Introduction to DECnet (Phase III)

AA–AV51A–TK

May 1982

This document is an overview of the concepts and capabilities of DECnet networks. It defines DECnet terms and describes the network functions that DECnet implementations can perform. Readers are expected to be familiar with DIGITAL operating systems.

This document has been reprinted directly from the manual *Introduction to DECnet*, Order No. AA–J055C–TK. Only the title page, reader's comment form and mailer have changed.

This revised document supersedes the *Introduction to DECnet* (Order No. AA–J055B–TK).

Software and manuals should be ordered by title and order number. In the United States, send orders to the nearest distribution center. Outside the United States, orders should be directed to the nearest DIGITAL Field Sales Office or representative.

Northeast/Mid–Atlantic Region

Digital Equipment Corporation
PO Box CS2008
Nashua, New Hampshire 03061
Telephone:(603)884–6660

Central Region

Digital Equipment Corporation
Accessories and Supplies Center
1050 East Remington Road
Schaumburg, Illinois 60195
Telephone:(312)640–5612

Western Region

Digital Equipment Corporation
Accessories and Supplies Center
632 Caribbean Drive
Sunnyvale, California 94086
Telephone:(408)734–4915

digital equipment corporation • marlboro, massachusetts

First Printing, May 1982

The information in this document 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.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

The following are trademarks of Digital Equipment Corporation:

digital™

DEC
DECmate
DECsystem–10
DECSYSTEM–20
DECUS
DECwriter
DIBOL
MASSBUS
PDP
P/OS
Professional
Rainbow
RSTS
RSX
UNIBUS
VAX
VMS
VT
Work Processor

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

Contents

Chapter 4 Logical Links

Chapter 5 Task-to-Task Communication

Chapter 6 Remote File Access

Chapter 7 DECnet Terminal Facilities

Chapter 8 Network System Management

Chapter 9 Down-line Loading and Up-line Dumping

Chapter 10 Loopback Testing

Appendix A DECnet Documentation

Index

Figures

Tables

Preface

This edition of the *Introduction to DECnet* has been revised to reflect new versions of the following products:

RSX DECnet and *DECnet-VAX*

The Purposes of the *Introduction to DECnet*

Introduction to DECnet is an overview of the concepts and capabilities of DECnet networks. A DECnet network consists of two or more DIGITAL computer systems that have been linked together via communication lines. Within such a network, each system runs DECnet software, which, jointly with the networking hardware, enables communication with the other systems in the network.

Although all implementations of DECnet embody the same network concepts, all do not support the same set of network functions. The specific capabilities of a DECnet network depend on the types of systems participating and on the network's application. The objectives of this *Introduction* are:

- To describe the major network concepts behind all implementations of DECnet
- To define the specific network functions that DECnet provides
- To identify the DECnet implementations that support each function

The DECnet implementations discussed in this *Introduction* are:

- RSX DECnet (DECnet-11M, DECnet-11M-PLUS, DECnet-11S)
- DECnet-IAS
- DECnet-VAX
- DECnet-RT
- DECnet/E
- TOPS 20 DECnet-20

Chapter 1
What Is DECnet?

DECnet is a family of software products that enables two or more DIGITAL computer systems to form a network. Such a network can link computers that run the same operating system; more significantly, however, computers that run different operating systems can be linked via DECnet, as Figure 1–1 illustrates. The network shown consists of six systems or nodes, each of which runs a different DIGITAL operating system.

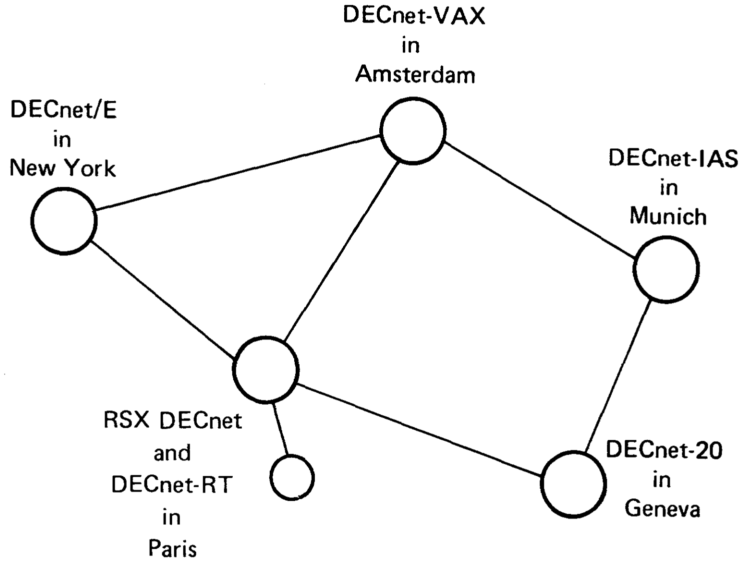
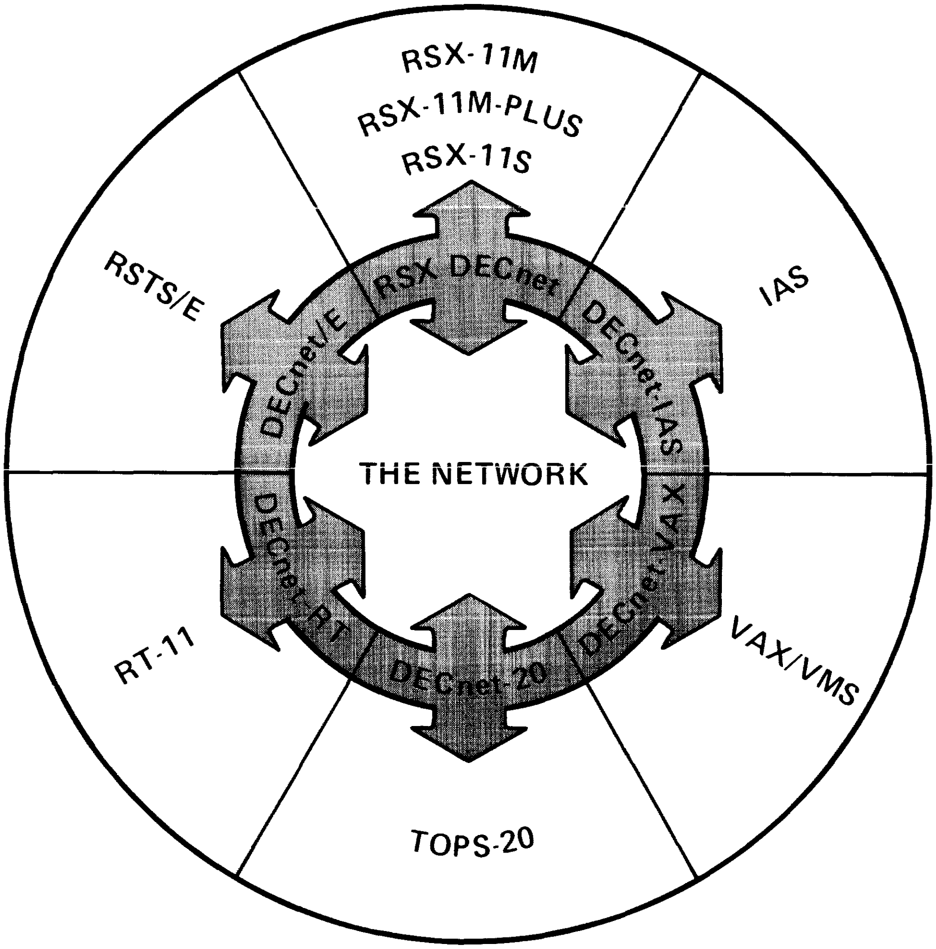


Figure 1–1: A DECnet Network

The DECnet implementation at each node acts as an interface between the node's operating system and the network (see Figure 1–2). On one hand, each implementation formats system-specific data and procedures according to common DECnet rules. All data traveling through a DECnet network has been formated in this way. Conversely, each recognizes the DECnet formats and converts them into formats recognizable to its own operating system.



Each implementation formats system-specific data and procedures according to common DECnet rules. Conversely, each implementation recognizes these DECnet formats and converts them into formats recognizable to its own operating system.

Figure 1–2: DECnet Implementations: Interfaces between Operating Systems and the Network

The ability to link different kinds of DIGITAL systems gives a great deal of flexibility to DECnet networks. A general characteristic of distributed processing is that 80% of computer resources are used to process local applications; work related directly to networking functions consumes only 20%. Therefore, every system within a network must be the right system for local requirements, or else considerable computer resources will be wasted.

DIGITAL offers a variety of operating systems designed for different types of applications and computers. Because DIGITAL also offers implementations of DECnet to extend most of these systems, users can match operating systems to local applications and then tie the various systems together with DECnet.

1.1 DECnet Functions

DECnet offers a wide range of networking functions. Some, like task-to-task communication, are provided throughout DECnet, while others can be supplied only by a subset of DECnet implementations. For example, loading a system image down-line to a satellite node is a function supported by RSX, IAS, VAX, and TOPS–20 DECnets, but not by DECnet/E or DECnet–RT. The following list identifies DECnet's major functions, which are discussed at greater length in the chapter or section as indicated. From these discussions, the reader will learn which implementations support the functions and what kind of facilities DECnet provides to perform them.

- **Task-to-Task Communication** (Chapter 5)

  DECnet enables two programs to exchange data over a logical link set up between them. The two programs can reside in the same or in different nodes.

- **Remote File Access** (Chapter 6)

  DECnet provides both terminal and program access to files that reside on remote nodes. Remote file access facilities allow users to perform the following operations:

  Transfer files between two nodes

  Manipulate files residing at a remote node, for example, open, delete, or append data to remote files

  Submit files containing operating system commands to a remote node in order to gain access to that node's resources

- **Terminal-to-Terminal Communication** (Section 7.1)

  A DECnet utility allows a terminal user to send messages to other terminals in the network.

- **Remote Terminal Facilities** (Section 7.2)

  DECnet allows a local terminal to be connected logically to a remote node, which then executes the commands typed at that terminal.

- **Network Management Facilities** (Chapter 8)

  DECnet provides facilities used by system managers to generate, define, monitor, and control network nodes.

- **Down-line Loading** (Chapter 9)

  An RSX–11S node, which has no disk storage of its own, can be loaded from an adjacent RSX, IAS, VAX, or TOPS–20 node. RSX DECnet, DECnet-VAX, DECnet-IAS, and DECnet-20 nodes also support up-line dumping; that is, if the RSX–11S system crashes, it automatically sends a system-image dump up-line to the adjacent node.

- **Loopback Testing** (Chapter 10)

  DECnet supplies tests that system managers can run to exercise various network capabilities and to isolate network problems.

When configuring a DECnet network, a system designer or manager takes into account the functions supported by each implementation. The range of functions that can be performed between any two network nodes is limited to the functions that they share. The network as a whole, however, is not limited to the functions common to all. The interaction between any two nodes is not determined by the capabilities of the other nodes in the network.

1.2 Using DECnet

DECnet provides user interfaces that are similar to those provided by DIGITAL's operating systems. To program task-to-task communication or remote file access, programmers use calls formated according to the operating system in which the program will run. Likewise, terminal users invoke DECnet utilities in a manner consistent with local operating system conventions. Therefore, using DECnet is similar to using purely local system functions. Nevertheless, network activity does entail varying degrees of complexity, depending on the type of work being performed and on the types of nodes within the network. For example, virtually all network functions involve cooperation between two programs. If both programs are user-written, as in task-to-task communication, a programmer must ensure not only that both programs run properly in their respective nodes, but also that the communication between them proceeds as intended.

Furthermore, although DECnet enables communication between nonhomogeneous systems, users performing such communication have to be aware of system differences. For example, DIGITAL operating systems support different file systems, a fact that has an effect on remote file access operations. Throughout the *Introduction*, discussions highlight circumstances in which users must take differing system characteristics into account.

The Readers of the *Introduction*

The *Introduction* is intended for readers who want to learn about the concepts and capabilities of DECnet systems. It assumes that the reader is familiar with DIGITAL operating systems but not with DECnet concepts or terms. Typical readers will include the personnel at the site of a newly installed DECnet system, who can read this manual to learn about the kind of work DECnet enables them to perform. Another group of readers will include system managers and designers who are thinking about using DECnet to expand their existing DIGITAL computer systems. And, finally, the *Introduction* is also intended for system managers and designers who do not yet use DIGITAL systems but who are considering the implementation of a computer network. This manual will inform them about the network capabilities that DECnet provides.

The Structure of the *Introduction*

The *Introduction* contains ten chapters, which can be divided into three parts:

- The first part, Chapters 1 to 4, introduces the concepts and the general uses for DECnet.

- The second part, Chapters 5 to 7, defines specific network functions and explains the mechanisms that programmers and terminal users can employ to implement those functions.

- The third part, Chapters 8 to 10, introduces DECnet functions relating to the management of the systems (called nodes) that make up a DECnet network.

Associated Documents

This *Introduction* discusses many topics that are explained in greater detail in other manuals. Appendix A lists, by implementation, all the DECnet manuals and their order numbers.

Chapter 2
The DIGITAL Network Architecture

The DIGITAL Network Architecture (DNA) is the model for all DECnet implementations and the standard that allows different DIGITAL operating systems to participate in the same network. This chapter highlights features of DNA to illustrate some basic concepts of DECnet. DNA consists of layers, each of which defines a distinct set of network functions and the rules for performing them. Accordingly, each DECnet implementation consists of software modules that perform these layered network functions as DNA dictates.

The DNA model also allows for implementation of the Comité Consultatif International Télégraphique et Téléphonique (CCITT) recommendation X.25. This recommendation defines a standard interface from a computer or a terminal to a public packet switching network (PPSN). DNA also defines a software interface, called Data Link Mapping (DLM), that creates a bridge between DECnet and X.25 implementations residing in the same node. DLM enables a DECnet node that includes an X.25 implementation to communicate through a PPSN to another DECnet node. (RSX DECnet includes a DLM interface to RSX–11 PSI [Packetnet System Interface], DIGITAL's implementation of the X.25 recommendation for RSX–11 systems.) Section 3.4 discusses PPSNs and the DLM interface.

2.1 The DNA Layers

Figure 2–1 illustrates the DNA functional layers. The following definitions summarize the purpose of each layer. For detailed information about DNA, see the *DIGITAL Network Architecture General Description* and other manuals that are listed in Appendix A.

Each layer defines a distinct set of functions as well as rules for implementing those functions.

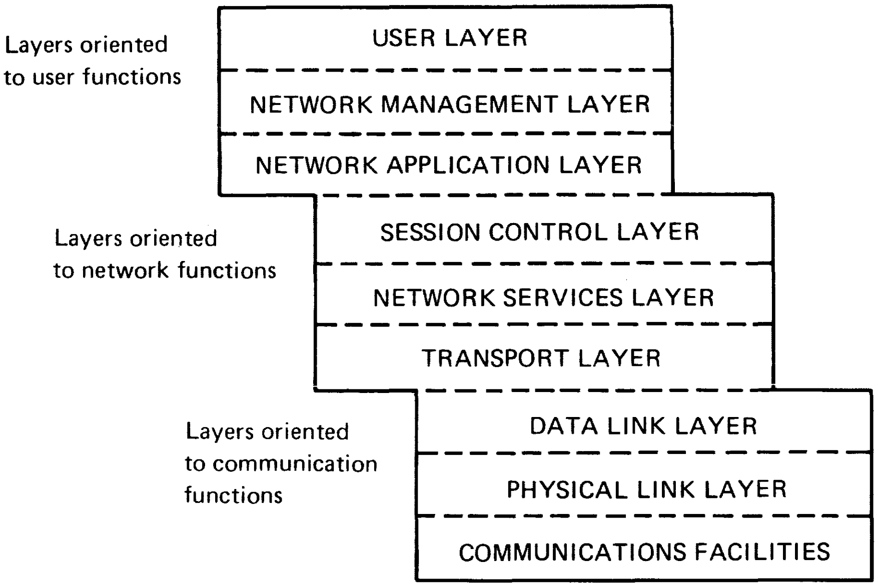

Layers oriented to user functions
USER LAYER
NETWORK MANAGEMENT LAYER
NETWORK APPLICATION LAYER
Layers oriented to network functions
SESSION CONTROL LAYER
NETWORK SERVICES LAYER
TRANSPORT LAYER
Layers oriented to communication functions
DATA LINK LAYER
PHYSICAL LINK LAYER
COMMUNICATIONS FACILITIES

Figure 2–1: The DIGITAL Network Architecture

Definitions of DNA Layers

- **User Layer.** The User layer encompasses user-written programs and services that access the network. It is the highest layer in the architecture.
- **Network Management Layer.** The Network Management layer defines the functions used by operators and programs to plan, control, and maintain the operation of DECnet networks.
- **Network Application Layer.** The Network Application layer defines network functions used by the two higher layers. The most important DECnet functions currently operating within this layer are remote file access, file transfer, and the remote terminal capability.
- **Session Control Layer and Network Services Layer.** Together these layers define a mechanism that allows a program in one node to communicate with a program in another node regardless of either program's location within the network. Modules in the User layer, Network Management layer, and Network Application layer can all use the mechanism provided by the Session Control and Network Services layers. This mechanism, called the logical link, is discussed in Chapter 4.
- **Transport Layer.** The Transport layer defines a mechanism for transporting a unit of data from one node to a specific node elsewhere in the network.

- **Data Link Layer.** The Data Link layer defines a mechanism for error-free communication between adjacent nodes. This layer is independent of communication device characteristics.
- **Physical Link Layer.** The Physical Link layer encompasses the software device driver for each communications device plus the communications hardware itself. The hardware includes interface devices, modems, and the communication lines.

2.2 DECnet Module Interfaces

DNA defines the interfaces between DECnet software modules operating within the same node. Reflecting the structure of DNA, each module can communicate with modules in a higher or a lower layer, but not with another module in the same layer. Using these vertical interfaces, each module uses the services provided by a module in a lower layer (see Figure 2–2). (DNA does not allow a module to use any services provided by a higher level module.) In building-block fashion, the modules in each layer support higher level modules by providing them with required network services.

Figure 2–2 illustrates a collection of modules residing in a typical DECnet node. The arrows represent the interfaces between modules. The arrows point down because each module uses the services provided by a module in a lower layer; a module cannot use services provided by a higher level module.

2.3 DNA Protocols

So far, DNA has been viewed in the context of an individual node. However, in addition to defining vertical interfaces, DNA also defines the relationship between modules in separate nodes: A module in one node communicates only with an equivalent module in another node, where *equivalent* means resident in the same layer and serving the same network function.

Communication between equivalent modules is governed by a set of rules called a protocol. Each protocol defines the form and content of messages to be exchanged by modules residing in the same layer, but in separate nodes. Equivalent modules use the same protocol.

Protocols for modules in the higher layers are more complex than protocols for lower layers. For example, a Physical Link layer protocol is defined in terms of electrical signals; whereas a protocol for modules residing in the Network Application layer defines message formats and rules for exchanging messages.

Figure 2–3 illustrates protocol communication between equivalent modules in separate nodes. Table 2–1 lists and briefly describes the function of each DNA protocol.

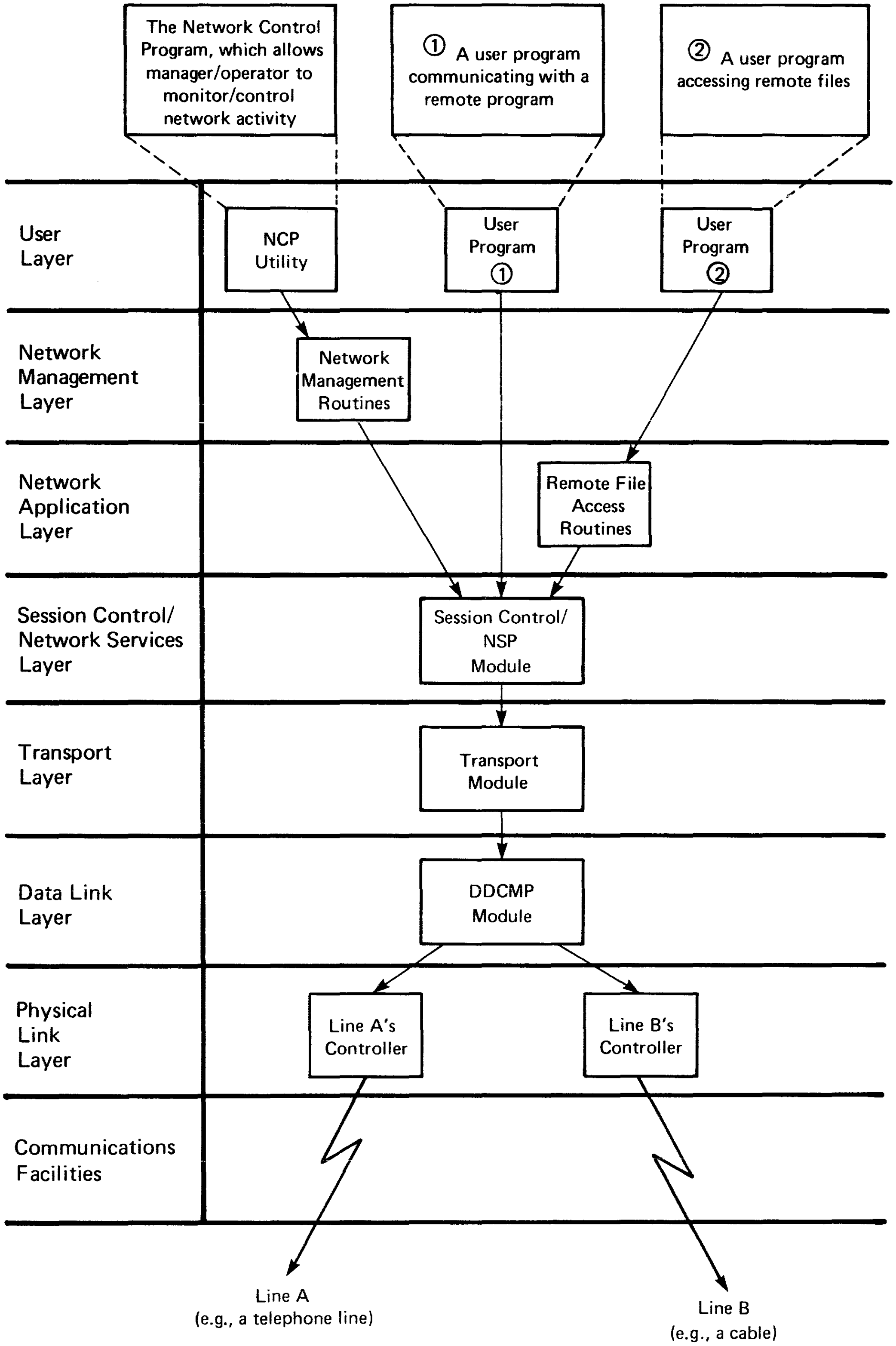

The Network Control Program, which allows manager/operator to monitor/control network activity
① A user program communicating with a remote program
② A user program accessing remote files
User Layer
NCP Utility
User Program ①
User Program ②
Network Management Layer
Network Management Routines
Network Application Layer
Remote File Access Routines
Session Control/ Network Services Layer
Session Control/ NSP Module
Transport Layer
Transport Module
Data Link Layer
DDCMP Module
Physical Link Layer
Line A's Controller
Line B's Controller
Communications Facilities
Line A (e.g., a telephone line)
Line B (e.g., a cable)

Figure 2–2: Vertical Interaction of DECnet Software Modules

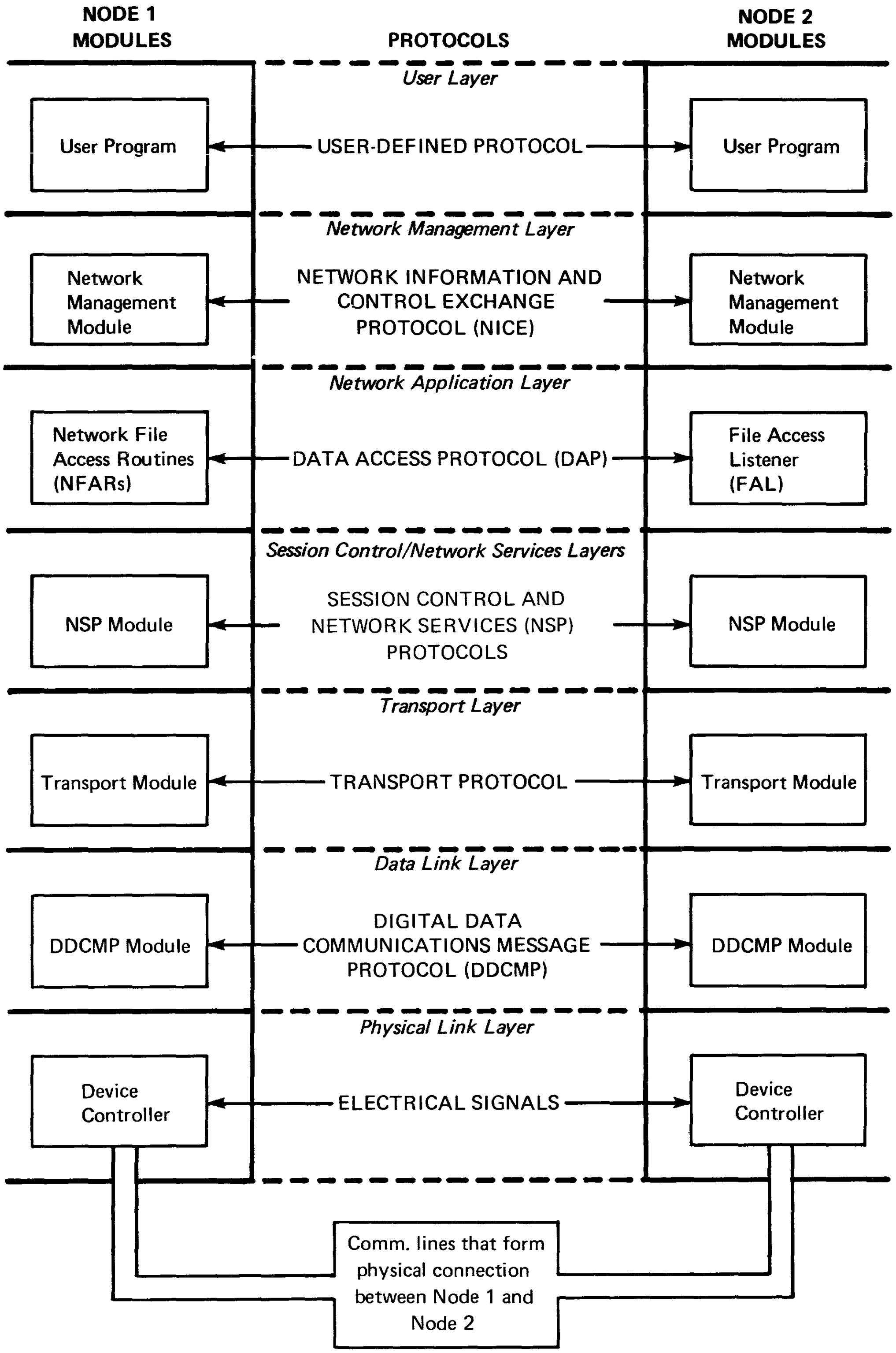

NODE 1 MODULES
PROTOCOLS
NODE 2 MODULES
User Layer
User Program
USER-DEFINED PROTOCOL
User Program
Network Management Layer
Network Management Module
NETWORK INFORMATION AND CONTROL EXCHANGE PROTOCOL (NICE)
Network Management Module
Network Application Layer
Network File Access Routines (NFARs)
DATA ACCESS PROTOCOL (DAP)
File Access Listener (FAL)
Session Control/Network Services Layers
NSP Module
SESSION CONTROL AND NETWORK SERVICES (NSP) PROTOCOLS
NSP Module
Transport Layer
Transport Module
TRANSPORT PROTOCOL
Transport Module
Data Link Layer
DDCMP Module
DIGITAL DATA COMMUNICATIONS MESSAGE PROTOCOL (DDCMP)
DDCMP Module
Physical Link Layer
Device Controller
ELECTRICAL SIGNALS
Device Controller
Comm. lines that form physical connection between Node 1 and Node 2

Figure 2-3: Protocol Communication between Equivalent Modules

DNA does not define protocols for all functional layers. For example, the User layer programs communicate over the network according to rules defined by the programmer. Furthermore, more than one protocol can be defined for the same layer because some layers support more than one function. For example, the Network Application layer can include modules that use the Data Access Protocol (DAP) as well as modules that use a protocol defined by users for a specific network application (transaction processing, for example). The protocols that DNA does define are also not exclusive; users can substitute their own protocols as long as they are implemented consistently by equivalent modules throughout the network.

Table 2-1: DNA Protocols

Protocol	Layer	Description
NICE	Network Management	The Network Information and Control Exchange protocol defines mechanisms for exchanging network, node, and configuration data, and for servicing requests from modules residing in the Network Management layer.
DAP	Network Application	The Data Access Protocol defines mechanisms for performing remote file access and remote file transfer on behalf of software modules residing in the Network Management layer (Phase III only) and the User layer. See Chapter 6.
NSP	Network Services	The Network Services Protocol defines a mechanism for creating and maintaining logical links between higher level modules residing in the same node or in different nodes.
Routing	Transport	The routing protocol defines a mechanism for dispatching data to any node in the network by the best possible route. This protocol is implemented in Phase III products only. See Section 2.5 and Chapter 3.
MOP	Data Link	The Maintenance Operation Protocol defines mechanisms for transmitting data over a communications channel to achieve specific functions: down-line loading of a remote node; up-line dumping from a remote node; testing a node and network connections; and starting up an unattended remote node.
DDCMP	Data Link	The Digital Data Communications Message Protocol defines a mechanism for ensuring the integrity and sequentiality of data transmitted over a communications channel.

This manual refers only to DNA-defined protocols because they are standard for all DECnet implementations. Some protocols are discussed in relation to the functions they serve. The Data Access Protocol, for example, is discussed in relation to remote file access. Further information about individual DNA protocols can be found in other DIGITAL publications. See Appendix A.

2.4 Relaying User Data through the Network

Before leaving its source node, user data travels down through the layers defined by DNA. A module in each layer adds control information to the unit of data it receives from above. This control information is dictated by the module's protocol. Ultimately, the user data reaches the Physical Link layer and is transmitted in a multisegmented envelope of protocol. Figure 2–4 illustrates how each module adds its protocol to the unit of data it receives from above.

