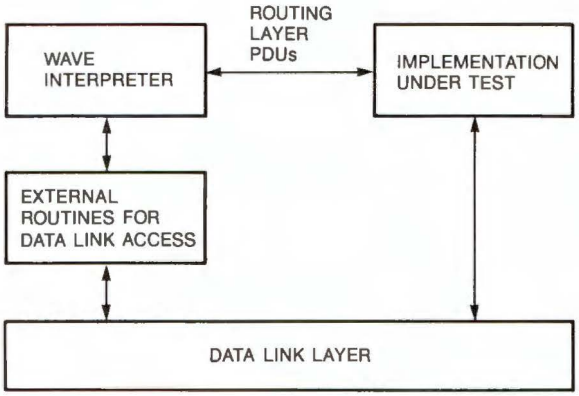


Figure 3 Routing Tester Configuration

employed by the routing tester. WAVE external routines (written in the BLISS programming language) interface the WAVE interpreter to the data link layer service. Routing layer PDUs are exchanged between the test script and the IUT during test script execution. The test script calls external routines, provided by the tester, for transmitting and receiving protocol messages.

For CSMA/CD data links, the following external routines were defined:

- LL$OPEN_CSMA ( ) – Open a channel on the CSMA/CD controller.
- LL$CLOSE_CSMA ( ) – Close a previously opened CSMA/CD channel.
- LL$TRANSMIT_WAIT_CSMA ( ) – Transmit a packet on the CSMA/CD channel. The I/O operation completes synchronously; i.e., it returns to the caller only when the packet has been transmitted.
- LL$RECEIVE_WAIT_CSMA ( ) – Receive a packet on the CSMA/CD channel. The I/O operation completes synchronously. An optional timeout parameter is provided to terminate the operation if no packet is received in the time period.

These external routines interface the WAVE script (by means of the WAVE interpreter) to the device driver that actually performs the transmit and receive operations.

```
!
! MACRO: LL$M$TRANSMIT_WAIT_CSMA
!
!       Transmit a message on the specified csma/cd channel. Synchronous
!       completion.
!
! Inputs:
!       buff            Storage location containing a message to transmit.
!       mesg_length     Length of the message to be transmitted.
!       channel         Channel number assigned to this open request.
!
! Outputs:
!       status          Status return from the external routine call.
!
! Defaults:
!       channel         1
!       status          (macro supplied variable)
!
!
#macro LL$M$TRANSMIT_WAIT_CSMA < buff, mesg_length,
                                 channel=1, status=LL$G_STAT >

    #ifnact buff
        #error 'Macro parameter "buff" must be supplied'
    #endif
    #ifnact mesg_length
        #error 'Macro parameter "mesg_length" must be supplied'
    #endif

    external LL$TRANSMIT_WAIT_CSMA (buff, mesg_length, channel, status)

#endmacro
```

Figure 4 External Routine Call Macro

External Routines for Extended Capabilities

The routing tool takes advantage of the external routine capabilities of WAVE to add extensions to the WAVE language. The WAVE language does not provide any built-in capabilities for printing the error message text associated with error codes returned by external routines. This capability is provided by the following external routine:

```
LL$GET_MESSAGE ( )
```

This routine, written in BLISS, accepts a parameter containing the error code returned by an external routine call. It then calls a VAX/VMS system service which associates the error code with the error message text. This text is returned to the calling script.

External Routine Call Macros

The routing tool provides WAVE language macros that simplify the calling of external routines by providing default values for external routine parameters. Figure 4 shows the macro LLMTRANSMIT_WAIT_CSMA which is used to call the LL$TRANSMIT_WAIT_CSMA external routine.

In this macro definition we see that a default value is provided for the channel parameter as well as the status parameter. Required parameters are also checked using a macro lexical function. If a required parameter is not present in the macro call, a compilation error is generated.

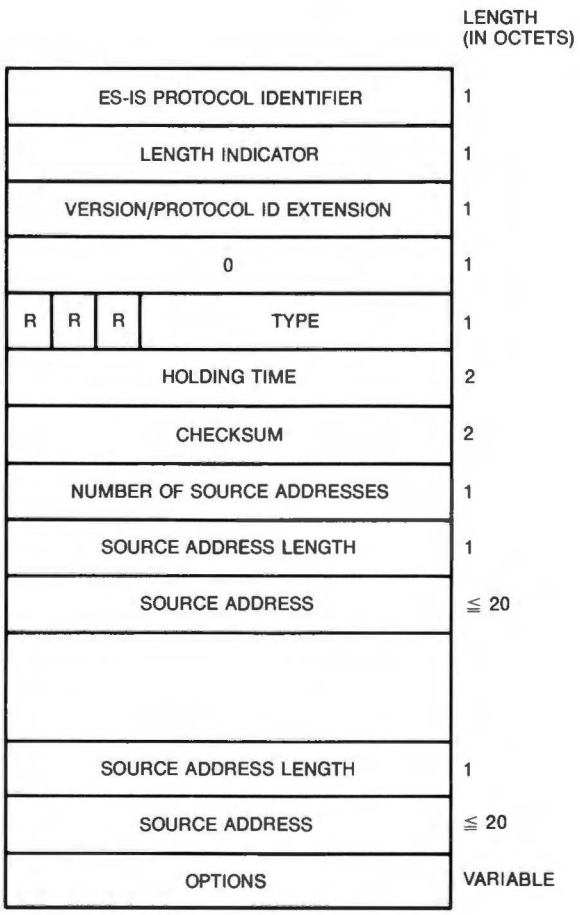


Figure 5 Format of the End System Hello PDU

Protocol Field Definitions

The routing tool provides WAVE language field definitions for the fields of each routing PDU. Figure 5 depicts the format of the End System Hello PDU. Figure 6 shows some of the field definitions provided for this PDU.

In the fields for this PDU, we see the flexibility of the field declaration statement put to use in defining the V$F_HOLDINGTIME field. The starting position of this field is determined by the expression ROC$G_BASE+5.

Error Return Codes

The routing tool provides WAVE language constant declarations which the script developer can use for checking the return status from external routine calls. The following example shows some of the constants that are defined for routing.

```
!
! File:     ERROR V02.10
! Created: 25-NOV-1988 14:26
!
constant ROC$_NORMAL  = %hl'08038019'
constant ROC$_BUFOVR  = %hl'08038024'
constant ROC$_INVCHN  = %hl'08038064'
constant ROC$_TIMOUT  = %hl'080380F4'
```

The script developer uses the include file capability of the WAVE compiler to make these constant definitions available to the script. Note that the error return codes follow the standard VAX/VMS convention for system error codes.

Sample Routing Test Script

In this section we present a sample routing test script for the ISO 9542 protocol. This script is shown in Figure 7. The purpose of the test script is to verify that the IUT sends a valid End System Hello PDU. The End System Hello (ES Hello) message is transmitted periodically by DECnet Phase V "endnodes" to inform "routers" of their existence. On a LAN this message is sent to a special address, called a multicast address, which addresses all routers on the LAN.

The script begins by using the include file capability of the WAVE compiler to read in error code

```
!
! Fields for ISO 9542 - End System Hello PDU
!
field V$F_PROTID              (ROC$G_BASE+0 : 1)
field V$F_LENGTH              (ROC$G_BASE+1 : 1)
field V$F_VERSION             (ROC$G_BASE+2 : 1)
field V$F_LIFERESV            (ROC$G_BASE+3 : 1)
field V$F_TYPE                (ROC$G_BASE+4,0 : 0,5)
field V$F_TYPERESV3           (ROC$G_BASE+4,5 : 0,3)
field V$F_HOLDINGTIME         (ROC$G_BASE+5 : 2)
field V$F_D_CHECKSUM          (ROC$G_BASE+7 : 2)
field   V$F_D_CHECKSUM_LOW    (ROC$G_BASE+7 : 1)
field   V$F_D_CHECKSUM_HIGH   (ROC$G_BASE+8 : 1)
```

Figure 6 Fields for the End System Hello PDU

values, PDU field declarations, and macros for calling external routines. Next are storage declarations.

The executable portion of the script starts with the "begin" statement. The script first opens a channel on the device that will be used to receive messages transmitted on the LAN. With the channel successfully open, a timer limiting the total run-time is started.

```
timeoper(start_down, MAX_WAIT_TIMER)
```

The statement immediately preceding "timeoper" illustrates a capability of WAVE that we have not previously discussed. The statement

```
conditoper(create, timexp, MAX_WAIT_HANDLER)
```

establishes MAX_WAIT_HANDLER as a WAVE condition handler routine.[3] WAVE condition handlers process asynchronous conditions that may occur during script execution. In the sample script, the routine MAX_WAIT_HANDLER is used to handle the WAVE timer expiration condition "timexp." This routine is called by the interpreter when a timer expires (counts down to zero). The statements that make up the handler are found at the end of Figure 7.

```
MAX_WAIT_HANDLER:
    put(terminal) "Test timer expired":ascii
    stop(inconclusive)
    return
```

The code fragment above shows the action taken by the script on timer expiration: An error message is displayed and script execution is terminated. The "return" statement is always the last statement in a condition handler routine or subroutine.

With the timeout condition handler established and the test timer started, the script proceeds to the main body of the test. There, an external routine is called (via a macro). This external routine attempts to receive a message transmitted on the LAN.

```
LL$M$RECEIVE_WAIT_CSMA <buff=RBUFF,
                        mesg_length=RLENGTH,
                        time_out=MAX_WAIT,
                        status=STAT>
```

If a message is successfully received, the script checks to ensure that it is the type of message expected by calling the subroutine CHECK_ESHELLO.

```
gosub CHECK_ESHELLO
```

If the subroutine indicates that an ES Hello was received, the test is completed by performing further checks on the validity of the PDU format. If the script does not receive a correct message, it continues in the loop where it attempts to receive another message from the LAN. If the message being sought never arrives, the test timer MAX_WAIT_TIMER will expire, causing the timeout condition handler to execute, as previously described.

Summary

The design of the WAVE tools base solves the unique problems of testing sophisticated protocols. The tool developer can quickly create a product quality test tool by adding a few simple components to the framework provided by WAVE. NAC Conformance Engineering Group has found WAVE to be the key element of its success in building conformance test tools for DECnet Phase V protocols and related distributed applications.

Acknowledgments

We would like to express our appreciation to the past and present members of the NAC Conformance Engineering Group who helped design and implement WAVE, making it the excellent tool it is today. Further, we would like to thank our managers past and present, Jane Morency and Paul Keresey, who had

```
#include 'error.inc'              ! Include external routine error codes
#include 'message.inc'            ! Include PDU message field definitions
#include 'macro.inc'              ! Include external routine call macros

variable STAT(4)          variable CHAN(2)          variable DEVICE(4)
variable RLENGTH(2)       variable MAX_WAIT(2)      variable GOT_ETHER(2)
variable GOT_802_ISO(2)   variable GOT_ESHELLO(2)
buffer RBUFF(1600)        timer MAX_WAIT_TIMER

begin
    DEVICE = "XQA0"
    LL$M$OPEN_CSMA <device_name=DEVICE, status=STAT>
    if not STAT then
        LL$M$PRINT_MESSAGE <code=STAT>
        stop(fail3)
    endif

    MAX_WAIT = ROC$K_PollESHelloRate * ROC$K_HoldingMultiplier
    MAX_WAIT_TIMER = MAX_WAIT * 100
    conditoper(create, timexp, MAX_WAIT_HANDLER)
    timeoper(start_down, MAX_WAIT_TIMER)

    loop
        LL$M$RECEIVE_WAIT_CSMA <buff=RBUFF, mesg_length=RLENGTH,
                                time_out=MAX_WAIT, status=STAT>
        if not STAT then
            LL$M$PRINT_MESSAGE <code=STAT>
            stop(inconclusive)
        endif

        gosub CHECK_ESHELLO
        if GOT_ESHELLO then
            timeoper(stop, MAX_WAIT_TIMER)
            if (RBUFF.LL$F_DESTINATION eql %hb'09,00,2B,00,00,05') and
               (RBUFF.V$F_VERSION eql 1) and (RBUFF.V$F_D_CHECKSUM eql 0) then
                exitloop
            else
                stop(fail1)
            endif
        endif
    endloop

    LL$M$CLOSE_CSMA < >
    stop(pass)

! Routine: CHECK_ESHELLO
!
CHECK_ESHELLO:
    GOT_ESHELLO = FALSE
    RTG$M$CHECK_802_ISO < buff=RBUFF, snap=GOT_802_ISO >
    if GOT_802_ISO then
        if (RBUFF.V$F_PROTID eql ROC$K_V_ESISProtocolIdentifier) and
                                        (RBUFF.V$F_TYPE eql 2) then
            GOT_ESHELLO = TRUE
        endif
    endif
    return

!Routine: MAX_WAIT_HANDLER
!
MAX_WAIT_HANDLER:
    put(terminal) "Test timer expired":ascii
    stop(inconclusive)
    return
end
```

Figure 7 Sample Routing Test Script

the confidence and foresight to give us the space and time we needed to try our ideas and invent something very unique.

Notes

1. The Tree and Tabular Combined Notation (TTCN), which is today described in ISO DP 9646-3, was still in its infancy at the time and was not targeted to be machine processable. TTCN is starting to be used today within ISO as a standard test notation, but it is neither easy to learn, easy to use, familiar looking, nor sufficiently flexible for our needs. TTCN is also primarily an abstract description language, whereas WAVE is an executable programming language. Hence, current work on TTCN does not eliminate the need for WAVE.
2. Although dynamic fielding would certainly be possible in a fully compiled language, the required code generation and run-time support would be very difficult to implement. The choice of an interpretive approach made implementing this facility quite straightforward.
3. WAVE condition handler routines provide an asynchronous event handling capability. They should not be confused with condition handlers of the VAX/VMS operating system.

Eugene Finkelstein
Richard A. Grawin

Performance Evaluation of Distributed Applications and Services in the DECnet Environment

Performance evaluation is critical in the design of distributed systems as well as for purposes of capacity planning. The Distributed System Technical Evaluation Group has developed a methodology for distributed system performance evaluation at the system level and has applied it in the DECnet environment. The core of the methodology is the systematic use of an empirical/analytical approach in which measurements and modeling are closely coupled. This paper focuses on the empirical component and provides details on experiment complexity reduction, an instrumented test-bed, and tools for data collection, reduction, and analysis. The case studies of VAX DNS, VAX DFS, and RSM, from which the methodology evolved, present performance evaluation results of product qualification testing. VAX/VMS Mail and VAX/VMS Copy case studies, to which the methodology was applied, include network file transfer performance and capacity planning data.

The growing demand for distributed computing has resulted in the emergence of large-scale, network-based distributed systems. These distributed environments offer customers the opportunity to combine and share limited resources for cost-effective operation.

Within the same configuration, various products must coexist without excessively interfering with one another and perform in a way transparent to the user. However, even when different products interoperate successfully within a certain configuration, the entire distributed system behavior under variable loads is still an issue. Therefore, the ability to evaluate distributed system bounds and bottlenecks with respect to alternative configurations and workloads is needed.[1,2]

In this paper, we present the methods and tools being developed and used by the Distributed Systems Technical Evaluation Group (DSTEG) in assessing the performance of DECnet-based distributed systems. Performance evaluation relative to product interoperability testing is one of the areas where the methodology may be applied. Here, we add new products to a distributed system test-bed and examine their impact on the performance of that system. Distributed system performance evaluation for the purposes of system design and capacity planning is another area where we are currently using the methodology. Here, we look at the means used to forecast the behavior of a distributed system under a variety of workloads, with configuration changes, or given hardware and software component additions or replacements. Both kinds of measurement studies have evolved the methods and tools described in this paper.

Our methodology is a unique merger of empirical and analytical techniques. Although we demonstrate how modeling and measurements are closely coupled in our approach, this paper focuses on the empirical component. This component includes a complexity reduction of experimental design, a controlled experiment environment, and data collection and analysis tools. Following the discussion of the methodology are case studies. The Distributed Name Service (DNS), the Distributed File Service (DFS), and the Remote System Manager (RSM) studies, from which the methodology and experiment environment have evolved, illustrate some performance results relative to the qualification testing of these products in the distributed system environment. More recent evaluations of local area network (LAN)-based distributed system performance use VAX/VMS Mail and Copy utilities as the workload. These evaluations demonstrate the successful use of the methodology. Finally, we deliberate upon the possible uses of the findings given in this paper.

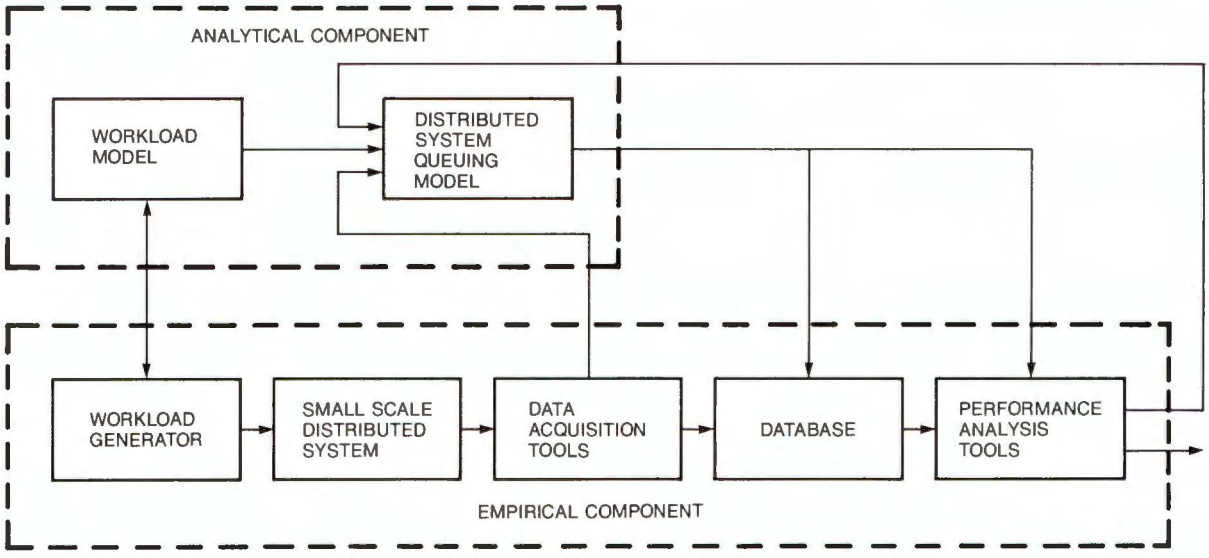


Figure 1 Distributed System Performance Evaluation Approach

Methodology

A distributed system performance evaluation methodology has been developed which relies on closely coupled empirical and analytical components. To encompass the wide scope of the problem and to provide accurate forecasting capabilities, modeling augments the measurement capabilities. In this section, we describe the dynamic interaction between measurements and modeling in the course of distributed system performance analysis. However, the focus of our discussion is the empirical component. We describe the means of complexity reduction, the test-bed features and tool capabilities, and specifics of the experimental design and data collection, reduction, and analysis.

The methodology can be applied to performance evaluation relative to product interoperability, performance sensitivity to dependent parameters, bottleneck analysis, boundary condition (failure) detection, and system design alternative selection.

The key performance metrics examined are user-perceived delays and network and nodal resource utilization. These user response times are decomposed when appropriate. For example, times related to DECnet session establishment and duration are obtained when session-oriented applications are evaluated. The primary resources monitored are the Ethernet bus, CPUs, disks, and memory. Component throughput levels and service times are obtained in product or application performance evaluations. Effective (data related) network utilization is examined through analysis of DECnet statistics. One goal in evaluating these metrics is to understand the trade-offs in minimizing user delays while maximizing shared resource utilizations.

Combined Empirical/Analytical Approach

The components of the distributed system performance evaluation methodology are illustrated in Figure 1. A dynamic interaction between an analytical model and the experimental test-bed is the core of this methodology. First, the fundamental ingredients of the small-scale distributed system (realizable in a laboratory) are measured and understood. Then, the corresponding queuing models are constructed and validated. Finally, these models are expanded to evaluate large-scale distributed systems for which performance evaluation of the entire system by measurements is infeasible. Use of the learning, constantly updated model significantly enhances performance evaluation capabilities offered by the combined empirical/analytical approach.

