EY-8324E-SG-0001

DECnet-VAX Internals

Student Workbook

Prepared by Educational Services
of
Digital Equipment Corporation

EY-8324E-SG-0001

First Edition, July 1987

Printed in U.S.A.

The manuscript for this book was created using DIGITAL Standard Runoff. Book production was done by Educational Services Development and Publishing in Nashua, NH.

CONTENTS

SG STUDENT GUIDE

1 REVIEW OF VMS I/O CONCEPTS

2 DIGITAL NETWORK ARCHITECTURE (DNA)

3 DECnet-VAX SOFTWARE COMPONENTS

4 DECnet-VAX DATA STRUCTURES

5 USING THE SYSTEM DUMP ANALYZER (SDA)

6 MAJOR NETWORK MECHANISMS

7 TRACING DECnet ACTIONS

8 NETWORK DESIGN AND PERFORMANCE CONSIDERATIONS

EX EXERCISES

AP APPENDIX

FIGURES

TABLES

EXAMPLES

STUDENT GUIDE

COURSE DESCRIPTION

This course is designed to provide an understanding of the various protocols, components, data structures, and algorithms used in implementing DECnet-VAX.

Topics studied include a review of protocols, major software components, DECnet data structure analysis, sample traces, and tools used for analyzing DECnet data structures and performance.

INTENDED AUDIENCE

The intended audience includes:

- Network programmers
- Technical support personnel
- Network managers interested in learning about the internals of DECnet-VAX

PREREQUISITES

Before enrolling in this course, the student should be familiar with:

- Basic Network Communications Concepts
- DECnet-VAX Programming and/or DECnet Management
- VMS Programming

The Educational Services courses that provide this information are:

- Programming in DECnet-VAX
- DECnet Management
- Utilizing VMS Features

LIST OF TOPICS

- A Review of VMS I/O Concepts
- DIGITAL Network Architecture Layers and Protocols
- Software Components of DECnet-VAX
- DECnet-VAX Data Structures
- Using SDA to Look at DECnet-VAX Data Structures
- Analysis of Major Network Functions
- Tracing of Selected Network Flows
- Performance-Related Parameters

TOPICS NOT DISCUSSED

- VMS Internals or Programming
- VAXcluster, LAT, SNA, PSI Protocols or Internals
- DECnet Management

COURSE GOALS

Upon completion of this course, students should be able to:

- Analyze the layers of DIGITAL Network Architecture (DNA)
- Explain the corresponding protocols of DNA
- Know the functions and interaction of the major DECnet-VAX software components
- Understand the format and use of DECnet data structures
- Be able to utilize the SDA utility to look at DECnet data structures
- Be able to trace the actions of major network mechanisms
- Be able to locate the appropriate code module to trace a network operation
- Understand network parameters related to network performance

RESOURCES

1. DECnet-VAX Internals - Student Workbook

2. DECnet-VAX Internals - Supplemental Listings - DNA Message Formats

3. DECnet-VAX Internals - Supplemental Listings - SDL Files

4. DECnet-VAX Source Listings

5. VAX/VMS Networking Manual (AA-Y512C-TE)

6. VAX/VMS Network Control Program Reference Manual (AA-Z425C-TE)

7. Digital's Network - An Architecture With a Future (EB-26013-42)

8. DECnet DNA (Phase IV) General Description (AA-N149A-TC)

9. DECnet-DNA NSP Functional Specification, Version 4.0 (AA-X439A-TK)

10. DECnet-DNA Routing Layer Functional Specification, Version 2.0 (AA-X35A-TK)

11. DNA Ethernet Node Product Architecture Specification, Version 1.0 (AA-X440A-TK)

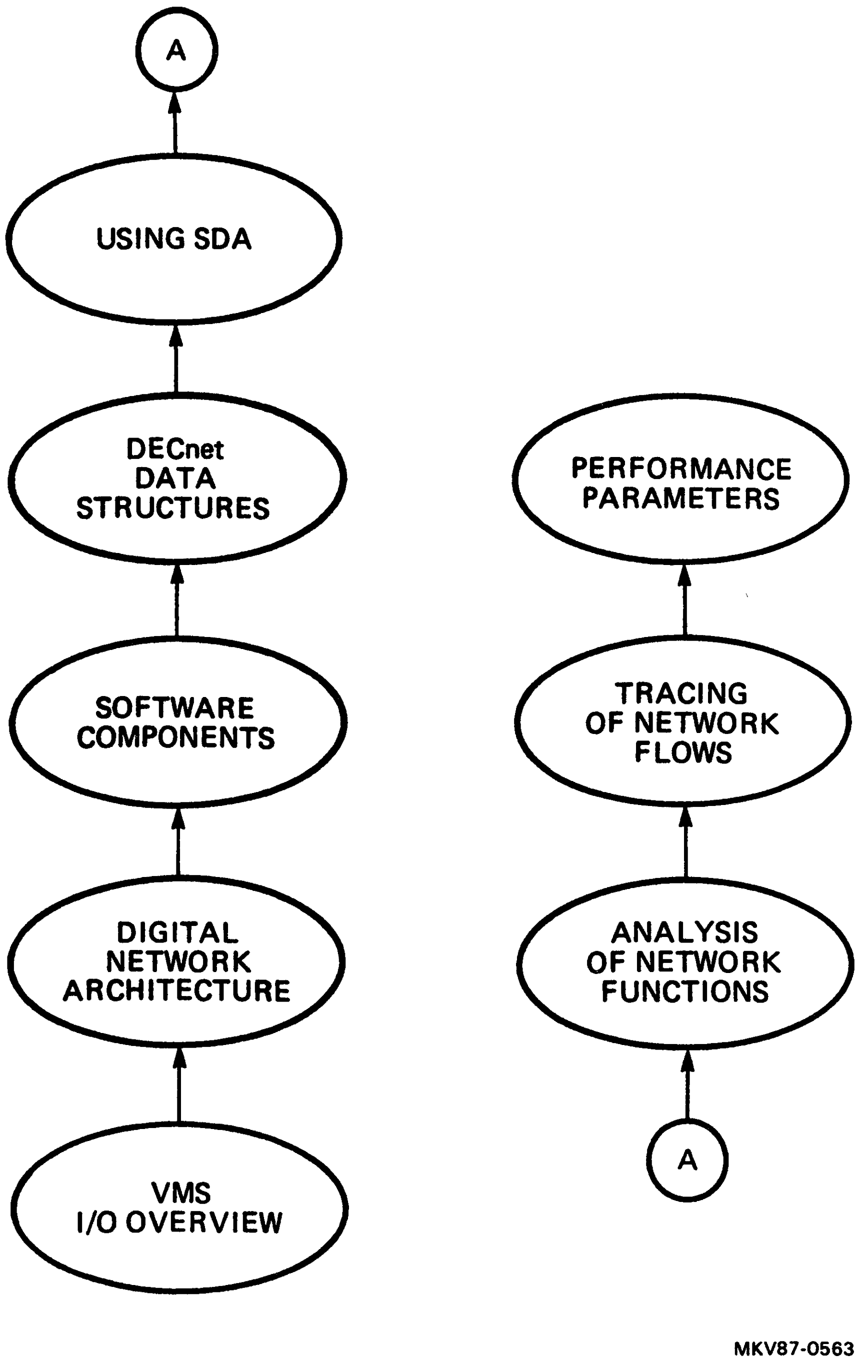
12. DECnet-DNA Ethernet Data Link Functional Specifications, Version 1.0 (AA-Y298A-TK)

13. Ethernet Specification, Version 2.0 (AA-K759B-TK)

14. DNA Maintenance Operations Functional Specification, Version 3.0 (AA-X436A-TK)

15. DNA Network Management Functional Specification, Version 4.0 (AA-X437A-TK)

COURSE MAP


A
USING SDA
DECnet
DATA
STRUCTURES
SOFTWARE
COMPONENTS
DIGITAL
NETWORK
ARCHITECTURE
VMS
I/O OVERVIEW
PERFORMANCE
PARAMETERS
TRACING
OF NETWORK
FLOWS
ANALYSIS
OF NETWORK
FUNCTIONS
A
MKV87-0563

COURSE OUTLINE

A. REVIEW OF VMS I/O CONCEPTS

- Virtual Memory
- QIO processing
- Device Driver Data Structures

B. DNA LAYERS AND PROTOCOLS

- Physical Link Layer
- Data Link Layer (DDCMP + NI protocol)
- Routing Layer and Protocol (ROUTING protocol)
- ECL Layer and Protocol (NSP protocol)
- Session Control Layer (SC protocol)
- Network Management Layer (MOP protocol)
- Network Applications Layer
- User Layer

C. MAJOR DECnet COMPONENTS FUNCTIONAL OVERVIEW

- Data Link Device Drivers
- NETDRIVER
- NETACP
- RMS, DAP Routines, and FAL_n
- RTTDRIVER, REMACP, and RTPAD
- Special Processes
- Objects
- NDDRIVER

- NCP, NML, MOM, MIRROR
- Other DECnet Components

D. DECnet-VAX DATA STRUCTURES

- Overall Data Structure Linkage
- VMS Data Structures Used by DECnet
- DECnet-VAX DATA STRUCTURES

  RCB Routing Control Block
  OA Output Adjacency Vector
  ADJ Adjacency Node Database Block
  LPD Logical Path Descriptor
  ICB Internal Connect Block
  LSB Logical Link Subchannel Block
  CNR Configuration Database Root Block
  CNF Configuration Data Block
  XWB Network Window Block
  NDI Remote Node Information
  LNI Local Node Information
  SPI Network Server Process Information
  WQE Work Queue Elements
  NDC Node Counter Block
  Object Information Block (OBI)
  LTB Logical Link Table
  AOA Area Output Adjacency Vector
  CRI Circuit Information
  PLI Physical Line Information

E. SDA ANALYSIS OF THE MAJOR DECnet DATA STRUCTURES

- Includes CCB, UCB, XWB, RCB, OA, ADJ, LPD and LTB structures

F. ANALYSIS OF MAJOR DECnet FUNCTIONS

- Logical Link Creation
- Routing Table Update
- Flow Control Mechanisms
- Message Segmentation

G. TRACING OF SELECTED NETWORK FLOWS

- Flow Diagrams of Major DECnet Actions
- Connect Initiate and Connect Confirm
- Transmit and Receiving Normal Data
- Other DECnet Actions

H. LOCATING INFORMATION IN LISTINGS

- Resources required before listing trace
- Techniques used in listing trace
- Modules and subroutines that make up the major DECnet components

I. HINTS ON SYSTEM TUNING

- System and User Buffering
- Timers
- Performance-Related Parameters

REVIEW OF VMS I/O CONCEPTS

INTRODUCTION

DECnet-VAX uses standard input/output (I/O) functions of VMS I/O processing. This chapter presents a brief review of VMS I/O processing.

Topics include:

- Virtual Memory
- QIO processing
- Device Driver Data Structures

1 VMS MEMORY

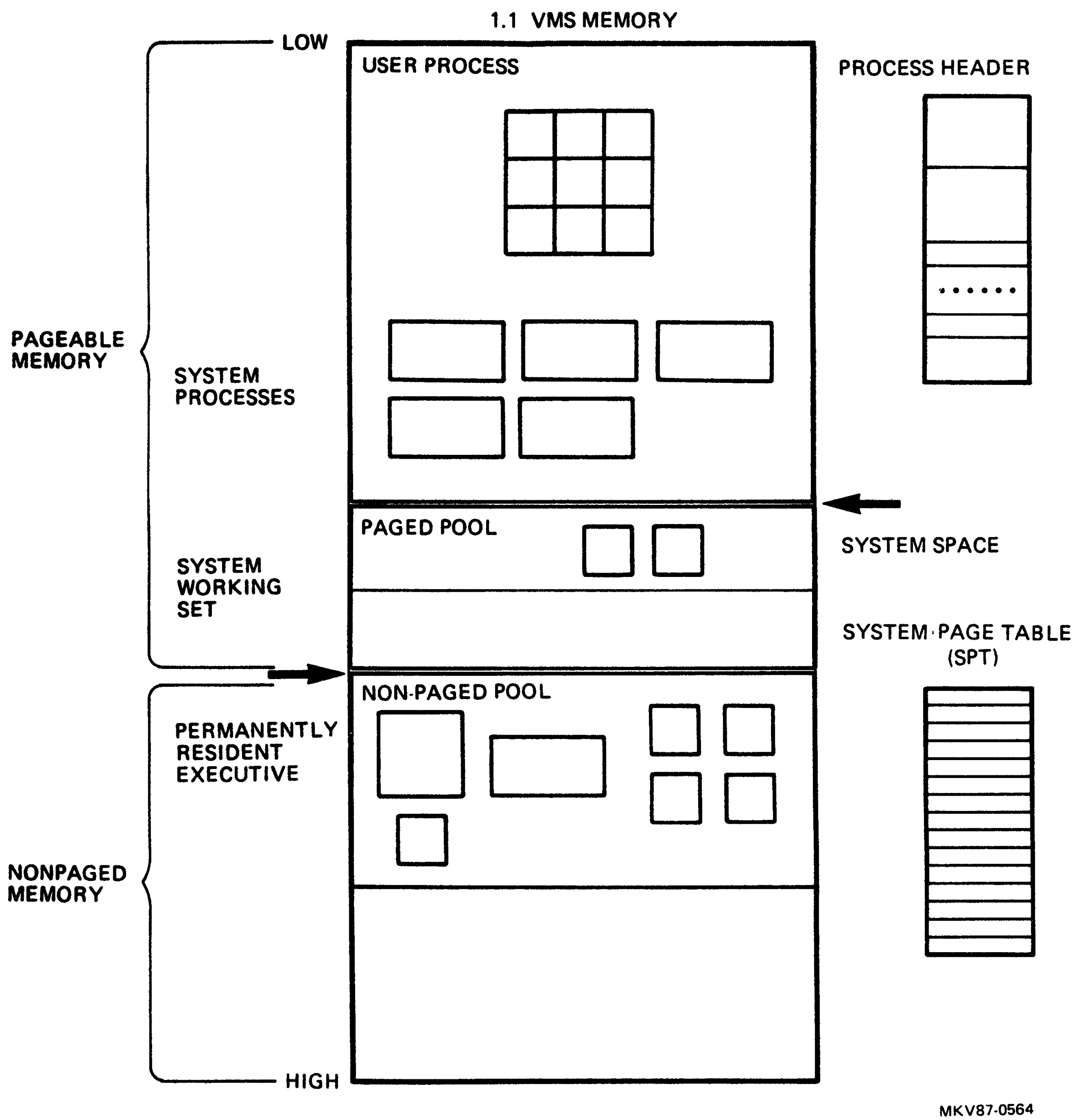


Figure 1-1 Physical Memory Layout

2 I/O HARDWARE/SOFTWARE INTERRUPTS

IPL 21 HARDWARE INTERRUPT

Execute Driver Interrupt Service Routine at IPL 21

Place itself on IPL 8 Fork Queue

SOFTINT #UCB$B_FIPL(R5)

REI

IPL 8 SOFTWARE INTERRUPT

Execute FORK DISPATCHER Interrupt Service Routine

Dequeue first Fork Block

Execute Driver IPL Fork Code

Access shared driver data or

Suppose I/O completed - need VMS completion code

Place data block on queue for VMS post processing

SOFTINT #IPL$_IOPOST

REI

#IPL$_IOPOST SOFTWARE INTERRUPT

Execute I/O POST PROCESSOR Interrupt Service Routine

Suppose need to access PCB

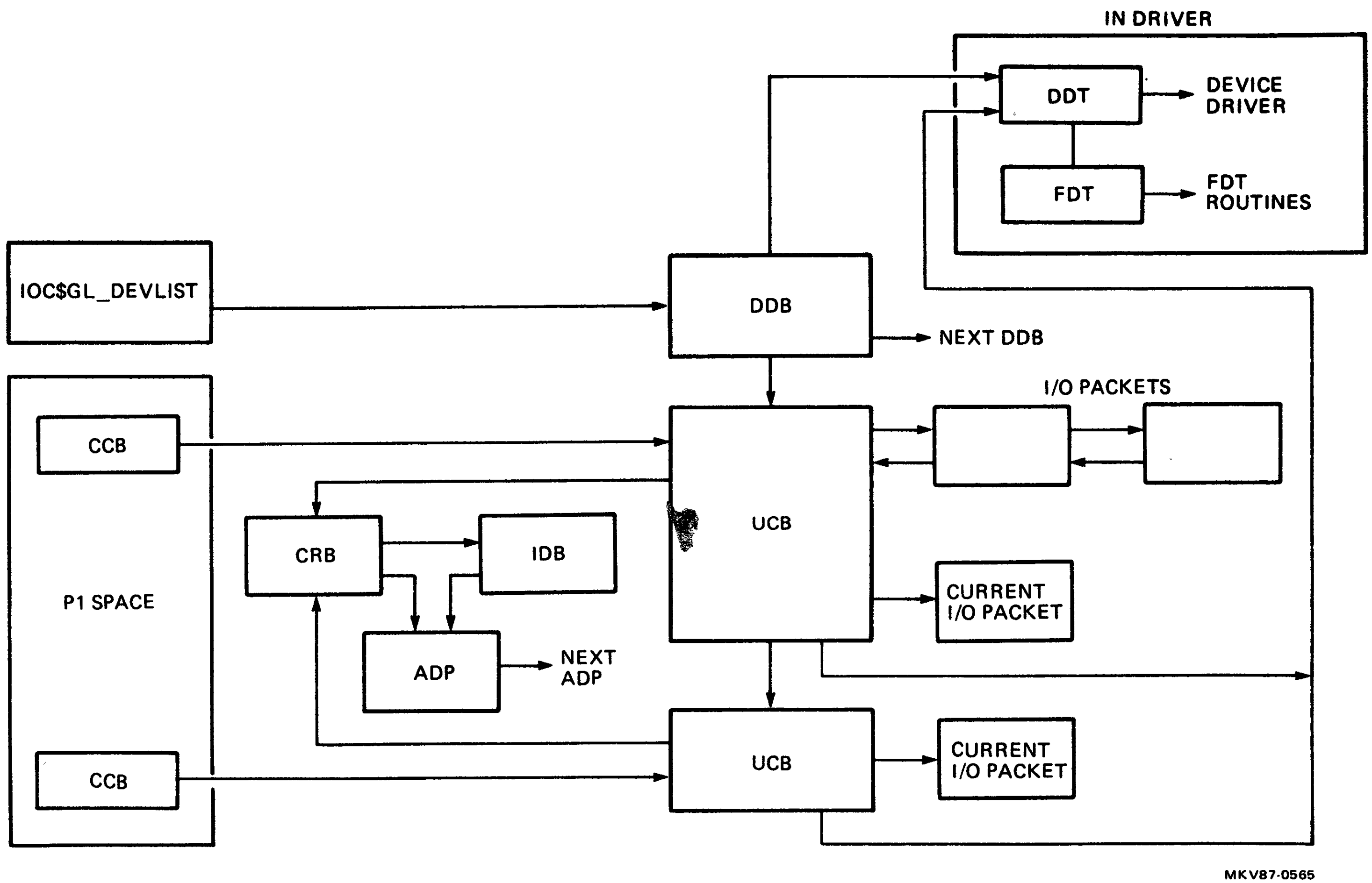
Raise IPL to SYNCH level

Set event flag, queue AST, etc.

Lower IPL to IPL$_IOPOST

Possibly initiate SCHEDULER - SOFTINT #IPL$_SCHED

3 DEVICE DRIVER DATA STRUCTURES



DIGITAL NETWORK ARCHITECTURE (DNA)

INTRODUCTION

DIGITAL Network Architecture (DNA) is the framework for all DIGITAL communications products. DECnet-VAX is one implementation of the DNA.

DNA includes the functional specifications that govern the interrelationship of the various software components making up DECnet. DNA defines **Interfaces** and **Protocols**.

Interfaces are definitions of specific functional boundaries between DECnet software components residing within a single node. The boundaries are structured as a hierarchical set of layers with DECnet software arranged as modules within these layers.

Protocols are sets of messages (and rules for exchanging the messages) between modules with equivalent functions in the same layer, but residing in different nodes.

Topics include:

- DECnet Layers and Protocols
- Functions of each Layer
- Message Types of each Layer

1 DECnet LAYERS AND PROTOCOLS

- Layers are defined in the DIGITAL Network Architecture (DNA)
- PHASE IV of DNA announced in May, 1982
- Similar to the International Standards Organization (ISO) Open System Interconnect (OSI) Model
- Each layer is responsible for certain functions logically different from those that are supported by another layer
- Layers communicate across a network by exchanging messages according to the rules specified by a corresponding protocol
- A network protocol is a set of formal rules representing a layer's logic and communication procedure
- Protocol is transparent across a network
- Implementation of a layer, as described by a particular protocol, is logically transparent in a network

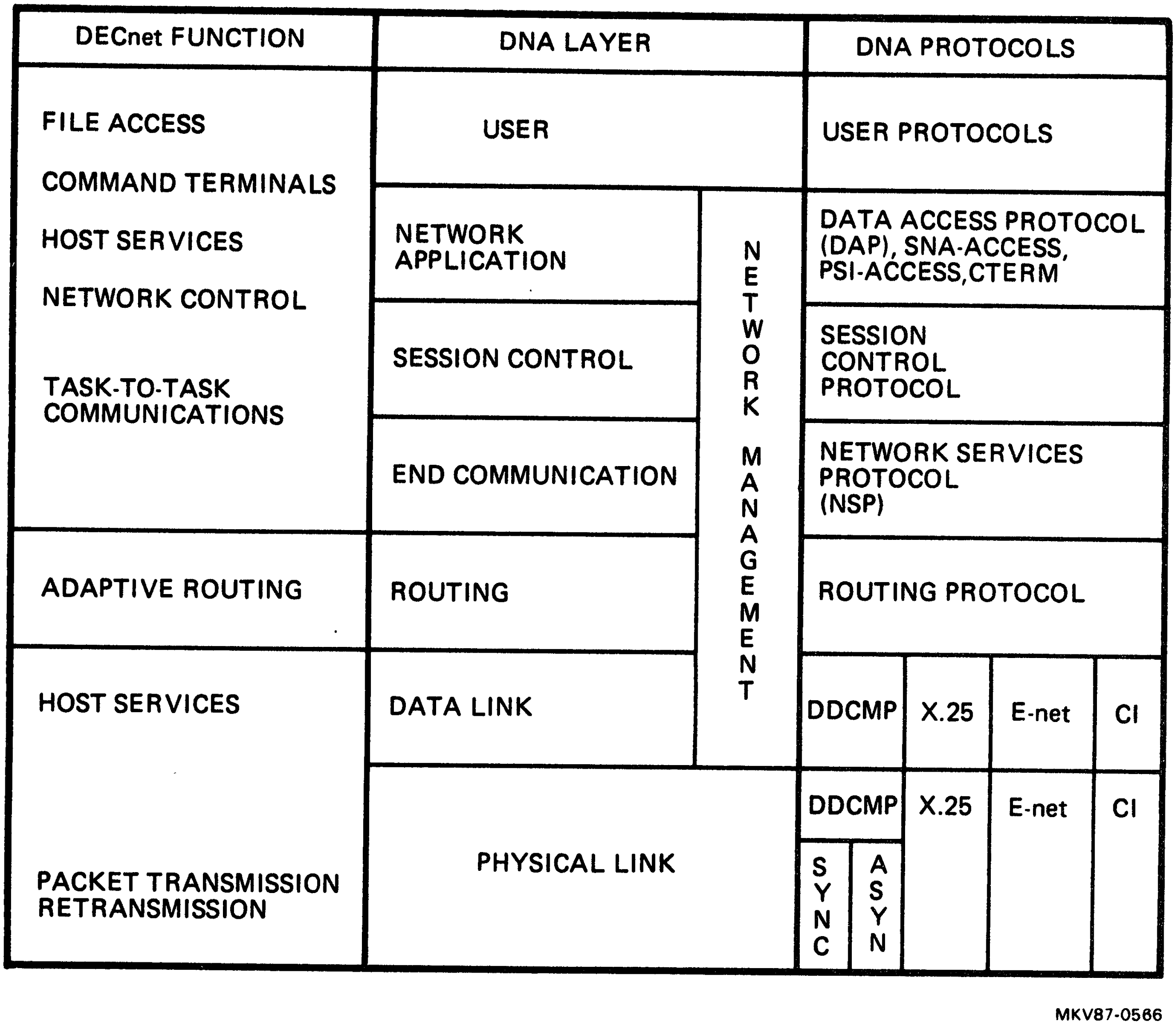


Figure 2-1 DECnet Functions and Related DNA Layers and Protocols

Table 2-1 DECnet Layers and Protocols

Layer	Protocol	Comments
User	User-specific	Supports user services and programs User determines the sequence of read and/or write operations (Protocol)
Network Management	NICE MOP Event Logger	Provides a means to configure, monitor, and control the local node and other nodes (Privilege is required for some of the operations and displays)
Network Application	Multiple	Invokes tasks on behalf of the user to perform specific functions
Session Control		Performs operating system dependent functions Mapping node name to address Creating processes Identifying end-user tasks Validating incoming connect requests
End Communications	NSP	Performs operating system independent functions Logical link management Message segmentation
Routing	Routing	Handles packet routing, congestion control, and packet lifetime control
Data Link	DDCMP Ethernet X.25	Transfers data to physical link layer
Physical Link	CI PCL	Transfers data over physical link to adjacent node

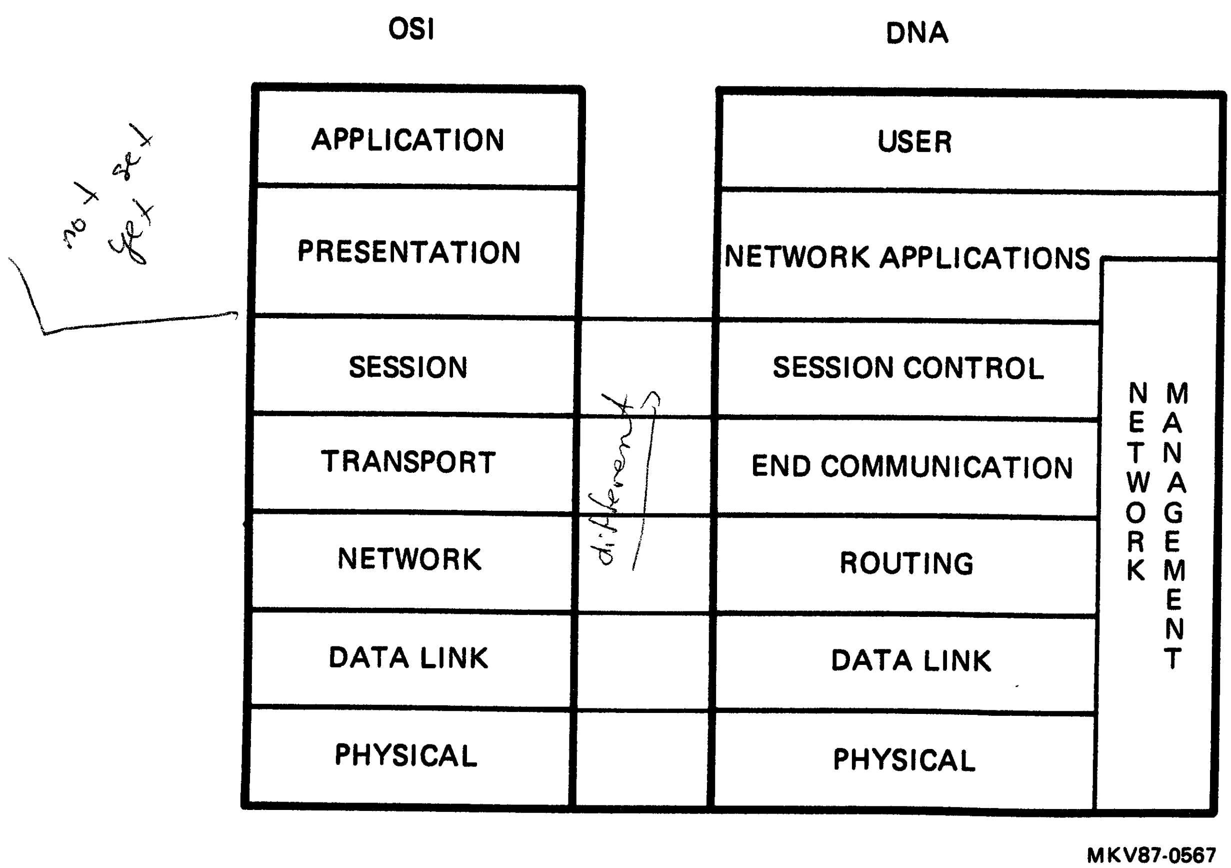


Figure 2-2 Layers of DNA vs Layers of OSI

Phase V is merging of OSI algorithms into DECnet implementation.

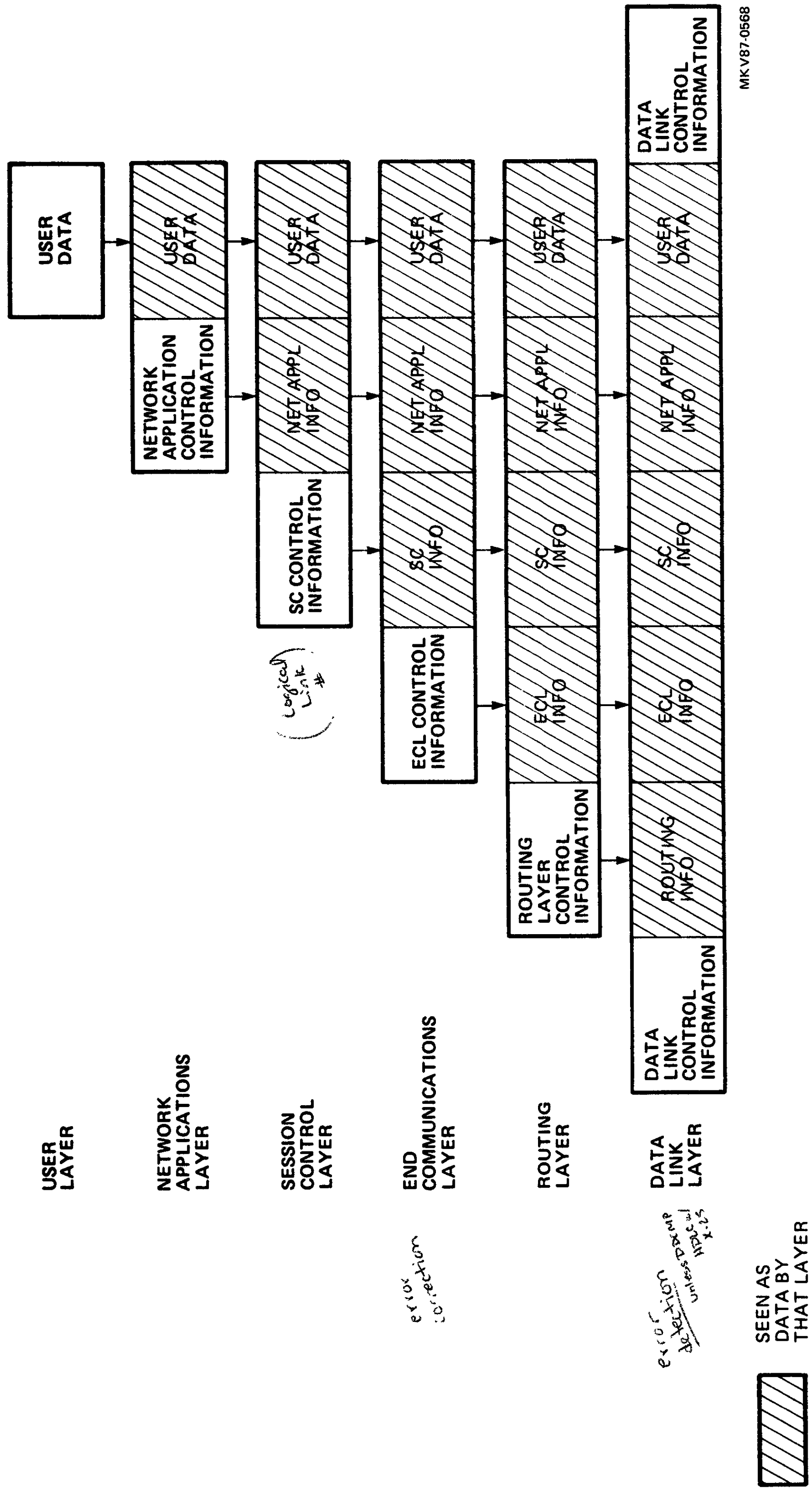


Figure 2-3 Data Enveloping

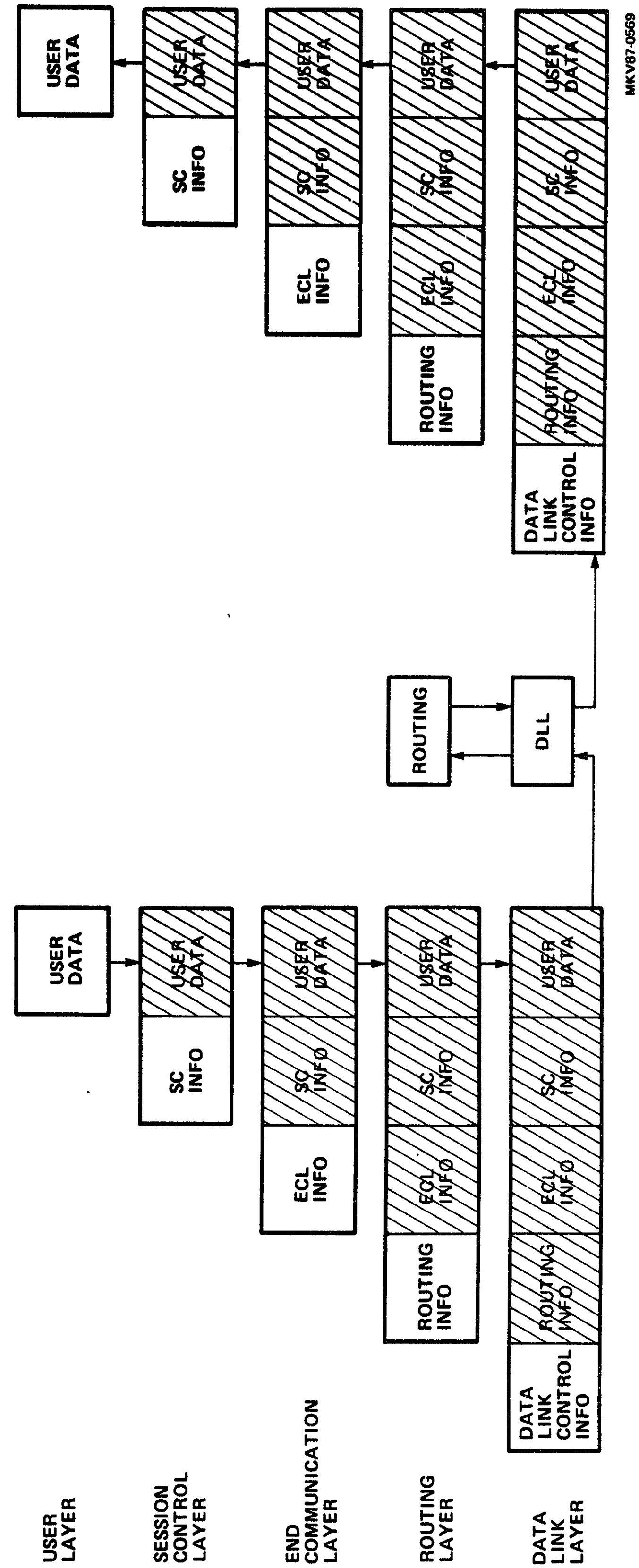


Figure 2-4 Data Flow Between Two Nonadjacent DECnet Nodes

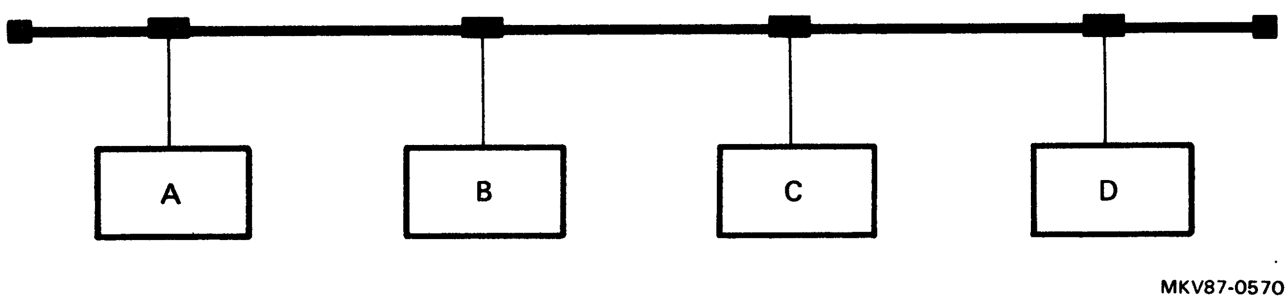


Figure 2-5 Multiple Node Ethernet Network

2 Ethernet DATA LINK AND PHYSICAL LINK LAYERS

- Specification developed jointly by DIGITAL, Intel, and XEROX
- Supported in DECnet PHASE IV
- Ethernet and IEEE 802.3 LAN standard include a physical link layer and data link layer specification
  - Physical link level

    Manchester-encoded, digital baseband signals
  - Data link level

    Carrier Sense Multiple Access Collision Detect CSMA/CD
    like people @ cocktail party
- Baseband, broadband or "thinwire" cables
- Transmission capability of 10 MBits
- Inherently low error rate
- Local area network

  (<= 2.8 KM (1.74 miles) on baseband)
  (<= 3.8 KM (2.36 miles) on broadband) doesn't need repeaters.
- Multiple physical Ethernet LANS may be connected using a LAN bridge
- Multiple node broadcast capabilities
- Supports up to 1024 nodes per Ethernet LAN
- Any/all/none of the nodes may/need be a router

- based on speed of light.
- if repeaters (2) need 9.9.2 secs to listen for collision
- baseban

2.1 Ethernet Addresses and Protocol Types

- Blocks of addresses assigned by XEROX Corporation to producers of Ethernet interfaces
- Address field is 48 bits in length
- Bytes are received and translated into the logical order in which they were transmitted (left to right)

  (Bits within the bytes are transmitted from right to left)

  SAMPLE ADDRESS: AA - 01 - 23 - 45 - 67 - FF   AB is multicast
  TRANSMITTED: 1 > 2 > 3 > 4 > 5 > 6

- Ethernet address can be one of three types:

  A. Physical address:

  Unique for every station

  1. Hardware Address - in ROM on controller
  2. Extended DECnet Address - computed using DECnet

  B. BROADCAST ADDRESS:
  To every node on the Ethernet
  FFFFFFFF

  C. MULTICAST ADDRESS:
  Associated with a group of nodes
  DELUA can recognize up to 10 multicast addresses

- Physical Address is set to Hardware Address on powerup
- When DECnet is started on an Ethernet line, the Physical Address is overwritten to be the extended DECnet Address
- Appends DECnet Node Address to constant AA-00-04-00
- DNA Phase IV Node Addresses are in the range AA-00-04-00-00-00 through AA-00-04-00-FF-FF

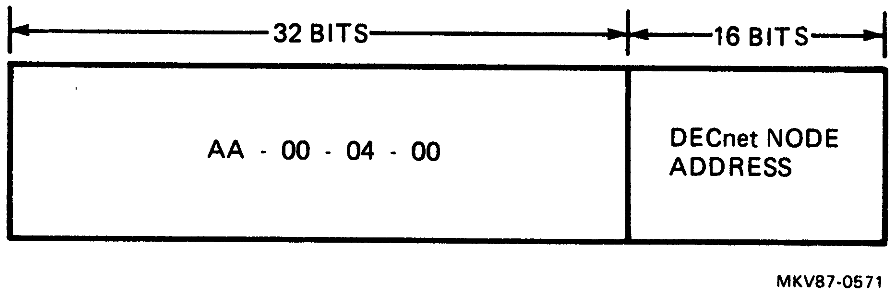


Figure 2-6 Format of Ethernet Physical Address

- To determine the Physical Address of a node:

1. Convert the Phase IV node address to its decimal equivalent using :

   DECnet address = (area-number * 1024) + node-number

2. Convert the decimal Node Address to its hexadecimal equivalent, reversing the order of the bytes

3. Append the hexadecimal Node Address to AA-00-04-00

   ASSUME DECnet Node Address = 4.171

   (4 * 1024) + 171 = 4096 + 171
   = 4267 (decimal)
   = 10AB (hexadecimal)

   Ethernet Decimal Address = AA-00-04-00-AB-10

Table 2-2 Ethernet Protocol Types and Multicast Addresses

DIGITAL Protocol Types:

Value	Meaning
60-01	DNA Dump/Load (MOP)
60-02	DNA Remote Console (MOP)
60-03	DNA Routing
60-04	Local Area Transport (LAT)
60-05	Diagnostics
60-06	Customer use
60-07	System Communication Architecture (SCA)

Cross-Company Protocol Type:

Value	Meaning
90-00	Loopback

DIGITAL Multicast Addresses:

Value	Meaning
AB-00-00-01-00-00	DNA Dump/Load Assistance (MOP)
AB-00-00-02-00-00	DNA Remote Console (MOP)
AB-00-00-03-00-00	All PHASE IV routers
AB-00-00-04-00-00	All PHASE IV end nodes
AB-00-00-05-00-00 thru AB-00-03-FF-FF-FF	Reserved for future use
AB-00-03-00-00-00	Local Area Transport (LAT)
AB-00-04-00-00-00 thru AB-00-04-00-FF-FF	Customer use
AB-00-04-01-00-00 thru AB-00-04-01-FF-FF	System Communication Architecture (SCA)

Cross-Company Multicast Addresses:

Value	Meaning
FF-FF-FF-FF-FF-FF	Broadcast
CF-00-00-00-00-00	Loopback Assistance

2.2 Ethernet Message Format

The data encapsulation function of the Data Link Layer comprises the construction and processing of frames. The subfunctions of framing, addressing, and error detection are reflected in the frame format as follows:

- Framing
- Addressing
- Error Detection

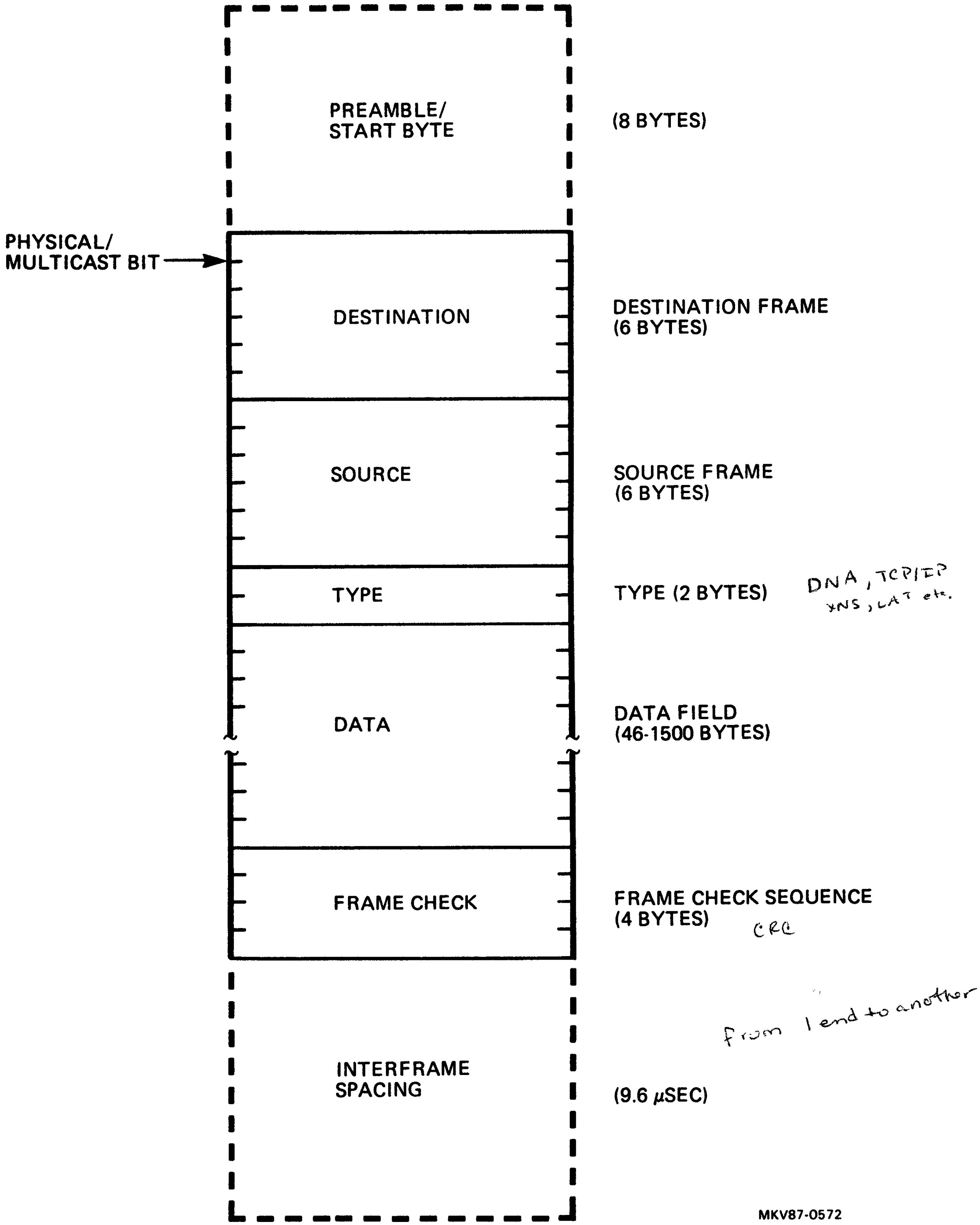


Figure 2-7 Ethernet Frame Format -
(No Padding - Data >=46 Bytes)