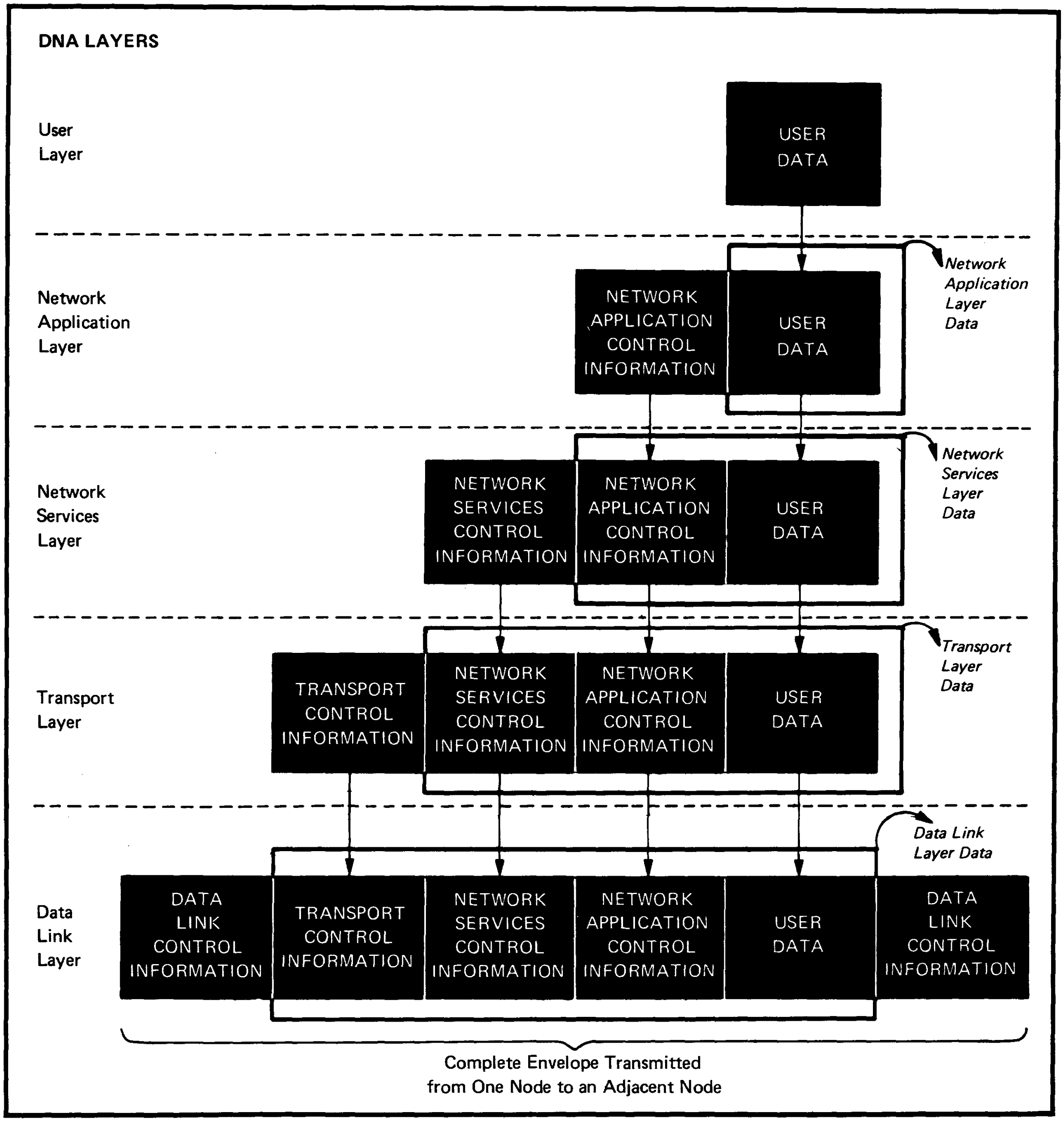

DNA LAYERS
User Layer
USER DATA
Network Application Layer
NETWORK APPLICATION CONTROL INFORMATION
USER DATA
Network Application Layer Data
Network Services Layer
NETWORK SERVICES CONTROL INFORMATION
NETWORK APPLICATION CONTROL INFORMATION
USER DATA
Network Services Layer Data
Transport Layer
TRANSPORT CONTROL INFORMATION
NETWORK SERVICES CONTROL INFORMATION
NETWORK APPLICATION CONTROL INFORMATION
USER DATA
Transport Layer Data
Data Link Layer
DATA LINK CONTROL INFORMATION
TRANSPORT CONTROL INFORMATION
NETWORK SERVICES CONTROL INFORMATION
NETWORK APPLICATION CONTROL INFORMATION
USER DATA
DATA LINK CONTROL INFORMATION
Data Link Layer Data
Complete Envelope Transmitted from One Node to an Adjacent Node

Figure 2–4: Enveloping User Data in Protocol

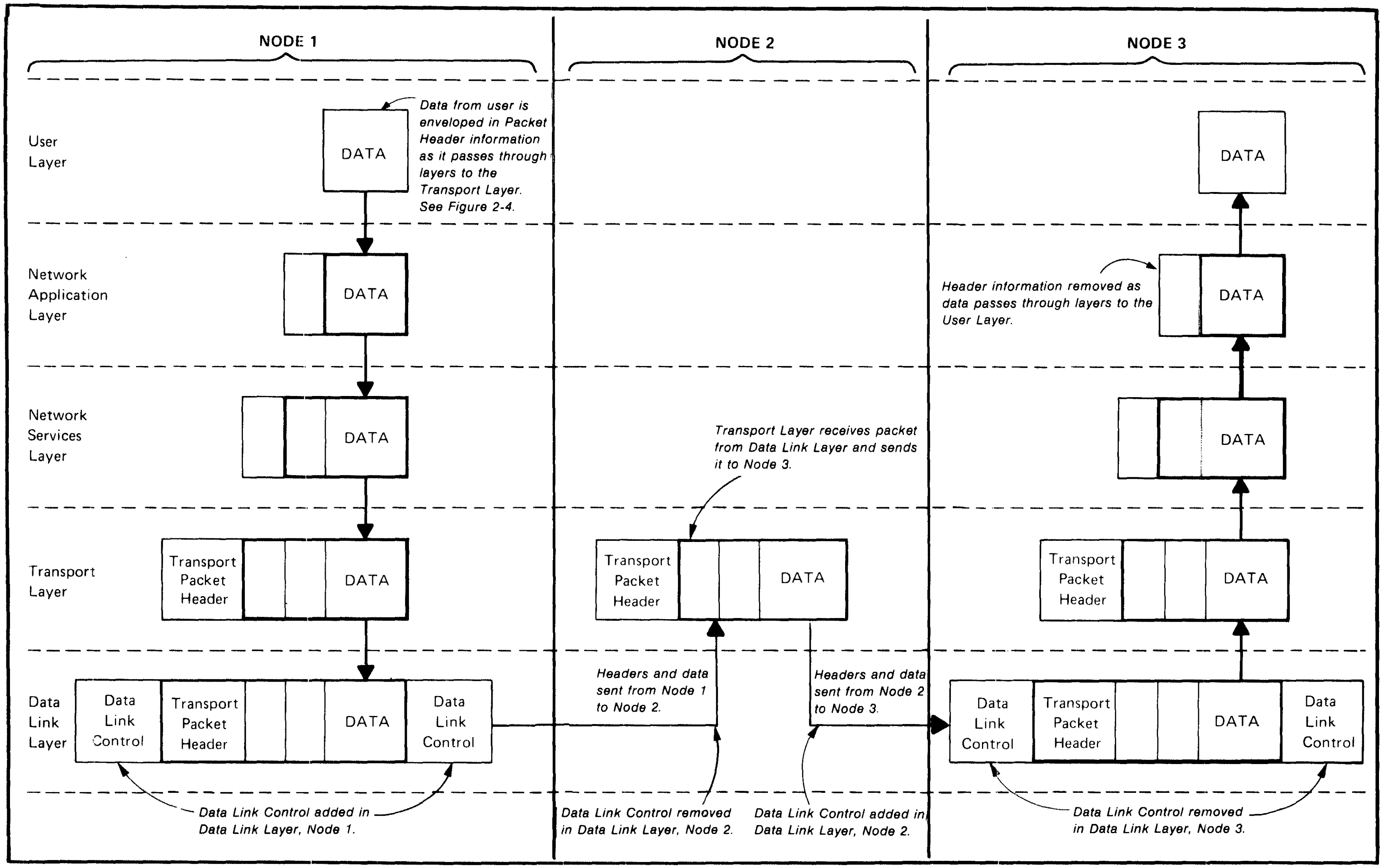
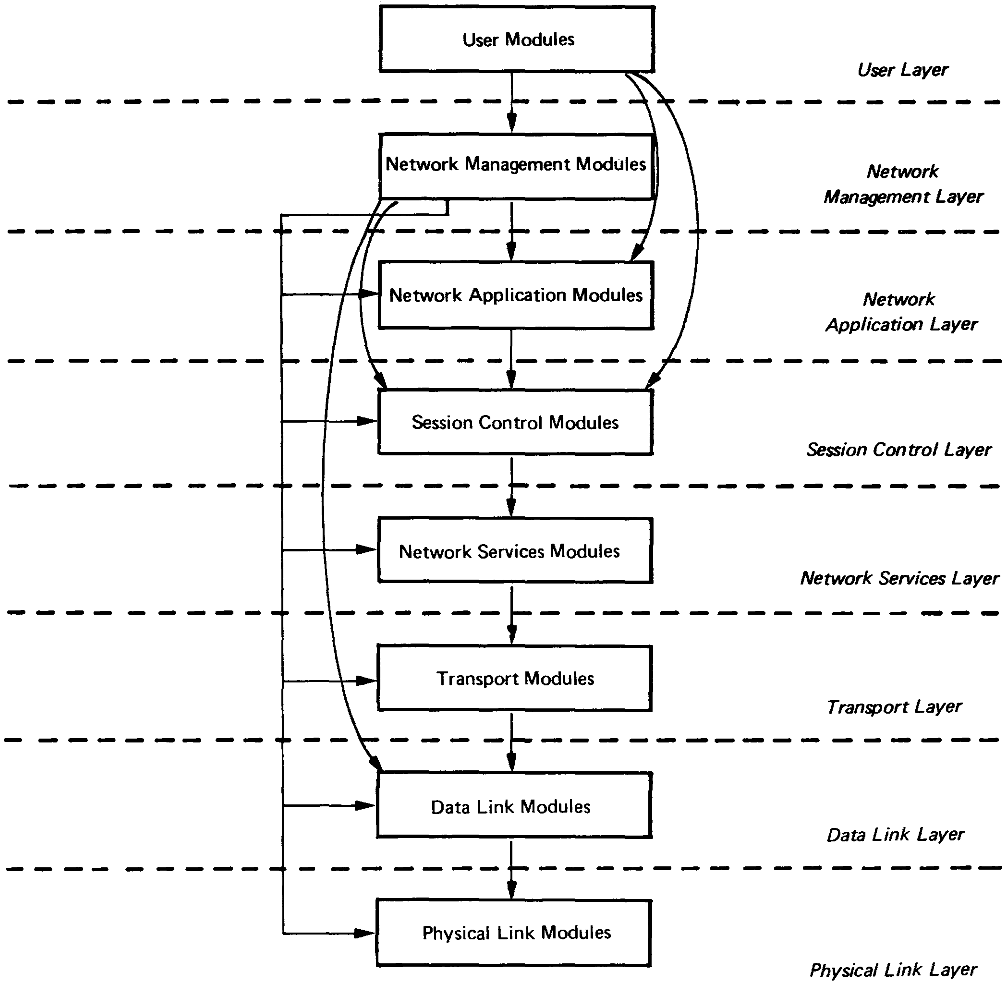

NODE 1
NODE 2
NODE 3
User Layer
Network Application Layer
Network Services Layer
Transport Layer
Data Link Layer
DATA
Transport Packet Header
Data Link Control
Data from user is enveloped in Packet Header information as it passes through layers to the Transport Layer. See Figure 2-4.
Data Link Control added in Data Link Layer, Node 1.
Transport Layer receives packet from Data Link Layer and sends it to Node 3.
Headers and data sent from Node 1 to Node 2.
Headers and data sent from Node 2 to Node 3.
Data Link Control removed in Data Link Layer, Node 2.
Data Link Control added in Data Link Layer, Node 2.
Header information removed as data passes through layers to the User Layer.
Data Link Control removed in Data Link Layer, Node 3.

Figure 2–5: Data Flow from Node 1 to Node 3 in a Three-Node Network

Each segment of protocol represents one module talking to an equivalent module elsewhere in the network. When the data envelope arrives at its destination node, each module reads and strips off the appropriate protocol segment and then hands the remaining data up to the next module. Figure 2–5 illustrates the process of adding and subtracting protocol as the user data travels from its source to its destination.

Note the action taken by the Transport module when the data needs to be forwarded or routed to another node. The routing mechanism is described in Chapter 3.



Black arrows show direct access for control and examination of parameters, counters, etc. Red arrows show interfaces between layers for normal user operations such as file access, down-line load, up-line dump, end-to-end looping and logical link usage.

Figure 2–6: Network Management: Relation to DNA

2.5 Differences between Phase II and Phase III DNA

DECnet products and the architecture on which they are based have been evolving since 1973. The basic layered structure of DNA has not changed, but new layers and protocols have been added and existing ones refined. The evolution of the architecture has coincided with the development and release of new DECnet products that incorporate up-to-date networking techniques and concepts.

The first version of DNA and the first releases of DECnet products were known as DECnet Phase I. Subsequent revisions of DNA and new releases of DECnet products are categorized as DECnet Phase II and III. Phase I products cannot participate in the same network with Phase II and Phase III products. However, with the exception of DECnet–RT, products implementing both Phase II and Phase III architecture are compatible within the same network.

The DECnet implementations described in this manual are either Phase II or Phase III products. The differences between these two product categories reflect the following changes to DNA:

- **The Addition of the Network Management Layer.** This layer lies between the User layer and the Network Application layer. Unlike most other layers, it has interfaces defined not only for adjacent layers but also for every other layer in the architecture, as Figure 2–6 illustrates. The multiple interfaces meet the special requirements of network system management. The functions defined by this layer allow system managers to oversee, control, maintain, and test all major facets of a network node. Phase II products also supported such functions, but in a system-dependent manner rather than in a manner dictated by DNA. See Chapter 8 for a detailed discussion of network management.
- **The Revision of the Transport Layer to incorporate a routing mechanism more sophisticated than point-to-point.** Although Transport was defined as a separate layer in Phase II DNA, the point-to-point routing mechanism it defined was sufficiently simple to be implemented by the Network Services module. Chapter 3 describes Phase II and Phase III routing mechanisms in greater detail.
- **The Addition of the Session Control Layer.** The Session Control layer defines local-node aspects of logical link management, whereas the Network Services layer handles the actual creation and management of logical links, which are discussed at length in Chapter 4. However, Phase III implementations include the Session Control functions within the same modules that perform the Network Service functions. In consequence, the management of logical links by Phase II and Phase III implementations is essentially the same.

Chapter 3
DECnet Routing

Routing is the function that determines the physical path or route along which data travels to its destination. In the context of routing, the unit of data to be transported is called a packet. Routing methods vary depending on the DECnet implementations operating within the network.

- Phase II implementations support point-to-point routing, which allows a node to send packets to physically adjacent nodes only (see Figure 3–1). Adjacent nodes are linked directly by a physical line.
- Phase III implementations support full routing, which allows one node to send packets to any other node in the same network. The source and destination nodes do not need to be adjacent because each packet is routed through any nodes that fall in between them (see Figure 3–2).
- A Phase III RSX DECnet node that also includes the RSX–11 PSI (Packetnet System Interface) product can route packets via a public packet switching network (PPSN) to remote DECnet nodes.

Sections 3.1 through 3.3 discuss point-to-point and full routing. Section 3.4 discusses the RSX DECnet/PSI interface to a PPSN.

3.1 Phase II and Phase III Configurations

Routing capabilities affect the configuration of DECnet networks. Point-to-point routing means that the most useful Phase II DECnet configurations are hierarchical or star-shaped; examples of such configurations are shown in Figure 3–3. In contrast, Phase III configurations do not have to be so formally structured since nonadjacent Phase III nodes can communicate directly with one another.

Node B can exchange data with Nodes A and C, but Nodes A and C cannot exchange data with each other.

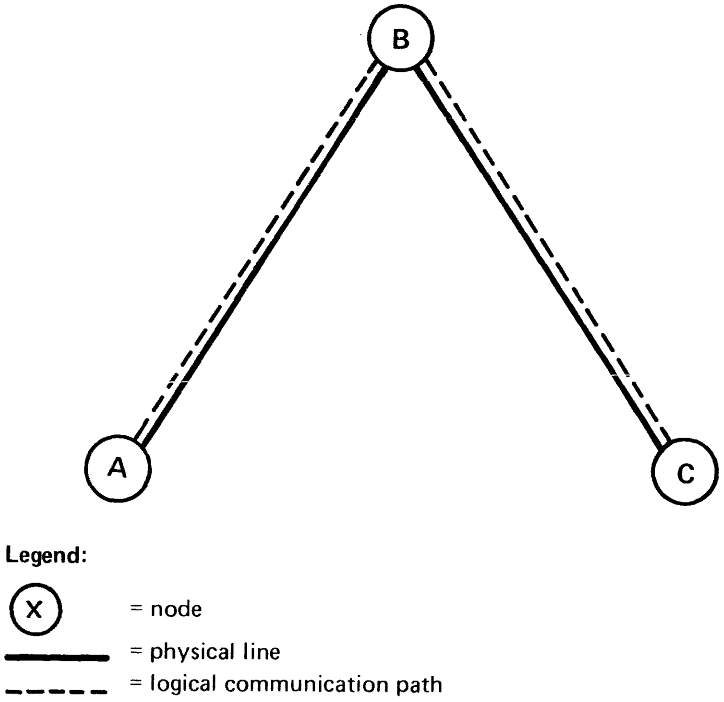


Figure 3–1: Phase II Point-to-Point Routing

A Phase III network can include three types of nodes, which support different degrees of routing function. A node's type determines its position within a Phase III configuration.

- **Routing nodes.** A routing node can forward packets to other nodes in the network and can be adjacent to all other types of nodes.
- **Nonrouting nodes.** A nonrouting node can send packets to other nodes in the network but packets cannot be forwarded or routed through it. It can be adjacent to one other node only; therefore it is always an end node in a Phase III configuration. For example, a DECnet-RT node is always a nonrouting end node because it has only one physical link to the network.
- **Phase II nodes.** A Phase II node is a node that runs a Phase II implementation of DECnet and therefore does not support full routing. It can send packets to adjacent nodes only and it cannot forward packets it receives onto other nodes in the network. It can be adjacent to one or more full routing nodes and/or to other Phase II nodes. Logically, it is an end node within a Phase III configuration.

Figure 3-4 illustrates configurations of nodes that support Phase III full routing. Figure 3-5 illustrates a Phase III configuration adjacent to a subnetwork of Phase II nodes.

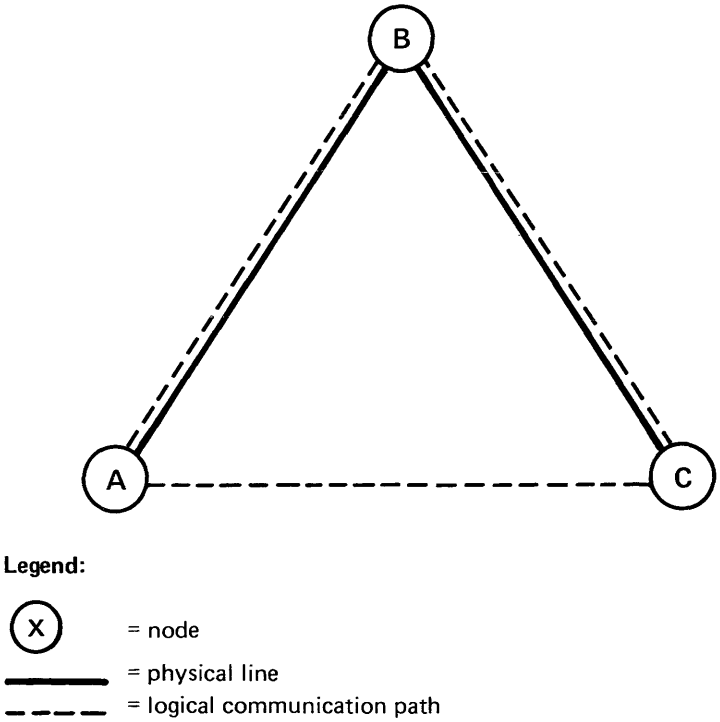
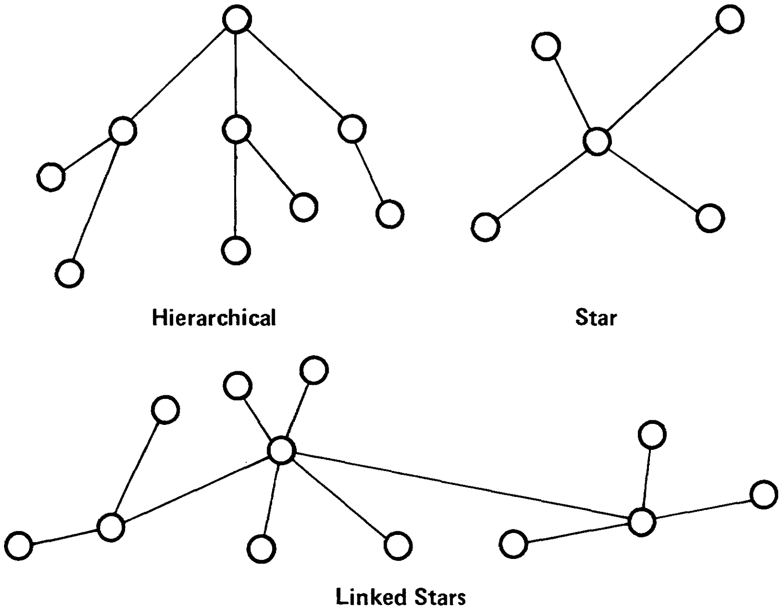


Figure 3–2: Phase III Full Routing


Hierarchical

Star

Linked Stars

Figure 3–3: Phase II DECnet Configurations

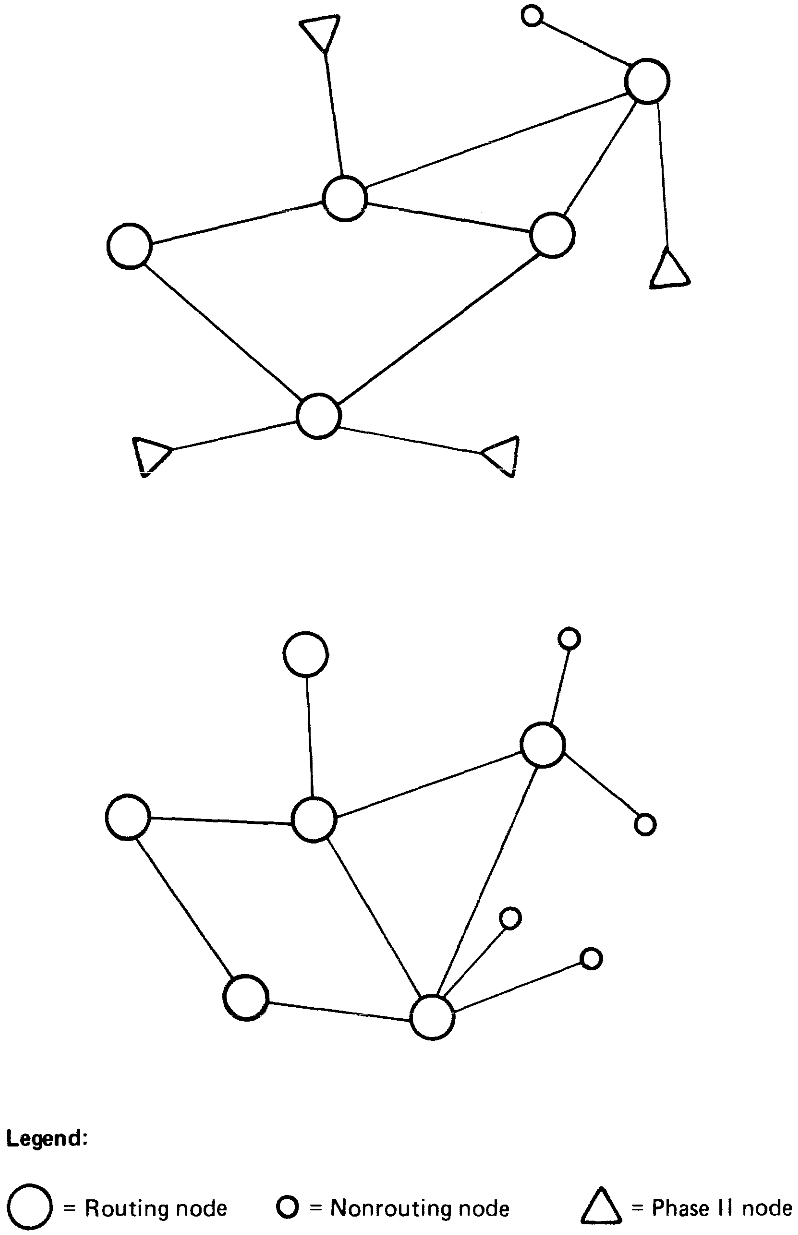


Figure 3–4: Phase III DECnet Configurations

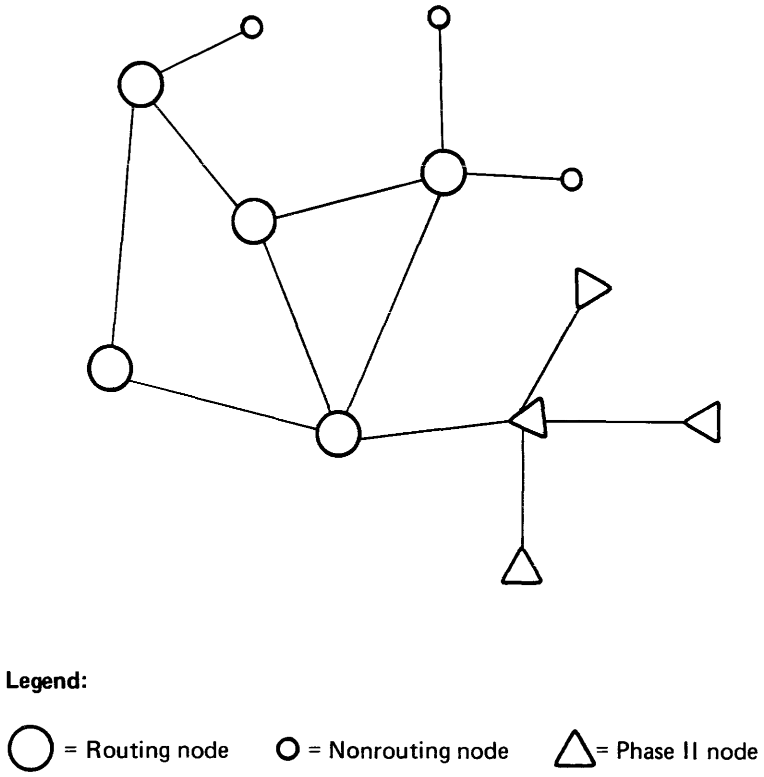


Figure 3–5: A Mixed Configuration: a Phase III Network Adjacent to a Phase II Star-shaped Network

3.2 Basic Concepts of Full Routing

This section introduces some of the concepts that underlie DECnet's implementation of full routing. The Transport layer of Phase III DNA defines both the full-routing functions and the methods to be used to implement those functions. In this discussion, the term transport module refers to the DECnet software that implements the Transport layer model.

3.2.1 Node Addresses and Node Names

Within a Phase III network, every node has a unique numeric address. A packet to be routed contains a destination node address in its header, which has been added by a Transport module (see Figures 2–4 and 2–5). The packet may arrive from the NSP module in the same node or from the Transport module in an adjacent node. As one would expect, the destination address determines where the Transport module sends the packet. Other factors, which are explained below, determine the path the packet travels to its destination.

In Phase II networks, nodes are addressed by unique alphanumeric names rather than by numbers. Because a Phase II node can send packets to adjacent nodes only, names are practical forms of addresses — the number of adjacent nodes is limited and a data base of unique names is easily maintained. Full routing, however, greatly increases the number of reachable nodes. As a result, routing modules can handle unique numeric addresses more efficiently than node names.

On the other hand, node names are an easier form of address for network users to deal with, where users are programmers, operators, system managers, etc. "Node DENVER" is easier to remember than "Node 27," for instance. To accommodate both the users and the transport modules, Phase III implementations of DECnet require the use of node names at the user level. These names, however, are valid only within the context of the local node; they are eventually translated into their corresponding numeric addresses, which are the only identifiers that uniquely describe all the nodes in the network.

3.2.2 Routing Terms

A path is the route a packet travels from its source to its destination. Path length and path cost, defined below, are important factors in the execution of full routing.

- **Path length.** Path length is the distance from the source node to the destination node, measured in hops. A hop is equal to a circuit between two nodes. (Circuits* are logical point-to-point communication paths; see Section 8.4.6.)

  A path never exceeds a maximum number of hops, which is a value set by a system manager or determined by the DECnet implementation.

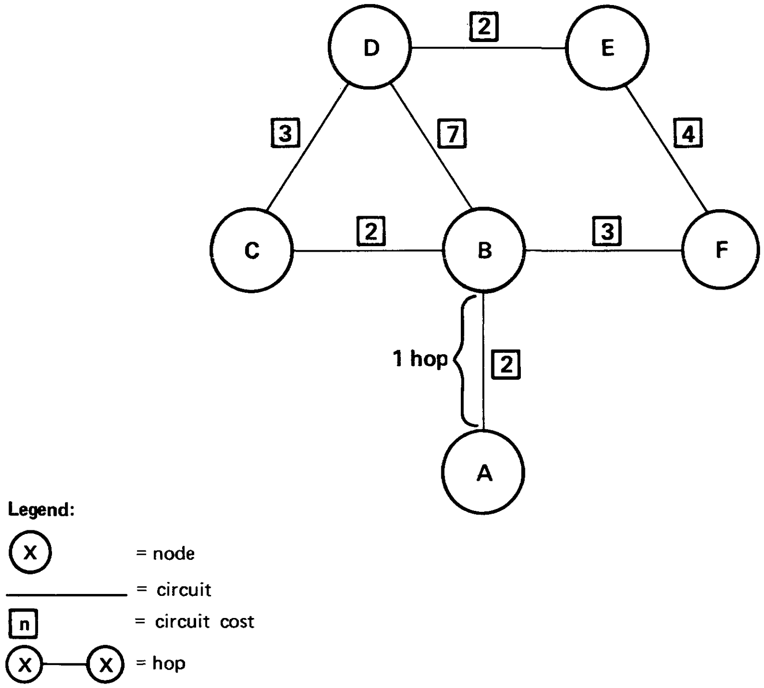
- **Path cost.** Path cost is the sum of positive integer values assigned to the circuits that compose the path: Each value is called a circuit cost.

  When generating a network data base, a system manager or operator assigns a cost to each circuit defined for that node. When the node is up and running, an operator can dynamically change individual costs to higher or lower values. Altering circuit costs can change packet-routing paths.

* DECnet-IAS and DECnet-RT implementations do not support the concept of circuits. In their cases, a hop equals the physical line between two adjacent nodes.

Maximum path lengths and assigning costs to circuits are discussed in Section 3.3.

Figure 3-6 is an annotated diagram of a Phase III network consisting of routing and nonrouting nodes. The annotations explain the meaning of the terms path, path length, hop, path cost, and circuit cost.