The empirical component requires workload generation capabilities and an instrumented test-bed capable of collecting the appropriate performance data. A data referencing and archival system and a set of performance analysis tools are used to expedite the analysis phase. The supporting tools and test-bed features have evolved through a series of case studies. Measurements take place within case studies not only to empirically evaluate distributed system performance characteristics but also to help construct, parameterize, and validate distributed system models. This feedback from the empirical to

the analytical component of this methodology is depicted in Figure 1.

The modeling component of this methodology, not detailed in this paper, is driven by distributed system performance evaluation needs. Modeling requires the abstraction of both the distributed system and the workload being investigated. Both the system and the workload can be simplified in the distributed system models. This simplification and an efficient solution algorithm based on approximation techniques allow evaluation of very large distributed systems in a timely manner.[3] In conjunction with the analytical modeling, complexity reduction techniques allow for the development of effective methods of distributed system performance evaluation. These techniques, described in the following section, significantly reduce the time needed to accomplish the evaluation.

Complexity Reduction

With the large number of hardware and software products, applications, and associated parameters of interest, a distributed system performance study can potentially require a very large experiment set. The methodology attempts to reduce the number of experiments by making use of commonalities between applications or products and of experiment design decomposition techniques.

As it follows from information theory, when dealing with two events with some similarities, it should take less effort to learn about one of them by using the knowledge already acquired about the other one.[4] In moving from one study to a subsequent one, any commonality between the two can be exploited, and hence, redundant or unnecessary experiments can be eliminated. For example, in evaluating two distributed system products with similar resource demands, one would be able to predict critical workload levels associated with the second product after empirically determining them for the first.

Experiment design decomposition provides another way of reducing the experiment set when many parameters are involved.[5] By understanding which variables of interest are dependent and independent (orthogonal), one can decompose the set of experiments to mutually exclusive subsets. One can then make decisions within experiment subsets about reducing the number of necessary experiments.

For example, the number of experiments needed in evaluating distributed system performance can increase combinatorially with respect to the number of initiator/target node types. Evaluating the distributed application performance for two node types implies four different initiator–target combinations. One approach to minimizing the number of experiments is to choose the best-case and worst-case node pairs and evaluate them, knowing that the other cases fall in between. Another approach is to measure performance for several node types and predict the performance for others by modeling. Similarly for application parametric analysis, one may have to rank the parameters and choose to evaluate the application's performance with respect to the most significant parameters or choose to evaluate performance under reasonable upper and lower parameter settings only.

Evaluation Process

The operational process put in place to evaluate distributed system performance is shown in Figure 2. Market driven performance issues and evaluation needs define why the study should be undertaken. Market concerns and needs are mapped, through the design center, into specific project requirements. The design center describes a distributed system environment — products, systems, and workloads — representative of a particular class of network users. This mapping keeps the study relevant for a class of Digital customers rather than specific to any single customer.

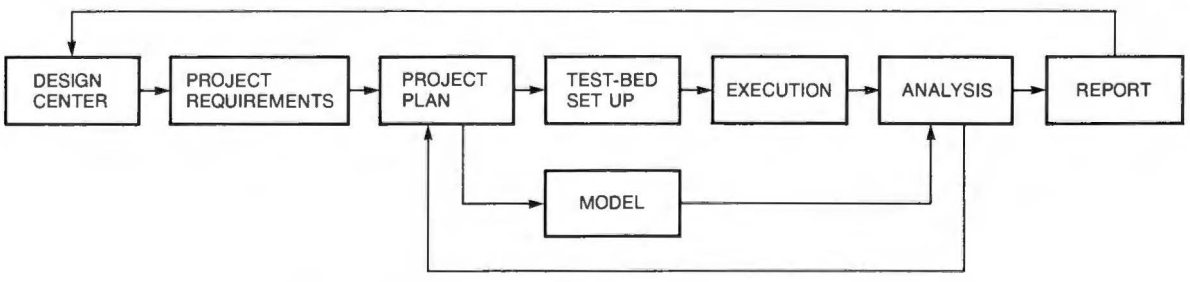


Figure 2 Distributed System Performance Evaluation Process

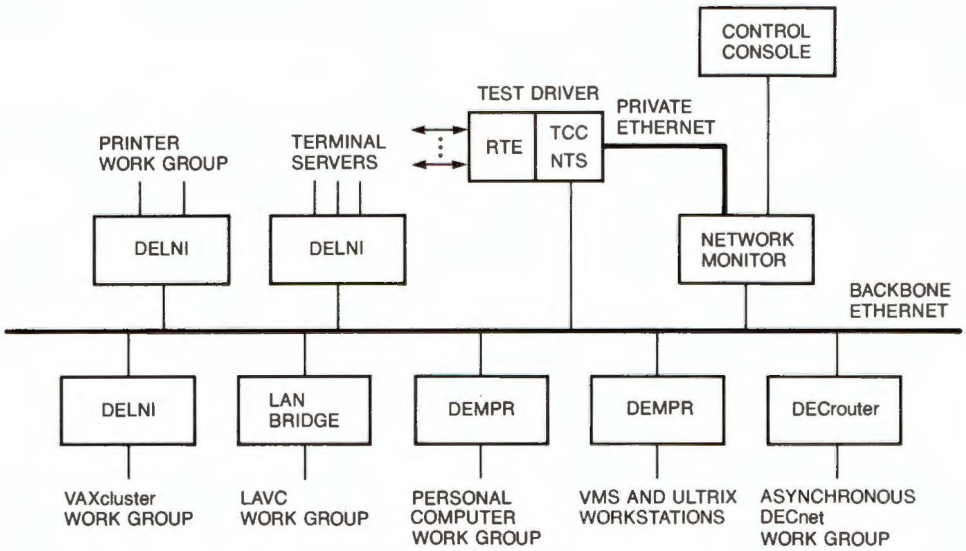
The project plan explains how the project requirements will be met and details proposed strategies (measurements and models), experiment designs, resource needs, and schedules. Experiment and/or model execution and analysis then take place. A key element of the process is the feedback loop from the analysis phase back to the planning phase. Performance studies are very unpredictable, and through the feedback loop, unnecessary experiments can be eliminated and new experiments can be defined.

Finally, a report is generated that provides data and answers questions posed by the project requirements. Based on the findings, recommendations can be made relative to the products and configurations investigated. As depicted in Figure 2, this information can also bring about modifications to the design center.

Experiment Environment

Instrumented Test-bed The laboratory currently facilitates both interoperability and detailed performance analysis. Depicted in Figure 3, the test-bed is currently a local LAN environment which will evolve towards the extended LAN (XLAN) and wide area network (WAN) environments. The following test-bed features exist:

- A very heterogeneous distributed system with several classes of systems under test (SUTs): Ethernet and asynchronously connected standalone systems running VMS and ULTRIX, VAXcluster and Local Area VAXcluster (LAVc) members running VMS, and personal computers running MS-DOS. Several systems of the same type are clones of one another to allow exact comparative analysis between systems and to reduce the test-bed system management overhead. Network-related components include terminal servers, printers, bridges, and routers.
- A workload generator which provides terminal user emulation at the interactive command level; semiautomatic and interactive script generation capabilities; event arrival definitions for exponential, uniform, normal, constant, or general distributions; and user-perceived task completion time measurements.



KEY:

RTE - REMOTE TERMINAL EMULATOR

TCC - TEST CONTROL CENTER

NTS - NETWORK TIME SYNCHRONIZER

DELNI - LOCAL NETWORK INTERCONNECT

DEMPR - MULTIPORT REPEATER

LAVC - LOCAL AREA VAXcluster

Figure 3 Heterogeneous Distributed System Test-bed

- A test control program that runs environment verification and initialization procedures at the beginning of an experiment and clean-up procedures at the end.
- Flexibility built into the hardware configuration and the test driver to allow easy redefinition of the set of active SUTs.
- A multiwindowed control console that centralizes control over experiment execution, including workload generation and data collection tools, and provides real-time monitoring of the progress of an experiment. Nonintrusive control of the test driver and network monitoring systems is accomplished over a LAN isolated from the SUTs.
- A utility to synchronize the time on all nodes and data collection systems.
- A performance measurement tool set that monitors CPU, disk, memory, and network resources.

Data Collection Time-stamped and sampled data are collected from all SUTs and the Ethernet. On all VMS systems, VAX/VMS System Performance Monitor (SPM) collects data on CPU, disk, memory, lock, and other nodal resource usage for both the whole system and on a per-process basis.[6] The VAX/VMS Accounting utility provides image and process data at each node. Network Control Program (NCP) counters are used to quantify the amount of network I/O sent and received by the active nodes and to indicate network error conditions. Disk and file system activities are monitored by a prototype measurement tool.

Two monitors are used on the Ethernet bus. The LAN Traffic Monitor (LTM) provides statistics of Ethernet data link protocol and packet size distributions, utilization and throughput, and top network traffic sources.[7] A second prototype monitoring tool provides Ethernet and DECnet information including data link, network, and transport layer packet counts. DECnet session layer data packet size and message size distributions are also obtained. Additional DECnet session level data available is source and target node identification, session establishment and termination time-stamps, DECnet session object number, and control and data packet and byte counts associated with each session.

Experiment Design

Some experiments are designed to provide an understanding of the performance impact (interoperability) of a newly introduced product. Other experiments address specific distributed system configuration, product, or application performance issues. The issues must be specific, since an exhaustive performance evaluation over multiple hardware platforms and multiple application parameters results in an excessively large experiment set. One means of limiting the experiment design space is to work within the hardware, software, and workload constraints of a defined design center, described in the earlier section Evaluation Process.

Product interoperability investigation is a two-stage process. In the first stage, the product under test is evaluated in a controlled, isolated environment. No other products are running in the distributed system. This forms a "base experiment." This experiment determines the readiness of the product to enter the second stage. In the second stage, the product is evaluated in a controlled, mixed environment. Users are exercising other distributed services and applications along with the product under test.

Each experiment is made up of test scripts which emulate users doing a predetermined amount of work. These network-oriented workloads include VMS Mail, Copy, and Set Host utilities as well as VAX Notes, VAX VTX, VMS File Services for MS-DOS, VAX DFS, VAX DNS, VAX DQS, and VAX RSM layered products. Users are categorized as performing a particular type of work. For example, some users may be defined as clerical, some as engineers, and others as system managers. Each type of user can be expected to have activities that are representative of the type of work the user is defined to do. Each user is assigned a script which is of the proper category and which contains functions associated with that type of user. Again, much of this workload is defined from design center specifications.

Key metrics associated with the product's performance and with distributed system performance are chosen for review. Typical metrics are nodal metrics such as CPU, memory, and disk usage for each SUT and network metrics such as Ethernet protocol distribution and throughput levels. The metrics quantify how well the product under test performs in the distributed system.

Differences in the values of these metrics before and after introducing the product under test yield the effect of adding that product to the distributed system. These differences are analyzed further to determine the effect on other network configurations.

Within the combined empirical/analytical approach to distributed systems performance evaluation, experiments are designed to (1) produce direct, empirical proof of distributed system performance, (2) contribute to the distributed system

model parameters, and (3) validate the modeling effort. The following general kinds of experiments are run:

- No users are active. This experiment quantifies the quiescent system activities, e.g., DECnet and LAVc protocol activities and VMS background activities. It is also used to quantify the intrusiveness of data collection tools.
- A single active user repetitively performs a single task or application. This workload can quantify the optimal user-perceived performance when no contention for resources is taking place. The workload also quantifies the user's per-application resource usage (CPU, disk, memory, Ethernet) for bottleneck analysis and model service time purposes. This simple workload is useful in application parametric behavior evaluation.
- Multiple active users repetitively perform a single application or a mixed workload. This experiment is run to empirically determine if user-perceived performance and resource usage is linear with respect to the number of active users. The experiment can also be used to investigate task or application boundary conditions, effects on bottleneck devices, and failure modes with respect to the workload. Finally, it can validate the output of a corresponding queuing network model.

Data Reduction and Analysis

Lots of data is collected from multiple tools on multiple systems during an experiment. This data needs to be reduced, merged, and analyzed in a timely manner. A set of VAX/VMS command files has been written to automate the data reduction and analysis process as much as possible. These command files first call upon programs to generate ASCII report files from binary data files. Secondly, the command files call upon SAS statistical analysis routines to reduce and analyze the data. The SAS routines generate both report files and data sets used in further analysis.

A first step in data reduction is to analyze the data for a selected subset of the SUTs. SAS routines then reduce this data by extracting from several files two set of data: (1) the key performance variables, and (2) the appropriate samples based on a time window of interest. These key metrics relate to user-perceived completion times, DECnet session times, and CPU, disk, memory, and Ethernet resource usage. The outcome of all the data reduction is an experiment summary file automatically generated after each experiment run. These summary reports allow the analyst to quickly determine if experiments ran successfully and whether or not any performance trends are evident across experiments.

The most time-consuming part of a study is typically the analysis phase. The experiment summary and the SAS analysis routines help minimize this time. Routines report the minimum, maximum, average, and variance of the key performance metrics. Associated with this, peak periods of resource usage can be located for transient analysis. For bottleneck analysis, several data sets are merged to derive aggregate per-application resource service times.[8] Device visit counts, service times, and network transfer size statistics are obtained for model input.

Case Studies — Evolving the Methodology

We briefly discuss the results of our performance evaluation with respect to qualification testing of three products — DNS, DFS, and RSM. These studies helped evolve the methodology and its associated experiment environment and tool set. Experiment designs and results are summarized to demonstrate the divergent kinds of experiments performed and data examined in analyzing these various products. Actual performance numbers associated with DNS and DFS have not been included since more recent findings are given in this issue of the *Digital Technical Journal*.[9] From these studies, we saw the need for a methodology, an experiment environment, and a data collection and analysis tool set that are flexible enough for effective use across different kinds of studies.

Distributed Name Service

Our main goal was to ensure that DNS version 1.0 was a quality product ready for customer release. We evaluated its functionality, manageability, reliability, performance, and documentation. Measured performance was compared with product requirements provided by the DNS Development Group. The DNS use of distributed system resources with multiple DNS servers and clients active was also quantified.

Three different test environments were used during the DNS study.

- A minimal configuration consisted of two MicroVAX systems connected by means of ThinWire Ethernet. Both systems ran VMS and both were used as DNS servers and clients. A flat DNS namespace with two clearinghouses in the root directory was defined. This simple configuration was useful in familiarizing ourselves with DNS and performing easily controlled experiments.

- A more complicated configuration consisted of 13 MicroVAX systems connected by a ThinWire Ethernet. Eleven of the systems ran the VMS operating system and were configured as both DNS servers and clients. The two remaining systems ran ULTRIX and DNS client software. A hierarchical DNS namespace was defined which contained 11 clearinghouses. Convergence parameters of all namespace directories were set to high. This DNS configuration was highly distributed and was used to stress the distributed properties of DNS, such as name server replication.[10]
- Thirdly, tests were run using a VAXcluster system. The cluster consisted of a VAX-11/785 system and a VAX 8600 as the DNS server, and two MicroVAX II systems as DNS clients — one running VMS and the other running ULTRIX. Here, timing tests were conducted by using the VMS name service clerk jacket routines to determine the response times of a subset of DNS functions. Random requests were made for each call, using namespace directories from 1 to 10 levels deep. We found that the response times were within the stated performance goals of DNS version 1.0.

For most of the evaluation, a test script was used which was designed to exercise the DNS transaction agent and background process. This script performed create, modify, and read operations within each namespace. We ran the script on both VMS and ULTRIX clients with the three configurations isolated and connected.

The DNS study demonstrated a need to design a flexible, heterogeneous test-bed where different groups of systems could easily be isolated from one another or combined to form a single test environment. Evaluation of DNS background activities that propagate throughout the distributed system demonstrated the need for capabilities such as system time synchronization and enhanced Ethernet (DECnet) monitoring.

Distributed File Service

The goal of DFS version 1.0 testing was to determine how DFS performed as the number of active DFS clients increased. The workload chosen was a file copy operation that placed a steady load on the DFS server. Experiments were conducted using two types of servers — a MicroVAX 3500 system and a MicroVAX II System 5. These are two of the smallest VAX systems to support RA disk drives and were chosen to demonstrate low-end DFS server capacities. Up to 22 MicroVAX II systems were used as DFS clients.

The experiment set involved four variables: DFS server type, file size, number of copy transactions per experiment, and number of active DFS clients. A power metric was defined to quantify the DFS server capabilities: Power = data rate × transaction rate.

This single metric is useful in comparing the effective work done by the DFS server when handling different file sizes. Large file transfers cause a higher data rate but a lower transaction rate, whereas small file transfers cause the opposite effects.

In the low-end server configurations studied, the results indicated that the DFS server CPU was the bottleneck. Again, measured results are not presented here; more recent findings on DFS performance are provided elsewhere in this issue.[9]

Besides the DFS performance evaluation results, this study gave us experience in selection of parameters, experiment design, and control of a moderately sized test-bed. The power metric defined and used in this study may prove useful in characterizing performance for many distributed system products and services.

Remote System Manager

We investigated the manageability, security, interoperability, and performance characteristics of RSM version 2.0.[11] Testing was performed in a configuration specific to the expanded capabilities of RSM version 2.0. This configuration included a VAXstation 2000 workstation and an ULTRIX node as RSM clients, and a MicroVAX II system, a VAXcluster system, and an LAVc boot member as RSM servers. A small WAN was also used during the RSM evaluation.

Our performance analysis of RSM measured its system backup and initial system load (ISL) functions. The goal was to determine the performance effects of an increasing number of RSM backups and ISLs on distributed system performance. SPM monitored the RSM server resource usage and LTM monitored the Ethernet. These experiments were run with RSM version 2.0, VMS version 4.6, ULTRIX version 2.0, and DNS version 1.0.

The following conclusions were reached on RSM version 2.0 performance:

- When multiple client backups to a RSM server are performed, the bottleneck is typically the disk on the server.
- ISLs place a heavier load on the server CPU than do backups.
- An ULTRIX ISL is faster and takes up less CPU on the server than a VMS ISL. With a VAXstation 2000 workstation as the RSM server, an ISL of a 74,000 block ULTRIX VAXstation II system takes 26:36 minutes; whereas an ISL of a 53,298 block VMS VAXstation II system takes 40:25 minutes.

- The RSM server CPU time and the Ethernet utilization increase linearly with the number of clients being simultaneously loaded. However, the total elapsed time associated with multiple ISLs only increases marginally.

  Also, the increase in elapsed time is less than the increase in RSM server CPU time. With a VAXstation 3000 as the RSM server, loading two VMS VAXstation II clients takes only 3 minutes more of elapsed time over the 40:25 minutes to load a single client. At the same time, the RSM server CPU time increased from 6.5 to 13 minutes.

The RSM study demonstrated the need to quantify or at least understand the product's resource usage of all potential bottleneck components. The marginal elapsed time increase when loading multiple RSM clients implies that there is a significant amount of concurrent activity taking place on both systems and on the network during an ISL. Understanding how to measure and characterize distributed system concurrency has become an important factor in our methodology. Finally, the need to understand ULTRIX system behavior with respect to distributed system products was also evident from this study.

Studies Implementing the Methodology — VAX/VMS Mail and VAX/VMS Copy

Using the methodology described in this paper, two studies were conducted to examine the performance of network transfers using the VAX/VMS Mail and VAX/VMS Copy utilities. The mail evaluation was a pilot study in implementing the methodology. One goal of the study was to test and improve the methodology and tool set. A second goal was to evaluate the performance of the VAX/VMS Mail utility with respect to different sender and receiver node types, file sizes, and mail load levels.