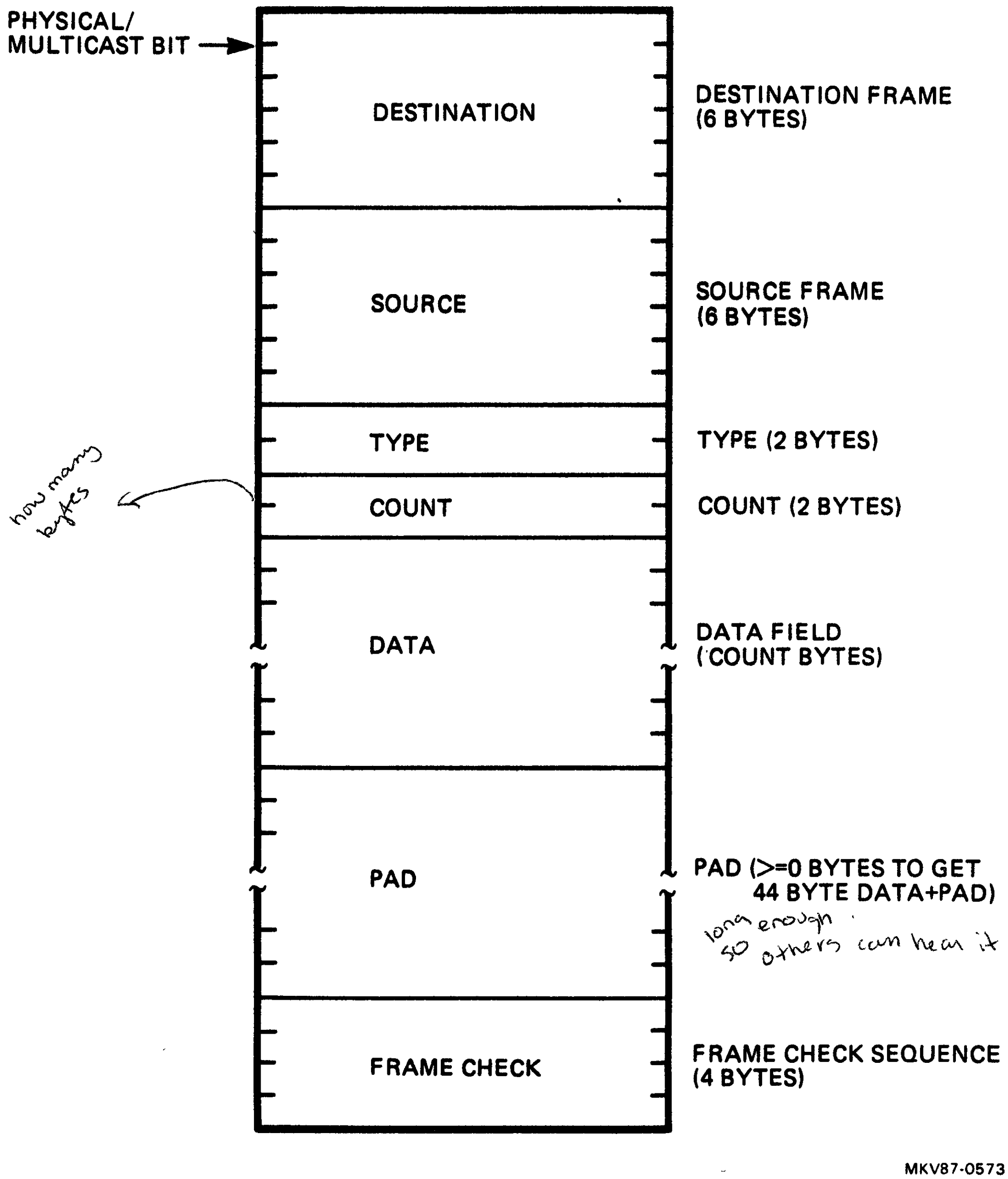


Figure 2-8 Ethernet Frame Format with Padding

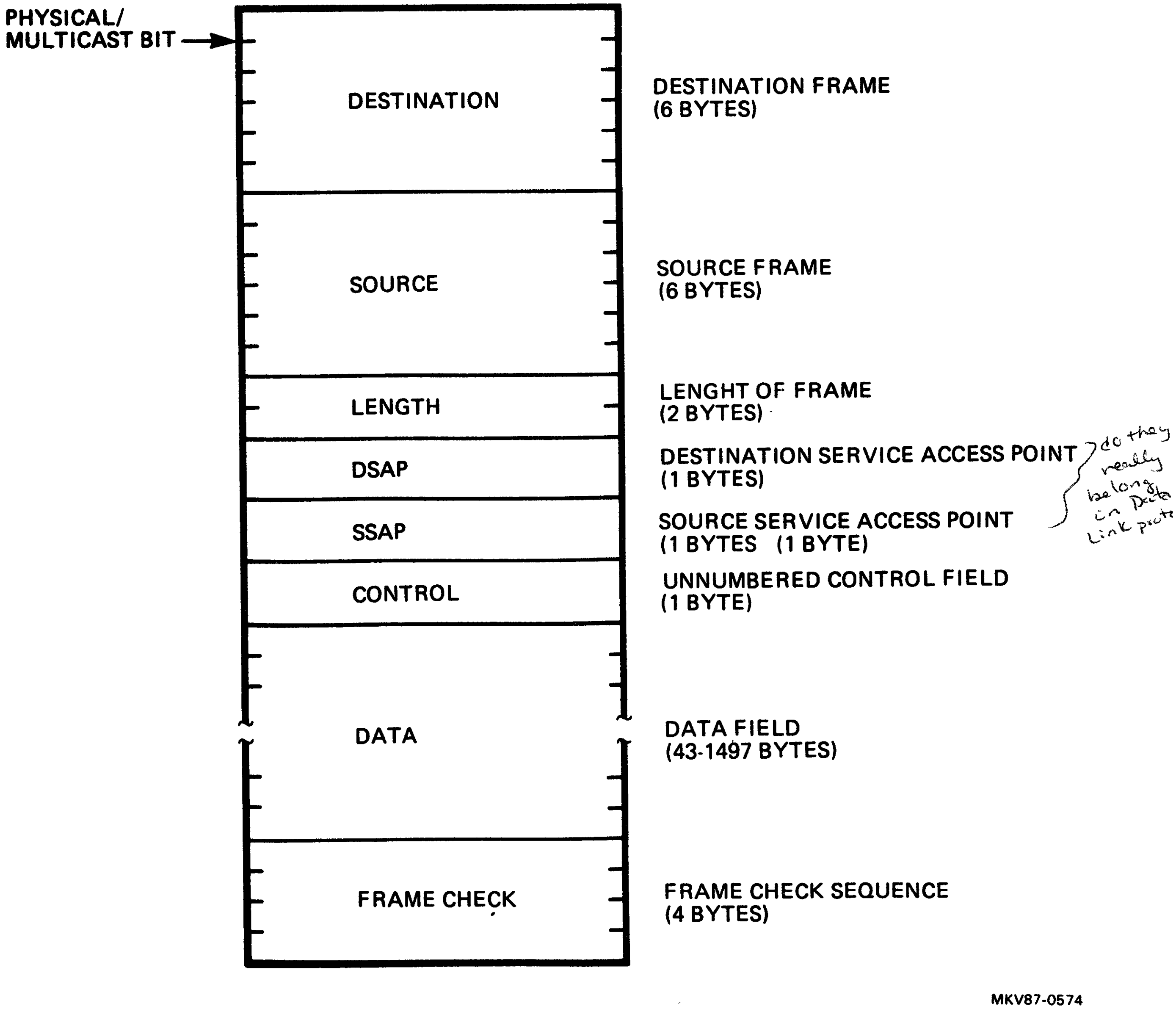


Figure 2-9 IEEE 802.3 Packet Format

3 DIGITAL DATA COMMUNICATIONS MESSAGE PROTOCOL (DDCMP)

- Developed by DIGITAL
- The first Data Link Protocol offered on DECnet
- A Physical and Data Link Protocol that provides control functions on the link between two adjacent nodes only
- Guarantees error-free sequential delivery of packets
- Uses sequenced message acknowledgment scheme (CRC-16, ACKs, NAKs, request for retransmits)
- Nonbroadcast circuit point-to-point and multipoint configurations
- Supports normal and maintenance mode
- Isolates higher layers from the physical characteristics of the line
- Provides efficiency on channels with long delay times
- Supports a wide variety of channel types
- Implemented both in software (driver) and hardware (Controller)

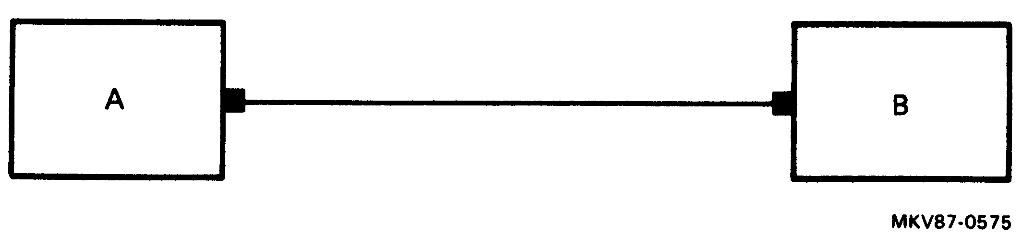


Figure 2-10 Two Node - Point-to-Point Connection

4 DDCMP MESSAGE TYPES

- Data message (SOH)
- Control message
  - ACK (pos. acknowledgment)
  - NAK (neg. acknowledgment)
  - REP (repl. for packet req.)
  - STRT (start DDCMP)
  - STACK (STRT ACK)
- Maintenance message (DLE)

NOTE

DDCMP message formats are detailed in the Supplemental Listings book.

```
User requests start-up
R=0,N=0,A=0,T=1,X=0
                       ------>
                       STRT              Notify user of start-up at
                                         other station
                                         User requests start-up
                                         R=0,N=0,A=0,T=1,X=0
                       <------
                       STRT
                       ------->
                       STACK             Enter running state
                       <------
Enter running state    ACK (RESP=0)
```

Example 2-1 DDCMP Start-Up Sequence with No Errors

```
User requests start-up  --//-->   8 byte control msg      on starting
      set line state on  STRT                              + blink every
                         !                                 3 secs
Time-out                 !
                         ------>
                         STRT            Notify user, user requests
                                         start-up
                         <------
                         STRT  !
                               !
                         --//-->
                         STACK !
                               !         Time-out
                         <------
                         STRT
                         ------>
                         STACK           Enter running mode
                         <--//--
                         ACK(RESP=0)
                         !
Time-out                 !
                         ------>
                         STACK
                         <------
                         ACK(RESP=0)
Enter running state
```

Example 2-2 DDCMP Start-Up Sequence with Errors

```
User requests transmit
R=0,N=1,A=0,T=2,X=1
                          ------>
                          DATA(NUM=1,RESP=0)
                                       Message received and given
                                       to user
                                       R=1,N=0,A=0,T=1,X=0
                          <------
                          ACK(RESP=1)
User requests transmit
R=0,N=2,A=1,T=3,X=2
                          ------>
                          DATA(NUM=2,RESP=0)
                                       Message received and user
                                       requests transmit
                                       R=2,N=1,A=0,T=2,X=1
                          <------
                          DATA(NUM=1,RESP=2)
Message received and user
requests transmit
R=1,N=3,A=2,T=4,X=3
                          ------>
                          DATA(NUM=3,RESP=1)
                                       Message received
                                       R=3,N=1,A=1,T=2,X=1
User requests transmit
R=1,N=4,A=2,T=5,X=4
                          ------>
                          DATA(NUM=4,RESP=1)
                                       Message received
                                       R=4,N=1,A=1,T=2,X=1
                                       User requests transmit
                                       R=4,N=2,A=1,T=3,X=2
                          <------
                          DATA(NUM=2,RESP=4)
Message received
R=2,N=4,A=4,T=5,X=4
                          ------>
                          ACK(RESP=2)
                                       ACK received
                                       R=4,N=2,A=2,T=3,X=2
```

Piggyback ack

Example 2-3   DDCMP Data Transfer with No Errors

whether to piggyback or not is implementation dependent.

```
User requests transmit
R=0,N=1,A=0,T=2,X=1          ----/-->
                             DATA(NUM=1,RESP=0)
                                                Received in error
                             <------            R=0,N=0,A=0,T=1,X=0
                             NAK(RESP=0)
NAK received
R=0,N=1,A=0,T=1,X=1
Retransmit
R=0,N=1,A=0,T=2,X=1          ------>
                             DATA(NUM=1,RESP=0)
                                                Message received and
                                                user requests transmit
                             <------            R=1,N=1,A=0,T=2,X=1
                             DATA(NUM=1,RESP=1)
                             ---/-->
                             ACK(RESP=1)
                                                Queue NAK for R=1
User requests 3 transmits
R=1,N=2,A=1,T=3,X=2          ------>
                             DATA(NUM=2,RESP=1)
                                                Message received
                                                R=2,N=1,A=1,T=2,X=1

R=1,N=3,A=1,T=4,X=3          ------>
                             DATA(NUM=3,RESP=1)
                                                Message received
                                                R=3,N=1,A=1,T=2,X=1

R=1,N=4,A=1,T=5,X=4          ------>
                             DATA(NUM=4,RESP=1)
                                                Message received
                                                R=4,N=1,A=1,T=2,X=1
                             <------            Queue ACK for R=4
                             NAK(RESP=1)        Queued NAK returned
NAK received
R=1,N=4,A=1,T=2,X=4
Retransmit
R=1,N=4,A=1,T=3,X=2          ------>
                             DATA(NUM=2,RESP=1)
                                                Message ignored
                             <------
                             ACK(RESP=4)        Queued ACK returned
All messages complete
R=1,N=4,A=4,T=5,X=2
```

Example 2-4 DDCMP Data Transfer with CRC Errors and NAKing

NAK 10 means last good message received ~~was~~ 10.

Window size ~~m~~ can be 256 but because of buffer restraints use smaller window (ie 8)

```
User requests transmit
R=0,N=1,A=0,T=2,X=1
                                ------>
                                DATA(NUM=1,RESP=0)
                                !
                                !          Message received
                                !          R=1,N=0,A=0,T=1,X=0
                                !
                                <--//--
                                !  ACK(RESP=1)
Time-out                        !
                                !------>
                                REP(NUM=1)
                                <------
                                ACK(RESP=1)   ACK response to REP
                                              R=1,N=0,A=0,T=1,X=0

ACK received
R=0,N=1,A=1,T=2,X=1
User requests transmit
R=0,N=2,A=1,T=3,X=2
                                !--//-->
                                !  DATA(NUM=2,RESP=0)
                                !
                                !
                                !
Time-out                        !
                                !------>
                                REP(NUM=2)
                                <------
                                NAK(RESP=1)   NAK response to REP
                                              R=1,N=0,A=0,T=1,X=0

NAK received
R=0,N=2,A=1,T=2,X=2
Retransmit
R=0,N=2,A=1,T=3,X=2
                                ------>
                                DATA<NUM=2,RESP=0)
                                              Message received
                                              R=2,N=0,A=0,T=1,X=0
                                <------
                                ACK(RESP=2)
ACK received
R=0,N=2,A=2,T=3,X=2
```

Example 2-5 DDCMP Data Transfer with Errors Causing Reply Timeouts (Sheet 1 of 2)

```
User requests transmit
R=0,N=3,A=2,T=4,X=3           !------>
                              ! DATA(NUM=3,RESP=0)
                              !
                              !              Message received
                              !              R=3,N=0,A=0,T=1,X=0
User requests transmit        !
R=0,N=4,A=2,T=5,X=4           !
                              !--//-->
                              ! DATA(NUM=4,RESP=0)
User requests transmit        !
R=0,N=5,A=2,T=6,X=5           !
                              !------>
                              ! DATA(NUM=5,RESP=0)
                              !
                              !              Message ignored
                              !              R=3,N=0,A=0,T=1,X=0
                              !
                              <--//--        ACK to received message
                              ! ACK(RESP=3)
Time-out                      !
                              !------>
                                REP(NUM=5)
                              <------
                                NAK(RESP=3)  NAK response to REP
                                             R=3,N=0,A=0,T=1,X=0
NAK received
R=0,N=5,A=3,T=4,X=5
Retransmit
R=0,N=5,A=3,T=5,X=4           ------>
                              DATA(NUM=4,RESP=0)
                                             Message received
                                             R=4,N=0,A=0,T=1,X=0
Retransmit
R=0,N=5,A=3,T=6,X=5           ------>
                              DATA(NUM=5,RESP=0)
                                             Message received
                                             R=5,N=0,A=0,T=1,X=0
                              <------
                              ACK(RESP=5)    ACK to received messages
All messages complete
R=0,N=5,A=5,T=6,X=5
```

Example 2-5 DDCMP Data Transfer with Errors Causing Reply Timeouts (Sheet 2 of 2)

5 X.25 DATA LINK AND PHYSICAL LINK PROTOCOL

- A procedure for delivering packets to a **DCE** (Data Communications Equipment) from a **DTE** (Data Terminating Equipment)
- Recommendation from **CCITT** (Comite Consultatif International Telephonique et Telegraphique) for interfacing to public packet-switching data networks
- Contains three levels:

  Level 1 - Physical
  Level 2 - Frame
  Level 3 - Packet

- The Level 2 (Frame Level) is LAPB (Link Access Protocol Balanced) which does link control, error detection and retransmission similar to DDCMP
- DECnet Uses data link mapping technique to incorporate x.25
- X.25 Line Devices:
  - DUP11
  - KMS11
  - KMV11
  - DMF32 (Synchronous Line Unit)

5.1 Data Link Mapping

- Available in nodes with both DECnet and PSI support
- A DECnet program on one node can communicate by means of PSI network to another DECnet program on another node
- A message already containing routing header is given to the X.25 module rather than the DDCMP or Ethernet module
- The X.25 module puts on the additional x.25 headers (Level 3 and Level 2) before transmitting it on the PSI network
- The DECnet node with PSI support on the other side strips off the X.25 headers and uses the routing header to determine routing
- The layered architecture enables all of this to take place transparently to the user program

6 COMPUTER INTERCONNECT (CI)

own protocol

- Supported as a DECnet data link layer protocol between VMS nodes
- CI interface may be used as a DECnet data link

  Uses the CI750 (750), CI780 (SBI Systems), or CIBCI (BI Systems)
- CNDRIVER is the DECnet driver for the CI

  Implemented as a multipoint line
- DECnet software is independent of cluster software
- Must have at least one router in a configuration of > 2 nodes

  Each node with >1 circuit turned on must be a routing node
- End nodes on the CI cluster can only send to a routing node

Last chance

poor performance.

7 DNA ROUTING LAYER

7.1 Routing Layer Overview

- Defines the mechanism for routing user data between two nonadjacent nodes
- Provides a "picture" of current network topology and determines path a packet will take between two nodes
- Masks the physical and topological characteristics of the network from higher layers
- DECnet Phase IV supports a network of up to 65000 nodes
- Routing Terms:
  - Path Length: The number of hops along a path between two nodes
  - Circuit Cost: Positive integer value (1-25) associated with using a circuit
  - Path Cost: The sum of the circuit costs along a path between two nodes

7.2 Routing Processes and Databases

A. Decision Process

B. Forwarding Process

C. Receive Process

D. Congestion Control

E. Packet Lifetime Control

F. Update Process

G. Routing Database

H. Forwarding Database

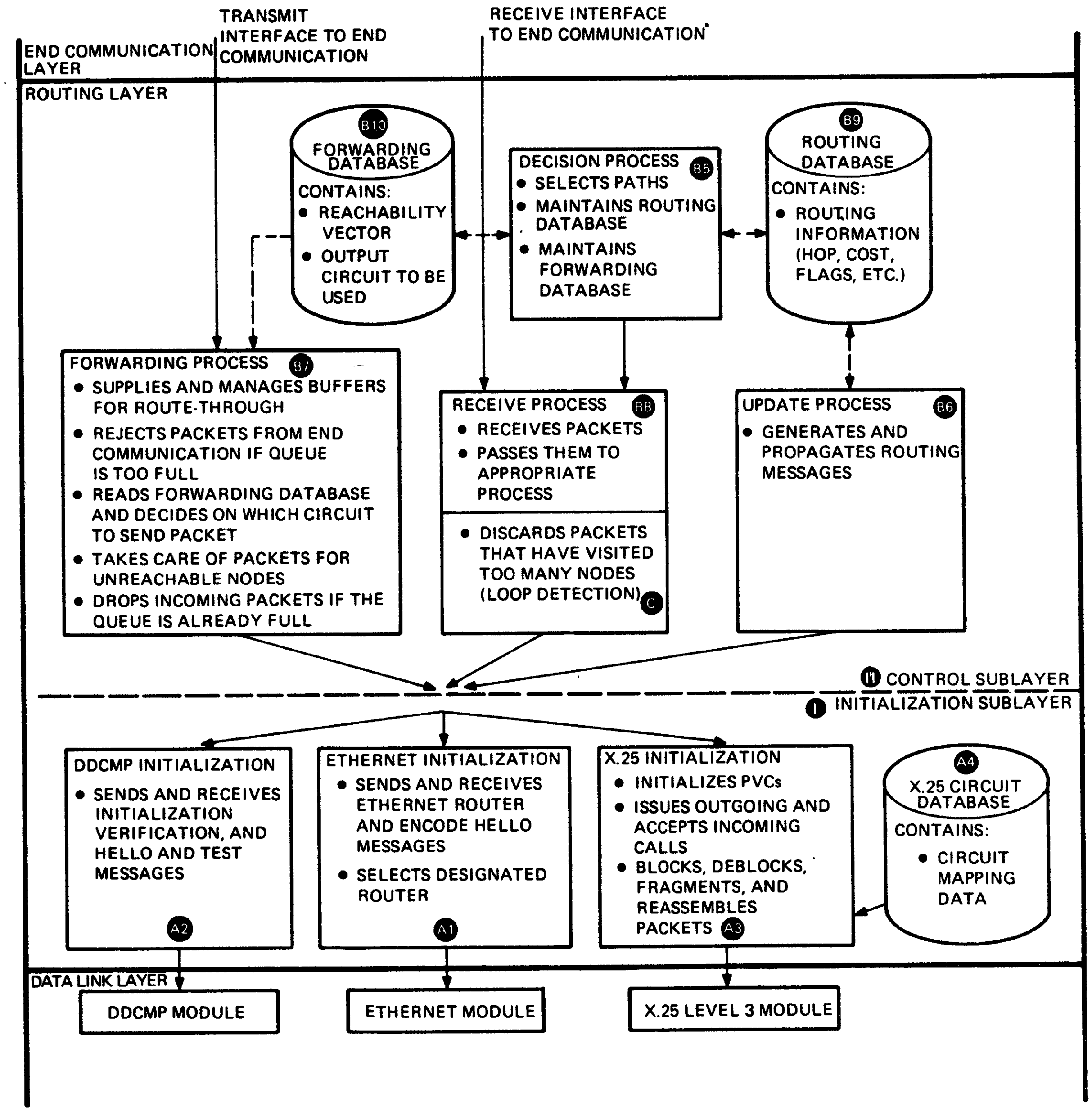


Figure 2-11 Routing Layer Components

7.2.1 Decision Process

- Selects routes to each destination in the network
- Consists of:
  - A connectivity algorithm that maintains path lengths
  - A traffic assignment algorithm that maintains path costs
- Receiving a routing message from an adjacent node causes the routing node to execute the decision process
- This results in the determination of:
  - <CIRCUIT, NEIGHBOR> pairs (known as adjacencies) along which to forward packets
  - Conclusions that a particular destination node is reachable or unreachable
- Parameters in the routing database that the decision process uses:

  Maximum address

  Maximum cost

  Maximum hops

  Maximum circuits

  Circuit costs

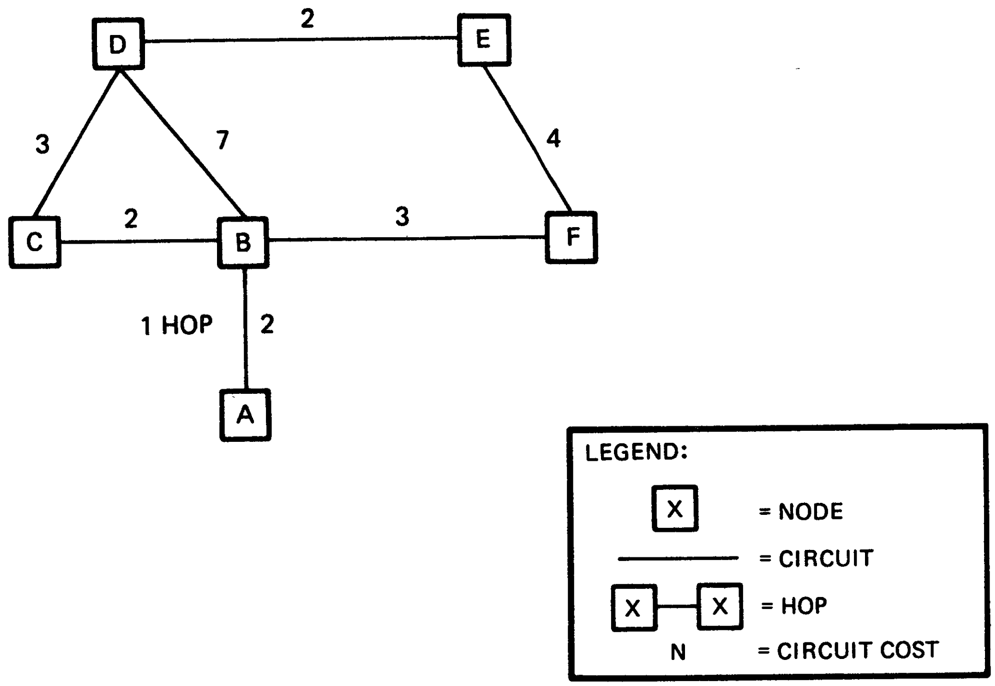
  Maximum broadcast routers

  Maximum broadcast nonrouters

  Maximum area (for area routers)

  Area maximum hops (for area routers)

  Area maximum cost (for area routers)



NODE A WANTS TO SEND A PACKET TO NODE D (THREE POSSIBLE PATHS)		
PATH	PATH COST	PATH LENGTH
A—B, B—C, C—D	2 + 2 + 3 = 7*	3 HOPS
A—B, B—D	2 + 7 = 9	2 HOPS
A—B, B—F, F—E, E—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.

MKV87-0577

Figure 2-12 Routing Overview

7.2.2 Forwarding Process

- Manages the buffers necessary to support packet route-through to all destinations
- Performs a table lookup to determine the output adjacency to use for forwarding to a given destination
- Strips off the area fields when forwarding to a Phase III node
- Fills in the area fields when receiving from a PHASE III node
- Marks Intra-Ethernet packets

7.2.3 Receive Process

- Inspects a packet's route header
- Dispatches the packet to an appropriate routing layer control component or to the End Communications Layer (ECL)

7.2.4 Congestion Control

- Manages buffers by limiting the maximum number of packets on the transmit queue for a circuit
- Regulates the ratio of packets received directly from ECL to route-through packets
- Checks the packet size for each packet to be sent

7.2.5 Packet Lifetime Control

The packet lifetime control component requires three processes:

- Loop Detector
  - Process prevents excessive packet looping
  - Counts the number of nodes a packet has visited and removes a packet when it exceeds **MAX** Visits
- Node Listener and Node Talker
  - Places an artificial load on the adjacency so failures can be detected
  - Provides for detection of adjacent routing layer halt and adjacent node identity change
  - Uses parameters

    Hello Timer
    Listen Timer
    Routing Timer
    Broadcast Routing Timer

7.2.6 Update Process

NOTE

The Routing Layer Update Process is covered in Module 6, Section 4.

7.3 Routing Protocol Message Types

A. Packet Routing Header

Used for data messages.

Routing data packets are headers that prefix the information that comes from the higher layer (End Communications).

Header information has the source and destination node addresses, a visit count field, and a route flag.

B. Routing Message

Provides information necessary for updating the routing database of an adjacent node.

Segmented routing messages are used. Each segment consists of a count, a start node ID followed by the minimum hop, and minimum cost information.

C. Ethernet Routing Node Hello Message

Used for initialization and periodic monitoring of routers on an Ethernet circuit.

Sent from routers to other routers and to end nodes. Used in selecting **Designated Router** on the Ethernet.

D. Ethernet End Node Hello Message

Used for initialization and periodic monitoring of end nodes on an Ethernet circuit.

Sent from end nodes to routers on the Ethernet.

E. Hello & Test

Tests an adjacent node to determine if an adjacency is operational.

F. Initialization

G. Verification

NOTE

Routing layer message formats are detailed in the Supplemental Listings book.

7.4 Routing Layer Initialization Examples

7.4.1 Routing Node to Routing Node on Nonbroadcast Circuit

Phase IV Routing Node A	Phase IV Routing Node B
Routing Init. Message	
	Routing Init Message
(Optional Verification Message)	
	(Optional Verification Message)
Segmented Routing Message	
	Segmented Routing Message

(May exchange more than 1 Segmented Routing Message)

Routing version etc.

set circ verification ena

on Node A: Set node B

trans pw —— goes to node B

Receive pw —— get from node B

which must ma[tch] recv passwd on B's data on

Routing tells Data Link drop Physical DL tells Physical drop carr

Dynamic Async must have info about all nodes in database ie recv passwd that will dial into you

7.4.2 Routing Node (Node A) Coming Up on the Ethernet

1. Node A multicasts the **Router Hello Message** to all Routers.

2. The routers update their databases to include Routing Node A.

   Each router also tries to work out who should be the **Designated Router.** The decision is based on the router priority from the **Router Hello Message.** The router with the highest priority will be the Designated Router. The highest node address breaks ties.

   The Designated Router periodically multicasts the Router Hello Message to all end nodes. Node A may or may not be the Designated Router.

3. Node A learns about the end nodes from the End Node Hello Messages and about the routing nodes from the Router Hello Messages.

4. Due to resource constraints, there should only be a limited number of routers on an Ethernet.

7.4.3 Ethernet End Node Support

- End nodes do not listen to routing messages
- No network state kept
- On non-Ethernet circuits, end nodes must consult with the router to find out if a node is reachable and must route all traffic through the router
- On Ethernet, end nodes perform end node caching

NOTE

The result of this cache is that the first message to a remote node must be routed through the Designated Router; all subsequent messages are sent directly to the node over the Ethernet.

7.4.4 Nonrouting Node (Node X) Coming Up on Ethernet

1. Node X multicasts the **End Node Hello Message** to all routers. This is repeated periodically to prevent corruption of the routing database due to messages loss over the NI.

2. The routers update their databases to include the nonrouting Node X.

   The routers multicast the **Router Hello Messages** periodically to all routers.

   The **Designated Router** (the one with highest router priority or highest address) additionally multicasts the Router Hello Message to all nonrouting or end nodes.

3. The Nonrouting Node X stores the information about this Designated Router.

4. When the nonrouting Node X wants to communicate with another Node Y, it does the following:

   a. Checks if information about Node Y is already in **cache.** If so, Node X addresses Node Y directly (Concept of node adjacency).

   b. If no information is in cache, Node X addresses the Designated Router to route the message to Node Y.

   c. If there is no Designated Router, Node X addresses Node Y directly.

5. On Node X, if a packet is received from Node Z with the **on-Ethernet** bit set within the routing header, Node Z's address can be stored in **cache** and packets can be sent to Node Z directly without going through the Designated Router.

8 END-TO-END COMMUNICATIONS LAYER

8.1 Functions of the End-to-End Communictions Layer

- Handles the Operating System Independent aspects of communications
- Creates, maintains, and destroys logical links
- Manages the movement of normal and interrupt data
- Breaks up user messages into segments
- Guarantees the delivery of data and control messages to a specified destination by means of an error control mechanism
- Some key concepts to know are:
  - Logical link
  - Interrupt and normal data
  - Message sequencing and acknowledgment

- Flow control

  Mechanism to determine when to send interrupt and normal data.

  During logical link formation, the NSP at each end of the link determines the kind of flow control it expects when acting as a data receiver. The term "data-receiving NSP" means an NSP acting as a data receiver.

  The types of flow control available to NSP are:

  1. NO FLOW CONTROL

  2. SEGMENT FLOW CONTROL (REQUEST COUNT)

  3. MESSAGE FLOW CONTROL

NOTE

1. **When NO FLOW CONTROL is specified, DECnet uses an ON/OFF control mechanism.**

2. **ON/OFF may also be used with SEGMENT FLOW CONTROL.**

3. **MESSAGE FLOW CONTROL is being phased out.**

4. **NSP flow control mechanisms are detailed in Module 6, Section 2.**

8.2 ECL - Network Services Protocol (NSP) Message Types

A. DATA SEGMENT

Carries a portion of a Session Control message

B. INTERRUPT DATA (OTHER DATA)

Carries urgent data, originating from higher DNA layers, and optionally a Data Segment acknowledgment

C. DATA REQUEST

Carries data flow control information, and optionally a Data Segment acknowledgment

out of band AST's ctrl Y

D. INTERRUPT REQUEST

Carries interrupt flow control information, and optionally a Data Segment acknowledgment

E. DATA ACKNOWLEDGMENT

Acknowledges receipt of either a Connect Confirm or one or more Data Segment messages, and optionally another interrupt message

F. OTHER DATA ACKNOWLEDGMENT

Acknowledges receipt of either one or more Interrupt, Data Request or Interrupt Request messages, and optionally a Data Segment message

G. CONNECT ACKNOWLEDGMENT

Acknowledges receipt of a Connect Initiate message

H. CONNECT INITIATE

Carries a logical link connect request from a Session Control Module

I. CONNECT CONFIRM

Carries a logical link connect acceptance from a Session Control Module

J. DISCONNECT INITIATE

Carries a logical link connect rejection or disconnect request from a Session Control Module

K. NO RESOURCE (DISCONNECT CONFIRM)

Sent when a Connect Initiate message is received and there are no resources to establish a new logical link

L. DISCONNECT COMPLETE (DISCONNECT CONFIRM)

Acknowledges the receipt of a Disconnect Initiate message

M. NO LINK (DISCONNECT CONFIRM)

Sent when a message is received for a nonexistent logical link

N. NO OPERATION

Does nothing

NOTE

NSP message formats are detailed in the Supplemental Listings book.

9 SESSION CONTROL LAYER

9.1 Functions of the Session Control Layer

- Acts together with the end communication layer to create and maintain logical links
- Performs the Operating System Dependent functions:
  - Maps node names to node addresses
  - Identifies end-user tasks using an operating system dependent algorithm
  - Creates or activates processes
  - Validates incoming connect requests
  - Sends and receives logical link requests
  - Disconnects and aborts logical links

NOTE

For logical link requests, disconnects and aborts:

- **When a request comes from an end-user process, it is passed directly to ECL.**
- **When a notification comes from ECL, it is passed directly to the end-user process.**

9.2 Session Control Protocol Message Types

No separate protocol header is added for session control messages.

Session control uses the CONNECT INITIATE and DISCONNECT message fields of the end communication messages.

A. CONNECT data

B. REJECT data

C. DISCONNECT data

NOTE

Session control message formats are detailed in the Supplemental Listings book.

10 NETWORK APPLICATIONS LAYER

10.1 Data Access Protocol (DAP)

- Uses local RMS
- Communicates with remote FAL
- Retrieves input files and creates output files
- Provides file transportability between nodes
- Provides error recovery
- Allows multiple data streams over a logical link
- Provides command file execution and submission
- Provides random access of records in a file
- Provides file deletion
- Provides directory listings

10.1.1 DAP Message Types

A. Configuration

B. Attributes

C. Access

D. Control

E. Continue-Transfer

F. Acknowledge

G. Access Complete

H. Data

I. Status

J. Key Definition Attributes Extension

K. Allocation Attributes Extension

L. Summary Attributes Extension

M. Date and Time Attributes Extension

N. Protection Attributes Extension

O. Name

```
SOURCE NODE                 MESSAGES                  TARGET NODE
===========                 ========                  ===========

CONFIGURATION  INFO     CONFIGURATION
(Buffer Size, OS, File  MESSAGE =======>
System, DECnet/DAP
Version)                  CONFIGURATION               CONFIGURATION INFO
                       <======= MESSAGE               RETURNED

FILE CHARACTERISITCS
(File Type, Record      ATTRIBUTE
Size & Attributes)      MESSAGE =======>

Access Request          ACCESS MESSAGE =======>
                                                      ACTUAL FILE
                              ATTRIBUTE               CHARACTERISTICS
                       <======= MESSAGE               RETURNED

                        ACKNOWLEDGEMENT               FILE OPENED
                       <======= MESSAGE

SET UP DATA STREAM      CONTROL (Initiate Data
                        Stream) MESSAGE =======>


                        ACKNOWLEDGEMENT               DATA STREAM
                       <======= MESSAGE               ESTABLISHED

REQUEST START OF        CONTROL (Get)
DATA TRANSFER           MESSAGE         =======>

                               | Record 1
                               |     .
                       <=======|     .                DATA SENT IN
                               |     .                RECORDS
                               | Record n

                       <======= STATUS MESSAGE        EOF DETECTED

REQUEST TO TERMINATE    ACCESS COMPLETE
FILE TRANSFER           MESSAGE         =======>

                        ACCESS COMPLETE               REQUEST COMPLETED
                       <======= RESPONSE              SUCCESSFULLY
```

Example 2-6 DAP Message Exchange (Sequential File Access)

10.2 Other Network Applications Protocols

1. Command Terminal (CTERM)

   SET HOST

2. SNA (Systems Network Architecture)

   Access to IBM SNA Networks

   SNA Capabilities

   - DECnet/SNA Remote Job Entry (RJE)
   - DECnet/SNA 3270 Terminal Emulator (3270 TE)
   - DECnet/SNA Distributed Host Command Facility (DHCF)
   - DECnet/SNA DISOSS Document Exchange Facility (DDXF)
   - DECnet/SNA Printer Emulator (PrE)
   - DECnet/SNA Application Interface (AI)
   - External Document Exchange (EDE) with IBM DISOSS

     Access to common DISOSS/370 database

     Complete access to Document Content Architecture (DCA) documents

     Edit and conversion utilities

     Accessible through ALL-IN-1, WPS-PLUS or DECmates

3. PSI-ACCESS - Access to Packet-Switching Data Network

4. Loopback Mirror Protocol

   - Handles node loopback functions

   - Message types

     A. Command

        Request a loop test and send the data to be looped

     B. RESPONSE

        Return status information and the loopedback data

11 NETWORK MANAGEMENT LAYER

- Provides facility to oversee, test, monitor, and control the network
- Allows an end-user to access local and remote databases, images, etc. for various network management functions using NCP (Network Control Program)
- Has direct access to each of the lower layers for control purposes
- Functions include:
  - Examining the configuration status
  - Examining the network parameters
  - Changing the parameters (such as those for node, line, circuit, logging)
  - Changing the network configuration and modifying message traffic patterns
  - Downline loading remote systems
  - Upline dumping of remote systems
  - Examining performance variables
  - Providing loopback tests for fault isolation
- Supports Multiple Protocols:
  - NICE - Network Information and Control Exchange
  - MOP - Maintenance Operations Protocol
  - Event Logging Protocol

11.1 NICE Protocol

- Handles most functions of the network management layer
- NML (Network Management Listener) uses NICE protocol to monitor status and activity of remote nodes
- Request and Response Message Types:

  A. Request downline load

  B. Request upline dump

  C. Trigger bootstrap

  D. Test

  E. Change parameter

  F. Read information

  G. Zero counters

  H. System-specific functions

  I. NICE response message

NOTE

NICE Message Formats are detailed in the Supplemental Listings book.

11.2 Maintenance Operation Protocol (MOP) Functions

- Functions do not require the services of any layers other than the network management and data link layers
- Downline loading
- Upline dumping
- Loopback testing
- Force entry into MOP mode
- Passes transfer addresses so that programs resident in memory will execute
- System console control for remote and/or unattended nodes

1. The target node transmits a REQUEST PROGRAM message to the LOAD ASSISTANCE multicast address (AB-00-00-01-00-00) - i.e., a request for any host.

2. The hosts check their own databases to see if they can downline load the target.

3. If so, they send the secondary bootstrap code. This message is addressed to the 48-bit address in the header of the REQUEST PROGRAM message.

4. The first host to respond is **chosen** by the target to continue the loading sequence. No message is returned to any other host.

5. The loading sequence continues with the tertiary loader. The tertiary loader will be sent in separate messages with a maximum of 1500 bytes per message (Ethernet limitation).

6. The tertiary loader in the target then requests the host for the operating system. The operating system will be sent in separate messages. The last segment is followed by a PARAMETER LOAD WITH TRANSFER ADDRESS message, which sets up extra values for the node identification and the host identification, in addition to the start address of the image just loaded.

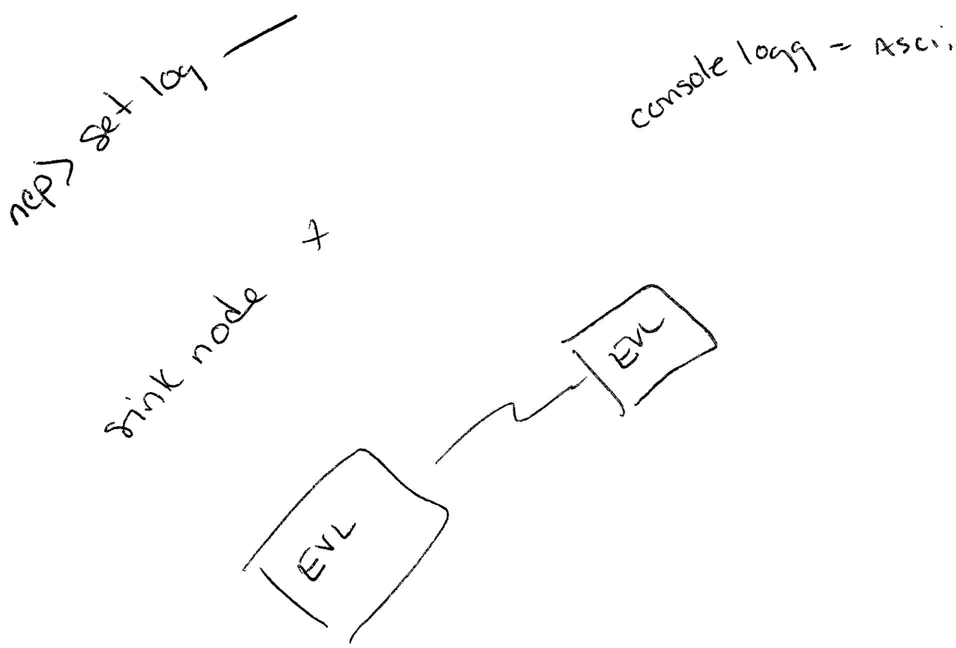
Example 2-7 Downline Loading on Ethernet

```
DECnet event 0.3, automatic line service
From node 1.1 (CANADA), 04-JUL-1986 17:37:07.46
Circuit UNA-0, Load, Requested, Node = 1.9 (SNAP)
File = PLUTO2.SYS, Secondary loader,
Ethernet address = 08-00-2B-02-8F-B1


DECnet event 0.3, automatic line service
From node 1.1 (CANADA), 04-JUL-1986 17:37:07.56
Circuit UNA-0, Load, Successful, Node = 1.9 (SNAP)
File = PLUTO2.SYS, Secondary loader,
Ethernet address = 08-00-2B-02-8F-B1


DECnet event 0.3, automatic line service
From node 1.1 (CANADA), 04-JUL-1986 17:37:07.66
Circuit UNA-0, Load, Requested, Node = 1.9 (SNAP)
File = PLUTO3.SYS, Tertiary loader,
Ethernet address = 08-00-2B-02-8F-B1


DECnet event 0.3, automatic line service
From node 1.1 (CANADA), 04-JUL-1986 17:37:08.99
Circuit UNA-0, Load, Successful, Node = 1.9 (SNAP)
File = PLUTO3.SYS, Tertiary loader,
Ethernet address = 08-00-2B-02-8F-B1


DECnet event 0.3, automatic line service
From node 1.1 (CANADA), 04-JUL-1986 17:37:09.10
Circuit UNA-0, Load, Requested, Node = 1.9 (SNAP)
File = SYS$SYSROOT:[PLUTO]SNAPTSV.SYS, Operating system
Ethernet address = 08-00-2B-02-8F-B1


DECnet event 0.3, automatic line service
From node 1.1 (CANADA), 04-JUL-1986 17:37:15.21
Circuit UNA-0, Load, Successful, Node = 1.9 (SNAP)
File = SYS$SYSROOT:[PLUTO]SNAPTSV.SYS, Operating system
Ethernet address = 08-00-2B-02-8F-B1
```

Example 2-8 EVL Messages for Downline Load

11.3 Event Logger Protocol

- Records significant events
- Event logs help monitor network activity and problems
- Uses only one message - the event message
- Provides information about:
  - The node at which the event is to be logged
  - Where event is to be logged (console, file, monitor)
  - Event type and class
  - Date and time
  - Name and address of source node
  - Whether the event relates to a DECnet module, node, circuit, or line
  - Specific data concerning the event



12 THE USER LAYER

- The highest layer of DNA
- Supports user services and programs
- The user interface for DECnet-VAX is very simple

  $ TYPE node::disk:[directory]filename.type

- The user decides on the protocol used
  - The sequence of reads and writes in a task-to-task communication

DECnet-VAX SOFTWARE COMPONENTS

INTRODUCTION

This chapter discusses the software components of DECnet-VAX.