Node A wants to send a packet to Node D. There are three possible paths.		
PATH	PATH COST	PATH LENGTH
A to B, B to C, C to D	2 + 2 + 3 = 7*	3 hops
A to B, B to D	2 + 7 = 9	2 hops
A to B, B to F, F to E, E to D	2 + 3 + 4 + 2 = 11	4 hops

*7 is the lowest path cost; Node A therefore routes the packet to Node D via this path.

Figure 3–6: Routing Terms

3.2.3 Routing Algorithms and Data Bases

The Transport module in each routing node implements routing algorithms and other functions defined by the Transport layer. These algorithms determine where the Transport module sends or forwards (routes through) a packet.

One routing algorithm calculates the path length and path cost to every possible destination via every circuit defined for the node. Another algorithm then determines which circuit represents the least costly path to each destination in the network. The algorithms operate on locally available values as well as on values supplied by all adjacent nodes.

The results of these algorithms are saved in routing data bases maintained by each node. Upon receipt of a packet to be routed, the Transport module first consults the data bases to find the least costly path to the packet's destination and then sends the packet via the appropriate circuit. A packet is discarded or returned to its sender if its destination is not accessible via any circuit known to the Transport module. (The sender in this context is the local NSP module.)

When a packet arrives at the next adjacent node on the path, the Transport module there also chooses the least costly path for the packet, according to the local node's routing data bases. The Transport module performs the same services for packets being routed through the local node as for packets that are starting out on their journey.

When a packet finally reaches its destination, the Transport module for that node recognizes that the destination address matches the address of the local node. The packet is then delivered to the module that implements the next higher architectural layer (the NSP module in most cases).

Each routing node executes the algorithms over and over again in response to events of different kinds within the network. For example, if a physical line somewhere in the network goes down, the routing algorithms recalculate the path length and cost to destinations affected by the line failure. An operator can also cause the algorithms to recalculate by changing the cost assigned to a circuit.

Whenever the algorithms reexecute, each node reveals the contents of its data bases to all adjacent nodes, which use that information to update their own routing data bases. In this way, changes affecting path lengths and costs ripple back and forth through the network, so that all data bases are updated to reflect the current state of the network. The rippling action stops when all the routing nodes' data bases are consistent.

3.2.4 Congestion Control

Each Transport module executes congestion control algorithms to limit the number of packets queuing up for transmission on individual circuits. One algorithm regulates the ratio of input packets to route-through packets. Input packets are those that originate from the local node. They are rejected in preference for route-through packets when a circuit's output queue starts to fill up. Another algorithm limits the maximum length of a circuit's output queue so that a single circuit cannot monopolize the available buffers.

3.2.5 Packet Lifetime Control

A packet lifetime algorithm tracks the number of nodes a route-through packet has visited and discards packets that have exceeded a predefined limit. This control ensures that packets can never loop endlessly through the network.

3.3 Affecting Routing Operation

Although Phase III routing has been designed to operate without need of direct user intervention, system managers and operators can exercise some indirect control over routing performance. This interface to the Transport module is provided by a network management module. Such indirect control involves manipulation of maximum path lengths and circuit costs. When building a DECnet system for a particular node, a system manager can define initial values for these parameters. The values are determined after careful consideration of their effects on the local node and on the network as a whole. Subsequently, the values can be modified to improve performance or to reflect changes in the network configuration.

3.3.1 Maximum Path Length

The maximum path length parameter is used to ascertain whether or not a destination is reachable. This parameter is always set to a value equal to or greater than the longest possible path within the network. For example, if a network consists of four nodes in a ring configuration, the longest possible path within that network comprises three hops. Therefore, a destination is unreachable if it cannot be reached in three hops. Among other reasons, a node might be unreachable because it has failed or been removed from the network or because the only physical line connecting it to the network has failed.

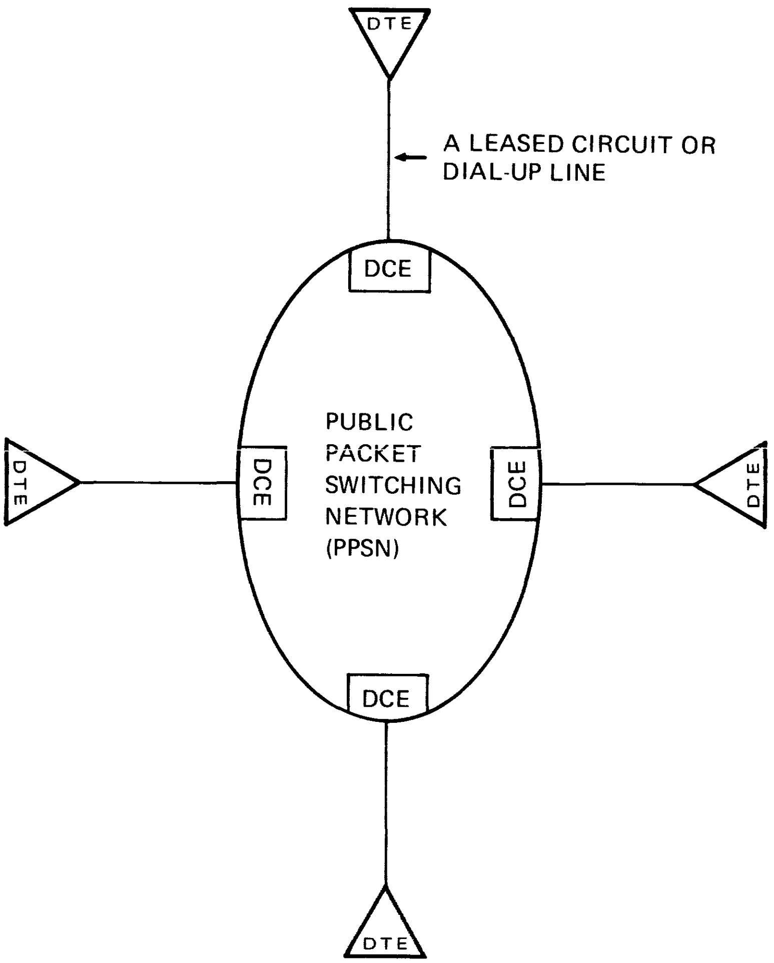
3.3.2 Circuit Costs

Circuit cost is a representation of the real or arbitrarily imposed characteristics of a circuit. By assigning a higher or lower cost (positive integer value), a system manager may influence how much the circuit will be used — higher cost tends to cause lower traffic volume. As Section 3.2.3 explains, path cost is the sum of the individual circuit costs incurred by using the path. Because a major function of routing is to send packets by the least costly paths, individual circuit costs are important factors in routing performance.

Section 8.4.4 lists these and other routing parameters that can be defined.

3.4 DECnet to DECnet Routing via a Packet Switching Network

An RSX DECnet node that includes RSX-11 PSI (an RSX DECnet/PSI node) can route packets to remote DECnet nodes via a Public Packet Switching Network (PPSN). The Data Link Mapping (DLM) interface provides this routing capability. Section 3.4.1 briefly defines PPSNs. Sections 3.4.2 and 3.4.3 explain the role of DLM and how access to a PPSN can expand the topology of a DECnet network.



Legend:

DCE = Data Circuit-terminating Equipment, a PPSN switching node.

DTE = Data Terminal Equipment, a user computer or terminal using the PPSN.

Figure 3–7: Components of a Public Packet Switching Network

3.4.1 Public Packet Switching Networks (PPSNs)

A PPSN is a data communications service offered by the Postal Telephone and Telegraph Authorities (PTT) or common carriers of many countries. A PPSN receives packets of data to be transmitted from users of the network. Each packet includes a header of control and destination information that the PPSN uses to route the packet to its proper destination and to deliver it in its proper sequence. The PPSN shares its transmission lines by interleaving packets from many senders. Neither the packet's sender nor its receiver can directly influence the route a packet takes to its destination.

Each PPSN consists of a number of geographically separated switching nodes that are connected by high-speed links. A circuit leased from the PTT or common carrier connects a user's computer to one of the PPSN switching nodes, while either a leased circuit or a dial-up line provides the connection for a terminal. The PPSN switching nodes are called network interfaces or Data Circuit-terminating Equipment (DCE). User computers or terminals connected to DCEs are called Data Terminal Equipment (DTE). Figure 3–7 illustrates the components of a PPSN.

3.4.1.1 CCITT Recommendations X.25, X.3, X.28, and X.29 — All DTEs use standard interfaces to the PPSN. DTEs that operate in packet mode, which include computers and intelligent terminals, use the CCITT recommendation X.25 interface. Recommendation X.25 defines the interface between the packet-mode DTE and the DCE to which it is connected. How the DCE itself functions within the PPSN is transparent to the DTEs and not relevant to X.25. To use a PPSN, non-packet-mode (character-mode) terminals require special support from the PPSN and the packet mode DTE with which it wants to communicate. CCITT recommendations X.3, X.28, and X.29 define the interfaces and mechanisms that enable such terminals to function as DTEs.

Any type or make of computer or terminal can become a DTE as long as it implements the appropriate CCITT recommendation(s). Transparently to the DTEs, the PPSN handles any differences in buffering and operating speeds of the various DTEs in the network. This transparent PPSN function and the use of the standard interfaces enable one DTE to communicate with any other DTE on the PPSN, regardless of individual type or make.

3.4.1.2 Virtual Circuits — Two DTEs communicate by means of a virtual circuit, which is a logical association set up by the PPSN either permanently or temporarily. Each virtual circuit handles the exchange of data between two specific DTEs. The DTE at each end assigns a logical channel and a corresponding channel number (LCN) to the circuit. When sending data, the DTE includes an LCN to identify the channel and corresponding circuit to which the data belongs.

When a DTE uses more than one virtual circuit at a time, the circuits are multiplexed over the physical link between the DTE and the DCE.

Virtual circuits are either permanent or temporary (switched):

- Permanent Virtual Circuits — A permanent virtual circuit (PVC) is analogous to a leased line between a local and a remote DTE. Either DTE can send data over the PVC at any time without issuing calls to set up or break the circuit. When a user subscribes to a PPSN, the administrators of the PPSN allocate the LCN for each PVC to be used.
- Switched Virtual Circuits — A temporary association between two DTEs is called a switched virtual circuit (SVC). A DTE sets up this type of circuit only when it wants to send data. The sending DTE assigns a channel, identifies the target DTE, and then obtains that DTE's agreement to communicate. When the DTEs have finished exchanging data, one or the other initiates a clearing sequence to terminate the SVC.

3.4.2 The DLM Interface

Using the Data Link Mapping (DLM) interface, DECnet users of an RSX DECnet/PSI node have transparent access to a PPSN. For these DECnet users, the PPSN is a means of routing packets to remote DECnet nodes. Specifically, DLM enables RSX-11 PSI to set up and manage virtual circuits on behalf of DECnet users. (DECnet users' limited access to the PPSN does not allow them to communicate with non-DECnet DTEs; only the RSX-11 PSI user interface provides that capability.) A major benefit of DLM is that it extends possible DECnet topologies by allowing a PPSN to form the physical link between RSX DECnet/PSI nodes. This section briefly explains what DLM does and how it works. Section 3.4.3 discusses in greater detail how DLM can affect the topology of a DECnet network.

In a DECnet/PSI node, the routing data base (Section 3.2.3) specifies circuits leading to the node's Data Link layer and DLM circuits leading to PSI software. Each DLM circuit is associated with a specific DTE (an RSX DECnet/PSI node) on the PPSN. When DECnet data is addressed to a remote node reached via a PPSN, the circuit the Transport module chooses for that data leads to the PSI software. PSI then handles the DECnet packets just the same as data it receives from PSI users: It envelops each DECnet packet in X.25 protocol and sends it over the PPSN to the destination DTE associated with the chosen DLM circuit. The target DECnet node may be the node adjoining the DTE or it may be a DECnet only node reached via the destination DTE node.

Figures 3–8, 3–9, and 3–10 illustrate this process. As shown in Figure 3–8, the DLM interface enables the Transport module to transfer a DECnet packet to the RSX-11 PSI software. In effect, an RSX system that runs both RSX DECnet and RSX-11 PSI has two network identities: (1) it is a node within a DECnet network and (2) it is a DTE within a PPSN. The X.25 recommendation defines three functional levels (levels 3, 2, and 1), which, in the context of DNA, operate mostly alongside the Data Link and Physical Link layers. (Figure 3–8 illustrates this relationship.) DLM creates a communication path between the node's Transport layer and the DTE's X.25 level 3.

Figure 3–9 is a diagram of a DECnet packet that has been prepared for transmission over the PPSN. X.25 level 3 software is responsible for setting up and maintaining virtual circuits and for adding a header of destination and control protocol to PSI user data. (In the context of an RSX DECnet/PSI node, DECnet packets received from DLM are the same as PSI user data.) By adding a header to PSI user data, level 3 software creates an X.25 packet. Level 3 transfers the X.25 packet to level 2, which forms a frame by adding its own protocol header and affixing a flag at both ends. The frame then passes to X.25 level 1, which controls its physical transmission from the DTE to the DCE. Figure 2–4 shows how DECnet modules implementing DNA layers build up user data in the same way, a process that Figure 3–8 reiterates on the DECnet side of the diagram.

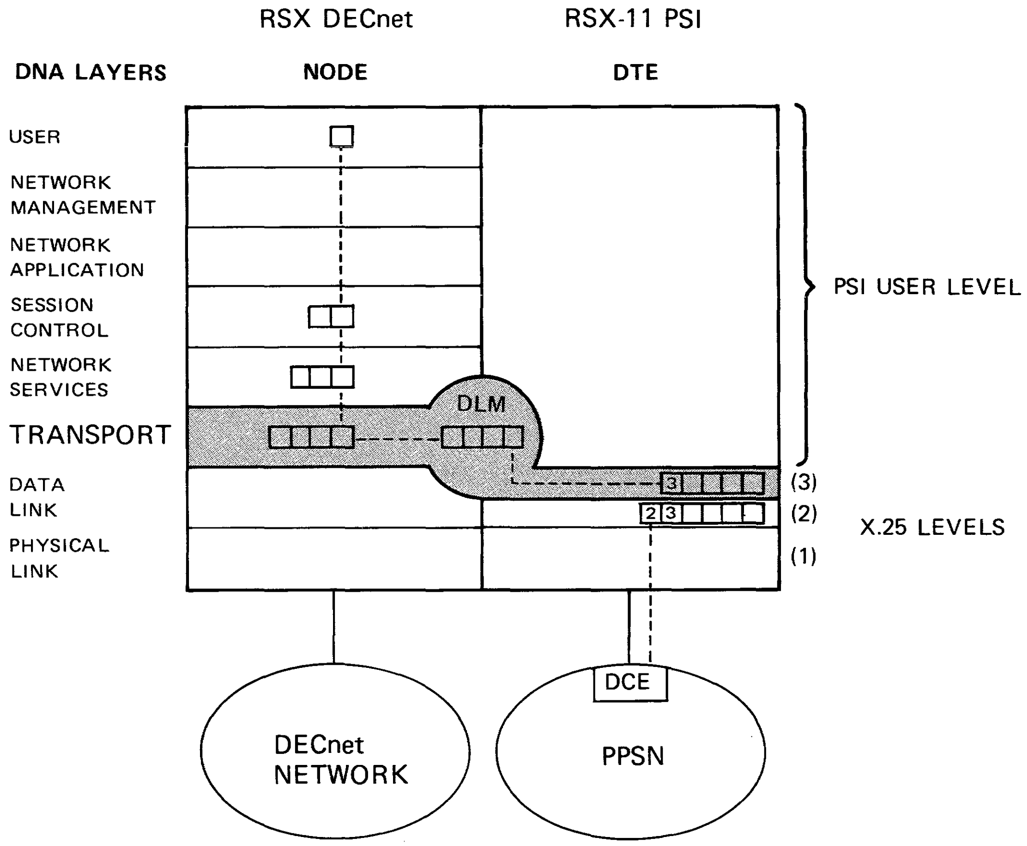


Figure 3–8: DLM Relaying DECnet Data to PSI Software

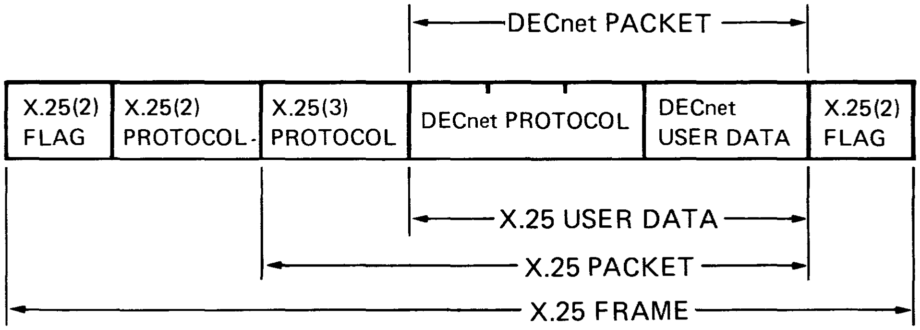
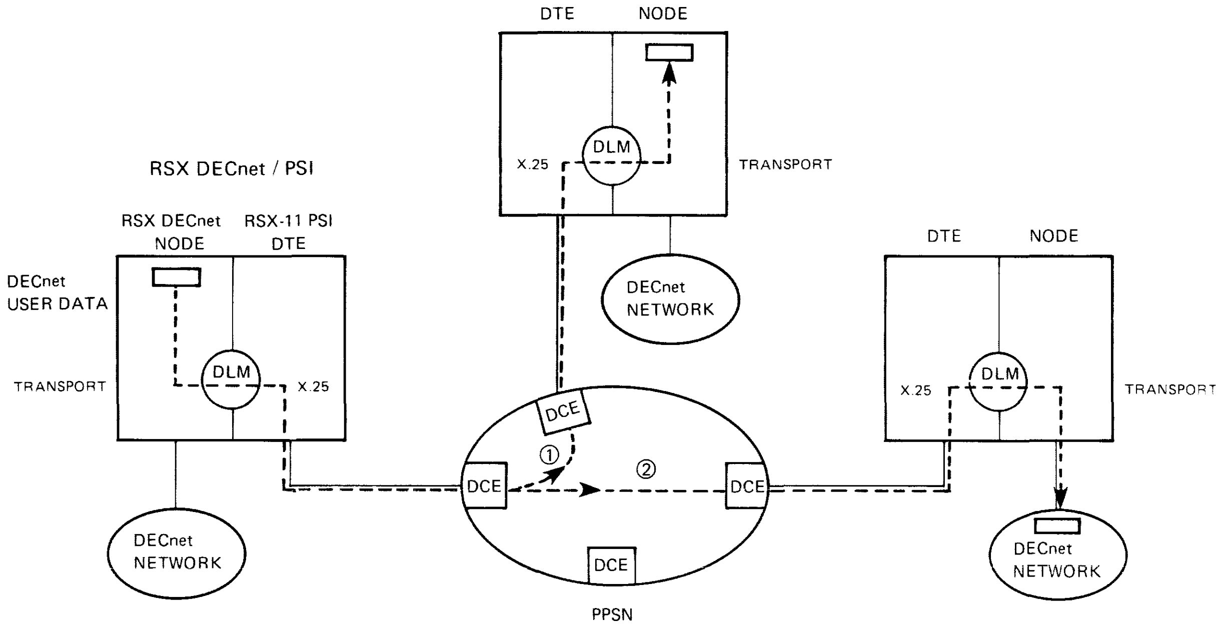


Figure 3–9: A DECnet Packet Nested in X.25 Protocol

Figure 3–10 shows two logical paths for DECnet data that is being routed via a PPSN. The first path terminates at the DECnet node that is also the destination DTE. The second path passes through the destination DTE and beyond to a target DECnet node. When the X.25 packet containing the DECnet data reaches the destination DTE, software at levels 2 and 3 strip off the X.25 protocol headers. Level 3 identifies the local node's Transport module as the target PSI user and accordingly hands over the DECnet packet. DECnet data following the first path in Figure 3–10 then passes to the target user in the local node. If DECnet data is following the second path, the Transport module in the RSX DECnet/PSI node forwards it to another DECnet node elsewhere in the network.



Legend:

Path ① = The target DECnet node is also a target DTE.

Path ② = The target DECnet node is reachable via a DTE.

Figure 3–10: Routing DECnet Data via a PPSN

3.4.3 The Effect of DLM on DECnet Topology

Phase III nodes able to communicate with an RSX DECnet/PSI node also have access to a PPSN through that node's DLM interface. Figure 3–10 shows how DECnet data can be routed beyond the destination DTE to other DECnet nodes. Similarly, the source node for DECnet data routed over the PPSN does not have to be a DTE. As long as the source node supports Phase III routing, it can also send data over a PPSN. Any intervening nodes forward the data to the DECnet/PSI node adjacent to the PPSN. From there, the data proceeds along one of the logical paths shown in Figure 3-10. Figure 3-11 shows the kind of Phase III network that DLM makes possible. All the DECnet nodes shown can communicate with one another and are therefore logically part of the same network.

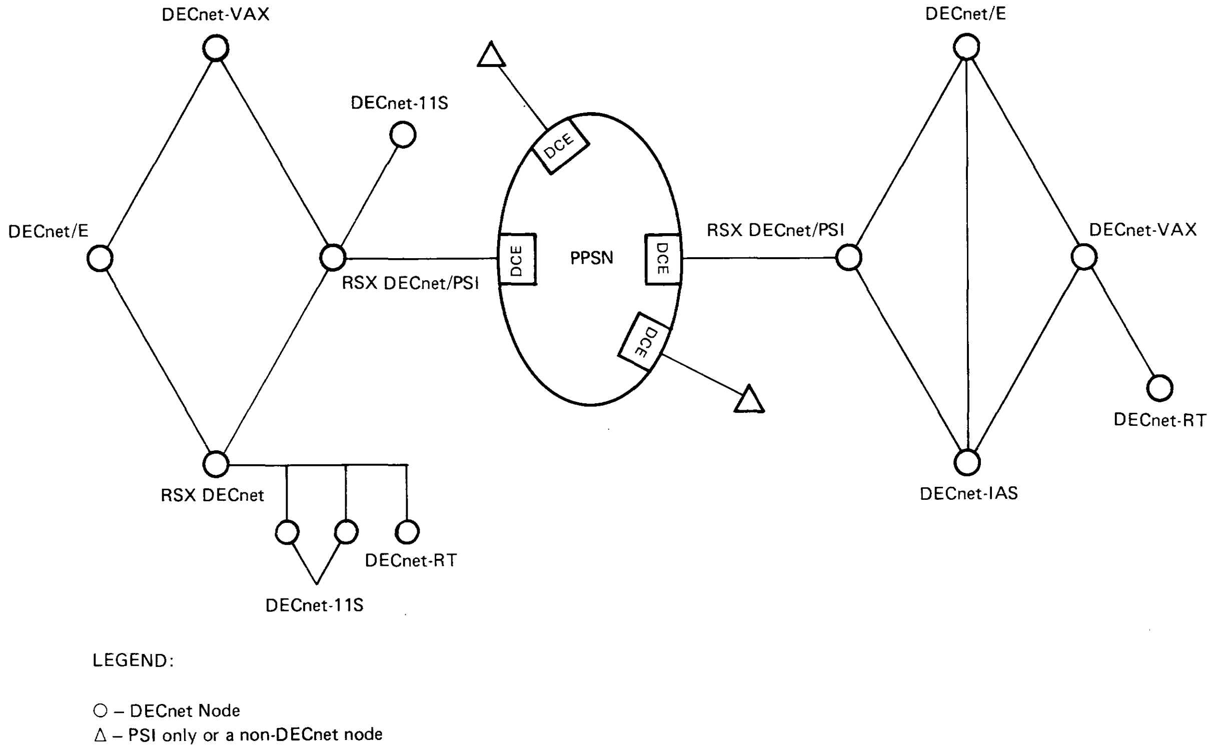


Figure 3–11: RSX DECnet/PSI Nodes within a Phase III Network

Chapter 4
Logical Links

DECnet uses a mechanism called a logical link to allow communication between programs running within the same node or in separate network nodes. Each logical link is a temporary data path connecting two specific programs. The two programs can exchange data over the link until one or the other program decides to terminate the connection (see Figure 4–1).

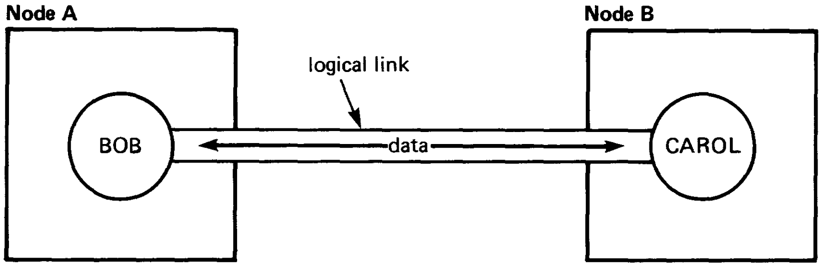


Figure 4–1: Logical Link Connecting Programs BOB and CAROL

The Network Services Protocol (NSP) defines the rules that govern the creation and operation of logical links. Every DECnet node includes a software module that implements NSP specifically for the node's operating system. For example, an RSX DECnet node and a DECnet–VAX node require different implementations of NSP. In Phase III nodes, these modules also perform the functions of the Session Control layer of DNA (see Section 2.5). NSP modules provide services that are analogous to those provided by the telephone company. Upon request by a caller, NSP modules set up a connection with a specific receiver somewhere in the network. Neither party cares how the module actually sets up the connection, and either party can hang up. Figure 4–2 illustrates the relationship between NSP modules and programs using logical links.

In addition to creating and operating individual logical links, NSP modules enable multiple logical links to share a single communications line. Within a DECnet network, each communications line carries data packets belonging to one logical link intermingled with data packets belonging to other logical links. NSP modules format outgoing logical link data for transmission by the communications hardware. Conversely, these modules separate incoming data into logical link streams and deliver the data to the appropriate local programs.

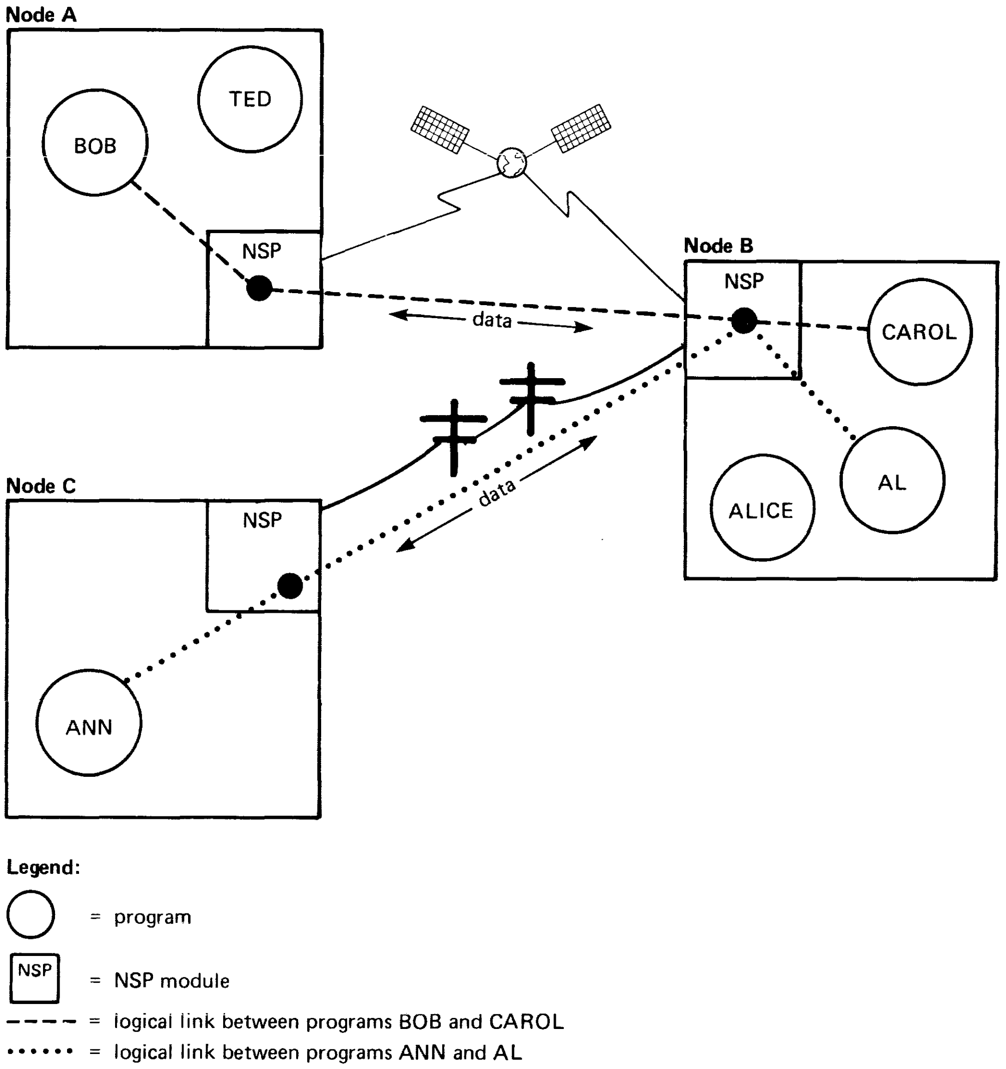

Node A
TED
BOB
NSP
data
Node B
NSP
CAROL
AL
ALICE
Node C
NSP
data
ANN
Legend:
○ = program
NSP = NSP module
– – – – = logical link between programs BOB and CAROL
• • • • • • = logical link between programs ANN and AL

Figure 4–2: NSP Modules and Logical Links

4.1 The Handshake Dialog and Logical Links

NSP modules in separate nodes create each logical link on behalf of two cooperating programs. Even though a logical link can connect programs running in the same node, this section describes logical links between programs residing in separate nodes. Two programs wishing to communicate over a logical link must follow the same procedures regardless of either program's location in the network.

Cooperation is essential for a successful connection. A logical link will not be created unless both programs agree to communicate. To use programs BOB and CAROL as an example, BOB cannot be linked to CAROL until CAROL agrees to the connection. The two programs must have a preliminary dialog, with the NSP modules acting as intermediaries, before exchanging data. The preliminary dialog is sometimes called a handshake — each program recognizes and agrees to be linked to the other program.

The following dialog illustrates how the handshake proceeds. The program that requests or initiates the link is the source program, and the program that must accept or reject the request is called the target program. In the dialog illustrated below, BOB is the source program and CAROL is the target program.

- BOB in Node A issues a request to NSP(A): I want to talk to CAROL in Node B.
- NSP(A) contacts NSP(B): BOB in Node A wants to talk to CAROL.
- NSP(B) contacts CAROL: Do you want to talk to BOB in Node A?
- CAROL responds to NSP(B): Yes, I will talk to BOB.
- NSP(B) responds to NSP(A): Yes, CAROL will talk to BOB.
- NSP(A) responds to BOB: Yes, CAROL will talk to you.

As soon as CAROL agrees to talk to BOB, NSP(A) and NSP(B) together establish the logical link that allows the two programs to communicate. Once the logical link exists, both programs can send and receive data on an equal basis, and either program can decide to terminate the link.