In our evaluation of the VAX/VMS Copy utility in the LAN environment, we built upon the mail study findings. Besides assessing the performance characteristics of VAX/VMS Copy, we also

- Examined block versus record modes of network file transfer
- Investigated the methodology of capitalizing on similarities between applications of interest to understand how this ability can improve the experimental process
- Determined the adequacies of the measurement tools used in capturing performance data for input into a distributed systems modeling effort

Experiments were designed to scale the mail and copy workloads. Application performance under light and heavy loads, including application failure, was analyzed. Parameters investigated included transfer size, number of destinations, number of active users, and workstation and diskless LAVc satellite node types. Performance metrics of interest were user-perceived completion times, resource utilization and throughput levels, and DECnet session characteristics — connection and duration timings and transfer byte and frame counts.

Both studies used a LAN test-bed consisting of VAXstation II standalone workstations and VAXstation II LAVc satellites. The LAVc boot member was a MicroVAX/KDA with a single RA81 disk. All systems were running VMS version 4.7 with default system and user parameters and quota settings maintained.

Mail and Copy Design Differences Although these two applications are at times used interchangeably to transfer files across DECnet-based networks, their differences in design and implementation quite strongly affect their performance and resource demands. Copy relies on VMS Record Management Services (RMS) to interface to the network (transparent DECnet), whereas mail places the related system calls to the VMS QIO interface (nontransparent DECnet). This intuitively tells us that mail may reduce some network interfacing costs in establishing DECnet sessions and performing network I/O. However, the measurement results demonstrate that the access, authentification, and user environment initialization differences outweigh the interface differences in affecting session connect times. On the receiving node, the file access listener (FAL) associated with copy must deal with user, directory, and file permissions more often than does mail. The measurements support this fact with longer initial connect times for copy.

Copy by means of the data access protocol (DAP) and mail by means of the Mail-11 protocol can send a file in either block mode or record mode. For block mode transfers, the largest Ethernet frame size seen is typically either 576 or 1498 bytes (the DECnet-negotiated segment size), depending on whether or not the two nodes are adjacent. For record mode transfers, the largest Ethernet frame size is the smaller of the DECnet segment size or the file's record length. The maximum acceptable record lengths are 512 bytes for mail and 4156 bytes for copy.

Both applications attempt to use block mode whenever they can. Mail uses record mode when transferring a sequential file with stream record format. Copy uses record mode when a file append operation is taking place. Both copy and mail use record mode when nodes with different operating systems (e.g., MS-DOS) or file systems are communi-

cating. However, DECnet-ULTRIX supports block mode DAP transfers between ULTRIX and VMS systems. With only VMS systems in the test-bed, block mode (by means of copy) and record mode (by means of mail) were compared using stream-formatted files. Three different stream files were used:

- 200-by-50 file, or 200 bytes per record times 50 records
- 10-by-1000 file, or 10 bytes per record times 1000 records
- 10-by-1 file, or 10 bytes per record times 1 record

For the 200-by-50 and 10-by-1000 file transfers, copy's block mode transfer used the same number of large frames (2 at 1249 to 1280 bytes and 5 at 1473 to 1504 bytes) on both files; whereas mail's record mode transfer generated frames associated with the file's record length (50 at 225 to 256 bytes for the 200-by-50 file, 1007 at 33 to 64 bytes for the 10-by-1000 file).

Under light node and network loads, the major factor relating to the completion time differences between copy and mail was the DECnet session time. For large (10K byte) files, these session time differences can be attributed to block versus record mode file transfer times. For small (10 byte) files, the larger mail session times result from the additional mail application processing and disk I/O subsequent to the message reception and prior to the session termination.

With page faults, direct I/O, and buffered I/O levels for mail being 2 to 4 times those of similar copy executions, one expects to see those kinds of ratios in CPU service demands. This is indeed the case when one compares the CPU seconds measured by VMS accounting data for the copy and mail images. The larger mail paging rates and associated disk demands make mail's performance more dependent than copy upon disk load levels. The effects of these disk demand differences are accentuated when comparing mail and copy performance on diskless LAVc satellites.

User-perceived Performance and Delay Decomposition Figure 4 depicts the mail completion times for workstation and diskless LAVc satellite users. This figure demonstrates that sender and receiver node types can significantly affect distributed system performance. For all load levels, mail destined for the LAVc takes longer than mail destined for workstations. Also, with the low-end system LAVc used in this study, completion times of LAVc-destined mail grow significantly with incremental increases in the number of active users.

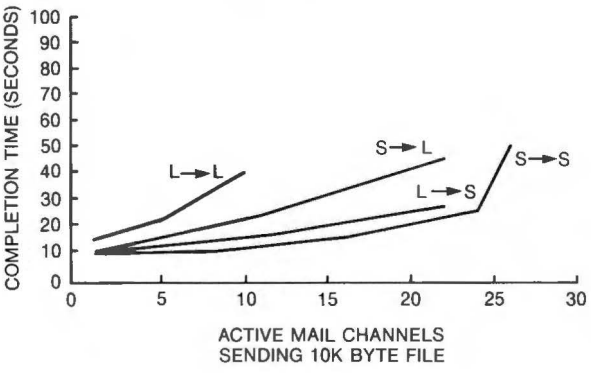


Figure 4 Mail Completion Time versus Load

As shown in Figures 5 and 6, the total elapsed time of DECnet session-oriented applications can be broken down into four components: (1) application start-up time, (2) DECnet session connect (link establishment) time, (3) DECnet session duration (includes file transfer) time, and (4) application run-down time. For single line commands like VMS Copy, users perceive only the total task completion times. For interactive VMS Mail, users can actually perceive all four of these delay components. The load on the source node most affects start-up and run-down times, whereas the load on the target node most affects session connect time. The file transfer part of the session duration time is affected by the loads on the Ethernet and on both sender and receiver nodes. Therefore, the file transfer is the most difficult element to model accurately. Only the file transfer time is significantly affected by network load.

For mail, the start-up times are approximately 1.4 seconds on a lightly loaded VAXstation standalone system. For a diskless LAVc satellite, these start-up times increase to 3.3 seconds. The large page fault and disk I/O activities associated with mail start-up contribute to the longer diskless LAVc satellite start-up time.

For copy, the start-up times are approximately 0.4 seconds on a lightly loaded VAXstation standalone system. For a diskless LAVc satellite, these start-up times increase to 1.4 seconds. The DCL parsing of the copy command line can require considerable processing and is included in the copy start-up times.

DECnet session connect times, given in Table 1, depend upon initial versus subsequent logical link establishments to the same destination user. The initial session connections take much longer than

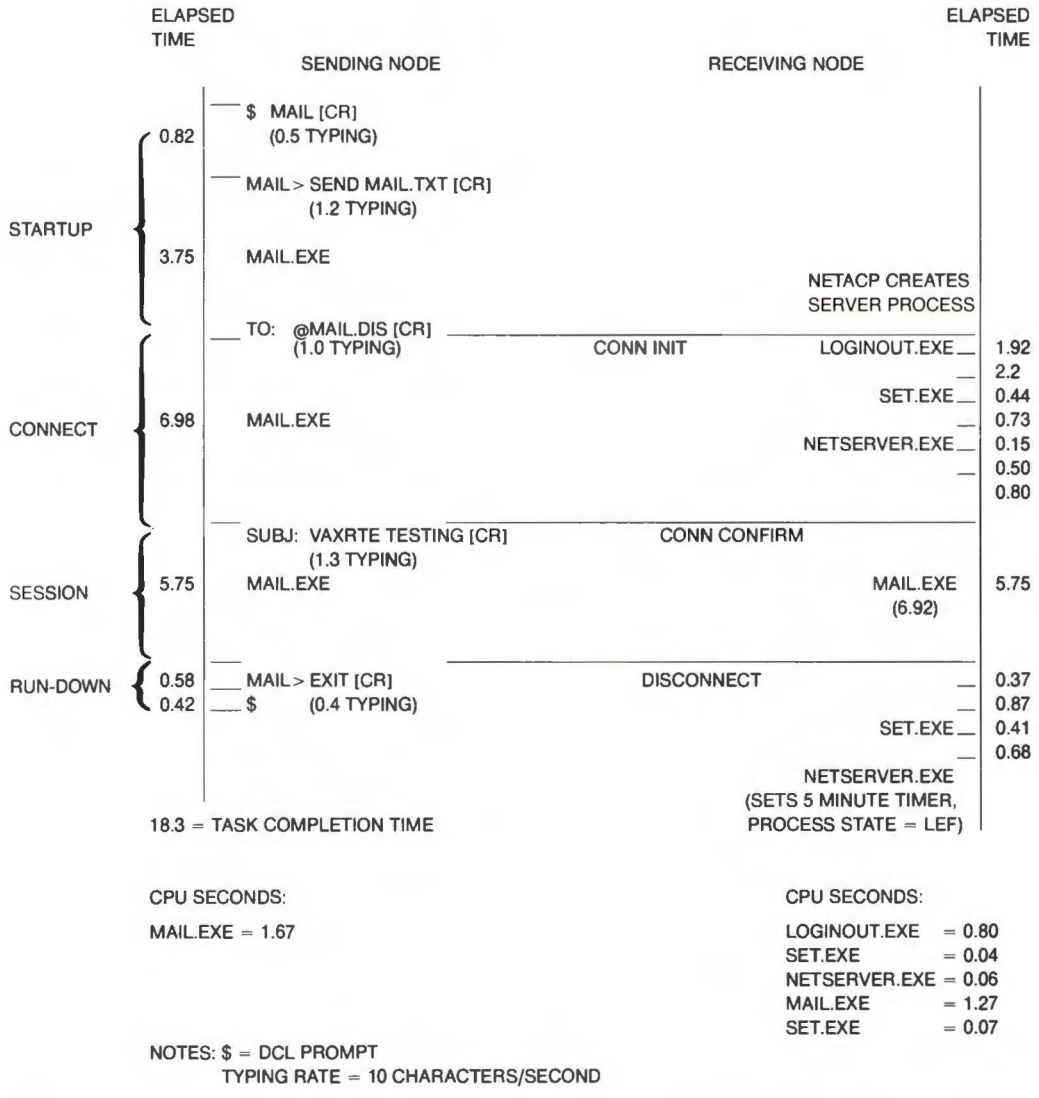


Figure 5 VAXstation-to-VAXstation Initial Mail of 200-by-50 File (All Time in Seconds)

subsequent ones to the same destination due to the receiver node's need first to create a network server process, and second, to run the user's authorization and initialization procedures.

Table 2 presents the delay decomposition under various loads for the transfer of a large record-oriented file by means of mail and copy. For mail, connect times (even for the given subsequent average) increase faster than session duration times. Delays for LAVc-destined mail increase faster than mail destined for workstations. Copy session establishments take about twice as long to the LAVc as to a standalone workstation. For the copy load levels used, a resource bottleneck did not occur; therefore, delays associated with the copy transfers do not degrade.

Resource Usage The per-application overhead figures for mail and copy are given in Tables 3 and 4. The CPU and disk service times given in seconds are derived using the Utilization Law.[8] These times can be used in bottleneck analysis and in modeling each application. Quantifying the number of I/O operations per application — direct I/O, buffered I/O, total disk I/O, and disk I/O relating to paging—along with the total number of page faults is useful in building more detailed models of each application.

For both the mail utility and the copy utility, the CPU time taken to send or receive files is not significantly affected by message length unless many records (more than 1,000) or blocks are transferred. Standalone systems require approximately half the CPU service time to mail or copy files than do LAVc

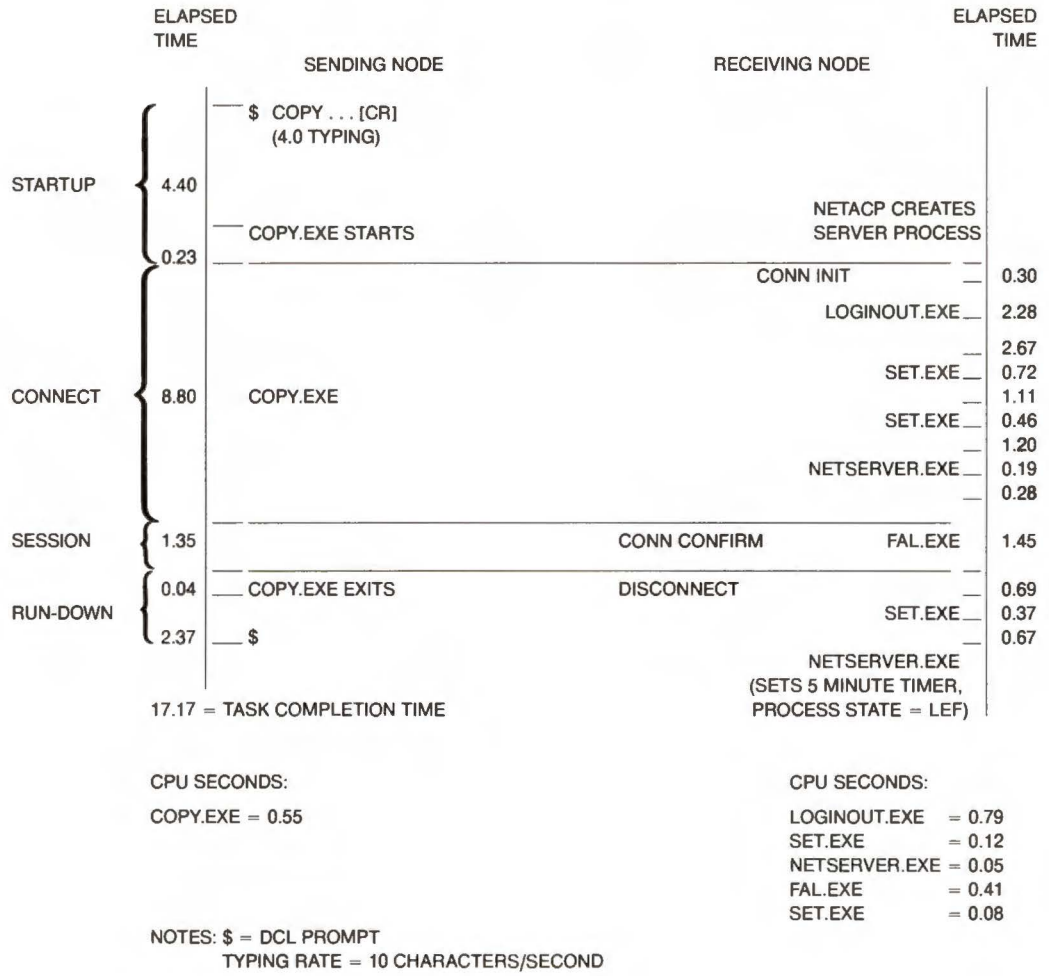


Figure 6 VAXstation-to-VAXstation Initial Copy of 200-by-50 File (All Time in Seconds)

satellites. Much of the mail CPU overhead is due to the high page-fault rates on both the sender and receiver. The buffered I/O counts directly correlate to the number of records (packets) transferred across the network, whereas the direct I/O and disk I/O counts are independent of it. The additive effect that active LAVc satellites have on boot member resource usage is apparent from the given boot member receiving costs.

Table 5 provides a breakdown of file system activity for the start-up, connect, session, and run-down phases of each application. For mail, the numbers of needed files, libraries, images, etc. (i.e., the working set) directly influence application start-up and run-down times. On the receiving node, a large portion of the disk I/O can be attributed to paging related to files associated with user authorization. Also, the mail program on the target node writes the message

Table 1 DECnet Session Connect Times under Light Load (All times in seconds)

	VMS Copy		VMS Mail	
	Initial	**Subsequent**	**Initial**	**Subsequent**
VAXstation to VAXstation	8.79	0.50	6.98	1.37
VAXstation to LAVc Alias	9.94	0.95	8.34	2.02
Diskless LAVc to VAXstation	8.75	0.50	6.87	1.39
Diskless LAVc to LAVc Alias	10.17	0.94	NA	NA

Table 2 Delays under Various Loads for 10-by-1000 File Transfer*

	Copy, 10-by-1000			Mail, 10-by-1000		
No. of users (*N*):	*N*=1	*N*=4	*N*=8	*N*=1	*N*=4	*N*=8
VAXstation to VAXstation						
Task	8.71	9.36	9.52	16.48	26.67	45.28
Connect	0.52	0.71	0.68	1.37	3.18	5.65
Session	1.30	1.60	1.52	9.66	14.01	23.94
VAXstation to LAVc Alias						
Task	9.49	9.40	10.01	19.69	34.14	52.82
Connect	0.98	1.00	1.16	2.08	5.37	4.71
Session	1.64	1.40	1.60	12.20	19.23	27.99
Diskless LAVc to VAXstation						
Task	9.24	9.54	9.82	19.16	33.70	42.22
Connect	0.50	0.57	0.60	1.35	2.25	6.42
Session	1.35	1.45	1.63	10.65	16.19	22.71
Diskless LAVc to LAVc Alias						
Task	10.08	10.05	NA	22.13	41.76	NA
Connect	0.94	1.00	NA	NA	NA	NA
Session	1.73	1.50	NA	NA	NA	NA

Notes:
*All times are in seconds and are averages excluding initial activations.
1 user =1 user at source sending to target.
4 users=4 users at source sending to target.
8 users=2 nodes with 4 users each.
Transfers between VAXstation II and LAVc systems: 4 VAXstation II systems to LAVc alias, and 4 diskless LAVc systems to VAXstation II system

Table 3 Mail Resource Usage per Application (Excludes the initial application activation)

Experiment Description	CPU Time (Seconds)	Page Faults	Direct I/O	Buffered I/O	Disk I/O	Page Fault Disk I/O	Disk Time (Sec)
Sending Costs:							
VAXstation II to VAXstation II							
Mail 200-by-50	3.46	355.5	30.7	136.4	85.8	51.5	2.67
Mail 10-by-1000	7.11	462.5	31.1	1082.0	90.6	51.7	2.85
Mail 10-by-1	3.19	453.6	30.0	87.5	85.7	59.1	2.75
Diskless LAVc Satellite to VAXstation II							
Mail 200-by-50	5.57	443.5	26.7	157.1	99.2	60.5	4.12
Mail 10-by-1000	9.22	435.2	26.3	1090.1	96.5	54.0	4.00
Mail 10-by-1	5.18	432.1	25.5	108.5	94.7	63.7	3.94
Receiving Costs:							
VAXstation II to VAXstation II							
Mail 200-by-50	3.03	500.9	79.0	133.0	127.9	39.6	4.34
Mail 10-by-1000	6.53	489.1	81.2	1076.7	128.7	31.1	4.39
Mail 10-by-1	2.47	495.0	57.1	73.1	101.4	37.8	2.94
VAXstation II to LAVc Boot							
Mail 200-by-50	4.98	634.7	94.8	162.5	119.2	42.2	3.31
Mail 10-by-1000	8.65	619.8	99.5	1121.0	119.4	27.8	3.97
Mail 10-by-1	4.34	607.5	74.2	103.6	97.4	37.0	2.74
Diskless LAVc Satellite to LAVc Boot							
Mail 200-by-50	6.71	655.4	98.1	163.7	225.8	84.6	6.49
Mail 10-by-1000	10.61	637.9	100.8	1116.6	225.7	74.1	6.65
Mail 10-by-1	5.94	614.7	75.2	104.0	193.5	89.2	5.18

Table 4 Copy Resource Usage per Application (Excludes initial application activation)

Experiment Description	CPU Time (Seconds)	Page Faults	Direct I/O	Buffered I/O	Disk I/O	Page Fault Disk I/O	Disk Time (Sec)
Sending Costs:							
VAXstation II to VAXstation II							
Copy 200-by-50	0.89	123.2	2.5	37.8	7.0	2.2	0.15
Copy 10-by-1000	0.85	133.0	2.3	38.2	10.3	2.7	0.22
Copy 10-by-1	0.65	78.6	2.0	35.4	7.1	2.7	0.13
Diskless LAVc Satellite to VAXstation II							
Copy 200-by-50	1.57	114.0	4.6	39.1	14.0	3.9	0.52
Copy 10-by-1000	1.67	113.3	4.2	39.0	15.1	4.2	0.52
Copy 10-by-1	1.35	101.1	4.0	35.8	13.6	4.5	0.47
Receiving Costs:							
VAXstation II to VAXstation II							
Copy 200-by-50	1.72	214.1	22.9	60.2	48.2	14.9	1.37
Copy 10-by-1000	1.78	215.0	22.3	60.0	47.7	13.4	1.32
Copy 10-by-1	1.49	164.4	21.0	57.4	45.6	17.8	1.23
VAXstation II to LAVc Boot							
Copy 200-by-50	2.33	234.1	26.0	70.2	42.3	15.3	1.50
Copy 10-by-1000	2.31	235.2	25.6	69.4	37.4	13.2	1.35
Copy 10-by-1	2.09	204.6	25.9	67.7	37.8	16.6	1.40
Diskless LAVc Satellite to LAVc Boot							
Copy 200-by-50	2.57	248.1	25.8	69.2	62.7	21.8	1.93
Copy 10-by-1000	2.56	243.2	25.7	69.4	61.6	23.0	2.06
Copy 10-by-1	2.30	199.7	25.2	67.1	64.2	21.7	1.87

to a temporary file (mail.txt), reads mail.txt, and writes it to a permanent mailnnn.mai file.