Topics include:

- Data Link Device Drivers
  - XMDRIVER
  - XDDRIVER
  - XGDRIVER
  - ETDRIVER
  - XEDRIVER
  - XQDRIVER
  - CNDRIVER
  - NODRIVER
  - NWDRIVER
- NETDRIVER
- NETACP
- RMS, DAP Routines, and FAL_n
- RTTDRIVER, REMACP, and RTPAD
- Special Processes
  - NETSERVER
  - MOM
- Objects
  - FAL
  - NML
  - EVL
  - MIRROR
  - DTR
  - MAIL
  - PHONE
  - HLD
- NDDRIVER
- Other DECnet Components
  - Permanent configuration database
  - Volatile configuration database
  - NCP
  - SYS$MANAGER:STARTNET.COM

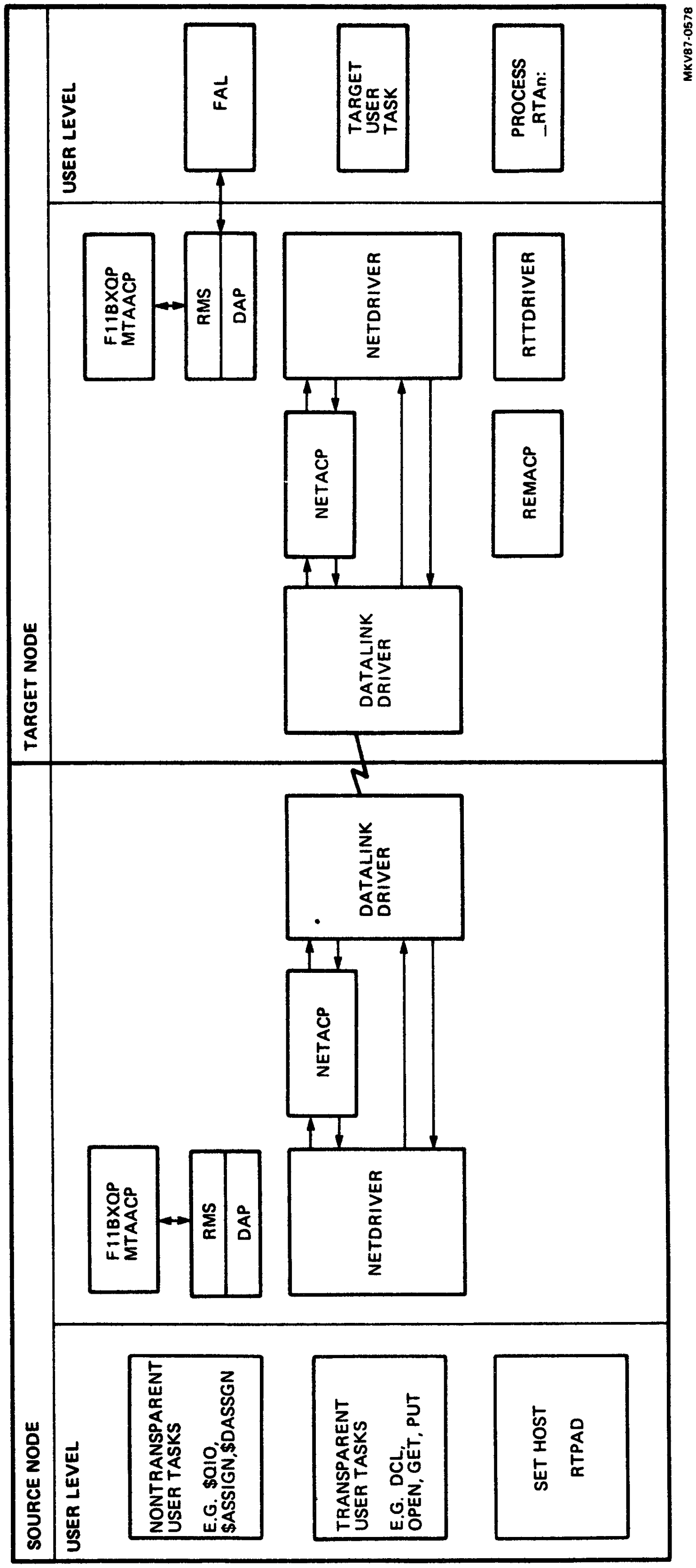


Figure 3-1 Block Diagram of DECnet-VAX

1 DATA LINK DEVICE DRIVERS

1.1 XMDRIVER

Device driver for the DDCMP microcode devices such as DMC11 or DMR11. The same driver is used for both the DMC and the DMR.

1.2 XDDRIVER

Device driver for the DMP11 interface. The DMP11 contains DDCMP microcode to handle multipoint.

1.3 XGDRIVER

Device driver for the DMF32 synchronous port interface. Uses software implementation of DDCMP.

1.4 ETDRIVER

Device driver for DIGITAL Ethernet BI Network Adapter (DEBNT).

1.5 XEDRIVER

interrupt & start I/O routines are differently

Device driver for DIGITAL Ethernet UNIBUS Adapter (DEUNA and DELUA).

The Ethernet data link is done in the microcode of the controller (DEUNA, DELUA).

1.6 XQDRIVER

Device driver for DIGITAL Ethernet Q-Bus Adapter (DEQNA).

1.7 CNDRIVER

Device driver for Computer Interconnect (CI). This driver is similar to the DMP11 driver and is used in local, multipoint, and high-speed environments.

1.8 NWDRIVER - For X.25 (Used for Datalink Mapping)

Adds layer of overhead

make VC's look like circuits.

sits over PSIDRIVER

1.9 NODRIVER

Device driver for asynchronous DECnet (DDCMP protocol). This driver can be used with any VMS-supported terminal driver.

2 NETDRIVER

- Implements routing, end communications, and session control layers
- Handles the time-critical code

  (Data transfers, mailbox message delivery, interrupt message handling, protocol, ACKs/NAKs, retransmissions)
- Handles the network QIO interface
- Passes control to NETACP to handle nontime-critical code or code that requires process context
  - Information passed by means of its mailbox or the ACP Queue Blocks (AQB)
  - NETACP is awakened to handle the request
  - NETACP may requeue the IRPs back to NETDRIVER and call its routines directly
- Has standard VAX/VMS device driver format and data structures
- The hardware interrupt code is missing
- The interface from NETDRIVER to the DATA LINK drivers is by means of internal QIOs

3 NETACP

- Implements routing, end communications, and session control layers

- Handles the nontime-critical code and code that requires process context for the network

  Connects, disconnects, line and circuit state transitions, volatile database maintenance, and routing updates

- Sets up the routing database in nonpaged pool (so NETDRIVER can access it)

  The volatile database is kept in paged memory

- Implemented as a process

- Hibernates when there is nothing to do

- Can be awakened by a mailbox AST or by a wake up request from the VMS executive

- IT scans one of its three queues:

  1. Timer Queue -- containing routing timers, inactivity timers

  2. Work Queue -- containing work queue elements from NETDRIVER

  3. CP Queue (AQB) -- containing IRPs for connect initiate

- If there is something to do, it dispatches to one of the processing routines and returns to the dispatch or hibernate loop when done

4 RMS, DAP ROUTINES, AND FAL_N

- Implement application layer for file transfer operations
- For normal user tasks requesting access to a file (i.e., OPEN, TYPE, DIRECTORY , etc.), the request will be passed to RMS
- When RMS detects the network file specifier ("::"), it knows that remote node access is needed, and sets up the appropriate $QIO
- The $QIO specifies the remote task and requests that the logical link be set up on behalf of the user task
- The interface between RMS and NETDRIVER is by means of Network QIOs

  (RMS is simply treated as any other DECnet user)

5 RTTDRIVER, REMACP, AND RTPAD

- Implement application layer for remote terminal access (SET HOST)
- Uses the command terminal (CTERM) protocol

6 SPECIAL PROCESSES

6.1 NETSERVER

- The mechanism used to start a network user process on a remote node and select the task to execute
- Allows single process to service multiple logical links
- Keeps SERVER_n processes around (for NETSERVER$TIMEOUT) waiting for inbound connect requests
- Created by NETACP and receives logical link requests by means of special QIO interface
- Selects the task to execute VIA LIB$DO_COMMAND OR LIB$RUN_PROGRAM

6.2 MOM

- Maintenance operation modules used for maintenance operations such as downline load, upline dump, and loopback test
- NML spawns a MOM process which interacts with the NDDRIVER, or an incoming request can cause NETACP to create a MOM process

7 OBJECTS

7.1 FAL

- File Access Listener for remote file access

7.2 NML

- Network management listener that manipulates permanent and volatile (by means of NETACP) configuration databases

7.3 EVL

- Event logger that collects network events from nonpaged pool, filters and passes them to the correct destination

7.4 MIRROR

- Loopback mirror program used in loopback tests

7.5 DTR

- Data receiver that interacts with the DTSEND program to check out and provide statistics on task-to-task communication

7.6 MAIL

- VAX/VMS Electronic mail utility

7.7 PHONE

- VAX/VMS phone utility

7.8 HLD

- Host task loader that interacts with the satellite task loader (sld) to downline load RSX-11S tasks

8 NDDRIVER

- This driver handles the direct line access functions on behalf of MOM process
- Passes control to NETACP for some non-time-critical code
- For Nonbroadcast Circuits:
  - The data link driver such as XMDRIVER passes control to NETDRIVER when a packet has been received
  - If the packet happens to be a MOP request function, NETDRIVER informs NETACP
  - The outstanding I/O on that UCB is aborted and NETACP gives control of the UCB to the NDDRIVER
- For Broadcast Circuits (e.g., Ethernet):
  - The data link driver (XEDRIVER) has multiple UCBs that are distinguished by their protocol type. The driver checks the protocol type of the packet.
  - If the protocol type is DECnet (60-03), the packet is passed to NETDRIVER.
  - If the protocol type is MOP (60-01), the UCB is set up for maintenance operation.
  - For a new request, NETACP is activated and creates a MOM process to handle this new request.
  - The MOM process uses the NDDRIVER to handle the request.

9 OTHER DECnet COMPONENTS

9.1 Permanent Configuration Database

- Files that define the network as known to the local node
  - SYS$SYSTEM:NETCIRC.DAT
  - SYS$SYSTEM:NETCONF.DAT
  - SYS$SYSTEM:NETLINE.DAT
  - SYS$SYSTEM:NETLOGING.DAT
  - SYS$SYSTEM:NETOBJECT.DAT
  - SYS$SYSTEM:NETNODE_LOCAL.DAT
  - SYS$SYSTEM:NETNODE_REMOTE.DAT

sys$common
exec params
remote

- Provides initial values for network parameters
- Loaded to volatile database in memory when the network is started
- Initially configured with SYS$MANAGER:NETCONFIG.COM
- Modify with the NCP commands **DEFINE|LIST|PURGE**

9.2 Volatile Configuration Database

- Resides in memory in NETACP process space
- Contains the specific parameters used by layers of DECnet
- Modify with the NCP commands **SET|SHOW|CLEAR|ZERO**

9.3 NCP

- Network Control Program that interacts with NML to control and monitor the network

9.4 SYS$MANAGER:STARTNET.COM

- Command procedure to start the network
- Calls LOADNET.COM to load NETDRIVER and start the NETACP process
- comment out & won't allow remote logins
- Calls RTTLOAD.COM to load RTTDRIVER and run the REMACP process if disable remote logins will still have memory.
- Calls STARTPSI.COM to configure PSI
- Configures the basic volatile database
- Sets up the node database (NCP> SET KNOWN NODES ALL)

DECnet-VAX DATA STRUCTURES

INTRODUCTION

This chapter reviews DECnet data structures and their format.

Topics include:

- Overall Data Structure Linkage
- Major VMS Device Driver Data Structures Used by DECnet
  - Unit Control Block (UCB)
  - Device Data Block (DDB)
  - Channel Request Block (CRB)
  - Interrupt Data Block (IDB)
  - Adapter Control Block (ADP)
  - Channel Control Block (CCB)
  - I/O Request Packet (IRP)
- DECnet-Specific Data Structures
  - Routing Control Block (RCB)
  - Output Adjacency (OA) and Area Output Adjacency (AOA)
  - Adjacency Node Database Block (ADJ)
  - Logical Path Descriptor (LPD)
  - Internal Connect Block (ICB)
  - Logical Link Subchannel Block (LSB)
  - Configuration Data Root Block (CNR) and Data Block (CNF)
  - Network Window Block (XWB)
  - Remote Node (NDI) and Local Node Information (LNI)
  - Network Server Process Information (SPI)
  - Work Queue Elements (WQE)
  - Node Counter Block (NDC)
  - Event Logger Data Structures
  - Object Information Block (OBI)
  - Circuit (CRI) and Physical Line Information (PLI)

wrong for looking thru SDA

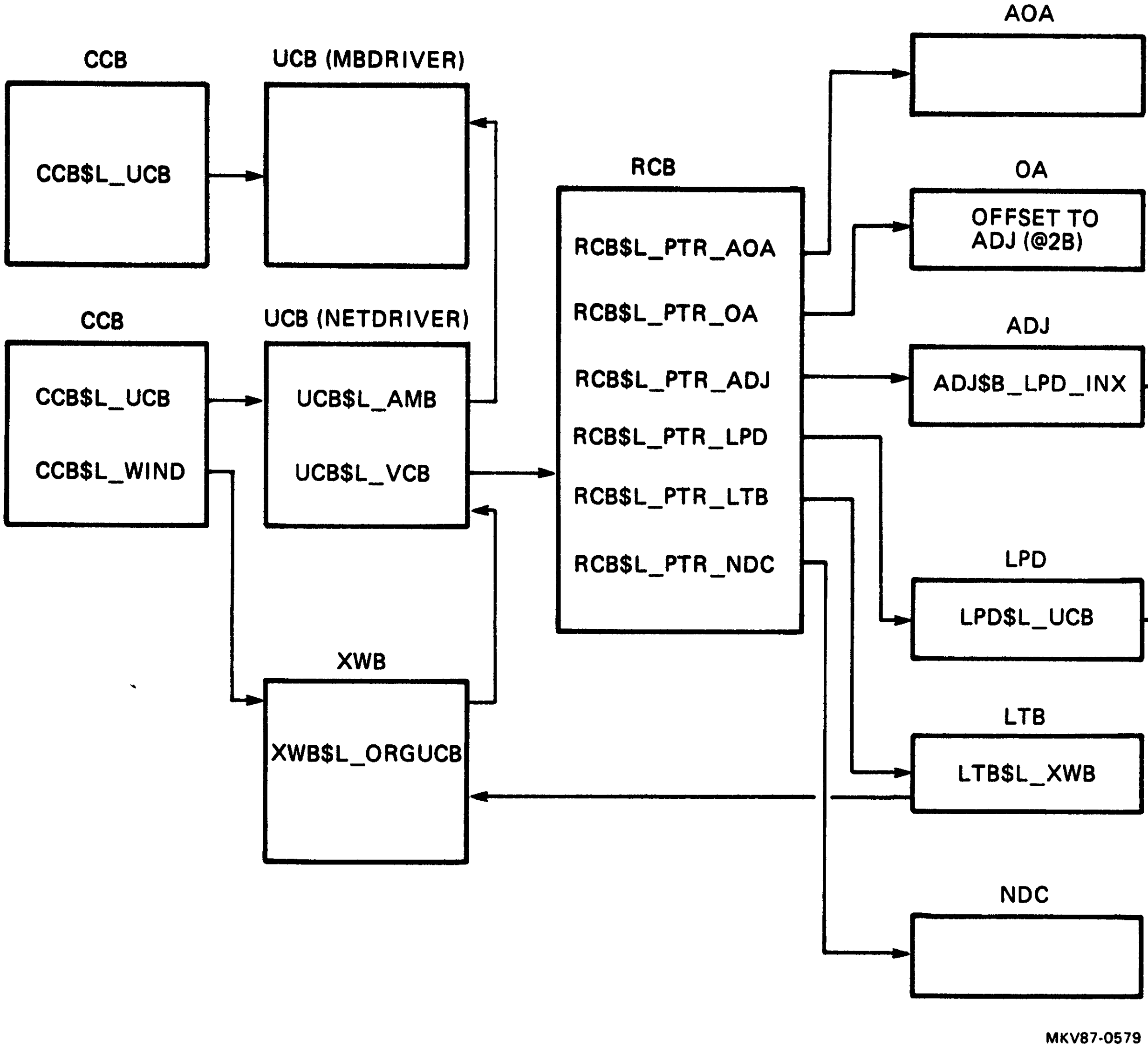


Figure 4-1 Overall Data Structure Linkage

1 MAJOR VMS DATA STRUCTURES USED BY DECnet

The interface of DECnet to VAX is by means of the QIO. Therefore, data structures in VMS relating to QIO processing will be involved. Their formats are defined in SYSDEF.STB. The main VMS IO data structures are the Unit Control Block (UCB), Device Data Block (DDB), Channel Request Block (CRB), Interrupt Data Block (IDB), Adapter Control Block (ADP), Channel Control Block (CCB), and I/O Request Packet (IRP).

1.1 Unit Control Block (UCB)

- Describes characteristics and current state of a specific unit
- The UCB is linked to the rest of the I/O data structures and contains information such as listheads for the pending I/O request packets
- Contains the fork block of the fork process executing code to perform I/O on this unit
- **The UCB, together with the network window block (XWB), can be considered the focal point of the I/O database for DECnet**
- UCBs of interest to DECnet are those for NETDRIVER, the datalink drivers, and MBDRIVER

look at files SDL for comment on what each field is for

1.1.1 UCB Fields

- UCB$K_Length - Length of standard UCB
- UCB$L_FQFL - Fork queue forward link
- UCB$L_FQBL - Fork queue backward link
- UCB$L_RQBL - NET -- RCV queue backward link
- UCB$W_SIZE - Size of UCB in bytes
- UCB$BTYPE - Structure type for UCB
- UCB$B_FIPL - Fork interrupt priority level
- UCB$T_PARTNER - NET -- Partner's nodename
- UCB$L_FIRST - NET -- ADDR of first seg of chained MSG
- UCB$W_BUFQUO - Buffered I/O quota charged for this UCB
- UCB$W_DSTADDR - NET -- remote connect number
- UCB$W_SRCADDR - NET -- local connect number
- UCB$L_ORB - Object's rights block address
- UCB$L_LOCKID - Device lock ID
- UCB$L_CPID - PID charged for BUFQUO by UCBCREDEL
- UCB$L_CRB - Address of primary channel request block
- UCB$L_DDB - Pointer to device data block
- UCB$L_PID - Process ID of owner process
- UCB$L_LINK - Address of next UCB for respective DDB
- UCB$L_VCB - Address of volume control block
- UCB$B_DEVCLASS - Device class
- UCB$B_DEVTYPE - Device type
- UCB$W_DEVBUFSIZ - Device default buffer size

- UCB$S_NET_DEVDEPEND - Network fields

  UCB$B_LOCSRV - Local link services
  UCB$B_REMSRV - Remote link services
  UCB$W_BYTESTOGO - No. of bytes left in rcv bfr

- UCB$W_RWAITCNT - Class Drivers -- threads waiting resources

  UCB$B_CM1 - Level 1 controller allocation mask
  UCB$B_CM2 - Level 2 controller allocation mask
  UCB$L_IRP - Current I/O request packet address
  UCB$W_REFC - Reference count of processes
  UCB$B_DIPL - Device interrupt priority level

- UCB$B_STATE - NET -- link state for network transitions

- UCB$L_AMB - Associated unit control block pointer (Pointer to MAILBOX for NETWORK UCB)

- UCB$L_STS - Device unit status

  Timeout enabled
  Unit timed out
  Interrupt expected
  Cancel I/O on unit
  Set if this is template UCB

  . . . .

  UCB is busy
  Too many bytes rcvd
  Link has declared a connect name
  Link is being broken

- UCB$W_QLEN - Device queue length

- UCB$L_SVPN - System virtual page/map register number

- UCB$W_BCNT - Byte count of transfer

- UCB$L_PDT - ADDR of port descriptor table

  . . . .

1.1.2 Device-Dependent UCB Extensions

- Network Logical Link (Network Mailbox) Extension

```
UCB$C_LOGLNK -     Connect is for logical link
UCB$L_NT_DATSSB -  ADDR of data subchannel status block
UCB$L_NT_INTSSB -  ADDR of interrupt subchannel status
                   block
UCB$W_NT_CHAN -    DDCMP channel number
UCB$V_LTYPE -      Link type bits
UCB$V_SEGFLO -     Segment request counts
UCB$V_MSGFLO -     Message request counts
UCB$V_MSGACK -     Message ACK/NAK
UCB$V_BACKP -      Backpressure
UCB$V_LNKPRI -     Link priority (Ignored)
```

- NI Device Extension

```
UCB$C_NI_LENGTH - Size of NI Device UCB
UCB$L_NI_HWAPTR - Address of NI device hardware
                  address
UCB$L_NI_MLTPTR - Address of protocol multicast table
```

1.2 Driver Prologue Table (DPT)

- Describes driver size and device type to driver loader
- Contains initial data for other data structures

1.3 Device Data Block (DDB)

- Describes device type, generic device name, controller designation and driver name
- Points to first entry of UCB

1.4 Channel Request Block (CRB)

hardware ie netdriver doesn't have them

- Describes the characteristics and current state of a specific controller
- Contains code to dispatch interrupts to the interrupt service routine
- **For NETDRIVER, there is no interrupt service routine**

1.5 Interrupt Data Block (IDB)

hardware

- Describes a specific controller
- Points to the controller's CSR and associated ADP
- **For NETDRIVER, there is no CSR and associated ADP**

1.6 Adapter Control Block (ADP)

hardware

- Describes the characteristics and current state of a specific adapter (i.e., MBA, UBA)
- **Not applicable to NETDRIVER**

1.7 Channel Control Block (CCB)

- Describes a device unit assigned to a software I/O channel

[redacted]

- Two CCBs of interest:

  - The one pointing to the Unit Control Block (UCB) of the mailbox driver (MBDRIVER).

    Unsolicited messages are handled by means of this CCB.

  - The one pointing to the UCB of the NETDRIVER.

    This one also points to the Extended Window Block (XWB). The normal data is handled by means of this CCB.

1.8 I/O Request Packets (IRP)

- Constructed by the $QIO system service

- Describes the I/O function to be performed

- This is the key data structure that "flows" through the I/O subsystem, moving from one list to another

- [redacted] differ from normal ones in that the longword containing a PID field inside the IRP is negative, and is the address of a processing routine in S0 space

1.8.1 Fields of the IRP

- IRP$L_PID - Process ID of requesting process
- IRP$L_AST/ASTPRM - Address/parameter of AST routine
- IRP$L_WIND - Address of window block
- IRP$L_UCB - Address of device UCB
- IRP$W_FUNC - I/O function code and modifiers
- IRP$B_EFN - Event flag number and event group
- IRP$B_PRI - Base priority of requesting process
- IRP$L_IOSB - Address of I/O status block
- IRP$W_CHAN - Process I/O channel number
- IRP$W_STS - Request status
- IRP$L_BCNT - Byte count of transfer
- IRP$S_NT_PRVMSK - Privilege mask for DECnet
- IRP$S_STATION - Station field for DECnet drivers
- IRP$L_ABCNT - Accumulated bytes transferred
- IRP$L_OBCNT - Original transfer byte count

2 DECnet DATA STRUCTURES

2.1 Routing Control Block (RCB)

The Routing Control Block (RCB) is used as the network Volume Control Block.

There is one RCB per system and it serves as the primary DECnet routing data structure. It is pointed to by the VCB fields of every UCB used by DECnet.

The RCB contains pointers to the following data structures which help it maintain the routing database:

- OA (Output Adjacency)
- AOA (Area Output Adjacency Node Database Block)
- ADJ (Adjacency)
- LPD (Logical Path Descriptor)
- LTB (Logical Link Table)
- NDC (Node Counter Block)

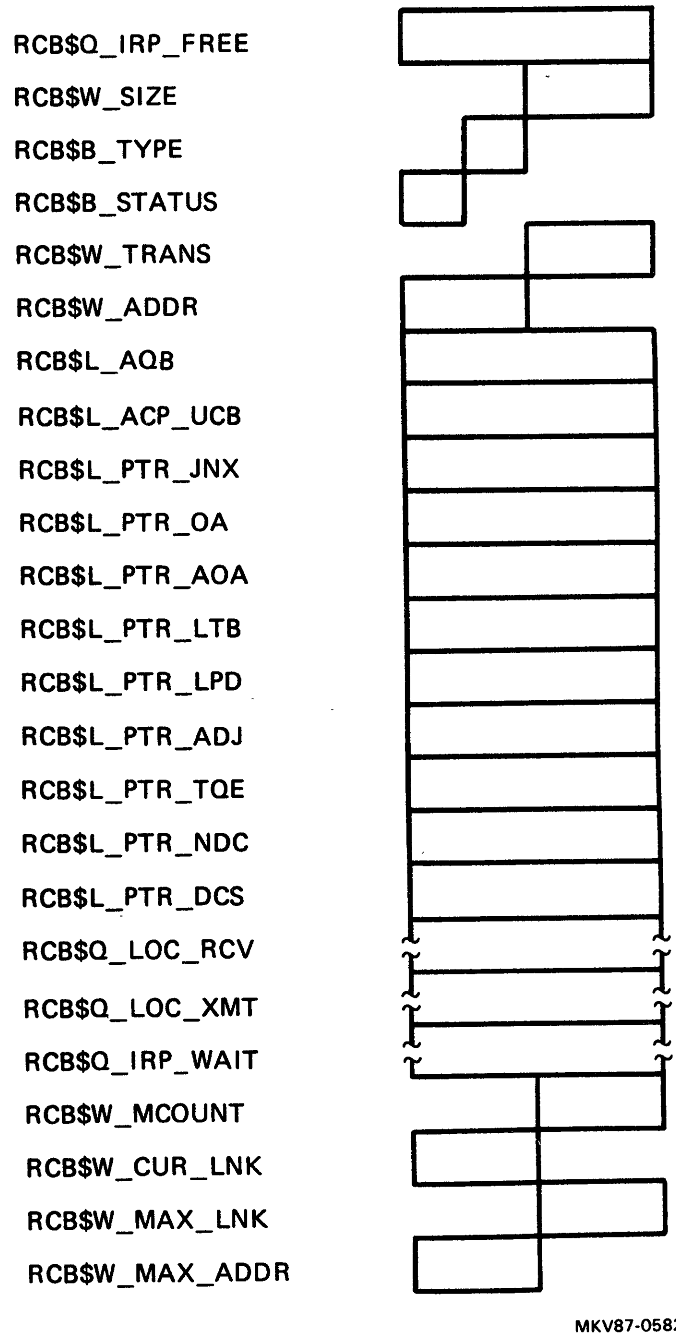


Figure 4-2 Routing Control Block (RCB)

2.2 Output Adjacency (OA)

- Maintained on routing nodes
- Each entry is 2 bytes long
- Number of entries corresponds to the maximum address
- Each entry offsets to the adjacency node database block (ADJ)
- Entry n corresponds to node address n
- If a node is unreachable, the value is zero

2.3 Area Output Adjacency (AOA)

- Maintained on Level II (area) routing nodes
- Used to determine the default adjacency to be used to get to a given area
- Each entry is 2 bytes long
- Number of entries corresponds to the maximum area
- The index is the area address, the vector cell contains the ADJ index
- A new AOA vector must be allocated whenever the maximum supported area address is increased

2.4 Adjacency Node Database Block (ADJ)

- Used together with the logical path descriptor (LPD) to describe the next hop destination to any reachable node in the network

- Stores information about the neighbor's node type, block size, listener interval (computed from neighbor's hello), etc.

- The number of entries in this data structure can be much less than the maximum address

  The value is given by

  MAXIMUM CIRCUIT +
  MAXIMUM BROADCAST NONROUTERS +
  MAXIMUM BROADCAST ROUTERS + 1

- Each entry is 13 bytes

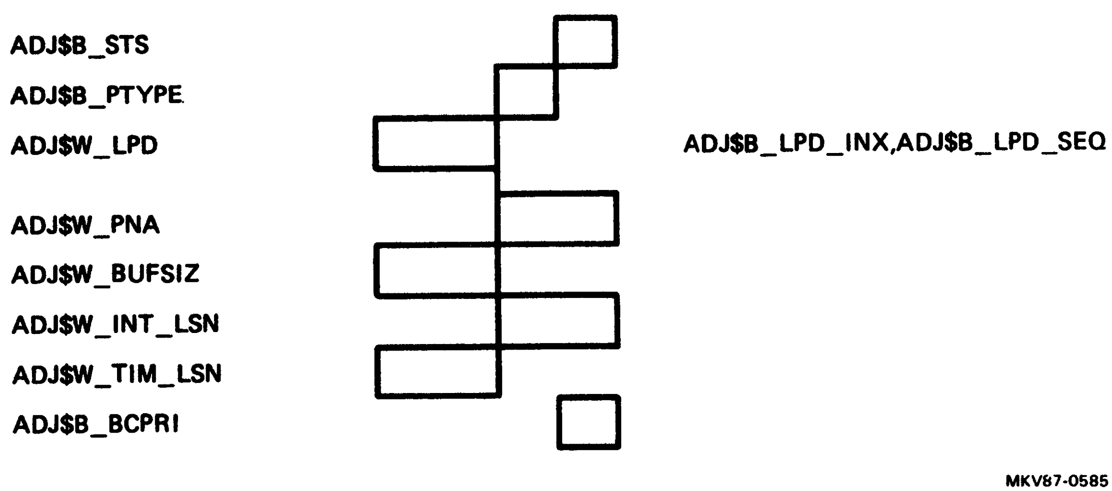


Figure 4-3 Adjacency Node Database Block (ADJ)

2.5 Logical Path Descriptor (LPD)

- Describes a path to a data link or a source
- For incoming messages, the LPD points to the End Communication Layer (ECL) driver (NETDRIVER) of the local node
- For outgoing messages, points to the UCB of the datalink driver (XMDRIVER)
- Allows messages to be transmitted over the correct line
- It also contains counters for packet sent or received on the circuit
- The number of LPD entries depends on the number of circuits
- Each LPD entry is 106 bytes
- On routing changes, the information in the ADJ changes to point to a new LPD

2.5.1 Fields in the LPD

- Number of data link start-up attempts since last "run"
- UCB address
- ACP channel to device
- "TALKER" timer
- "TALKER" interval (used to INIT TIMTLK)
- Number of test messages left to send before entering
- Number of outstanding IRPS queued by NETDRIVER
- Local node type on this circuit
- Output "square root limiter" value
- Output queue "input packet limiter"

- Path ID, index and sequence
- Status on control info
- XMIT flags
- X.25 PVC startup flags
- Circuit substate
- Circuit cost
- Circuit NI router priority/designated router on NI
- For broadcast circuits, LPDS, address of "most recently received election message" from router hello messages
- Transport layer counters
- Data-link buffer size including transport overhead
- Address of end node cache storage (end nodes only)

2.5.2 Logical Link Table (LTB)

- This structure is maintained by NSP (NETDRIVER)
- Contains all local End Communications Layer parameters and a vector of logical link slots
- The LTB has pointers back to the XWB for that logical link
- The first entry after the 12 byte header is a link pointer that points to the first XWB used
- Each slot in the vector is 4 bytes

  If the low bit is set, the slot is available and its sequence number (number of times used) is found in the high-order word.

  If the low bit is clear, the slot contains a pointer to the XWB with the link context and state information.

- The size of this data structure is governed by the executor parameter maximum links

  (HEADER + MAX LINKS * 4)

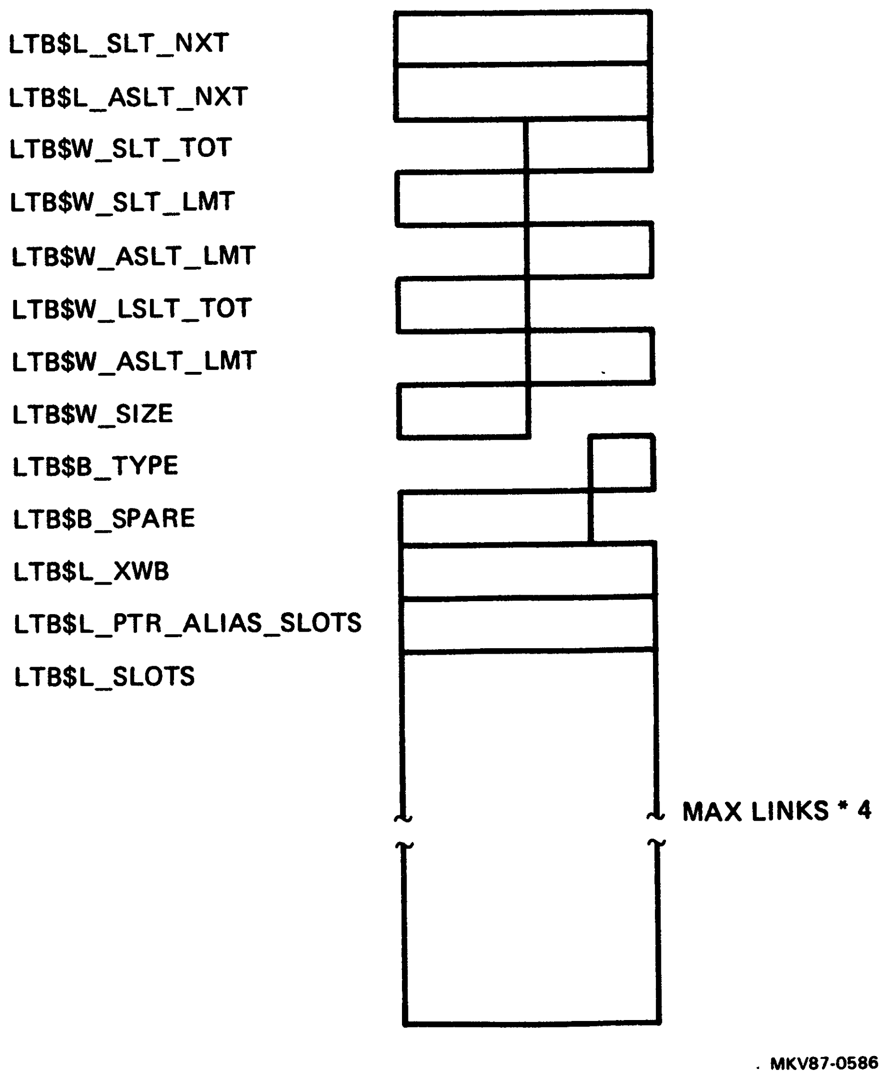


Figure 4-4 Logical Link Table (LTB)

2.6 Internal Connect Block (ICB)

This data structure is used to pass generic connect informatior between the Network ACP and an End Communications Layer (ECL) driver (for example, NETDRIVER).

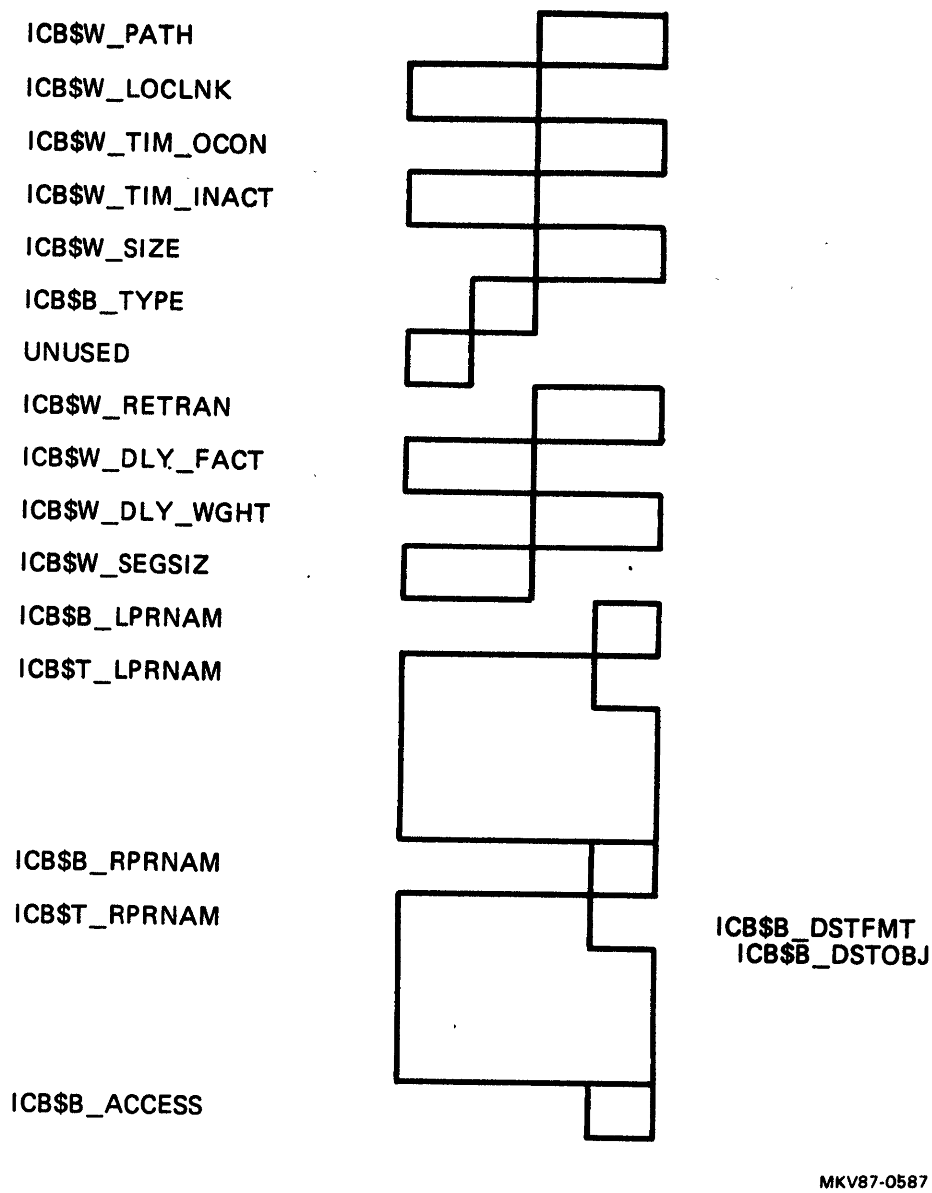


Figure 4-5 Internal Connect Block (ICB) (Sheet 1 of 2)

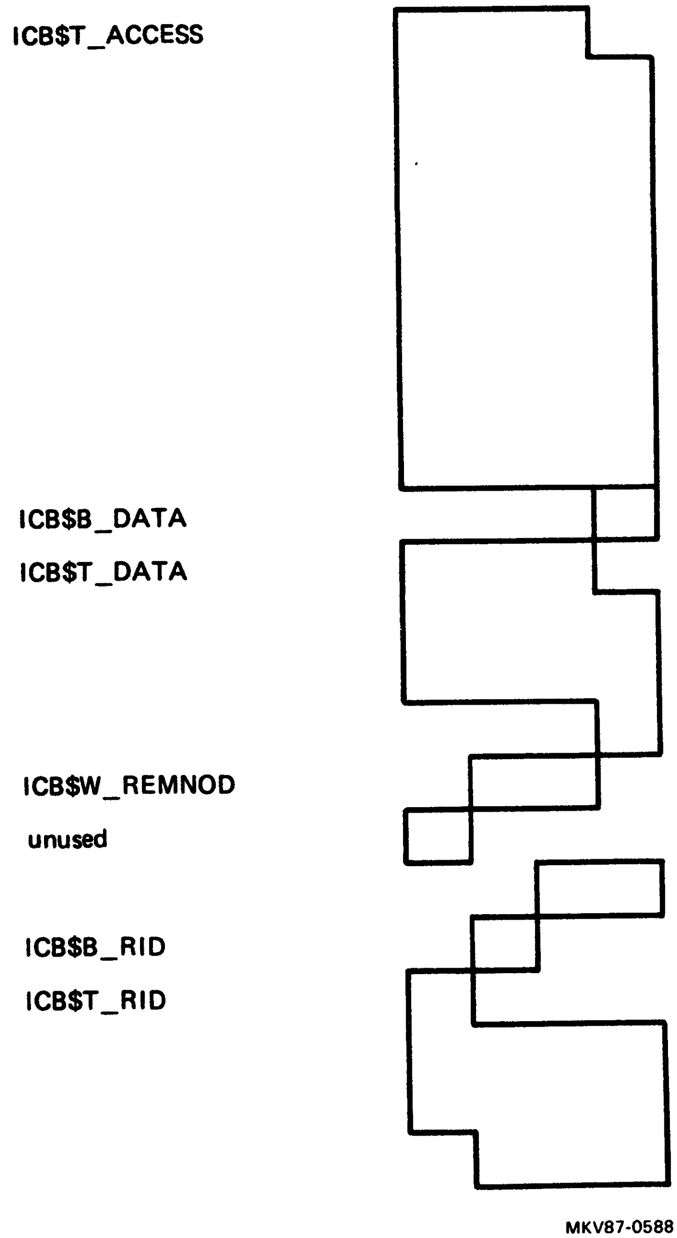


Figure 4-5 Internal Connect Block (ICB)
(Sheet 2 of 2)

2.7 Logical Link Subchannel Block (LSB)

This block is used to control the activity on a logical link subchannel. There are two subchannels: the DATA subchannel and the INTERRUPT/LINK subchannel. Every logical link has both subchannels.

2.7.1 LSB Fields

- Transmitter Control Variables

  Last segment number assigned to a segment
  Last segment number transmitted
  Highest segment number sendable
  Highest ACK number received
  Highest ACK number acceptable
  Flow control credits from remote receiver
  Packet window adjustment counter
  Size of the transmit-packet-window
  Number of active transmit CXBs
  Max total CXBs allowed
  Total CXBs both active on on the free queue
  Listhead for transmit IRPs containing data
  Listhead for transmit IRPs with data moved to CXBs
  Transmit CXB (message segments) listhead

- Receiver Control Variables

  Receive IRP listhead
  Received CXB (message segments) listhead
  Highest numbered message received and accepted
  Highest ACK transmitted
  Number of CXBs in LSB list (unACKed)
  Max rcv CXBs that can be buffered by NSP before some are passed to the Session Layer

- Miscellaneous

  Status bits
  Set for LS/INT subchannel
  Next segment to send has NSP$V_DATA_BOM set
  Next segment to send has NSP$V_DATA_EOM set
  Pointer to "cross-channel" LSB
  Length for use by XWB definition

2.8 Configuration Database Root Block (CNR)

This block serves as the listhead for the CNFs of a particular component in the configuration database. It contains all of the component's semantics.

2.9 Configuration Data Block (CNF)

This is a general block structure used to carry a sub-block in the configuration database of NETACP. The CNF and sub-block semantics for each component type are stored in the associate CNR.

2.10 Network Window Block (XWB)

- The XWB describes activity on a logical link
- Includes information such as flow control count, local and remote node logical link addresses
- There is one XWB per logical link
- This control block serves as the network window control block; as such its header section must look like a WCB
- The remainder of the structure is network-specific
- It contains the pointers to the data and interrupt subchannel block
- **This is the primary data structure describing logical links**

2.11 Remote Node Information (NDI)

This block has information that is commonly defined for all nodes known to the local node.

It includes information that is used to downline load a node that is not up on the network yet.

2.11.1 NDI Fields

- Information for Active Nodes

  Node address - zero if NDI is for local node
  Counter timer (units = sec)
  Absolute due timer for counters to be logged
  Node Name
  Privileged user id
  Privileged account
  Privileged password
  NonPrivileged user id
  NonPrivileged account
  NonPrivileged password
  Receive password
  Transmit password
  Access switch (inbound, outbound, etc.)
  Proxy access switch (inbound, outbound, etc.)
  System node version
  Async Line - Inbound node type

- Information for Inactive Nodes (To be downline loaded)

  Service device type
  CPU type
  Software type
  Host address (input and output)
  Service line
  Service password
  Load file
  Secondary loader
  Tertiary loader
  Software ID
  Dump address/Dump count
  Dump file/Secondary dumper
  Diagnostic load file
  NI hardware address for node

2.12 Local Node Information (LNI)

The LNI defines information about the local node including its state and the setting of executor parameters.

2.12.1 LNI Fields

- Node address
- Node State
- Local node type
- Maximum links allowed
- Maximum node address
- Maximum transport buffers
- Maximum cost
- Maximum hops
- Maximum visits
- Maximum circuits
- Default LOOP data/count/length/help type
- Transport forwarding buffer size
- Transport segment buffer size
- Routine suppression interval (units = sec)
- Inactivity timer (units = sec)
- Incoming timer (units = sec)
- Outgoing timer (units = sec)
- Routing timer (units = sec)
- Broadcast routing timer (units = sec)
- Maximum broadcast end nodes
- Maximum broadcast routers
- Delay factor
- Delay weight
- Retransmit factor
- Default access (inbound, outbound, etc.)
- Default proxy access (inbound, outbound, etc.)
- Pipeline quota
- X.25 subaddress range
- Maximum areas
- Area maximum hops
- Area maximum cost
- Alias local address (cluster node address)
- Alias maximum links
- Node name
- Counters
- System identification
- NSP, Routing, and Network Management version
- Physical NI address

2.13 Network Server Process Information (SPI)

Network Server Process Information contains information that is used by a waiting network server process.

2.13.1 SPI Information

- Server PID
- IRP of waiting DECLSERV QIO (0 if process active)
- Remote node address which initially started server
- Channel associated with DECLSERV IRP
- Access Control used initially to start server process
- Remote user ID which initially started server
- Last (current) filespec given to server
- Last (current) NCB given to server
- Last (current) process name given to server
- Structure size

2.14 Work Queue Elements (WQE)

Work Queue Elements (WQE) are used by NETACP to serialize and standardize all schedulable but non-IRP oriented work. Datalink state transition control and events originating from ASTs are examples.

The WQE structure is depicted in Figure 4-6.

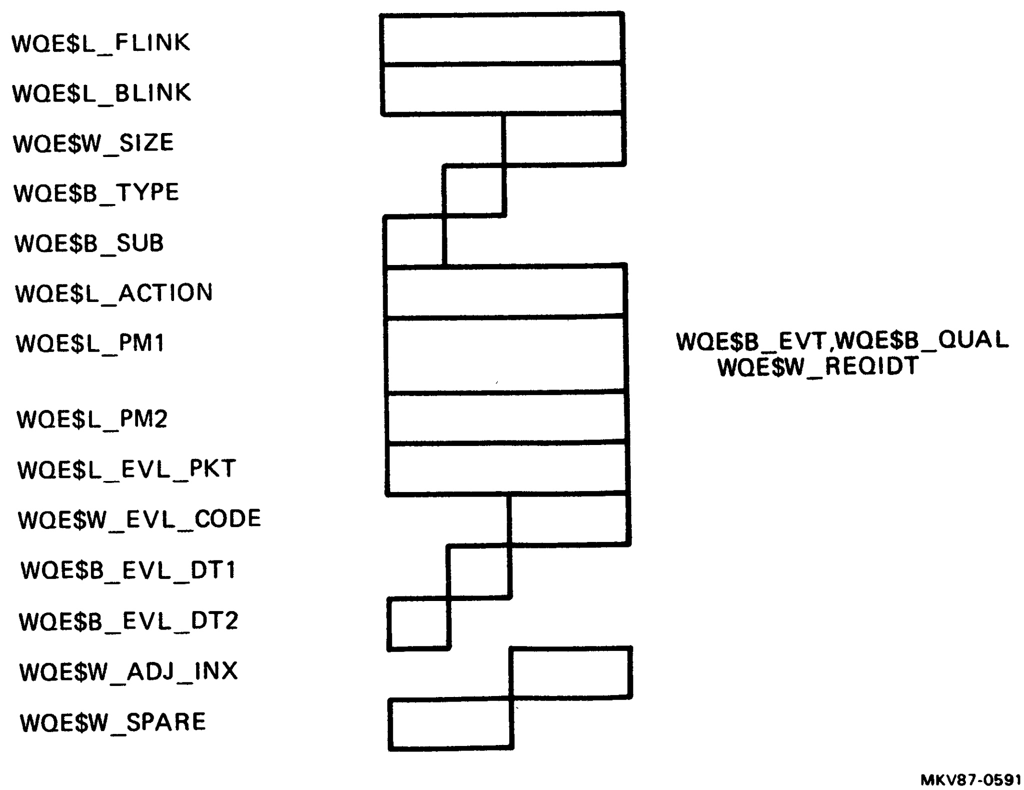


Figure 4-6 Work Queue Elements (WQE)

2.14.1 Queuing and Dequeuing WQES

The WQE$B_SUB field is used to determine if any special processing is needed when the WQE is queued or dequeued as follows:

- Spawned during normal internal ACP activity, e.g., during IO$_ACPCONTROL QIO activity (then no special action is required when it is queued.)