4.2 Logical Link Identifiers and Addresses

A successful handshake accomplishes several goals:

- It confirms that both programs agree to talk with one another.
- It causes a logical link to be created.
- It provides important addressing information at the beginning of the programs' conversation. In subsequent data transfers, the programs only need to specify a link identifier.

During the handshake sequence, each program specifies a link identifier to the local NSP module. If the connection is successful, the program uses the link identifier to address all messages to be sent over the link. In turn, the NSP modules cooperate to assign their own link addresses that define the link uniquely to each of them. At either end of the link, each NSP module associates the NSP level addresses with the local program's link identifier. Figure 4–3 illustrates the interrelationship of the programs' identifiers and the NSP modules' addresses.

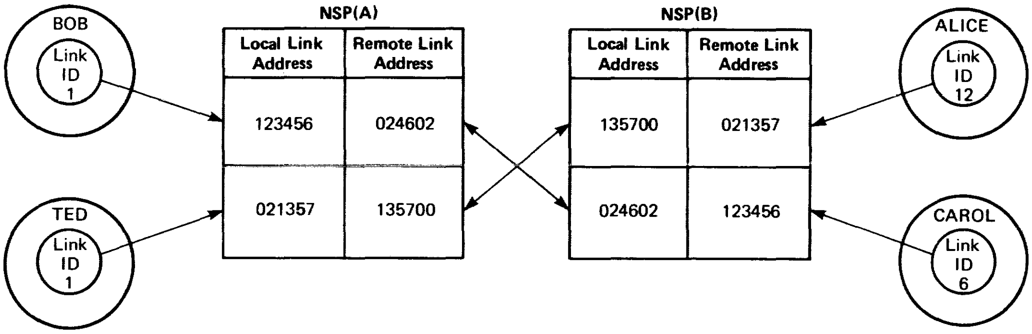


Figure 4–3: Interrelationship of Link Identifiers and Addresses

4.3 Logical Links and Individual Programs

A program can use more than one logical link at a time, up to a maximum number determined by the programmer and/or by a system restriction. Each link identifier assigned by the program must be unique to differentiate among simultaneous logical links. Note that the identifier assigned to a link by one program has no relevance to the identifier assigned to the same link by the remote program. As Figure 4-3 shows, the responsible NSP modules ensure that both identifiers actually refer to the same logical link.

A program can establish logical links that communicate with different programs; or, a program can establish several logical links with the same program to exchange data intended for different purposes. For example, two programs can establish two logical links between them; one link can be used to transmit transaction data, while the other can be used to transmit control information. In Figure 4-4, program BOB in Node A operates logical links between itself and programs CAROL and ALICE, while program ALICE operates a second logical link to program TED in Node A.

Section 5.1 describes the use of calls within a program to send and receive data over a logical link.

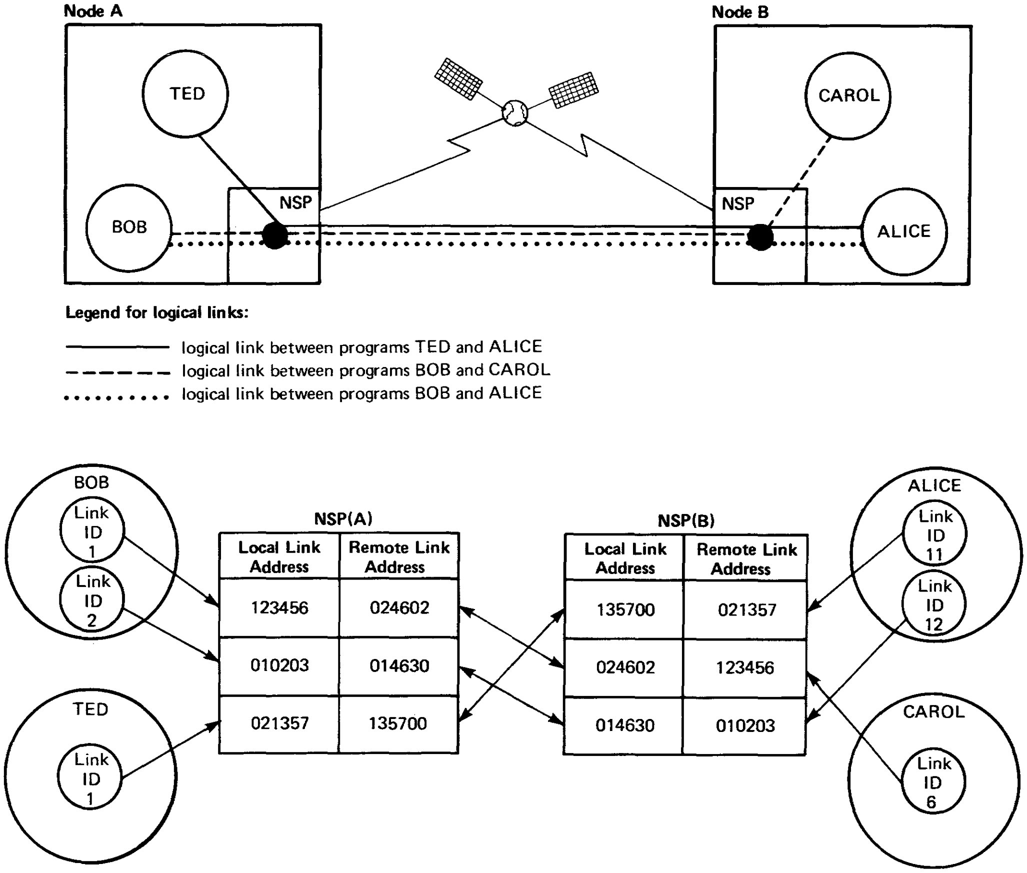
4.4 NSP Control Messages

In the handshake dialog discussed in Section 4.1, the NSP modules actually carry on most of the conversation. NSP(A) and NSP(B) exchange control messages to set up the link between BOB and CAROL. The NSP protocol defines the control messages used by all NSP modules for creating and controlling logical links.

4.5 Sending and Receiving Data

After creating a logical link, the NSP modules begin to orchestrate data transfers between the connected programs. When a program hands over a unit of data for transmission, the local NSP module may send the data in one data packet, or it may divide the data into segments and send each segment in a separate packet. In this case, the remote NSP module reassembles the segments before delivering the data to the remote program.

Normal data constitutes the subject matter of a dialog. To interrupt the dialog, either of the linked programs can usually send interrupt data, which breaks through the current dialog. The means of delivering interrupt data are system and program dependent, but the receiver usually accesses it before accessing any normal data that may be pending.


Node A
Node B
TED
BOB
NSP
CAROL
ALICE
NSP
Legend for logical links:
———— logical link between programs TED and ALICE
– – – – – logical link between programs BOB and CAROL
• • • • • • logical link between programs BOB and ALICE
BOB
Link ID 1
Link ID 2
TED
Link ID 1
ALICE
Link ID 11
Link ID 12
CAROL
Link ID 6

NSP(A)

Local Link Address	Remote Link Address
123456	024602
010203	014630
021357	135700

NSP(B)

Local Link Address	Remote Link Address
135700	021357
024602	123456
014630	010203

Figure 4–4: Programs Supporting Multiple Logical Links

4.6 Segment Acknowledgment and Retransmission

The NSP modules that manage both ends of a logical link guarantee:

- That all transmitted data is received
- That all received data is given to the target program in the proper sequence

To guarantee proper segment sequencing, an NSP module numbers the segments transmitted over the link. The receiving NSP module, using the transmit numbers for identification, must acknowledge the delivery of the segments. If a segment is not acknowledged within a certain period of time, the sending NSP module retransmits it.

NSP modules assign a different set of transmit numbers to interrupt messages. The separate sets of numbers logically divide normal data and interrupt data into separate data streams within the logical link. See the detailed specification of NSP for further information.

The detailed execution of segment acknowledgment and retransmission varies depending on the NSP implementations involved. However, despite variations in detail, all NSP modules use these mechanisms to guarantee delivery of all transmitted segments and to ensure that the segments are delivered in the proper sequence.

4.7 Flow Control

Network programs and NSP modules both require a certain amount of buffer space for temporary message storage. For example, an NSP module keeps a copy of every message it sends over a link until the receiver acknowledges receipt of the message. At the program level, buffer space is necessary to hold inbound messages waiting to be processed. NSP modules and programs need buffer space for other purposes as well, depending on the application and the DECnet implementations.

Without some kind of control, message traffic could easily cause available buffer space to overflow. To prevent this, the programs and NSP modules exercise flow control*. In most implementations, programs coordinate, send and receive calls: An NSP module transmits data from a source program only if the target program has issued a receive call. In some implementations, however, the target node's NSP module must merely have sufficient buffer space available to hold the data. In either case, the NSP modules handling the link between the programs ensure that the appropriate condition is satisfied before any data is actually sent.

The NSP modules exchange link service messages to request and convey information about the availability of buffer space and about other conditions that pertain to data flow on the link. The detailed operation of flow control depends on the DECnet nodes on the link.

4.8 Logical Link Applications

Almost all network functions require the services of a logical link between programs. Exceptions include some maintenance functions, such as certain loopback tests and the down-line loading of satellite systems. NSP modules create a logical link to accomplish any of the following types of connection:

- **A user program connected to another user program.** BOB and CAROL provide an example of this type of connection.
- **A user program connected to a DECnet module.** For example, BOB connected to a remote FAL module in order to access a remote file (see Section 6.1).

* Both RSX DECnet and DECnet-IAS allow flow control to be turned on or off. Under certain circumstances, a network manager may choose to turn off flow control to improve network performance.

- **A DECnet module connected to another DECnet module.** For example, a terminal user invokes a DECnet system program in the local node, which in turn makes a connection with a cooperating DECnet module in a remote node.

In the first type of connection, a DECnet application programmer must directly control the link by including DECnet calls in the source and target programs. In the second type of connection, the programmer must include calls that initiate and control the link in the user program only; the DECnet module automatically handles its end of the link. In the last case, the creation and operation of the logical link is a level removed from the user; user programming is not involved in the connection at all. Typically, the DECnet modules exchange messages based on locally supplied input.

Chapter 5
Task-to-Task Communication

All DECnet implementations allow two programs within a network to perform task-to-task communication, that is, to exchange data over a logical link. For example, an RSX–11M program can use local DECnet–RSX facilities to communicate with a program running in a RSTS/E node in the same network.

The language used to write this kind of network program depends on the node in which the program will run. DIGITAL operating systems do not all support the same languages, and not all languages supported by each system can be used for task-to-task communication. Furthermore, the programming language used does not depend on the remote program's language or operating system. For example, an RSX–11M program written in FORTRAN IV–PLUS can communicate with a RSTS/E program written in BASIC–PLUS–2. The NSP modules at either end of the programs' logical link provide the necessary interfaces.

Table 5–1 lists the programming languages that support DECnet task-to-task communication according to the applicable DECnet implementation.

Table 5–1: Languages Supporting Task-to-Task Communication

	DECnet–RT	DECnet/E	RSX DECnet	DECnet–IAS	DECnet–VAX	DECnet–20
BASIC–PLUS		X				
BASIC–PLUS–2		X	X	X		
VAX BASIC					X	
BLISS					X	
COBOL		X	X	X	X	
CORAL				X	X	
FORTRAN	X	X	X	X	X	
MACRO	X	X	X	X	X	X
PASCAL					X	
PL/I					X	

In most DECnet implementations, performing task-to-task communication is similar to performing I/O. The logical link between two programs is like an I/O channel over which both programs can send and receive data. The RSTS/E operating system has a native ability to form a communication path between two local programs. DECnet/E therefore implemented communication with remote programs as an extension of RSTS/E's local send/receive services.

5.1 DECnet Task-to-Task Calls

A network program uses DECnet calls to communicate with a remote program. As Section 4.1 explains, NSP modules actually set up and control logical links. The DECnet task-to-task calls activate routines that request the local NSP module to perform specific functions. A network program specifies parameters in the calls to pass information to the local NSP module.

The form of the DECnet calls that a programmer can use depends on the source language; they may actually be calls, macros, or system directives. The DECnet task-to-task capability translates a variety of system-dependent language calls into the same set of NSP level messages. For example, in response to a connect request call, an NSP module sends the same type of NSP level message to a remote node regardless of the source program's native language.

Every DECnet implementation provides a network program with the means to perform the following functions:

- Request a logical link
- Receive a logical link request
- Accept or reject a logical link request
- Send data
- Receive data
- Send interrupt data
- Receive interrupt data
- Terminate the logical link

There is not always a one-to-one correspondence between a system-specific call and one of the above steps. In some cases, a program must issue three separate calls to initiate a logical link; in other cases, a program needs to issue only one call.

5.2 Addressing a Connect Request

In the handshake dialog that starts up a logical link, the source program issues a connect request call that includes network addressing information. The format of a connect request call depends on the language used to write the source program and on the DECnet implementation in which it will run. In some cases, a source program must issue more than one call to generate a connect request.

A connect request call passes all or part of the following information to the local NSP module:

- **Link identifier.** This differentiates the requested link from any other links currently being used by the source program.[1] (See Section 4.2 for a discussion of logical link identifiers.) If the connect request succeeds, the source program uses the link identifier to address data to be sent over the link. The source program's link identifier is called a logical unit number (lun) by RSX DECnet, DECnet IAS, and DECnet-RT users, a channel number by DECnet-VAX users, a user link address (ula) by DECnet/E users, and a job file number (jfn) by DECnet-20 users.
- **Target node identifier.** This can be a unique identifier that distinguishes the node from all others in the network, or it can be a locally defined name that the local DECnet software translates into a unique node identifier (see Section 3.2.1).
- **Object type or name.** Object is another term for a network program. A network program or object has a special identifier for use in network calls. This identifier consists of an object type and/or an object name. Section 5.2.1 explains the significance of the object type and name.
- **Access control information.** This information describes the source program and includes a user identification, a password, and optionally, an account number. The information is equivalent to the data a user supplies when logging into a system.

  In most implementations, the target program uses this information as a factor in its decision to accept or reject the connect request. RSX DECnet, DECnet-IAS, and DECnet-VAX nodes also use it to verify the source program's connect request at the target node (see Sections 5.3.1 and 5.3.2).
- **Optional user data.** A source program usually has the option of sending 16 bytes of data to the target program as part of a logical link connect or disconnect request.

5.2.1 Object Types and Names

A network program makes itself known to the local NSP module by declaring its object type and name. In most DECnet implementations, a program must declare its object type and name in order to be eligible to receive link requests. (In some implementations, a system manager can use a DECnet command at a terminal to declare a program's object type and name.[2]) The name may be a special network name for the program, or it may be the same name by which the program is known to the local operating system.

[1] A source program using DECnet/E's concise COBOL interface (see Section 5.6.2) does not supply a link identifier in a connect request. Such a program can only use one logical link at a time.

[2] In DECnet-VAX, any command procedure on disk can be the object of a link request. The procedure need not have been previously declared as an object; DECnet-VAX looks it up when the request is issued.

A source program uses one of two formats to specify the target program in a connect request:

- An object type equal to 0 and an ASCII name
- An object type equal to a positive integer (from 1 to 255) and a null name

The first format identifies a program by name, whereas the second format identifies a program by numeric type.

To address a target program, a connect request specifies either a name or an object type, but not both. The first format — object type 0 plus name — is commonly used to address user-written network programs. To use this format in a connect request, a source program must know the target program's declared object name. Note that the maximum length allowed for a name depends on the local node's operating system.

The second format provides an abbreviated means of identifying a frequently used network function, usually a DECnet module such as the File Access Listener (FAL), which is discussed in Chapter 7. A specific type always represents the same generic function within a network, even if the program that actually performs the function has a different name at each node. For example, the program that performs the FAL function can always be identified by object type 17 (decimal) or 21 (octal). In this way, a network program can address the FAL function without knowing the FAL program's name in the target node.

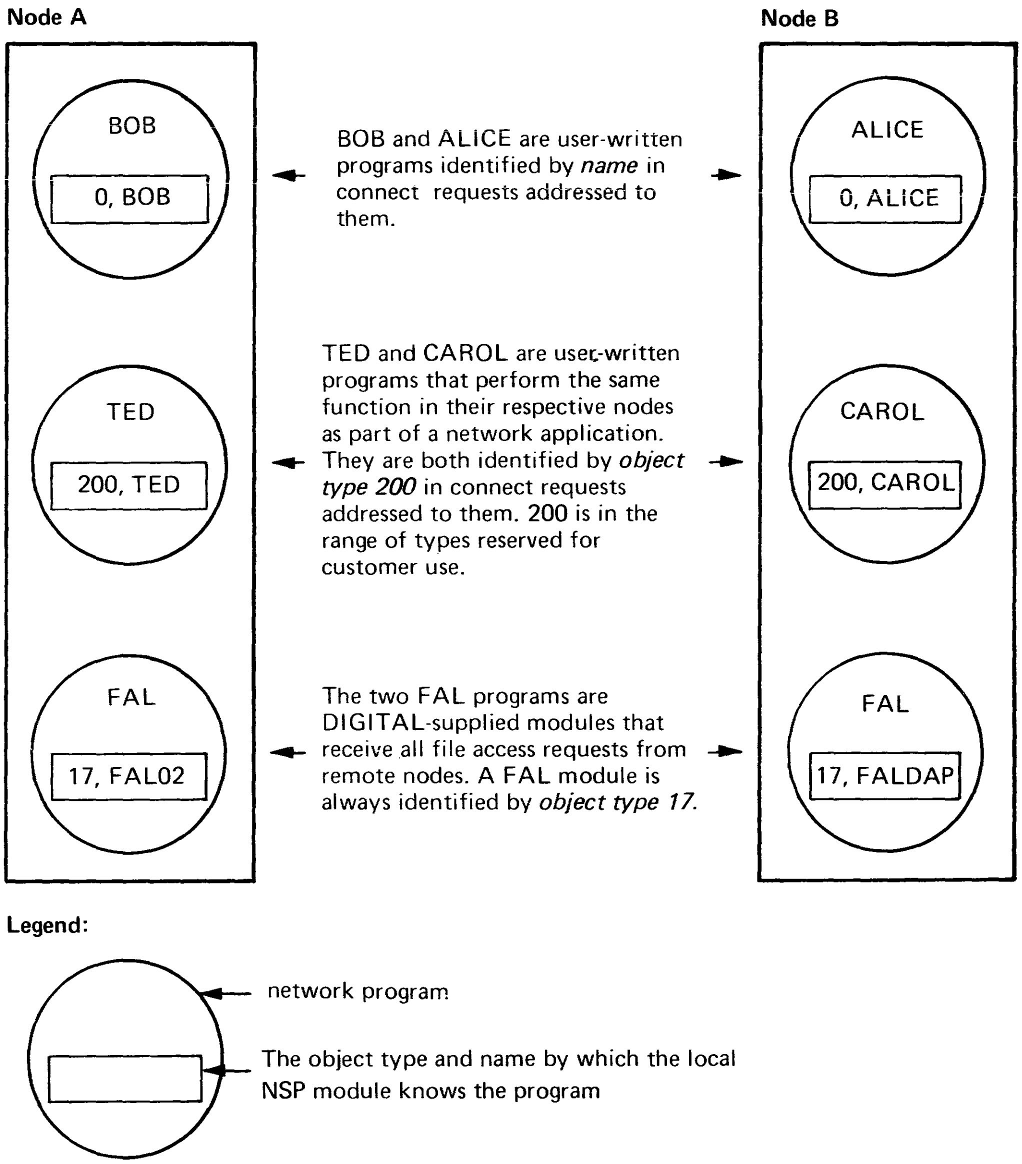
DIGITAL reserves a range of object types for DECnet system programs. Types within this range are used consistently across all DECnet implementations to refer to the same functions. The reference manuals for each DECnet implementation list the object types reserved for DECnet use.

Unreserved types can be used for user-written network programs. For example, in a user-written transaction-processing application, each node might have a resident program for recording statistics on transactions performed within the last 24 hours. The application's designer could choose an unreserved object type to identify all such programs throughout the network.

Figure 5–1 illustrates the object identifiers for several programs in Node A and Node B.

5.2.2 Connect Blocks and Network Specifications

Depending on the local implementation of DECnet, a source program creates either a connect block or a network specification to supply the addressing information required in a connect request. A connect block is a data area



BOB and ALICE are user-written programs identified by *name* in connect requests addressed to them.

TED and CAROL are user-written programs that perform the same function in their respective nodes as part of a network application. They are both identified by *object type 200* in connect requests addressed to them. 200 is in the range of types reserved for customer use.

The two FAL programs are DIGITAL-supplied modules that receive all file access requests from remote nodes. A FAL module is always identified by *object type 17.*

Figure 5–1: Addressing Network Objects

within the source program and is called one of several names: A connect block (DECnet-RT), a connect data block (DECnet/E), a connect or target block (RSX/IAS DECnet), or a network connect block (DECnet-VAX).

For the programmer's convenience, most sets of DECnet task-to-task calls include one or more calls to build the connect block. The target node, target object, access control information, and, optionally, up to 16 bytes of data are specified as parameters to the connect block calls. Then in the DECnet call that passes the connect request to the local NSP module, the source program specifies the address or label of the previously created connect block. A programmer can also build a connect block without using these calls.

A network specification is an ASCII string included in the connect request itself. Like the connect block, it includes a target node identifier, access control information, and a target object type or name. Network specifications are used by DECnet-20 programs in all connect requests and by DECnet-VAX programs that are performing transparent communication. (The distinction between transparent and nontransparent task-to-task communication is explained in Section 5.6.5.) DECnet/E supports a form of task-to-task communication called the concise COBOL interface (see Section 5.6.3), which is similar to DECnet-VAX transparent communication. A connect request using this interface includes a network specification rather than a reference to a connect block. Only DECnet-20 network specifications can include optional data.

5.3 Accepting/Rejecting a Connect Request

The source NSP module forwards the connect request to the node specified in the connect block or network specification. The NSP module in the target node checks that the program addressed in the connect request is a valid object and then, if necessary, verifies the source program's identification (see Section 5.3.1). If the target program is a known object and any required verification checks out, the NSP module delivers the connect request to the target program.

Note that the way an NSP module delivers a connect request message and the way a target program receives the message depend on the applicable DECnet implementation and programming language.

After examining the incoming connect request, the target program either accepts or rejects* it. The target node's NSP module forwards the appropriate response back to the source node. The target program usually has the option of sending 16 bytes of data along with the acceptance or rejection of the link. For example, a connect reject response might include data that tells the source program why the connect request was not accepted.

If the target program agrees to the connection, it specifies its own logical link identifier in a connect accept call.

5.3.1 RSX DECnet and DECnet-IAS Access Control

RSX DECnet and DECnet-IAS nodes can include a verification module that screens all incoming connect requests. For each object, the system manager can assign a verification level, which determines the type of access control exercised for incoming requests to connect with that object. The verification module recognizes three different levels:

- **Level 0.** The source program's user identification and password are verified against the local node's system account file. If the identification and password do not exactly match an entry in the account file, the connect request is immediately rejected; the target program never even receives the connect request.

* Transparent communication in DECnet-VAX (see Section 5.6.5) and the concise COBOL interface in DECnet/E (see Section 5.6.2) do not allow a program to reject a connect request.

- **Level 1.** The source program's user identification and password are verified as for level 0, except that the connect request is forwarded to the target program regardless of the outcome. The verification module tells the target program whether or not the source program checked out against the system account file and whether or not the source program has a privileged identification.
- **Level 2.** Connect requests are forwarded directly to the target program without any kind of verification.

5.3.2 DECnet–VAX Access Control

DECnet–VAX controls access to a node with a procedure similar to the RSX DECnet level 0 verification. At a DECnet–VAX node, a verification module intercepts all incoming connect requests. The module checks each request's access control information against a file that identifies all users authorized to use the local node. If the access control specifies an authorized user, the module forwards the connect request to the target program. If the access control does not specify an authorized user, the module rejects the connect request. In either case, the target program itself never sees the access control information sent by the source program.

5.4 Exchanging Data

After a logical link has been established, the program sending data is called the source and the program receiving it is called the target. The connected programs can swap roles from one data transmission to another, or they can both send and receive data simultaneously. Because the connected programs exchange data on an equal basis, there is no longer a distinction between the program that initiated the connection (previously the source) and the program that accepted the connection (previously the target).

Two kinds of data can be sent over a logical link: normal data and interrupt data. As Section 4.5 explains, normal data makes up the subject matter of the programs' dialog, whereas interrupt data conveys special high priority information.

5.4.1 Normal Data

To convey normal data over the link, a source program issues one or more calls to send the data, and a target issues one or more calls to receive it. RSX DECnet, DECnet–IAS, and DECnet–VAX all require the source and target programs to coordinate calls for sending and receiving data. The NSP module at the source node will not transmit data unless the target program has already issued a receive call. Each receive call allocates the buffer space needed by the target program to store the data. Figure 5–2 is a flowchart that outlines this procedure.

Other DECnet implementations allow the source NSP to transmit data over any logical link as long as the target NSP has access to enough system buffer space to hold the data. The NSP module at the target node then delivers the data it has received when the appropriate target program allocates its own buffer space by issuing one or more receive calls.

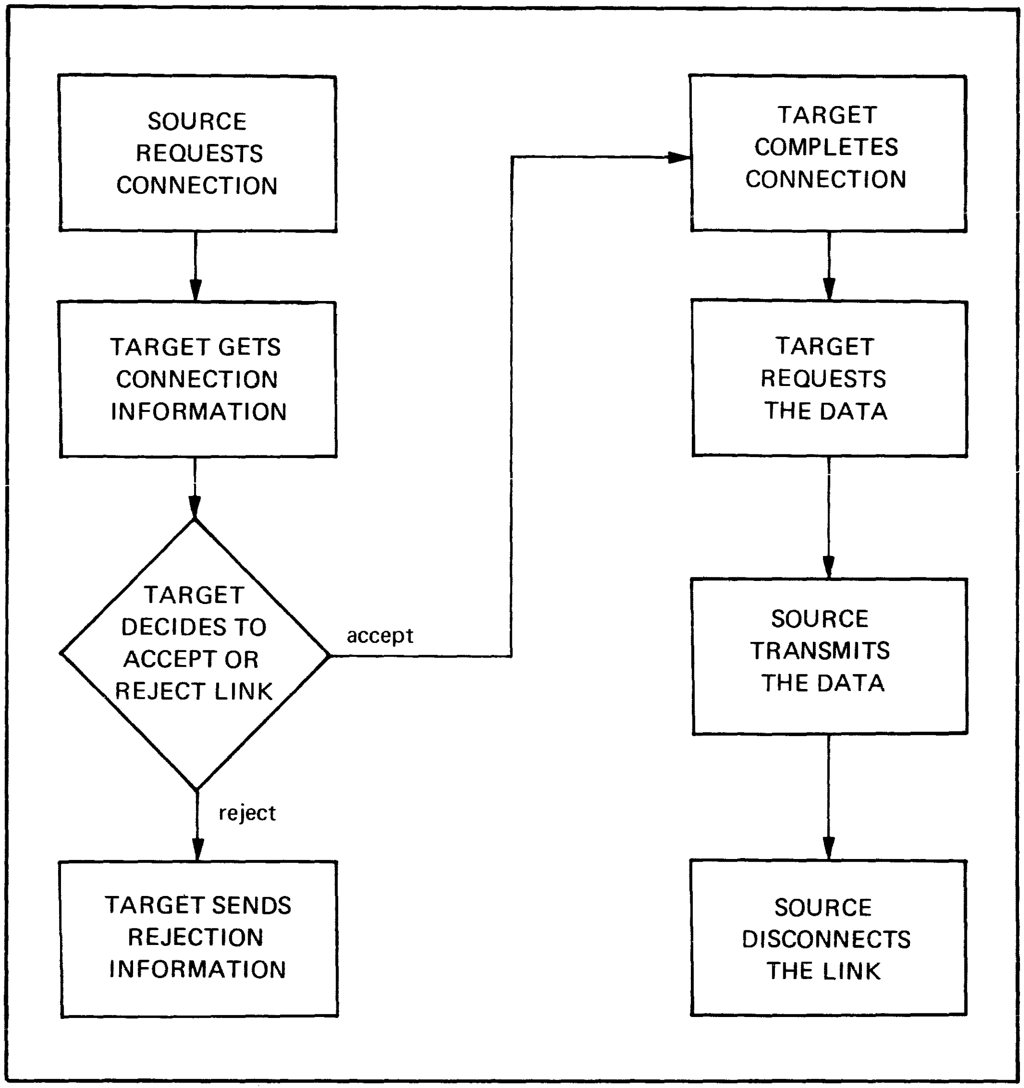

SOURCE REQUESTS CONNECTION
TARGET GETS CONNECTION INFORMATION
TARGET DECIDES TO ACCEPT OR REJECT LINK
accept
reject
TARGET SENDS REJECTION INFORMATION
TARGET COMPLETES CONNECTION
TARGET REQUESTS THE DATA
SOURCE TRANSMITS THE DATA
SOURCE DISCONNECTS THE LINK

Figure 5–2: Transmitting Normal Data

Regardless of the implementations involved, the NSP modules in the source and target nodes exchange link service messages to determine whether the target is prepared to receive a message. This precaution is part of DECnet's flow control mechanism, described in Section 4.7.

5.4.2 Interrupt Data

Interrupt data can consist of up to 16 bytes of information to be delivered immediately to the target program. If the target NSP module has a queue of normal data already received, but not yet processed, the interrupt data is placed either at the head of that queue or in a separate queue that the target program can access without first reading the normal data.

5.5 Disconnecting the Link

Either program can issue a call at any time to disconnect the link in one of two ways. One way, which disconnects the link in an orderly fashion, is normally used by a program to terminate a session that has proceeded as expected. All pending transmissions are completed before the link is dissolved. The programmer must decide which of the two programs disconnects under normal circumstances. When discussing these orderly disconnections, the various DECnet user's guides call them disconnects or synchronous disconnects.

The second way to disconnect forces the link to be terminated whether or not the remote NSP has acknowledged previously transmitted data. In most user's guides, this method of disconnecting the link is called aborting the link. When the caller's NSP receives notification to abort a link, it cancels all messages waiting to be transmitted over the link. A program may choose to abort in response to some unusual system event, like an impending emergency shutdown.

Whether a program simply disconnects or aborts the link, it can simultaneously send up to 16 bytes of data to the other program.

5.6 Summaries of Task-to-Task Communication Calls

This section summarizes the task-to-task communication calls provided by DECnet implementations.

5.6.1 DECnet–RT Calls

DECnet–RT programmers can use either MACRO–11 or FORTRAN–IV for task-to-task communication. Table 5–2 lists all the task-to-task calls, which are the same as those used by RSX DECnet and DECnet–IAS programmers.

Table 5–2: Task-to-Task Calls for DECnet–RT, RSX DECnet, and DECnet–IAS

Call*	Function	Description
OPNx	Access network services	Grants the program access to network services and creates the program's data queue for holding incoming messages.
CONx	Connect request	Requests a logical link connection.
CONB$ BACC BFMT0 BFMT1	Build connect block	These are calls issued to build the connect block referred to in a connect request call. MACRO–11 programs use the CONB$ call, and all higher level languages use the remaining calls.