For copy, the disk I/O is minimal on the sending node. On the receiving node, a large portion of the disk I/O can again be attributed to paging related to files associated with user authorization.

The network activity levels associated with the mail experiments are shown in Figure 7. The median is a more stable indicator of the trend of the Ethernet utilization when most of the data is close to zero. The effects of increased mail load on Ethernet utilization are negligible for standalone-only traffic but steadily increase when LAVc satellites are active. A major portion of this utilization is due to the maximum length Ethernet packets seen in the protocol exchanges between the LAVc boot member and its satellites. Again, a large portion of these exchanges is due to the large amount of mail paging.

Boundary Conditions The initial mail workload used was stressful, with all users starting simultaneously. This arbitrary design tended to cause VAX/VMS Mail failures (incomplete mail attempts) much earlier in the experiment set than was expected. Subsequent experiments with staggered

Table 5 Disk Measurements for Initial Transfer of 200-by-50 File

Time Window	Disk Reads	Disk Writes
Copy Sender		
Start-up	5	0
Connect	0	0
Session	7	0
Run-down	0	1
Copy Receiver		
Connect	62	8
Session	5	4
Run-down	1	0
Mail Sender		
Start-up	26	0
Connect	0	0
Session	6	0
Run-down	1	1
Mail Receiver		
Connect	73	6
Session	45	33
Run-down	7	8

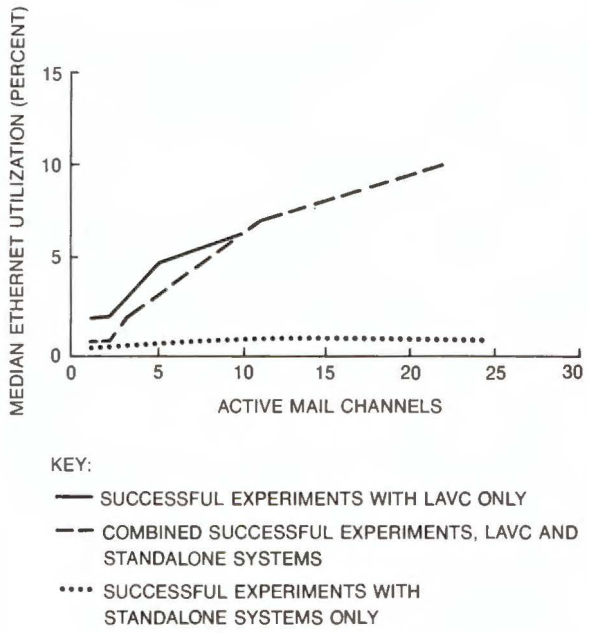


Figure 7 Median Ethernet Utilization versus Mail Load

initial mail activations produced failures further into an experiment run but did not extend the distributed system's workload capacity.

Within the experiments, mail failures occurred for both standalone systems and LAVc satellites when they were attempting to send mail and receive mail from six or more sources at the same time. Also, failures experienced by LAVc-to-LAVc mail took place at a much lower offered load than with the other node types.

VAX/VMS Mail uses the Mail-11 application layer protocol on top of the supporting DECnet protocols. Mail-11 is a master/slave protocol that allows the source and destination to exchange configuration, message format, and other mail application information along with the actual mail messages. Mail-11 does not implement any timers but relies on NCP session establishment timers: incoming (45 seconds) and outgoing (60 seconds).

Mail channel establishments were timed using an internally developed network monitor and were observed to be as large as 35 to 45 seconds in experiments with heavy loads. These maximum connect times were an order of magnitude larger than the average connect times measured in the same experiment. This variability makes one suspicious of NCP timeouts occurring.

Figure 8 is the SPM measurement of the LAVc boot member disk rate before and during the time of failure. It is quite evident that the RA81 on the LAVc boot member has exceeded its estimated capacity, and that excessive queuing time for disk I/O may have caused protocol timeouts to occur. Here, the frequent sampling of SPM data provided some insights into the transient demands on a critical resource during application failure.

Bottleneck Analysis The aggregate resource demands for an application as discussed in the previous section are easily obtainable. Although too gross for input into models, this information can be used for capacity planning purposes. A simple approach, called asymptotic bounds analysis, can be used for two purposes: to investigate the influences of bottleneck devices, and to determine a user or job threshold above which performance will degrade.[8]

The following assumptions are made in using asymptotic analysis:

- Each system (sender and receiver) is doing no other work than the application of interest.

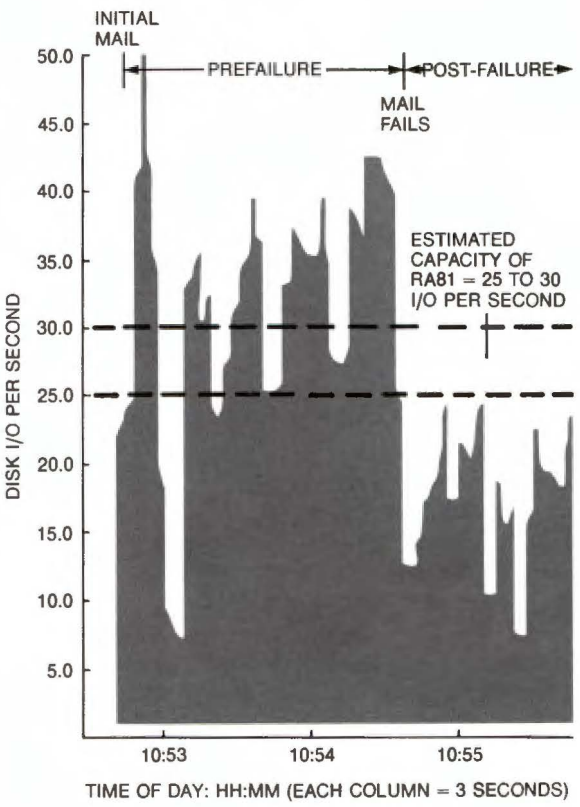


Figure 8 LAVc Boot Member Disk I/O per Second during Mail Failure

- Simultaneous resource possession does not take place. (A pessimistic assumption.)
- Service times are load independent. (An optimistic assumption.)

The following definitions are associated with asymptotic analysis:

- D is the sum of all resource service times in seconds associated with one activation of the application.
- D_{max} is the maximum service time in seconds associated with any one resource. The inverse of D_{max} is, then, the throughput limit in applications per second.
- Z is the average think time in seconds.
- N is the number of users in the system.

The bounds on application throughput crossover from a light load bound to a heavy load bound at a critical user population level $N^* = (D+Z)/D_{max}$. For N greater than N^*, the heavy load bound applies and performance degradation is more probable.

Tables 6 and 7 present the results of applying asymptotic analysis to the mail and copy resource usage data. The tables present the throughput limits for each initiator/target node type and the user thresholds for three different "think times," i.e., job arrival rate and values. Here, the following definitions of total resource demands are used.

For standalones:

$$D = CPU_{src} + DISK_{src} + ENET + CPU_{dest} + DISK_{dest}$$

For diskless LAVc satellites:

$$D = CPU_{src} + CPU_{bm} + DISK_{bm} + ENET + CPU_{dest} + DISK_{dest}$$

For mail, the sending CPU is the bottleneck when the target is a VAXstation standalone system, whereas the receiving CPU is the bottleneck when the target is the LAVc. For copy, the bottleneck device happens to be the destination node CPU in all cases. These bottlenecks are determined, of course, by looking at device service demands associated with one complete application activation. Previously, we hypothesized that the LAVc boot member disk was a bottleneck which resulted in mail application failure. The disk I/O measurements in Table 5 also give evidence that, during the session establishment phase of mail, the disk on the receiver and not the CPU is probably the bottleneck. Determining resource usage during critical phases of an application may be more meaningful than looking at resource usage over the entire application.

The population threshold differences between mail and copy demonstrate the efficiency of the copy operation over mail. The threshold numbers reflect the sensitivity copy performance has to receiver node type. Mail is somewhat less sensitive to receiver type, but like copy, supports the minimal user population when intra-LAVc transfers occur. Knowing this service demand information about multiple applications, network planners can forecast population thresholds associated with mixed workloads composed of these applications.

Summary

The empirical approach discussed in this paper has proven viable for both product interoperability and detailed distributed system performance evaluation. Use of the commonality between products and the applied task decomposition techniques reduced the complexity of the experimental design to a reasonable level. The test-bed and instrumentation demonstrated a great deal of flexibility and capability. They can be easily expanded to investigate the performance of applications and services other than those addressed in the case studies. The tools employed for data acquisition, reduction, and analysis allowed observation of trends of distributed system behavior and quantitative characterization of end user and resource usage parameters. Additionally, they provided sufficient information for queuing model parameterization and validation.

Combined with appropriate modeling techniques, the experimental approach provides a powerful vehicle for the performance evaluation of the wide range of heterogeneous distributed systems covering LANs, XLANs, and even WANs. The knowledge gained in using this methodology can help to develop such systems and plan their growth. The interoperability testing of DNS, DFS, and RSM demonstrated that these products affect the performance of the network at tolerable levels and meet product performance specifications. A decomposition of network transfers by VAX/VMS Mail and VAX/VMS Copy quantified how these two popular distributed applications differ in delay and resource usage and how these differences can affect capacity planning.

Acknowledgments

The authors wish to acknowledge the efforts of Wally Hager and Joe Shih who managed this work and of our colleagues who contributed to the planning, execution, analysis, and support of the case study activities.

Table 6 Asymptotic Bounds on VAX/VMS Mail Performance

Application Description	Total Service Demand D (sec)	Bottleneck Service Demand D_{max} (sec)	Application Throughput Limit (Mail/sec)	User Population Threshold N^* For Think Times (Z=sec) $Z=0$	$Z=60$	$Z=120$
VAXstation to VAXstation						
Mail 200-by-50	13.6	4.34	0.23	3.1	17.0	30.8
Mail 10-by-1000	20.98	7.11	0.14	3.0	11.4	19.8
Mail 10-by-1	11.45	3.19	0.31	3.6	22.4	41.2
VAXstation to LAVc Alias						
Mail 200-by-50	14.94	4.98	0.20	3.0	15.0	27.1
Mail 10-by-1000	23.18	8.65	0.12	2.7	9.6	16.6
Mail 10-by-1	13.13	4.34	0.23	3.0	16.9	30.7
Diskless LAVc to VAXstation						
Mail 200-by-50	17.53	5.57	0.18	3.1	13.9	24.7
Mail 10-by-1000	24.55	9.22	0.11	2.7	9.2	15.7
Mail 10-by-1	14.91	5.18	0.19	2.9	14.5	26.0
Diskless LAVc to LAVc Alias						
Mail 200-by-50	19.82	6.71	0.15	3.0	11.9	20.8
Mail 10-by-1000	27.72	10.61	0.09	2.6	8.3	13.9
Mail 10-by-1	17.41	5.94	0.17	2.9	13.0	23.1

Table 7 Asymptotic Bounds on VAX/VMS Copy Performance

Application Description	Total Service Demand D (sec)	Bottleneck Service Demand D_{max} (sec)	Application Throughput Limit (Mail/sec)	User Population Threshold N^* For Think Times (Z=sec) $Z=0$	$Z=60$	$Z=120$
VAXstation to VAXstation						
Copy 200-by-50	4.15	1.72	0.58	2.4	37.3	72.2
Copy 10-by-1000	4.19	1.78	0.56	2.4	36.1	69.8
Copy 10-by-1	3.52	1.49	0.67	2.4	42.6	82.9
VAXstation to LAVc Alias						
Copy 200-by-50	4.96	2.33	0.43	2.1	27.9	53.6
Copy 10-by-1000	4.79	2.31	0.43	2.1	28.0	54.0
Copy 10-by-1	4.33	2.09	0.49	2.1	30.8	59.5
Diskless LAVc to VAXstation						
Copy 200-by-50	5.65	1.73	0.58	3.3	38.0	72.6
Copy 10-by-1000	5.83	1.74	0.57	3.4	37.8	72.3
Copy 10-by-1	4.70	1.4	0.71	3.4	46.2	89.1
Diskless LAVc to LAVc Alias						
Copy 200-by-50	6.36	2.57	0.39	2.5	25.8	49.2
Copy 10-by-1000	6.82	2.56	0.39	2.7	26.1	49.5
Copy 10-by-1	6.35	2.30	0.43	2.8	28.8	54.9

References

1. L. Kleinrock, "Performance Evaluation of Distributed Computer Communications Systems" (Technical Report, UCLA Computer Science Department, April 1988).
2. S. Lavenberg, *Computer Performance Modeling Handbook* (New York: Academic Press, 1983).
3. E. Pinsky, "A Polynomial-time Algorithm for the Approximate Analysis of Large BCMP Networks" (Technical Report, Computer Science Department, Boston University, March 1988).
4. C. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, vol. 27 (July 1948): 379–423.
5. G. Taguchi, *System of Experimental Design*, vols. 1 and 2 (White Plains: UNIPUB/Kraus International Publications, 1987).

6. *Guide to VAX SPM* (Maynard: Digital Equipment Corporation, Order No. AA-G139A-TE, 1986).

7. *LAN Traffic Monitor User's Guide* (Maynard: Digital Equipment Corporation, Order No. AA-JP16A-TE, June 1987).

8. E. Lazowska, et al., *Quantitative System Performance: Computer System Analysis Using Queueing Network Models* (Englewood Cliffs: Prentice-Hall, 1984).

9. V. D'Silva and R. Hsiao, "Measurement and Analysis Techniques for DECnet Products," *Digital Technical Journal* (June 1989, this issue): 78–86.

10. S. Martin, J. McCann, and D. Oran, "Development of the VAX Distributed Name Service," *Digital Technical Journal* (June 1989, this issue): 9–15.

11. D. Griffin and B. Johnson, "Remote System Management in Network Environments," *Digital Technical Journal* (June 1989, this issue): 29–36.

Vijay G. D'Silva
Ruei-Hsin Hsiao

Measurement and Analysis Techniques for DECnet Products

The Network Performance and Conformance Engineering Group is actively involved in the stages of product design, development, and implementation. Specifically to evaluate an implementation, analysts first define metrics that effectively reflect a product's performance. They use and sometimes develop software tools for testing, and the results are then generalized and extended using a wide variety of analytical techniques. Two cases serve to demonstrate the utility of their approach: the performance evaluations of VAX Distributed File Service and those of VAX Distributed Name Service software. The various factors affecting performance of these cases were identified, and these findings led to increased product performance.

Increasingly in today's market, performance determines a product's acceptance. Therefore performance analysis and modeling now play a prominent role in system design and development. This paper presents some commonly used measurement and analysis methods and their application to two of Digital's networking products: VAX Distributed File Service (DFS) and VAX Distributed Name Service (DNS).

Performance evaluation and analysis techniques can be applied during the three stages of a product's development: (1) the design stage, (2) the development process, and (3) the finished product. Analysis and simulation in the design stage can detect potential problems and correct them early in the cycle. During the development process, testing can identify which of several alternative code segments to adopt for best performance. Finally, testing and analysis of the finished product involves benchmarking the product. This stage is essential for capacity planning and aids in the design of the next generation of products.

The performance evaluation group within Network Performance and Conformance Engineering (NPACE) is actively involved in all three of these stages. This paper concentrates on testing the finished product and presents a methodology with which software products can be made more efficient and competitive using a simplified and systematic approach.

The following section discusses the metrics that characterize a product's performance. We then describe test methods and analysis techniques and their applications in testing specific products. The later sections present the two case studies of DFS and DNS and describe the measurement and analysis of these products.

Performance Metrics

When evaluating a product, the analyst must choose metrics that effectively and appropriately reflect the product's performance. The performance metrics can often be applied to a class of products even though individual test methods may differ. The job of the performance analyst is to identify those metrics that most effectively and succinctly characterize the system's behavior. Each of these metrics is in turn affected by several factors, and an n-dimensional matrix would be necessary to demonstrate this. The performance analyst must concentrate on those factors that have the maximum impact on the system.

Performance metrics can be broadly classified into two categories: those that apply to performance perceived directly by an individual user, and those that apply to system performance.

Latency

Latency is simply the time taken to complete an operation. For example, an analysis of VAX FTAM, Digital's implementation of the Open System Interconnect (OSI) File Transfer, Access and Management protocol, revealed that response time depended heavily on the record size in one of the three file types supported. With this information, we were able to derive equations for the latency in terms of the file size and record size. These results are being

used in the field to configure systems. The response time consists of the execution time for the operation and queuing delays due to other processes on the system. Separating these delays from the execution time often facilitates analysis.

Throughput

Throughput is the number of operations that can be processed per unit of time. For data transfer operations, which form the bulk of the Network and Communications (NAC) Group's product set, the throughput is often the amount of user data (not including protocol overhead) transferred per second. The response time for those operations is easily calculated from the start-up time and the steady-state throughput.

CPU Utilization

Considered vital in determining the efficiency of the software and the drain on CPU resources, CPU utilization is a measure of the time that the local or remote CPU is busy. Conventional CPU measurement is accomplished by attributing each time slice to a particular process. This method has the disadvantage of not accounting for interrupt-level routines, other referenced processes, and the scheduling overhead itself. To factor these effects into our calculations, we decided to write a software program that measures the amount of CPU time not used by the product. We were surprised to find that often the CPU load was much higher than we previously thought. This method of measurement has become the standard within NPACE.

Line Utilization

Primarily of use when dealing with point-to-point links, the line utilization is a measure of the percentage of time that a line is busy transferring data. In several tests we measured the maximum capacity of the DECserver 200, an Ethernet terminal server for as many as eight terminals. We found that all eight terminal lines could be as much as 91 percent utilized without loss of data. This measurement underscored the performance of the DECserver 200 product and, more importantly, defined a theoretical limit for its throughput.

Measurements and Analysis Methods

Much of the measurement at NPACE occurs at the Digital command language (DCL) interface. The reason for this is twofold: the results from such measurement are reproducible by the typical user, and the measurements also reflect the delays experienced by the user. To bypass the random effect of other processes using the same resources, performance testing is invariably conducted on standalone machines and private networks, i.e., resources dedicated to the current measurement.

Performance tools can be divided into general-purpose, reusable tools and special-purpose test programs. The general-purpose tools are software performance monitors (VAX SPM and VAX PCA) and hardware monitors, such as logic analyzers, datascopes, and Ethernet monitors. Depending on a product's functionality and the type of results required, specific tools often have to be developed. For example, the LAT protocol allows packets from different sessions to be combined into a single packet. During the evaluation of the DECserver 500, an Ethernet terminal server, it was impossible to analytically predict the effect of heavy loads on the unit. A test program to simulate as many as 128 terminals acting in conjunction had to be written to facilitate our analysis.

Like performance tools, analysis methods are chosen depending on the rationale behind the performance evaluation process. In some cases, the performance evaluation merely determines benchmarks, which are sets of numbers that effectively describe the performance of the product. In these cases simple equations that define the CPU load and response time for different configurations are sufficient. In other cases the performance evaluation focuses on problem areas in the prototype. In these cases the goal is to avoid problems early in the design phase by performing a detailed analysis along with a series of tests.