  When it is dequeued, dispatch directly to the action routine that is responsible for deallocating it.

- Consequence of a miscellaneous AST (a datalink QIO AST).

  When it is dequeued, dispatch directly to the action routine that is responsible for deallocating it.

- Consequence of a mailbox read AST.

  When it is dequeued, it is sent to the mailbox servicing routine, which permanently owns the WQE.

- Consequence of a timer AST.

  When it is dequeued, another VMS timer must be set if there are any more elements in the WQE timer queue.

NOTE

If any of the AST related elements is the first element queued, $ WAKE NETACP.

2.15 Node Counter Block (NDC)

- Used to maintain statistics for each node in the network
- A hash of these structures is contained in NETACP
- The number of entries depends on maximum address
- Each entry is 28 bytes
- Contains information that can be displayed using the NCP show node xx counters command

2.15.1 NDC Fields

- Absolute time counter block was last zeroed
- Transmitted connect rejects due to resource errors
- Response timeouts
- Connects received
- Connects sent
- Bytes received
- Bytes sent
- Packets received
- Packets sent

2.16 Event Logger Data Structures

There are two data structures used by EVL to do event logging, the Event Logging Filter Information (EVI) block and the Event Logging Sink Information (ESI).

2.17 Object Information Block (OBI)

The Object Information Block (OBI) has information on network objects.

2.17.1 OBI Fields

- Object number
- Proxy login switch (inbound, outbound, etc)
- Channel over which declaration occurred
- Low-order privilege mask
- High-order privilege mask
- Associated NET UCB if declared task
- Associated process i.d. if declared task
- Name
- File id
- User id
- Account
- Password

2.18 Circuit Information (CRI)

The CRI contains information about each circuit. There is one CRI block for each circuit.

2.18.1 CRI Fields

- Circuit name
- PID of temporary owner of line in service state
- Absolute due time for counter logging
- State
- Loopback name
- Hello timer
- Cost
- Maximum recalls
- Recall timer
- Call Number
- Type
- DTE
- Maximum block
- Maximum window
- Tributary
- Babble timer
- Transmit timer
- X.25 channel
- X.25 Usage
- Maximum receive buffers
- Maximum transmits
- Active base
- Active increment
- Inactive base
- Inactive increment
- Inactive threshold
- Dying base
- Dying increment
- Dying threshold
- Dead threshold
- Transport protocol
- Maximum routers on NI
- Router priority on NI
- Async line verification Enabled/Disabled
- X.25 network name
- Loopback name

2.19 Physical Line Information (PLI)

The PLI includes information about the physical lines on the system. There is one PLI block for each line.

2.19.1 PLI Fields

- Line name
- Absolute time to counter logging
- Number of buffers in receive pool
- State
- Substate
- Protocol
- Counter timer
- Service timer
- Holdback timer
- Retransmit timer
- Maximum block
- Maximum retransmits
- Maximum window
- Scheduling timer
- Dead timer
- Delay timer
- Stream timer
- NI hardware address [READ ONLY]
- X.25 KMX microcode dump file [WRITE ONLY - ONE SHOT]
- Ethernet protocol type
- X.25 mode (DTE, DCE, etc.)
- Transmit pipeline
- Async Line - Line speed
- Async Line - Switch
- Async Line - Hangup
- Buffer size to override executor buffer size
- X.25 network name

USING THE SYSTEM DUMP ANALYZER (SDA)

INTRODUCTION

The System Dump Analyzer (SDA) utility is a useful tool to help analyze a crash dump or look at a running system. It allows you to format various data structures used by DECnet and to examine the contents of memory.

Topics include:

- Location of Data Structures
- Using SDA to Look at DECnet Data Structures

1 LOCATION OF DATA STRUCTURES

- Data Structures in Nonpaged Pool
  - Data link drivers
  - Data link buffers (CXB)
  - Circuit data (LPD)
  - Routing table (RCB)
  - Adjacency data (ADJ)
  - Logical link data (XWB)
  - Output adjacency (OA)
  - Area output adjacency (AOA)
- Data Structures in Paged Memory (from Volatile Database)
  - Local node database (LNI)
  - Remote node database (NDI)
  - Line database (PLI)
  - Circuit database (CRI)
  - Object database (OBI)
  - Event database (EFI,ESI)

2 DATA STRUCTURE TRACE USING SDA

The following system crash dump was taken on VMS version 4.4.

The purpose is not to analyze the cause of the crash but rather to trace the key DECnet data structures discussed.

The NCP and DCL commands show the executive, circuit, and system characteristics prior to the crash.

The SDA trace looks at key DECnet data structures relating to the network data link drivers, network pseudo devices (NETn), the routing database (RCB,ADJ,OA), logical link structures (LTB), and the node counter block.

Numbered comments explaining the analysis of the data appear after the trace in Section 2.11.

2.1 Looking at the System Before the Crash

```
$ SHOW NETWORK

VAX/VMS Network status for local node  1.2 THUD on 23-MAY-1986 13:57:06.13

        Node            Links  Cost  Hops   Next Hop to Node

   1.2      THUD          0      0     0     (Local)   ->  1.2    THUD
   1.1      SPLASH        4      1     1     UNA-0     ->  1.1    SPLASH
   1.4      BAROOM        1      1     1     UNA-0     ->  1.4    BAROOM
   1.5      CLICK         0      1     1     UNA-0     ->  1.5    CLICK
   1.6      DRIP          0      1     1     UNA-0     ->  1.6    DRIP

              Total of 5 nodes.


$ SHOW SYSTEM

VAX/VMS V4.4   on node THUD 23-MAY-1986 13:55:51.27  Uptime   0 00:42:55
  Pid     Process Name State  Pri   I/O       CPU        Page flts Ph.Mem
00000080 NULL          COM     0      0   0 00:33:15.03       0       0
00000081 SWAPPER       HIB    16      0   0 00:00:00.79       0       0
00000084 ERRFMT        HIB     8     41   0 00:00:00.72      70      89
00000085 OPCOM         LEF     8     48   0 00:00:01.25     247     121
00000086 JOB_CONTROL   HIB     8     62   0 00:00:00.74      83     204
00000089 NETACP        HIB     9    144   0 00:00:07.19     315     231
0000008A EVL           HIB     5     53   0 00:00:01.71     816      32  N
0000008B REMACP        HIB     9     23   0 00:00:00.31      74      40
0000008F FAL_1031      LEF     6    196   0 00:00:07.55     748     211  N
00000090 NET_1032      LEF     6    103   0 00:00:04.42     425     263  N
00000091 NET_1034      LEF     6    124   0 00:00:05.06     887     150  N
00000092 Scott         HIB     7    250   0 00:00:17.38    1163     249
00000097 SYSTEM        CUR     4    183   0 00:00:04.64     762     196


$ SHOW USERS

          VAX/VMS Interactive Users              23-MAY-1986 13:56:19.70
      Total number of interactive users = 2
 Username         Process Name        PID      Terminal
 SCOTT            Scott             00000092   OPA0:
 SYSTEM           SYSTEM            00000096   RTA1:
```

```
$ MCR NCP

NCP> SHOW EXECUTOR CHARACTERISTICS

Node Volatile Characteristics as of 23-MAY-1986 13:52:40

Executor node = 1.2 (THUD)

Identification             = VAX 750 - Standalone System
Management version         = V4.0.0
Incoming timer             = 45
Outgoing timer             = 45
NSP version                = V4.0.0
Maximum links              = 32
Delay factor               = 80
Delay weight               = 5
Inactivity timer           = 60
Retransmit factor          = 10
Routing version            = V2.0.0
Type                       = routing IV
Routing timer              = 600
Broadcast routing timer    = 180
Maximum address            = 15
Maximum circuits           = 10
Maximum cost               = 10
Maximum hops               = 5
Maximum visits             = 10
Maximum area               = 63
Max broadcast nonrouters   = 64
Max broadcast routers      = 32
Area maximum cost          = 1022
Area maximum hops          = 30
Maximum buffers            = 15
Buffer size                = 576
Nonprivileged user id      = DECNET
Default access             = incoming and outgoing
Pipeline quota             = 5000
Default proxy access       = incoming and outgoing
Parameter #2743            = 32
```

```
NCP>SHOW KNOWN NODES SUMMARY

Executor node = 1.2 (THUD)

State                      = on
Identification             = VAX 750 - Standalone System

    Node             State        Active  Delay   Circuit      Next node
                                  Links

 1.1  (SPLASH)   reachable         4       1     UNA-0          1.1 (SPLASH)
 1.3  (ZIP)      unreachable
 1.4  (BAROOM)   reachable         1       1     UNA-0          1.4 (BAROOM)
 1.5  (CLICK)    reachable                       UNA-0          1.5 (CLICK)
 1.6  (DRIP)     reachable                       UNA-0          1.6 (DRIP)
 1.13 (FIZZ)     unreachable
 1.14 (SNAP)     unreachable
 1.15 (POOF)     unreachable
```

Names are in volatile databa
Names never stored in non paged po

```
NCP> SHOW ACTIVE CIRCUITS CHARACTERISTICS

Circuit = DMC-0

State                      = on
Substate                   =
Service                    = enabled
Cost                       = 2
Hello timer                = 15
Verification               = disabled

Circuit = UNA-0

State                      = on
Service                    = enabled
Designated router          =  1.6 (DRIP)
Cost                       = 1
Router priority            = 64
Hello timer                = 15
Type                       = Ethernet
Adjacent node              =  1.6 (DRIP)
Listen timer               = 45
```

walking thru LPD's

NCP>SHOW KNOWN LINKS STATUS

Known Link Volatile Summary as of 23-MAY-1986 13:56:30

Link	Node	PID	Process	Remote link	Remote user	State
1031	1.1 (SPLASH)	0000008F	FAL_1031	8223	SCOTT	run
1032	1.1 (SPLASH)	00000090	NET_1032	7200	SCOTT	DI received
1036	1.1 (SPLASH)	00000092	Scott	8225	FAL	run
1045	1.1 (SPLASH)	0000008B	REMACP	8232	SCOTT	run
1034	1.4 (BAROOM)	00000091	NET_1034	1028	DRV2	run

walking down LTB pulling out XWB all info stored there

disconnect init

NCP>TELL SPLASH SHOW KNOWN LINKS

Known Link Volatile Summary as of 23-MAY-1986 13:58:00

Link	Node	PID	Process	Remote link	Remote user	State
8223	1.2 (THUD)	2060126F	Scott #3	1031	FAL	run
8225	1.2 (THUD)	2060177C	FAL_8225	1036	SCOTT	run
8232	1.2 (THUD)	2060126F	Scott #3	1045	CTERM	run
8231	1.2 (THUD)	2060137E	NML_8231	1042	SYSTEM	run

NCP>TELL BAROOM SHOW KNOWN LINKS

Known Link Volatile Summary as of 23-MAY-1986 13:58:36

Link	Node	PID	Process	Remote link	Remote user	State
1028	1.2 (THUD)	20A0009B	DRV2	1034	TARGET1.EXE	run
2055	1.2 (THUD)	20A0009F	NML_2055	1044	SYSTEM	run

Sync DI send disconnect, QIO returns status, other side confirmed or not.

FAL does a synchronous disconnect. (object sends disconnect) declared object that stays around, you will have to request disconnect.

Async → tell other side you are going away & do it right away.

2.2 Getting Started with SDA

```
SDA> READ SYS$SYSTEM:SYSDEF          !VMS data structure definitions

SDA> READ SYS$SYSTEM:NETDEF          !DECnet data structure definitions


SDA> SHOW SUMMARY

Current process summary
-----------------------

 Extended Indx Process name Username State Pri   PCB      PHD    Wkset
 -- PID -- ---- ------------ -------- ----- --- -------- -------- -----

 00000080 0000 NULL                    COM    0 800024A8 80002328     0
 00000081 0001 SWAPPER                 HIB   16 80002748 800025C8     0
 00000084 0004 ERRFMT       SYSTEM     HIB    8 801545D0 802EA200    89
 00000085 0005 OPCOM        SYSTEM     LEF    8 80155490 8031BE00   121
 00000086 0006 JOB_CONTROL  SYSTEM     HIB   10 801557A0 80334C00   204
 00000089 0009 NETACP       DECNET     HIB   10 80162B70 8034DA00   236
 0000008A 000A EVL          DECNET     HIB    6 80163D50 80366800    43
 0000008B 000B REMACP       SYSTEM     HIB    9 801681C0 8037F600    40
 0000008F 000F FAL_1031     SCOTT      LEF    6 80165190 80398400   211
 00000090 0010 NET_1032     SCOTT      LEF    6 80169AD0 803B1200   263
 00000091 0011 NET_1034     NET1       LEF    6 80169D70 803CA000   150
 00000092 0012 Scott        SCOTT      CUR    6 80155370 80303000   198
 00000097 0017 SYSTEM       SYSTEM     LEF    4 80169970 803E2E00   148
```

for cluster

PID = IPID internal PID
index into process slot table.
Low order part EPID

EPID extended
has an identifier that is
unique clusterwide

2.2.1 SDA> SHOW CRASH

```
System crash information
------------------------
Time of system crash: 23-MAY-1986 14:02:32.21

Version of system: VAX/VMS VERSION V4.4

VAXcluster node name: THUD

Reason for BUGCHECK exception: OPERATOR, Operator requested system shutdown

Process currently executing: Scott

Current image file: THUD$DRA0:[SYS0.][SYSEXE]OPCCRASH.EXE;1

Current IPL: 31  (decimal)

General registers:

    R0  = 0000000B   R1  = 00000241   R2  = 7FFDB390   R3  = 80000FF8
    R4  = 80155370   R5  = 00000001   R6  = 7FFED78A   R7  = 7FFED78A
    R8  = 7FFED052   R9  = 7FFED25A   R10 = 00000000   R11 = 7FFE33DC
    AP  = 00000000   FP  = 7FFE7DD0   SP  = 7FFE7DD0   PC  = 0000036E
    PSL = 00DF0000

Processor registers:

    P0BR    = 80307E00     PCBB    = 00501478     ACCS    = 00000001
    P0LR    = 00000004     SCBB    = 007EA200     TBDR    = 00000000
    P1BR    = 7FB1BE00     ASTLVL  = 00000004     CADR    = 00000000
    P1LR    = 001FF9DA     SISR    = 00000000     MCESR   = 00000004
    SBR     = 007EE600     ICCS    = 800000C1     CAER    = 00000000
    SLR     = 00004680     ICR     = FFFFEA94     CMIERR  = 00080310
                           TODR    = 596DD6FC

    ISP     = 802E9C00
    KSP     = 7FFE7DD0
    ESP     = 7FFE9E00
    SSP     = 7FFED04E
    USP     = 7FF3DB94
```

2.2.2 SDA> SHOW POOL/SUMMARY

0/0's are a little light

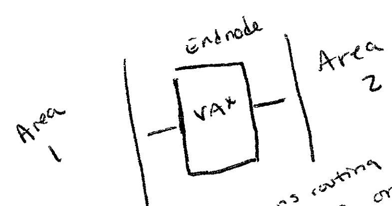
```
IRP lookaside list
------------------
Summary of IRP lookaside list

    3  CRB        =      624  (2%)
   90  FCB        =    18720  (67%)
   20  IRP        =     4160  (14%)
    1  WCB        =      208  (0%)
    2  NET        =      416  (1%)
   11  JIB        =     2288  (8%)
    6  RSB        =     1248  (4%)
    1  INIT       =      208  (0%)

Total space used = 27872 out of 108160 total bytes, 80288 bytes left
Total space utilization = 25%

LRP lookaside list
------------------
Summary of LRP lookaside list

    8  CXB        =    12672  (80%)
    1  DPT        =     1584  (10%)
    1  LKID       =     1584  (10%)

Total space used = 15840 out of 33264 total bytes, 17424 bytes left
Total space utilization = 47%

SRP lookaside list
------------------
Summary of SRP lookaside list

    3  ADP        =      288  (1%)
    2  AQB        =      192  (1%)
   10  CRB        =      960  (6%)
   11  DDB        =     1056  (6%)
   13  IDB        =     1248  (8%)
   11  TQE        =     1056  (6%)
   89  WCB        =     8544  (55%)
    8  BUFIO      =      768  (4%)
    2  NET        =      192  (1%)
    1  PTR        =       96  (0%)
    7  LKB        =      672  (4%)
    1  RSB        =       96  (0%)
    1  RIGHTSLIS  =       96  (0%)
    1  CIA        =       96  (0%)
    1  INIT       =       96  (0%)

Total space used = 15456 out of 71616 total bytes, 56160 bytes left
Total space utilization = 21%
```

```
Nonpaged dynamic storage pool
------------------------------
Summary of nonpaged pool contents

   40  UNKNOWN   =    16816  (11%)
    1  ADP       =      608  (0%)
   10  PCB       =     2880  (1%)
    2  RVT       =    34304  (23%)
   26  UCB       =    10544  (7%)
    4  VCB       =      960  (0%)
    1  WCB       =    21808  (14%)
    4  NET       =     2624  (1%)
    5  DPT       =    31744  (21%)
    2  RBM       =      992  (0%)
    3  VCA       =     4896  (3%)
    1  RSHT      =     1040  (0%)
    5  INIT      =    18912  (12%)

Total space used = 148128 out of 287744 total bytes, 139616 bytes left

Total space utilization = 51%

Paged dynamic storage pool
--------------------------
Summary of paged pool contents

   10  UNKNOWN   =    69392  (34%)
    1  PQB       =     2256  (1%)
   85  GSD       =     4096  (2%)
   71  KFE       =     4560  (2%)
    3  MTL       =       96  (0%)
    1  JNLWCB    =    22864  (11%)
   56  KFRH      =    17776  (8%)
    1  TWP       =    12336  (6%)
    1  RSHT      =      528  (0%)
  208  LNM       =    15392  (7%)
    1  FLK       =    11824  (5%)
    1  RIGHTSLIS =     2560  (1%)
    2  KFD       =       96  (0%)
    1  KFPB      =       16  (0%)
    2  CIA       =    19456  (9%)
    1  PFB       =    16720  (8%)
    1  ORB       =     2048  (1%)

Total space used = 202016 out of 253952 total bytes, 51936 bytes left

Total space utilization = 79%
```

2.3 Using SDA to Look at Data Link Drivers

2.3.1 SDA> SHOW DEVICE XEA

```
                 DDB list
                 --------
     Address     Controller     ACP       Driver      DPT      DPT size
     -------     ----------     ---       ------      ---      --------
    802B15E0     XEA                      XEDRIVER  80156F80   3930

Controller: XEA
---------------
        --- Device Data Block (DDB) 802B15E0 ---

Driver name          XEDRIVER   Alloc. class       0 DDT address      80157050
                                SB address      80000EF4
                                UCB address     8015A8B0

        ---  Primary Channel Request Block (CRB) 802B1580 ---

Reference count           4  Wait queue       empty  Aux. struct.      8015AAA0
IDB address        802B1700  Datapath             0  Map reg.             48(6)
ADP address        8014FA00  Unit init.    801572FC  Int. service      80158AB3
Unit start rout.801573A2                             Ctrl. init.       801572F8

        --- Interrupt Data Block (IDB) 802B1700 ---

CSR address       8002F548  Owner UCB addr. 00000000  ADP address      8014FA00
Number of units          8  Interrupt vector  000120

     --- Driver Dispatch Table (DDT) 80157050 ---

Errlog buf sz            0  Diag buf sz          76  FDT size                76
Start I/O         80158148  Register dump 801597D3  FDT address       80157088
Alt start I/O     801578BA  Unit init        return  Mnt verify        8000CF43
Cancel I/O        80159E92  Unsol int        return  Cloned UCB        801572F9


XEA0                                       DEUNA          UCB address:  8015A8B0

Device status:    00002010 online,template
Characteristics:  0C042000 net,avl,idv,odv
                  00000000
Owner UIC [000000,000000] Operation count          0 ORB address     8015AA3B
      PID         00000000 Error count              0 DDB address     802B15E0
Class/Type           20/0E Reference count          0 DDT address     80157050
Def. buf. size         512 BOFF                  0000 CRB address     802B1580
DEVDEPEND         00000000 Byte count            0000 I/O wait queue     empty
DEVDEPND2         00000000 SVAPTE            00000000
FIPL/DIPL            08/15 DEVSTS                0000
Charge PID        00000000
    *** I/O request queue is empty ***
```



if nobody using can delete

DNA port for ethernet device

```
XEA1                                              DEUNA        UCB address:  80164320

Device status:    00010010 online,deleteucb
Characteristics:  0C042000 net,avl,idv,odv
                  00000000
Owner UIC [000001,000004] Operation count      2410   ORB address    801644AB
      PID         00010009 Error count            0   DDB address    802B15E0
Class/Type           20/0E Reference count        2   DDT address    80157050
Def. buf. size        1498 BOFF                0098   CRB address    802B1580
DEVDEPEND         00000800 Byte count          0047   AMB address    80162C90
DEVDEPND2         00000000 SVAPTE          808AE46C   I/O wait queue   empty
FIPL/DIPL            08/15 DEVSTS              0075
Charge PID        00010009
    *** I/O request queue is empty ***

XEA2                                              DEUNA        UCB address:  80164720

Device status:    00010010 online,deleteucb
Characteristics:  0C052000 net,shr,avl,idv,odv
                  00000000
Owner UIC [000000,000000] Operation count         1 ORB address      801648AB
      PID         00000000 Error count            0 DDB address      802B15E0
Class/Type           20/0E Reference count        2 DDT address      80157050
Def. buf. size        1498 BOFF                004E CRB address      802B1580
DEVDEPEND         00000800 Byte count          0042 I/O wait queue     empty
DEVDEPND2         00000000 SVAPTE          802AF840
FIPL/DIPL            08/15 DEVSTS              001D
Charge PID        00010009
    *** I/O request queue is empty ***

XEA3                                              DEUNA        UCB address:  80164910

Device status:    00010010 online,deleteucb
Characteristics:  0C052000 net,shr,avl,idv,odv
                  00000000
Owner UIC [000000,000000] Operation count         1   ORB address    80164A9B
      PID         00000000 Error count            0   DDB address    802B15E0
Class/Type           20/0E Reference count        2   DDT address    80157050
Def. buf. size        1500 BOFF                004E   CRB address    802B1580
DEVDEPEND         00000800 Byte count          0042   I/O wait queue   empty
DEVDEPND2         00000000 SVAPTE          802AF780
FIPL/DIPL            08/15 DEVSTS              001D
Charge PID        00010009
    *** I/O request queue is empty ***
```

LAT or MOP

EVL

2.3.2 SDA> SHOW DEVICE XMA

DDCMP driver

```
                DDB list
                --------
    Address     Controller     ACP      Driver       DPT     DPT size
    -------     ----------     ---      ------       ---     --------

   802B1C40     XMA                     XMDRIVER   8015AD80   1000

Controller: XMA
---------------

        --- Device Data Block (DDB) 802B1C40 ---

Driver name         XMDRIVER Alloc. class          0    DDT address 8015ADEC
                             SB address     80000EF4
                             UCB address    8015BD80

    ---  Primary Channel Request Block (CRB) 80241AC0 ---

Reference count          1     Wait queue         empty
IDB address       802B1BE0     Datapath               0    Map reg.         78(2)
ADP address       8014FA00     Unit init.      8015AE90    Int. service 8015B8A8

    --- Interrupt Data Block (IDB) 802B1BE0 ---

CSR address       8002DC38     Owner UCB addr. 00000000    ADP address   8014FA00
Number of units          8     Interrupt vector  000300


        --- Driver Dispatch Table (DDT) 8015ADEC ---

Errlog buf sz            0   Diag buf sz          68    FDT size             64
Start I/O         8015B255   Register dump 8015BACE    FDT address     8015AE24
Alt start I/O     8015AFF4   Unit init        return    Mnt verify      8000CF43
Cancel I/O        8015BB9E   Unsol int        return    Cloned UCB        return


XMA0                                        DMR11       UCB address:    8015BD80

Device status:     00000010 online
Characteristics:   0C042000 net,avl,idv,odv
                   00000000
Owner UIC [000000,000000]     Operation count     338    ORB address    8015BF9E
       PID         00010009   Error count           0    DDB address    802B1C40
Class/Type            20/02   Reference count       2    DDT address    8015ADEC
Def. buf. size          576   BOFF               01C8    CRB address    80241AC0
DEVDEPEND          86000800   Byte count         0006    AMB address    80162C90
DEVDEPND2          00000000   SVAPTE         808AE460    I/O wait queue    empty
FIPL/DIPL             08/15   DEVSTS             0008
Charge PID         00000000
    *** I/O request queue is empty ***
```

2.4 Using SDA to Look at Network (NETxx) Devices

2.4.1 SDA> SHOW DEVICE NET

```
I/O data structures
-------------------
                DDB list
                --------

     Address    Controller     ACP        Driver      DPT     DPT size
     -------    ----------     ---        ------      ---     --------

    802A7BC0    NET                     NETDRIVER  8015E520   3B00

Controller: NET
---------------

        --- Device Data Block (DDB) 802A7BC0 ---

Driver name        NETDRIVER Alloc. class          0  DDT address 8015E5B0
                             SB address     80000EF4
                             UCB address    80162020

    ---  Primary Channel Request Block (CRB) 802A7920 ---

Reference count        11    Wait queue         empty
IDB address      802A72C0    Unit init.    8015E892  Int. service    8015E891
Unit start rout.8015EF19                             Ctrl. init.     8015E891

        --- Interrupt Data Block (IDB) 802A72C0 ---

CSR address      00000000    Owner UCB addr. 00000000  ADP address 00000000
Number of units         8


        --- Driver Dispatch Table (DDT) 8015E5B0 ---

Errlog buf sz           0  Diag buf sz            0  FDT size            88
Start I/O        8015EA40  Register dump     return  FDT address   8015E5E8
Alt start I/O    8015F493  Unit init         return  Mnt verify    8000CF43
Cancel I/O       8015EE2B  Unsol int       8015F5F0  Cloned UCB      return
```

Logical Link

```
NET0                                          Unknown     UCB address:  80162020

Device status:    00002010 online,template
Characteristics:  0C1C2000 net,avl,mnt,mbx,idv,odv
                  00000000

Owner UIC [000001,000001] Operation count         0   ORB address   801620B0
      PID         00000000 Error count             0   DDB address   802A7BC0
Class/Type           00/00 Reference count         0   DDT address   8015E5B0
Def. buf. size         256 BOFF                 0000   VCB address   80162E90
DEVDEPEND         00000001 Byte count           0000   CRB address   802A7920
DEVDEPND2         00000000 SVAPTE           00000000   I/O wait queue   empty
FIPL/DIPL            08/08 DEVSTS               0000
Charge PID        00000000

    *** I/O request queue is empty ***

        --- Volume Control Block (VCB) 80162E90 ---

Transactions      6    Mount count         11992    AQB address      80162E70

            --- ACP Queue Block (AQB) 80162E70 ---

ACP requests are serviced by process NETACP whose PID is 00010009

Status: 01 unique
Mount count         1    ACP type              net     Linkage           802A3E40
                         ACP class             128     Request queue        empty

    *** ACP request queue is empty ***


NET1                                  Unknown          UCB address:  80162D80

Device status:    00010010 online,deleteucb
Characteristics:  0C1C2000 net,avl,mnt,mbx,idv,odv
                  00000000

Owner UIC [000001,000004]  Operation count        0   ORB address   80162E10
      PID         00010009  Error count            0   DDB address   802A7BC0
Class/Type           00/00  Reference count        1   DDT address   8015E5B0
Def. buf. size         256  BOFF                0000   VCB address   80162E90
DEVDEPEND         00000001  Byte count          0000   CRB address   802A7920
DEVDEPND2         00000000  SVAPTE          00000000   AMB address   80162C90
FIPL/DIPL            08/08  DEVSTS              0002   I/O wait queue   empty
Charge PID        00010009

    *** I/O request queue is empty ***
```

```
        --- Volume Control Block (VCB) 80162E90 ---

Transactions          6    Mount count     11992    AQB address      80162E70

           --- ACP Queue Block (AQB) 80162E70 ---

ACP requests are serviced by process NETACP whose PID is 00010009

Status: 01 unique
Mount count       1     ACP type             net     Linkage          802A3E40
                        ACP class            128     Request queue       empty

    *** ACP request queue is empty ***


NET3                                 Unknown           UCB address:  80163C60

Device status:    00010010 online,deleteucb
Characteristics: 0C1C2000 net,avl,mnt,mbx,idv,odv
                 00000000

Owner UIC [000001,000004]  Operation count      7   ORB address   80163CF0
      PID        0001000A  Error count          0   DDB address   802A7BC0
Class/Type          00/00  Reference count      1   DDT address   8015E5B0
Def. buf. size        256  BOFF              0000   VCB address   80162E90
DEVDEPEND        00000001  Byte count        0000   CRB address   802A7920
DEVDEPND2        00000000  SVAPTE        00000000   I/O wait queue   empty
FIPL/DIPL           08/08  DEVSTS            0002
Charge PID       0001000A

    *** I/O request queue is empty ***

        --- Volume Control Block (VCB) 80162E90 ---

Transactions        6     Mount count       11992    AQB address      80162E70

           --- ACP Queue Block (AQB) 80162E70 ---

ACP requests are serviced by process NETACP whose PID is 00010009

Status: 01 unique

Mount count             1    ACP type         net     Linkage          802A3E40
                             ACP class        128     Request queue       empty

    *** ACP request queue is empty ***
```

```
NET4                                  Unknown           UCB address:  80163F60

Device status:     00010010 online,deleteucb
Characteristics:   0C1C2000 net,avl,mnt,mbx,idv,odv
                   00000000
Owner UIC [000001,000004]   Operation count       2    ORB address   80163FF0
      PID          0001000A   Error count           0    DDB address   802A7BC0
Class/Type            00/00   Reference count       1    DDT address   8015E5B0
Def. buf. size          256   BOFF               0000    VCB address   80162E90
DEVDEPEND          FFFFFFFF   Byte count         0000    CRB address   802A7920
DEVDEPND2          00000000   SVAPTE         00000000    AMB address   80163E70
FIPL/DIPL             08/08   DEVSTS             0002    I/O wait queue  empty
Charge PID         0001000A
    *** I/O request queue is empty ***

        --- Volume Control Block (VCB) 80162E90 ---
Transactions        6     Mount count          11992    AQB address      80162E70

           --- ACP Queue Block (AQB) 80162E70 ---
ACP requests are serviced by process NETACP whose PID is 00010009

Status: 01 unique
Mount count       1      ACP type              net     Linkage          802A3E40
                         ACP class             128     Request queue       empty
    *** ACP request queue is empty ***


NET30                                 Unknown           UCB address:  801654E0

Device status:     00010010 online,deleteucb
Characteristics:   0C1C2000 net,avl,mnt,mbx,idv,odv
                   00000000
Owner UIC [000100,000025]   Operation count       7    ORB address   80165570
      PID          0001000F   Error count           0    DDB address   802A7BC0
Class/Type            00/00   Reference count       1    DDT address   8015E5B0
Def. buf. size          256   BOFF               0058    VCB address   80162E90
DEVDEPEND          00000001   Byte count         0005    CRB address   802A7920
DEVDEPND2          00000000   SVAPTE         802AAEC0    AMB address   80165010
FIPL/DIPL             08/08   DEVSTS             0002    I/O wait queue  empty
Charge PID         0001000F
    *** I/O request queue is empty ***

        --- Volume Control Block (VCB) 80162E90 ---
Transactions             6     Mount count          11992    AQB address      80162E70

           --- ACP Queue Block (AQB) 80162E70 ---
ACP requests are serviced by process NETACP whose PID is 00010009

Status: 01 unique
Mount count       1      ACP type              net     Linkage          802A3E40
                         ACP class             128     Request queue       empty
    *** ACP request queue is empty ***
```

```
NET40                                Unknown            UCB address:  801656C0

Device status:    00010010 online,deleteucb
Characteristics:  0C1C2000 net,avl,mnt,mbx,idv,odv
                  00000000
Owner UIC [000100,000025]  Operation count      7   ORB address   80165750
      PID         00010012  Error count          0   DDB address   802A7BC0
Class/Type           00/00  Reference count      1   DDT address   8015E5B0
Def. buf. size         256  BOFF              004A   VCB address   80162E90
DEVDEPEND         00000001  Byte count        0005   CRB address   802A7920
DEVDEPND2         00000000  SVAPTE        802B07A0   I/O wait queue empty
FIPL/DIPL            08/08  DEVSTS            0002
Charge PID        00010012
    *** I/O request queue is empty ***
        --- Volume Control Block (VCB) 80162E90 ---

Transactions        6    Mount count       11992    AQB address     80162E70
            --- ACP Queue Block (AQB) 80162E70 ---

ACP requests are serviced by process NETACP whose PID is 00010009

Status: 01 unique
Mount count       1     ACP type            net    Linkage           802A3E40
                        ACP class           128    Request queue        empty
    *** ACP request queue is empty ***

NET57                                Unknown            UCB address:  80168530

Device status:    00010010 online,deleteucb
Characteristics:  0C1C2000 net,avl,mnt,mbx,idv,odv
                  00000000
Owner UIC [000001,000003]  Operation count    113   ORB address   801685C0
      PID         0001000B  Error count          0   DDB address   802A7BC0
Class/Type           00/00  Reference count      1   DDT address   8015E5B0
Def. buf. size         256  BOFF              006B   VCB address   80162E90
DEVDEPEND         00000001  Byte count        0005   CRB address   802A7920
DEVDEPND2         00000000  SVAPTE        8024B520   AMB address   80164F20
FIPL/DIPL            08/08  DEVSTS            0002   I/O wait queue  empty
Charge PID        0001000B
    *** I/O request queue is empty ***

        --- Volume Control Block (VCB) 80162E90 ---
Transactions        6    Mount count       11992    AQB address     80162E70

            --- ACP Queue Block (AQB) 80162E70 ---
ACP requests are serviced by process NETACP whose PID is 00010009

Status: 01 unique
Mount count       1     ACP type            net    Linkage           802A3E40
                        ACP class           128    Request queue        empty
    *** ACP request queue is empty ***
```

2.5 Using SDA to Look at Mailbox (MBAN) Devices

2.5.1 SDA> SHOW DEVICE MB

```
I/O data structures
-------------------
                DDB list
                --------

    Address     Controller     ACP       Driver      DPT      DPT size
    -------     ----------     ---       ------      ---      --------

    800013BC     MBA                     MBDRIVER   8000185C   0602

Controller: MBA
---------------

    --- Device Data Block (DDB) 800013BC ---

Driver name       MBDRIVER    Alloc. class    0   DDT address      800019AC
                                 SB address      80000EF4
                                 UCB address     800014E8

    ---  Primary Channel Request Block (CRB) 80001814 ---

Reference count          6     Wait queue        00000000
IDB address       80001814

        --- Interrupt Data Block (IDB) 80001814 ---

CSR address       00000000  Owner UCB addr. 00000000  ADP address   00000000
Number of units          6

        --- Driver Dispatch Table (DDT) 800019AC ---

Errlog buf sz            0  Diag buf sz          0    FDT size            64
Start I/O         80001DC2  Register dump   return    FDT address   800019E4
Alt start I/O       return  Unit init       return    Mnt verify    8000CF43
Cancel I/O        80001AF8  Unsol int       return    Cloned UCB      return
```

```
MBA1                                    Unknown          UCB address:  800014E8

Device status:     00000110 online,bsy
Characteristics:   0C150001 rec,shr,avl,mbx,idv,odv
                   00000200 nnm

Owner UIC [000001,000004]  Operation count     17   ORB address   80001578
      PID          00000000  Error count          0   DDB address   800013BC
Class/Type            A0/00  Reference count      2   DDT address   800019AC
Def. buf. size         1024  BOFF              0000   CRB address   80001814
DEVDEPEND          00000000  Byte count        0400   IRP address   8024DC20
DEVDEPND2          00000000  SVAPTE        00000000   Fork R4       0000003C
FIPL/DIPL             0B/0B  DEVSTS            0001   I/O wait queue   empty
Charge PID         00000000

              I/O request queue
              -----------------

STATE    IRP      PID     MODE CHAN  FUNC  WCB     EFN  AST      IOSB     STATUS

C    8024DC20  00010006  U    FFD0  0021 00000000 0 000025F8 00001004 0403
     readlblk bufio,func,mbxio


MBA2                                    Unknown          UCB address:  800015D0

Device status:     00000110 online,bsy
Characteristics:   0C150001 rec,shr,avl,mbx,idv,odv
                   00000200 nnm

Owner UIC [000001,000004]  Operation count     12   ORB address   80001660
      PID          00000000  Error count          0   DDB address   800013BC
Class/Type            A0/00  Reference count      2   DDT address   800019AC
Def. buf. size         2560  BOFF              0000   CRB address   80001814
DEVDEPEND          00000000  Byte count        0A00   IRP address   802509A0
DEVDEPND2          00000000  SVAPTE        00000000   Fork R4       00000014
FIPL/DIPL             0B/0B  DEVSTS            0001   I/O wait queue   empty
Charge PID         00000000

              I/O request queue
              -----------------

STATE  IRP     PID  MODE  CHAN  FUNC   WCB      EFN   AST      IOSB     STATUS

C    802509A0 00010005 U    FFD0  0021  00000000   3  00000000 7FF9D5A4  0403
     readlblk bufio,func,mbxio
```

```
MBA3                                        MBX            UCB address:  80154B30

Device status:    00000010 online
Characteristics:  0C150001 rec,shr,avl,mbx,idv,odv
                  00000200 nnm

Owner UIC [000010,000040]   Operation count        0   ORB address   80154BC0
      PID         00000000  Error count            0   DDB address   800013BC
Class/Type           A0/01  Reference count        0   DDT address   800019AC
Def. buf. size          18  BOFF                0000   CRB address   80001814
DEVDEPEND         00000000  Byte count          0000   LNM address   80112BE0
DEVDEPND2         00000000  SVAPTE          00000000   I/O wait queue   empty
FIPL/DIPL            0B/0B  DEVSTS              0001
Charge PID        00000000

    *** I/O request queue is empty ***

MBA4                                        MBX            UCB address:  80162C90

Device status:    00000110 online,bsy
Characteristics:  0C150001 rec,shr,avl,mbx,idv,odv
                  00000200 nnm

Owner UIC [000001,000004]   Operation count      117   ORB address   80162D20
      PID         00000000  Error count            0   DDB address   800013BC
Class/Type           A0/01  Reference count        4   DDT address   800019AC
Def. buf. size         150  BOFF                0000   CRB address   80001814
DEVDEPEND         00000000  Byte count          0096   LNM address   8013FAA0
DEVDEPND2         00000000  SVAPTE          00000000   IRP address   8023E6C0
FIPL/DIPL            0B/0B  DEVSTS              0002   I/O wait queue empty
Charge PID        00010009
                                  I/O request queue
                                  -----------------

STATE IRP      PID      MODE  CHAN  FUNC  WCB       EFN  AST       IOSB      STATUS

  C 8023E6C0 00010009   K     FFC0  0021  00000000  2    0000C581  0000A480  0403
       readlblk bufio,func,mbxio
```

```
MBA5                                            MBX            UCB address:  80163E70

Device status:     00000110 online,bsy
Characteristics:   0C150001 rec,shr,avl,mbx,idv,odv
                   00000200 nnm

Owner UIC [000001,000004]   Operation count      6   ORB address    80163F00
      PID          00000000   Error count          0   DDB address    800013BC
Class/Type            A0/01   Reference count      2   DDT address    800019AC
Def. buf. size          256   BOFF              0000   CRB address    80001814
DEVDEPEND          00000000   Byte count        0040   IRP address    80250660
DEVDEPND2          00000000   SVAPTE        00000000   I/O wait queue   empty
FIPL/DIPL             0B/0B   DEVSTS            0002
Charge PID         0001000A
                                 I/O request queue
                                 -----------------

STATE IRP       PID      MODE CHAN  FUNC   WCB      EFN  AST       IOSB      STATUS

  C   80250660 0001000A  U  FFB0  0021  00000000   2   00001C8E 00001680  0403
         readlblk bufio,func,mbxio


MBA24                                     MBX                UCB address:  80165010

Device status:     00000110 online,bsy
Characteristics:   0C150001 rec,shr,avl,mbx,idv,odv
                   00000200 nnm

Owner UIC [000100,000025]   Operation count      0   ORB address    801650A0
      PID          00000000   Error count          0   DDB address    800013BC
Class/Type            A0/01   Reference count      2   DDT address    800019AC
Def. buf. size           64   BOFF              0000   CRB address    80001814
DEVDEPEND          00000000   Byte count        0040   IRP address    8024E920
DEVDEPND2          00000000   SVAPTE        00000000   I/O wait queue   empty
FIPL/DIPL             0B/0B   DEVSTS            0002
Charge PID         0001000F

                   I/O request queue
                   -----------------

STATE IRP        PID     MODE CHAN  FUNC    WCB      EFN  AST       IOSB      STATUS

  C   8024E920  0001000F U  FFC0   0021  00000000   3  00003457 00015030   0403
   readlblk bufio,func,mbxio
```

2.6 Looking at RT Devices

2.6.1 SDA> SHOW DEVICE RT

might be opposite on production system

old style

```
                                DDB list
                                --------
   Address      Controller     ACP         Driver       DPT     DPT size
   -------      ----------     ---         ------       ---     --------
   802B3B00      RTA           REMACP      RTTDRIVER  80165A60   0A20
   802B4160      RTB           REMACP      CTDRIVER   80166710   1AB0

Controller: RTA
---------------
                --- Device Data Block (DDB) 802B3B00 ---
Driver name     RTTDRIVER   Alloc. class      0   DDT address     80165AEC
ACP ident             REM   SB address  80000EF4
ACP class            SLOW   UCB address 80165350

          ---  Primary Channel Request Block (CRB) 802B3BC0 ---
Reference count        2    Wait queue          empty
IDB address   802B3CE0                             Int. service    80165FBF

                --- Interrupt Data Block (IDB) 802B3CE0 ---
CSR address  00000000    Owner UCB addr. 00000000    ADP address  00000000
Number of units     8

                --- Driver Dispatch Table (DDT) 80165AEC ---
Errlog buf sz       0    Diag buf sz            0    FDT size           64
Start I/O      return    Register dump     return    FDT address  80165B24
Alt start I/O  return    Unit init         return    Mnt verify   8000CF43
Cancel I/O   80166052    Unsol int       801660F8    Cloned UCB     return

RTA0                                   Unknown          UCB address:  80165350

Device status:    00000000
Characteristics:  0C040007 rec,ccl,trm,avl,idv,odv
                  00000204 rtt,nnm
Owner UIC [000001,000004]  Operation count      0    ORB address 80165488
      PID         00000000  Error count          0    DDB address 802B3B00
Class/Type           42/00  Reference count      0    DDT address 80165AEC
Def. buf. size          80  BOFF              0000    CRB address 802B3BC0
DEVDEPEND         180012A0  Byte count        0000    I/O wait queue empty
DEVDEPND2         00000000  SVAPTE        00000000
FIPL/DIPL            08/08  DEVSTS            0000
        *** I/O request queue is empty ***

                --- Volume Control Block (VCB) 80164230 ---
```

```
RTA1                                  LA24          UCB address:  8016A6B0

Device status:    00000010 online
Characteristics:  0C0C0007 rec,ccl,trm,avl,mnt,idv,odv
                  00000204 rtt,nnm
Owner UIC [000001,000004]   Operation count     318   ORB address 8016A7E8
      PID         00010017  Error count           0   DDB address 802B3B00
Class/Type           42/25  Reference count       2   DDT address 801667A4
Def. buf. size         132  BOFF               0000   VCB address 80164230
DEVDEPEND         420803A0  Byte count         0000   CRB address 802B3BC0
DEVDEPND2         00800000  SVAPTE         00000000   I/O wait queue empty
FIPL/DIPL            08/08  DEVSTS             0001
        *** I/O request queue is empty ***

                --- Volume Control Block (VCB) 80164230 ---

Controller: RTB
---------------
                --- Device Data Block (DDB) 802B4160 ---
Driver name           CTDRIVER  Alloc. class      0  DDT address 801667A4
ACP ident                  REM  SB address     80000EF4
ACP class                 SLOW  UCB address    80164D90

            ---  Primary Channel Request Block (CRB) 802B44C0 ---
Reference count       1     Wait queue          empty
IDB address     802B4400                          Int. service    80167139

                --- Interrupt Data Block (IDB) 802B4400 ---
CSR address     00000000    Owner UCB addr. 00000000 ADP address   00000000
Number of units        8
Errlog buf sz          0    Diag buf sz         0    FDT size            64
Start I/O         return    Register dump  return    FDT address   801667DC
Alt start I/O     return    Unit init      return    Mnt verify    8000CF43
Cancel I/O      801672B9    Unsol int    801675C3    Cloned UCB      return

RTB0                              Unknown          UCB address:  80164D90

Device status:    00000800 valid
Characteristics:  0C040007 rec,ccl,trm,avl,idv,odv
                  00000004 rtt
Owner UIC [000001,000004]   Operation count       0   ORB address 80164EC8
      PID         00000000  Error count           0   DDB address 802B4160
Class/Type           42/00  Reference count       0   DDT address 801667A4
Def. buf. size          80  BOFF               0000   CRB address 802B44C0
DEVDEPEND         180012A0  Byte count         0000   I/O wait queue empty
DEVDEPND2         00000000  SVAPTE         00000000
FIPL/DIPL            08/08  DEVSTS             0000
        *** I/O request queue is empty ***
```