(continued on next page)

* The lower case x concluding most calls is a variable determined by the source language of the program using the call. In MACRO–11 programs, x equals $ (OPN$, for example). In FORTRAN, BASIC–PLUS–2, COBOL, or CORAL programs, x equals NT (OPNNT, for example).

Table 5–2 (Cont.): Task-to-Task Calls for DECnet–RT RSX DECnet, and DECnet–IAS

Call*	Function	Description
GNDx	Get network data	Retrieves unsolicited messages from the program's network data queue. Unsolicited messages include connect requests, interrupt messages, and disconnect or abort messages.
ACCx	Connect accept	Accepts a logical link connection request.
REJx	Connect reject	Rejects a logical link connection request.
SNDx	Send data	Sends data over a logical link.
XMIx	Send interrupt data	Sends interrupt data over a logical link.
RECx	Receive data	Receives data over a logical link.
DSCx	Disconnect	Disconnects a logical link.
ABTx	Abort	Aborts a logical link.
CLSx	End network activity	Ends a program's network activity; the converse of the OPNx call.

* The lower case x concluding most calls is a variable determined by the source language of the program using the call. In MACRO–11 programs, x equals $ (OPN$, for example). In FORTRAN, BASIC–PLUS–2, COBOL, or CORAL programs, x equals NT (OPNNT, for example).

5.6.2 DECnet/E Calls

For DECnet/E programmers, task-to-task calls are available for use with several source languages: BASIC–PLUS or BASIC–PLUS–2, FORTRAN or FORTRAN–IV–PLUS, COBOL, or MACRO–11. In addition, DECnet/E provides a set of five task-to-task calls for use in COBOL programs only. These calls make up the concise COBOL interface, which allows the programmer to regard the network as a file and data sent and received as records. A COBOL program using the concise interface can have only one logical link running at a time.

Table 5–3 lists and describes the full set of DECnet/E calls, and Table 5–4 summarizes the smaller set of calls for the concise COBOL interface.

Table 5–3: DECnet/E Task-to-Task Calls

Call	Function	Description
MDCL	Access network services	Registers the program with the operating system for send/receive services.
MSLD	Send local data	Transmits user data to a local program.
NTLN	Get local node parameters	Returns information to the calling program concerning the local node's network parameters.

(continued on next page)

Table 5–3 (Cont.): DECnet/E Task-to-Task Calls

Call*	Function	Description
NTEV	Log user event	Permits a user-written program to queue an event to the system event processor for logging.
NTCI	Connect request	Requests a logical link connection.
NTCC	Connect accept	Accepts a logical link connection requested by another program.
NTCR	Connect reject	Rejects a logical link connection requested by another program.
NTDM	Send network data	Transmits user data to a network program over an established logical link.
NTIN	Send interrupt data	Transmits interrupt data to a network program over an established logical link.
NTLS	Link service	Requests data over a flow-controlled logical link.
NTDI	Disconnect	Disconnects an established logical link, after all pending messages have been sent.
NTLA	Abort	Disconnects an established logical link immediately, destroying any messages waiting to be sent.
MRCV	Receive data	Receives a message from the queue of pending messages.
MREM	Remove receiver	Terminates send/receive operations.

Table 5–4: DECnet/E Concise COBOL Interface Calls

Call	Function	Description
CNTCON	Connect request	Requests a logical link connection.
CNTACP	Connect accept	Accepts a logical link connection request from another program in the network.
CNTSND	Send "record"	Sends up to 32,767 bytes of data to a remote program over an established logical link.
CNTRCV	Receive "record"	Delivers up to 32,767 bytes of data to the calling program.
CNTDIS	Disconnect	Disconnects an established logical link.

5.6.3 RSX DECnet Calls

RSX DECnet provides task-to-task calls available to five programming languages: MACRO-11, FORTRAN-IV, FORTRAN-IV-PLUS, BASIC-PLUS-2, and COBOL. These calls, which are basically the same for all five languages, are summarized in Table 5–2.

5.6.4 DECnet–IAS Calls

DECnet–IAS provides task-to-task communication calls for use in MACRO–11, FORTRAN–IV, FORTRAN–IV–PLUS, BASIC–PLUS–2, COBOL and CORAL programs. The calls are identical to those used in MACRO–11 or FORTRAN programs written to run at an RSX DECnet node (see Table 5–2).

5.6.5 DECnet–VAX Calls

DECnet–VAX supports two forms of task-to-task communication: transparent and nontransparent. Transparent communication is the simpler form, in which a program can send and receive only normal data. MACRO, FORTRAN, BASIC, COBOL, PASCAL, PL/I, and BLISS can all be used to write this kind of program. Table 5–5 lists the system service calls used by a program to perform transparent communication. These calls are described in the appropriate language user's guide or reference manual. Programs written in higher level languages also can use standard sequential I/O statements to perform transparent task-to-task communication. For example, VAX–11 FORTRAN programs use the OPEN, READ, WRITE, and CLOSE statements to perform task-to-task communication.

Table 5–5: DECnet–VAX Transparent Task-to-Task System Service Calls

Call	Function	Description
$CREATE/$OPEN	Connect request	Requests a logical link connection and assigns a channel number to the requested link.
$OPEN	Connect accept	Accepts a connect request and assigns a channel number to the link.
$PUT	Send data	Sends data.
$GET	Receive data	Receives data.
$CLOSE	Disconnect	Disconnects the logical link immediately.

All programs also can perform nontransparent communication, which allows increased control over network operations and more flexibility in program development. This form of communication allows a program to use multiple logical links, to send interrupt as well as normal data, and to disconnect a link either synchronously or immediately (an abort). Table 5–6 summarizes the calls that allow a system service program to perform nontransparent communication.

5.6.6 DECnet–20 Calls

DECnet–20 allows a program written in MACRO–20 to perform task-to-task communication. Table 5–7 summarizes the calls such a program uses to communicate over the network.

Table 5–6: DECnet–VAX Nontransparent Task-to-Task System Service Calls

Call	Function	Description
$CREMBx $ASSIGN $QIO (IO$_ACCESS)	Connect request	These three calls perform the functions necessary to request a logical link connection using nontransparent communication: (1) create a 'mailbox' for queuing unsolicited incoming messages, (2) assign an I/O channel number to the network, and (3) request a logical link connection to a target program.
$QIO (IO$_ACPCONTROL)	Declare network name	Assigns a network name to the issuing program, making it eligible to accept multiple connect requests.
$QIO (IO$_ACCESS)	Connect accept	Accepts a logical link connection request.
$QIO (IO$_ACCESS !IO$M_ABORT)	Connect reject	Rejects a logical link connection request.
$QIO (IO$_WRITEVBLK)	Send data	Sends data.
$QIO (IO$_WRITEVBLK !IO$M_INTERRUPT)	Send interrupt data	Sends interrupt data to the target task (which is not possible via DECnet–VAX transparent communication).
$QIO (IO$_READVBLK)	Receive data	Receives data.
$QIO (IO$_DEACCESS !IO$M_SYNCH)	Disconnect	Disconnects the logical link in an orderly fashion, that is, synchronously.
$QIO (IO$_DEACCESS !IO$M_ABORT)	Abort	Disconnects a logical link immediately.
$DASSGN		Disconnects a logical link immediately.

Table 5–7: MACRO–20 Task-to-Task Calls

Call	Function	Comments
GTJFN	Program prepares itself for becoming a target task	This GTJFN call assigns a JFN* to SRV:, which is a logical device name that represents the target task.
GTJFN	Source program identifies target	This GTJFN call assigns a JFN to DCN:, which is a logical device name for the logical link to be requested.
OPENF	Source program requests connection with a target	This OPENF call specifies the JFN of the logical/device name DCN:, associated with the target task in a previous GTJFN call.
OPENF	Target program declares its readiness to receive connect requests	This OPENF call specifies the JFN of SRV: as defined by a previous GTJFN call.
MTOPR	Program enables itself to receive network interrupt data	This MTOPR call assigns a channel for receiving interrupt data. The channel is associated with the JFN of SRV: if the issuing program was originally the target or with the JFN of DCN: if the issuing program was originally the source.
MTOPR	Program reads network data received over the link	This MTOPR call specifies the appropriate JFN for the link and a function code that determines the type of network data to be read. The name of the source program, the various components of access control information, and interrupt data are some examples of types of network data.
MTOPR	Target program accepts or rejects a connect request	This MTOPR call specifies the JFN of SRV: and the function code for accepting or rejecting a connect request.
1. SOUTR 2. SOUT	Program sends normal data: 1. as individual messages 2. as a continuous byte stream	The sending and receiving programs must coordinate SOUTR calls with SINR calls or SOUT calls with SIN calls; that is, the programs must agree on how to send/receive normal data: 1. as individual messages or 2. as a continuous byte stream.
1. SINR 2. SIN	Program receives normal data: 1. as individual messages 2. as a continuous byte stream	
MTOPR	Program sends interrupt data	This MTOPR call specifies a JFN and the function code that indicates interrupt data.
MTOPR	Program terminates the connection	This MTOPR call specifies a JFN and the function code for terminating the connection.

* JFN stands for Job File Number. DECnet–20 programs handle logical links as if they were files.

Chapter 6
Remote File Access

Using DECnet, a program in one node can access a file in another node, despite differences in the two node's operating and file systems. This remote file access capability has the following applications:

- A user-written program can incorporate DECnet I/O calls that allow it to perform record-level operations on remote files (see Section 6.3).
- A terminal user can run a DECnet utility or issue a command to manipulate remote files (see Section 6.7).
- A terminal user can run a DECnet utility or issue a command to execute a command file in a remote node (see Section 6.7.4).

Like other DECnet functions, remote file access requires the cooperation of two network programs: A program in one node issues a remote file access request, and in the target node, a DECnet program receives the file access request and translates it into a form recognizable to the local file system. Before the first program can issue the remote access request, the two programs establish a logical link between them by exchanging handshake messages. The program making the request is the source; the program receiving the request is the target.

6.1 The File Access Listener (FAL)

In the context of remote file access, the source program is the accessing program, and the target program is a DECnet system program called the File Access Listener (FAL). The accessing program can be a user-written program, a DECnet utility, or a system command, depending on the application. The target program is always a version of FAL, whose role is to receive remote access requests from the network. FAL completes connections initiated by remote accessing programs and translates the incoming requests into calls to the file system at FAL's node. FAL then sends the resulting file data back to the accessing program, where special routines reformat the data as required to make it conform to local file structures.

Figure 6–1 illustrates the role of FAL and other software components in remote file access.

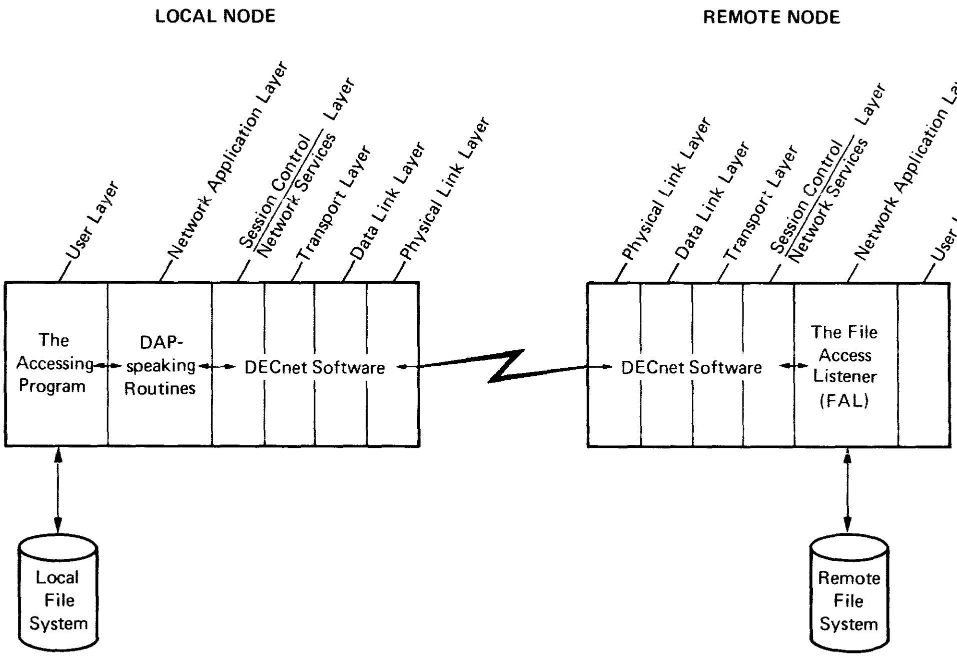


Figure 6–1: Remote File Access

File data can flow in either direction between the accessing program and FAL. The file accessed can reside on a mass storage device like a disk or a magnetic tape, or it can be associated with an I/O device such as a line printer or a terminal. The actual file operations that an accessing program can perform depend on the capabilities of both nodes.

6.2 The DAP Interface

FAL and the accessing program exchange Data Access Protocol (DAP) messages to perform remote file access operations. DAP resides in the network application layer of the DNA architecture and uses the logical link services of NSP. DAP defines a set of messages that controls the execution of remote file access and outlines procedures to accomplish specific file operations. For example, to create a file. an accessing program and a remote FAL must exchange a subset of DAP messages in a prescribed sequence. For more detailed explanations, see the DAP specification.

A user level program does not handle DAP messages directly. All DECnet implementations include system software that implements DAP functions by sending and receiving DAP messages on behalf of the users. The FAL module residing in each DECnet node provides the passive DAP function, which is to receive and process file access requests from elsewhere in the network. The accessing program can be one of the following types of software module:

- A version of the Network File Transfer (NFT) utility
- A system command
- A user-written program that accesses remote files via calls to DECnet-subroutines or to subroutines provided by the local file system.

Section 6.3 discusses the subroutines that a program can call to gain access to remote files. Section 6.7 describes how to access remote files from a terminal.

6.3 Programming Remote Access

To gain access to a remote file, a user program incorporates DECnet I/O calls, which activate the remote file access subroutines. The function of these subroutines is to build, send, and interpret DAP messages. In several DECnet systems, they are called Network File Access Routines (NFARs). Calls to the NFARs are different from calls that access a program's local file system.

In some systems (DECnet–VAX, for example), remote file access is transparent to the accessing program because DAP functions are incorporated in the file system. A program uses the same I/O calls regardless of the accessed file's location within the network. To access a remote rather than a local file, a program simply includes a node identifier in the specification of the file to be accessed. The file system transparently performs the same function that the NFARs do in other DECnet systems.

The DECnet systems that allow a user program to access a remote file are

- RSX DECnet
- DECnet–IAS
- DECnet–VAX
- DECnet–RT

All of these systems include facilities that allow a user program to perform the following operations on a remote file:

- Open an existing file
- Create a new file
- Read records from a file
- Write records to a file
- Close a file
- Delete a file

In addition to these operations, specific DECnet systems allow user programs to manipulate remote files in other ways. Section 6.6 summarizes the remote file access calls and operations made available by each DECnet implementation.

Programmer reference manuals for each DECnet implementation describe how to use the remote file access calls available to user programs (see Appendix A). Sections that follow discuss factors in preparing to write remote file access programs that apply to all DECnet implementations.

6.4 File System Capabilities

A programmer needs to be familiar with the file system resident at the target node. File organization, access modes, and other characteristics are dependent on the type of file system that an operating system supports. When the local and remote nodes have the same operating and file systems, programming remote access is similar to programming local I/O operations, especially if the local file system itself includes DAP capabilities (VAX-11 RMS). However, when the remote access bridges different types of operating systems, the programmer faces certain variables and restrictions. And at some nodes that run the same DECnet implementation, different DAP capabilities can result from different configuration choices. For example, RSX DECnet provides two different FALs. One is based on FCS-11 and provides basic FCS file services only, while the other is based on RMS-11 and provides a full set of record access services.

Generally, the theory of the lowest common denominator applies: An accessing program can perform those functions that its source language provides and that the remote file system supports. For example, a VAX-11 program cannot use all the remote access functions provided by DECnet-VAX if the target node runs RSX-11M.

By cross-checking the file system characteristics with the available remote access calls, a programmer can find out the kinds of file operations that are possible between two different DECnet implementations. This information is provided in Table 6-1, which names the file system supported by each DIGITAL operating system, and in Table 6-2, which summarizes the remote file and record access capabilities of the various DECnet implementations. Section 6.6 describes the remote access calls provided by each DECnet implementation.

Table 6-1: DIGITAL File and Record Management Systems

File Systems	Operating Systems	Record Management System	Operating System
Files-11	RSX-11M RSX-11M-PLUS IAS VAX/VMS	RMS	RSX-11M RSX-11M-PLUS VAX/VMS RSTS/E IAS
RSTS/E	RSTS/E		
RT-11	RT-11		
TOPS-20	TOPS-20		

Table 6–2: DECnet Remote File and Record Access Capabilities

		RSX DECnet DECnet-IAS NFARs	DECnet-11M DECnet-11M-PLUS DECnet-IAS FCS FAL	DECnet-11M DECnet-11M-PLUS RMS FAL	DECnet-11S FAL	DECnet-VAX	DECnet-RT	DECnet/E	DECnet-20 Incoming	 Outgoing
Data Types	ASCII	X	X	X	X	X	X	X	X	X
	Image	X	X	X	X	X	X	X	—	X
File Organization	Sequential	X	X	X	X	X	X	X	X	X
	Relative	X[1,12]	—	X	—	X	—	X[12]	—	—
	Indexed	X[1,12]	—	X	—	X	—	X[12]	—	—
Access Methods	Sequential	X	X	X	X	X	X	X[1]	X	X
	Random	—	—	X	—	X[2]	X[2,4]	—	—	—
Record Formats, ASCII and Image	Fixed length	X	X	X	X	X[7]	X	X[1,6]	—	X
	Variable length	X	X	X	X	X[7]	X[5]	X[1,6]	—	X
	VFC	X[13]	X[12]	X	X[13]	X[7]	—	X[1,6]	—	—
	Stream ASCII	X	—	X	—	X	X	X[1,6]	X	X
Transmission Mode	Record	X	X	X	X	X	X	X[1]	X	X
	Block	X[1]	—	X	—	X	X	X[1]	—	—
Remote File Operations	Command/batch file submission	X[3]	X	X	—	X	X[8]	X[9]	X[9]	X[9]
	Command/batch file execution	X[3]	X	X	—	X	X[8]	X[9]	X[9]	X[9]
	File deletion	X	X	X	—	X	X	X[1]	X	X
	File retrieval	X	X	X	X	X	X	X[1]	X[11]	X
	File storage	X	X	X	X	X	X	X[1]	X[11]	X
	Record storage	—	—	X	—	X[2]	X[2,10]	X	—	—
	Record retrieval	—	—	X	—	X[2]	X[2,10]	X	—	—
	Directory	X[1]	X	X	—	X	X	X	X	X
	Rename	X[1]	—	X	—	—	—	—	—	—

[1] User interface only (that is, NFT or VAX/VMS commands)
[2] Programmable interface only
[3] ASCII data type only
[4] Fixed length sequential only
[5] Interprets as stream ASCII from DECnet-11M,-11M-PLUS,-IAS,-VAX
[6] Sequential access only
[7] Sequential and random access
[8] Only valid to nonhomogeneous systems
[9] Stream ASCII only coming into DECnet/E; any form going out
[10] Sequential and random access on sequential files only
[11] Stream ASCII only
[12] Block mode only
[13] 2-byte header only

6.5 Initiating the Remote Access

Like task-to-task communication, remote file access requires a handshaking sequence at the beginning of the operation. Not only does DECnet software set up a logical link between the accessing program and the remote FAL, it also exchanges initial DAP messages to prepare for the file operation to be performed over the link.

This extended handshake, which is transparent to the accessing program, happens automatically when the program issues a call to open a remote file. The form of the open call varies from system to system and from language to language, but the call always provides much of the information exchanged in the handshake. Using the information supplied by the call as well as system-supplied data about the local file system, the remote access subroutines — either NFARs or routines in the local file system — generate DAP messages addressed to the remote FAL. In response, the FAL sends back DAP messages to define characteristics of its local file system.

For most remote access operations, the open call issued by the accessing program includes the following information:

- A logical unit or channel number
- A file specification
- Access control information
- Characteristics of the file to be accessed

Depending on the open call's particular function, for example, open for reading or open for appending, the call passes other information as well.

6.5.1 The Logical Unit/Channel Number

The logical unit or channel number serves one and sometimes two purposes in remote file access operations. It identifies the data stream associated with the requested I/O operation, and in RSX and IAS nodes, the local NSP module treats the number as the program's logical link identifier.

6.5.2 The File Specification

The file specification identifies the remote file to be accessed. Because the remote file system actually carries out the requested file operation, the programmer must know how the file is identified by users in its local node. Table 6–3 shows the file specification formats used by each operating system.

Table 6–3: File Specifications for DIGITAL Operating Systems

Operating System	**File Specification**	**Examples**
RSX–11M RSX–11M–PLUS RSX–11S * IAS	*dev:[ufd]filename.typ;ver*	DK0:[200,200]PROG7.MAC:1 MT3:[1,7]ACCNT.DAT;13 DB1:[300,316]PEEK.LST;2
VAX/VMS	*dev:[directory]filename.typ;ver*	DBA0:[HIGGINS]STAT.FTN;1
	or	
	dev:<directory>filename.typ;ver	MTA3:<CHARLES>TEST.DAT.3
RT–11	*dev:filnam.typ*	RK0:CHART1.DAT SY0:TEXT.RNO
RSTS/E	*dev:[ppn]filename.typ*	SY:[216,212]AFIL.BAS DK1:NEWFIL.COB MT1:BCKUP.LST
TOPS–20	*dev:<directory>filename.typ.ver*	PS<FRED>FISCAL.EXE;1 PS:<JDOE>PROD81.LST;12

[*] RSX–11S supports unit record devices only.

6.5.3 Access Control Information

The access control information identifies the program to the remote system and consists of:

- A user identification code or name
- A password associated with the user identification
- Additional accounting information as required by the remote system

If this information matches an account or guest account entry in the remote system's user file, the program gains access to that system's resources. Like the file specification, the access control information must be recognizable to the remote system, and therefore specified according to its syntax.

Gaining access to the remote system does not guarantee that requested file operations will succeed. In most DIGITAL operating systems, each file has a corresponding protection code that determines the types of access allowed to defined groups of users. The user identification — a code or a name — specified by the accessing program determines the program's group category and therefore determines the types of access it can make to each file.

The RT–11 file system enforces a simpler version of access control and file protection than that used by other DIGITAL file systems. To access files at a DECnet–RT node, a remote user supplies a password, which is either privileged or nonprivileged. The privileged password allows the remote user unrestricted access to the node's files, whereas the nonprivileged password allows read and directory access only. Furthermore, if a file is marked protected, privileged and nonprivileged users alike are allowed read access only (see Section 6.7.2).

6.5.4 File Characteristics

The file characteristics define the file to be accessed in the following ways:

- **Access method — sequential or random.** The accessing program indicates how it will access the file. All DIGITAL file systems support sequential access in which each I/O operation reads or writes the next record. Selected file systems permit random access, which allows the program to access a specific record anywhere in the file (refer to Table 6–2).
- **File organization — sequential, relative, or indexed.** Sequential file organization means that records in a file are arranged in one after the other fashion. This organization is supported for all types of devices. Relative file organization means that records within a file are identified by a relative record number. This number identifies the record's position relative to the beginning of the file. Indexed organization (RMS only) is a complex file structure that allows both sequential and random access and uses record keys for identification. The keys used to identify individual records are defined at file creation. Relative and indexed organizations are supported for disk devices only.

- **Data type — ASCII or image.** ASCII data is subject to formatting conversion by the DECnet software, depending on the data's record attributes (see below). Image data is a stream of bits, to which the software applies no interpretation.
- **Record format — fixed length, stream, variable length, or variable with fixed length control (VFC).** A VFC record includes a fixed length control field in addition to the variable length data portion.
- **Record attributes.** This characteristic indicates the type of vertical format control that applies to the file.

6.6 DECnet Remote File Access Calls

This section summarizes the remote file access calls provided by DECnet implementations.

6.6.1 DECnet-RT Calls

DECnet-RT programmers can use either MACRO-11 or FORTRAN-IV to write remote file access programs. Table 6-4 summarizes the macro calls available for remote file access, and Table 6-5 summarizes the FORTRAN calls. The MACRO-11 interface allows both sequential and random access operations, while the FORTRAN interface allows sequential access only.

Table 6-4: DECnet-RT Remote File Access Macro Calls

Macro Call	Function
.NLOOKUP	Opens an existing file.
.NENTER	Opens a new file.
.NAPPEND*	Opens an existing file for appending records (sequential access only).
.NREAD	Reads a record from a remote file (sequential or random access).
.NWRITE	Writes a record to a remote file (sequential or random access).
.NSPOOL*	Opens a new file and spools it to a line printer on closing.
.NCLOSE	Closes a remote file.
.NDELETE	Deletes a remote file.
.NPURGE	Cancels the remote file access and restores affected files to their original state.
.NSUBMIT*	Opens a new remote command file for submission.
.NEXECUTE*	Executes a remote command file.

* This call cannot be used if the remote node runs DECnet-RT.

Table 6–5: Higher Level Language Remote File Access Calls (DECnet–RT, DECnet–IAS, and RSX DECnet)

Call	Function
OPWNFW	Creates and opens a remote sequential file for writing.
OPRNFW	Opens an existing remote sequential file for reading.
OPANFW	Opens an existing remote sequential file for appending.
GETNFW	Reads a record from a remote file.
PUTNFW	Writes a record to a remote file.
PRGNFW*	Discards a previously opened file.
CLSNFW	Closes a remote file.
DELNFW	Deletes a remote file.
SPLNFW*	Opens, writes, and prints a file.
EXENFW*	Executes an existing command file.
SUBNFW*	Opens, writes, and executes a command file.

* DECnet–IAS does not support these calls.

6.6.2 RSX DECnet Calls

RSX DECnet supports remote file access from programs written in FORTRAN-IV, FORTRAN-IV-PLUS, COBOL, or BASIC-PLUS-2. The remote access calls that an RSX programmer can use are summarized in Table 6–5.

6.6.3 DECnet–IAS Calls

Programs accessing remote files from a DECnet–IAS node may be written in FORTRAN-IV, FORTRAN-IV-PLUS, COBOL, BASIC-PLUS-2, or CORAL. The calls are the same as those available to RSX DECnet programmers and are listed in Table 6–5.

6.6.4 DECnet–VAX Calls

VAX–11 RMS includes routines that implement DAP functions. Thus a programmer can use the same calls to access remote as well as local files. The programmer simply includes a node identifier in the file specification. All native mode languages can be used to write programs that access remote files. Both MACRO and higher level language programs directly incorporate calls to RMS routines; these calls are summarized in Table 6–6. Programs written in the higher level languages use standard I/O calls, which are described in the appropriate language user's guide or reference manual.

Table 6–6: VAX–11 RMS File Access Calls

Type of Processing	Call	Function
File Processing	$CREATE	Creates a new file and leaves it open.
	$OPEN	Opens a file for subsequent processing.
	$CLOSE	Closes a file and terminates file processing.
	$DISPLAY	Retrieves file attribute information about a file.
	$EXTEND	Increases the amount of space allocated to a VAX-11 RMS disk file.
	$ERASE	Deletes a closed file and removes its directory entry.
	$PARSE	Parses a file specification.
	$SEARCH	Searches a directory for a file name.
Record Processing*	$CONNECT	Establishes a record stream to an open file.
	$DISCONNECT	Terminates a record stream from an open file.
	$GET	Retrieves a record from the file.
	$PUT	Inserts a record into the file.
	$UPDATE	Modifies a record in the file.
	$DELETE	Deletes a record from the file (for relative and indexed file organizations only).
	$WAIT	Waits for an asynchronous I/O completion.
	$FIND	Positions a pointer to a specified record in the file.
	$REWIND	Positions to the first record of the file.
	$RELEASE	Unlocks a record.
	$TRUNCATE	Truncates a sequential file.
	$FLUSH	Forces blocked records to be written to the file.
	$FREE	Unlocks all previously locked records.
Block I/O Processing	$READ	Reads data in block I/O mode.
	$WRITE	Writes data in block I/O mode.
	$SPACE	Spaces forward or backward in the file.

* All record-processing calls except $DELETE can be used for either sequential or random access operations.

6.7 Accessing Remote Files from a Terminal

All DECnet implementations support terminal-based access of some kind to remote files. Except for DECnet–VAX, all the implementations use the Network File Transfer (NFT) utility to perform this function. DECnet–VAX uses VAX/VMS commands, which can manipulate both local and remote files. A terminal user can perform the following file operations:

- Transfer (copy) a file to or from a remote node or between two remote nodes.
- Delete a remote file.
- Submit a local command file for execution at a remote node or execute a command file already existing at a remote node. (Command files cannot be submitted to or executed at a DECnet–RT node.)
- Append one or more local or remote files to an existing local or remote file (not supported by DECnet–20).
- Obtain listings of remote directories. A directory file lists all the files residing on a device that belongs to a specific user or category. One of the fields in a file specification indicates the directory in which that file is listed.

  For example, the directory for the file specified as DB1:[301,15]FILE.DAT;3 is indicated by the code [301,15]. In RSX and IAS systems, the code is called a user file directory (ufd); the same type of code is called a project-programmer number (ppn) in RSTS/E systems. The corresponding field in a VAX/VMS specification is called the directory field; it can be a code like a uic or ppn, or it can be an alphanumeric string, for example, DBA2:[SMITH]FILE.DAT;2.
- Queue one or more files to a line printer. The files can be remote and the printer local, or the files can be local and the printer remote (not supported by DECnet–20).

On behalf of the terminal user, NFT creates a logical link between itself and the remote FAL. From the input a user types at the terminal, NFT formulates the appropriate DAP messages, which it then sends over the link. In turn, FAL interprets the DAP messages it receives and interfaces with its local file system as requested. FAL then returns information and any requested file data to NFT by sending back DAP messages.

Normal VAX/VMS commands can be used to gain access to remote files. To do so, a terminal user or batch job simply enters a network file specification rather than a local file specification as a command parameter. The commands that can be used on remote as well as local files are:

- COPY
- TYPE
- APPEND
- OPEN, READ, WRITE, CLOSE
- ANALYZE
- SEARCH

- DELETE
- PURGE
- DIRECTORY
- SUBMIT/REMOTE
- PRINT/REMOTE
- CREATE
- BACKUP
- DIFFERENCES
- DUMP
- PURGE
- CONVERT

When a user supplies a network specification with one of these commands, DAP-speaking routines within VAX–11 RMS send, receive, and interpret DAP messages to perform the requested remote file access.

Table 6–7 summarizes the remote file operations that a DECnet user can perform from a terminal. Under each implementation, the table shows the utility or system command used to perform a supported function.

6.7.1 Access Control

A terminal user must supply the same access control information that a program must pass to the NFARs or, where applicable, to the DAP-speaking routines in the local file system. This information — user identification, password, and optional account data — must be available whenever an NFT command line or a VAX/VMS command refers to a remote node. See Section 6.5.3 for an explanation of the significance of access control to remote file access.

NFT receives the access control information in various ways, depending on the implementation. Generally, the user supplies the information directly or uses default values that have been defined previously.