Time-stamping individual events helps break down a delay or a response time into its components, and this method is used extensively. In the evaluation of the LAN Bridge 100, Digital's high-performance Ethernet bridge, we employed two major performance metrics:

- The double-side packet discard rate — the rate at which the bridge can process packets that it *does not* have to forward
- The single-side packet forward rate — the rate at which the bridge can process packets that it *does* have to forward

We used an Ethernet monitor and logic analyzer to trace the instructions executed in the critical code path and time-stamped the major events in each hardware component. This analysis identified the exact location of the performance bottlenecks and determined the maximum capacity of the LAN Bridge 100 product. When the next version of the bridge was designed, more powerful hardware was appropriately chosen, effectively reducing the response time.

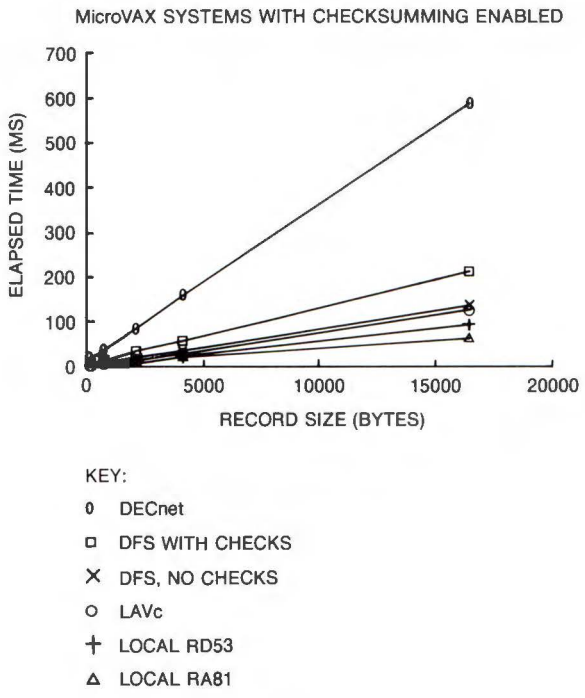


Figure 1 Comparison of DECnet-DAP, DFS, LAVc, and Local Access for Read

Parametric analysis is used to detect a product's sensitivity to packet sizes, buffer sizes, and number of buffers. By continually varying different parameters, we can derive an *n*-dimensional graph of the response time and other parameters. Bottleneck analysis techniques are used to approximate the total capacity of a system with asynchronously executing threads. These techniques are also used extensively as will be shown later in the section Case Study: Distributed Name Service.

Finally, simulation can save much time and developmental effort when direct examination of the interaction of different components is not practical. By using simulation, we are able to predict a system's performance in a given situation without having to use resources and time in actual measurement.

Case Study: Distributed File Service

VAX DFS is layered software that implements a client-server model.[1] For most operations, the location of a file, whether local or remote, is almost transparent to the user. Thus, with a few exceptions, the only difference between local and remote files is the speed with which the user can access and manipulate the file.

To enhance the performance of the DFS server, two methods were used: (1) caching on the server, and (2) interrupt stack execution of frequent operations. (For a detailed description of DFS, see reference 1.) Briefly, the server cache stores the most recently requested blocks in buffers of 16 blocks each; interrupt stack execution means the requests can be processed as they arrive and do not have to wait until the server processor schedules them.[2] Both approaches have performance implications that are particularly evident under certain situations. These implications are discussed in the section Performance Analysis of DFS.

To evaluate such a system, we decided that the logical place to start was the single-server, single-client case. The aim was twofold: first, to compare DFS to other methods of access running on the VMS operating system; and second, to analyze a multi-client system so that the performance bounds of a DFS system could be predicted.

Comparative Measurements

To measure the single-client case, we considered several different types of tests, which are described in subsequent sections. In a real-life situation, it is likely that machines act as both servers and clients, depending on the setup. To represent Digital's wide range of computers, we also decided to repeat the complete set of tests between MicroVAX II and VAX 8800 systems. In choosing the operations to measure, we decided on the DCL-level copy operation and the file access routines create, open/close, and delete. These operations are executed frequently by the typical user. However, in most cases the vehicle by which files are accessed by higher level programs is Record Management Services (RMS). To evaluate the effect of using DFS with RMS, we measured sequential file access as well as randomly accessed index files. Since caching was implemented on the server, a tool was also written to issue repeated reads to the same data, causing a consistent cache hit with no disk access.

When files are read sequentially by RMS, disk accesses are minimized by reading large buffers (specified by the multiblock count) into the internal buffers of RMS. Thus for small records, most of the reads are internally satisfied and only the occasional read necessitates a disk access. For index files, the performance is based on the order in which records are accessed. To find a record, it may be necessary to read several buckets before a particular key (and record) is found.

Extensive results representing various situations were obtained from these measurements. Figure 1 shows a comparison of DFS, local area VAXcluster (LAVc), DECnet-DAP and local access in which the operation is the RMS primitive, sys$read. In comparing these access methods, we noticed the degree to which performance is related to closeness of sys-

tem coupling. As we progressed from loosely coupled to tightly coupled systems, our tests revealed an improvement in performance proceeding from DECnet-DAP, to DFS, to LAVc, to local disk access.

Performance Analysis of DFS

A model of a DFS server being accessed by several clients enabled us to predict how an installed DFS system would behave in a given user environment. A series of measurements with a single client was conducted to determine the input parameters to the model. To obtain analytical results, the model was reduced to its main components. (See Figure 2.)

The following assumptions were made:

- Each client process can have only one outstanding request at a time. In this way the total number of requests in the system is equal to the number of clients.
- The time elapsed between a client's receiving a response and issuing its next request is independent of the number of users, i.e., the client behavior can be modeled as "think" time.
- The server consists of two components: a processor and a disk subsystem. The CPU time is the server response time when a cache hit occurs. The disk time is the time needed by a disk to service a request.
- The model represents a system in which data on the server is accessed by the clients, and the flow of user data is predominantly from the server to the clients. We also assumed that control operations such as file open, close, delete, and extend do not appreciably affect the performance of the data access operations.

Input Parameters The CPU time used is directly related to the amount of data being transferred. The disk service time consists of a seek time, a rotational delay, and a data access time. The seek time depends on the type of access. For a single process that sequentially reads data, the seek time is small because the data within a file is assumed to be contiguous. When several processes access data on the same disk, the seek time is higher since the head must move about the disk to access different files. We measured the average server CPU time per operation and the individual response times of the server and client.

From our measurements, we noticed that a cache miss almost always resulted in an extra 20 milliseconds (ms) in the response time. To explain this, we considered that the RA-series disks rotate at 3600 rpm (16.67 ms per revolution), and the RA81, in particular, stores approximately three 16-block buffers per track. In the case of a single contiguous file being read 16 blocks at a time, each read eventually has to be synchronized with the revolution of the disk. Moreover, after every three reads, a single-track seek is necessary. This explanation coincides with the 20-ms delay that was measured. When several files are being simultaneously accessed on the disk, the head is required to seek the next request. (The average RA81 seek time is 28 ms.) This seek could overlap with the rotational delay and result in a total access time of approximately 45 ms.

Analytical Results Mean value analysis provides a way of examining the delays that the average request is expected to encounter. This analysis does not involve itself in the distributions of the individual delays, which may frequently be theoretically impossible to derive. Using this approach, we attempted to formulate an equation for the delay experienced by a single request.

In a system that has N_{dfs} client processes, an average server CPU time of T_{cpu}, and an average of N_{cpu} requests waiting for the CPU, the time in and waiting for the server CPU, T_{uc}, is

$$T_{uc} = N_{cpu} \times T_{cpu} + T_{cpu}$$

Similarly, the time spent in and waiting for the disk subsystem, T_{ud}, is

$$T_{ud} = N_d \times T_d + T_d$$

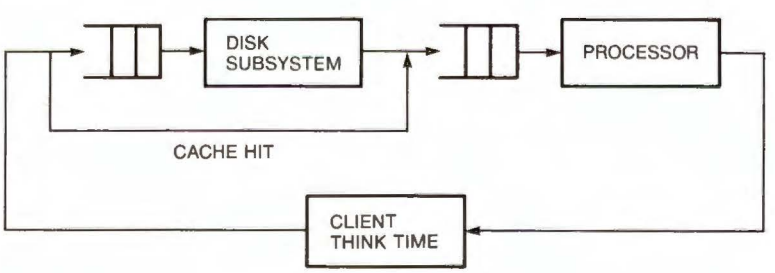


Figure 2 Model of the DFS System

The total round trip time for the average request is given by

$$T_{cl} + T_{wc} + p_m \times T_{wd}.$$

Therefore, if we look at the system from the point of view of one of the clients, we expect N_{cpu} and N_d to be

$$N_{cpu} = (N_{dfs} - 1) \times T_{wc}/(T_{cl} + T_{wc} + p_m \times T_{wd})$$
$$N_d = (N_{dfs} - 1) \times T_{wd}/(T_{cl} + T_{wc} + p_m \times T_{wd})$$

where p_m is the probability that a cache miss occurs. This result could also be obtained from Little's Law if we consider that the effective traffic seen by the current request is that imposed by the other request.[3]

$$N = \lambda T$$

where $\lambda = (N_{dfs} - 1)/(T_{cl} + T_{wc} + p_m \times T_{wd})$, the arrival rate of the other requests to the resource, and T, the service time of the resource.

Likewise, the time for the server to satisfy the request would thus be

$$T_{server} = T_{wc} + p_m \times T_{wd}$$

These equations are of special interest when we consider the following cases:

Case 1: $p_m = 0$, $N_{dfs} = 1$. In case 1 a single client repeatedly accesses a limited amount of data (cache hits). The equations above reduce to

$$T_{server} = T_{cpu}$$

Case 2: $p_m = 1$, $N_{dfs} = 1$. In case 2 a single client always requests new data (cache misses). The equations above now reduce to

$$T_{server} = T_{cpu} + T_d$$

Case 3: $p_m = 0$, $N_{dfs} >> 1$. In case 3 several clients bottleneck the server, and the cache is always hit. The time in the server is now given by

$$T_{server} = N_{dfs} T_{cpu} - T_{cl}$$

This result can also be obtained from a fluid approximation of the system, with the assumption that the CPU is always fully utilized and every request is either in the queue for the CPU or in service.

Simulation Results and Validation To give ourselves a greater amount of flexibility and detail, we wrote a simulation model of the DFS system, shown in Figure 2. A modification in the model accounted for post-I/O processing and disk service times on shared disks. To validate the model, a program that sequentially reads data from a large file ran from one to six processes, and the throughput gained was measured. This measurement was compared to the throughput predicted by the simulation model for two cases: when all different files were resident on a single disk, and when the data was equally shared among four disks. As Figure 3 indicates, the simulated throughput was within 10 percent of that measured for all cases.

For a single user the throughput of the system is 1.05 megabits (Mb) per second. For several users, the maximum throughput measured with a MicroVAX II server was found to be 2.6Mb per second. These results were also confirmed using a program that consistently reads data directly from the cache. Since the server CPU was being fully utilized at this point, increasing the capacity of any of the other components would not have improved performance. However, the easy-to-use technique of bottleneck analysis described below allowed us to arrive at the same result with little or no effort.

Bottleneck Analysis When configuring large systems in which the type of traffic is known, bottleneck analysis gives a good estimate of the system's maximum capacity. Associating a service time with each component and assuming complete asynchronicity of the operations, we used this method to predict the throughput for different configurations. This analysis assumes that the slowest component, i.e., the bottleneck, invariably determines the maximum throughput of the whole system in the same way that fewer lanes on a highway determines how many drivers using that road get to work on time.

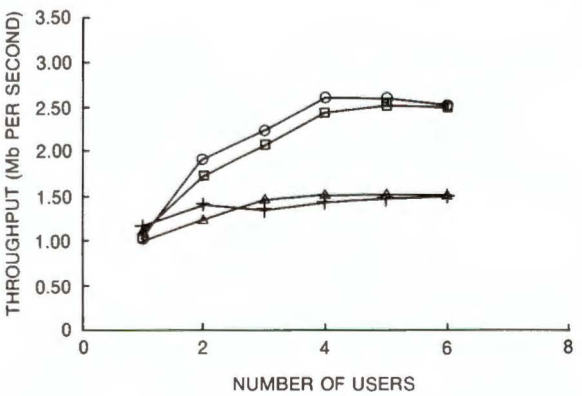


Figure 3 Simulated versus Measured Results of I/O Processing

Table 1 Service Times of DFS Components

Resource	Service Time
Server CPU, MicroVAX II	4.9+1.26 × blocks ms
Server CPU, VAX 8700	1.4+0.28 × blocks ms
Disk time for contiguous data	20 ms
Disk time for noncontiguous data	44 ms
Effective network service time	1.08 × blocks ms
Client CPU, MicroVAX II (continuous read)	6.2+1.26 × blocks ms
Client CPU, VAX 8700 (continuous read)	1.4+0.28 × blocks ms

Table 2 Workload Statistics for Two Servers

	Server 1 Cluster	Server 2 Cluster
Approximate number of operations	20,591K	816K
Probability of persona cache miss	0.2%	0.5%
Probability of data cache miss (for read)	23.0%	86.0%
Average read size (approximate)	3.4 blocks	13.5 blocks
Average write size (approximate)	~16.0 blocks	~16.0 blocks
Read/write ratio	20:1	11:1

Table 1 lists the service time for each component of a DFS system. We assume that the most frequent operation is the read primitive.

The maximum throughput that a system of several clients and a single server can thus process is given by

$$T = 1/max(T_{cpu}, p_m \times T_d/D, T_{net}, T_{cl}/C)$$

where D is the number of DFS disks serviced by the CPU, T_{net} is the effective time to transfer the data across the network, and C is the number of clients that use the DFS server.

In the case of users on C-client CPUs accessing noncontiguous data on a MicroVAX II server with an RA81 disk, an average read size of 6 blocks, and p_m equals 0.25, the throughput is thus

$$1/max(4.9 + 1.26 \times 6, 0.25 \times 44, 1.08 \times 6, (6.2 + 1.26 \times 6)/C)$$

That is, for 73 operations per second, n equals 1; and for 80 operations per second, C is greater than 1.

Sample Workload Characterization

Approaching the problem from a different perspective, we examined installed DFS systems to obtain information about real-life workloads. The VAXcluster system called Server 1 is a DFS server acting primarily as a database for product save-sets, documentation, and other such reference data. The cluster called Server 2 is similarly used, though with a greater developmental and engineering bias. The statistics obtained are shown in Table 2.

Considering a file read sequentially n blocks at a time, the cache miss ratio is $n/16$. Thus an average read size of 3.4 blocks would imply a cache miss ratio of 21 percent. This coincides remarkably with the Server 1 system's measured cache hit ratio of 23 percent. For the Server 2 cluster, the predicted and measured values are 84 percent and 86 percent, respectively. We thus concluded that in both systems, files are very seldom already in the cache when initially requested by a user, and the major benefit of caching is that small access sizes do not necessarily result in disk accesses.

On the DFS server, the persona cache keeps track of user authorization information. The persona cache and its implications have been omitted in this paper precisely because this cache is so effective that a cache hit almost always occurs.[1]

Case Study: Distributed Name Service

A name service maps a name of an individual, group, organization, facility, or resource into a set of labeled properties, each of which is stored as a string of characters. The major purpose of a name service is to enhance the logical organization of large networks by allowing the names of network applications, systems, and other network-accessible objects to be independent of their physical location in the network.

VAX DNS is Digital's implementation of the Digital Network Architecture (DNA) naming service.[4] The service is implemented as a client-server model in which the node running DNS acts as the server. Figure 4 shows the relationship between two name servers and a client.

A client accesses DNS through a naming service module called the clerk. The clerk communicates with as many name servers as necessary to find one that satisfies a request.

Each name stored in a name server refers to a single, unique network-accessible object. The transaction agent module on a name server performs DNS transactions requested by clerks. A typical DNS transaction requested by a clerk would be reading the attributes associated with an object. In this case, upon receiving the request from a clerk, the

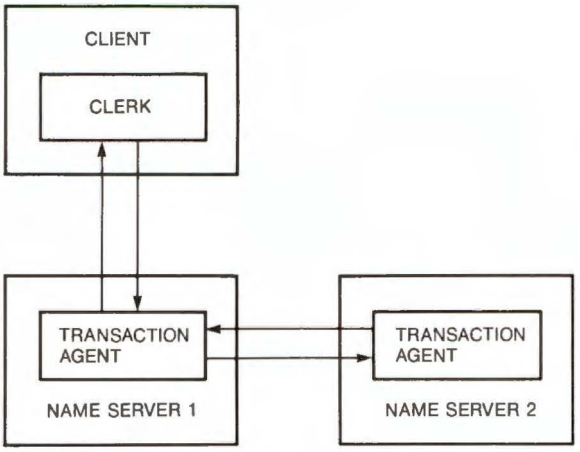


Figure 4 Distributed Name Service

transaction agent searches the name server clearinghouse for the attributes associated with the object, and sends the information to the clerk.

The goals of the DNS performance evaluation were to characterize its behavior, identify the bottleneck, and find ways to improve DNS performance.

The performance metrics of DNS include

- Response time (latency) per DNS transaction requested by clerks
- Name server and clerk CPU time per DNS transaction
- Name server capacity
- Line utilization

We examined three aspects of the name server's performance:

- DNS performance in a local area network (LAN)
- DNS operations in a wide area network (WAN)
- Program counter (PC) histogram and Ethernet packet trace

DNS Performance in a Local Area Network

The most commonly executed DNS transactions are read, add, and remove attributes of objects. Of these three, read transactions are used most frequently. In practice, an application program usually requests a specific attribute of an object (such as the DECnet node address of a particular DFS volume mounted on a DFS server). During a read attribute operation, the clerk chooses the name server from the clerk cache, issues the request to that name server, and waits for the reply. On receiving a request, the name server searches the clearinghouse for the object and returns the attribute requested.

Clerk Operation The actual creation of the request packet and its transmission takes 31 ms; this is the overhead that will be experienced for all requests. However, when the clerk first receives a request from an application program, an additional 30 to 40 ms is involved in building the DNS context, initializing associated data structures, and choosing a name server from its cache.

Server Operation The clearinghouse is stored on disk and is accessed by means of RMS. RMS maintains its own cache, and the response time is 25 ms when the data is in the cache. Sometimes, however, the data may not be in the cache, and a disk access is necessary. This step involves another 6 ms of CPU time and adds 42 ms (for an RA81 disk) or 62 ms (for an RD54 disk) to the response time.

The RMS cache hit and cache miss ratios are determined by the number of RMS buffers, the number of index buckets, and the number of data buckets. The number of data buckets is determined by the number of DNS objects in the clearinghouse, the size of each DNS object, and the bucket fill rate.[5] To reduce the number of disk I/O operations per DNS transaction, we recommended that the number of RMS buffers should be greater than the number of index buckets. In that way RMS can cache all the index buckets in the RMS buffers. The cache hit ratio for data buckets is then

$$\text{Cache-hit ratio} = \frac{(\text{Number of RMS buffers} - \text{number of index buckets})}{\text{Number of data buckets}}$$

The name server's average response time directly depends on the cache-hit ratio.

Communications Load The response times given above are for requests that use an existing logical link between the clerk and the server. However, logical links may not necessarily exist; and in the absence of a logical link, one must be created. This additional task affects the performance of DNS, both in terms of the response time and the CPU loads on the two systems. The establishment and disconnection of a logical link takes 88 ms and uses up 60 ms and 68 ms of CPU time, respectively, on the clerk and server. If a logical link had to be created for every request, the maximum throughput of DNS would drop from 33 to only 10 transactions per second.

The DECnet session overhead could be avoided by caching at least one logical link on the name server for every clerk. However, that operation depletes VMS system pool space, and the performance costs are prohibitive when the name server has to serve a large number of clerks. Another approach uses a datagram service for DNS operations within a local

area network. With this approach, however, issues such as authentication have to be resolved so that unauthorized access to the DNS clearinghouse can be prevented. The session layer (DECnet logical link) overhead is a major concern in most client-server implementations (such as file services, remote procedure calls, and name services); and for many client-server implementations that are LAN-based, a datagram-based service could provide better performance than a connection-based service.