2.7 Looking at Specific UCBX

2.7.1 UCB for NETDRIVER

```
SDA> FORMAT  80162020                        !UCB for NETO

80162020    UCB$L_FQFL                        0000003F
            UCB$L_RQFL
            UCB$W_MB_SEED
            UCB$W_UNIT_SEED
80162024    UCB$L_FQBL                        00000000
            UCB$L_RQBL
80162028    UCB$W_SIZE                            0090
8016202A    UCB$B_TYPE                          10
8016202B    UCB$B_FIPL                        08
8016202C    UCB$L_ASTQFL                      8016202C
            UCB$L_FPC
            UCB$T_PARTNER
80162030    UCB$L_ASTQBL                      8016202C
            UCB$L_FR3
80162034    UCB$L_FIRST                       00000000
            UCB$L_FR4
            UCB$W_MSGMAX
            UCB$W_MSGCNT
80162038    UCB$W_BUFQUO                          0000
            UCB$W_DSTADDR
8016203A    UCB$W_SRCADDR                     0000
8016203C    UCB$L_ORB                         801620B0
80162040    UCB$L_CPID                        00000000  --> start w/ 8
            UCB$L_LOCKID
80162044    UCB$L_CRB                         802A7920
80162048    UCB$L_DDB                         802A78C0
8016204C    UCB$L_PID                         00000000
80162050    UCB$L_LINK                        80162D80
80162054    UCB$L_VCB                         80162E90
80162058    UCB$L_DEVCHAR                     0C1C2000
            UCB$Q_DEVCHAR
8016205C    UCB$L_DEVCHAR2                    00000000
80162060    UCB$B_DEVCLASS                          00
80162061    UCB$B_DEVTYPE                         00
80162062    UCB$W_DEVBUFSIZ                   0100      --> Known value
80162064    UCB$B_LOCSRV                            01
            UCB$B_SECTORS
            UCB$L_DEVDEPEND
            UCB$L_JNL_SEQNO
            UCB$Q_DEVDEPEND
```

```
80162065   UCB$B_REMSRV                 00
           UCB$B_TRACKS
80162066   UCB$W_BYTESTOGO            0000
           UCB$W_CYLINDERS
           UCB$B_VERTSZ
80162068   UCB$L_DEVDEPND2            00000000
           UCB$L_TT_DEVDP1
8016206C   UCB$L_IOQFL                8016206C
80162070   UCB$L_IOQBL                8016206C
80162074   UCB$W_UNIT                     0000
80162076   UCB$B_CM1                    00
           UCB$W_CHARGE
           UCB$W_RWAITCNT
80162077   UCB$B_CM2                  00
80162078   UCB$L_IRP                  00000000
8016207C   UCB$W_REFC                     0000
8016207E   UCB$B_DIPL                   08
           UCB$B_STATE
8016207F   UCB$B_AMOD                 00
80162080   UCB$L_AMB                  00000000
80162084   UCB$L_STS                  00002010
           UCB$W_STS
80162088   UCB$W_DEVSTS                   0000
8016208A   UCB$W_QLEN                 0000
8016208C   UCB$L_DUETIM               00000000
80162090   UCB$L_OPCNT                00000000
80162094   UCB$L_LOGADR               00000000
           UCB$L_SVPN
80162098   UCB$L_SVAPTE               00000000
8016209C   UCB$W_BOFF                     0000
8016209E   UCB$W_BCNT                 0000
801620A0   UCB$B_ERTCNT                     00
801620A1   UCB$B_ERTMAX                   00
801620A2   UCB$W_ERRCNT               0000
801620A4   UCB$L_JNL_MCSID            00000000
           UCB$L_PDT
801620A8   UCB$L_DDT                  8015E5B0
801620AC   UCB$L_MEDIA_ID             00000000
           UCB$C_LENGTH
```

2.7.2 UCB for NETACP

Force it to be a UCB

```
SDA> FORMAT  80162D80/TYPE=UCB        !UCB for NET1

80162D80    UCB$L_FQFL                80162D80
            UCB$L_RQFL
            UCB$W_MB_SEED
            UCB$W_UNIT_SEED
80162D84    UCB$L_FQBL                80162D80
            UCB$L_RQBL
80162D88    UCB$W_SIZE                    0090
80162D8A    UCB$B_TYPE                  10
80162D8B    UCB$B_FIPL                08
80162D8C    UCB$L_ASTQFL              00000000
            UCB$L_FPC
            UCB$T_PARTNER
80162D90    UCB$L_ASTQBL              00000000
            UCB$L_FR3
80162D94    UCB$L_FIRST               00000000
            UCB$L_FR4
            UCB$W_MSGMAX
            UCB$W_MSGCNT
80162D98    UCB$W_BUFQUO                  0000
            UCB$W_DSTADDR
80162D9A    UCB$W_SRCADDR             0000
80162D9C    UCB$L_ORB                 80162E10
80162DA0    UCB$L_CPID                00010009
            UCB$L_LOCKID
80162DA4    UCB$L_CRB                 802A7920
80162DA8    UCB$L_DDB                 802A7BC0
80162DAC    UCB$L_PID                 00010009
80162DB0    UCB$L_LINK                80163C60
80162DB4    UCB$L_VCB                 80162E90
80162DB8    UCB$L_DEVCHAR             0C1C2000
            UCB$Q_DEVCHAR
80162DBC    UCB$L_DEVCHAR2            00000000
80162DC0    UCB$B_DEVCLASS                  00
80162DC1    UCB$B_DEVTYPE                 00
80162DC2    UCB$W_DEVBUFSIZ           0100
80162DC4    UCB$B_LOCSRV                    01
            UCB$B_SECTORS
            UCB$L_DEVDEPEND
            UCB$L_JNL_SEQNO
            UCB$Q_DEVDEPEND
```

80162DC5	UCB$B_REMSRV	00
	UCB$B_TRACKS	
80162DC6	UCB$W_BYTESTOGO	0000
	UCB$W_CYLINDERS	
	UCB$B_VERTSZ	
80162DC8	UCB$L_DEVDEPND2	00000000
	UCB$L_TT_DEVDP1	
80162DCC	UCB$L_IOQFL	80162DCC
80162DD0	UCB$L_IOQBL	80162DCC
80162DD4	UCB$W_UNIT	0001
80162DD6	UCB$B_CM1	90
	UCB$W_CHARGE	
	UCB$W_RWAITCNT	
80162DD7	UCB$B_CM2	00
80162DD8	UCB$L_IRP	00000000
80162DDC	UCB$W_REFC	0001
80162DDE	UCB$B_DIPL	08
	UCB$B_STATE	
80162DDF	UCB$B_AMOD	00
80162DE0	UCB$L_AMB	80162C90
80162DE4	UCB$L_STS	00010010
	UCB$W_STS	
80162DE8	UCB$W_DEVSTS	0002
80162DEA	UCB$W_QLEN	0000
80162DEC	UCB$L_DUETIM	00000000
80162DF0	UCB$L_OPCNT	00000000
80162DF4	UCB$L_LOGADR	00000000
	UCB$L_SVPN	
80162DF8	UCB$L_SVAPTE	00000000
80162DFC	UCB$W_BOFF	0000
80162DFE	UCB$W_BCNT	0000
80162E00	UCB$B_ERTCNT	00
80162E01	UCB$B_ERTMAX	00
80162E02	UCB$W_ERRCNT	0000
80162E04	UCB$L_JNL_MCSID	00000000
	UCB$L_PDT	
80162E08	UCB$L_DDT	8015E5B0
80162E0C	UCB$L_MEDIA_ID	00000000
	UCB$C_LENGTH	

2.7.3 UCB for FAL

```
SDA> FORMAT  801654E0                        !UCB for  NET30

801654E0    UCB$L_FQFL                      801654E0
            UCB$L_RQFL
            UCB$W_MB_SEED
            UCB$W_UNIT_SEED
801654E4    UCB$L_FQBL                      801654E0
            UCB$L_RQBL
801654E8    UCB$W_SIZE                          0090
801654EA    UCB$B_TYPE                        10
801654EB    UCB$B_FIPL                      08
801654EC    UCB$L_ASTQFL                    00000000
            UCB$L_FPC
            UCB$T_PARTNER
801654F0    UCB$L_ASTQBL                    00000000
            UCB$L_FR3
801654F4    UCB$L_FIRST                     00000000
            UCB$L_FR4
            UCB$W_MSGMAX
            UCB$W_MSGCNT
801654F8    UCB$W_BUFQUO                        0000
            UCB$W_DSTADDR
801654FA    UCB$W_SRCADDR                   0000
801654FC    UCB$L_ORB                       80165570
80165500    UCB$L_CPID                      0001000F
            UCB$L_LOCKID
80165504    UCB$L_CRB                       802A7920
80165508    UCB$L_DDB                       802A7BC0
8016550C    UCB$L_PID                       0001000F
80165510    UCB$L_LINK                      801655D0
80165514    UCB$L_VCB                       80162E90
80165518    UCB$L_DEVCHAR                   0C1C2000
            UCB$Q_DEVCHAR
8016551C    UCB$L_DEVCHAR2                  00000000
80165520    UCB$B_DEVCLASS                        00
80165521    UCB$B_DEVTYPE                       00
80165522    UCB$W_DEVBUFSIZ                 0100
80165524    UCB$B_LOCSRV                          01
            UCB$B_SECTORS
            UCB$L_DEVDEPEND
            UCB$L_JNL_SEQNO
            UCB$Q_DEVDEPEND
```

```
80165525    UCB$B_REMSRV                00
            UCB$B_TRACKS
80165526    UCB$W_BYTESTOGO         0000
            UCB$W_CYLINDERS
            UCB$B_VERTSZ
80165528    UCB$L_DEVDEPND2         00000000
            UCB$L_TT_DEVDP1
8016552C    UCB$L_IOQFL             8016552C
80165530    UCB$L_IOQBL             8016552C
80165534    UCB$W_UNIT                  001E
80165536    UCB$B_CM1                 90
            UCB$W_CHARGE
            UCB$W_RWAITCNT
80165537    UCB$B_CM2               00
80165538    UCB$L_IRP               00000000
8016553C    UCB$W_REFC                  0001
8016553E    UCB$B_DIPL                08
            UCB$B_STATE
8016553F    UCB$B_AMOD              00
80165540    UCB$L_AMB               80165010
80165544    UCB$L_STS               00010010
            UCB$W_STS
80165548    UCB$W_DEVSTS                0002
8016554A    UCB$W_QLEN              0000
8016554C    UCB$L_DUETIM            00000000
80165550    UCB$L_OPCNT             00000007
80165554    UCB$L_LOGADR            00000000
            UCB$L_SVPN
80165558    UCB$L_SVAPTE            802AAEC0
8016555C    UCB$W_BOFF                  0058
8016555E    UCB$W_BCNT              0005
80165560    UCB$B_ERTCNT                  00
80165561    UCB$B_ERTMAX                00
80165562    UCB$W_ERRCNT            0000
80165564    UCB$L_JNL_MCSID         00000000
            UCB$L_PDT
80165568    UCB$L_DDT               8015E5B0
8016556C    UCB$L_MEDIA_ID          00000000
            UCB$C_LENGTH
```

2.8 Using SDA to Look at Routing Data Structures

2.8.1 Routing Control Block

```
SDA> FORMAT  80162E90/TYPE=RCB

80162E90   RCB$Q_IRP_FREE         8024CD80
80162E94                          80250180
80162E98   RCB$W_SIZE                 0000
80162E9A   RCB$B_TYPE                11      ----> this is actually type for VCB
80162E9B   RCB$B_STATUS           02
80162E9C   RCB$W_TRANS                0006
80162E9E   RCB$W_ADDR             0402
80162EA0   RCB$L_AQB              80162E70
80162EA4   RCB$L_ACP_UCB          80162D80
80162EA8   RCB$L_PTR_JNX          00000000
80162EAC   RCB$L_PTR_OA           802A62AC
80162EB0   RCB$L_PTR_AOA          00000000   not area router
80162EB4   RCB$L_PTR_LTB          80162FF0
80162EB8   RCB$L_PTR_LPD          802A9128
80162EBC   RCB$L_PTR_ADJ          801634D8
80162EC0   RCB$L_PTR_TQE          80162FC0
80162EC4   RCB$L_PTR_NDC          80163120   if not debug will = 0
80162EC8   RCB$L_PTR_DCS          00000000
80162ECC   RCB$L_PTR_AREG         00000000   not alias in cluster
80162ED0   RCB$Q_LOC_RCV          8024AB60
80162ED4                          8024AB60
80162ED8   RCB$Q_LOC_XMT          80162ED8
80162EDC                          80162ED8
80162EE0   RCB$Q_IRP_WAIT         80162EE0
80162EE4                          80162EE0
80162EE8   RCB$W_MCOUNT               0006
80162EEA   RCB$W_CUR_LLNK         0005
80162EEC   RCB$W_MAX_LNK              0040
80162EEE   RCB$W_MAX_ADDR         000F
80162EF0   RCB$B_MAX_LPD                0B   11 circuits (10 circ + 1 internal)
80162EF1   RCB$B_MAX_SNK              0B
80162EF2   RCB$B_MAX_VISIT          0A
80162EF3   RCB$B_INT_PTH          00
80162EF4   RCB$B_ACT_DLL                02
80162EF5   RCB$B_STI                  01
80162EF6   RCB$B_ECL_RFLW           08     }
80162EF7   RCB$B_ECL_RFA          0A       }
80162EF8   RCB$B_ECL_DFA                05 }  timing of
80162EF9   RCB$B_ECL_DWE              05   }  ECL logical links
80162EFA   RCB$B_ECL_DAC            03     }
80162EFB   RCB$B_ECL_DPX          03       }
```

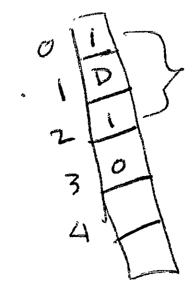
```
80162EFC   RCB$W_MAX_ADJ            006B
80162EFE   RCB$W_MAX_RTG        002B
80162F00   RCB$L_DLE_XWB        00000000
80162F04   RCB$W_TIM_RSI            0000
80162F06   RCB$W_TIM_RTI        0000
80162F08   RCB$W_TIM_IAT            003C
80162F0A   RCB$W_TIM_CNI        002D
80162F0C   RCB$W_TIM_CNO            002D
80162F0E   RCB$W_TIM_CTI        0000
80162F10   RCB$W_ECLSEGSIZ          0231
80162F12   RCB$W_TOTBUFSIZ      028C
80162F14   RCB$W_CUR_PKT            0003
80162F16   RCB$W_MAX_PKT        000F
80162F18   RCB$W_PKT_FREE           0000
80162F1A   RCB$W_PKT_PEAK       0000
80162F1C   RCB$B_PKT_FAIL             00
80162F1D   RCB$B_MEM_FAIL           00
80162F1E   RCB$B_ETY              04
80162F1F   RCB$B_HOMEAREA       01
80162F20   RCB$B_MAX_AREA             3F
80162F21   RCB$B_ACT_TIMER          17
80162F22   RCB$W_ALIAS          0000
80162F24   RCB$W_MAX_ALNK           0020
80162F26   RCB$W_MAX_LLNK       0020
80162F28   RCB$W_CUR_ALNK           0000
80162F2A   RCB$L_ABS_TIM        012B
80162F2C                            0000
80162F2E   RCB$B_CNT_1ST          00
           RCB$B_CNT_NOL
80162F2F   RCB$B_CNT_APL        00
80162F30   RCB$B_CNT_OPL              00
80162F31   RCB$B_CNT_PFE            00
80162F32   RCB$B_CNT_RUL          00
80162F33   RCB$B_CNT_VER        00
80162F34   RCB$W_CNT_NUL            0000
80162F36   RCB$W_CNT_XRE        0000
80162F38   RCB$W_CNT_MLL            0007
80162F3A   RCB$Q_CXB_FREE       2F3A
80162F3C                        2F3A8016
80162F40                            8016
80162F42   RCB$B_LSN_ADJ          00
80162F43   RCB$B_AQB_CNT        00
80162F44   RCB$W_DRT                0000
80162F46   RCB$W_LVL2           0000
           RCB$C_LENGTH
```

2.8.2 Output Adjacency

```
SDA> EXAMINE   802A62AC;20                  !OUTPUT ADJACENCY (MAX ADDRESS=15)
0000000C 000E000F 00000001 000D0001  ................       802A62AC
00000000 00000000 00000000 00000000  ................       802A62BC
```

2.8.3 Looking at Specific Node Entries

```
SDA> EXAMINE   802A62AC         !node 0       (local node)
802A62AC:  000D0001    "...."

SDA> EXAMINE   802A62AC + 2     !SPLASH=1.1  (with 2 bytes per entry)
802A62AE:  0001000D    "...."

SDA> EXAMINE   802A62AC + 4     !THUD=1.2     (local node)
802A62B0:  00000001    "...."

SDA> EXAMINE   802A62AC + 6     !ZIP=1.3      (unreachable node)
802A62B2:  000F0000    "...."

SDA> EXAMINE   802A62AC + 8     !BAROOM=1.4
802A62B4:  000E000F    "...."

SDA> EXAMINE   802A62AC + A     !CLICK=1.5
802A62B6:  000C000E    "...."
```



2.8.4 Adjacency Index Table

```
SDA> EXAMINE  801634D8;50          !ADJ INDEX TABLE  (from RCB)
801636BA 801636AD 801636A0 0017078D  ................    801634D8
801636EE 801636E1 801636D4 801636C7  ................    801634E8
80163722 80163715 80163708 801636FB  ................    801634F8
80163756 80163749 8016373C 8016372F  ................    80163508
8016378A 8016377D 80163770 80163763  ................    80163518

SDA> EXAMINE 801634D8+34    !OFFSET D*4(SPLASH) INTO TABLE FOR ADJ ADDRESS
80163508:   8016373C     "/7.."

SDA> EXAMINE 801634D8+38    !OFFSET E*4 (CLICK) INTO TABLE FOR ADJ ADDRESS
80163510:   80163749     "I7.."

SDA> FORMAT  8016373C /TYPE=ADJ

8016372F    ADJ$B_STS                 0F
80163730    ADJ$B_PTYPE             04
80163731    ADJ$B_LPD_INX         03
            ADJ$W_LPD
80163732    ADJ$B_LPD_SEQ       02
80163733    ADJ$W_PNA               0401
80163735    ADJ$W_BUFSIZ        05DA
80163737    ADJ$W_INT_LSN           002D
80163739    ADJ$W_TIM_LSN       0027
8016373B    ADJ$B_BCPRI               40
            ADJ$C_LENGTH

SDA> FORMAT  80163749 /TYPE=ADJ

80163749    ADJ$B_STS                 0F
8016374A    ADJ$B_PTYPE             04
8016374B    ADJ$B_LPD_INX         03
            ADJ$W_LPD
8016374C    ADJ$B_LPD_SEQ       02
8016374D    ADJ$W_PNA               0405
8016374F    ADJ$W_BUFSIZ        0240
80163751    ADJ$W_INT_LSN           002D
80163753    ADJ$W_TIM_LSN       0021
80163755    ADJ$B_BCPRI               40
            ADJ$C_LENGTH
```

2.8.5 Logical Path Descriptor

```
SDA> EXAMINE  802A9128;50                   !LPD TABLE INDEX (from RCB)

80243E80 80244290 80162F50 00170050  ................     802A9128
00000107 00000106 00000105 00000104  ................     802A9138
0000010B 0000010A 00000109 00000108  ................     802A9148
0000010F 0000010E 0000010D 0000010C  ................     802A9158
010B7324 802A7D40 00000111 00000110  ................     802A9168


SDA> EXAMINE  802A9128+C           !LPD TABLE INDEXED BY 3*4 BYTES
802A9134:   80243E80    "...."     !use ADJ$B_LPD_INX (cicuit is UNA-0)



SDA> FORMAT  80243E80/TYPE=LPD       !for UNA-0

80243E80    LPD$Q_REQ_WAIT           80243E80
80243E84                             80243E80
80243E88    LPD$W_SIZE                   006A
80243E8A    LPD$B_TYPE                 17
80243E8B    LPD$B_STARTUPS           01
80243E8C    LPD$L_WIND               00000000
80243E90    LPD$L_UCB                80164320
80243E94    LPD$W_CHAN                   0090
80243E96    LPD$W_TIM_TLK            000A
80243E98    LPD$W_INT_TLK                000F
80243E9A    LPD$B_TSTCNT               00
80243E9B    LPD$B_ASTCNT             01
80243E9C    LPD$B_IRPCNT                   01
80243E9D    LPD$B_ETV                    04
80243E9E    LPD$B_XMT_SRL              08
80243E9F    LPD$B_XMT_IPL            06
80243EA0    LPD$B_PTH_INX                  03
            LPD$W_PTH
80243EA1    LPD$B_PTH_SEQ                02
80243EA2    LPD$W_STS                2471
80243EA4    LPD$B_XMTFLG                   00
80243EA5    LPD$B_PVCFLG                 00
80243EA6    LPD$B_STI                  09
80243EA7    LPD$B_SUB_STA            0A
80243EA8    LPD$B_PLVEC                    7F
80243EA9    LPD$B_COST                   01
80243EAA    LPD$B_BCPRI                40
80243EAB                             00
```

80243EAC	LPD$W_DRT	000C
80243EAE	LPD$L_RTR_LIST	462C
80243EB0		8016
80243EB2	LPD$L_RCV_IRP	0000
80243EB4		0000
80243EB6	LPD$L_ABS_TIM	013C
80243EB8		0000
80243EBA	LPD$B_CNT_1ST	E8
	LPD$L_CNT_APR	
80243EBB		02
80243EBC		0000
80243EBE	LPD$L_CNT_TPR	0000
80243EC0		0000
80243EC2	LPD$L_CNT_DPS	0293
80243EC4		0000
80243EC6	LPD$L_CNT_TPS	0000
80243EC8		0000
80243ECA	LPD$W_CNT_ACL	0000
80243ECC	LPD$W_CNT_TCL	0000
80243ECE	LPD$B_CNT_LDN	00
80243ECF	LPD$B_CNT_IFL	00
80243ED0	LPD$W_BUFSIZ	05DA
80243ED2	LPD$B_SRM_POS	13
80243ED3	LPD$B_SRM_LEFT	00
80243ED4	LPD$B_ASRM_POS	00
80243ED5	LPD$B_ASRM_LEFT	00
80243ED6	LPD$G_SRM	0000
80243ED8		0000
80243EDA	LPD$G_XMT_SRM	0000
80243EDC		0000
80243EDE	LPD$G_ASRM	FFFF
80243EE0		FFFF
80243EE2	LPD$G_XMT_ASRM	0000
80243EE4		0000
80243EE6	LPD$L_CACHE	0000
80243EE8		0000
	LPD$C_LENGTH	

if this was an endnode and there was a cache

2.9 Logical Link Data Structures

2.9.1 Link Tables

```
SDA> EXAMINE  80162FF0;120     !LTB table for logical links (from RCB)
                               !size = header (36) + maxlinks(64) * 4

003E0020 0020003E 00000000 80163068  ................   80162FF0
00000000 80165820 00000000 01240000  ................   80163000
04030001 04020001 04010001 00000000  ................   80163010
80165820 08060001 04050001 04040001  ................   80163020
040B0001 80169E90 08090001 80166480  ................   80163030
040F0001 040E0001 040D0001 80169BF0  ................   80163040
04130001 04120001 04110001 04100001  ................   80163050
00170001 00160001 8016A2A0 04140001  ................   80163060
001B0001 001A0001 00190001 00180001  ................   80163070
001F0001 001E0001 001D0001 001C0001  ................   80163080
00230001 00220001 00210001 00200001  ................   80163090
00270001 00260001 00250001 00240001  ................   801630A0
002B0001 002A0001 00290001 00280001  ................   801630B0
002F0001 002E0001 002D0001 002C0001  ................   801630C0
00330001 00320001 00310001 00300001  ................   801630D0
00370001 00360001 00350001 00340001  ................   801630E0
003B0001 003A0001 00390001 00380001  ................   801630F0
FFFFFFFF 003E0001 003D0001 003C0001  ................   80163100
```

Logical Link is open

```
SDA> FORMAT  80162FF0/TYPE=LTB

80162FF0   LTB$L_SLT_NXT              80163068
80162FF4   LTB$L_ASLT_NXT             00000000
80162FF8   LTB$W_SLT_TOT                  003E
80162FFA   LTB$W_SLT_LMT              0020
80162FFC   LTB$W_ASLT_LMT                 0020
80162FFE   LTB$W_LSLT_TOT             003E
80163000   LTB$W_ASLT_TOT                 0000
80163002   LTB$W_SIZE                 0124
80163004   LTB$B_TYPE                       00
80163005   LTB$B_SPARE_1                  00
80163006   LTB$W_SPARE_2              0000
80163008   LTB$L_XWB                  80165820
8016300C   LTB$L_PTR_ALIAS_SLOTS      00000000
80163010   LTB$L_SLOTS                00000000
           LTB$C_LENGTH
```

total logical links

alias logical links

because of formating this instead of examining Like above.

2.9.2 Extended (NETWORK) Window Block

```
SDA> FORMAT  80165820/TYPE=XWB

80165820    XWB$L_WLFL          80162E70
80165824    XWB$L_WLBL          80162E70
80165828    XWB$W_SIZE              017C
8016582A    XWB$B_TYPE                1C
8016582B    XWB$B_ACCESS        00
8016582C    XWB$W_REFCNT            0001
8016582E    XWB$W_STS           1000
80165830    XWB$L_ORGUCB        801654E0
80165834    XWB$Q_FORK          800029D0
80165838                        800029D0
8016583C    XWB$W_FLG               02E0
8016583E    XWB$B_STA                 05
8016583F    XWB$B_FIPL          08
80165840    XWB$L_FPC           801602DE
80165844    XWB$L_FR3           00000203
80165848    XWB$L_FR4           0000000D
8016584C    XWB$L_LINK          80166480
80165850    XWB$L_VCB           80162E90
80165854    XWB$L_PID           0001000F
80165858    XWB$W_PATH              0203
8016585A    XWB$W_REMNOD        0401
8016585C    XWB$W_REMLNK            201F
8016585E    XWB$W_LOCLNK        0407
80165860    XWB$W_LOCSIZ            05B5   --> doesn't include header
80165862    XWB$W_REMSIZ        05B5
80165864    XWB$W_R_REASON          0064
80165866    XWB$W_X_REASON      0064
80165868    XWB$W_TIM_ID            0039
8016586A    XWB$W_ELAPSE        0011
8016586C    XWB$W_TIM_INACT         001E
8016586E    XWB$W_DELAY         0001
80165870    XWB$W_TIMER             000C
80165872    XWB$W_PROGRESS      0000
80165874    XWB$W_RETRAN            000A
80165876    XWB$W_DLY_FACT      0005
80165878    XWB$W_DLY_WGHT          0005
8016587A    XWB$B_PRO                 19
8016587B    XWB$B_DATA          00
8016587C    XWB$T_DATA          ""
8016587D                        000000
80165880                        00000000
80165884                        00000000
80165888                        00000000
8016588C    XWB$B_X_FLW               00
```

```
8016588D   XWB$B_X_FLWCNT          00
8016588E   XWB$B_SP3             00
8016588F   XWB$B_RID         0C
80165890   XWB$T_RID
80165890                     544F4353
80165894                     20202054
80165898                     20202020
8016589C                     20202020
801658A0   XWB$L_IRP_ACC     00000000
801658A4                     00000000
801658A8                     00010000
801658AC                     00000001
801658B0                     00000038
801658B4                     00000052
801658B8                     00000077
801658BC                     00000077
801658C0                     00000000
801658C4   XWB$B_LPRNAM            03
           XWB$T_DT          "..."
801658C5   XWB$T_LPRNAM      ""
801658C6                     0003
801658C8                     00030003
801658CC                     60FD0003
801658D0                     00080001
801658D4                     00000000
801658D8   XWB$B_RPRNAM            00
801658D9   XWB$T_RPRNAM      ""
801658DA                     0000
801658DC                     00000000
801658E0                     8023C7E0
801658E4                     00000000
801658E8                     00030003
801658EC   XWB$B_LOGIN             00
801658ED   XWB$T_LOGIN       "......"
           XWB$T_LI
801658F5                     003900
801658F8                     00390039
801658FC                     00010039
80165900                     00000001
80165904                     00000000
80165908                     00000000
8016590C                     00000000
80165910                     00000000
```

```
80165914                                00000000
80165918                                003A003A
8016591C                                01000100
80165920                                801658C4
80165924     XWB$L_DEA_IRP              00000000
80165928                                00000000
8016592C     XWB$L_ICB                  00000000
80165930     XWB$W_CI_PATH                  0203
80165932                                0000
80165934                                00000000
80165938     XWB$Q_FREE_CXB             80165938
```

2.10 Network Counter Data Structures

2.10.1 Node Counter Block

1 per node in Adj

```
SDA> EXAMINE  80163120;100            !NODE counter block  (from RCB)

0000012B 001703A8 00000021 0000001C  ................       80163120
00000000 00000000 00000000 00000000  ................       80163130
00000000 0000012B 00000000 00000000  ................       80163140
00000000 00000000 00000000 00000000  ................       80163150
00000000 00000000 0000012B 00000000  ................       80163160
00000000 00000000 00000000 00000000  ................       80163170
00000000 00000000 00000000 0000012B  ................       80163180
0000012B 00000000 00000000 00000000  ................       80163190
00000000 00000000 00000000 00000000  ................       801631A0
00000000 0000012B 00000000 00000000  ................       801631B0
00000000 00000000 00000000 00000000  ................       801631C0
00000000 00000000 0000012B 00000000  ................       801631D0
00000000 00000000 00000000 00000000  ................       801631E0
00000000 00000000 00000000 0000012B  ................       801631F0
0000012B 00000000 00000000 00000000  ................       80163200
00000000 00000000 00000000 00000000  ................       80163210
```

2.11 Notes on the SDA Trace

1. The analysis was done with several logical links created to this node and to other nodes on the same Ethernet.

2. The READ commands read the Network Data Structures and VMS System Data Structures files. This allows easy formatting of data structures by SDA.

3. The SHOW SUMMARY command finds out how many processes were running when the system crashed.

4. The SHOW POOL/SUMMARY command gives information about structures called NET. These are DECnet data structures.

5. The SHOW DEVICE XM command gives information on the DMC/DMR type devices and the SHOW DEVICE XE gives information on the ETHERNET DEUNA type devices. The information that is of particular interest is the UCB addresses.

6. For the structures in the display, the header information gives the size and type of the data structure.

   Bytes 9 and 10 give the size of the data structure and byte 11 gives the type of data structure.

   The NET structures have type 17.

7. The SHOW DEVICE NET command displays the UCB address of NETDRIVER. There can be many UCBs for NETDRIVER depending on network activity.

   Multiple UCBs have been highlighted for this example. The reason some of the numbers appear high is that each time the ASSIGN command is used to create a UCB for a NET device the number is incremented by one.

8. The Routing Control Block (RCB) is pointed to by the VCB address of each of the NETn devices.

   From the RCB, most of the other data structures could be traced. The RCB address for this system is 80162E90.

9. The SHOW DEVICE MB command gives information about the mailbox drivers. The UCB values can be used to confirm the linkage between the NETDRIVER and MBDRIVER.

NOTES

1. **The UCB of NET1 is at 80162080. The mailbox (AMB address or UCB$L_AMB) is 80162C90.**

   This is the address of MBA4.

2. **NET1 is NETACP and XEA1 and XMA0 also use this as their mailbox (AMB address).**

10. An example of the FORMAT command is shown using the UCB address of NET0 (NETDRIVER).

    Each NETn device has the same value for the field which is the pointer to the RCB (UCB$L_VCB = 80162E90).

11. The key data structure (RCB) is formatted.

    The fields of interest are:

RCB$L_PTR_OA	Output Adjaceny vector
RCB$L_PTR_LTB	Logical link table
RCB$L_PTR_LPD	Logical path descriptor
RCB$L_PTR_ADJ	Adjaceny node database block
RCB$L_PTR_NDC	Node counter block

12. The Output Adjaceny vector (OA) is examined.

    The numbers are indices to entries in the Adjacency Node Database Block (ADJ).

NOTE

These numbers are in HEX so 0015 points to entry 21, 0016 to entry 22 and so on.

13. To check whether the destination node is reachable, the OA vector is used. If the destination node address is 5, for example, the fifth entry is checked.

    If the fifth entry contains zero, the destination is not reachable. An error message is returned to the source program.

14. If the fifth entry is nonzero, it contains an offset to the ADJ. The ADJ identifies the logical path (circuit) on which to send the connect initiate message.

15. The Adjaceny node database block (ADJ) is examined here.

16. The addresses expanded are for node 1.1 (SPLASH) and 1.5 (CLICK).

17. Information of interest includes the node type (PTYPE), physical node address (PNA), buffer size (BUFSIZ) and (Broadcast) Router Priority (BCPRI).

```
Node type 3 is a Phase IV Area Routing Node
Node type 4 is a Phase IV Routing Node
Buffer size 05DA = 1498 bytes.


PNA = area * 1024 + node

1.1  = 1024 + 1 = 1025 = 401 (HEX)
1.5  = 1024 + 5 = 1029 = 405 (HEX)


Buffer size for SPLASH = 5DA (HEX) = 1498
   (VAX on Ethernet)
Buffer size for CLICK  = 240 (HEX) =  576
   (PDP on Ethernet)
```

18. The Logical Path Descriptor (LPD) table is examined.

19. The LPD for the circuit used (UNA-0) is obtained by indexing into the table to ADJ$B_LPD_INX * 4 BYTES PER ENTRY. ADJ$B_LPD_INX is from the ADJ and it is 3 for both nodes examined, as they are on the same circuit.

20. The Logical Path Descriptor (LPD) contains the UCB of the physical device.

21. To demonstrate the connection between the LPD and NETDRIVER, note that the LPD$L_UCB field points to the UCB of the XEA1.

    LPD$L_UCB will contain a UCB for the device used on that circuit (XM, XE, etc.).

    NOTE

    LPD$L_UCB = 80164320 which is the UCB of XEA1.

22. The Logical Link Table (LTB) is examined, and LTB$L_XWB points to the XWB describing the first logical link.

23. This XWB (@80165820) has been expanded.

    NOTE

    XWB$L_ORGUCB = 801654E0 which points to the UCB of NET30 (FAL). The local link (407 HEX = 1031) and remote link number (201F HEX = 8223) also point to local process FAL_1031.

24. The Node Counter block is examined. After the 12 byte header, it contains information about the local node (node 0) and then the rest of the nodes.

25. Information in the Node Counter block is updated as network activity occurs.

MAJOR NETWORK MECHANISMS

INTRODUCTION

Some major DECnet mechanisms are reviewed. The flow of the actions are traced and the routines executed are referenced.

Topics include:

- Logical Link Creation
- NSP Flow Control Mechanisms
- Sending and Receiving Normal Data and Interrupt Data
- Routing Table Update
- Other Network Mechanisms

1 CREATING A LOGICAL LINK

1.1 Operating Sequence of a Remote Network Access

1. DCL command (for example, COPY) calls RMS on the local node

2. RMS calls DECnet on the local node

3. DECnet calls NETDRIVER which calls NETACP on the remote node

4. NETACP creates the process on the remote node using access control information or proxy (runs SYS$SYSTEM:LOGINOUT.EXE in the context of this process)

   - SYS$SYLOGIN: EXECUTES

   - SYS$LOGIN:LOGIN.COM OR /LGICMD EXECUTES

   - F$MODE() = "NETWORK"

   - SYS$OUTPUT = SYS$LOGIN:NETSERVER.LOG

   - SYS$INPUT = NETSERVER.COM

   - SYS$SYSTEM:NETSERVER.COM runs SYS$SYSTEM:NETSERVER.EXE

   - NETSERVER.EXE calls LIB$DO_COMMAND or LIB$RUN_PROGRAM for the inbound connect request

     (FAL, MAIL, PHONE, user-specified procedure or image)

   - Returns to NETSERVER.COM which executes NETSERVER.EXE again (in the context of the current process)

   - Wait for NETSERVER$TIMEOUT for another inbound connect request

1.2 Sample DECnet-VAX Operation (Creating a Logical Link)

- Logical links are established by an interaction of DECnet calls specified in two cooperating programs (SOURCE and TARGET)
- The handshake procedure takes place in a prescribed order
- Once established, there is no distinction between SOURCE and TARGET
- The logical link is logically full-duplex!
- The logical link can be considered to comprise two separate data subchannels, each carrying messages in both directions

```
SENDING NORMAL DATA
===================

SYS$QIOW(,%VAL(NET_CHAN),%VAL(IO$_WRITEVBLK),IOSB,,,
1              DATA_ARRAY,%VAL(512),,,,)


SENDING INTERRUPT DATA
======================

SYS$QIOW(,%VAL(NET_CHAN),
1              %VAL(IO$_WRITEVBLK.OR.IO$M_INTERRUPT),IOSB,,,
2              INT_MESSAGE,%VAL(16),,,,)
```

- For a program to receive and process interrupt data, that program (SOURCE or TARGET) must create a mailbox and tie it to the network channel

```
SYS$CREMBX(,MBX_CHAN,,,,,'NET_MBX')

SYS$ASSIGN('_NET:'NET_CHAN,'NET_MBX',)
```

1.3 Basic Steps in Nontransparent Task-to-Task Communication

SOURCE	TARGET
======	======

1. Obtain network channel number

```
SYS$ASSIGN('_NET:',SRC_CHAN,,)
```

2. Request connect with TARGET using Network Connect Block (NCB)

```
SYS$QIOW(,%VAL(SRC_CHAN),%VAL(IO$_ACCESS),
1           IOSB,,,,NCB_DESC,,,,)
```

3. Image run in process created due to connect request by SOURCE

4. Obtain network channel number

```
SYS$ASSIGN('_NET:',TGT_CHAN,,)
```

5. Translate SYS$NET to obtain NCB

```
SYS$TRNLNM(,'LNM$FILE_DEV','SYS$NET',,Items)
```

6. Accept/reject the connection with SOURCE

```
SYS$QIOW(,%VAL(TGT_CHAN),%VAL(IO$_ACCESS),
1           IOSB,,,,NCB,,,,)
```

7. SOURCE and TARGET may Send and Receive data - coordinating access

```
SYS$QIOW(,%VAL(TGT_CHAN),%VAL(IO$_READVBLK),
1           IOSB,,,DATA_ARRAY,%VAL(512),,,,)
```

```
SYS$QIOW(,%VAL(SRC_CHAN),%VAL(IO$_WRITEVBLK),IOSB,,,
1           DATA_ARRAY,%VAL(512),,,,)
```

8. SOURCE and/or TARGET may disconnect the link

```
SYS$QIOW(,%VAL(SRC_CHAN),%VAL(IO$_DEACCESS),IOSB,,,,,,,,)
```

1.4 How DECnet Identifies Logical Links

- TARGET and SOURCE will have a different network channel number for each logical link they establish
- DECnet associates the channel number (port) with a unique link identifier
- ECL modules on each side will agree on a pair of link addresses to associate with that link
  - A 16-bit numerical address is assigned to each end of a logical link
  - A given 16-bit address must not be assigned to two ports concurrently
  - A given 16-bit address must not be reassigned for a long period following its deassignment
  - The port at one end of the link contains the address of the port at the other end of the link, and vice versa
  - The complete identification of the link is a 32-bit number
- The logical link identifier is used as an index into the link table (LTB) to find the associated XWB for the logical link

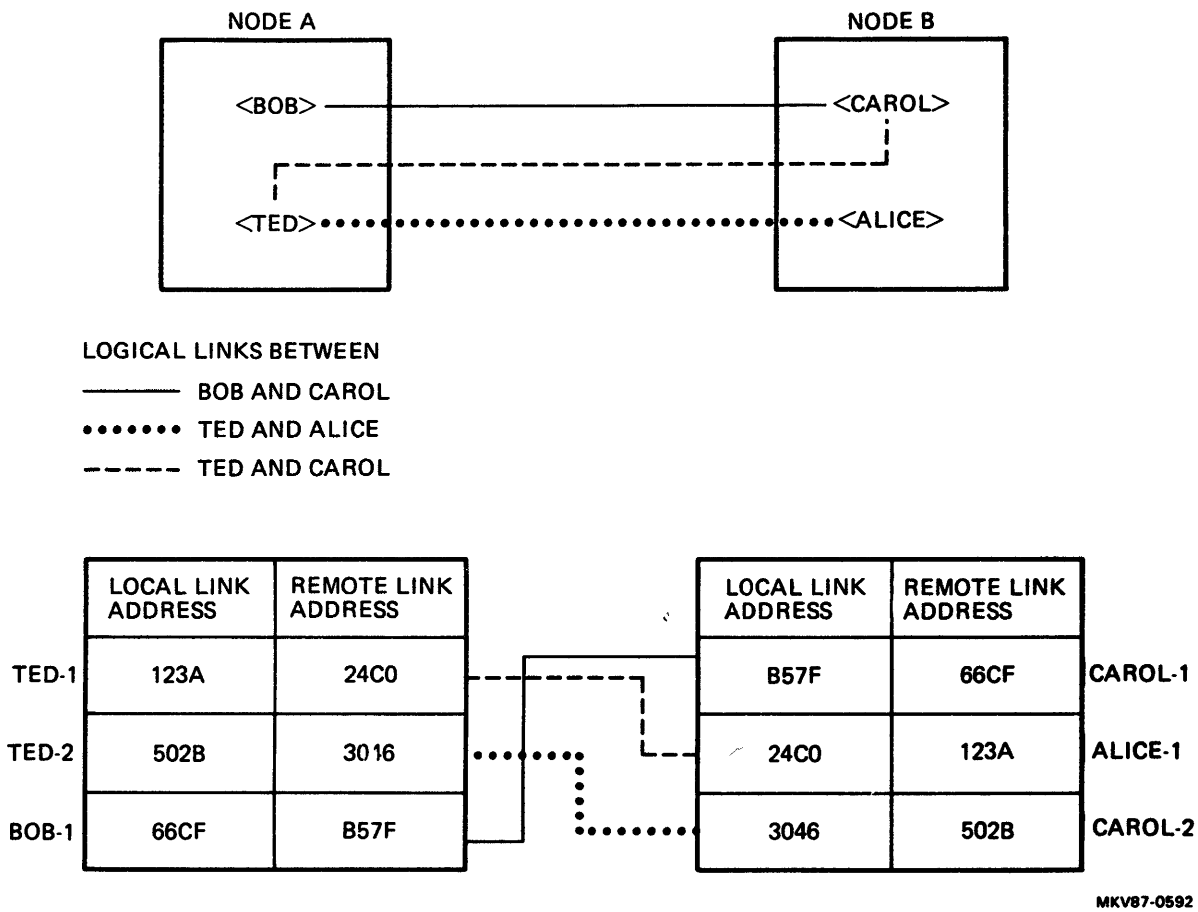


Figure 6-1 Identifying Logical Links

1.5 Seeing Logical Link Addresses

Logical links are accessed by the fields XWB$W_REMLNK and XWB$W_LOCLNK in the extended window block (XWB).

Logical link addresses may also be seen with NCP> SHOW KNOWN LINKS command.

```
FROM NODE 1.1
=============

NCP> SHOW KNOWN LINKS

Known Link Volatile Summary as of 15-APR-1986 23:59:17

                                                      Remote   Remote
   Link       Node            PID       Process       Link     User

  12328   1.4 (CARIBU)    2060176B  Sweet Pea       4113     CTERM
  12323   1.4 (CARIBU)    20601A0F  Song Bird       8204     TARGET.EXE
  12324   1.4 (CARIBU)    20601977  Reader          8205     TARGET
  11301   1.4 (CARIBU)    20601619  FAL_11301       5134     ROBIN
  13350   1.4 (CARIBU)    20601A24  PHONE_13350     6159     20400537


FROM NODE 1.4
=============

NCP> SHOW KNOWN LINKS

Known Link Volatile Summary as of 15-APR-1986 23:59:47

                                                    Remote   Remote
   Link       Node            PID       Process     Link     User

  8204    1.1 (WOLVES)    204003B8  NET_8204      12323    FINCH
  8205    1.1 (WOLVES)    20400539  NET_8205      12324    SWAN
  5134    1.1 (WOLVES)    204003B0  Spring        11301    FAL
  6159    1.1 (WOLVES)    20400537  Mallard       13350    PHONE
  4113    1.1 (WOLVES)    20400117  REMACP        12328    SARA
```

Example 6-1 Looking at Logical Links with NCP

1.6 Cluster Alias Internals

- Some or all VAXcluster nodes can share a DECnet node address
- Each node can still be accessed by its own address
- Must have at least one router in the cluster
- Shared address is **ALIAS**
- Each router in the cluster keeps a list of participants

  Cluster link ID table - Alias REGistration Block (AREG)

  4 words per entry: node address - flags and version - number of links in use

  (Uses the distributed lock manager to maintain the list)
- Incoming connection is assigned a cluster-wide link ID by the router
- Participating nodes are selected **round robin** with a bias for **ALIAS MAXIMUM LINKS** (in multiples of 16)
- Packet are always forwarded to the receiving node by means of the router
- Outgoing connection is assigned a cluster-wide link ID by its own node

see who has most amt of links available

1.7 SCL Tasks in Requesting a Logical Link Connection

- Identify the destination node address or channel number for the ECL by using the node-name mapping table
- Format connect data for the ECL layer
- Issue a connect request to the ECL
- Start the outgoing timer

1.8 SCL Tasks in Receiving a Logical Link Request

- Start the incoming timer
- Parse connect data to obtain source and target end-user process names and access control information
- Validate any access control information
- Identify and either activate or create the target process
- Map the source node's address to a node name
- Deliver the incoming connect request to the end-user process

1.9 NSP Functions in Establishing and Disconnecting Logical Links

A source NSP and a destination NSP exchange messages to establish and destroy (in other words, to connect and disconnect) logical links.

Figures 6-2 through 6-6 summarize the message exchanges.