Depending on the implementation, the information may be supplied within a network file specification or in response to NFT prompts. Table 6–8 shows the various formats for specifying access control information within a file specification.

Each implementation also provides one or more ways to define default access control information. In some cases, the defaults must be defined each time the user logs into a system or each time the user runs NFT. In other cases the local system retains the default information beyond the duration of a single NFT or terminal session. Section 6.7.5 includes examples that show how users can default access control information.

A DECnet–VAX terminal user supplies access control information within a network file specification. The information can be included as part of a node specifier. Or a user can assign a logical name to a node name/access control string and include the logical name in the file specification.

Section 6.7.5 contains examples of NFT and VAX/VMS command lines and illustrates some of the ways a user specifies access control information.

Table 6–7: Remote File Operations from a Terminal

Function	RSX DECnet	DECnet–IAS	DECnet–VAX [1]	DECnet–RT	DECnet/E	DECnet–20
Transfer files (local←→remote)	NFT	NFT	COPY, TYPE commands	NFT	NFT	NFT [2]
Delete local or remote files	NFT	NFT	DELETE, PURGE commands	NFT	NFT	NFT
Append to an existing local or remote file	NFT	NFT	APPEND command	NFT [3]	NFT	not supported
Submit a local command or batch file for execution at a remote node	NFT	NFT	COPY, SUBMIT/REMOTE commands	NFT [3]	NFT	NFT
Execute a remote command or batch file	NFT	NFT	SUBMIT/REMOTE command	NFT [3]	NFT	NFT
Obtain a listing of a remote directory file	NFT	NFT	DIRECTORY command	NFT	NFT	NFT
Queue local or remote files to a local or remote line printer	NFT	NFT	PRINT/REMOTE	NFT [3]	NFT	not supported
Rename a remote file	NFT	not supported	not supported	not supported	not supported	not supported

[1] DECnet–VAX supports other VAX/VMS commands in addition to the ones listed in this table (Section 6.7).

[2] Only stream ASCII files can be copied to a remote DECnet–20 node.

[3] The remote node cannot be DECnet–RT.

Table 6–8: Specifying Access Control Information

DECnet Implementation	Format	Examples

RSX DECnet	*node/userid/password/account::*
	or
	node“userid password account”::
		`TEWKS/CHRIS/MAC::`
		`CONN"FREAN CMF"::`

DECnet-IAS	*node/userid/password/account::*	`BSTN/SMITH/RJS::`
DECnet-VAX	*node“userid password account”::*	`HTFD"CHARLES CAF"::`
DECnet-RT	*node//password/::* or *node//password::*	`NYC//CLAUDIA::`
DECnet/E	*node“userid password account”::*	`MILL"PERRY LAB"::`

6.7.2 File Protection

The access control information supplied locally by the terminal user determines the user's access rights at the remote node. As Section 6.5.3 describes, the remote file system compares the user's identification with the protection code associated with the file to be accessed. The file system carries out a requested access only if the protection code grants access to the identified user.

A DECnet-RT node protects files in two ways. First, a remote user must specify a privileged password to gain unrestricted access to any local files (see Section 6.5.3). A nonprivileged password grants a remote user read and directory access only. Second, the RT-11 file system restricts access to individual files marked protected. Protected status prevents a file from being deleted, duplicated, or manipulated in any way. So even a privileged remote user can only read a protected file.

6.7.3 Remote File Specifications

A specification that describes a file on a remote node must conform to that node's syntax rules. (Table 6-3 lists the specification formats defined by DIGITAL operating systems.) Unless the remote and local nodes run the same operating system, one file specification in a command line may be meaningful only to the remote node. The foreign specification may contain fields that the local system does not recognize, or the specification may be qualified by switches that cannot be interpreted by the local node.

To prevent the local node from checking a foreign syntax, some implementations require the terminal user to enclose the foreign specification in double or single quotes. (DECnet-RT and DECnet-20 do not require the user to set off foreign specifications in this way.)

The following examples illustrate NFT command lines that contain foreign specifications.

- Local node runs RSX DECnet. Remote node runs DECnet–VAX.

  ```
  NFT> ARGO/DRAKE/LAB/::"DBB0:[ARCH]"/LI
  ```

  In this example, the quoted string names a directory ([ARCH]) on the device DBB0: at the remote DECnet–VAX node. The NFT command switch /LI requests a listing of that directory at the local RSX node.

- Local node runs DECnet–RT. — Remote node runs RSX DECnet.

  ```
  NFT> COPY RK1:FAST.MAC NODEB::DK0:[200,200]FAST.MAC
  ```

  Even though the output file's directory field ([200,200]) is meaningless to RT–11, quotes to signal a foreign specification are not necessary.

- Local node runs DECnet/E. — Remote node runs DECnet–VAX.

  ```
  NFT> COPY DTVAX::"DMA2:[INVENTORY]TEST.DAT"=TEST.DAT
  ```

  This command copies a file from a DECnet/E node to a DECnet–VAX node. The output file specification, (DTVAX::"DMA2:[INVENTORY]TEST.DAT")contains VAX/VMS syntax understandable only to the target DECnet–VAX node. Quotes set off the foreign specification to prevent the local node from parsing it.

6.7.4 Remote Command File Submission

All versions of the NFT utility allow a terminal user to execute a command file in a remote system. A command file contains ASCII command lines equivalent to the command lines a user enters at a terminal. The remote node reads and executes these commands when a user submits the file. The COPY and SUBMIT/REMOTE commands perform this function when the local node runs DECnet–VAX: COPY transfers the command file to a remote node and the SUBMIT/REMOTE command submits the command file for execution there. If the remote node runs DECnet/E or DECnet–VAX, the file is submitted to a batch processor, and if the remote node runs DECnet–11M–PLUS or DECnet–IAS, submission to a batch processor is an option.

Remote command file submission and execution are valuable means of accessing remote system resources. At the local node a user can run a text editor to create an ASCII file of commands that conform to the syntax of the remote node's command language. Subsequently, the user can invoke the NFT utility (or the COPY and SUBMIT/REMOTE commands at a DECnet–VAX node) to copy the file to the remote node for execution there.

- **RSX DECnet.** If the remote node runs DECnet–11M, files are submitted to the RSX–11M indirect command file processor. This processor accepts special commands in addition to conventional MCR commands. The special commands provide for conditional processing and string substitution, among other features. At a DECnet–11M–PLUS node, command files are submitted either to the command processor or to a batch processor, depending on a choice made at network generation.

- **DECnet–IAS.** If the remote node runs DECnet–IAS, a user can submit command files either to the indirect command file processor or, optionally, to the batch processor.
- **DECnet–RT.** A DECnet–RT user can submit command files for execution at non-RT remote nodes, but RT–11 itself does not support command file execution.

6.7.5 NFT and VAX/VMS Command Examples

This section gives examples of NFT and VAX/VMS command lines and illustrates some system-dependent ways to specify access control information.

DECnet/E NFT

```
NFT>NODESPECIFICATION NODEB::
NODE:     NODEB
PPN:      [2,270]
PASSWORD: netcat
ACCOUNT:  5
NFT>APPEND NODEB::ACCT.BAS=AACCT.BAS,BACCT.BAS
NFT>DELETE NODEB::OLD.BAS
NFT>EXIT
```

In this example, a DECnet/E user preassigns access control information for NODEB (the first five lines). The next command copies two local files, AACCT.BAS and BACCT.BAS, to remote NODEB and appends them to the file ACCT.BAS. The user then deletes the remote file OLD.BAS and exits from NFT by typing the EXIT character.

The DECnet/E version of NFT retains access control information preassigned as shown above until the user terminates the NFT session. A more permanent way to preassign information is to run the DECnet/E NETACT utility. NETACT allows a user to define a file of access control information for one or more nodes. NFT automatically uses this file, if the current user has created one, to obtain required access control information. Whenever a command line refers to a remote node for which no access control information has been preassigned, NFT prompts for it.

DECnet–RT NFT

```
.RUN NFT
NFT>COPY/PRINT/ASCII ACNT.DAT BOS/KENT/JOE::[214,200]ACNT.DAT
NFT>APPEND/ASCII ACNT.DAT NYC/KENT/JOE::[100,104]ACBKUP.DAT
NFT>EXIT
```

The DECnet–RT implementation of NFT uses commands and optional switches to specify file operations. Within the NFT command line, the input file appears on the left and the output file on the right, with one or more spaces separating the files.

In this example, a DECnet-RT user runs NFT to copy a local ASCII file (ACNT.DAT) to a remote RSX node. The command switch (/PRINT) also causes the remote node to print out the copied file. The second command then appends the same local file to a remote file called ACBKUP.DAT at a DECnet/E node. In both command lines, the node specifier includes explicit access control information.

DECnet-RT also supports the creation of alias node names with associated default access control information. See the description of alias node names in the discussion of the RSX DECnet NFT example.

RSX DECnet or DECnet-IAS NFT

```
>NCP SET ALIAS BOSTON DESTINATION NODE4/PEEK/FREAN
>NFT
NFT>BOSTON::[21,5]PAY.LST=[21,10]PAY.LST
NFT>BOSTON::OLDPAY.LST;*/DE
NFT>[21,4]ORD.DAT=NY/PEEK/FREAN::[21,7]ORD.DAT
NFT>^Z
```

This sequence of commands shows two ways of specifying access control information. The first command invokes a network management utility called the Network Control Program (NCP). See Section 8.1. The command instructs NCP to assign the alias node name BOSTON to a node known as NODE4 and to associate the user name and password string /PEEK/FREAN with that alias. An alias node name is a temporary name that a user assigns to a node. As the example above shows, the NCP SET ALIAS command allows a user to associate default access control information with an alias. In the commands that refer to node BOSTON, NFT assumes that /PEEK/FREAN is the required access control information. The alias node name assignment lasts until the user logs off the terminal.

The last NFT command includes the access control information in the node specifier (NY/PEEK/FREAN::). This method applies when the node name (NY in this case) is not an alias node name or when the user wants to override access control information previously assigned to an alias.

The first NFT command transfers a local file to a file at remote node BOSTON and the second command deletes all versions of a file called OLDPAY.LST stored at node BOSTON. The asterisk (*) in the version field of the file specification specifies all versions of the file. The third NFT command transfers a file from a remote node called NY to the local node. Finally, the user terminates the NFT session by typing the CTRL/Z character.

DECnet-VAX commands

```
$ ASSIGN "NODEB""PEEK NET""::"  NB
$ COPY LOCSTAT.DAT;3  NB:REMSTAT.DAT
$ SUBMIT/REMOTE TRNTO"PEEK NET"::DMA3:[PEEK]ORDERS.COM
```

A DECnet-VAX terminal user provides access control information directly or indirectly in a network specification, as these commands illustrate. The ASSIGN command associates the logical name NB with the node name/access control string NODEB"PEEK NET"::. (Two sets of quotes are necessary only when the access control string is being equated with a logical name.) The COPY command that follows transfers the local file LOCSTAT.DAT to a file called REMSTAT.DAT on remote NODEB, referred to by means of the logical name NB.

The last command shows how the access control information can be included directly in the command line. The user's name and password are enclosed in quotes within the node specifier (TRNTO"PEEK NET"::). In response to this command, remote node TRNTO, a DECnet-VAX node, submits the command file ORDERS.COM to the batch processor. The command file must already be located at the remote node.

Chapter 7
DECnet Terminal Facilities

DECnet provides several terminal facilities for interactive access to the network. These include:

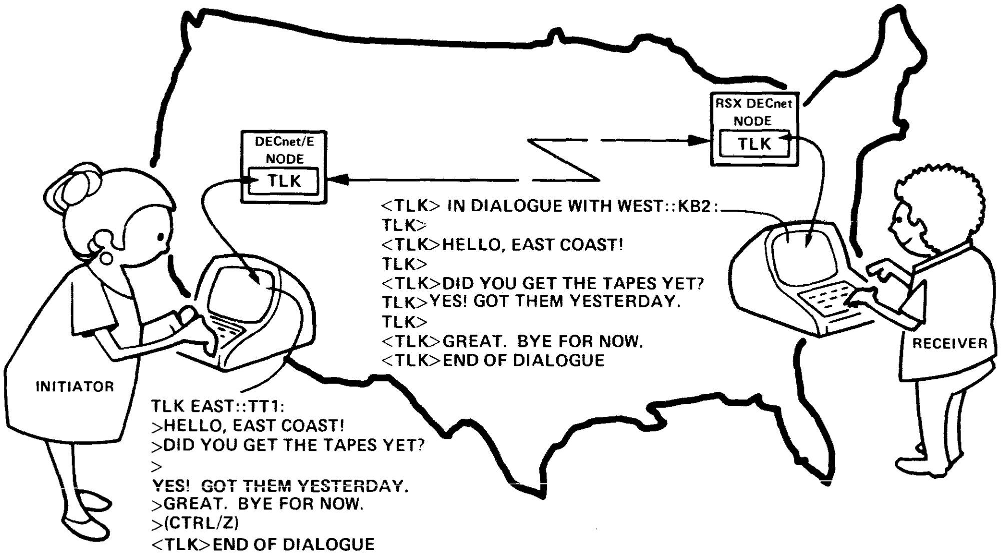
- Terminal-to-terminal communication, provided by the TLK utility or by the VAX/VMS PHONE command.
- Direct access to a homogeneous remote node's operating system, provided either by a network command terminal utility or by a VAX/VMS command.
- The VMSMAIL utility for DECnet–VAX users.
- Remote file access from a terminal, provided by the NFT utility or, for DECnet–VAX nodes, by VAX/VMS commands

This chapter presents overviews of terminal-to-terminal to communication and direct access to a remote node's operating system. Accessing remote files from a terminal is discussed in Section 6.7.

7.1 The TLK Utility and the VAX/VMS PHONE Command

All DECnet implementations, except DECnet–VAX and DECnet–20, support TLK, a means of communication between computer sites linked by DECnet. DECnet–VAX users can issue the VAX/VMS PHONE command to communicate with terminal users at other DECnet–VAX nodes (Section 7.1.4). A user can invoke TLK at a local terminal to exchange messages with other terminal users in the network (see Figure 7–1). The TLK utilities in the source and target nodes use NSP services to create a connection between the sending and receiving terminals.

In an initial command line to TLK, a user addresses messages to be sent by entering a node name and terminal identifier (TT4:, for example). Because the TLK utility does not access protected resources such as disk files, access control information is not required. By omitting a node name from the address, a user can also send messages to other terminals on the same local node. And if the terminal identifier is omitted, TLK sends the message to the operator's console on the remote node. In most cases, TLK breaks through any current terminal activity to deliver a message.



```
<TLK> IN DIALOGUE WITH WEST::KB2:
TLK>
<TLK>HELLO, EAST COAST!
TLK>
<TLK>DID YOU GET THE TAPES YET?
TLK>YES! GOT THEM YESTERDAY.
TLK>
<TLK>GREAT. BYE FOR NOW.
<TLK>END OF DIALOGUE
```

```
TLK EAST::TT1:
>HELLO, EAST COAST!
>DID YOU GET THE TAPES YET?
>
YES! GOT THEM YESTERDAY.
>GREAT. BYE FOR NOW.
>(CTRL/Z)
<TLK>END OF DIALOGUE
```

Figure 7–1: The TLK Utility

7.1.1 One-line Mode and Dialog Mode

All implementations of TLK support both a one-line message mode and a message dialog mode. A user can activate TLK to send a single message contained on one line or to start up a dialog. The dialog mode of TLK also establishes an interactive connection with the target terminal so that a user there can respond instantly to the TLK messages received. In the example in Figure 7–1, note that the TLK prompt (TLK>) appears in between lines displaying the received messages. These prompts give a user the opportunity to send messages back to the terminal that started the dialog (TT1: at NODEA in this case).

One-line mode examples:

- Display at sender's terminal (DECnet–RT)

```
.R TLK
TLK> GREEN__TT2:'TURN ON YOUR LINE PRINTER, CATHY.
```

Display at receiving terminal (RSX DECnet)

```
>
<TLK>MYNRD::TT0:'TURN ON YOUR LINE PRINTER, CATHY.
>
```

7.1.2 The TLK Split Screen Option

RSX DECnet and DECnet–IAS support a TLK option that splits a video screen in half to display the two parts of a dialog. TLK reserves the top half of the screen for the source user's input and the bottom half for the target user's responses. Figure 7–2 shows how a sample dialog might appear at two corresponding terminals.

The video terminal must be a VT52 or a VT100, and both the source and the target nodes must support TLK's dialog mode. However, it is not necessary for both nodes to support the split screen option, in which case the messages will appear on the remote terminal as for normal dialog.

7.1.3 TLK Command Files

The TLK utility implemented by DECnet–IAS and RSX DECnet can send messages read from a TLK command file. TLK command files are useful for sending many messages at once and for storing and sending sets of messages that need to be sent more than once. To prepare for sending messages in this manner, the user runs a text editor to create an ASCII file composed of TLK command lines. The format of the command lines depends on the desired transmission mode (one-line or dialog). Command lines within a command file conform to the same syntax as lines entered from a terminal.

In the following example, a system manager creates a file of TLK commands to be broadcast at the end of every work day. Each line looks like a command entered from a terminal for transmission in one-line mode. The messages remind operations staff at various sites within the network of a routine shutdown procedure.

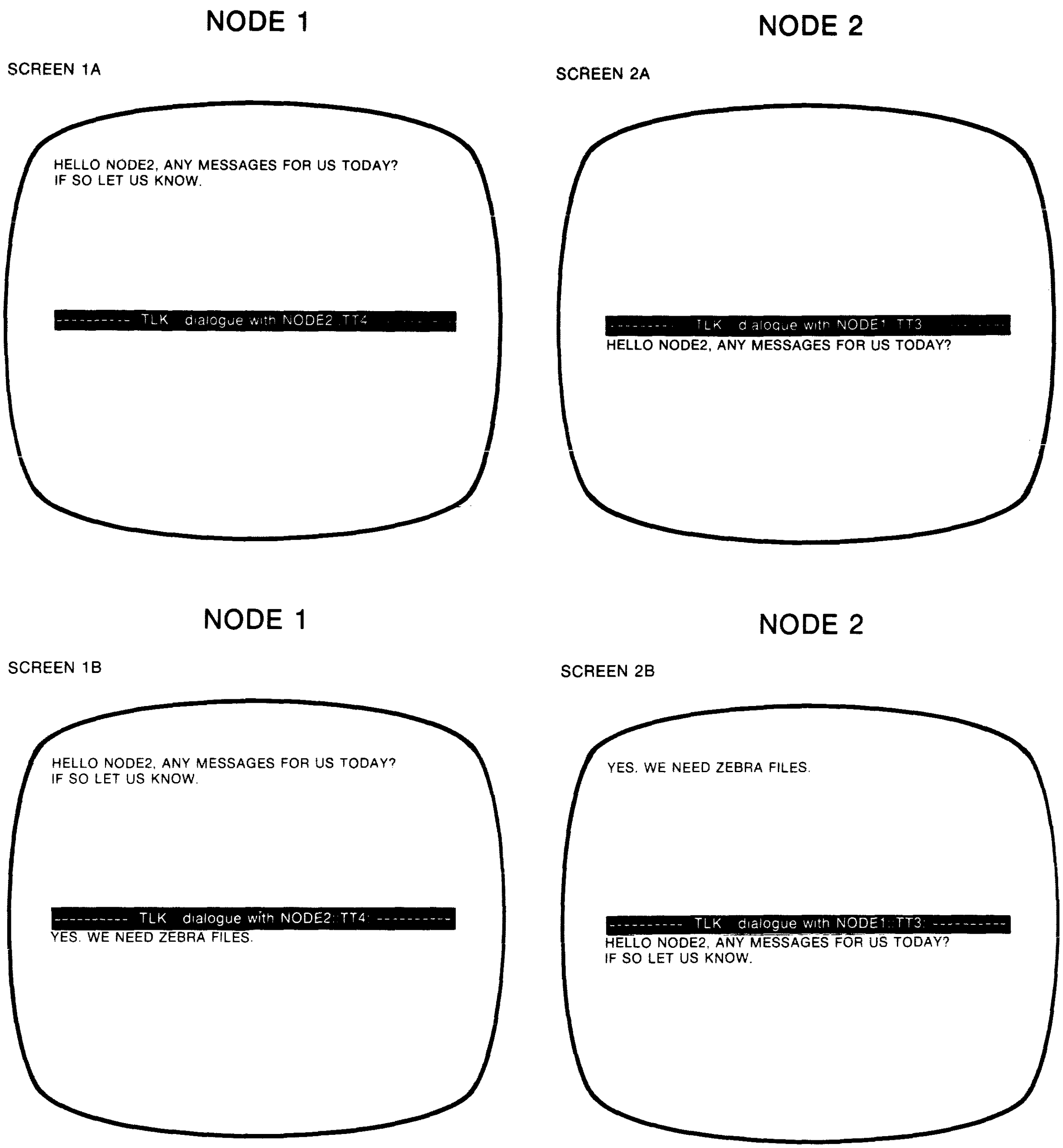
RSX/IAS DECnet example:

```
>EDI SHUTDOWN.CMD RET
[CREATING NEW FILE]
INPUT
NODEB::TT0: 'REMEMBER TO BACK UP ACNT FILES RET
NASHVC:: 'REMEMBER TO BACK UP ACNT FILES RET
MAYNRD::TT3: 'REMEMBER TO BACK UP ACNT FILES RET
'REMEMBER TO BACK UP ACNT FILES RET
@MORE.CMD RET
RET
*EXIT RET
[EXIT]
>TLK @SHUTDOWN.CMD RET
```

The last line shows one way to submit the command file to the TLK utility. The sign @, followed by the command file specification (@filespec) replaces a TLK command line. The command file itself contains the command line "@MORE.CMD". When TLK reaches this line, it proceeds to read and execute TLK commands contained within the file MORE.CMD. This second command file is *nested* within SHUTDOWN.CMD. Nested command files allow a user to submit more than one command file at a time. In the context of the example, the command lines in SHUTDOWN.CMD can be sent unchanged every day, but the manager also needs to send messages that depend on the events of a particular day. These variable messages can be stored in a separate command file, invoked from within SHUTDOWN.CMD.

7.1.4 The PHONE Command

The PHONE command invokes the VAX/VMS Telephone Utility, which allows a DECnet–VAX user to set up a dialog with another local or remote DECnet–VAX terminal user. (The command does not support dialogs with terminals at non-VAX nodes.) The utility closely simulates a real telephone service by providing a "hold button," conference calls, telephone directories, and other such facilities.



```
HELLO NODE2, ANY MESSAGES FOR US TODAY?
IF SO LET US KNOW.

---------- TLK  dialogue with NODE2::TT4: ----------
```

```
---------- TLK  dialogue with NODE1::TT3: ----------
HELLO NODE2, ANY MESSAGES FOR US TODAY?
```

```
HELLO NODE2, ANY MESSAGES FOR US TODAY?
IF SO LET US KNOW.

---------- TLK  dialogue with NODE2::TT4: ----------
YES. WE NEED ZEBRA FILES.
```

```
YES. WE NEED ZEBRA FILES.

---------- TLK  dialogue with NODE1::TT3: ----------
HELLO NODE2, ANY MESSAGES FOR US TODAY?
IF SO LET US KNOW.
```

Figure 7–2: DECnet–IAS TLK Split Screen Option

Example:

```
$ PHONE PEPPER::CLAUDIA
```

This example places a call to a user named CLAUDIA at node PEPPER. If she is currently logged on at that node, her terminal will display a message to indicate that someone is phoning her:

```
SALT::HALL is phoning you on PEPPER::
```

To answer the call from SALT::HALL, she types

```
$ PHONE ANSWER
```

This reply causes both terminals to display a split screen similar to the TLK utility's split screen option (Figure 7–2). The top half of the screen displays the local user's input and the bottom half displays the remote user's input. Both parties to the dialog can enter text at the same time.

For further information about the VAX/VMS Telephone Utility, see the *VAX–11 Command Language User's Guide.*

7.2 The VMSMAIL Utility (DECnet–VAX Only)

DECnet–VAX users can run the VMSMAIL Utility to send messages to users either at the local node or at remote DECnet–VAX nodes. This utility differs from TLK or PHONE in that it delivers a message whether or not the addressee is currently logged on. The VAX/VMS MAIL command invokes the utility, which allows a user to send mail, to read mail that has been received, and to perform a variety of other message-handling functions.

To send a message, a user enters the MAIL command and either specifies a file to be sent or types the message directly in response to a prompt. A command parameter indicates to whom the mail is to be delivered. For example:

```
$MAIL/SUBJECT=SICKPAY HEALTHBEN5.TXT LONDON::GRAHAM
```

This command line delivers the contents of the file HEALTHBEN5.TXT to a user named GRAHAM at node LONDON::. If GRAHAM is logged on at a terminal, a message like the following appears:

```
New mail from YORK::PEEK
```

To read his new mail, GRAHAM enters the MAIL command and then types carriage return (<CR>) in response to the MAIL prompt:

```
$ MAIL

You have 1 new message.

MAIL> <CR>

From: YORK::PEEK
TO:   LONDON::GRAHAM
Subj: Sickpay
.
.
.
```

The utility displays the details shown in the example, followed by the contents of the file that was delivered. Whenever a user issues the MAIL command, a message indicates whether any new mail has arrived since the last time he or she issued the command.

In the following example, the user types a message in response to a command prompt.

```
$ MAIL

MAIL> send
TO:   london::graham
SUBJ: sick pay
Enter your message below. Press CTRL/Z when complete, CTRL/C to quit:
You get two weeks of sick leave on full pay. <CTRL/C>

$
```

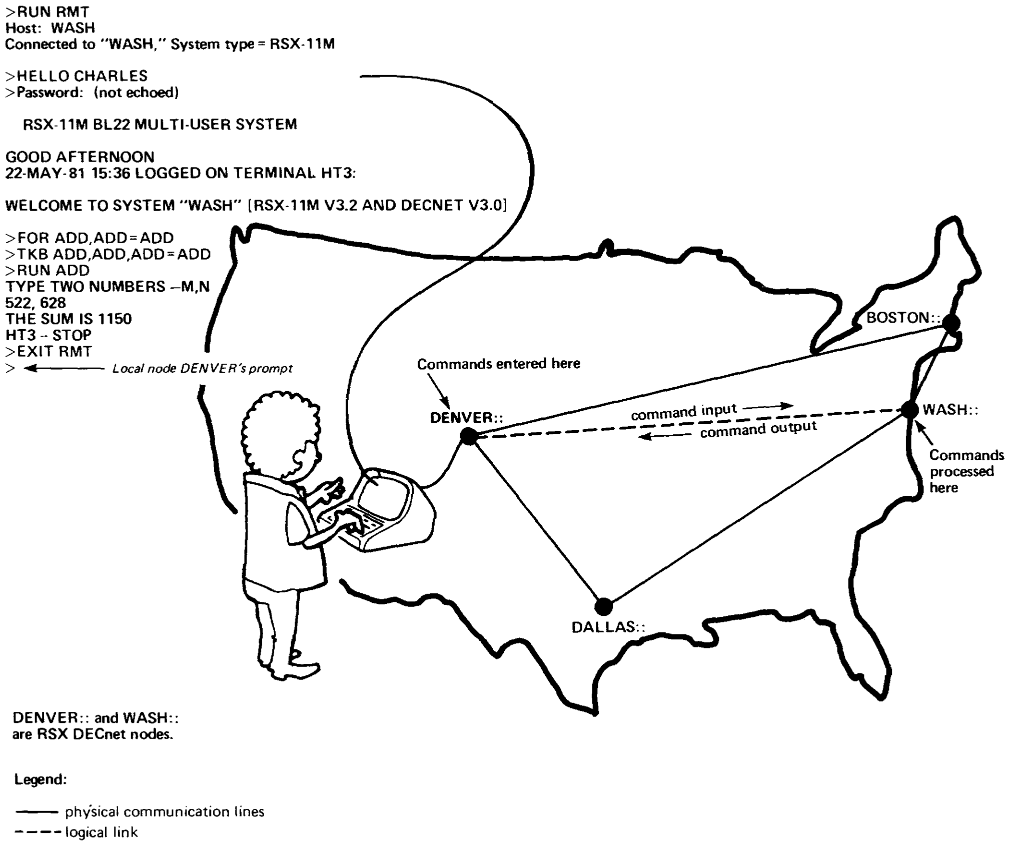
For further information about the VMSMAIL Utility see the *VAX-11 Utilities Reference Manual.*

7.3 Network Command Terminal Facilities

With the exception of DECnet-RT, all implementations have a utility that logically connects a local terminal to a remote node's operating system. These utilities, which are listed below, set up connections between nodes that run the same operating system. For example, the NET utility can connect a RSTS/E terminal to a remote DECnet/E node, but not to an RSX, VAX, or IAS node. (Intermediate routing nodes, however, can be any DECnet implementation.)

Implementation	Utility
RSX DECnet DECnet-IAS	Remote Command Terminal (RMT) utility
DECnet/E	Network Command Terminal (NET) utility
DECnet-VAX	SET HOST command
DECnet-20	SETHOST utility

By issuing the appropriate commands, a terminal user can temporarily become a *local* user of a specific remote node. The network command terminal utility or command in the source node sets up a logical link with a cooperating program in the remote node. The resulting connection allows the user to perform most functions that the remote node allows its local users to perform (see Figure 7–3).



```
>RUN RMT
Host: WASH
Connected to "WASH," System type = RSX-11M

>HELLO CHARLES
>Password: (not echoed)

  RSX-11M BL22 MULTI-USER SYSTEM

GOOD AFTERNOON
22-MAY-81 15:36 LOGGED ON TERMINAL HT3:

WELCOME TO SYSTEM "WASH" [RSX-11M V3.2 AND DECNET V3.0]

>FOR ADD,ADD=ADD
>TKB ADD,ADD,ADD=ADD
>RUN ADD
TYPE TWO NUMBERS –M,N
522, 628
THE SUM IS 1150
HT3 -- STOP
>EXIT RMT
> ←——— Local node DENVER's prompt
```

DENVER:: and WASH:: are RSX DECnet nodes.

Legend:

——— physical communication lines
– – – – logical link

Figure 7–3: Remote Terminal Processing

The ability to set up a logical connection between a terminal and a remote node has innumerable applications. Remote, interactive program development is possibly the most common and important application. If the local node does not have the resources necessary to support program development (a DECnet-11S node, for example), a programmer can obtain access to a remote node that has the required resources.

DECnet-11M, DECnet-11M-PLUS, and DECnet-11S nodes can all run RMT to initiate a remote terminal session, but a DECnet-11S node cannot be the target node. The target node is the node to which the user's terminal is logically connected; whereas the local node is the node to which the user's terminal is physically connected.

7.3.1 Setting Up a Command Terminal Session

To begin a command terminal session, a user invokes the appropriate utility or command and specifies a remote node. After a connection has been made, the user logs into the remote system just as if he or she were sitting at one of the remote node's terminals.