DNS Operations in a Wide Area Network

If a clerk is trying to communicate with a name server that is not on the same LAN, the message that contains the DNS request is forwarded by routers to the name server. Both the router processing and data transmission add latency to the response time. Figure 5 shows a typical configuration for this case. A DECrouter 2000 introduces about 1 ms latency for every message it routes. The latency caused by the transmission on the low-speed line depends on the amount of data exchanged and the speed of the line. When the same read-attributes DNS transaction is performed over a WAN, we observed an extra latency of 50 ms at a line speed of 64 kilobits (Kb) per second. The extra latency reduces to 16 ms as the line speed is increased to 256 Kb per second.

Program Counter Histogram and Ethernet Packet Trace

An Ethernet monitor was used to trace and time-stamp all the packets for a DNS transaction. VAX Software Performance Monitor (SPM) generated a system-wide PC histogram on the name server. Figure 6 shows the Ethernet packet traces for a read-attributes transaction. The trace indicates that the response time of the name server is 24.8 ms, excluding disk I/O and the establishment of DECnet session.

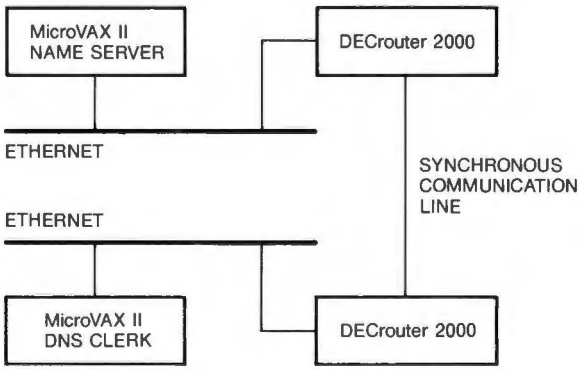


Figure 5 DNS Operation over a Wide Area Network

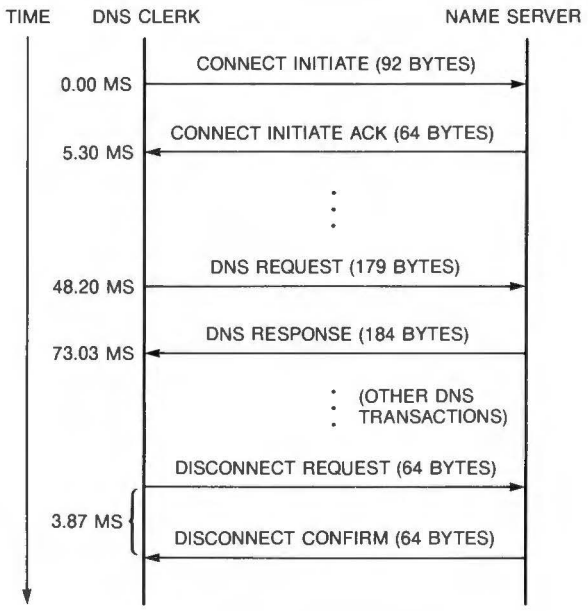


Figure 6 Message Trace for a DNS Operation

The server CPU times spent in each major module for read-attribute transactions are summarized in Figure 7. It was found that most of the name server time is spent in system modules (76 percent). Only 24 percent of the CPU overhead is in the process space (P0 and P1 space).

DNS Server under Severe Load

On a rare occasion the name server may be besieged by requests, for example, after a power outage when several workstations reboot and try to execute a name service lookup simultaneously. Under such conditions the name server may not be able to satisfy every request before the client times out. Each workstation then has the following options:

A. To try a few times and cease

B. To persistently retry until successful

C. To retry with backoff

This scenario was created with 460 client processes on 46 machines and a MicroVAX II name server. Using algorithm A, only a few clients managed to access the server due to intense congestion at the server. Algorithm B could cause severe thrashing if the network parameters are not set up carefully. The backoff algorithm efficiently eliminated the congestion on the server by delaying the requests. Use of this algorithm thus results in the best throughput and still completes the request from every client program.

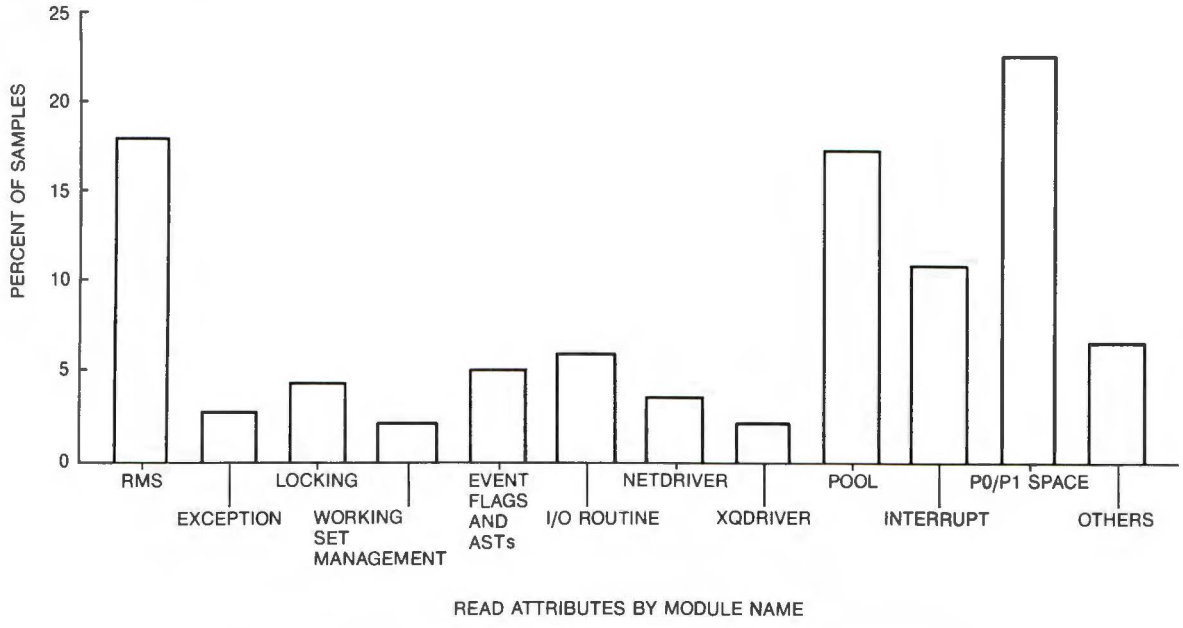


Figure 7 Name Server CPU Usage for a Read-attribute Transaction

Conclusions

The analysis and evaluation of computer network performance has become an important and demanding field. Performance is the primary consideration in evaluating computer networks and requires an in-depth understanding of computer systems and network mechanisms. The present complexity of network performance measurement, analysis, and evaluation methods is a result of the increase in the number of different computer systems and networks. Employing some commonly used performance evaluation techniques and tools, we were able to effectively characterize the performance of two network products, DFS and DNS, and present the trade-offs involved when using them. Similar performance evaluation methods can be applied to other networking products in different environments.

Acknowledgments

The authors wish to acknowledge Ray Kao and Darwen Rau of NPACE Performance Group who contributed to the performance evaluation of DNS and DFS. Special thanks to Dah-Ming Chiu who reviewed this paper and provided valuable insight and information in the section DNS Server under Server Load. We are also grateful to Dinesh Mirchandani, Janet McCann, and Danny Cobb for critiquing this paper and thus helping to clarify it.

References

1. W. Nichols and J. Emer, "Design and Implementation of the VAX Distributed File Service," *Digital Technical Journal* (June 1989, this issue): 16–28.

2. L. Kenah and S. Bate, *VAX/VMS Internals and Data Structures* (Bedford: Digital Press, 1984).

3. L. Kleinrock, *Queueing Systems, Volume I: Theory* (New York: John Wiley & Sons, Inc., 1975).

4. S. Martin, J. McCann, and D. Oran, "Development of the VAX Distributed Name Service," *Digital Technical Journal* (June 1989, this issue): 9–15.

5. *VMS Manual Set, Programming, Volume 6A File System, Guide to File Applications* (Maynard: Digital Equipment Corporation, Order No. AA-LA78A-TE, April 1988).

John P. Morency
Richard P. Pitkin
Ramasamy Jesuraj
Ambrose C. Kwong

Modeling and Analysis of the DECnet/SNA Gateway

The DECnet/SNA Gateway links the DECnet and SNA environments to provide a rich range of services to users. To simulate arbitrary session mixes over this gateway, the Network and Communications Group developed a programmed model primarily as a tool for capacity planning. Designers chose to develop a simulation model — as opposed to a queuing model — to support the DECnet/SNA terminal emulator, the data transfer facility, and remote job entry. Special tools and techniques were created during development to ensure consistent and efficient collection and reporting of experiment results. Validation of the model reveals the model's accuracy; a margin of less than 10 percent is shown between measured results and model output.

Local area networks (LANs) have made distributed computing a reality. For distributed computing to truly be effective, however, connectivity to diverse data resources is required. Very often these data resources reside within an IBM Systems Network Architecture (SNA) network. Because SNA is generally not the networking architecture utilized within the LAN environment, communications gateways must provide protocol mapping between the LAN and SNA environments.

This paper describes a simulation model of the DECnet/SNA Gateway developed by the Network and Communications Group within Digital Equipment Corporation. The generic nature of Digital Network Architecture (DNA) to SNA interconnection and the associated performance issues are described first. Next, the goals and objectives of the DECnet/SNA Gateway model are presented, followed by an in-depth description of the model.

We also describe some of the tools the team developed to validate the model effectively and the problems associated with validating a model of this nature. We conclude with a discussion of the validation of the model and its applicability to various types of workloads.

The DECnet/SNA Gateway Product

The DECnet/SNA Gateway is a combined hardware and software system with communications links to the DECnet and SNA environments. The system is complemented by client products, called access routines, which reside within the DECnet environment and the SNA host environment. In the DECnet environment, these access routines reside on VAX/VMS, ULTRIX, or MS-DOS systems. In the SNA environment, the access routines reside on MVS, VM or DOS/VSE systems.

Capabilities provided by the DECnet/SNA Gateway in conjunction with the access routines include

- Digital VT-class terminal access to an SNA host
- IBM 3270 terminal access to a DECnet host
- Bidirectional file transfer services in which the transfer can be initiated from either the DECnet or the SNA side of the system
- Bidirectional mail services

The design of both the gateway and the associated products is sufficiently modular to allow any combination of layered functions to proceed in parallel across the gateway. Thus a rich range of services is available to users in either network.

The structure of the combined DECnet/SNA environment is shown in Figure 1.[1]

The Need for a Gateway Model

The diversity of concurrent functions possible across a gateway prompted us to develop a programmed model to simulate arbitrary session mixes. The model would be an automated tool to aid customers in the capacity planning process. Moreover, we soon discovered that new gateway designs could be rapidly analyzed and tested for feasibility by a "what-if" analysis tool such as the model we planned.

An additional benefit derived from developing the model is the insight it gave us into the internal

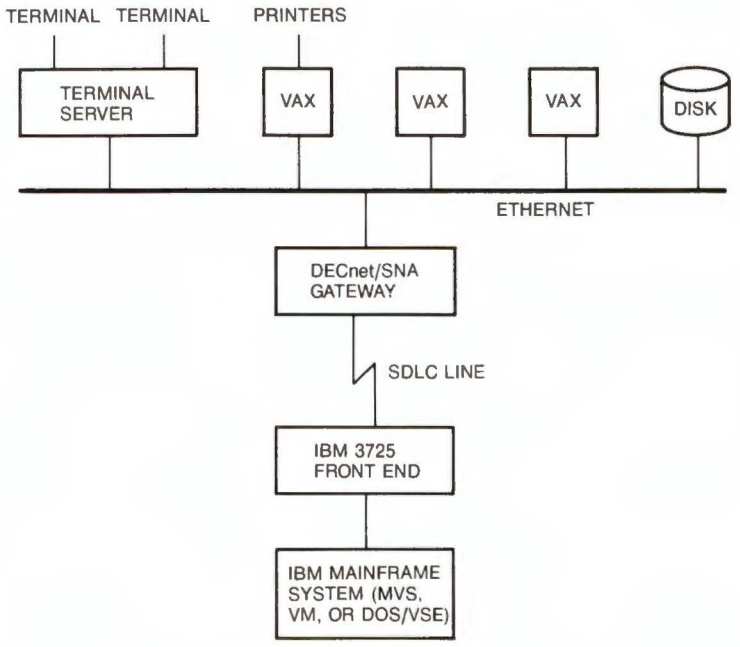


Figure 1 DECnet/SNA Physical Configuration Overview

properties of DECnet and SNA networks. Our development approach was top-down and evolutionary; that is, we designed the simulation to accurately reflect the behavior of the products with respect to overall gateway performance. By studying the performance behavior of one product at a time, we more clearly understood the properties in the two architectures that affected the desired level of interoperability. These products are described in the next section.

DECnet/SNA Products Supported by the Model

DECnet/SNA 3270 Terminal Emulator

The DECnet/SNA 3270 Terminal Emulator (TE) runs on a DECnet host, typically a VAX/VMS system. The TE allows a Digital VT-series terminal, which is generally asynchronous, to emulate an IBM 3178/3278 device using the 3270 base data stream. The TE code uses internal routines to support the gateway access protocol (GAP) to the gateway access server (GAS) process within the gateway.[1] This relationship is illustrated in Figure 2.

To effectively perform its function, the TE image itself processes all keystrokes entered directly by the end user. The TE therefore functions much as would a 3270 cluster control unit on behalf of an attached control unit terminal (CUT) display. In addition, to maintain timely state synchronization, the TE image also supports both the SNA data flow control and transmission control layers. This implementation structure results in a highly reliable emulation which completely supports the GAP protocol for access to lower-level services within the gateway.

Data Transfer Facility

One of the major strengths of DECnet software is the interactive file transfer capability supported by the data access protocol (DAP). This capability can be easily seen on most DECnet systems by suitable variants of a COPY command.

The DECnet/SNA data transfer facility (DTF) extends that copy capability to include catalogued data sets within an SNA environment. This extended capability is provided by a DAP server process which appears as a normal DECnet network server. The actual mechanism encapsulates DAP in an SNA logical unit type 0 session attached to the "real" DAP server process which resides on an MVS host. (The "real" DAP server is a Digital-supplied component on the MVS system.)

The initiator of the COPY command in this context is termed the DTF client; the process that provides the look-alike DAP service server functions is termed the DTF server. It is the DTF server that maintains the SNA session with a cooperating task on the MVS host for the purposes of effecting the end-to-end transfer as seen by the end user. The DTF server process is also a GAP protocol client. The overall DTF structure

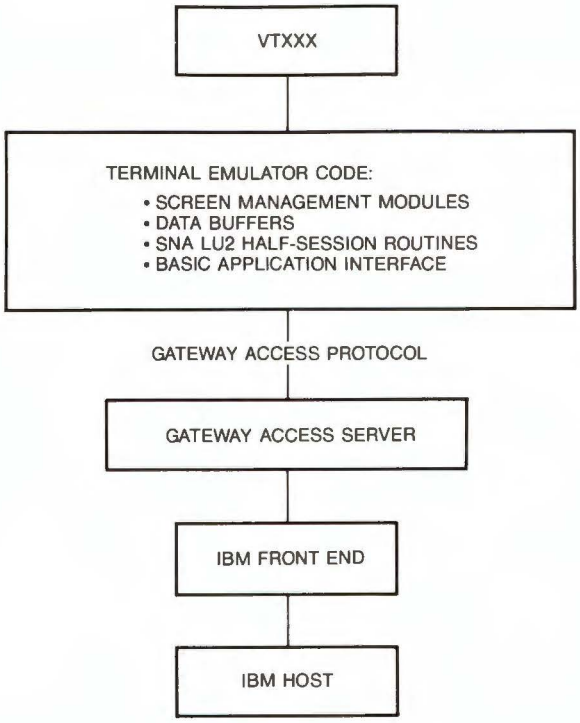


Figure 2 DECnet/SNA Terminal Emulator Overview

is illustrated in Figure 3. Transfers from the IBM side are structured similarly and supported by ISPF panel requests.

Remote Job Entry

As its name implies, DECnet/SNA remote job entry (RJE) allows an end user to submit job control language (JCL) formatted files as units of work to a job entry subsystem (JES) within the SNA network. This same facility allows that end user to receive either printed or punched output from the SNA host when a submitted job has completed. Inbound reader streams (to an SNA host) and outbound printer or punch streams (from an IBM host) are both transferred by means of the DAP protocol from the DECnet/SNA Gateway to the JES subsystem. (In the gateway, full protocol conversion exists to and from an SNA logical unit type 1 data stream.) Figure 4 illustrates this process.

In contrast to the TE and DTF, RJE supports an application-layer protocol conversion paradigm in the gateway.

Gateway Model

In this section we discuss the logical structure of the gateway model. Details of this structure, performance statistics required, and objectives established for the model follow in subsequent sections.

In our approach to structuring the model, we chose to group multiple servers logically and to focus on the characteristics of the larger server. Specifically, we modeled the end users, the VAX/VMS systems, and the Ethernet as constituting the DECnet server; and we modeled the front end, mainframe host, and target applications as representing the SNA server. Our reason for this decision can be seen by examining Figures 1 and 5.

As shown in Figure 1, numerous components must be considered. These include the DECnet node, the Ethernet, the DECnet/SNA Gateway, the synchronous data link control (SDLC) line, the 3725 front-end processor, and finally the mainframe (in our experiments, a 4381-P13 system running the MVS/XA operating system).