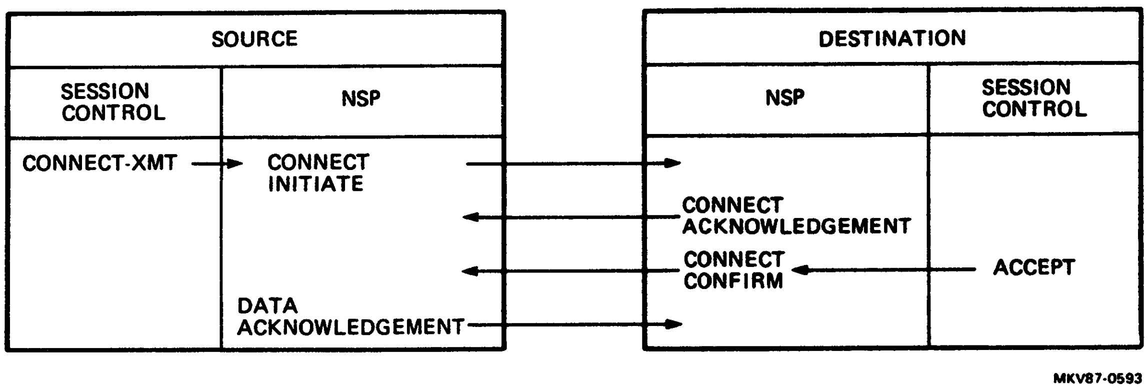


Figure 6-2 Connection with Acceptance

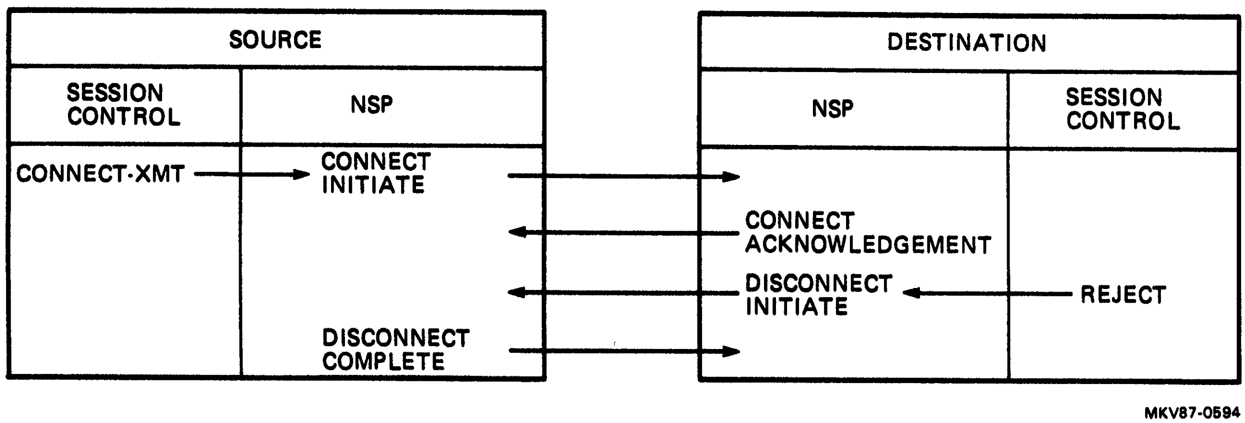


Figure 6-3 Connection with Rejection

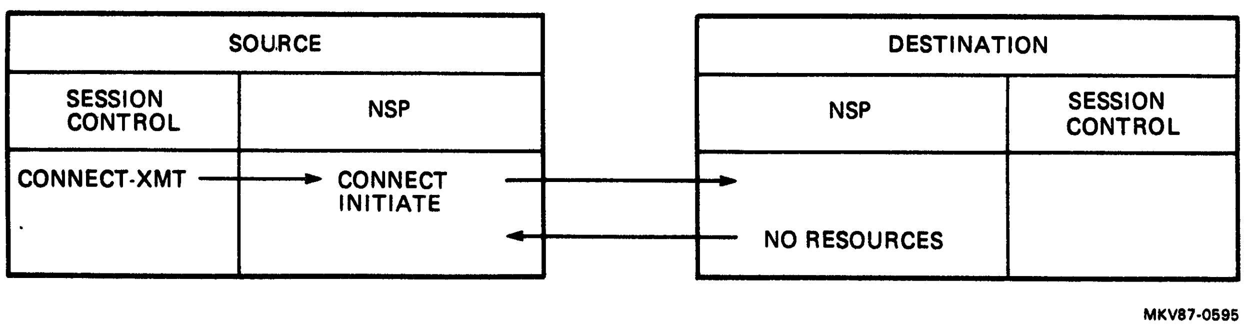


Figure 6-4 Connection Attempt with No Resources

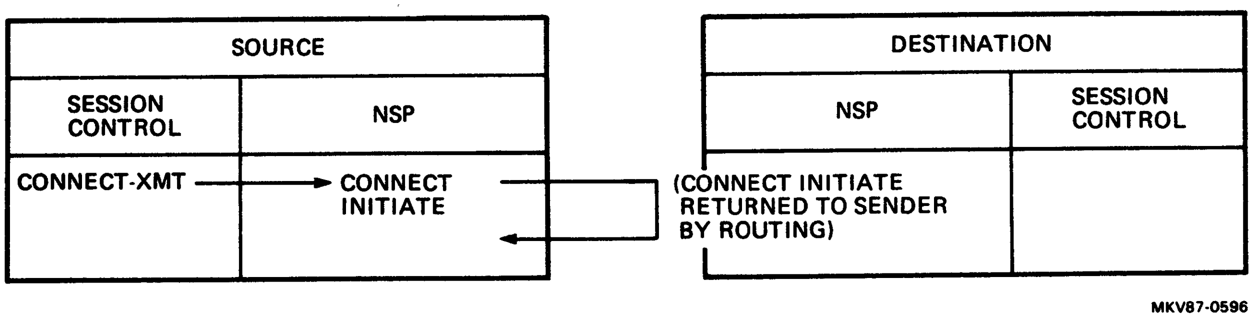


Figure 6-5 Connection Attempt with No Communication

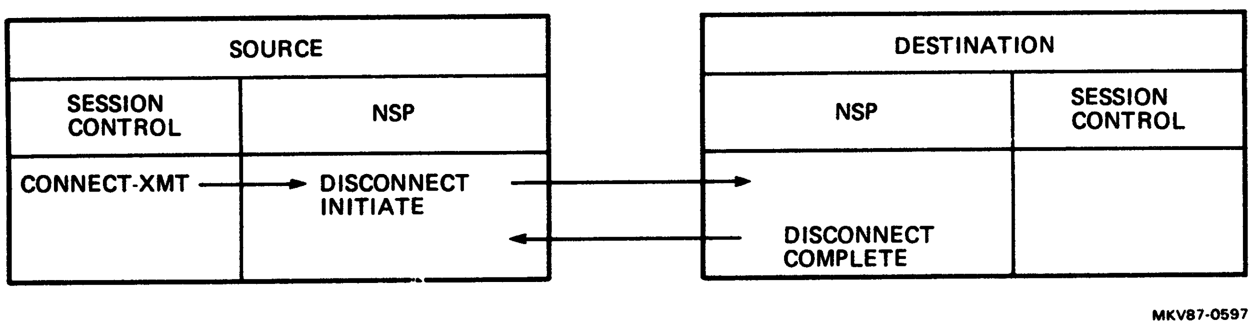


Figure 6-6 Disconnection

2 NSP FLOW CONTROL MECHANISMS

- Control buffer space for temporary message storage
- Ensure that data is not lost for lack of buffering capability and that deadlocks do not occur
- Both normal and interrupt data are subjected to flow control
- Mechanism used determined at logical link formation
- There is a choice of:

  1. No flow control
  2. Segment flow control (request count)
  3. Message flow control

- Choice is indicated by means of fields in connect initiate, retransmitted connect initiate, and connect confirm messages
- Each data-transmitting NSP must accept the type of flow control the data-receiving NSP expects

NOTES

1. **Message flow control is obsolete and will be eliminated at a future time.**
2. **DECnet-VAX always uses no flow control. It supports segment flow control for compatibility with other systems that may request it (RSX).**

2.1 No Flow Control

If the data-receiving NSP selected **no flow control,** the data-transmitting NSP may transmit data at any time (subject to the ON/OFF constraint).

1. A data transmitting NSP will send as many messages as it can put in the pipeline.
2. When the data receiving NSP cannot accept any more messages, it will send a link service data message with a DO NOT SEND (XOFF) indicator.
3. When the value is DO NOT SEND (backpressure), the data-transmitting NSP may not transmit normal data.
4. When the data receiving NSP can receive more packets, it will send a SEND (XON) indicator.

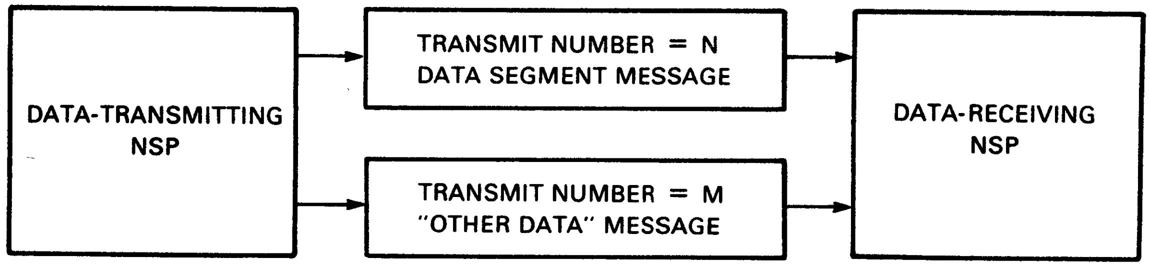
```
Process A                                  Process B
---------                                  ---------
Link Service Data Request     -->
(ON - send data)              <--          Data segment 1
                              <--          Data segment 2
                              <--          Data segment 3
Link Service Data Request
(OFF - do not send data)      -->
(ACK message 3)

Link Service Data Request
(ON - send data)              -->

                              <--          Data sement 4
                              <--          Data segment 5
Ack segment 5                 -->
```

Example 6-2 XON/XOFF Flow Control

3 SENDING AND RECEIVING NORMAL AND INTERRUPT DATA

- NSP uses a basic acknowledgment mechanism to ensure that messages are delivered
- NSP has four types of data messages:
  - A. Data segment messages
  - B. Interrupt messages
  - C. Data request messages
  - D. Interrupt request messages
- Uses same acknowledgment mechanism for each type of data message
- On a logical link, the four data messages can be thought of as moving in two subchannels
- Messages in each subchannel are numbered sequentially by the transmitting NSP
- The transmitting NSP waits up till retransmit time for an acknowledgment and then it retransmits the message
- It is not necessary to acknowledge each message individually

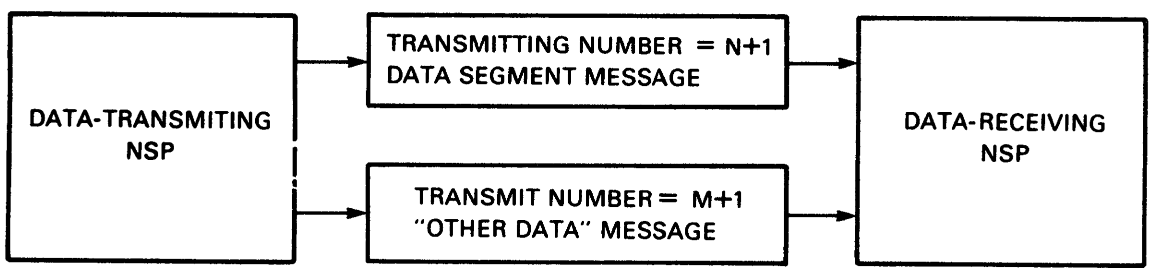
1 The data-transmitting NSP assigns a transmit number to a message, transmits the message, and starts a timer.



DATA SUBCHANNELS

2 If the timer times out, the message is retransmitted.

3 If the timer does not time out, and the flow control mechanism allows another message to be sent, the data-transmitting NSP assigns the transmit number plus one to the next data message transmitted in that subchannel.



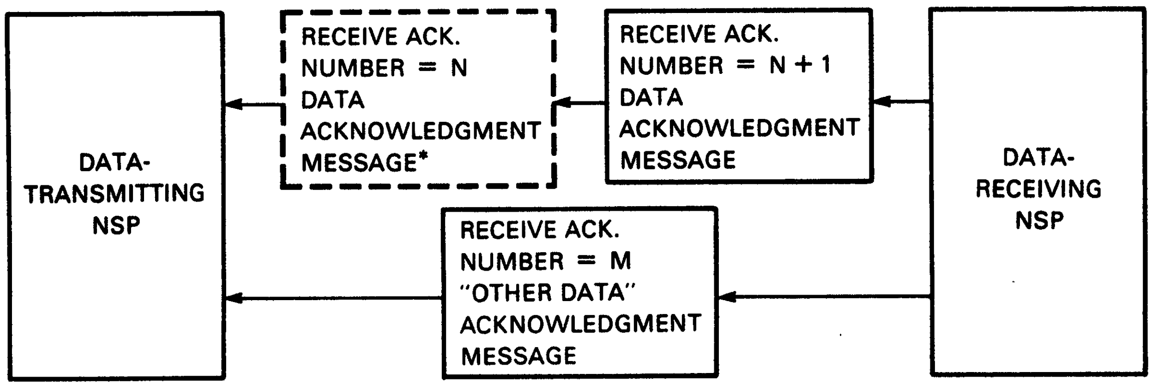
DATA SUBCHANNELS

4 When the message with the first transmit number is received by the data-receiving NSP, it returns that number as an acknowledgement number within the first acknowledgement.

MKV87-0606

Figure 6-7 NSP Segment Acknowledgement (Sheet 1 of 2)

5 If the next data message transmit number received is equal to the current acknowledgment number plus one, the data-receiving NSP accepts the data message, incrementing the acknowledgment number. It then sends the new receive acknowledgment number back to the data-transmitting NSP within an acknowledgment message.



DATA SUBCHANNELS

* The data-receiving NSP might not send an acknowledgment for each data message received. The received acknowledgment number implies that all previous numbers were received.

6 However, if the data-receiving NSP receives a data message transmit number less than or equal to the current receive acknowledgment number for that subchannel, the data segment is descarded. The data-receiving NSP sends an acknowledgment back to the data-transmitting NSP. The acknowledgment contains the receive acknowledgment number.

7 If the data-receiving NSP receives a data message transmit number greater than the current receive acknowledgment number plus one for that subchannel, the data segment may be held until the preceding segments are received or it may be discarded.

MKV87-0607

Figure 6-7 NSP Segment Acknowledgement (Sheet 1 of 2)

3.1 Pipeline Quota

- Pipeline quota specifies the maximum number of bytes of nonpaged pool that NSP will use for transmission over logical links
- Increase for satellite links (6000 +)
- Limit is in multiples of packets, set to pipeline quota/buffer size
- Packets also get buffered on the output queues for each circuit
- Each circuit has a set of private receive buffers

The pipeline factor, n, will be

$$n = \frac{\text{pipeline quota}}{\text{buffer size}}$$

- Transmitter can send up to n segments before receiving ACK on the first.

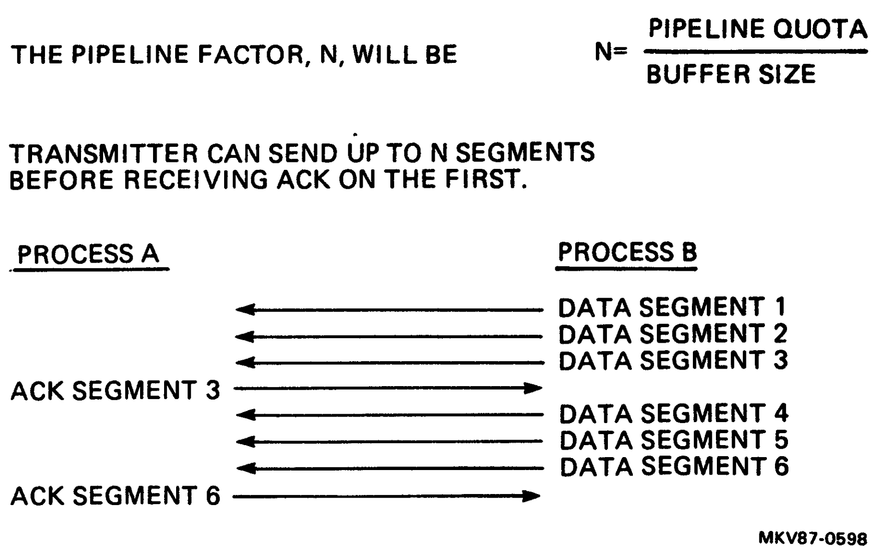


Figure 6-8 Operation of Pipeline Quota

4 ROUTING UPDATE PROCESS

- Constructs and sends a segmented routing message to adjacent nodes after determining that certain conditions are met:
  - A Level 1 Routing Message containing a given set of destinations is sent on a circuit when a routing configuration change has occurred
  - Level 1 Routing Messages are sent to adjacent routing nodes within the node's home area
  - Routing updates are forced after the elapsing of the **routing timers**
  - A Level 2 Routing Message is sent under similar circumstances for AREA routing nodes
  - Routing Message is sent to the multicast ID "all routers" on Ethernet circuits
  - Sent to adjacent nodes on nonbroadcast circuits
- Accepts the following as input:
  - The minimum hop vector, MINHOP
  - The minimum cost vector, MINCOST
  - The area minimum hop vector, AMINHOP
  - The area minimum cost vector, AMINCOST
  - The Send Routing Message flags, SRM
  - The Area Send Routing Message flags, ASRM

4.1 Processing of Routing Update Message

1. Check to see that this is a new message
2. Store the changed cost/hops info (**NODE CHANGES** vector)
3. Re-evaluate the cost/hops for node 0
4. Check information for each node in routing message
   - Build the packed cost/hops field by merging hops/cost information
   - If the cost or hops to this node exceed our maximums (MAXHOPS/MAXCOST), then declare the node unreachable by forcing the cost and hops to infinity (^X7FFF)
   - Record changes so that EVL may be notified
   - Determine least cost path to this node
     a. Get the cost/hops for this node over this adjacency, and increase it by the hop for ourself
     b. Also compute the new cost for this path
     c. For a broadcast circuit, the cost/hops buffer contains the state of all end-nodes on that broadcast circuit
     d. Check to see if this path is "less cost" than the previous minimum cost (MH/MC vector)
     e. **Use the node address of the adjacent node as a tiebreaker**
   - If this path is the "least cost", remember it as the best
   - Send routing message only if min hops/cost has changed

```
      1                             2                              3

CIRCUIT NO.:    #1            #2              #1            #1
     +----------+                 +----------+                 +----------+
     |          |                 |          |                 |          |
     |  NODE A  |-----------------|  NODE B  |-----------------|  NODE C  |
     |          |                 |          |                 |          |
     +----------+                 +----------+                 +----------+
CIRCUIT COST:  @4             @2              @6            @4
```

NODE A

	HOP/COST 0	HOP/COST 1	MH/MC	FORWARDING DB/ REACHABILITY VECTOR
(A)	0/0	2/4 (2)	0/0	0
(B)	∞	1/4	1/4	1
(C)		2/10	1	1

index into OA

NODE B

	HOP/COST 0	HOP/COST 1	HOP/COST 2	MH/MC	FORWARDING DB/ REACHABILITY VECTOR
(A)	0/0	3/12	1/2	1/2	2
(B)		2/10	(2) 2/6	0/0	0
(C)		1/6		1/6	1

NODE C

	HOP/COST 0	HOP/COST 1	MH/MC	FORWARDING DB/ REACHABILITY VECTOR
(A)		2/6	2/6	1
(B)		1/4	1/4	1
(C)	0/0	2/10	0/0	0

Example 6-3 Routing and Forwarding Database Tables

(2) to B from A

2	1	0	0	1	4

to ? from B

3	2	0	0	1	2

(2) to A from B

2	2	0	0	1	2

```
        #1+----------+#2            #1+----------+#2
          |          |                |          |
    +-----|  NODE A  |----------------|  NODE B  |--------+
    |     |          |                |          |        |
    |   @4+----------+@2            @6+----------+@1      |
    |                                                     |
    |                                                     |
    |                  #1+----------+#2                   |
    |                    |          |                     |
    +--------------------|  NODE C  |---------------------+
                         |          |
                       @4+----------+@2
```

NODE A	HOP/COST 0	1	2	MH/ MC	FORWARDING DB/ REACHABILITY VECTOR
(A)	0,0	***	***		
(B)	***				
(C)	***				

NODE B	HOP/COST 0	1	2	MH/ MC	FORWARDING DB/ REACHABILITY VECTOR
(A)	***				
(B)	0,0	***	***		
(C)	***				

NODE C	HOP/COST 0	1	2	MH/ MC	FORWARDING DB/ REACHABILITY VECTOR
(A)	***				
(B)	***				
(C)	0,0	***	***		

Example 6-4 Routing UPDATE Exercise

5 MESSAGE SEGMENTATION

5.1 Segmentation and Reassembly of User Messages

- Buffer size and number of buffers are limited on each node
- User messages from session control buffers cannot always be sent in one piece
- These messages contain user data as well as NSP header information
- A data-transmitting NSP breaks up data contained in a single session control buffer into segments (packets)
  - Transmits the segments by means of Data Segment messages
  - The maximum length of a data segment is the lesser of:
    1. The size of a transmit buffer in the source node

       (EXEC BUFFER SIZE or LINE BUFFER SIZE)
    2. The maximum length that the data-receiving NSP can receive

       (~~SEGMENT BUFFER SIZE or~~ LINE BUFFER SIZE)
- A data-receiving NSP reassembles the segments into the correct sequence

NOTE

The segment acknowledgment scheme (Section 3) ensures that all data segments are received.

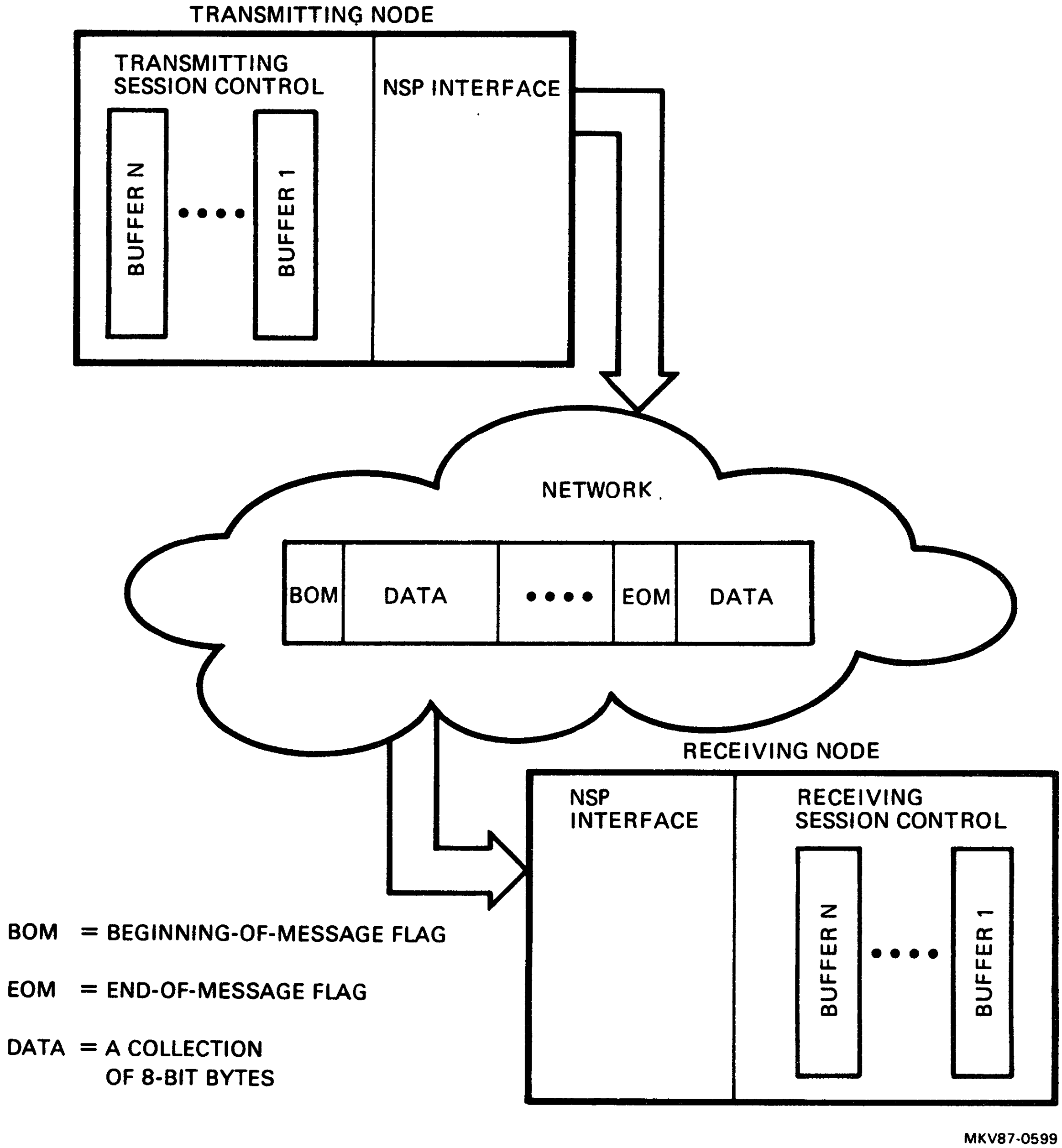


Figure 6-9 Model of Data Flow as Seen by Session Control

5.1.1 Message Segmentation Steps

- For a user packet being transmitted, the routine is XMIT_COPY in NETDRVNSP

- Get a free transmit buffer

- The size of the transmit buffer will be

```
      XWB$W_REMSIZ       (maximum transmit segment size
                          from the remote node)
    + NSP$C_HSZ_DATA     (size of NSP header)
    + TR3$C_HSZ_DATA     (size of ROUTING LAYER header)
    + CXB$C_OVERHEAD     (size of CXB header)
```

- Get a fixed sized buffer and enter as much of the message as possible into it (CXB)

  (Take XWB$W_REMSIZ of the data at a time)

- Process that segment

- Repeat for each segment from BOM to EOM

TRACING DECnet ACTIONS

INTRODUCTION

The DECnet-VAX Source Listings are available on microfiche or on-line. This chapter presents hints for reading the source listings to analyze a DECnet mechanism, and also traces various actions.

Topics include:

- Hints On Reading the Source Listings
- Major Modules of DECnet-VAX
- Tracing the Creation of a Logical Link
- Tracing the Sending of Normal Data
- Other DECnet Actions
- Selected Routine Listings

1 HINTS ON READING THE SOURCE LISTINGS

I. Preparation

A. Have a good, general idea of the major components and their interaction.

B. Have sure copies of the map and the listings available either on paper or microfiche.

C. Learn how to use the map and listings effectively:

- Identifying the starting address of the major component, if any
- Determining how many modules make up a major component
- Identifying which module and which line of code defines a global symbol

II. Pass 1

A. Treat the listings like a road map. **Make sure you don't get lost!**

B. Focus on instructions that will pass control to another module or subroutine. **Do not try to follow every instruction!**

C. **Read the comments!**

D. If you have paper listings, you can use a highlight pen to trace the major paths.

E. If you use microfiche, chart out a simple major path map.

III. Pass 2

A. Assume an error-free path.

B. Use the map or highlighted listings to extract some interesting information.

C. Interesting paths to trace through:

- Sending connect initiate
- Receiving connect initiate
- Sending connect confirm
- Sending/receiving normal data
- Sending/receiving interrupt data
- Disconnecting a logical link

D. After tracing through the different paths, it will be clear that a lot of common code is executed.

E. The different traces can be used to get an overall map.

IV. Pass 3

A. Relate the data structure to the code.

B. Put in sufficient comments to get a "tour guide" through the listings.

V. Subsequent Passes Should be Used to Obtain Specific Information

2 MAJOR MODULES OF DECnet-VAX

NETACP (Reference: NETACP.MAP)

Module Name	Bytes	Description
NETACPTRN	4265	Control network local node state transitions
NETPROCRE	3798	Process creation
NETTRN	1171	Main ACP loop and misc. subroutines
NETCNF	3057	Configuration database access routines
NETDLLTRN	26405	Routing and Datalink layer
NETCTLALL	3736	Process ACP control QIOs
NETEVTLOG	1907	Process Event logging needs
NETLLICNT	1484	Counter support for nodes and logical links
NETGETLIN	720	Check for DECnet license
NETCONFIG	11556	Local configuration database
NETCNFACT	6565	Configuration database access action routines
NETCNFDLL	6007	Datalink database action routines
NETDLE	2218	NETACP DLE processing
NETTREE	3266	BINARY TREE processing routines
NETCONECT	3000	Process user connect requests
NETOPCOM	165	Operator communications
NETCLUSTR	10992	Cluster node name routines
NETJNXCO	433	

NETDRIVER (Reference: NETDRIVER.MAP)

Module Name	Bytes	Description
NETDRVSES	3717	DECnet session control module for NETDRIVER
NETDRVNSP	5532	DECnet NSP module for NETDRIVER
NETDRVXPT	4661	NETDRIVER routing layer
NETDRVQRL	741	DECnet Quick Routing Layer module
NETDRVJNX	433	

NDDRIVER (Reference: NDDRIVER.MAP)

Module Name	Bytes	Description
NDDRIVER	2400	DECnet DLE driver

The Appendix contains a detailed listing of DECnet-VAX modules.

3 TRACING THE CREATION OF A LOGICAL LINK

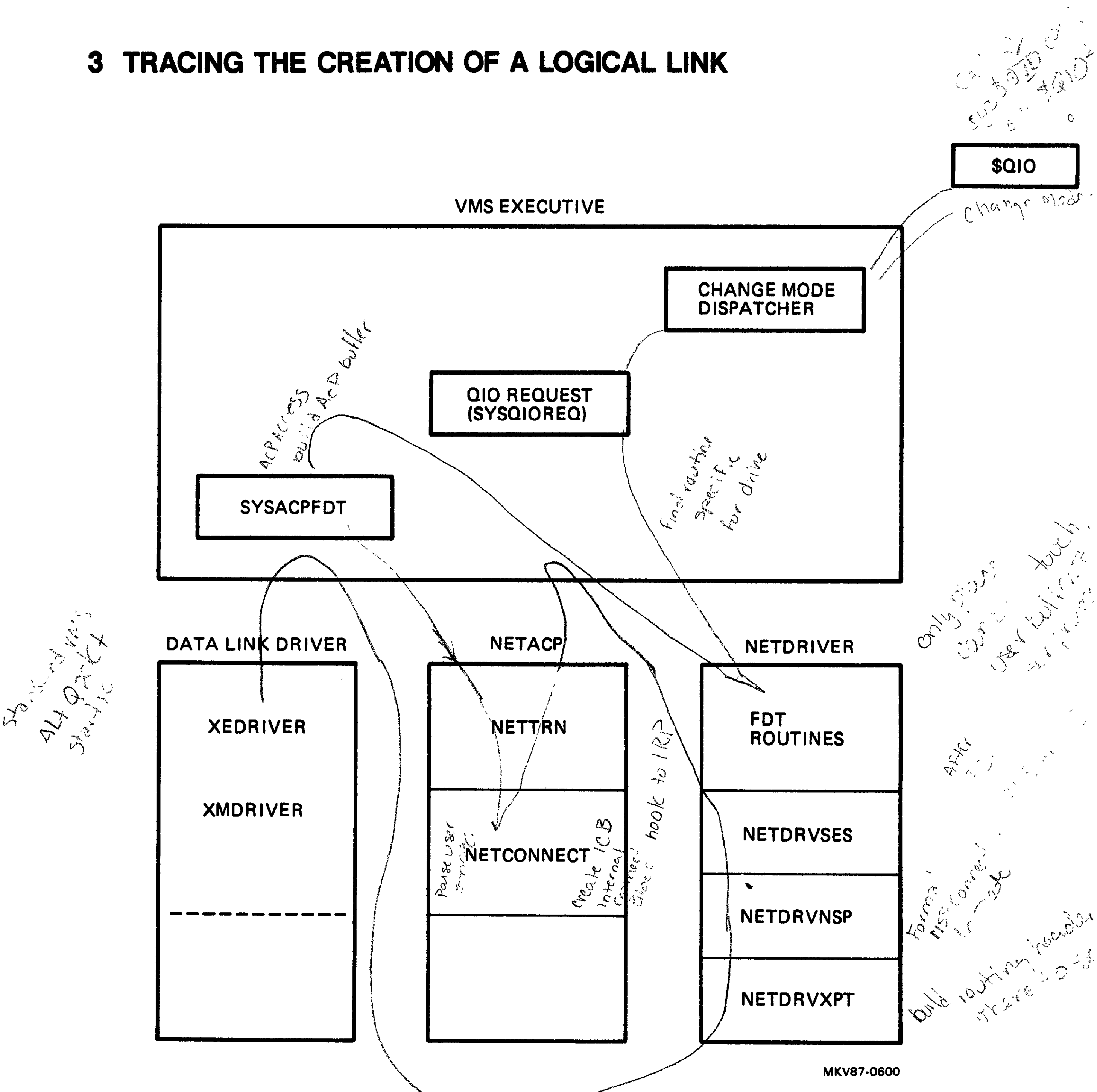


Figure 7-1 Environment of Sending a Connect Initiate

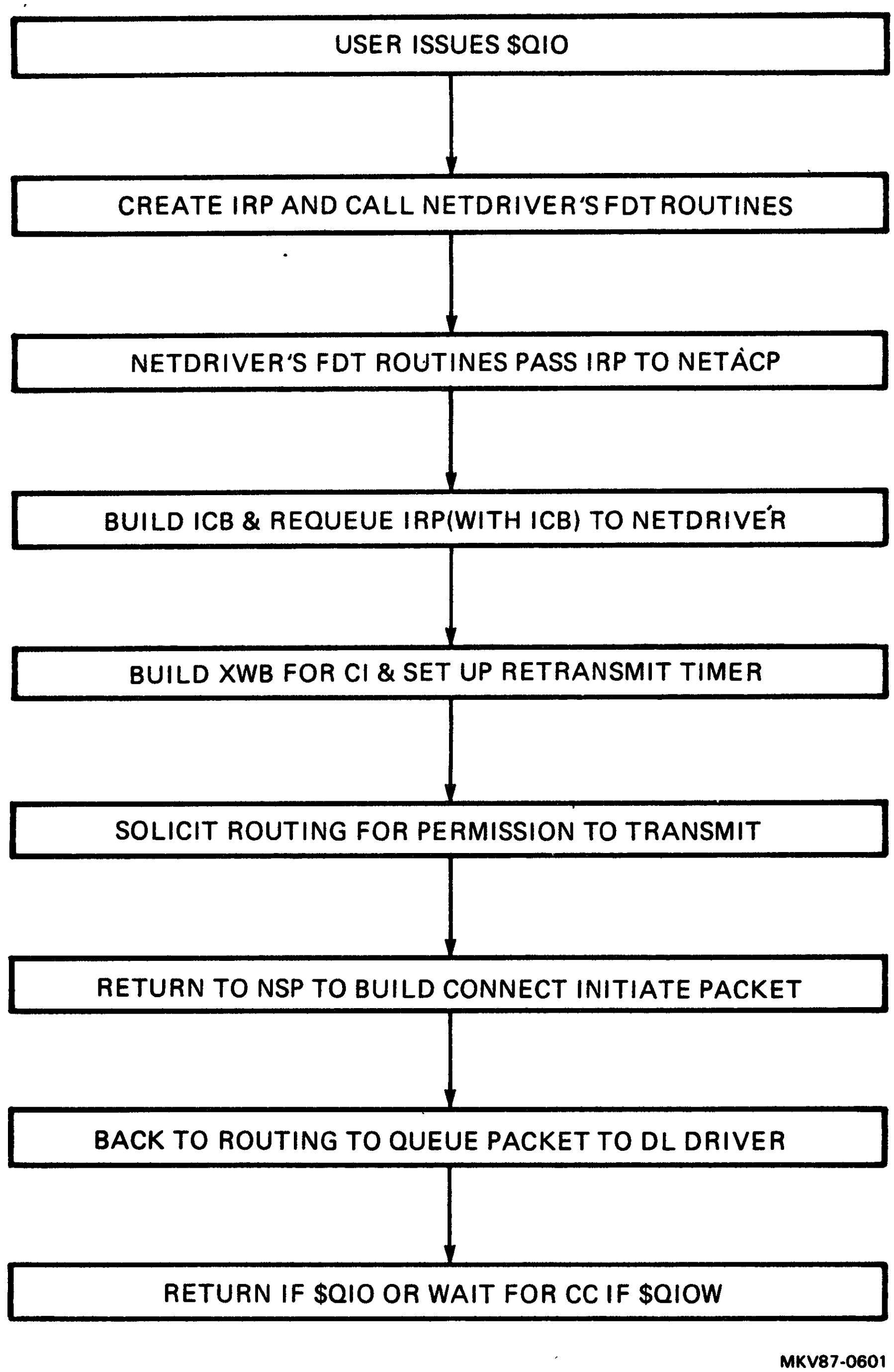


Figure 7-2 Simplified Flow of Sending a Connect Initiate

3.1 Overview of Sending a Connect Initiate (CI)

A. User program issues $QIO

B. EXE$QIO creates IRP

C. NETDRIVER FDT routines create **complex buffer** and pass IRP to VMS EXEC. jmp ACP$ACCESSNET

D. VMS EXEC sets up ACP buffer and queues it to NETACP

E. Overview of NETACP actions

NETACP reads the user connect info to build an Internal Connect Block (ICB) which it attaches to the IRP$LDIAGBUF field of the IRP. It then requeues the IRP to the driver. exe$insioq

The role of NETACP is to look up default access control (user name, password, account) information in its database and translate node and object names to numbers.

F. Overview of NETDRIVER actions

NET$ACCESS reads the ICB and determines the type of connect. It builds an XWB for connect initiate events and locates an already existing XWB for all others. NET$ACCESS stores the appropriate event code in R7 and returns expecting the caller (NETDRIVER) to call the event dispatcher (NET$EVENT).

Note that the size of the XWB **is not charged against** the **user byte count or byte limit quotas.** Logical links are charged against the file limit quota of a process.

G. NET$EVENT dispatches to ACT$INITIATE to set up the retransmit timer and switch to the solicit state

H. Solicit permission from routing to transmit the packet and if granted build the CI message

I. Queue the packet to the UCB of the data link driver by means of the internal QIO mechanism

J. Return to the user ($QIO) or wait for a connect confirm ($QIOW)

3.1.1 Sending a Connect Initiate (CI)

1. User program, after setting up the Network Connect Block (NCB), issues a QIO call with a function of IO$_ACCESS

```
        $QIOW    CHAN=NET_CHAN,     -
                 FUNC=#IO$_ACCESS,  -
                 IOSB=IOSB,         -
                 P2=NCB_DESC

NCB:    .ASCII  /CANADA::"TASK=NONTRTGT.EXE"/

NCB_DESC:
        .LONG   .-NCB
        .ADDRESS NCB
```

2. EXE$QIO

   - Executed by means of the change mode dispatcher (CMODSSDSP) in the VMS EXEC

   - Forms IRP to be passed to NETDRIVER such that:

```
IRP$B_TYPE    = DYN$C_IRP
IRP$L_PID     = user process ID
IRP$L_UCB     = NETDRIVER
IRP$W_FUNC    = IO$_ACCESS
IRP$W_CHAN    = channel index
IRP$W_STS     = init to zero
IRP$L_SVAPTE  = NCB_DESC
```

   - Enters the Function Dispatcher Routines (FDT routines) in NETDRIVER

   - For connect initiate, goes to NET$FDT_ACCESS

3. NET$FDT_ACCESS - NETDRIVER (NETDRVSES)

   - Packages the user parameters into a "complex buffer"
   - The IRP is passed by means of the executive to NETACP at NETTRN
   - Passes it by means of an ACP Queue Block Mechanism (AQB)
   - Jumps to ACP$ACCESSNET

4. ACP$ACCESSNET - VMS EXEC (SYSACPFDT)

   - Access (connect) to network function processing
   - Build ACP buffer
   - Queue it to ACP
   - Wake up NETACP - enter NET$DISPATCH in NETTRN module

5. NET$DISPATCH - NETACP (NETTRN)

   - Major NETACP work dispatching loop
   - NETACP has three queues to handle (in order):

     A. Timer queue - Timer ASTs

     B. Work queue - requests from NETDRIVER

     C. AQB queue - requests from user program

   - If there is a request in one of its three queues, NETACP dispatches to one of its processing routines
   - When the queues are empty, NETACP hibernates
   - For connect initiate, the request is in the AQB
   - Dispatch to NETCONECT (for CI)

6. NETCONECT - Network ACP

- This module performs processing for logical link connect requests including connect initiate, connect confirm, and connect reject

- Parse the Network Connect Block (NCB)

- Build an Internal Connect Block (ICB) containing the parse and hang it onto the IRP

  NCBs have the same form as the translation of the logical name "SYS$NET"

  node"access control info"::"taskname/linknumber+userdata"

'node'	may be specified either by name or number.
'object'	may be specified either by name or number.
'taskname'	legal taskname formats are:
	"objectname=
	"objectnumber=
	"TASK=taskname
	"0=taskname

- Obtain the default access control information (user name, password, account) and set up the proxy login state for this OBI (Object)

- Set up the Remote User ID (RID) for display purposes

- Translate the node name to node address

- Check for access restrictions set for the remote node

- See if the connect is allowed based on local node state

STATE	Allow Connect If
ON	Always
RESTRICT	If this is a connect initiate, or if the partner node is the local node
SHUT	Never
OFF	Never

- See if the remote node is reachable

  Check the ADJ and LPD for that node

- If node is adjacent, try to increase the buffer size

- Requeue the IRP to the NETDRIVER

  (IRP$L_DIAGBLF field points to the ICB)

- Return to NET$DISPATCH

7. NET$DISPATCH - NETACP (NETTRN)

   - Finish IRP processing

   - Jump to EXE$INSIOQ

8. EXE$INSIOQ - VMS EXEC (SYSQIOREQ)

   - If NETDRIVER is busy, call EXE$INSERTIRP to QUEUE the IRP

   - If NETDRIVER is idle, call IOC$INITIATE to process the IRP set the busy bit

9. IOC$INITIATE - VMS EXEC (IOSUBNPAG)

   - Put the IRP address and user parameters into the UCB

   - Clear the device status bits

   - Jump to NET$STARTIO (NETDRIVER's start I/O entry point)

10. NET$STARTIO - NETDRIVER (NETDRVSES)

- Start the I/O operation
- Extract function code from IRP
- Dispatch to NET$ACCESS

11. NET$ACCESS - NETDRIVER (NETDRVSES)

- Read the ICB and determine the type of connect
- Set up outbound connect timer (outgoing timer)
- Put the event code in R7
- Return to NET$STARTIO, which calls the event dispatcher (NET$EVENT)

Helena:: DMPD$:[crash.T52-1]sysdump.dmp

QAR$disk:<QAR.CRASH-Star.Attach>01666.

Crash

calypso - Rigel - FT

89

Crash-star-ft

DIMONDQAR$server:: DM.

12. NET$EVENT - NETDRIVER (NETDRVSES)

- State table event dispatcher used to determine what is to be done and what state the XWB is to enter next

- Inputs:

R7	Code of event to process
R6	The UCB address
R5	Address of XWB
R3	QIO IRP address

- Output:

R0	Status code from the action routine to be returned to the caller of the event dispatcher

- Dispatch to ACT$INITIATE based on the event code

- On return, change logical link state

- Actions dispatched by ACT_DISPATCH of NET$EVENT

Routine Called	Description
ACT$ABORT	Abort logical link
ACT$BUG	Bugcheck action routine
ACT$CANLNK	Cancel logical link
ACT$CONFIRM	Process connect confirm
ACT$DEACCESS	Process IO$_DEACCESS
ACT$ENT_RUN	Enter RUN state
ACT$INITIATE	Process connect initiate
ACT$LOG	Log-event action routine
ACT$NOP	Do nothing
ACT$RES_DISC	Resume disconnect processing
ACT$RCV_CA	Respond to Connect Acknowledge
ACT$RCV_CC	Respond to a received connect confirm
ACT$RCV_CI	Process a received connect initiate
ACT$RCV_CR	Process a retransmitted connect initiate
ACT$RCV_DATA	Process a received data message
ACT$RCV_DTACK	Process a received data ACK
ACT$RCV_DX	Process a received DI or DC message
ACT$RCV_LI	Process a received INT/LS message
ACT$RCV_LIACK	Process a received INT/LI ACK
ACT$RCV_RTS	Receive CI being 'returned to sender'
ACT$RTS_NLT	Return to sender as 'no link terminate'
ACT$SHRLNK	Abort the QIO
ACT$SSABORT	Abort the QIO (link was disconnected)

13. ACT$INITIATE - NETDRIVER (NETDRVSES)

- Connect initiate action routine
- Get ICB from IRP and attach it to XWB
- Move remaining parameters from the ICB into XWB fields
- Set up timer and delay so that the connect message will retransmit periodically if necessary

$$\text{Retransmit Time} = \frac{\text{Delay * Delay Factor}}{16}$$

$$\text{Delay} = \frac{\text{MSG Delay + Old Delay * Delay Weight}}{\text{(1 + Delay Weight)}}$$

- Return to NET$EVENT

14. NET$EVENT - NETDRIVER (NETDRVSES)

- Switches to next state (solicit)
- Solicits permission from NSP to send connect request
- Execute NET$SCH_MSG

15. NET$SCH_MSG - NETDRIVER (NETDRVSES)

- Schedule message transmission by means of NSP$SOLICIT

16. NSP$SOLICIT - NETDRIVER (NETDRVNSP)

- Solicit permission to transmit from routing
- Call TR$SOLICIT in routing layer to get an IRP and a buffer to send the connect request to the remote node

17. TR$SOLICIT - NETDRIVER (NETDRVXPT)

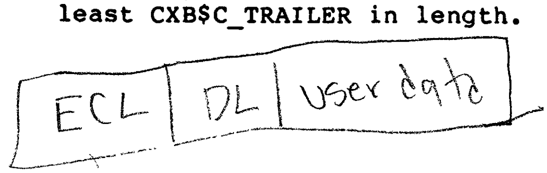
- Process ECL request to transmit into the network
- Find the appropriate logical path (LPD)
- Check availability of the transmission resources
- If okay, grant permission to transmit (TR$GRANT)
- Else, enter the request block onto a wait queue

18. TR$GRANT - NETDRIVER (NETDRVXPT)

- Set up the routing fields of the Complex Buffer (CXB)
- Reactivate solicitor (NSP$SOLICIT @QUICK_SOL) granting permission to transmit

On return, the CXB and registers are set up as follows:

Standard VMS Buffer Header	11 bytes long. CXB$L_FLINK and CXB$L_BLINK may be used by the transport layer. CXB$W_SIZE must be correct. CXB$B_TYPE must be DYN$C_CXB.
ECL Pure Area	Starts with CXB$B_CODE (byte 11) and continues to CXB$C_LENGTH. This area is read-only to transport and below. It cannot even be saved/restored.
Datalink Layer Impure Area	Starts at CXB$C_LENGTH and is at least CXB$C_DLL bytes long. Used by the datalink for protocol header or state information.
Body of Message	Must be quadword-aligned and starting no sooner than CXB$C_LENGTH + CXB$C_DLL (= CXB$C_HEADER).
Datalink Layer Impure Area	Used by the datalink layer for protocol (e.g., checksum) or state information. Must be at least CXB$C_TRAILER in length.