RSX DECnet RMT example:

```
>RMT RET
Host: BASIN
Connected to ""BASIN'', System type = RSX-11M
System ID: MAPPED RSX-11M V3.2 BL32

>HELLO FREAN
PASSWORD: (not echoed)

    RSX-11M BL32 MULTI-USER

GOOD MORNING
22-MAY-82 12:03 LOGGED ON TERMINAL HT3:

WELCOME TO SYSTEM BASIN [RSX-11M V4.0 AND DECNET V3.1]

>
```

DECnet/E NET example:

```
RUN $NET
NET V2.0-00 RSTS V7.1-00 DECnet System
NET>BOSTON
Connection established to node BOSTON
HELLO
RSTS V7.1-00 Timesharing Job 25 KB27 27-JUNE-82 11:46AM
USER:2,218
Password: (not echoed)

Ready
```

DECnet–VAX SET HOST example:

```
$SET HOST BOSTON
Username: FREAN
Password: (not echoed)

WELCOME TO VAX/VMS VERSION 2.0 ON NODE BOSTON

$
```

DECnet–20 SETHOST example:

```
@SETHOST BOSTON

[Type ^Y to return to node KL2102]
Welcome to node BOSTON,TOPS-20
```

The login procedure will not succeed unless the terminal user is authorized to use the remote node and he or she supplies the correct identification and password.

7.3.2 Issuing Commands to the Remote Node

After logging in, the user can issue commands that are actually executed at the remote node. Most commands normally accepted by the remote operating system are allowed. The RMT or NET utility or the SET HOST command in the source node forwards the commands to the remote node for processing; then a cooperating program in the remote node forwards command output back to the source node. The output is displayed at the issuing terminal as it would appear at a terminal directly connected to the remote system.

In the following example, an RSX DECnet user (CHARLES) connects to remote node THORIN, logs on, requests a listing of the files contained in his directory, and then terminates the session. How a user terminates the session and returns control to the local node depends on the utility or command being used.

```
>RUN RMT
Host:
Connected to "THORIN", System type = RSX-11M
System ID: MAPPED RSX-11M V3.2 BL32

>HELLO CHARLES
Password: (not echoed)

RSX-11M BL32 MULTI-USER SYSTEM

GOOD AFTERNOON
22-MAY-82 14:16 LOGGED ON TERMINAL HT3:

WELCOME TO SYSTEM THORIN [RSX-11M V4.0 AND DECNET V3.1]

>PIP/LI

DIRECTORY DB0:[305,360] 22-MAY-82 14:16

ADD.FTN;1       1.      22-MAY-82 10:06
ADD.MAP;1       4.      22-MAY-82 10:19
ADD.STB;1       3.      22-MAY-82 10:19
ADD.OBJ;1       2.      22-MAY-82 10:18
ADD.TSK;2       29.   C 22-MAY-82 10:38

TOTAL OF 39./49. BLOCKS IN 5. FILES

>EXIT RMT
>       (Local node's prompt)
```

Chapter 8
Network System Management

This chapter outlines the network-related tasks of a DECnet system manager and describes the facilities DECnet provides to perform those tasks. In so doing, the chapter also provides many definitions of network entities and parameters that pertain to network system management. In practice, many people may be responsible for carrying out the tasks that are described as follows. For the sake of simplicity, these people are collectively defined by the job title "system manager." Those who manage network nodes are responsible to both local and network users and should be aware that while local applications usually demand the greatest share of system resources, the remote users, a potentially very large group, must be sure of each node's response to network applications.

The procedures for performing Network Management functions can vary from system to system. For example, the procedure for generating a RSTS/E DECnet/E node is different from the procedure for generating an RSX DECnet node. RSTS/E is a timesharing system with a software structure quite unlike the real-time, event-driven RSX-11M. Because each DECnet implementation (DECnet/E, DECnet-VAX, and so on) is an extension of an operating system, the different ways to manage a node reflect the differences between the basic DIGITAL operating systems. The purpose of this chapter is to provide an overview of Network Management functions rather than to describe actual procedures for performing them. Procedural information can be found in the documentation provided for a specific implementation.

Network system management functions include the following:

- **Planning for node generation** (see Section 8.2). This process tailors DECnet software to suit a specific node's network application.

- **Generating network software** (see Section 8.3). This section discusses building the tailored DECnet software to create an active node.

- **Defining and redefining network parameters** (see Section 8.4). This introduces various network parameters whose definitions determine many aspects of a node's role within a specific DECnet configuration.

- **Operating a node** (see Section 8.5). This discusses operational functions such as starting up and shutting down a node and the physical lines connected to a node.
- **Monitoring node activity** (see Section 8.6). This discusses monitoring the day-to-day performance of a node by gathering and analyzing logging data that DECnet makes available.

Network system management responsibilities also include two other functions described in the next chapters. Chapter 9 describes down-line loading a satellite node and Chapter 10 describes procedures called loopback tests that can be performed to exercise various levels of DECnet software and hardware.

8.1 Network Management Utilities

Network Management utilities are the means by which a system manager performs most of the functions described in this chapter. All DECnet implementations support one or more utilities that provide access to DECnet management modules. In Phase III, these modules perform the functions defined by the Network Management layer (see Section 2.1). All implementations support a utility called the Network Control Program (NCP). Table 8–1 briefly describes the function of NCP and the other Network Management utilities that each DECnet implementation uses. Each utility accepts commands that activate DECnet management modules either to perform specific tasks or to request information about the current state of the local node or the network.

All of the utilities are described in detail in the system manager's or user's guide to each DECnet implementation.

8.2 Planning for Node Generation

Planning for node generation entails gathering and consolidating information. DIGITAL-supplied DECnet software provides generalized network capabilities, but the users must supply the data and programs that create a live DECnet application. A system manager accumulates these data and programs for eventual incorporation into the local node.

Each system manager must ensure that managers elsewhere in the network receive information that they need about other nodes. Programmers responsible for network applications should cooperate with system managers by exchanging information. For example, programmers must use correct addresses in calls that generate connect requests (see Section 5.2), while system managers must know the correct names and object types that the network programs use to identify themselves.

Table 8–1: DECnet Systems and Network Management Utilities

System	Utility	Function
RSX DECnet	NCP	Loads, controls, monitors, and tests DECnet software; down-line loads a DECnet/11S node.
	CFE	Changes parameters in the configuration file CETAB.MAC, which is produced at network generation.
	VNP	Changes the disk image of an RSX DECnet system. VNP cannot be run from a DECnet/11S node.
DECnet/E	NCP	Loads, controls, monitors, and tests DECnet software; maintains the DECnet/E parameter file; reports the current status of active logical links, of known physical lines and remote nodes, and of programs using local and remote send/receive services.
DECnet-IAS	NCP	Loads, controls, monitors, and tests DECnet software; defines configuration data base parameters.
	CFE	Changes parameters in the configuration file CETAB.MAC, which is produced at network generation.
DECnet-RT	NCP	Loads, controls, monitors, and tests DECnet software; defines and changes configuration data base (CETAB.MAC) parameters.
DECnet-VAX	NCP	Loads, controls, monitors, and tests DECnet software; defines configuration data base parameters; down-line loads a DECnet/11S node.
DECnet-20	NCP	Loads, controls, monitors, and tests DECnet software; defines the configuration data base.

Legend:
CFE - Configuration File Editor
NCP - Network Control Program
VNP - Virtual Network Processor

8.2.1 Configuration Data Bases

Every DECnet node has some form of configuration data base that defines characteristics of the local node and determines how that node functions within the network. In some cases, DIGITAL-supplied software already includes such a data base to provide initial default values for many data base entries. For other implementations, the network generation procedure creates the data base. Table 8–2 shows the term that each DECnet implementation uses to identify its configuration data base.

Depending on the type of DECnet node, the configuration data base may need to be updated periodically to reflect changes in the network or to tune the performance of the network. Section 8.4 explains many of the parameters typically included in a configuration data base. The facts and figures needed to define them must be gathered before a node can actively participate in a network.

Table 8–2: Configuration Data Base Terms

System	Term	Comments
RSX DECnet DECnet–IAS	Configuration File	This file (CETAB.MAC) is created during network generation and subsequently can be modified by the Configuration File Editor (CFE).
DECnet/E	Parameter File	This file ($NETPRM.SYS) is created and subsequently modified by NCP commands.
DECnet–RT	Configuration File	This file (CETAB.MAC) is created during network generation and subsequently can be modified by Network Management commands.
DECnet–VAX	Configuration Data Base	The initial data base is provided within the DECnet software supplied by DIGITAL. NCP commands are subsequently used to modify the data base.
DECnet–20	REV-CONFIG.-CMD	This file is the product of a network generation and is named by the user.

8.2.2 Network Generation Planning Aid

DECnet–IAS and DECnet–RT provide an aid to help users plan for node generation. The aid consists of a command file that contains questions pertaining to node generation options. A system manager runs this command file from a terminal and answers the questions according to the requirements of the local node. Using the system manager's responses, the command file then generates several worksheets. Each worksheet tells the system manager how to generate some part of DECnet to reflect local requirements.

See the appropriate network generation manual for a complete description of the command file and the worksheets it generates.

8.3 Generating Network Software

DECnet software arrives from DIGITAL on distribution media such as magnetic tapes or floppy disks. The type of media depends on the DECnet implementation and, in some cases, on the hardware configuration of a specific system. For example, RSX DECnet is distributed on one of several media, depending on the user's system.

The procedures for using the distributed software to generate an active node are different for each implementation of DECnet. To generate an RSX or an IAS node, a system manager has to regenerate the operating system first and then tailor and build the network application on top as a second procedure. Other implementations do not require a system manager to rebuild the distributed software. For example, to create a DECnet–VAX node, the system manager simply transfers the software from distribution media to system storage and uses NCP commands to define the configuration data base. And in the case of DECnet–RT, regenerating the RT–11 operating system and performing a network generation may both be unnecessary. The installation requirements depend on the current state of the operating system and on the planned DECnet application.

A specific list of DECnet software modules depends on the implementation and on the specific network application. However, the distributed software generally includes modules like the ones listed below, many of which have been discussed in previous chapters.

- Network device controllers
- A DDCMP module
- Routing and other transport layer modules (Phase III)
- An NSP module
- Network utilities like NCP, NFT, and TLK
- DAP-speaking modules for the FAL process and the NFAR routines

8.4 Defining Configuration and Other Static Parameters

The parameters that make up a node's configuration data base are relatively permanent or static because changing them tends to change the way the node functions within the network. In addition to parameters that are strictly part of a configuration data base, a system manager must define other parameters, such as local network object descriptions, that affect the way a node functions within a network. Depending on the implementation, configuration and other static parameters are defined in various ways. They can be defined at network generation, by means of NCP commands, or they may be predefined in DIGITAL-supplied software.

The following subsections provide brief descriptions of static parameters that are typically defined for most types of DECnet nodes.

8.4.1 Node Addresses and Names

Within a Phase III network, the system manager must assign a numeric address that uniquely identifies that node within the network. Phase II nodes are identified by unique alphanumeric names as well as unique addresses if they are part of a mixed Phase II and Phase III network. See Section 3.2.1 for a discussion of node addresses and node names.

8.4.2 Node Verification Passwords

Whenever one of its circuits or lines is turned on, a node exchanges initialization messages with the remote node at the other end (Section 8.5.2). The messages exchange information such as the version numbers of the node's DECnet software modules or the node's type — routing or nonrouting if the node runs Phase III DECnet — or the node's name — if the node runs Phase II DECnet. With one exception, Phase II and Phase III nodes can initialize with one another. The exception is Phase III DECnet–RT, which is always a nonrouting end node adjacent to another Phase III node.

After exchanging initialization messages, a node can request that the adjacent node verify its identity by supplying a password. If verification is required, adjacent nodes must supply passwords to gain access to the local node. If verification is not required, adjacent nodes do not have to supply passwords, and they automatically gain access to the local node after exchanging initialization messages. The access gained or denied at this stage is the ability to send and receive messages over the line or circuit between the two nodes.

Within a DECnet configuration that enforces node verification, each node maintains a data base of passwords that it sends to and expects to receive from its neighbors:

- **Receive password.** The password that the local node expects to receive from the adjacent node. If the password actually received does not match the receive password expected, the local node denies access to the adjacent node.

- **Transmit password.** The password that the local node sends to the adjacent node. The transmit password must match the receive password that the adjacent node expects from the local node.

8.4.3 Network Object Parameters

Most nodes maintain a data base that describes all the network objects, both user-written programs and DECnet modules, currently residing in the node and capable of engaging in network activity. (Network objects are described in Section 5.2.1.) The manner in which a node's object data base is maintained is dependent on each node's implementation. Typically, a node stores the following kinds of information:

- Object types, names, and addresses

- Access control and verification information associated with each object

- The number of copies of a specific object that DECnet can run to satisfy incoming connect requests

8.4.4 Transport Parameters (Phase III nodes only)

Several parameters affect the operation of the Transport module in Phase III full-routing nodes. (See Chapter 3 for a discussion of Phase III full routing.) These parameters determine:

- A maximum path cost that limits possible routes to paths that cost the same as or less than this value.

- Individual line or circuit costs that figure in routing algorithms used by the Transport module. Each line or circuit cost is a number from one to the maximum path cost set for the node. Higher cost can reduce traffic on the line or circuit because the Transport modules dispatch packets on the least costly paths. Lowering cost does not necessarily increase the traffic on a specific line or circuit. For example, if a line or circuit leads to an end node, the assigned cost does not affect the flow of data to that node.

- A maximum number of hops per path. A node is unreachable if it cannot be reached within the maximum number of hops.
- A routing timer that determines the interval between automatic updates of the local node's routing data base.
- A buffer size for the unit of data actually transmitted over physical lines by the transport module.
- A buffer count that determines the size of the transport module's pool of available buffers.

8.4.5 Line Identification

Each physical line leading from a node has a unique identification. These line indentifications, which are recorded in the configuration data base, have the following format:

dev-c-u

where

dev is a mnemonic for the type of device.

Examples:

DUP	—	the DUP11-DA
DMC	—	the DMC11-DA/AR, -MA,AL, or -FA/AR
DZ	—	the DZ11-A or -B
DMR	—	the DMR11

c is a number (0 or a positive integer) designating the device's hardware controller.

u is a unit/line number (0 or a positive integer) included if the device is a multiplexer.

Line identification examples:

Identification	Description
DMR-0	DMR11, controller 0
DZ-1-0	DZ11, controller 1, unit 0

8.4.6 Circuit Parameters

In addition to identifying actual physical lines, RSX DECnet, DECnet-VAX, and DECnet/E also define and identify circuits, which are logical, point-to-point communication paths. At nodes with these implementations, the circuit rather than the line is manipulated and defined to control the flow of data between nodes. Physical lines become the medium over which circuits operate. As a reflection of this concept, RSX DECnet, DECnet-VAX, and DECnet/E users usually specify circuits when users at other DECnet nodes specify lines in equivalent Network Management commands.

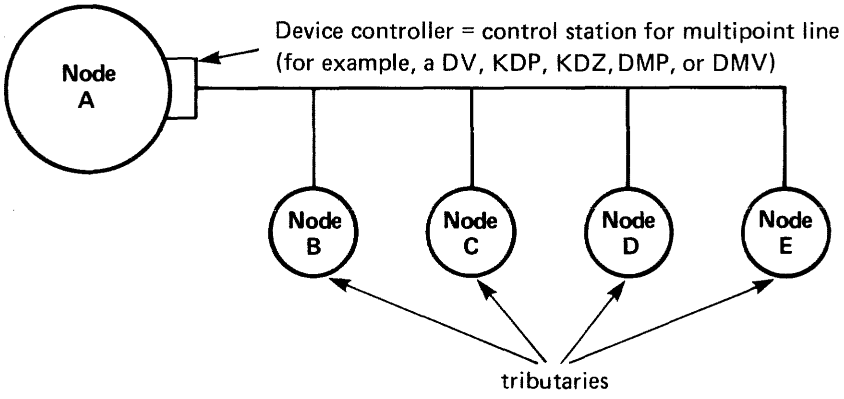
Circuits that handle DECnet traffic correspond closely to the physical lines that actually transmit the data. When a circuit corresponds to a point-to-point line, circuit and line identifications are exactly the same, For example, the string DMC-0 can identify either the line or the circuit in a Network Management command. However, circuit and line identifications associated with a multipoint line and its tributaries differ slightly. Section 8.4.7 explains how they differ. At an RSX DECnet/PSI node, DLM circuits to be mapped to PSI are not associated with specific physical lines. The identification of such a circuit starts with the mnemonic DLM.

For each circuit, the system manager must define various parameters, which differ depending on the type of circuit (for example, whether it is associated with a DDCMP point-to-point or multipoint line or with a PSI virtual circuit). See the appropriate implementation's system manager's guide for further information about circuit parameters.

8.4.7 Multipoint Line and Circuit Parameters

A multipoint line is a single communications line connected to more than two nodes. (A line connecting two nodes is called a point-to-point line.) The DECnet implementations supporting multipoint are the Phase III versions of RSX DECnet, DECnet-VAX, DECnet/E, and DECnet-RT.

Figure 8-1 is a diagram of a multipoint line, showing multipoint components.


Device controller = control station for multipoint line
(for example, a DV, KDP, KDZ, DMP, or DMV)
Node A
Node B
Node C
Node D
Node E
tributaries

Legend

DV = DV11–AA/BA synchronous line multiplexer

KDP = KMC11/DUP11–DA syncronous line multiplexer

KDZ = KMC11/DZ–11–A asynchronous line multiplexer

DMP = DMP11 synchronous link

DMV = DMV11 synchronous link

Figure 8–1: A Multipoint Line

The control station is the device controller responsible for overseeing data transmissions to and from all the nodes attached to the line. The devices attaching the other nodes to the line are called tributaries. An RSX DECnet, DECnet–VAX, or DECnet/E node can support either a control station or a tributary device. A DECnet–RT node, on the other hand, can support only a tributary device on a multipoint line.

From the perspective of the control station, the multipoint line and the tributaries connected to it constitute a single line, but the separate paths to each tributary represent individual circuits. A multipoint line therefore has more than one circuit associated with it. Figure 8–2 illustrates the relationship between a multipoint line and the circuits that correspond to its tributaries.

A tributary supports only one physical link to the control station. From the tributary's perspective, the DDCMP line that links it to the control station is point-to-point; the line and corresponding circuit are therefore equivalent.

In the context of full-routing implementations, whether a node supports a control station or a tributary on a multipoint line is not significant. The mechanisms for handling data transmissions on multipoint lines are transparent to the Transport layer modules (see Chapter 4).

At the control station's node, a system manager needs to define several parameters that affect the operation of the multipoint line and its corresponding circuits.

- **Tributary Addresses.** The data base at the control station's node must contain correct tributary addresses. The system manager must therefore record the unique decimal line address of each tributary on the line.

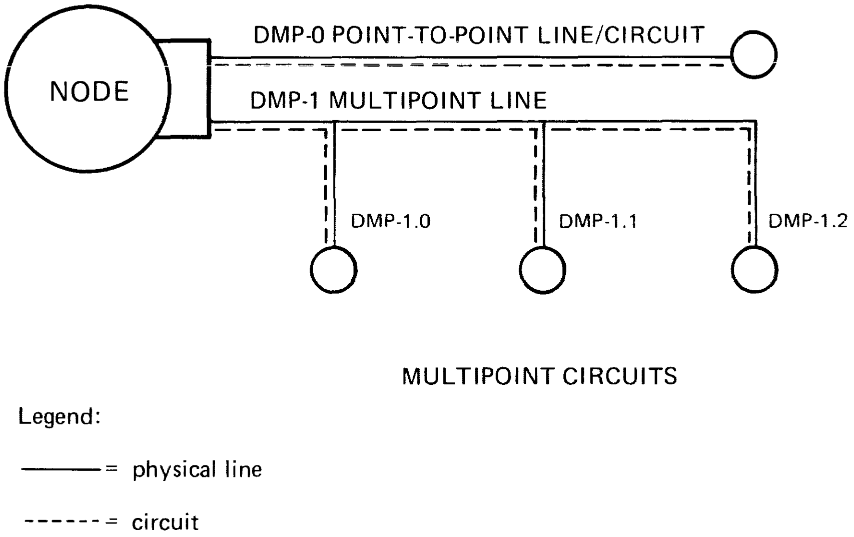
- **Polling Ratios.*** Whenever necessary, the control station delivers data addressed to tributaries under its control. In order to handle data originating from its tributaries, the control station periodically polls them; that is, periodically asks each tributary whether it has data waiting to be transmitted. When the control station polls a tributary that has such data, it allows the tributary to transmit.

  The frequency with which a tributary is polled depends on the frequency of its data transmissions. For the sake of efficiency, the control station polls active tributaries more often than inactive or dead tributaries. A dead tributary is one that has not responded within a predefined period of time.

* DECnet/E does not implement the polling technique discussed here.

A system manager can exercise control over the polling of specific tributaries. If for some reason a tributary should not be polled as often as others, the system manager can issue a command to assign an active polling ratio to that tributary. A command can also be issued to set a dead polling ratio that applies to all inactive tributaries.

A polling ratio is a number from 1 to 255. If a tributary's active polling ratio is 5, the control station passes through the active polling list five times before polling that particular tributary.



DMP stands for a DMP11 synchronous link.

Figure 8–2: Multipoint Circuits

8.4.8 Transmission Mode

The system manager sets the transmission mode for every line or circuit connected to the node. The transmission mode is either half duplex or full duplex:

Half Duplex. This means that the line or circuit can transmit data in either direction, but only in one direction at any given time. In other words, data cannot be sent and received simultaneously.

Full Duplex. This means that the line or circuit can transmit data in both directions simultaneously. Full duplex allows a node to send and receive data at the same time.

8.5 Operating a Node

To start up a node, a system manager or operator issues commands from a terminal to load and activate required DECnet software and to turn on communication lines and circuits. In response to the start-up procedure, the local DECnet software initializes with DECnet software in adjacent nodes (see Section 8.4.2). Shutting down the node reverses the procedure; commands are issued to halt network activity involving the local node, to shut off lines and circuits, and to unload DECnet software.

8.5.1 Controlling the State of a Node

For most implementations of DECnet, a system manager turns a node on and off by manipulating the node's state.

For example:

```
NCP>SET EXECUTOR STATE ON
```

This command activates the DECnet software at the RSX node currently defined to be the Executor. The Executor is the node at which the NCP command actually executes. The system manager, using an NCP command, determines whether the Executor is the local or a remote node.

Most DECnet implementations define three distinct node states: ON, SHUT, and OFF.

ON. The local node is enabled for performing network functions.

SHUT. The node maintains all existing logical links but does not permit any new links to be created. When existing links are disconnected, the node's state changes to OFF.

OFF. The local node cannot participate in any network activity, and existing logical links are aborted.

8.5.2 Controlling Line or Circuit States

A node cannot actively participate in a network until one or more communication lines or circuits (RSX DECnet, DECnet–VAX, and DECnet/E) have been turned on. By issuing an NCP command, a system manager sets the state of a line or a circuit to ON, OFF, or SERVICE:

ON. The line or circuit is available for use by the DECnet software responsible for routing data packets (the NSP module in Phase II nodes and the Transport module in Phase III nodes). When a line or circuit is turned on, the local node exchanges initialization messages and, optionally, node passwords with the remote node.

OFF. The line or circuit is not available for any kind of network activity.

SERVICE. The line or circuit is available for special network functions only: down-line loading, up-line dumping, or loopback testing. (Note that some implementations of DECnet do not recognize SERVICE as a state explicitly separate from ON; such implementations may impose the SERVICE state by their own internal means.)

When a line or circuit is in the ON state, DECnet software uses the data link protocol that ensures data integrity and sequentiality for normal network transmissions. (The standard DECnet protocol for normal traffic is DDCMP; see Section 2.3.) In the SERVICE state, a line or circuit transmits data embedded in a protocol provided for the the special network functions. (The standard protocol for these functions is called the Maintenance Operation Protocol, abbreviated to MOP; see Section 2.3.)

8.6 Monitoring Node Activity

At each node, DECnet provides access to the node's network information. A system manager can use Network Management utilities to display the following information at a terminal:

- The current state of local and remote nodes and of physical lines or circuits
- Values currently defined for configuration data base and other static parameters
- The contents of various counters that DECnet software maintains to track network performance

In addition to displaying information on request, DECnet can automatically log certain events both at the operator's console and in a file. Event logging records operational events such as a line starting up or shutting down. (DECnet–RT does not support event logging.)

DECnet uses counters to track other types of information. A system manager can periodically record these counters in a file or display them at a terminal to obtain detailed statistics on the node's network activity. Node counters maintain statistics on logical link operations: for example, how many connect requests have been sent and how many received; how many messages have been sent over logical links and how many received. If the node is a Phase III implementation, counters record Transport layer activity as well: for example, how many errors of different kinds have been found in packet headers; how many line or circuit initialization and verification failures have occurred.

DECnet maintains individual communication line or circuit counters. These counters record statistics like the number of data blocks sent and received successfully; the number of blocks received with errors; the number of times a tributary has passed from active to dead state.

After counters have been displayed or recorded in a file, NCP commands can be issued to set the counters to zero. In this way, a system manager can manipulate the time span that the counters monitor. For example, a system manager could set all node counters to zero as programmers begin to test a network application. At the end of the test, the counters could be examined to see how the application affected the node's performance.

Chapter 9
Down-line Loading and Up-line Dumping

RSX DECnet, DECnet–IAS, DECnet–20, and DECnet–VAX all support down-line loading, which means loading a memory or system image from a file at one node to a separate target node. The target node is usually an RSX–11S DECnet node, a memory-only system with no disk-based file storage of its own.*

At an RSX–11M or RSX–11M–PLUS node, the system manager generates the RSX–11S DECnet system image. Once generated, the image can be modified by VNP commands at an RSX node. The load itself can be initiated in one of two ways. An operator can issue NCP commands to load the image down-line to the target node, or an operator at the target can initiate the load by triggering a bootstrap ROM. (Section 9.3 explains these procedures.)

Up-line dumping is a function that complements down-line loading. The DECnet/11S target node copies the contents of its memory to a remote node in response to a system crash. To be capable of dumping its own image up-line, a target RSX–11S node must be generated to include a routine called NETPAN.

9.1 Down-line Loading Definitions

The down-line loading function is distributed among two or more nodes in a DECnet configuration. The following definitions clarify the roles played by the various nodes.

- **The command node** is the node from which the NCP load commands are issued.
- **The executor node** actually executes the NCP commands; it must be adjacent to the target node.
- **The target node** receives the system image loaded down the line or dumps a system image up the line.

A single node can act as both the command and the executor node.

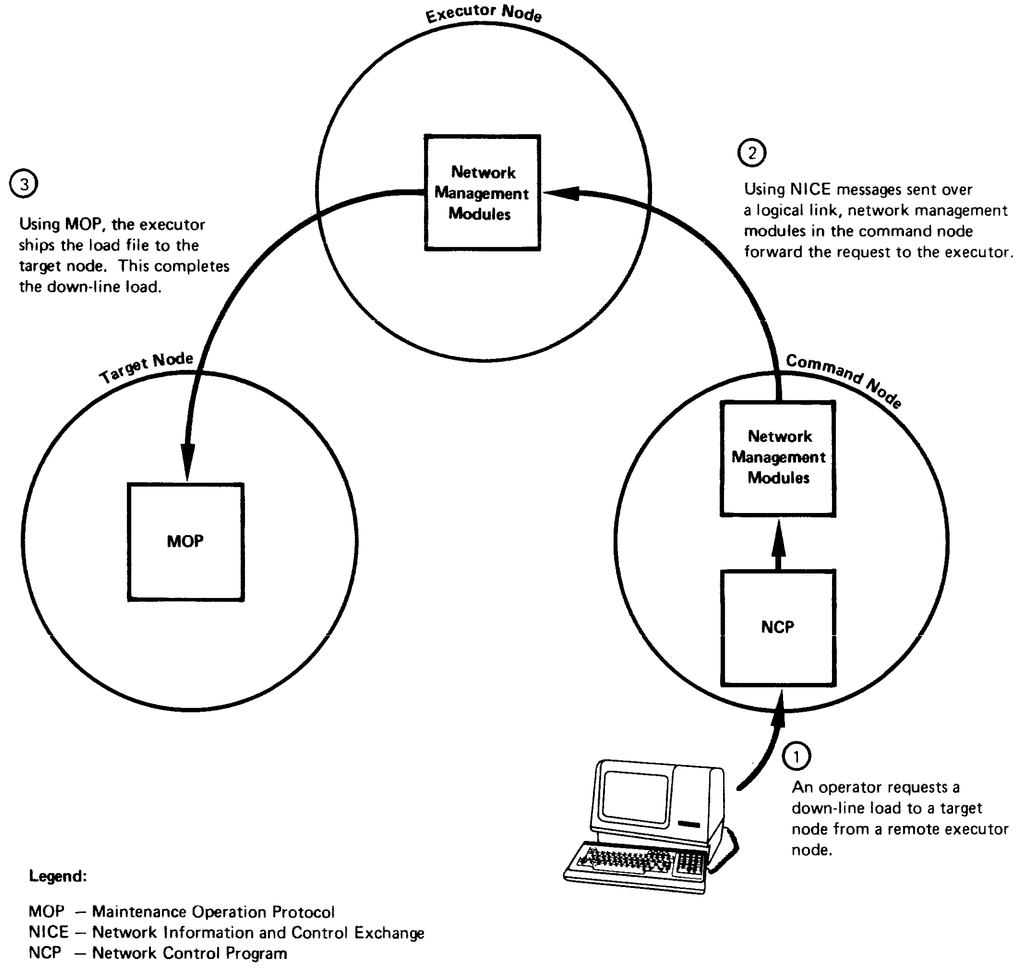
* A DECnet-20 node can load a system image down-line both to the DN20 front end and to a specially adapted 11S node called a DN200, which supports a card reader and a line printer, and which serves as a remote batch station.

9.2 Down-line Loading Data Base Parameters

For every target node to be down-line loaded, the Executor has access to a permanent data base. Each data base contains default parameters for down-line loading a specific target node. The system manager can override these defaults by providing parameter values in an NCP LOAD command. The parameters are defined initially at network generation and can be redefined when necessary.

9.3 Performing a Down-line Load

Whether a remote command node or the target itself initiates the load, the target must have local access to a cooperating program called a primary loader. This loader is usually contained in a bootstrap ROM (Read Only Memory) incorporated in the target. During the down-line load procedure, a series of programs may be loaded on top of the primary loader; each program calls the next until the system image itself is loaded down-line.



③ Using MOP, the executor ships the load file to the target node. This completes the down-line load.

② Using NICE messages sent over a logical link, network management modules in the command node forward the request to the executor.

① An operator requests a down-line load to a target node from a remote executor node.