We can look at this same physical configuration from a queuing system standpoint, as diagrammed in Figure 5. Figure 5 represents each of the physical entities as logical servers and also indicates the units of data passed between the servers. For example, an inbound flow (i.e., from the gateway to the IBM front end) can be modeled as follows. End users at a DECnet node use a product to transmit DECnet packets to the gateway over the Ethernet. An actual SNA

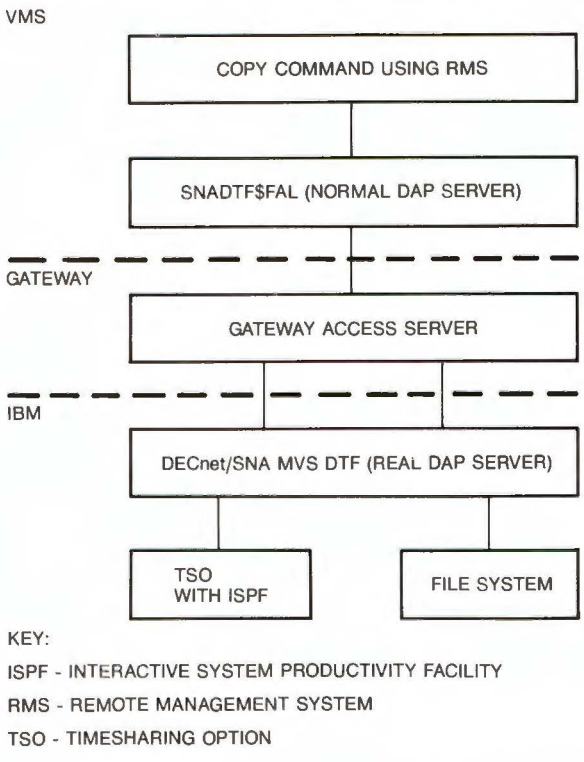


Figure 3 Data Transfer Facility Overview

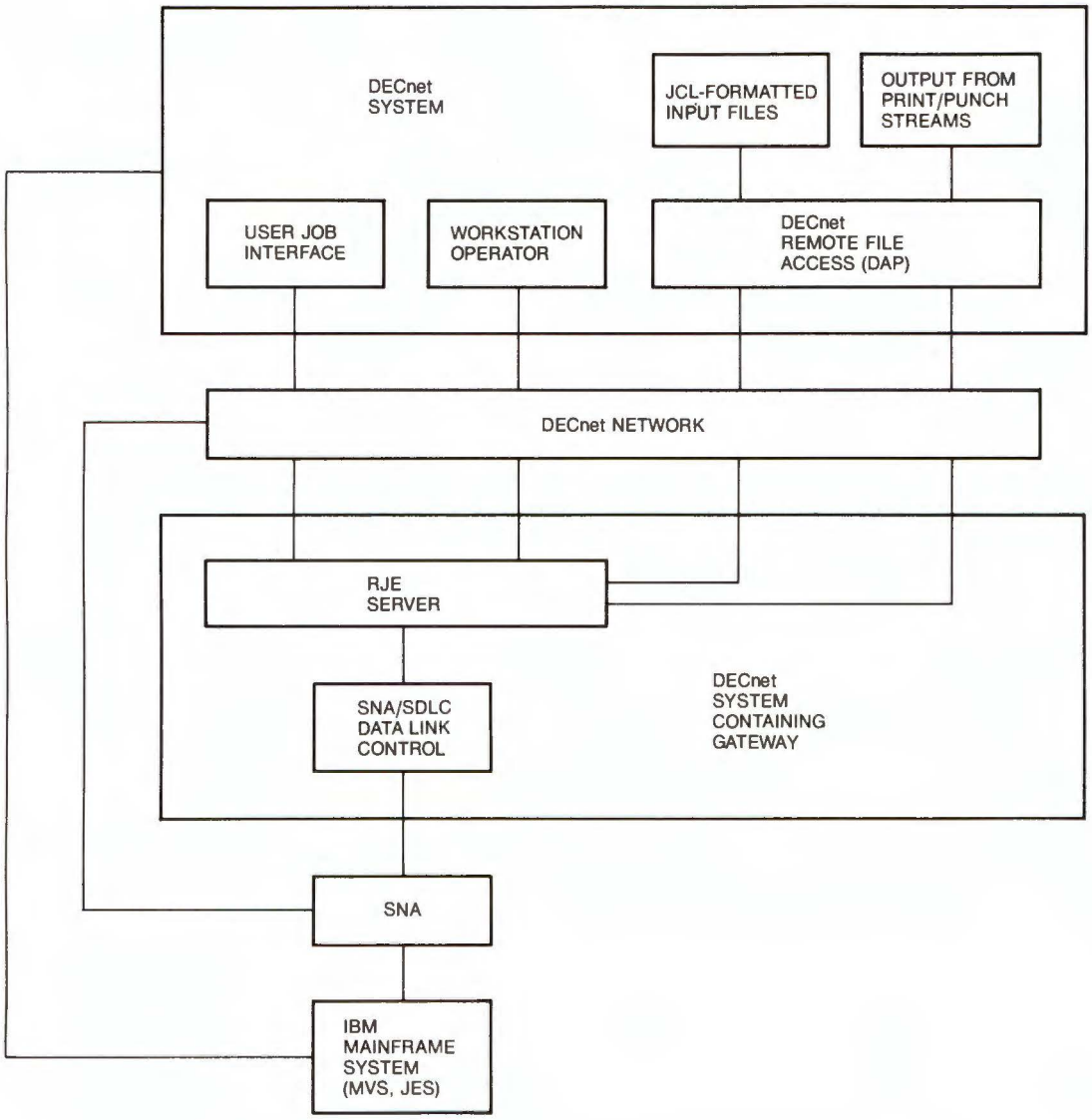


Figure 4 DECnet/SNA Remote Job Entry Overview

request/response unit (RU) is encapsulated within a DECnet message.[1] The gateway then removes all DECnet enveloping, adds the necessary path control and SDLC headers, and transmits SDLC information frames (I-frames) over the physical line to the front-end processor in the SNA network. The SNA network then delivers the RU to the mainframe application. Data flowing in the opposite direction follows the reverse path.

Given that each logical server could have its own discrete model associated with it, validating such a model would be an extremely time-consuming task.

Thus the final queuing network was represented by the DECnet server, the DECnet/SNA Gateway server (composed of the gateway CPU server and the SDLC line server), and the SNA server in tandem. The DECnet server and SNA server properties were analyzed in the context of particular products relative to their packet arrival/departure properties. Thus, the DECnet/SNA server analysis is essentially a processor simulation for the CPU in tandem with a line simulation for the associated SDLC line(s). This simplified relationship is shown in Figure 6. Details of these analyses are presented in the following section.

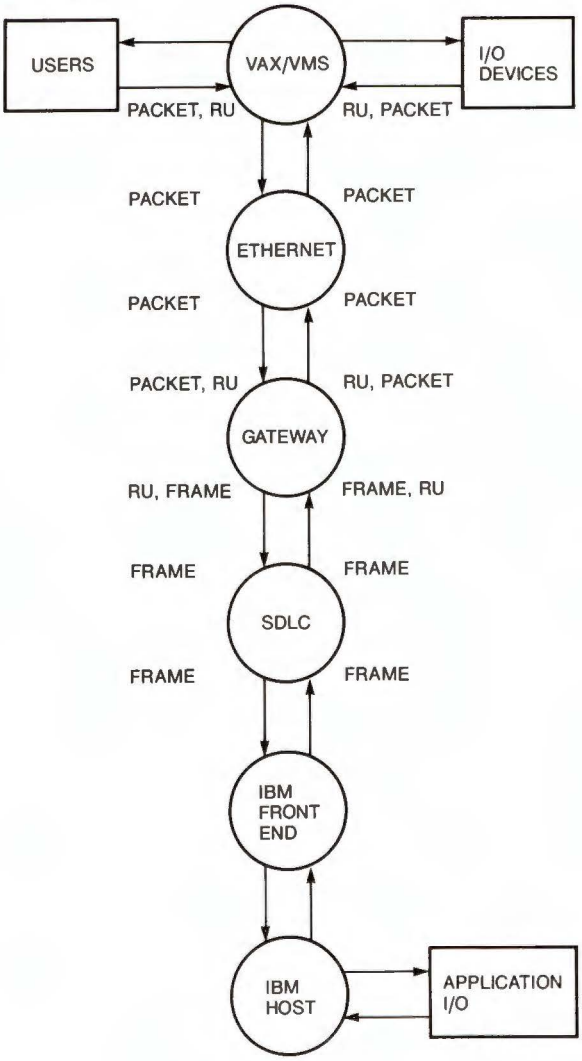


Figure 5 Queuing View of Model Environment

Processor and Line Server Simulation

In this section we cover the interaction between the two major functions provided by the gateway CPU. One function of the gateway CPU is to handle the real-time functions of the SDLC lines. The second function is to pass or modify data into and out of the SNA network. We discuss here the internal software components and interdependencies that were required to develop the simulation model of the gateway.

The DECnet/SNA Gateway supports three internal software servers — the GAS and RJE server processes, already discussed, and a host command facility (HCF) server process. The HCF server allows IBM 3270 terminals to function as command terminals when accessing a VAX/VMS host. Each of these software servers involved understanding the CPU utilization characteristics, with a limited multitasking environment in mind.

To keep the initial version of the model simple, we chose to model only the GAS and RJE servers.

The gateway CPU has been modeled as a single server providing service for three priority classes. In order of priority, these are

1. SDLC polling
2. GAS sessions
3. RJE sessions

Arrivals within a priority class are serviced on a first-come, first-served basis. The service discipline across the priority classes is preemptive resume. Since there can be at most one polling message per line in the system at any given time, a queue for that priority class is not needed. Although the multiple lines do require some coordinating mechanisms, each of the other two priority classes (GAS and RJE) has an associated queue. The model is flexible and allows the CPU service time distribution for any session to be exponential (the default), deterministic, or uniform. For validation purposes, we modeled the CPU as an exponential server.

The SDLC line is modeled as an alternating server servicing two queues; one is outbound (IBM to gateway), and the other is inbound (gateway to IBM). The number of frames served at any visit to a queue simulates the implementation. The time at which the server visits these queues depends upon the polling rate, the polling service time, and the amount of traffic. The service time distribution here is

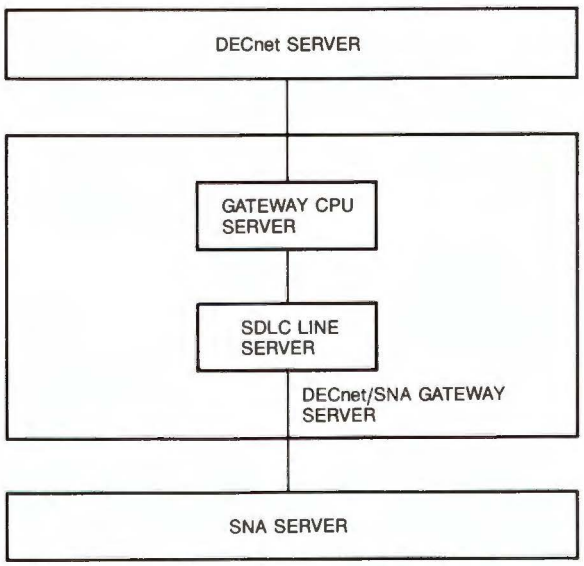


Figure 6 Model Environment, Simplified View

deterministic, in the sense that it is a linear function of the size of the frame to be transmitted.

The polling process is modeled in two states. The first state models the action of the IBM front end when no data (i.e., I-frames) is transmitted or received. In this state polls are sent to the gateway at periodical times (that is, deterministic rate). The second state models the action of the front end when data has been received from the gateway. In this state, the front end almost immediately polls the gateway again regardless of the deterministic rate. We also took into consideration the fact that the time to transmit a number of I-frames may overlap the periodical poll time, hence no poll could be sent. These situations are simulated in the gateway model as part of the line server.

The gateway CPU time needed to service a poll is modeled separately from the main CPU service. The separate models were created because gateway CPU time occurs at a rate determined by a front-end parameter (specifically, the advanced communication function/network control program [ACF/NCP] line-pause value). Further it exhibits particular properties based upon whether a productive or nonproductive poll occurred, as well as on how system scheduling properties can affect the effective throughput per poll.

The processor server simulation required additional analysis. Two major types of data flow needed to be considered in the simulation. One flow analyzed was the bulk data paradigm. This paradigm describes both the DTF and RJE products. In this paradigm data messages can arrive at the gateway as fast as the associated system can send them. To understand the effects of this type of session, we needed to saturate either the SDLC line or the gateway processor. Once this was done, the characteristics of the CPU server and line server in the gateway model were understood and the model developed.

The second type of data flow considered for the simulation was the interactive paradigm, as exhibited by the 3270 terminal emulator. In this paradigm, messages arrive at the gateway based nominally on the think time and processing time at the Digital system and the processing time at the IBM mainframe system. See the DECnet and SNA servers in Figure 6. The arrival rate at the gateway (hence the CPU and line server work) therefore is a function of both these additional system characteristics. For the interactive paradigm, these servers and their characteristics have been included in the model. Additionally response time as seen by the user is factored into the overall model's output. For those users of the model who do not know all the details for the DECnet and SNA servers, a set of heuristics has been employed to reflect a common set of defaults.

Performance Statistics

In a queuing model, the statistics of interest generally include server utilization, mean waiting time, and mean queue lengths. In the context of our work, the major servers of interest were the gateway CPU and the SDLC line. However, users of this particular model wanted to understand not only overall processor and circuit utilization, but also the utilizations attributable to individual sessions and the average session throughput across the gateway. Our model provides these performance statistics as well as the response times for the 3270 terminal emulator sessions.

Queuing versus Simulation Model

A major issue which arose when we began this work was whether the desired information could be supplied by a closed-form queuing model or by a simulation model. A solution using a closed-form queuing model is possible provided that enough is known about the DECnet and SNA servers in the general model. (See Figure 5.) However, we soon discovered a tractable solution is difficult to achieve. In this section, we discuss some of the reasons why we chose to develop a simulation model.

The management of polling for data-link-level I/O presented more than a few complications. In particular, the number of frames transmitted on the SDLC line on a poll cycle could vary depending upon whether the frame direction was inbound or outbound. Given this discontinuous behavior, a closed-form queuing formula could not be developed. The details in the message flow can be best illustrated by an example, which is provided below.

Assume that at the time the SDLC server starts servicing the inbound queue (i.e., when a poll arrives at the gateway), more than seven I-frames are waiting for service. According to modulo-8 SDLC protocol, only the first seven can be serviced at that particular time. The remaining frames have to wait for service at the next poll frame arrival. A similar scenario is followed for servicing the outbound direction.

Consider now the case in which service initiation begins and only four I-frames are waiting. In addition, new packets may be en route at the upper layers and could be eligible for service before completion of the current polling cycle. Due to the scheduling properties of the DECnet/SNA Gateway, however, only those four which are waiting would be serviced within that particular poll cycle. (Poll queue service is preemptive of user processing that would result in new queue entries.) All new arrivals therefore must wait until the start of the next cycle. (The above flow is not followed when the ACF/NCP dat-mode parameter has a value of full.)

This is not the procedure followed for the outbound flow from the IBM front end. New frames that arrive during a cycle are transmitted until the total number of frames transmitted is seven or until no more frames are to be transmitted, whichever occurs first. This approach is a function of the scheduling properties of the ACF/NCP software in the IBM front end. As such, this service represents a hybrid of the limited and gated service disciplines.

In addition to scheduling properties, other attributes of both the data link and higher levels affect the I-frame behavior. These attributes include the RU size of the session, I-frame message size (as specified by the ACF/NCP PU maxdata size), polling rate, pacing window size, and session paradigm (that is, either interactive or bulk transfer). All affect the steady-state RU rate, and so I-frame rate, as measured on the SDLC line. Consequently, our ability to tractably model data-link behavior in a closed-form model was diminished by the myriad of relevant variables.

In addition, session-level activity across the various server processes within the gateway for different session paradigms introduces more variables. Moreover, three server types — GAS, RJE, and HCF — operating at different priority levels affect both arrival rates and service times, particularly when all three server processes are concurrently active.

For these reasons, we chose a simulation over an analytical approach as the most realistic means of modeling actual behavior. However, even simulation modeling does not eliminate all variability. For example, gateway server processes behaved quite well for bulk transfers. However, for interactive traffic involving the 3270 terminal emulator, we had to account for the behavior of each component in the path rather than amalgamate multiple physical components into one virtual component. Thus, the end-to-end response time was critical in determining average arrival rates through the gateway. We could not assume a steady-state Poisson rate, as we could for bulk transfer.

Input to the Model

A major objective of the project was to build a model that was easy to use. Our goal was to require as little performance expertise as possible of the model user. The user therefore would need to know little more than how to install the product. Typically, the user would have to know the proper virtual terminal access method (VTAM) and ACF/NCP parameters as well as those parameters required by the target application subsystem on the SNA host.

Specific inputs to the model are as follows. The model is driven by a sequential file in which three major record types are present. The first record type describes general characteristics of the gateway as a whole and contains attributes such as DECnet buffer sizes and gateway CPU power. The second record type describes characteristics of the SDLC lines supported by the gateway to be modeled. These characteristics include associated speeds, polling rates, error rates, and segment sizes. The third record type describes the characteristics of each session in the modeled workload. These session characteristics include the associated SDLC line (from the second record type) over which this session traffic flows, the product that made use of the session, inbound and outbound RU sizes, and pacing window sizes.

Our principal objective is to allow the end user to define a static "snapshot" of the gateway's physical configuration and of the session workload. The model will then dynamically determine how well the gateway will support this workload on the defined configuration. The user has the freedom to either specify session arrival rates or accept the default. If the user chooses the default, intelligent heuristics within the model determine arrival rates.

Standard error and consistency checking is also performed within the model to ensure that a supplied workload mix can be executed.

Both algorithmic and rule-of-thumb techniques were used to determine overall workload feasibility.

We were able to provide particularly useful heuristics for obtaining data on CPU service time for a given session type. As a result of extensive internal measurement and careful curve fitting, we were able to derive linear equations of the form $A+B\times X$, where A is the base CPU utilization for a given RU, B is the CPU utilization per byte of RU processing, and X is the number of bytes in a given RU. Both A and B were measured in milliseconds. Validation testing has shown that for both the GAS and RJE servers, this form of equation is accurate to within 10 percent.

In addition to CPU service time, the model also determines effective arrival rates as a function of product type. An objective for the original design was to provide a "value-added" function that facilitated determination of arrival rates. With this function, most users of the model need not perform extensive arrival rate experiments and measurements to obtain meaningful data. In past work in modeling, performance engineering experience was required to obtain arrival rates, thus reducing the number of people who could obtain useful information from the model.

For bulk transfer sessions, an effective arrival rate is simulated using the internal buffer management algorithm in the gateway; for interactive traffic, the end-to-end traffic is totally simulated (assuming

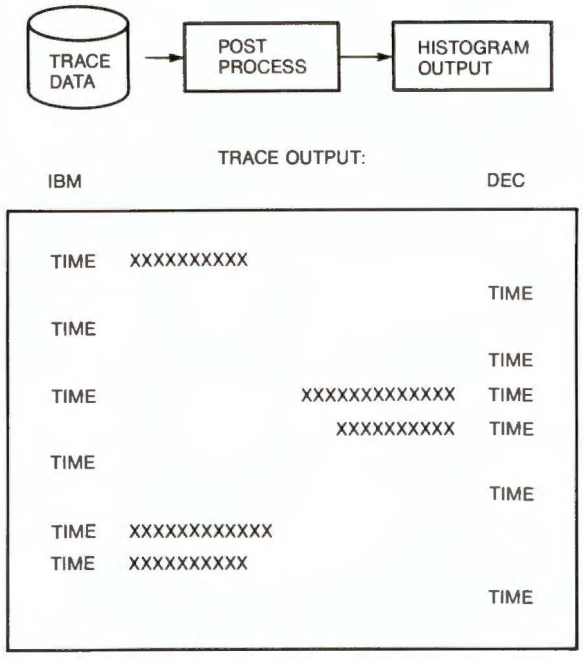


Figure 7 Histogram Representation of Frames

certain service time heuristics in some of the end-to-end path components such as the front-end processor). The section Validation of the Model presents actual output of the model and samples.

The model has been extensively tested against the TE, DTF, RJE, and DISOSS document exchange facility (DDXF) products for both single and multisession cases. For each product, more than 40 sets of measurements were performed to ensure overall consistency between measured and actual results. "Mixed server" experiments (with concurrent GAS and RJE activity) were also carried out. The error deviation between measured and actual results was generally 10 percent.

The following section further describes the tools used for load generation and measurement validation, and the test suites used.

Tools and Measurement Methodology

This section describes the tools and techniques we developed to deal with the operational issues which arose during the development of the model.

Histogram of Trace Data

During the initial measurement work, the results of experiments on the DDXF product were not consistent. This inconsistency highlighted our overall problem of being unable to identify the exact message flow patterns that result from a given experiment.

Test results were consistent for small subsets of experiments in which parameters were clustered together, but not for the entire body of experiments. Therefore, we could not validate the model for the experimental data across all experiments and attain our goal of 10 percent error margin between predicted and actual results. We decided that a histogram showing the size and pattern of messages would help us determine where the differences between the model predictions and the product tests existed.

To produce the histogram, we began with the gateway product. The gateway produces a trace of SDLC frames received from and transmitted to the IBM host. This trace is performed by a network application that creates a binary file of the data collected.

The trace was enhanced to include time stamps on each data message received by and transmitted from the gateway system. This additional information in the binary file allowed us to represent the trace data as a scaled line in a relative time position. The sample output represented in Figure 7 shows the times listed down both sides of the page. The data messages starting on the left are from the front-end processor (SDLC primary station); right-justified data messages are from the gateway (the SDLC secondary station). This output provided a visual representation of the pattern and size of messages for a given experiment. For example, it showed that for a 1024-byte RU sent to the IBM front-end processor using an SDLC frame size of 521 bytes, two frames would be transmitted. Whereas for a 1024-byte RU sent *from* the IBM front-end processor using 521 byte frames, three frames would be sent. This implementation detail of the front end seriously affected our interpretation of the circuit counters.