R9	ADJ address
R8	LPD address
R3	IRP address -- unmodified from call
R0	Low bit set - if message is to be transmitted Low bit clear - if no message to transmit

19. NSP$SOLICIT - NETDRIVER (NETDRVNSP @ QUICK_SOL)

    - Build the Connect Initiate (CI) message (BLD_DISPATCH)
    - Set up the routing header info
    - Update information in the Node Counter Block (NDC)
    - Return to the routing layer to send the message (TR$SOLICIT)

20. TR$SOLICIT - NETDRIVER (NETDRVXPT @ QUICK_SOL)

    - Make sure NETACP is still active
    - Return to TR$GRANT (NETDRVXPT) which branches to FINISH_XMT_HDR

21. FINISH XMTHDR - NETDRIVER (NETDRVXPT)

    - Build a routing header based on the output path
    - Finish building the IRP and queue it to the UCB of the data link driver (EXE$ALTQUEPKT)
    - If for the local node, use TR$LOC_DLLXMT - NETDRIVER (NETDRVXPT)

22. EXE$ALTQUEPKT - VMS EXEC (SYSQIOREQ)

    - Activate the driver at its ALTSTART entry point using the internal QIO mechanism
    - The packet will now be sent out over the circuit and control will return to NET$STARTIO

23. NET$STARTIO - NETDRIVER (NETDRVSES)

    - Exits with the IRP still queued to the UCB
    - Waits for either a connect confirm or disconnect
    - Returns to EXE$QIO

24. EXE$QIO - VMS EXEC

    - If $QIO, returns to the user indicating that the $QIO has been processed - the IOSB not filled in
    - If $QIOW, waits for connect confirm or disconnect before returning to the user with the IOSB filled in
    - User does not have a logical link until the connect confirm is generated and processed

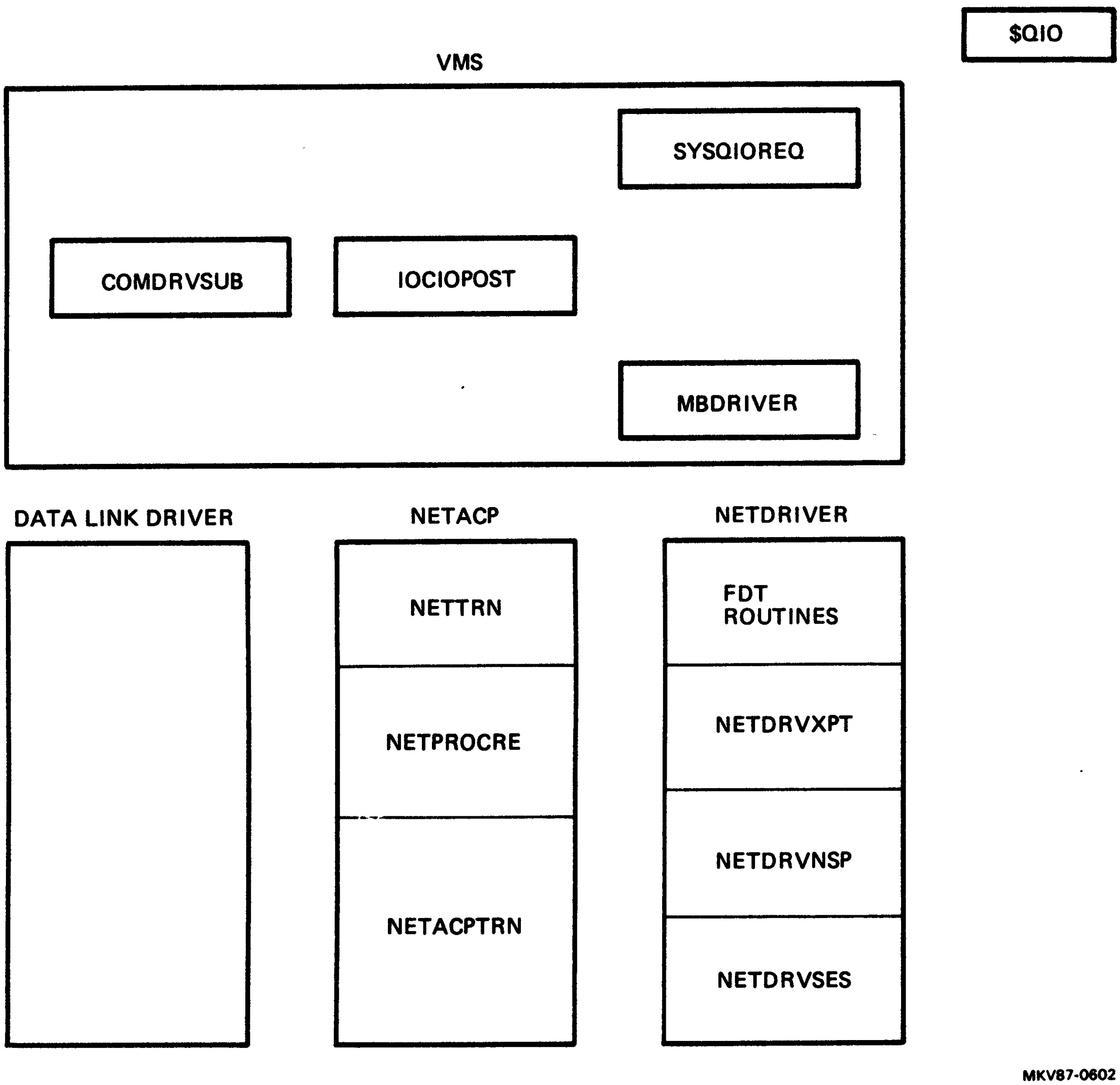


Figure 7-3 Environment of Receiving a Connect Initiate

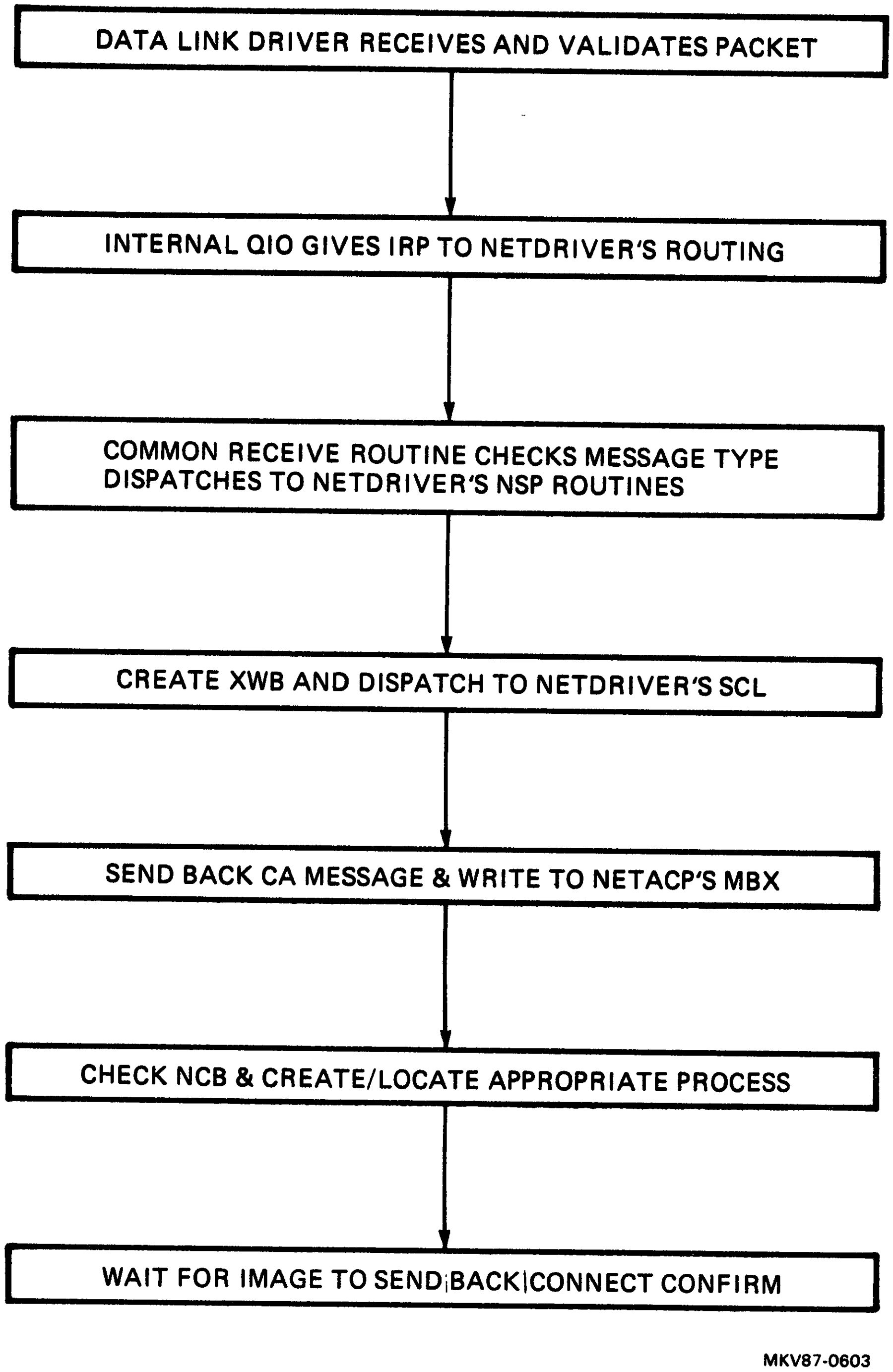


Figure 7-4 Simplified Flow of Receiving a Connect Initiate

3.2 Overview of Receiving a Connect Initiate

A. Data link drivers perform the necessary data link checking

B. The data is passed to the routing layer (NETDRVXPT)

C. The message type and the routing header are checked

D. If the message is for the local node, control is passed to ECL and the destination logical link address is examined

E. Session control sends back a connect acknowledge message and calls NETACP by means of a mailbox write

F. NETACP gets the process information from the NCB

G. NETDRIVER is informed that the process has been created

3.2.1 Receiving a Connect Initiate

1. The datalink device driver (XEDRIVER, XMDRIVER, etc.) receives and validates a data message.

   - Jumps to COM$POST to perform the standard VAX/VMS I/O completion processing

2. COM$POST - VMS EXEC (COMDRVSUB)

   - Used by the terminal, mailbox, and data link drivers to complete I/O operations independent of the status of the unit

   - Increment operation count

   - Insert packet on the I/O posting queue (IOC$GL_PSBL)

   - Request a software interrupt for I/O post processing (SOFTINT #IPL$_IOPOST)

3. IOCIOPOST - VMS EXEC (IOCIOPOST)

   - Implements the device-independent completion processing for I/O packets

   - Performs all appropriate completion activity required for the packet

     Setting event flags, unlocking buffer pages, releasing buffers

   - Because the initial request was from an internal QIO, the IRP is given to the NETDRIVER at either TR$RCV_DIO_DATA for direct I/O, or TR$RCV_BIO_DATA for buffered I/O

4. TR$RCV_DIO_DATA - NETDRIVER (NETDRVXPT)
   TR$RCV_BIO_DATA - NETDRIVER (NETDRVXPT)

   - Receive direct/buffered I/O from datalink layer
   - The IRP is being returned by the data link driver after a receive operation
   - Statistics are taken and the packet is routed to its destination
   - Reset byte count quota (for BIO)
   - Detach the CXB from the IRP
   - Requeue the received IRP to the datalink of the same device for another receive
   - Process the message by branching to RCV_DIO_BIO

5. RCV_DIO_BIO - NETACP (NETDRVXPT)

   - Common receive IRP processing
   - Check for success of the read request
   - Process the received message
   - Determine size of message
   - Dispatch on message type by branching to DISP_RCV_MSG

6. **DISP_RCV_MSG - NETDRIVER (NETDRVXPT)**

- Process the received message by dispatching to the appropriate action routine
- The first byte of the received message should be one of the following:

<0000 1000>	Phase II NOP
<0101 1000>	Phase II Start
<0100 xx10>	Phase II route header
<000x x010>	Phase III route header
<000x x010>	Phase IV nonbroadcast circuit route header
<00xx 0x10>	Phase IV broadcast circuit route header
<0000 0001>	Phase III init
<0000 0011>	Phase III verification
<0000 0101>	Phase III hello message
<0000 0111>	Phase III routing message
<0000 1001>	Phase IV Level 2 routing message
<0000 1011>	Phase IV broadcast circuit router hello message
<0000 1101>	Phase IV broadcast circuit endnode hello message

- Find the adjacency using the message source address
- For Ethernet end nodes, use the designated router adjacency
- For broadcast circuit, first try the OA vector to look for a match in the ADJ database
- If no match, assume this message came from a broadcast router and scan the BRA portion of the ADJ vector
- Otherwise, scan the entire ADJ database for the node
- For nonbroadcast circuit, use ADJ index in the LPD
- Save the source ADJ index in the CXB
- Parse the message and dispatch to TR_RTHDR (CI message has a phase IV routing header)

7. TR_RTHDR - NETDRIVER (NETDRVXPT)

   - Process received message's route header

   - Check type of route header (phase and broadcast, or not) and process it

   - For a broadcast end node, update the endnode cache

   - Fall through to TR_ECL

8. TR_ECL - NETACP (NETDRVXPT)

   - Update ARRIVING PKTS RCVD and PMS (monitor) database

   - Set up the packet and call the ECL layer with:

     R7 Size of ECL message
     R6 Received CXB address
     R2 RCB address
     R1 Points to first byte in ECL message

CXB$L_R_RCB	RCB address (copy of R2)
CXB$L_R_MSG	Points to ECL message (copy of R1)
CXB$W_R_BCNT	Size of ECL message (copy of R7)
CXB$W_R_SRCNOD	Source node address
CXB$W_R_DSTNOD	Destination node (the ECL) address
CXB$B_R_FLG	Low bit clear if CXB can be consumed Low bit set if CXB must be returned Second bit clear if no return-to-sender packet Second bit set if packet returned-to-sender
CXB$W_R_PATH	I.D. of receiving LPD

   - Branch to NET$UNSOL_INTR

9. NET$UNSOL_INTR - NETDRIVER (NETDRVNSP)

- This "unsolicited interrupt" routine is called by routing whenever it has received a message addressed to NSP

- NSP must process the message completely and return to transport

- The message can be found in a single buffer of "complex chained" (CXB) format

- If NSP wishes to keep the message, it must zero the CXB pointer before returning to transport

- Map the message into an event code and check for message size violations

- For #NSP$C_MSG_CI (CONNECT INITIATE)

  Check to see if this is a message being returned because the remote node is unreachable

  For an incoming connect request - create an XWB

  Get source and remote node addresses for subroutine calls

  Check for reachability

  Get UCB address

  Get a new XWB and link slot

  Increment "connects received"

  Store the path over which the message was received in XWB$W_CI_PATH

  Start inbound timer since we will break the link if we time out before user issues the IO$_ACCESS function

  Store remote link address

- Branch to NET$EVENT to process the message

10. NET$EVENT - NETDRIVER (NETDRVNSP)

    - State table event dispatcher used to determine what is to be done and what state the XWB is to enter next

    - Dispatch to ACT$RCV_CI

11. ACT$RCV_CI - NETDRIVER (NETDRVNSP)

    - Parse the link characteristics

    - Parse the remainder of the message

    - Queue the XWB to NETACP by branching to NET$QUE_XWB

12. NET$QUE_XWB - NETDRIVER (NETDRVSES)

    - If the XWB is busy, then return

    - If not busy, then set the XWB$V_STS_SOL bit to lock it

    - Queue the XWB to the NETACP's AQB

    - Jump to SCH$WAKE to wake up NETACP

13. SCH$WAKE - VMS EXEC (RSE)

    - Wake up NETACP at NET$DISPATCH in NETTRN

14. NET$DISPATCH - NETACP (NETTRN)

- NETACP's major work dispatching loop
- Dispatch from AQB element to NET$PROC_XWB

15. NET$PROC_XWB - NETACP (NETPROCRE)

- NETDRIVER has passed us an XWB either to be linked into the LTB and assigned a local logical link address, or to be unhooked from the LTB and deallocated
- If both the XWB$W_REMLNK and XWB$W_LOCLNK fields are zero, then this request comes from the NETACP code that handles the IO$_ACCESS request for CIs
- NETACP is responsible for the LTB maintenance and the XWB linkage
- When the LTB slot and its place in the XWB list have been found, link XWB into the LTB and set up a local link number
- Create logical link and insert it into database
- Branch to NET$DELIVER_CI

16. NET$DELIVER_CI - NETACP (NETPROCRE)

- Determine whether the connect is to be handed to a task that has a declared name or an object type
- Initialize descriptors for process creation
- Get scratch buffer from NPP
- Branch to BUILD_NCB

17. BUILD_NCB - NETACP (NETPROCRE)

    - Build the NCB string for the connect

    - NCB will be passed later to destination process (in a number of different ways)

    - Get info from XWB

    - Return to NET$DELIVER_CI which branches to GET_PROC

18. GET_PROC - NETACP (NETPROCRE)

    - Find the OBI block for the local object

    - If the OBI is for a declared name or object then pass the NCB to the declaring process's mailbox

    - Otherwise, get ready to create a process

    - Look for an available server process to receive the connect

    - If there's a matching server process, then send the connect to the process

    - Otherwise, create the process

```
$CREPRC_S                        -    ; Create the user process
  INPUT=  NET_Q_PROC,            -    ; Network NETSERVER.COM filename
  OUTPUT= NET_Q_ACC,             -    ; Access control strings
  ERROR=  NET_Q_NCB,             -    ; First NCB (solely for LOGIN proxy)
  PRCNAM= NET_Q_PRC,             -    ; Process name
  IMAGE=  NET_Q_IMAGE,           -    ; Image (LOGINOUT) to run first
  PIDADR= NET_L_PID,             -    ; Place to store process id
  BASPRI= G^SYS$GB_DEFPRI,-           ; Priority
  UIC=    #<^O1@16+^O3>,         -    ; UIC is [1,3]
  STSFLG= #<STS_M_NETLOG>,-           ; This is a network process
  MBXUNT= MBX_UNIT                    ; MBX for termination notification
```

    - Return to NET$DELIVER_CI which branches to TELL_DRV

19. TELL_DRV - NETACP (NETPROCRE)

    - Branches to CALL_NETDRIVER

20. CALL_NETDRIVER - NETACP (NETACPTRN)

    - Jump to NET$ACP_COMM in NETDRIVER

21. NET$ACP_COMM - NETDRIVER (NETDRVSES)

    - Informs NETDRIVER of a change of status based on the function codes:

```
NETUPD$_CONNECT    - Pass NCB to Declared Name mailbox
NETUPD$_PROCRE     - Process created to received connect
NETUPD$_ABORT      - Process couldn't start
NETUPD$_EXIT       - Started process is exiting

NETUPD$_DLL_ON     - Datalink has come online - post a receive
NETUPD$_DLL_DLE    - Datalink online for service facts
NETUPD$_REACT_RCV  - Reactivate datalink receiver
NETUPD$_SEND_HELLO - Force datalink to send a hello message

NETUPD$_CRELNK     - Create a logical link control structure
NETUPD$_DSCLNK     - Graceful disconnect of single link
NETUPD$_ABOLNK     - Force immediate disconnect of all links

NETUPD$_BRDCST     - Broadcast mailbox message
NETUPD$_REPLY      - Reply to associated mailbox
```

    - Inform NETDRIVER of process created to receive connect and return

22. On return we have to wait for the requested task to issue a connect accept or connect reject

    Another possibility is an incoming timer timeout if the requested task does not respond

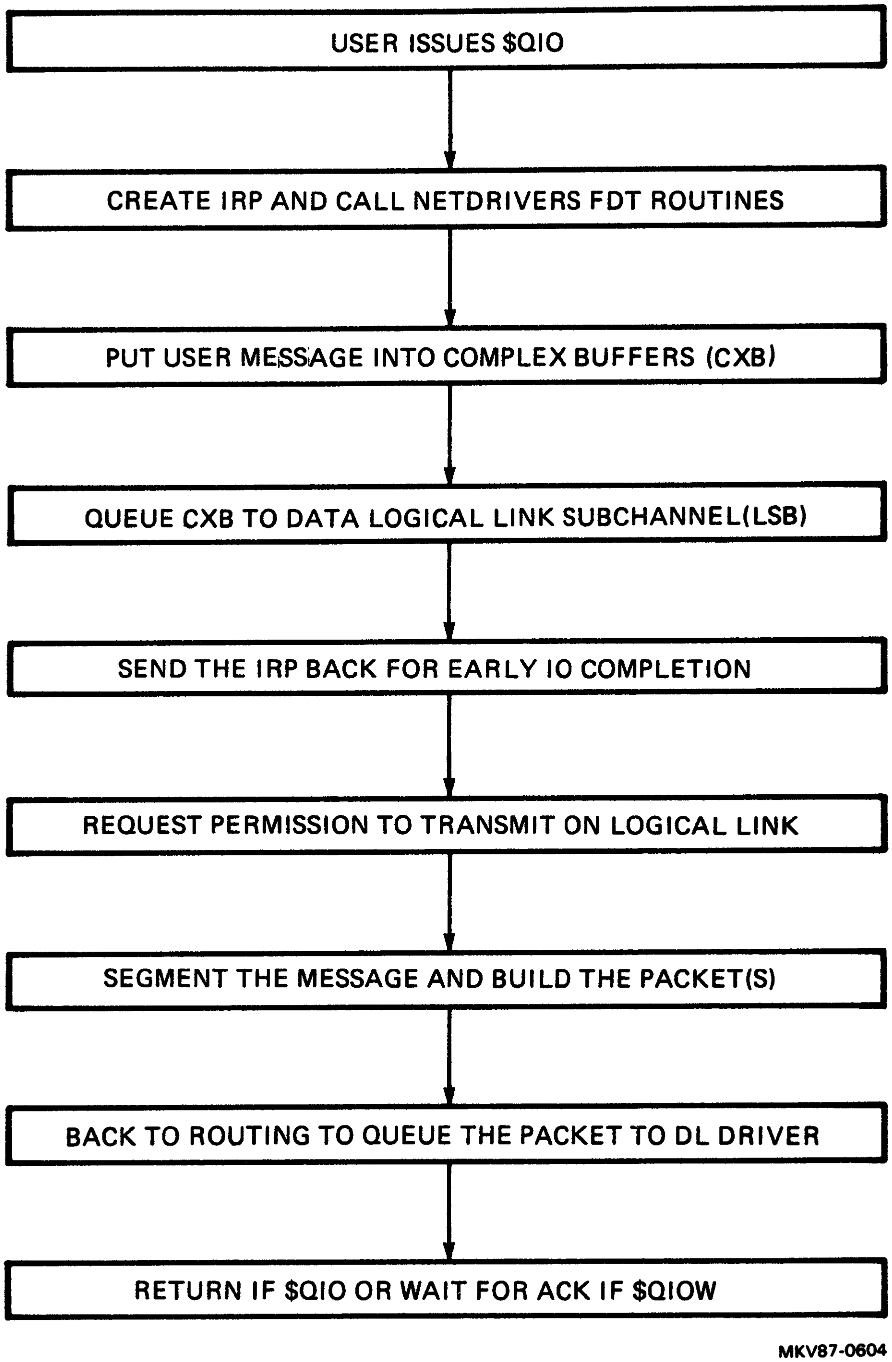


Figure 7-5 Simplified Transmission Flow of Normal Data

3.3 Overview of Transmission of Normal Data

A. User program issues $QIO

B. $QIO calls NETDRIVER which dispatches to transmit routines

C. The user message is segmented into CXBs (complex buffers)

D. The CXB buffers are queued to the data LSB (logical link subchannel)

E. Request permission to transmit on the logical link - put as many messages as possible in the pipeline

F. Check with routing for permission to transmit

G. Transmit at the appropriate time

H. Return success to the user

3.3.1 Transmission of Normal Data

- User program issues $QIO

```
SYS$QIOW(,%VAL(NET_CHAN),%VAL(IO$_WRITEVBLK),IOSB,,,
1          DATA_ARRAY,%VAL(DATA_SIZE),,,,)
```

  - EXE$QIO forms IRP and enters the FDT routines in NETDRIVER
  - Dispatches to NET$_FDT_XMT

- NET$_FDT_XMT - NETDRIVER (NETDRVNSP)

  - Inputs:

```
AP    Pointer to the QIO P1 parameter
R8    Must be saved/restored on return to next FDT routine
R7    I/O function code without modifiers
R6    CCB address
R5    UCB address
R4    PCB address
R3    IRP address
```

  - Switch to XWB context
  - Get data LSB
  - Segment the data message into CXB buffers
  - Jump to XMT_COPY
  - On return the CXB buffers are queued to the data LSB
  - These CXBS are to be passed to the routing layer for transmission at the appropriate time

- XMT_COPY - NETDRIVER (NETDRVNSP)
  - Get a free CXB and enter message type code into the CXB
  - Segment the message - XWB$W_REMSIZ bytes at a time
  - Process that segment and each segment from BOM to EOM flags
  - Update user virtual address descriptor in IRP and copy data into CXB
  - Set up NSP info (SEG #, ACK #)
  - Attach CXB to CXB queue and GOTO XMT_REQ_DONE_OK
- XMT_REQ_DONE_OK - NETDRIVER (NETDRVNSP)
  - Remove IRP from LSB$L_X_PND (pending LSB)
  - Make sure we can transmit the packet
  - Try to maximize the transmit pipeline
  - Post I/O by means of COM$POST and signal early I/O completion
  - Return to XMIT_RCV_CO which branches to NET$SCH_MSG
- Schedule message for transmission by means of NSP$SOLICIT which calls the routing layer of NETDRIVER

NOTE

The remainder of the path is the same back to EXE$QIO as in sending a CI.

4 OTHER DECnet MECHANISMS

4.1 Transmission of Interrupt Data

A. Standard VAX/VMS QIO interface

   Logical link has been established. All DECnet data structures are present

B. If NETDRIVER is busy, the IRP is queued to NETDRIVER AND control is returned

C. If NETDRIVER is not busy, the routine IOC$INITIATE in IOSUBNPAG starts the NETDRIVER

D. Queue the IRP in the Logical Subchannel Block (LSB) for interrupt messages

   Only one interrupt message is allowed per logical link

E. The appropriate Logical Path Descriptor (LPD) is found

F. An internal IRP is queued to the data link driver and the message is transmitted

4.2 Reception of Normal Data

A. Standard VAX/VMS QIO interface

Logical link has been established. All DECnet data structures are present

B. The FDT routines set the type of buffer to use as complex chained buffers (CXBs).

Control is passed to the main code of NETDRIVER routing layer

C. If data is received, it will be copied from the CXBS into the user area

D. If no data is received, a link service message (to update flow control count) will be sent

E. The flow is very much like the transmission of normal data except that in this case link service is sent

4.3 Reception of Interrupt Data

A. The data link driver posts outstanding receives using an internal IRP with a routine in NETDRVXPT (TR$RCV_DATA)

Logical link has been established. All DECnet data structures are present

B. The message is for the local node and is an interrupt message

C. The message is written to the user's mailbox

D. The user reads the mailbox message (write attention ast or synchronous mailbox read)

4.4 Sending a Disconnect Initiate

A. Standard VAX/VMS QIO interface

B. The function code is:

IO$_DEACCESS!IO$M_SYNCH OR IO$_DEACCESS!IO$M_ABORT

C. Control is passed to NETACP, which requeues packet back to the data link driver

D. NETDRIVER is called at its start IO entry point

E. The reason for the disconnect and the optional disconnect, data is copied to the XWB

F. The XWB would be queued to the routing module

The admission policy of routing determines whether the message can be sent out

G. Internal IRP is queued to the data link driver to get the message out

4.5 Receiving a Disconnect Initiate

A. Data link driver always posts outstanding receives using an internal IRP with a routine in NETDRVXPT (TR$RCV_DATA)

B. A special IRP is used to associate the message buffer containing the disconnect initiate message

C. The user is notified by means of MSG$_DISCON message written to the mailbox

D. A disconnect confirm is sent back to the source node

   The data structures associated with the logical link will be deallocated

4.6 Receiving a Disconnect Confirm

A. A process similar to receiving a disconnect initiate happens when receiving a disconnect confirm

4.7 Processing ACP Control Functions

A. Standard QIO interface

B. NETDRIVER FDT routines pass control to NETACP

C. NETCTLALL deals with most of the control functions

D. Some are requeued to NETDRIVER (turning lines off/on)

E. Dispatch on NFB function

```
<NFB$C_LOGEVENT,  NET$LOG_EVENT>,-
<NFB$C_READEVENT, NET$READ_EVENT>,-

<NFB$C_DECLNAME,  DCL_NAME>,-
<NFB$C_DECLOBJ,   DCL_OBJECT>,-
<NFB$C_DECLSERV,  DCL_SERVER>,-

<NFB$C_FC_SET,    CTL_DATABASE>,-
<NFB$C_FC_CLEAR,  CTL_DATABASE>,-
<NFB$C_FC_SHOW,   CTL_DATABASE>,-
<NFB$C_FC_DELETE, CTL_DATABASE>,-
<NFB$C_FC_ZERCOU, CTL_DATABASE>,-
```

5 SELECTED ROUTINE LISTINGS

5.1 XMT_COPY (NETDRVNSP.LIS)

```
NETDRVNSP                          - DECnet NSP module for NETDRIVER        22-MAR-1986 14:27:28  VAX/VMS Macro V04-00          Page
X-8                                ACT$RCV_DATA - Process rcv'd DATA messag 18-OCT-1985 17:52:07  [NETACP.SRC]NETDRVNSP.MAR;1

                                   09FB  2497          .SBTTL  ACT$RCV_DATA     - Process rcv'd DATA message
                                   09FB  2498 ;++
                                   09FB  2499 ;
                                   09FB  2500 ;  A received data segment is processed.  If it is acceptable then the
                                   09FB  2501 ;  IRP's message buffer (CXB) is moved to the LSB.

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

                                   0D94  3112 XMT_COPY:                                                  ;
         57    10 A8         D0    0D94  3113          MOVL    LSB$L_X_PND(R8),R7                      ; Get next IRP
                  B9         13    0D98  3114          BEQL    100$                                    ; If EQL, none left
               05 5B         E8    0D9A  3115          BLBS    R11,30$                                 ; If LBS, okay to goto IPL 2
      B2 0C A7    1F         E1    0D9D  3116          BBC     #31,IRP$L_PID(R7),400$                  ; If BC not ALTSTART, must
                                   0DA2  3117 30$:     ;
                                   0DA2  3118          ;
                                   0DA2  3119          ;     Get a free CXB.  Expand CXB list if needed and if possible.
                                   0DA2  3120          ;
                                   0DA2  3121          ;
      56    0118 D5          0F    0DA2  3122          REMQUE  @XWB$Q_FREE_CXB(R5),R6                  ; Get next CXB
                  28         1C    0DA7  3123          BVC     50$                                     ; If VC, got one
                  56         D4    0DA9  3124          CLRL    R6                                      ; Init pointer
               0F A8         91    0DAB  3125          CMPB    LSB$B_X_CXBCNT(R8),-                    ; Can we allocate another CXB ?
               0E A8               0DAE  3126                  LSB$B_X_CXBQUO(R8)                      ;
                  A1         1E    0DB0  3127          BGEQU   100$                                    ; If GEQ, no
         51    42 A5         3C    0DB2  3128          MOVZWL  XWB$W_REMSIZ(R5),R1                     ; Get remote size
                                   0DB6  3129                                                          ;; trim upon ENT_RUN if needed
         51    5B A1         9E    0DB6  3130          MOVAB    NSP$C_HSZ_DATA -                       ; Add in overhead
                                   0DBA  3131                  +TR3$C_HSZ_DATA -                       ;
                                   0DBA  3132                  +CXB$C_OVERHEAD(R1),R1                  ;
      00000000'GF            16    0DBA  3133          JSB     G^EXE$ALONONPAGED                       ; Allocate the buffer
               94 50         E9    0DC0  3134          BLBC    R0,200$                                 ; If LBC then allocation failur
            56    52         D0    0DC3  3135          MOVL    R2,R6                                   ; Setup CXB pointer
         0A A6    1B         90    0DC6  3136          MOVB    #DYN$C_CXB,CXB$B_TYPE(R6)               ; Setup block type
         08 A6    51         B0    0DCA  3137          MOVW     R1,CXB$W_SIZE(R6)                      ; Setup the size
               0F A8         96    0DCE  3138          INCB    LSB$B_X_CXBCNT(R8)                      ; Account for CXB
```

```
                      0DD1  50$:    ;
                      0DD1          ;
                      0DD1          ;      Enter message type code.  Process 'bom' and 'eom' flags.
                      0DD1          ;
                      0DD1          ;
                      0DD1          ASSUME   NSP$C_MSG_DATA   EQ  0            ; 'Data' message type code for
                      0DD1          ASSUME   NSP$V_DATA_BOM   EQ  LSB$V_BOM
                      0DD1          ASSUME   NSP$V_DATA_EOM   EQ  LSB$V_EOM
                                                                               ;
2B A8    DF 8F    8B  0DD1          BICB3    #^C<LSB$M_BOM>,LSB$B_STS(R8),-    ; Enter message type code
         4E A6        0DD6                   CXB$B_X_NSPTYP(R6)
                                                                               ;
   2B A8    20    8A  0DD8          BICB     #LSB$M_BOM,LSB$B_STS(R8)          ; Preset next type code
   52    42 A5    3C  0DDC          MOVZWL   XWB$W_REMSIZ(R5),R2               ; Get segment size
   3A A7    52    B1  0DE0          CMPW     R2,IRP$L_IOST1+2(R7)              ; More data left after this ?
            12    1F  0DE4          BLSSU    70$                               ; If LSSU, more data left
   52    3A A7    3C  0DE6          MOVZWL   IRP$L_IOST1+2(R7),R2              ; Else, take it all
09 20 A7    07    E0  0DEA          BBS      #IO$V_MULTIPLE,IRP$W_FUNC(R7),70$  ; If BS, not 'end of msg'
4E A6    40 8F    88  0DEF          BISB     #NSP$M_DATA_EOM,CXB$B_X_NSPTYP(R6) ; Set 'end of message flag'
   2B A8    20    88  0DF4          BISB     #LSB$M_BOM,LSB$B_STS(R8)           ; Preset next type code
                      0DF8  70$:    ;
                      0DF8          ;
                      0DF8          ;      Update user VA descriptor in IRP, copy data into CXB
                      0DF8          ;
                      0DF8          ;
04 A6    57 A6    9E  0DF8          MOVAB    CXB$T_X_DATA(R6),4(R6)            ; Setup destination pointer
   51    3C A7    D0  0DFD          MOVL     IRP$L_IOST2(R7),R1                ; Get address of user data
      66    51    D0  0E01          MOVL     R1,(R6)                           ; Save user VA
   0C A6    52    B0  0E04          MOVW     R2,CXB$W_LENGTH(R6)               ; Save # user bytes in CXB
   3C A7    52    C0  0E08          ADDL     R2,IRP$L_IOST2(R7)                ; Update address
   3A A7    52    A2  0E0C          SUBW     R2,IRP$L_IOST1+2(R7)              ; Consume bytes
      FF4A CF     9F  0E10          PUSHAB   XMT_COPY1                         ; Setup return address
                      0E14                                                     ; fall thru to COPY_DATA
                      0E14                   .DSABL  LSB
```

```
                        0E14
                        0E14      COPY_DATA:
     36 0C A7    1F  E0 0E14                BBS     #31,IRP$L_PID(R7),70$          ; If BS, then 'ALTSTART'
                        0E19                ;
                        0E19                ;
                        0E19                ;    Probe the user buffer.  The probing code relies on the fact that
                        0E19                ;    the first and last pages in the probe range are simultaneously
                        0E19                ;    probed -- hence we can probe two pages at a time.
                        0E19                ;
                        0E19                ;    Enter data into message.  Since we may be at IPL 2, it is possible
                        0E19                ;    that an NET$C_IPL event could cause the link to break and the
                        0E19                ;    IRP's to be cleaned up.  Therefore, the IRP cannot be referenced
                        0E19                ;    once we go below NET$C_IPL.
                        0E19                ;
                        0E19                ;
54   0B A7   02  00  EF 0E19                EXTZV   #0,#2,IRP$B_RMOD(R7),R4        ; Get request access mode
        53   FE00 8F 32 0E1F                CVTWL   #-512,R3                       ; Set addition constant
        50   51  52  C1 0E24                ADDL3   R2,R1,R0                       ; Calc end of buffer
        51   01FF 8F AA 0E28                BICW    #VA$M_BYTE,R1                  ; Must go to begining of page
                        0E2D                                                       ; (since two pages worth of dat
                        0E2D                                                       ; could otherwise span 3 frames
             50  51  C2 0E2D                SUBL    R1,R0                          ; Calc # of bytes to probe
                        0E30                SETIPL  #IPL$_ASTDEL                   ; Allow paging
                        0E33                                                       ;
     08 2A A7    01  E1 0E33      30$:      BBC     #IRP$V_FUNC,IRP$W_STS(R7),50$  ; If BC, IO$_WRITELBLK
                        0E38                                                       ; (IRP$W_STS is remains valid
                        0E38                                                       ;  even if IRP has been sent to
                        0E38                                                       ;  IOPOST).
        61   50  54  0D 0E38                PROBEW  R4,R0,(R1)                     ; Can user VA be written ?
                 08  12 0E3C                BNEQ    60$                            ; If NEQ, yes
                 26  11 0E3E                BRB     200$                           ; Report access violation
                        0E40                                                       ;
        61   50  54  0C 0E40      50$:      PROBER  R4,R0,(R1)                     ; Can user VA be read ?
                 20  13 0E44                BEQL    200$                           ; If EQL no, report error
             51  53  C2 0E46      60$:      SUBL    R3,R1                          ; Update address of buffer
          50   6043  3E 0E49                MOVAW   (R0)[R3],R0                    ; Shrink total by 2 pages
                 E4  14 0E4D                BGTR    30$                            ; If GTR, more to probe
                        0E4F                                                       ;
  04 B6   00 B6  52  28 0E4F      70$:      MOVC3   R2,@(R6),@4(R6)                ; Enter data
                        0E55                                                       ;
                        0E55      100$:     SETIPL  #NET$C_IPL                     ; Go back to synchronizing IPL
        55   FF5C C8 9E 0E58                MOVAB   -XWB$T_DT(R8),R5               ; Recover XWB address
     14 1C A5    07  E1 0E5D                BBC     #XWB$V_FLG_SDT,XWB$W_FLG(R5),210$ ; If BC, not in RUN state
             50  01  90 0E62                MOVB    #1,R0                          ; Say "success"
                     05 0E65                RSB                                    ; Done
```