Legend:

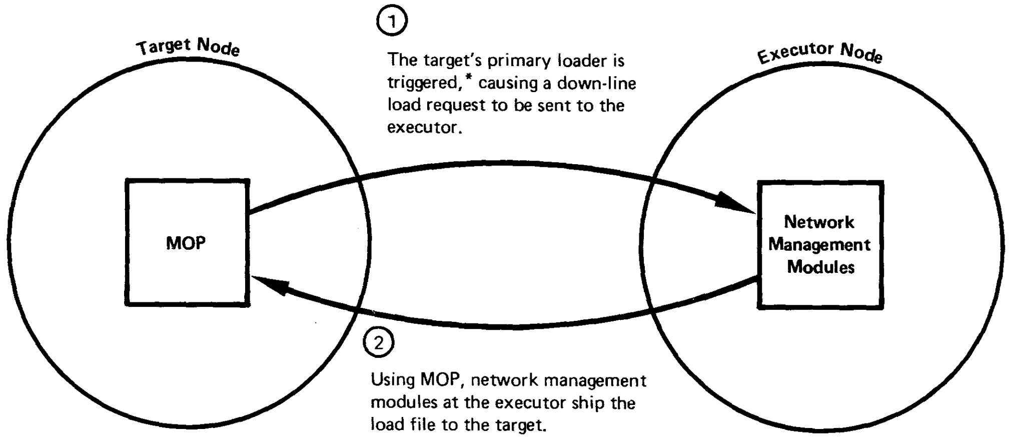
MOP – Maintenance Operation Protocol
NICE – Network Information and Control Exchange
NCP – Network Control Program

Figure 9–1: A Down-line Load Initiated by a Command Node

The line or circuit between the Executor and the target is in SERVICE state during the procedure (see Section 8.5.2). Either the system manager explicitly sets the state to SERVICE or the DECnet software sets the state automatically. How the state is set depends on the implementation and on the way a load is initiated.

9.3.1 The LOAD Command

The NCP LOAD command is the means of initiating a down-line load from a remote command node. As soon as the LOAD command has been issued, an operator at the target must manually trigger the bootstrap ROM, unless the line's device controller is a DMC11 or DMR11 device. These devices can trigger the target's primary loader automatically if the LOAD command passes down the correct password (see Section 9.2). Figure 9–1 illustrates a down-line load initiated by a command node.



1 The target's primary loader is triggered,* causing a down-line load request to be sent to the executor.

2 Using MOP, network management modules at the executor ship the load file to the target.

*An operator triggers the loader manually or the completion of an up-line dump triggers it automatically.

Legend:

MOP — Maintenance Operation Protocol

Figure 9–2: A Down-line Load Initiated by a Target Node

9.3.2 Target-initiated Down-line Loads

An operator at the target node can request a load by manually triggering the primary loader. In addition, the loader is triggered automatically at the completion of an up-line dump from the target. Target-initiated down-line loads always use the parameter values defined in the permanent data base for the target. Figure 9–2 illustrates a target-initiated down-line load.

9.4 Up-line Dumping

An RSX-11S target must include a routine called NETPAN in order to dump its image up-line to the Executor. If the target node crashes, control automatically passes to NETPAN, which then initiates the up-line dump. DECnet software responds by setting the line or circuit to SERVICE state and copying the target's image to a dump file, which is specified in the target's permanent data base.

When the dump completes, the NETPAN routine automatically triggers the target's primary loader. This action causes the executor to reload the target (see Section 9.3.2), which can then continue operating normally.

9.5 Down-line Loading and Checkpointing RSX–11S Tasks

DECnet–11M, DECnet–11M–PLUS, and DECnet–VAX support two capabilities relating to a DECnet–11S node. The first is called down-line task loading. RSX–11S tasks can be stored at a DECnet–11M, DECnet–11M–PLUS, or DECnet–VAX node and loaded down to the RSX–11S node. The second is called checkpointing, which is a standard RSX–11M capability. An executing RSX–11S task can be interrupted, then copied in its interrupted state up the line, and be replaced by a higher priority task loaded down-line from the Executor. When the higher priority task has completed, the interrupted task is reloaded down-line and allowed to continue executing.

At a DECnet–11M or DECnet–11M–PLUS node, the operating system regularly checkpoints tasks to local disk storage. However, RSX–11S nodes are basically memory-only systems, so the only way to checkpoint tasks is to use the Executor's disk storage.

These two capabilities give flexibility to an RSX–11S node that would not be possible without DECnet. To change the set of resident tasks at a stand-alone RSX–11S system, an operator would have to reboot with a different system image. See the *RSX DECnet System Manager's Guide* and the *DECnet–VAX System Manager's Guide* for further information.

Chapter 10
Loopback Testing

Loopback tests are procedures that exercise network software and hardware by repeatedly sending data through a number of network components and then returning the data to its source. If a test succeeds, the data loops back to its source without being corrupted. If a test fails, the data does not return to its source or it returns in a corrupted state. A system manager can run variations of the loopback tests to isolate the network component responsible for losing or corrupting the data. DIGITAL software services personnel routinely run loopback tests after installing DECnet software at a node. Successful tests verify that both the software modules and hardware equipment within a node are operating correctly.

This chapter describes loopback tests initiated by NCP commands as well as tests initiated by user programs. As part of the Network Management function, DECnet implementations provide the software mechanisms required to loop data through various network components. Figures shown below illustrate the functional layers actually exercised by specific tests. Some of the tests require the system manager to set up a hardware mechanism that physically loops the test data back from a device controller, from a modem, or from some point on a physical line.

Users can also write their own test programs that use standard DECnet capabilities. For example, one user program can send data over a logical link to another user program, which can then return the data to the first program. Finally, the first program can verify that the data it receives matches the data it sent. If the two programs reside in different nodes, the test exercises a variety of DECnet functions at both nodes: the logical link mechanism, the Transport functions, the Data Link functions (DDCMP), and the communications hardware between the nodes. If both programs reside in the same node, they test the logical link mechanism and the Transport functions in that node.

Basically, there are two categories of loopback tests: node level tests and line level tests.

- **Node level tests.** Node level tests all use logical links to loop test data through a specified loopback node. The loopback node can be the local node, a remote node, or a loopback node name that has been associated with a specific physical line or circuit. Variations of the node level tests allow a system manager to exercise all the layers of network function in a local as well as in a remote node.

  Most node level tests can execute simultaneously with normal node and line activity.

- **Line or circuit level tests.** Line or circuit level tests directly exercise the operation of communications hardware. These tests do not use logical links to circulate the test data. Instead, an NCP LOOP LINE or LOOP CIRCUIT command causes the test data to be delivered directly to a Data Link layer module (MOP), which transmits the data. The data may be looped back by a hardware device inserted somewhere on the line or it may be looped back by DECnet management software in a remote node.

  While this kind of test is running, the line or circuit being tested cannot be used for any other activity.

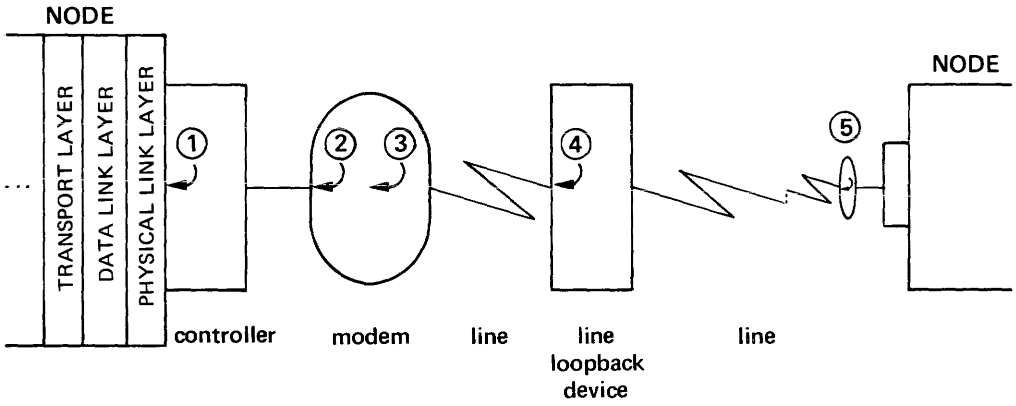
All Phase III implementations, except for DECnet/E, support a common set of node level and line or circuit level loopback tests, which can be initiated by the same set of NCP commands. DECnet/E supports node level tests only. DECnet-20, which is a Phase II implementation, supports a set of node level tests only, which differ slightly in detail from Phase III loopback tests.

10.1 Hardware Loopback Devices

Depending on the type of test to be run, a system manager may need to prepare a hardware loopback device before running the test. Various hardware loopback devices can be used to test specific parts of the communications hardware. These devices physically turn test data around at one of several points:

- Within the device controller
- Within the modem
- At some point on the physical line

To use a device controller as a loopback device, it must be set to loopback mode. If the controller is a DMC11 or a DMR11 at a DECnet-VAX or a DECnet/E node, a system manager can issue an NCP command to enable loopback mode. In all other cases, loopback mode must be set manually. To loop data through the controller as far as the modem, the system manager manually sets the modem to loopback. Usually, the modem itself has two possible loopback points: one at the interface with the controller and one at the interface with the physical line. To test stretches of the physical line, a hardware loopback device must actually be inserted at some point along the line. The type of device required depends on the type of line to be tested. Figure 10-1 illustrates the possible loopback points within the communications hardware.



① Loopback at controller

② Loopback at modem on controller side

③ Loopback at modem on line side

④ Loopback at hardware loopback device inserted in line

⑤ Loopback at a remote modem

Figure 10–1: Hardware Loopback Devices

10.2 Node Level Loopback Tests

Node level loopback tests can be initiated either by an NCP LOOP NODE command or by a user-written test program. In either case, the test may require a few preparatory steps, including setting up a hardware device. Whether such steps are necessary depends on the test to be run. Section 10.2.1 summarizes the NCP commands that pertain to node level testing. These commands are then illustrated in Sections 10.2.2 and 10.2.3, which discuss command-initiated and program-initiated loopback tests respectively.

10.2.1 Node Level Loopback Commands

The following commands are used to prepare for and to run node level loopback tests.

- **SET NODE *name* LINE *line-id* or SET NODE *name* CIRCUIT *cir-id*.** This command associates a special loopback node name with the line or circuit specified. This special node name can then be used in a LOOP NODE command or in a test program's request to form a logical link. When DECnet recognizes the special node name, it transmits forthcoming test data over the line associated with that name. Depending on the type of test, the data is looped back by hardware somewhere on the line, or the Transport software at a remote node uses its routing algorithm to determine the path on which the test data will be looped back.
- **LOOP NODE *name*.** This command requests DECnet to perform a node level loopback test; the node specified is the node to be connected to and which will loop back the data. The node named can be the local node, a special loopback node, or a remote node.

The command accepts further input parameters that determine the number of times the test data is to be looped, the contents of the test data, and the length of the test data in bytes.

The next two sections illustrate how these commands are used to perform variations of the node level loopback tests.

10.2.2 Using Commands to Initiate Tests

Figure 10–2 shows four different node level tests that can be performed by issuing an NCP LOOP NODE command. The figure includes any required setup commands. By diagramming the layers through which the test data travels, the figure identifies the DECnet software and/or hardware components exercised by each test.

10.2.3 Using Programs to Initiate Tests

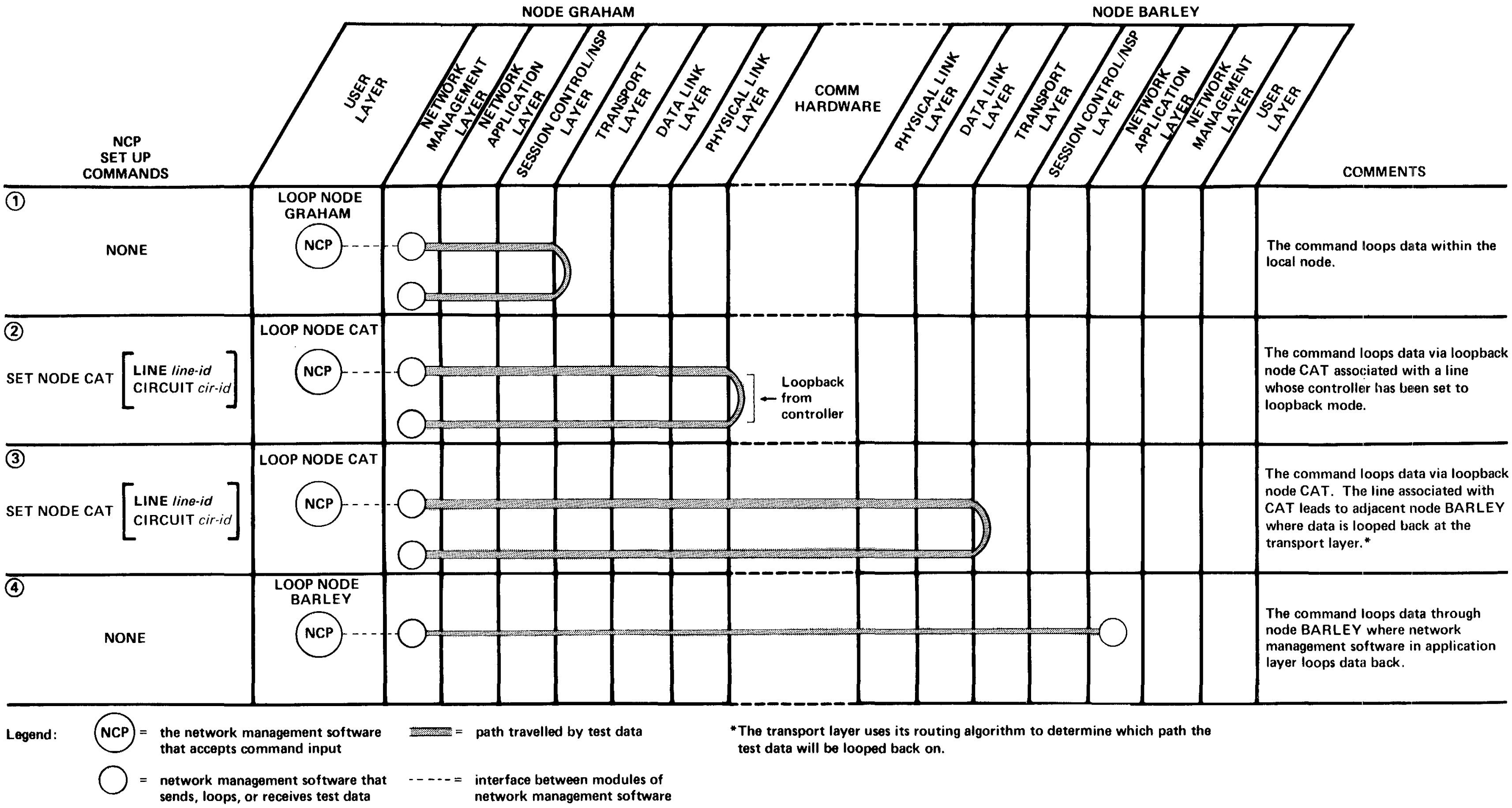
Figure 10–3 illustrates node level loopback tests that are initiated by user programs. As the figure shows, some of these tests require someone to issue one or two preparatory commands to set up loopback conditions. Hardware loopback devices may also need to be prepared. Because programmers can devise their own loopback test variations, the tests diagrammed here are merely representative.

10.3 Line/Circuit Level Loopback Tests

Line or circuit level loopback tests are provided to test communications hardware rather than DECnet software components. Therefore, in response to a line or circuit level test command (LOOP LINE *line-id* or LOOP CIRCUIT *cir-id*), the DECnet management software delivers the test data directly to a Physical Link layer module called MOP (Maintenance Operation Protocol). The MOP module, which resides in the same functional layer as DDCMP, operates when a line is in SERVICE state (see Section 9.5.2). MOP transmits the test data, which then loops back at one of several points within the communications hardware.

10.3.1 Line/Circuit Level Loopback Commands

Line or circuit level tests are normally initiated by NCP commands rather than by user programs. In addition to the command that actually requests a test, one or more preparatory commands may be required. Furthermore, to exercise specific parts of the communications hardware, a system manager may need to enable or put into place a hardware loopback device before running a test.



	NCP SET UP COMMANDS	USER LAYER	COMMENTS
①	NONE	LOOP NODE GRAHAM	The command loops data within the local node.
②	SET NODE CAT [LINE *line-id* / CIRCUIT *cir-id*]	LOOP NODE CAT	The command loops data via loopback node CAT associated with a line whose controller has been set to loopback mode.
③	SET NODE CAT [LINE *line-id* / CIRCUIT *cir-id*]	LOOP NODE CAT	The command loops data via loopback node CAT. The line associated with CAT leads to adjacent node BARLEY where data is looped back at the transport layer.*
④	NONE	LOOP NODE BARLEY	The command loops data through node BARLEY where network management software in application layer loops data back.

Legend:

(NCP) = the network management software that accepts command input

○ = network management software that sends, loops, or receives test data

▨ = path travelled by test data

- - - - = interface between modules of network management software

*The transport layer uses its routing algorithm to determine which path the test data will be looped back on.

Figure 10–2: Command-initiated Loopback Tests

The NCP commands pertaining to line or circuit level loopback tests are as follows:

- **SET LINE *line-id* or SET CIRCUIT *cir-id* STATE SERVICE.** This command enables the service functions provided by MOP for the specified line or circuit. Depending on the implementation, the system manager may or may not have to set the state to SERVICE explicitly. However, the line or circuit must at least be turned ON.

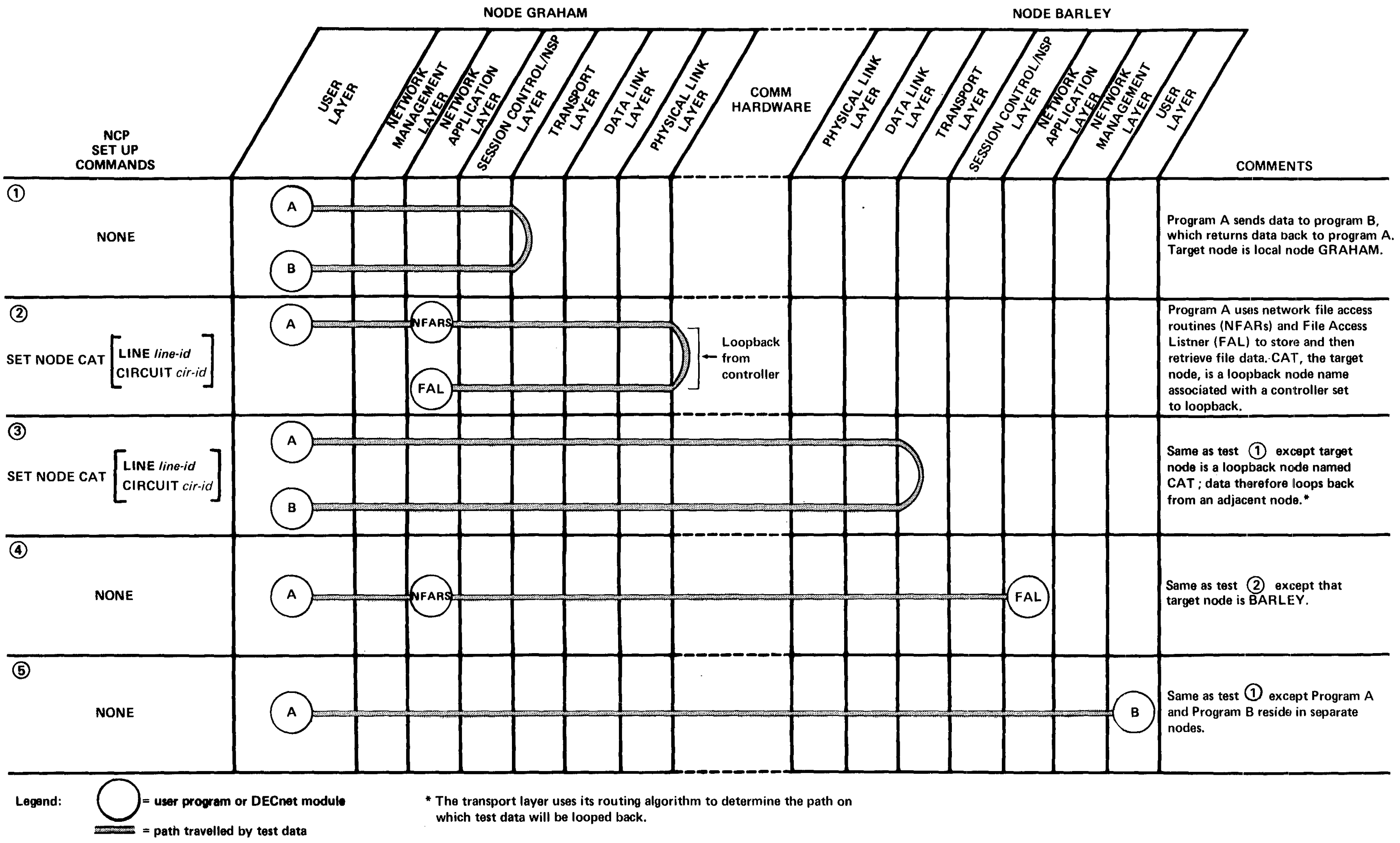
- **LOOP LINE *line-id* or LOOP CIRCUIT *cir-id*.** This command requests DECnet management software to perform a line or circuit level loopback test on the specified line. Where the test data loops back depends on the position of a hardware loopback device, if any. If the test data does not encounter a hardware loopback device, the data is looped back by DECnet management software in the remote node at the other end of the line.

  Like the LOOP NODE command, the LOOP LINE or LOOP CIRCUIT command accepts further input parameters that determine the number of times the test data is to be looped, the contents of the test data, and the length of the test data in bytes.

The following section diagrams some tests requested by NCP LOOP LINE or LOOP CIRCUIT commands.

10.3.2 Examples of Line/Circuit Level Tests

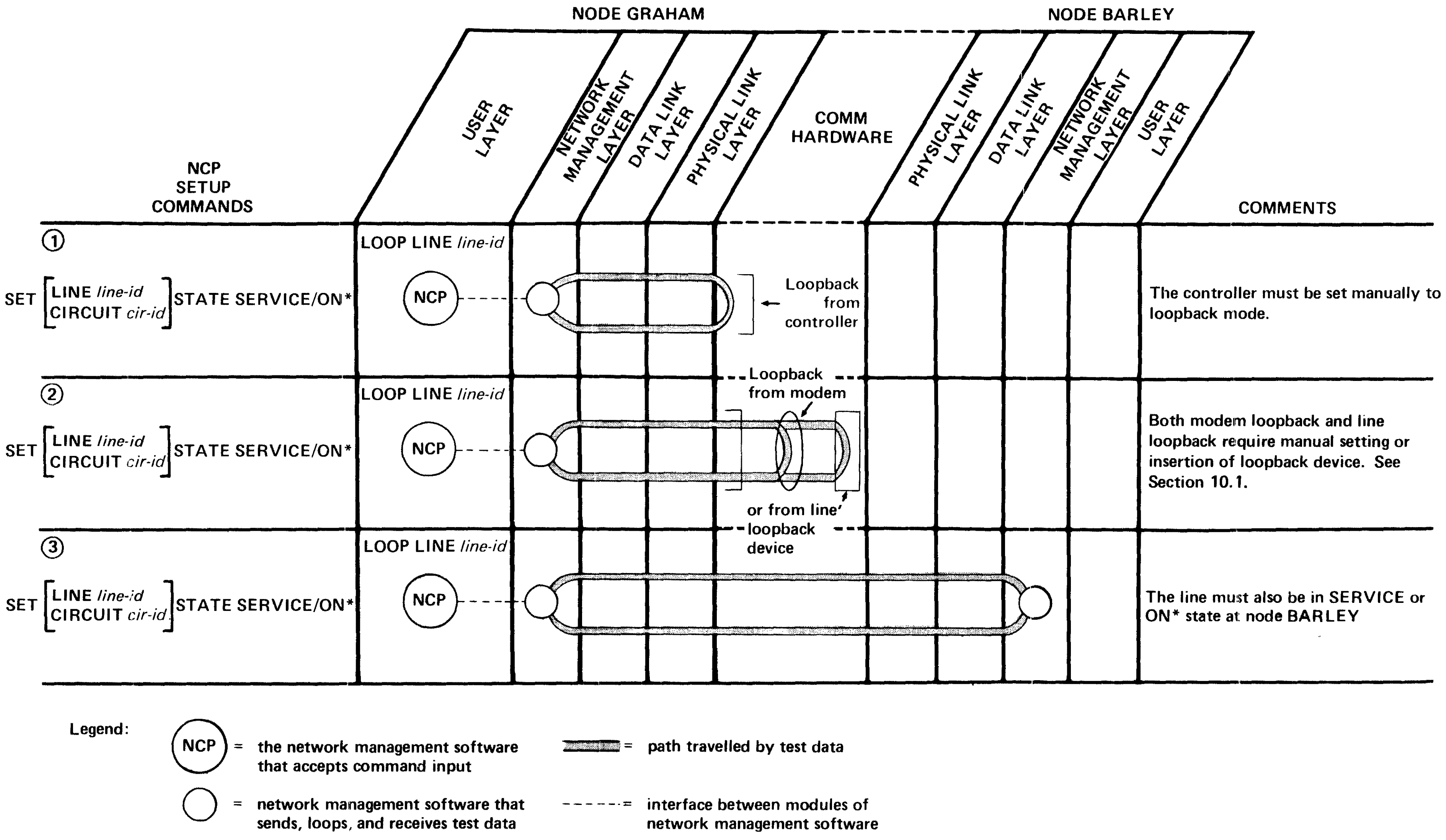
Figure 10–4 contains diagrams of three line or circuit level loopback tests. The figure shows the commands issued to run the tests and where the data travels before looping back.



	NCP SET UP COMMANDS	COMMENTS
①	NONE	Program A sends data to program B, which returns data back to program A. Target node is local node GRAHAM.
②	SET NODE CAT [LINE *line-id* CIRCUIT *cir-id*]	Program A uses network file access routines (NFARs) and File Access Listner (FAL) to store and then retrieve file data. CAT, the target node, is a loopback node name associated with a controller set to loopback.
③	SET NODE CAT [LINE *line-id* CIRCUIT *cir-id*]	Same as test ① except target node is a loopback node named CAT ; data therefore loops back from an adjacent node.*
④	NONE	Same as test ② except that target node is BARLEY.
⑤	NONE	Same as test ① except Program A and Program B reside in separate nodes.

* The transport layer uses its routing algorithm to determine the path on which test data will be looped back.

Figure 10–3: Program-initiated Loopback Tests


NODE GRAHAM
NODE BARLEY
NCP SETUP COMMANDS
USER LAYER
NETWORK MANAGEMENT LAYER
DATA LINK LAYER
PHYSICAL LINK LAYER
COMM HARDWARE
PHYSICAL LINK LAYER
DATA LINK LAYER
NETWORK MANAGEMENT LAYER
USER LAYER
COMMENTS
①
SET [LINE line-id / CIRCUIT cir-id] STATE SERVICE/ON*
LOOP LINE line-id
NCP
Loopback from controller
The controller must be set manually to loopback mode.
②
SET [LINE line-id / CIRCUIT cir-id] STATE SERVICE/ON*
LOOP LINE line-id
NCP
Loopback from modem
or from line' loopback device
Both modem loopback and line loopback require manual setting or insertion of loopback device. See Section 10.1.
③
SET [LINE line-id / CIRCUIT cir-id] STATE SERVICE/ON*
LOOP LINE line-id
NCP
The line must also be in SERVICE or ON* state at node BARLEY
Legend:
NCP = the network management software that accepts command input
= network management software that sends, loops, and receives test data
= path travelled by test data
------= interface between modules of network management software

*SERVICE or ON — depends on the implementation

Figure 10-4: Line/Circuit Level Loopback Tests

Appendix A DECnet Documentation

	Order Number
RSX DECnet Documentation	
Introduction to DECnet	AA-J055C-TK
Overview of RSX DECnet	AA-M096A-TC
RSX DECnet Guide to User Utilities	AA-H223B-TC
RSX DECnet System Manager's Guide	AA-H224B-TC
RSX DECnet Network Generation and Installation Guide	AA-H225B-TC
RSX DECnet Post-Installation Checkout Procedures	AA-J319B-TK
RSX DECnet Programmer's Reference Manual	AA-M098A-TC
RSX DECnet Reference Card	AA-M097A-TC
RSX DECnet Release Notes	AA-J517B-TC
DECnet–IAS Documentation	
Introduction to DECnet	AA-J055B-TK
DECnet-IAS Programmer's Reference Manual	AA-L718A-TC
DECnet-IAS Guide to User Utilities	AA-L717A-TC
DECnet-IAS System Manager's Guide	AA-L714A-TC
DECnet-IAS Network Generation and Installation Guide	AA-L715A-TC
DECnet-IAS Post-Installation Checkout Procedures	AA-L716A-TC
DECnet–VAX Documentation	
DECnet-VAX System Manager's Guide	AA-H803B-TE
DECnet-VAX User's Guide	AA-H802B-TE
DECnet-VAX Cross-System Notes	AA-M544A-TE
DECnet-VAX Software Installation Guide	AA-M537A-TE

DECnet/E Documentation

Introduction to DECnet	AA-J055B-TK
DECnet/E Network Programming in BASIC-PLUS and BASIC-PLUS-2	AA-H501B-TC
DECnet/E Network Programming in MACRO-11	AA-H265A-TC
DECnet/E Network Programming in FORTRAN	AA-L266A-TC
DECnet/E Network Programming in COBOL	AA-H503B-TC
DECnet/E System Manager's Guide	AA-H505B-TC
DECnet/E Guide to User Utilities	AA-H504B-TC
DECnet/E Network Installation Procedures	AA-K714A-TC

DECnet-RT Documentation

Introduction to DECnet	AA-J055B-TK
DECnet-RT Guide to User Utilities	AA-K215A-TC
DECnet-RT Programmer's Reference Manual	AA-L268A-TC
DECnet-RT System Manager's Guide	AA-K250A-TC
DECnet-RT Network Generation and Installation Guide	AA-K252A-TC
DECnet-RT Unsupported Software	AA-L527A-TC
DECnet-RT Release Notes	AA-K254A-TC

DECnet-20 Documentation

Introduction to DECnet	AA-J055A-TK
TOPS-20 DECnet-20 Programmer's Guide and Operations Manual	AA-5091B-TM

Phase III DNA Documentation

The DIGITAL Network Architecture General Description	AA-K179A-TK
The Digital Data Communications Message Protocol (DDCMP) Functional Specification Version 4.1.0	AA-K175A-TK
The Network Services Protocol Functional Specification (NSP), Version 3.2.0	AA-K176A-TK
The Session Control Functional Specification, Version 1.0.0	AA-K182A-TK
The Data Access Protocol (DAP) Functional Specification, Version 5.6.0	AA-K177A-TK
The Maintenance Operations Protocol (MOP) Functional Specification, Version 2.0.0	AA-K178A-TK
The Transport Functional Specification, Version 1.3.0	AA-K180A-TK
The Network Management Functional Specification, Version 2.0.0	AA-K181A-TK

Index

Introduction to DECnet
AA-J055C-TK

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent.

- □ Assembly language programmer
- □ Higher-level language programmer
- □ Occasional programmer (experienced)
- □ User with little programming experience
- □ Student programmer
- □ Other (please specify) ______

Name ______ Date ______

Organization ______

Street ______

City ______ State ______ Zip Code ______
or
Country

—Do Not Tear - Fold Here and Tape —

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE DOCUMENTATION
1925 ANDOVER STREET TW/E07
TEWKSBURY, MASSACHUSETTS 01876

Do Not Tear - Fold Here and Tape

Cut Along Dotted Line