The histogram report also confirmed what patterns were being established for a set of experimental parameters. The histogram reports demonstrated that the front end, after receiving data from the secondary station, quickly polled this station again. We used the histogram tool to obtain reports on messages flows for a number of products. We were thus able to understand precisely the effects of parameter changes and to represent accurately the critical areas of processing in the model.

Remote Terminal Emulator Testing Environment

After conducting a number of tests by hand, we determined that a consistent and automated test environment was needed. The same type of data was needed from each of the tests.

- For the gateway, we required SDLC circuit counters and CPU utilization.

- For the system under test (SUT), with which the access product was run, we needed the DECnet counters and VAX CPU utilization measures.

The test environment would also have to accommodate another goal of the project which was to run a heterogeneous mix of products simultaneously. Also, because the different products had different startup times, interactive control was needed. With such control, we could coordinate experimental sample collection after a steady-state data flow was established.

With these requisites in mind, we chose to develop scripts using an internal remote terminal emulator (RTE). The RTE provides consistent control as a pseudo-interactive user. Figure 8 is an overview of the major components. The auxiliary VAX system is needed to collect the SNAP CPU utilization data from the gateway.[1] This data is transported by DECnet, and if sent to the system under test, the DECnet counters in the system under test do not represent solely the data that was specific to the product's steady-state data flow. The RTE also allows multiple systems to be used when multiple sessions are required; at the same time, the RTE is a single point of coordination for us.

The scripts needed for each product were written to interface with the common collection scripts. Thus scripts could be mixed and matched as required. Scripts were coordinated so that the startup time for each product was taken into account. Further, the scripts were constructed so that the major parameters for test, such as line name and RU size, could be passed through the RTE to the product being tested to create the test.

SNAP Server and Computation of CPU Utilization Time

As noted in the section above, the SNAP replacement tool that collects gateway CPU data saves the values in a file. A sample is received every second. The sample provides the number of kernel mode ticks (samples taken when in the base operating system), user mode ticks (samples taken when in a user task), and null ticks (samples taken when the CPU is idle). One tick equals one-sixtieth of a second. If K, U, and N represent the number of kernel mode, user mode, and null ticks, respectively, then the percentage of CPU utilization is given by

$$((K+U)/(K+U+N)) \times 100$$

Gateway CPU utilization obtained this way consists of three components:

- RU processing
- Poll servicing
- SNAP server processing

To calculate CPU utilization for processing the RUs, one must know the overhead for servicing the polls and the SNAP server utilization. We obtained these numbers through a set of tests that measured the CPU utilization of SNAP first with no other process running, then with the line being polled with SNAP. Together these tests resulted in the estimates of the overhead numbers that we used in subsequent analysis. The overhead can then be subtracted from the total CPU utilization to obtain the CPU utilization for processing the RUs.

The CPU service time per RU can be calculated by dividing this RU-specific CPU utilization by the arrival rate as determined by the circuit counters. The arrival rate is the average number of RUs per second, which we discuss further in the next section.

SDLC Counters and Arrival Rate

The SDLC counters for the gateway contain elapsed time, poll count, and error count. They also contain separate byte and frame counts for inbound and outbound messages. The arrival rate (RUs per second) can be calculated from the SDLC counters. The calculation is based on the fact that, under steady-state data flow conditions, throughput is the same as arrival rate.

To begin with, let us assume that for all the sessions in the same direction, the RU sizes are the same and the pacing window sizes are the same. Then, for

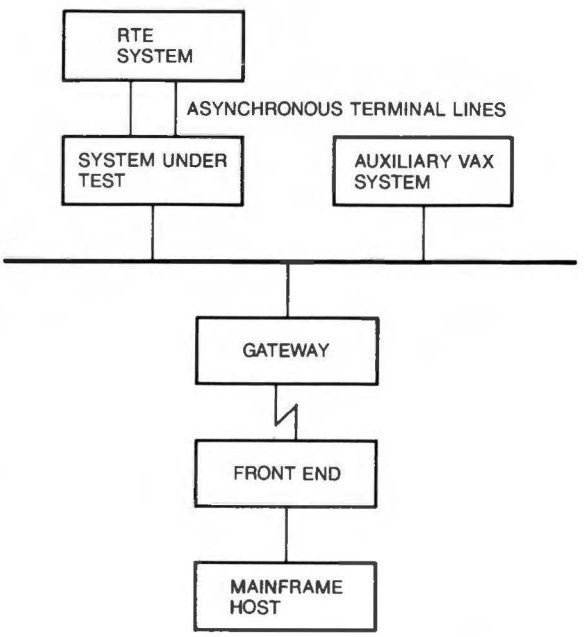


Figure 8 Overview of the Logical Testing Environment

unidirectional traffic, that is, data frames in one direction only, the arrival rate is given by the total data frames in direction of data divided by the number of segments per RU times the elapsed time in seconds.

In calculating arrival rate, we again assume that the RU size and the pacing window size are the same for all sessions in the same direction. When data is flowing in both directions, the calculation of the arrival rate is more complicated.

We have derived equations that allow us to calculate the arrival rate from the experimental data when homogeneous sessions are transmitting data in both inbound and outbound directions. The equations become geometric series that converge. The derivation of these equations is beyond the scope of this paper, but has been used to validate the correctness of the model.

If the sessions are not homogeneous, it will be difficult to compute the arrival rate from experimental data. However, the model (being a simulation) can be used to project this arrival rate and hence the throughput.

CATCH2

During work on the SNA gateway model, we were interested in the elapsed time taken by a message to travel through the gateway. The main flow of data is from the VAX to the IBM system, or from the IBM to the VAX system. Information about elapsed time is critical to understanding how well the gateway performs. We were also interested because we desired an independent method to validate the information gathered for the model. For the simulation to be successful with low data rates, we needed to know the elapsed time through the gateway with a resolution to parts of milliseconds. Without this level of information, errors generated when large numbers of sessions and fast lines are simulated could produce accumulated errors that would exceed our error margins.

In addition, queuing delays in the gateway could only be inferred from the measurements taken. We had already done work to time-stamp messages on the IBM line using a modified SNA trace routine, as noted above. However, the system clock in the gateway was accurate to only 16 milliseconds. Further, SNA trace measurements are intrusive; that is, as the SDLC line I/O rate goes up, the Ethernet traffic also increases and can more than double.

In summary, we obtained measurements on message traffic accurate to only about 1 millisecond by using statistical methods with the SNAP results. The methods being used to measure elapsed time through the gateway needed to be validated. In addition, we needed a greater understanding of the entire message flow, and we desired timing data measured to the level of 1 millisecond or less. Therefore a program, CATCH2, was written to supply this information.

The CATCH2 program has two functions:

- To monitor the lines and record messages on the disk
- To analyze the recorded data and display it for human interpretation

The problem was to capture messages from two different devices, time-stamp both sets of messages from a common clock, and record the messages and times in a single file for analysis. Figure 9 shows the hardware configuration and logical connections that were required for the solution.

The VAX CPU microsecond counter could measure the time in less than milliseconds. Experiments showed that the standard QIO driver could obtain reasonably accurate measurements by time-stamping messages as close as possible to interrupt level. We obtained the source code for the UNA and DMB32 drivers. The drivers were modified to use a VAX 8200 system to time-stamp messages at VMS fork level. At this level, individual messages were accessible to the driver code.

The problem of timing messages breaks down to two physical connections and a small set of software parameters. Given that we want to time messages sent from a VAX system on the Ethernet, through the gateway, to the IBM front-end processor, the steps are as follows:

- Set the UNA controller to the same address as the gateway. This setting allows the 8200 UNA to receive the same messages as the gateway.
- Monitor the receive side of the line into the IBM front end. The monitor connection is attached by a modified cable to a DMB32 controller on the tracing VAX system; modem signals to the DMB32 allow for message reception. (Data terminal ready is strapped to data set ready.)

Additionally, the DMB32 line must have the same SDLC hardware address as the gateway line.

This method could be extended to measure elapsed time through other systems such as routers. Any configuration in which messages are transferred through a system could be measured using this base technology.

Validation of the Model

The validity of the model is determined by comparing model results to some known results obtained by laboratory tests or real-world configurations using the same parameters.

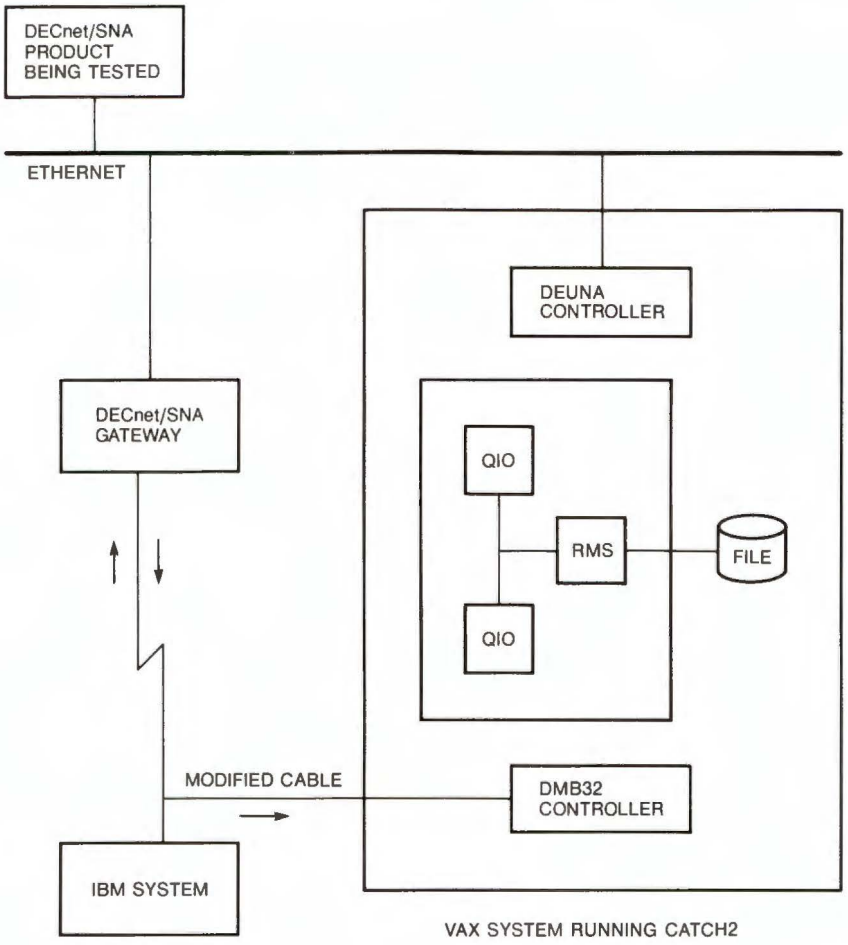


Figure 9 Overview of CATCH2 Logical Environment

Validation, in general, is a time-consuming and technically difficult task. The difficulties are often caused by the limitations and availability of the hardware. In the extreme case, the product to be modeled may still be in the design or development stage. In other cases, the amount of hardware required is not available; for example, it would be difficult, if not impossible, to set up a 1,000-node network to measure its performance.

For the validation of the DECnet/SNA gateway simulation model, we had two hardware limitations: the number of SDLC lines available (especially lines with speeds higher than or equal to 56 kilobits (Kb) per second); and the number and power of VAX CPUs available to act both as hosts and as remote terminal emulators to emulate multiple concurrent sessions under controlled conditions.

For these reasons, the maximum number of concurrent sessions in the validation were limited to four. Though this may appear to be a severe limitation, in practice it is not. The lines become saturated by four sessions with bulk data transfer.

Performance Metrics for Validation

From the experimental results, the performance metrics that can be easily computed are

- The utilization of the gateway CPU
- The utilization of the line
- Steady-state throughput of the gateway

The values of these statistics from the model output are compared to the values derived from experiments. The arrival rate and gateway CPU service time are input to the model along with other configuration parameters. These parameters include line speed, frame size, pause between polls, number of sessions, session type, RU size, and pacing value. These parameter values are taken directly from the parameters used in corresponding experiments.

Experimental versus Model Results

For the validation of the model, the calculated values of gateway CPU utilization and SDLC line utilization from measured results were compared to the values output from the model. Our goal was a margin of less than 10 percent difference between the two results, as mentioned above. Three categories were validated: bulk transfer/batch job (RJE and DTF), interactive data (the TE), and a mix of these applications. The results from the RJE and DTF validation are presented here in order to give the reader some feel for the relative accuracy of the model and to illustrate some performance properties of the DECnet/SNA Gateway itself.

Remote Job Entry Results A DECnet/SNA VMS RJE session is serviced by the RJE server in the gateway. This bulk data server provides the full functional transformation of data from IBM format to the DNA format. Since the RJE server in the gateway has to perform a full level-two through level-seven protocol conversion, its operation is CPU intensive. Therefore, we expected to find that the gateway was a bottleneck in most circumstances. As such, the throughput of the gateway remains relatively constant with different numbers of concurrent active streams. However, the time for finishing individual jobs will be longer when a larger number of concurrent streams are active.

Tables 1 through 3 compare the experimental results to the actual results for single RJE sessions supporting print, punch, and reader streams. In the actual experiments, we used a MicroServer Gateway with a line speed of 128K bits per second to connect to the IBM system. Record sizes for all three streams were varied between 40 and 132 bytes. MicroVAX II systems were used as DECnet hosts for the input and output transfers. As can be seen, the deviation between predicted and actual results was quite low. In addition, the results reflect the relatively high levels of gateway processor utilization attributable to performing the full protocol conversion of the RJE logical unit type 1 protocols to the DECnet data access protocol.

DTF Results To validate the bulk data transfer rate in the model using the GAS, we used the DECnet/SNA VMS data transfer facility. DTF provides steady-state

Table 1 RJE Validation Results for Print Stream

Record Size (Bytes)	Percent Gateway CPU Utilization			Percent Line Utilization		
	Experiment	Simulation	Error	Experiment	Simulation	Error
40	76.21	75.66	0.72	32.78	33.76	2.99
80	67.43	66.05	2.04	32.96	33.48	1.58
132	71.53	69.96	2.19	40.39	40.73	0.84

Table 2 RJE Validation Results for Punch Stream

Record Size (Bytes)	Percent Gateway CPU Utilization			Percent Line Utilization		
	Experiment	Simulation	Error	Experiment	Simulation	Error
40	72.74	72.28	0.63	32.41	33.4	3.37
80	75.06	73.79	1.69	39.31	39.89	1.48

Table 3 RJE Validation Results for Reader Stream

Record Size (Bytes)	Percent Gateway CPU Utilization			Percent Line Utilization		
	Experiment	Simulation	Error	Experiment	Simulation	Error
80	54.67	53.31	1.28	35.33	35.73	1.13
132	56.51	55.51	1.77	36.90	37.59	1.87

Table 4 DTF Multisession Validation Results for DECnet Hosts Transmitting to an IBM System

RU Size	Percent Gateway CPU Utilization			Percent Line Utilization		
	Experiment	Simulation	Error	Experiment	Simulation	Error
1024	11.20	10.96	2.14	89.10	88.85	0.28
2048	12.77	12.42	2.74	97.40	92.76	4.76
4096	9.81	9.73	0.81	92.53	93.14	0.66

Table 5 DTF Multisession Validation Results for an IBM System Transmitting to DECnet Hosts

RU Size	Percent Gateway CPU Utilization			Percent Line Utilization		
	Experiment	Simulation	Error	Experiment	Simulation	Error
1024	11.24	10.70	4.86	92.85	92.85	0.00
2048	10.69	10.08	5.70	93.37	92.82	0.56
4096	8.23	7.78	5.46	94.21	94.49	0.30

data traffic. For this product, the gateway CPU is generally not the bottleneck. Depending on the line speed and the availability of VAX CPU support to the concurrent sessions, either the SDLC line or the VAX host(s) becomes the bottleneck. For the validation, enough VAX CPUs were used so that the VAX host(s) never became the bottleneck. Thus, the SDLC line is the only resource that can be the bottleneck in the experiment, if at all. This fact is amply illustrated in the four session results for DTF shown in Tables 4 and 5. In these scenarios, four MicroVAX II DECnet hosts make use of inbound and outbound DTF sessions to an IBM mainframe over a 128K-bits-per-second circuit with different RU sizes. Note that the use of large packet sizes results in gateway operation that provides high line utilization, yet at the same time low overall usage of the gateway CPU. (The DTF model, unlike RJE, does not perform full protocol conversion within the gateway.)

Applicability to Mixed Applications The model simulates operation of three products: DTF, RJE, and TE. In addition, the model handles mixed RU sizes and a mix of any or all of these session types concurrently. Validation of mixed RU sizes and mixed applications is currently in progress.

Summary

A large number of parameters are involved in evaluating the performance of the DECnet/SNA Gateway, and hardware, time, and human resources are limited. It is therefore impossible to obtain experimental results for all configurations. However, for any specific configuration, the simulation model can provide expected performance results.

The user can easily run the simulation with different parameters and plot graphs to analyze the relationship of the different parameters. For example, one can plot the steady-state throughput in conjunction with RU size to find out how RU size affects throughput. Or one can plot TE-session delay in the gateway in conjunction with the number of concurrent users to analyze the "unacceptable" level of delay with increasing numbers of concurrent users. The results achieved may be used to plan system capacity or to develop system response time projections.

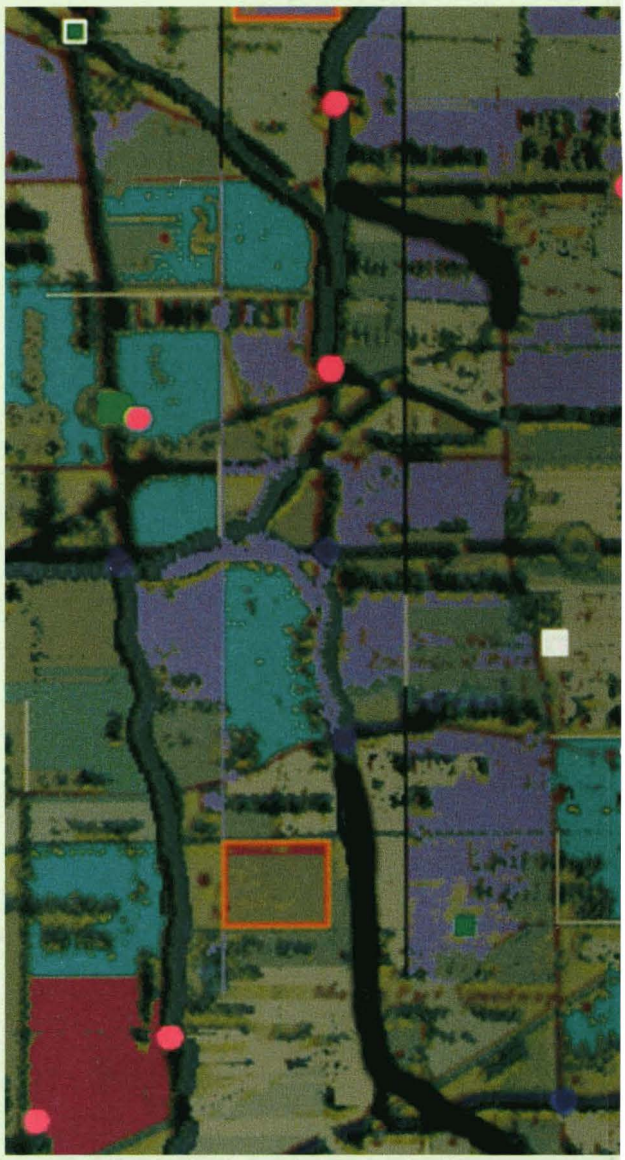
Acknowledgments

The authors would like to acknowledge the contributions of the following members of the NPACE Group without whose diligent efforts much of our success would not have been possible: Jane Morency, Atul Shrivastava, Rajan Subramanian, Carolyn Kay.

Reference

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

digital™



ISSN 0898-901X

Printed in USA EY-C179E-DP Copyright © June 1989 Digital Equipment Corporation