5.2 GET PROC (NETPROCRE.LIS)

```
NETPROCRE                       - Process creation                          22-MAR-1986 14:20:07  VAX/VMS Macro V04-00        Page
X-5                             NET$DELIVER_CI - Process and Deliver Inb     7-NOV-1985 11:42:36  [NETACP.SRC]NETPROCRE.MAR;1

        00000010'EF   50   D0  05B5  1141         MOVL    R0,NET_A_NCB               ; save its address in case NCB
                               05BC  1142                                            ; is to be passed to NETDRIVER
                               05BC  1143                                            ; for a declared name
                 00C2      30  05BC  1144         BSBW    GET_PROC                   ; Find/create process to
                               05BF  1145                                            ; receive the connect
                 050A      30  05BF  1146 10$:    BSBW    TELL_DRV                   ; Tell driver about connect
   50    00000018'EF       D0  05C2  1147         MOVL    PTR_NCB_BUF,R0             ; Address of buffer
                FA34'      30  05C9  1148         BSBW    NET$DEALLOCATE             ; Deallocate the buffer
   50    0000001C'EF       D0  05CC  1149         MOVL    PTR_CON_BUF,R0             ; Address of scratch buffer
                FA2A'      30  05D3  1150         BSBW    NET$DEALLOCATE             ; Deallocate scratch storage
                           05  05D6  1151         RSB                                ; Done
                               05D7  1152
                               0681  1231 .SBTTL  GET_PROC        - Locate process to accept connect
                               0681  1232 ;+
                               0681  1233 ;
                               0681  1234 ; Find the OBI block associated with the local object.  If the OBI is
                               0681  1235 ; for a declared name or object then pass the NCB to the declaring
                               0681  1236 ; process's mailbox, otherwise create a process to receive the connect.
                               0681  1237 ; If there is a server process waiting for more work, then tell the
                               0681  1238 ; server process that it can have the connect request.
                               0681  1239 ;
                               0681  1240 ; Inputs:
                               0681  1241 ;
                               0681  1242 ;        R6 = XWB address
                               0681  1243 ;
                               0681  1244 ;        Own storage
                               0681  1245 ;
                               0681  1246 ; Outputs:
                               0681  1247 ;
                               0681  1248 ;        None
                               0681  1249 ;-
                               0681  1250 GET_PROC:                                 ; Get process to accept the connect
  5B    00000000'EF        D0  0681  1251         MOVL    NET$GL_CNR_OBI,R11         ; Set up OBI CNR
     51     00A5 C6        9E  0688  1252         MOVAB   XWB$T_LPRNAM(R6),R1        ; Address local task specifier
                 03F0      30  068D  1253         BSBW    GET_PR_ZNA                 ; Get its ZNA field
                31 50      E9  0690  1254         BLBC    R0,10$                     ; If LBC then format error
```

```
                                0693  1255            ;
                                0693  1256            ;
                                0693  1257            ;   Find the OBI CNF
                                0693  1258            ;
                                0693  1259            ;
                 04      3C     0693  1260            MOVZWL   #NET$C_DR_NOBJ,-          ; Assume failure due to unknown object
         00000008'EF            0695  1261                          NET_L_REASON           ;
                 5A      D4     069A  1262            CLRL     R10                       ; Indicate no current CNF
                                069C  1263            $SEARCH eql,obi,s,zna              ; Find OBI block with this CNF
              19 50      E8     06AB  1264            BLBS     R0,20$                    ; If LBS then CNF was found
                 68      95     06AE  1265            TSTB     (R8)                      ; Is this a numbered object connect ?
                 12      12     06B0  1266            BNEQ     10$                       ; If NEQ then no such object
57       0000000B'EF     7D     06B2  1267            MOVQ     NET_Q_TASKZNA,R7          ; Else use default TASK  ZNA descriptor
           51    00      9A     06B9  1268            MOVZBL   S^#NFB$C_OP_EQL,R1        ; Specify match operator
                 5A      D4     06BC  1269            CLRL     R10                       ; Start from head of list
               F93F'     30     06BE  1270            BSBW     CNF$KEY_SEARCH            ; Look for the CNF
              5C 50      E8     06C1  1271            BLBS     R0,25$                    ; If LBS then found, br to continue
               029F      31     06C4  1272 10$:       BRW      100$                      ; Complete with error
                                06C7  1273            ;
                                06C7  1274            ;
                                06C7  1275            ;   The OBI CNF has been found.  See if the object has been "declared"
                                06C7  1276            ;   If not, build the .COM file file i.d. and setup its descriptor.
                                06C7  1277            ;
                                06C7  1278            ;
                 0D      E1     06C7  1279 20$:       BBC      #XWB$V_STS_ALIAS,-        ; Cluster link?
           13 0E A6             06C9  1280                      XWB$W_STS(R6),22$        ; ... if BC, no
                                06CC  1281            $GETFLD obi,v,ali                  ; Get OBJECT ALIAS INBOUND flag
              03 50      E9     06D9  1282            BLBC     R0,22$                    ; If LBC, not set; default ENABLED
                                06DC  1283            ASSUME   NMA$C_ALINC_ENA EQ 0
                                06DC  1284            ASSUME   NMA$C_ALINC_DIS EQ 1
              E5 58      E8     06DC  1285            BLBS     R8,10$                    ; If not enabled, no such object
                                06DF  1286 22$:       $GETFLD obi,l,ucb                  ; Get the associated UCB
00000014'EF      58      D0     06EF  1288            MOVL     R8,NET_L_UCB              ; Save the UCB pointer
                                06F6  1289            $GETFLD obi,l,pid                  ; Get the declarer's EPID
              4B 50      E9     0703  1290            BLBC     R0,30$                    ; If LBC then treat as undeclared
           50    58      D0     0706  1291            MOVL     R8,R0                     ; Convert from EPID to IPID
         00000000'GF     16     0709  1292            JSB      G^EXE$EPID_TO_IPID        ; ...
00000004'EF      50      D0     070F  1293            MOVL     R0,NET_L_PID              ; Save the PID
                 02      9A     0716  1294            MOVZBL   #NETUPD$_CONNECT,-        ; Setup the function code
         00000000'EF            0718  1295                          NET_L_FCT              ;
               0246      31     071D  1296            BRW      100$                      ; Return to pass NCB to mailbox
                                0720  1297            ;
```

```
                                0720       ;    The object is a named object which could not be found in the
                                0720       ;    object database.  Use the requested object name to construct
                                0720       ;    the name of the command procedure, rather than consulting the
                                0720       ;    OBI entry (we are currently set to the "TASK" OBI).  If the
                                0720       ;    object name starts with a "$", then the object is "reserved
                                0720       ;    to DEC", and we get the command procedure from SYS$SYSTEM.
                                0720       ;
      51   00A5 C6        9E    0720  25$: MOVAB    XWB$T_LPRNAM(R6),R1     ; Address local task specifier
                034C      30    0725       BSBW     GET_PR_NAM              ; Get its name
  53   00000038'EF        D0    0728       MOVL     NET_Q_TSK+4,R3          ; Get address of output buffer
          24     68       91    072F       CMPB     (R8),#^A"$"             ; Does the name start with "$"?
                 10       12    0732       BNEQ     28$                     ; If so,
                 58       D6    0734       INCL     R8                      ; Strip "$" off front of name
                 57       D7    0736       DECL     R7
      00000032'EF         28    0738       MOVC     NET_Q_SYSTEM,-          ; Prefix name with "SYS$SYSTEM:"
  63  00000036'FF               073E                @NET_Q_SYSTEM+4,(R3)
     63    68    57       28    0744  28$: MOVC3    R7,(R8),(R3)            ; Move the name
  00000034'EF    53       C0    0748       ADDL     R3,NET_Q_TSK            ; Update filename size
                 31       11    074F       BRB      40$                     ; Continue
                                0751       ;
                                0751       ;    Build filespec of object command procedure
                                0751       ;
                 04       3C    0751  30$: MOVZWL   #NET$C_DR_NOBJ,-        ; Assume error
      00000008'EF               0753                      NET_L_REASON      ;
                                0758       $GETFLD obi,s,sfi                ; Get parsed file id
                                0765       BLBC_W   R0,55$                  ; If LBC then file id is invalid
  00000034'EF    57       7D    076B       MOVQ     R7,NET_Q_TSK            ; Update filename descriptor
                                0772       ;
                                0772       ;
                                0772       ;    Create a process name.
                                0772       ;
                                0772       ;
                                0772       $GETFLD obi,s,nam                ; Get object name for prefix
               07 50      E8    077F       BLBS     R0,50$                  ; If LBS then name was found
  57   00000000'EF        7D    0782  40$: MOVQ     NET_Q_NETPREFIX,R7      ; Setup standard prefix descriptor
                                0789  50$: ;
                                0789       ;    Process name is either NET_linkid or OBJNAM_linkid. If it's
                                0789       ;    the latter, make sure the full process name won't exceed
                                0789       ;    15 characters in length. Since the link ID can be 5 characters,
                                0789       ;    and the underscore takes up 1, we'll truncate the object name
                                0789       ;    after 9 (it's the link ID portion which is guaranteed unique).
                                0789       ;
          09     57       B1    0789       CMPW     R7,#MAX_PREFIX          ; Check length of prefix
                 03       15    078C       BLEQ     53$                     ; If LEQ, it's short enough
          57     09       D0    078E       MOVL     #MAX_PREFIX,R7          ; Truncate prefix as needed
00000030'FF  68  57       28    0791  53$: MOVC3    R7,(R8),@NET_Q_PRC+4    ; Move the prefix
       83     5F 8F       90    0799       MOVB     #^A'_',(R3)+            ; Move the delimiter
```

```
50   0000000C'EF   D0  079D  1345          MOVL    NET_L_LNK,R0              ; Get the local link number
            F859'  30  07A4  1346          BSBW    NET$BIN2ASC               ; Convert to ascii and append as
                       07A7  1347                                            ; the suffix
0000002C'EF    53  C0  07A7  1348          ADDL    R3,NET_Q_PRC              ; Done with process name
                       07AE  1349          ;
                       07AE  1350          ;   If the connect did not use format type 2, then don't attempt
                       07AE  1351          ;   a proxy login.
                       07AE  1352          ;
     02   00B9 C6  91  07AE  1353          CMPB    XWB$T_RPRNAM(R6),#2       ; Format type 2?
               07  13  07B3  1354          BEQL    51$                       ; Branch if so
0000004A'EF    00  90  07B5  1355          MOVB    #NMA$C_ACES_NONE,INT_B_PRX ; Disallow proxy access
                       07BC  1356 51$:     ;
                       07BC  1357          ;
                       07BC  1358          ;   If no access control was specified, use default from OBI block
                       07BC  1359          ;
                       07BC  1360          ;
                       07BC  1361          $GETFLD obi,l,prx                 ; Get proxy login state
            07 50  E9  07C9  1362          BLBC    R0,52$                    ; If LBC then none specified
 00000049'EF   58  90  07CC  1363          MOVB    R8,OBI_B_PRX              ; Store it
     58   00CC C6  9E  07D3  1364 52$:     MOVAB   XWB$B_LOGIN(R6),R8        ; Get address of access info
           57  88  9A  07D8  1365          MOVZBL  (R8)+,R7                  ; Get total size
           03  57  91  07DB  1366          CMPB    R7,#3                     ; Is it 3 null (counted) strings
               17  13  07DE  1367          BEQL    60$                       ; If so use access info in OBI
               00  90  07E0  1368          MOVB    #NMA$C_ACES_NONE,-        ; Disallow proxy access
      0000004A'EF      07E2  1369                      INT_B_PRX             ; Store it
       75 8F   57  91  07E7  1370          CMPB    R7,#NET$C_MAXACCFLD*3     ; Too long ?
               17  1B  07EB  1371          BLEQU   70$                       ; If LEQU then move the strings
               2B  3C  07ED  1372          MOVZWL  #NET$C_DR_IMLONG,-        ; Indicate network failure type
      00000008'EF      07EF  1373                      NET_L_REASON          ;
             016F  31  07F4  1374 55$:     BRW     100$                      ; Continue
                       07F7  1375                                            ;
                       07F7  1376 60$:     $GETFLD obi,s,iac                 ; Get inbound access control
                       0804  1377 70$:     ;
                       0804  1378          ;
                       0804  1379          ;   Enter the flags word followed by the access control strings
                       0804  1380          ;
                       0804  1381          ;
53   00000040'EF   D0  0804  1382          MOVL    NET_Q_ACC+4,R3            ; Get pointer to access control buffer
               83  B4  080B  1383          CLRW    (R3)+                     ; Clear the flags word
```

```
                                   080D      $DISPATCH  TYPE=B,INT_B_PRX -   ; Don't set flag if proxy disallowed
                                   080D      <-
                                   080D        <NMA$C_ACES_OUTG, 80$>-
                                   080D        <NMA$C_ACES_NONE, 80$>-
                                   080D      >
                                   081B      $DISPATCH  TYPE=B,OBI_B_PRX -   ; Don't set flag if proxy disallowed
                                   081B      <-
                                   081B        <NMA$C_ACES_OUTG, 80$>-
                                   081B        <NMA$C_ACES_NONE, 80$>-
                                   081B      >
          FE A3    01    A8        0829      BISW     #1,-2(R3)              ; Say "proxy login allowed"
       63    68    57    28        082D 80$: MOVC3    R7,(R8),(R3)           ; Move access control strings,
                                   0831                                      ; even if it's null
   0000003C'EF     53    C0        0831      ADDL     R3,NET_Q_ACC           ; Complete string size calc.
                 02AA    30        0838      BSBW     UP_CASE                ; Up-case all pertinent strings
                                   083B      ;
                                   083B      ;   Attempt to find an available server process which is waiting
                                   083B      ;
   5B     00000000'EF    D0        083B      MOVL     NET$GL_CNR_SPI,R11     ; Get root of SPI database
                   5A    D4        0842      CLRL     R10                    ; Start at beginning of list
                   58    D4        0844 81$: CLRL     R8                     ; Search key is zero
                                   0846      $SEARCH  neq,spi,l,irp          ; Find an SPI with an IRP NE 0
                03 50    E8        0856      BLBS     R0,82$                 ; Br if found, check process
                 0093    31        0859      BRW      89$                    ; Else, create process
                34 A6    D5        085C 82$: TSTL     XWB$L_PID(R6)          ; Is this connect "tagged" for a
                   16    13        085F      BEQL     83$                    ; specific process?
                                   0861      $GETFLD  spi,l,pid              ; If so, get PID of this server
                D3 50    E9        086E      BLBC     R0,81$                 ; (if not present, error, skip entry)
          34 A6    58    D1        0871      CMPL     R8,XWB$L_PID(R6)       ; Is this server the intended process?
                   CD    12        0875      BNEQ     81$                    ; If not, then continue searching
                                   0877 83$: ;
                                   0877      ;   Always check the access control, even for processes started
                                   0877      ;   with proxy requested.  This way, if different default access
                                   0877      ;   control is used (each object can specify a unique account,
                                   0877      ;   including NONE), the wrong process isn't matched.
                                   0877      ;
                                   0877      $GETFLD  spi,s,acs              ; Get ACS for server process
                BD 50    E9        0884      BLBC     R0,81$                 ; (if not present, error, skip entry)
      50  0000003C'EF    7D        0887      MOVQ     NET_Q_ACC,R0           ; Get access string for new connect
61    50    00    68    57    2D   088E      CMPC5    R7,(R8),#0,R0,(R1)     ; Does it match?
                   AE    12        0894      BNEQ     81$                    ; If no match, keep searching
```

```
                                0896       ;
                                0896       ;   Make sure the process's "proxy request" flag matches.
                                0896       ;
                                0896       $GETFLD spi,v,prl                 ; Get proxy login flag
                   9E 50   E9   08A3       BLBC    R0,81$                    ; (if not present, error, skip entry)
58  00000040'FF  01   00   ED   08A6       CMPZV   #0,#1,@NET_Q_ACC+4,R8     ; Does proxy login flag match?
                      93   12   08AF       BNEQ    81$                       ; If not, try to find another server
                                08B1       ;
                                08B1       ;   For logical links which request proxy access, require
                                08B1       ;   that the requesting node and username match as well.
                                08B1       ;
                   36 58   E9   08B1       BLBC    R8,87$                    ; If proxy requested,
                                08B4       $GETFLD spi,l,rna                 ; Get remote node address for server
                   80 50   E9   08C1       BLBC    R0,81$                    ; (if not present, error, skip entry)
              58   3A A6   B1   08C4       CMPW    XWB$W_REMNOD(R6),R8       ; Is it the same node as the connect?
                                08C8       BNEQ_W  81$                       ; If not, try to find another server
                                08CD       $GETFLD spi,s,rid                 ; Get remote user ID for server
                   0F 50   E9   08DA       BLBC    R0,88$                    ; (if not present, error, skip entry)
              50   6F A6   9A   08DD       MOVZBL  XWB$B_RID(R6),R0          ; Get length of RID for new connect
  70 A6   50   00   68  57   2D   08E1       CMPC5   R7,(R8),#0,R0,XWB$T_RID(R6) ; Does it match?
                      02   12   08E8       BNEQ    88$                       ; If no match, then skip it
                      7B   11   08EA  87$: BRB     SEND_TO_SERVER            ; Server ok, send it the connect
                                08EC
                    FF55   31   08EC  88$: BRW     81$                       ; (Branch helper to top of loop)
                                08EF  89$: ;
                                08EF       ;
                                08EF       ;   Create the user process
                                08EF       ;
                                08EF       $CREPRC_S       -                 ; create a process
                                08EF           INPUT=  NET_Q_PROC,-          ; Network NETSERVER.COM filename
                                08EF           OUTPUT= NET_Q_ACC,-           ; Access control strings
                                08EF           ERROR=  NET_Q_NCB,-           ; 1st NCB (solely for LOGIN proxy use)
                                08EF           PRCNAM= NET_Q_PRC,-           ; Process name
                                08EF           IMAGE=  NET_Q_IMAGE,-         ; Image (LOGINOUT) to run first
                                08EF           PIDADR= NET_L_PID,-           ; Place to store process id
                                08EF           BASPRI= G^SYS$GB_DEFPRI,-     ; Priority
                                08EF           UIC=    #<^O1@16+^O3>,-       ; UIC is [1,3]
                                08EF           STSFLG= #<STS_M_NETLOG>,-     ; This is a network process
                                08EF           MBXUNT= MBX_UNIT              ; MBX for termination
                                0939                                         ; notification
                   09 50   E8   0939       BLBS    R0,90$                    ; If LBS process was created
                      01   3C   093C       MOVZWL  #NET$C_DR_RSU,-           ; Assume because couldn't get
             00000008'EF        093E               NET_L_REASON              ; the resources
                      21   11   0943       BRB     100$                      ; Take common exit
```

```
50    00000004'EF   D0  0945   1470 90$:    MOVL     NET_L_PID,R0            ; Get the EPID returned by CREPRC
          58    50   D0  094C   1471         MOVL     R0,R8                   ; Save EPID
      00000000'GF    16  094F   1472         JSB      G^EXE$EPID_TO_IPID      ; Convert to internal PID format
00000004'EF     50   D0  0955   1473         MOVL     R0,NET_L_PID            ; Use internal format of PID
                04   3C  095C   1474         MOVZWL   #NETUPD$_PROCRE,-       ; Say "process created"
      00000000'EF        095E   1475                       NET_L_FCT          ;
                         0963   1476         ;
                         0963   1477         ;    The network process is created.  Now create an SPI database entry
                         0963   1478         ;    so we can keep track of it.
                         0963   1479         ;
              0097   30  0963   1480         BSBW     CREATE_SPI              ; Create SPI database entry
                         0966   1481                                          ; Ignore errors if can't be inserted
                     05  0966   1482 100$:   RSB                              ; Common exit
```

5.3 UPDATE_CACHE (NETDRVXPT.LIS)


```
0EAD           .SBTTL  UPDATE_CACHE    - Update the BC cache table
0EAD
0EAD  ;+
0EAD  ;  UPDATE_CACHE - Update the BC cache table
0EAD  ;
0EAD  ;  INPUTS:        R10     Scratch
0EAD  ;                 R9      ADJ address
0EAD  ;                 R8      LPD address associated with receiving datalink
0EAD  ;                 R7      Size of ECL message
0EAD  ;                 R6      Received CXB address
0EAD  ;                 R5      Contents of first byte in message
0EAD  ;                 R4,R3   Scratch
0EAD  ;                 R2      RCB address
0EAD  ;                 R1      Ptr to source node address in message
0EAD  ;                 R0      Destination node address
0EAD  ;
0EAD  ;                 CXB$W_R_SRCNOD  "Last Hop" node address
0EAD  ;
0EAD  ;
0EAD  ;  OUTPUTS:
0EAD  ;                 R3,R4,R10       Garbage
0EAD  ;                 All other registers are preserved.
0EAD  ;
0EAD  ;
0EAD  ;-
0EAD  UPDATE_CACHE:                                   ; Update the LPD's cache table
0EAD            ;
0EAD            ;
0EAD            ;       First we will check the source node address
0EAD            ;       against the PNA for the DRT. If they match, then
0EAD            ;       it must be the "Designated Router" (DRT) who sent the
0EAD            ;       message, since the "Main Adjacency" would have a node
0EAD            ;       address of -1. We will then set the ADJ to point to the
0EAD            ;       DRT, else we will scan the CACHE table for the received
0EAD            ;       LPD, treating this like a Non-BC circuit and use the ADJ
0EAD            ;       index of the LPD.
0EAD            ;
0EAD            ;   CACHE TABLE HANDLING:
0EAD            ;
0EAD            ;       If the DRT is not a real BRA, then we will scan the LPD
0EAD            ;       CACHE table to try and find the entry. If the entry was not
0EAD            ;       found then it will be inserted at the first available slot,
0EAD            ;       as long as the Intra-NI bit is set or the source of the packet
0EAD            ;       was the same as the last hop.
0EAD            ;
```

```
     53     61    3C  0EAD  3069         MOVZWL  (R1),R3                  ; Get the source node address
  04 A9     53    B1  0EB0  3070         CMPW    R3,ADJ$W_PNA(R9)         ; Do the node addresses match?
            43    13  0EB4  3071         BEQL    100$                     ; Br if YES - must have come from
                      0EB6  3072                                          ;  the "Designated Router", skip it
  5A     66 A8    D0  0EB6  3073         MOVL    LPD$L_CACHE(R8),R10      ; Else, get the CACHE table for LPD
            3D    13  0EBA  3074         BEQL    100$                     ; Br if none available - leave now
  54     FA AA    3C  0EBC  3075         MOVZWL  -6(R10),R4               ; Get number of entries in CACHE
                      0EC0  3076         ;
                      0EC0  3077         ;       Scan CACHE
                      0EC0  3078         ;
     53     8A    B1  0EC0  3079 10$:    CMPW    (R10)+,R3                ; Node address in cache?
            2D    13  0EC3  3080         BEQL    60$                      ; Br if yes
            8A    B5  0EC5  3081         TSTW    (R10)+                   ; Skip timer cell
         F6 54    F5  0EC7  3082         SOBGTR  R4,10$                   ; Loop if more
                      0ECA  3083         ;
                      0ECA  3084         ;       CACHE scan failed, find empty cell and enter new Node
                      0ECA  3085         ;       address into the CACHE. If an empty cell is not found,
                      0ECA  3086         ;       then throw the oldest entry away!
                      0ECA  3087         ;
                      0ECA  3088         ;       Make sure the Intra-NI bit is set before entering in CACHE.
                      0ECA  3089         ;
  2B 55     05    E1  0ECA  3090         BBC     #TR4$V_RTFLG_INI,R5,100$ ; Br if Intra-NI packet, insert entry
  5A     66 A8    D0  0ECE  3091         MOVL    LPD$L_CACHE(R8),R10      ; Get the CACHE table for LPD, again
  54     FA AA    3C  0ED2  3092         MOVZWL  -6(R10),R4               ; Get size of CACHE table
     53     5A    D0  0ED6  3093         MOVL    R10,R3                   ; Make a copy of the oldest entry
                      0ED9  3094                                          ; ..assume first is oldest
            6A    D5  0ED9  3095 30$:    TSTL    (R10)                    ; Empty entry?
            12    13  0EDB  3096         BEQL    50$                      ; Br if yes
02 A3    02 AA    B1  0EDD  3097         CMPW    2(R10),2(R3)             ; Is this the new oldest?
            03    14  0EE2  3098         BGTR    40$                      ; Br if not
     53     5A    D0  0EE4  3099         MOVL    R10,R3                   ; Else, set new oldest
            8A    D5  0EE7  3100 40$:    TSTL    (R10)+                   ; Skip to next
         ED 54    F5  0EE9  3101         SOBGTR  R4,30$                   ; Loop if more
     5A     53    D0  0EEC  3102         MOVL    R3,R10                   ; Else, purge the oldest entry
                      0EEF  3103 50$:    ;
                      0EEF  3104         ;       Enter new Node Address into CACHE table.
                      0EEF  3105         ;
     8A     61    B0  0EEF  3106         MOVW    (R1),(R10)+              ; Enter new node address
8A  00000000'GF   B0  0EF2  3107 60$:    MOVW    G^EXE$GL_ABSTIM,(R10)+   ; Enter current time
                  05  0EF9  3108 100$:   RSB                              ; Return to caller
```

NETWORK DESIGN AND PERFORMANCE CONSIDERATIONS

INTRODUCTION

The network manager must understand which factors influence the performance of the network and the relationship between these factors.

This chapter discusses some of the basic network design and performance considerations.

Topics include:

- Buffers
- DECnet Transfer Process
- NETACP
- Routing Considerations
- Timers
- VMS Nonpaged Pool Parameters
- Other Performance Considerations

1 BUFFERS

- Allocated from pool
- Faster lines perform better with larger buffers
- Error-prone lines (asynchronous) do better with smaller buffers
- Buffer size passed in logical link request
- Receive buffers
  - Default buffer size set for executor
    NCP> DEFINE EXECUTOR BUFFER SIZE 576
  - May override executor buffer size for a line

    NCP> DEFINE LINE UNA-0 LINE BUFFER SIZE 1498
- **Make the executor buffer size the same for all routing nodes in the network**
- Transmit buffers
  - Transmit - originating
  - Transit - route through
  - NCP> DEFINE EXECUTOR SEGMENT BUFFER SIZE 576
- **1 <= SEGMENT BUFFER SIZE <= EXEC BUFFER SIZE <= 65535**

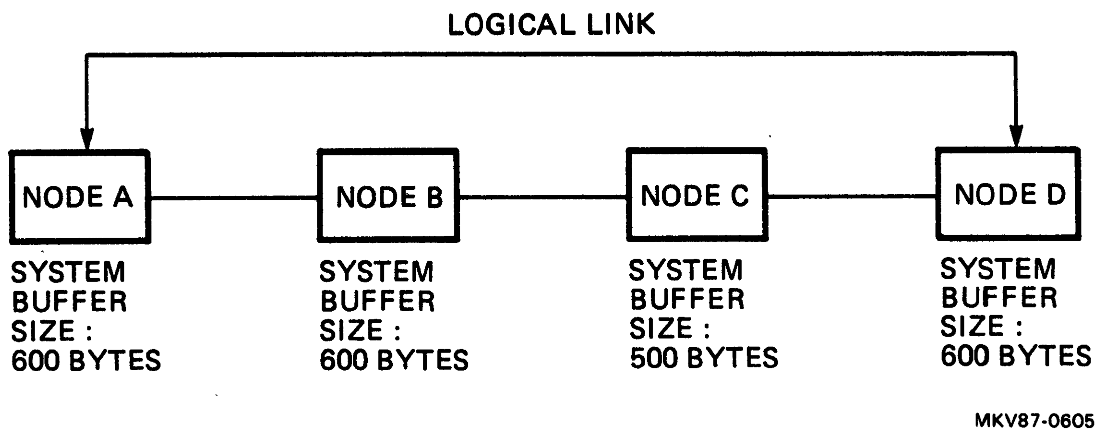


Figure 8-1 Routing Problem with Varying Buffer Sizes

1.1 Meaning for Maximum Buffers

- In Phase III the parameter EXECUTOR MAXIMUM BUFFERS caused the pre-allocation of those buffers. This resulted in waste if the number was too large or congestion loss if the number was too small.

- Now, the transmit buffer pool floats to its optimal level based on usage, up to a limit of MAXIMUM BUFFERS.

```
NCP> SET EXECUTOR MAXIMUM BUFFERS n

NCP> SHOW EXECUTOR CHARACTERISITCS
          . . . .

     Maximum buffers           = 15
     Buffer size               = 576

          . . . .
```

- Each second, the pool is reduced by one buffer, as long as a minimum of 10 buffers is present.

- When all buffers in the pool are in use, and a new buffer is needed, one is allocated.

- There is always one buffer guaranteed to be available for each circuit.

- The new default value of MAXIMUM BUFFERS is 100, which simply limits the total pool that can be allocated by DECnet. There is no need to specify this parameter in normal cases.

- This new mechanism greatly reduces nonpaged pool consumption while the network is running.

2 SYSTEM AND USER BUFFERING LEVELS

Two types of buffers are considered: system and user.

System buffers are used by the operating system and communication software to interface to the communication device(s).

User buffers are allocated by the application program to hold data to interface to the operating system and communication software.

In general:

- Have the same system buffer size across the network, or at least on the routing nodes
- An average user buffer size should correspond to the system buffer size, considering the protocol's overhead in the system buffer
- The smaller the average message size, the lower the potential throughput
- The greater the variation in message size, the more degraded the performance

3 DECnet TRANSFER PROCESS

1. User issues a $QIO or $QIOW to transmit data
2. Packets are buffered by the system in nonpaged pool before the QIO completes
3. $QIO completes as soon as the packet is delivered to the local DECnet
4. The IOSB gets filled in, event flags set, and ASTs queued as soon as the packet is delivered to the remote DECnet

NOTES

1. By design, everything that happens at the network level is supposed to be transparent to the user. This process can and does change without warning.
2. The amount of buffering that can happen within any logical link is related to the pipeline quota (see Section 3.1).

4 AFFECTING NETACP

NETACP is created and run in SYS$MANAGER:LOADNET.COM. It is designed for normal network activity. For large networks, you may want to give NETACP a larger working set and page file quotas.

Define the following logical names :

NETACP$EXTENT	:	WSEXTENT
NETACP$MAXIMUM_WORKING_SET	:	WSQUOTA
NETACP$PAGE_FILE	:	PAGEFILE QUOTA

```
$ IF P1 .EQS. "" THEN P1 = "SYS$SYSTEM:NETACP"
$ !
$ !    Check for user-supplied defaults.
$ !
$ MAX_WORK = F$LOGICAL("NETACP$MAXIMUM_WORKING_SET")
$ IF MAX_WORK .EQS. "" THEN MAX_WORK = "350"
$ PAGE_FILE = F$LOGICAL("NETACP$PAGE_FILE")
$ IF PAGE_FILE .EQS. "" THEN PAGE_FILE = "8192"
$ EXTENT = F$LOGICAL("NETACP$EXTENT")
$ IF EXTENT .EQS. "" THEN EXTENT = "1500"
$ RUN 'P1' -
       /NOACCOUNTING-
       /NOAUTHORIZE-
       /AST_LIMIT=100-
       /BUFFER_LIMIT=65535-
       /EXTENT='EXTENT'-
       /FILE_LIMIT=10-
       /IO_BUFFERED=32767-
       /IO_DIRECT=32767-
       /QUEUE_LIMIT=16-
       /MAXIMUM_WORKING_SET='MAX_WORK'-
       /PAGE_FILE='PAGE_FILE'-
       /PRIORITY=8-
       /PRIVILEGES=CMKRNL-
       /PROCESS_NAME=NETACP-
       /UIC=[1,3]
```

Example 8-1 Startup of NETACP from LOADNET.COM

5 ROUTING PARAMETERS

- Maximum Hops
  - 1 <= MAX HOPS <= 30
  - MAX HOPS <= MAX VISITS
- Maximum Cost
  - 1 <= MAXIMUM COST <= 1022
- Maximum Visits
  - 1 <= MAX VISITS <= 63
  - MAX HOPS <= MAX VISITS <= 2 TO 3 * MAX HOPS
- Maximum Address
  - 1 <= MAX ADDRESS<= 1023

5.1 Area Routing Parameters

- Area maximum cost
- Area maximum hops
- Maximum area

5.2 Ethernet Routing Parameters

- Maximum routers
- Maximum broadcast nonrouters
- Maximum broadcast routers
- Router priority

  (0 to 127 default = 64)

6 LIMITING LOGICAL LINKS

6.1 Maximum Links

- Specifies maximum logical links count for the local node
- Includes inbound and outbound connections
- Limits DECnet activity where this node is the source or destination

6.2 Alias Maximum Links

- Specifies maximum logical links count for the local node that can use the alias node identifier
- Used to balance the load for inbound alias connects (multiples of 16)

7 SYSGEN PARAMETERS RELATING TO NONPAGED POOL

- NPAGEDYN/NPAGEVIR
- SRPSIZE
- SRPCOUNT/SRPCOUNTV
- IRPCOUNT/IRPCOUNTV
- LRPSIZE
- LRPCOUNT/LRPCOUNTV

8 PERFORMANCE OF RMS FAL

- DAP buffer size increased to 4156 from 544 in VMS V4
- Maximum record size is 4156
- Very significant performance improvement over previous versions, especially for applications that do a lot of file GETS and PUTS
- VMS SYSGEN parameter **RMS_DFNBC**

  FAL sends **RMS_DFNBC** blocks at a time

  Default network transfer block count - determines the default message size in disk blocks to be used when performing DECnet file transfer operations

SYSGEN> SHOW RMS_DFNBC

Parameter Name	Current	Default	Minimum	Maximum	Unit	Dynamic
RMS_DFNBC	8	8	1	127	Blocks	D

Adjust this to your DECnet file applications

Still broken down to ECL segments

Large file transfer
Less disk I/O
but more page faults

EXERCISES

PROTOCOL LAB EXERCISES

1. Examine and interpret the following protocol message that came over the Ethernet.

   SUPER is node 2.230 and HARDY is node 2.149

```
PACKET COME IN FROM RIGHT-TO-LEFT TOP-TO-BOTTOM
================================================

001C0360 08950004 00AA08E6 000400AA

95000400 AA000008 E6000400 AA000026

00000000 82D10817 64160400 00000008

  44B89B C0000000 00000000 00000000
```

```
PARTIAL BREAKDOWN
=================

0360 0895000400AA 08E6000400AA

0400AA00 0008E600 0400AA00 0026001C
000082D1 08176416 04000000 00089500
      00 00000000 00000000 00000000

44B89BC0
```

PROTOCOL LAB EXERCISES

2. Interpret the following message received on a DDCMP circuit.

(READ FROM LEFT TO RIGHT)

A. RECEIVED MESSAGE:

0506C000 00017597

B. RECEIVED MESSAGE:

810DC005 0601DDB1

020204050401

04 010B 0108

0180 86D3

C. RECEIVED MESSAGE:

8122C006 0801FC17

020204050401

60 010B 0108 0100

'HELLO, THIS IS A TEST' (ASCII CODE)

0612

PROTOCOL LAB EXERCISES

3. Protocol is:

   a. The DNA interface between adjacent layers at the same node

   b. A special message sent over the network to control data transmission

   c. The DNA interface between the same layers in different nodes

4. What type of delivery service does the Ethernet Data Link offer to its clients?

   a. Positive acknowlegement with automatic retransmissions

   b. Worst-Case control with automatic retransmissions

   c. Best-Effort with indefinite automatic retransmissions if collisions are detected

   d. Best-Effort with no more than 16 automatic retransmissions if collisions are detected

5. Etherent data messages can send between ______ and ______ bytes of client data.

6. For the following Segmented Routing Message:

   (Bytes come in right-to-left)

   ..... 072A100005000100FF7FFF7FFF7F04050000F57F

   a. How many nodes are reported by the message?

   b. What is the transmitting node's address?

   c. What is the address of the first node reported in this message?

PROTOCOL LAB SOLUTIONS

1. Ethernet Data

Work out what the node addresses would look like for SUPER and HARDY.

Area	Node No.
6 bits	10 bits

230 = E6 (hex) and 149 = 95 (hex)

Thus SUPER would be 08E6 and HARDY would be 0895.

Break up the Ethernet Protocol

Ethernet Header

Source Addr:	0895000400AA	Dest. Addr:	08E6000400AA
Protocol Type:	0360 DNA	Frame Chk Seq:	44B89BC0

Data

```
0400AA00 0008E600 0400AA00 0026001C
000082D1 08175416 04000000 00089500
    0000 00000000 00000000 00000000
```

The interpretation should be as follows:

Dest. Host: 08E6000400AA

In Ethernet notation, this would be AA-00-04-00-E6-08. From the handout, the number AA-00-04-00 is always added to the 16-bit node address. The 16-bit node addresses should be interpreted as 08E6 (SUPER).

PROTOCOL LAB SOLUTIONS

Source Host: 0895000400AA

From similar reasoning, the source host would be 0895 or node HARDY.

Protocol Type: 0360 DNA

Frame Chk Seq: 44B89BC0 (This is a 32-bit CRC check)

PROTOCOL LAB SOLUTIONS

Data

```
0400AA00 0008E600 0400AA00 0026001C
000082D1 08176416 04000000 00089500
    0000 00000000 00000000 00000000
```

001C	This is the message length field (28 bytes)	
26	Bits 0-1	Message type 2 - Long Format
	Bit 2	Direct from end node
	Bit 5	Intra-NI packet
0000		Reserved
AA000400E608		Destination ID
0000		Reserved
AA0004009508		Source ID
00		Reserved
00		Visit count = 0
0000		Reserved

(There are 24 bytes of transport header on Ethernet.)

04	NSP data ack
6416	Dest. logical address
0817	Source logical address
82D1	Acknum

(Note that there are a total of 28 bytes in the message)

0000....	Padding with 0 up to 46 bytes

PROTOCOL LAB SOLUTIONS

2. DDCMP Exercises

A. DDCMP Initialization

The DDCMP initialization sequence is this message. The following is a breakdown of the various fields (refer to your DDCMP specification for details).

FIELD NAME	HEX CONTENT	INTERPRETATION
ENQ	05	DDCMP CONTROL MESSAGE
TYPE	06	DDCMP START MESSAGE (1 - ACK 2 - NAK 3 - REP 6 - STRT 7 - STACK)
FLAG	CO	SELECT = 1 QSYNC = 1
RESPONSE NO.	00	NOT USED IN CONTROL MESSAGES
TRANSMIT NO.	00	NOT USED IN CONTROL MESSAGES
STATION NO.	01	ALWAYS 01 ON PT-TO-PT LINKS
BLOCK CHECK	7595	CYCLIC REDUNDANCY CHECK

B. NSP Data Acknowledgement Message

This message is sent in response to the NSP connect ACK and NSP connect confirm messages. It has a routing header.

FIELD NAME	HEX CONTENT	INTERPRETATION
DDCMP HEADER	810DC0050601DDB1	

PROTOCOL LAB SOLUTIONS

ROUTE HEADER	020204050401	02 = ROUTING DATA MESSAGE 0402 = SRC NODE ADDRESS 1.2 0504 = DST NODE ADDRESS 1.5 01 = VISIT COUNT FIELD
MSGFLG	04	NSP DATA ACKNOWLEDGEMENT
DSTADDR	010B	DESTINATION LOGICAL LINK ADDRESS
SRCADDR	0108	SOURCE LOGICAL LINK ADDRESS
ACKNUM	0180	THE FORMAT FOR THIS WOULD BE: 15 14 12 11 0 1 QUAL NUMBER THUS 8001 MEANS ACKNOWLEDGE NSP MESSAGE 1
BLOCK CHECK	86D3	

C. NSP Data Message

This is the message containing user data.

FIELD NAME	HEX CONTENT	INTERPRETATION
DDCMP HEADER	8122C0060801FC17	
ROUTE HEADER	020400050000	
MSGFLG	60	SINGLE SEGMENT MESSAGE BEGINNING OF SESSION CONTROL MESSAGE AND END OF SESSION CONTROL MESSAGE BOTH EQUAL 1
DSTADDR	010B	DESTINATION LOGICAL LINK ADDRESS
SRCADDR	0108	SOURCE LOGICAL LINK ADDRESS
SEGNUM	0100	SEGNUM = 1 - NO DELAYED ACK
DATA	HELLO, THIS IS A TEST	
BLOCK CHECK	0612	

PROTOCOL LAB SOLUTIONS

D. NSP Data Acknowledgment Message

This message acknowledges the receipt of a good message.

FIELD NAME	HEX CONTENT	INTERPRETATION
DDCMP HEADER	810DC00B0A01B972	
ROUTE HEADER	020500040000	
MSGFLG	04	NSP DATA ACKNOWLEDGMENT MESSAGE
DSTADDR	0108	DESTINATION LOGICAL LINK ADDRESS
SRCADDR	010B	SOURCE LOGICAL LINK ADRESS
ACKNUM	0180	ACKNOWLEDGE DATA MESSAGE 1
BLOCK CHECK	9F45	

PROTOCOL LAB SOLUTIONS

3. Protocol is:

   a. The DNA interface between adjacent layers at the same node

   b. A special message sent over the network to control data transmission

   c. The DNA interface between the same layers in different nodes.

4. What type of delivery service does the Ethernet Data Link offer to its clients?

   a. Positive acknowlegement with automatic retransmissions

   b. Worst-Case control with automatic retransmissions

   c. Best-Effort with indefinite automatic retransmissions if collisions are detected

   d. Best-Effort with no more than 16 automatic retransmissions if collisions are detected

5. Etherent data messages can send between __46__ and __1500__ bytes of client data.

6. For the following Segmented Routing Message:

   ..... 072A100005000100FF7FFF7FFF7F04050000F57F

   a. How many nodes are reported by the message?

   __5__

   b. What is the transmitting node's address?

   __102A__ = __1.6__

   c. What is the address of the first node reported in this message?

   __1.1__ (Note that the area bits are cleared)

SDA LAB EXERCISES

This analysis will be done on a crash dump taken on a VAX 750 with 3 MB of memory. The node was a Phase IV routing node, running VMS Version 4.4.

The following information is a list of DCL and NCP commands taken before the crash.

```
NCP> SHOW KNOWN NODES SUMMARY

Known Node Volatile Summary as of  3-SEP-1986 21:23:15

Executor node = 1.2 (THUD)

State                     = on
Identification            = VAX 750 - Standalone System

    Node             State        Active  Delay   Circuit  Next node
                                  Links

 1.1 (SPLASH)     reachable          4       1    UNA-0    1.1 (SPLASH)
 1.3 (ZIP)        reachable                       UNA-0    1.3 (ZIP)
 1.4 (BAROOM)     reachable                       UNA-0    1.4 (BAROOM)
 1.5 (CLICK)      reachable                       UNA-0    1.5 (CLICK)
 1.6 (DRIP)       unreachable
 1.10 (NOISES)    reachable          1       1    UNA-0    1.4 (BAROOM)
 1.13 (FIZZ)      unreachable
 1.14 (SNAP)      unreachable
 1.15 (POOF)      unreachable
```

```
$ SHOW NETWORK

VAX/VMS Network status for local node 1.2 THUD on 3-SEP-1986 21:22:12.98

       Node            Links  Cost  Hops   Next Hop to Node
    1.2     THUD         0      0     0     (Local)    ->  1.2     THUD
    1.1     SPLASH       3      1     1     UNA-0      ->  1.1     SPLASH
    1.3     ZIP          0      1     1     UNA-0      ->  1.3     ZIP
    1.4     BAROOM       0      1     1     UNA-0      ->  1.4     BAROOM
    1.5     CLICK        0      1     1     UNA-0      ->  1.5     CLICK
    1.10    NOISES       1      1     1     UNA-0      ->  1.4     BAROOM
                Total of 6 nodes.

$ SHOW SYSTEM

VAX/VMS V4.4  on node THUD  3-SEP-1986 21:22:19.97  Uptime 0 00:17:47
  Pid    Process Name    State  Pri    I/O        CPU     Page flts Ph.Mem
00000080 NULL            COM      0      0   0 00:13:12.79      0     0
00000081 SWAPPER         HIB     16      0   0 00:00:01.02      0     0
00000084 ERRFMT          HIB      8     28   0 00:00:00.53     70    89
00000085 OPCOM           LEF      8     70   0 00:00:01.45    184    46
00000086 JOB_CONTROL     HIB      8    532   0 00:00:06.83    181   302
00000089 SYMBIONT_0001   HIB      4     12   0 00:00:00.67    152    36
0000008E DBMS_MONITOR    LEF     10     17   0 00:00:00.97    216   314
00000090 NETACP          HIB     10    111   0 00:00:04.87    313   236
00000091 EVL             HIB      6     43   0 00:00:02.08    586    30   N
00000092 ACMS_SWL        HIB      9     12   0 00:00:00.56    137   200
00000093 REMACP          HIB      9     23   0 00:00:00.31    74     40
00000094 Scott           LEF      8    302   0 00:00:16.37    800   563
00000095 FAL_1026        LEF      6    144   0 00:00:05.64    883   150   N
00000096 SCOTT           LEF      4    373   0 00:00:08.27   1000   176
00000097 PHONE_1034      LEF      6     94   0 00:00:03.49    589   150   N
0000009B NET3            LEF      4     59   0 00:00:02.17    354   177
0000009C Commuter        CUR      4    141   0 00:00:15.62    903   280

$ SHOW USERS

          VAX/VMS Interactive Users        3-SEP-1986 21:22:26.63
     Total number of interactive users = 4
 Username       Process Name        PID      Terminal
 NET3           NET3              0000009B   RTA1:
 SCOTT          Scott             00000094   LTA1:
 SCOTT          SCOTT             00000096   LTA2:
 SCOTT          Commuter          0000009C   LTA3:
```

```
$ SHOW MEMORY/PHYSICAL/POOL/FULL

              System Memory Resources on  3-SEP-1986 21:22:41.25
Physical Memory Usage (pages):     Total     Free      In Use     Modified
  Main Memory (3.00Mb)              6144      928        4804          412

Small Packet (SRP) Lookaside List         Packets      Bytes        Pages
    Current Total Size                        746      71616          140
    Initial Size (SRPCOUNT)                   746      71616          140
    Maximum Size (SRPCOUNTV)                 2984     286464          560
    Free Space                                413      39648
    Space in Use                              333      31968
    Packet Size/Upper Bound (SRPSIZE)                     96
    Lower Bound on Allocation                             32
I/O Request Packet (IRP) Lookaside List   Packets      Bytes        Pages
    Current Total Size                        520     108160          212
    Initial Size (IRPCOUNT)                   520     108160          212
    Maximum Size (IRPCOUNTV)                 2080     432640          845
    Free Space                                294      61152
    Space in Use                              226      47008
    Packet Size/Upper Bound (fixed)                      208
    Lower Bound on Allocation                             97
Large Packet (LRP) Lookaside List         Packets      Bytes        Pages
    Current Total Size                         21      33264           65
    Initial Size (LRPCOUNT)                     6       9504           19
    Maximum Size (LRPCOUNTV)                   60      95040          186
    Free Space                                  9      14256
    Space in Use                               12      19008
    Packet Size/Upper Bound (LRPSIZE + 80)              1584
    Lower Bound on Allocation                           1088
Nonpaged Dynamic Memory
    Current Size (bytes)      287744   Current Total Size (pages)      562
    Initial Size (NPAGEDYN)   287744   Initial Size (pages)            562
    Maximum Size (NPAGEVIR)   863744   Maximum Size (pages)           1687
    Free Space (bytes)        114176   Space in Use (bytes)         173568
    Size of Largest Block     107072   Size of Smallest Block           16
    Number of Free Blocks         19   Free Blocks LEQU 32 Bytes         3
Paged Dynamic Memory
    Current Size (PAGEDYN)    253952   Current Total Size (pages)      496
    Free Space (bytes)         28624   Space in Use (bytes)         225328
    Size of Largest Block      26912   Size of Smallest Block           16
    Number of Free Blocks         23   Free Blocks LEQU 32 Bytes        17
```

```
NCP> SHOW EXEC CHARACTERISTICS

Node Volatile Characteristics as of  3-SEP-1986 21:22:56

Executor node = 1.2 (THUD)

Identification             = VAX 750 - Standalone System
Management version         = V4.0.0
Incoming timer             = 45
Outgoing timer             = 45
NSP version                = V4.0.0
Maximum links              = 32
Delay factor               = 80
Delay weight               = 5
Inactivity timer           = 60
Retransmit factor          = 10
Routing version            = V2.0.0
Type                       = routing IV
Routing timer              = 600
Broadcast routing timer    = 180
Maximum address            = 15
Maximum circuits           = 10
Maximum cost               = 10
Maximum hops               = 5
Maximum visits             = 10
Maximum area               = 63
Max broadcast nonrouters   = 64
Max broadcast routers      = 32
Area maximum cost          = 1022
Area maximum hops          = 30
Maximum buffers            = 15
Buffer size                = 576
Nonprivileged user id      = DECNET
Default access             = incoming and outgoing
Pipeline quota             = 5000
Default proxy access       = incoming and outgoing
Alias maximum links        = 32
```

```
NCP> SHOW ACTIVE CIRCUITS CHARACTERISTICS

Active Circuit Volatile Characteristics as of  3-SEP-1986 21:23:25

Circuit = DMC-0

State                    = on
Service                  = enabled
Cost                     = 2
Hello timer              = 15
Verification             = disabled
Adjacent node            =  1.1 (SPLASH)
Listen timer             = 30


Circuit = UNA-0

State                    = on
Service                  = enabled
Designated router        =  1.1 (SPLASH)
Cost                     = 1
Router priority          = 64
Hello timer              = 15
Type                     = Ethernet
Adjacent node            =  1.1 (SPLASH)
Listen timer             = 45


Circuit = UNA-0

Adjacent node            =  1.4 (BAROOM)
Listen timer             = 45


Circuit = UNA-0

Adjacent node            =  1.3 (ZIP)
Listen timer             = 45


Circuit = UNA-0

Adjacent node            =  1.5 (CLICK)
Listen timer             = 45
```

```
NCP> SHOW KNOWN LINKS STATUS

Known Link Volatile Summary as of  3-SEP-1986 21:23:37

   Link        Node         PID       Process Remote link  Remote user  State

  1026    1.1 (SPLASH)  00000095  FAL_1026        3094  NET9           run
  1033    1.1 (SPLASH)  00000094  Scott           4128  PHONE          run
  1038    1.1 (SPLASH)  00000093  REMACP          4136  SCOTT          run
  1039    1.1 (SPLASH)  00000093  REMACP          3114  SUE            run
  1034    1.10 (NOISES) 00000097  PHONE_1034      2817  VICKI          run


NCP> TELL SPLASH SHOW KNOW LINKS STATUS

Known Link Volatile Summary as of  3-SEP-1986 21:23:46

   Link        Node       PID         Process Remote link  Remote user  State

  3094    1.2 (THUD)  20600288  NET9              1026  FAL            run
  4128    1.2 (THUD)  2060030D  PHONE_4128        1033  SCOTT          run
  4136    1.2 (THUD)  2060030F  SCOTT             1038  CTERM          run
  3114    1.2 (THUD)  2060027F  sur la plage...1039  CTERM          run
  4139    1.2 (THUD)  2060028A  NML_4139          1040  SCOTT          run
  2817    1.2 (THUD)  2060030B  VICKI             1034  PHONE          run
```

```
NCP> SHOW KNOW NODES COUNTERS

Known Node Counters as of  3-SEP-1986 21:24:21

Executor node = 1.2 (THUD)

          6  Maximum logical links active
          0  Aged packet loss
          0  Node unreachable packet loss
          0  Node out-of-range packet loss
          0  Oversized packet loss
          0  Packet format error
          0  Partial routing update loss
          0  Verification reject


Remote node =   1.1 (SPLASH)

        725  Seconds since last zeroed
       8709  Bytes received
      15719  Bytes sent
        287  Messages received
        310  Messages sent
          6  Connects received
         10  Connects sent
          7  Response timeouts
          0  Received connect resource errors


Remote node =   1.3 (ZIP)

No information available


Remote node =   1.4 (BAROOM)

No information available


Remote node =   1.5 (CLICK)

No information available


Remote node =   1.6 (DRIP)

No information available
```

```
Remote node =   1.10 (NOISES)

        437  Seconds since last zeroed
        139  Bytes received
          1  Bytes sent
         32  Messages received
         28  Messages sent
          1  Connects received
          0  Connects sent
          0  Response timeouts
          0  Received connect resource errors


Remote node =   1.13 (FIZZ)

No information available


Remote node =   1.14 (SNAP)

No information available


Remote node =   1.15 (POOF)

No information available
```

SDA LAB SOLUTIONS

1. Invoke SDA with the file DECNETINT$SOLN:SDALAB.DUMP

```
$ ANALYZE/CRASH_DUMP DECNETINT$SOLN:SDALAB.DUMP
```

NOTE

Ignore the message regarding the shortage of physical memory pages contained in the dump file. It is less than that on the current system, but is all the pages from the crashed system.

2. Read in the system and network definition files.

```
SDA>  READ SYS$SYSTEM:SYSDEF
SDA>  READ SYS$SYSTEM:NETDEF
```

3. Set the output to go to a file and issue the following commands:

```
SDA> SET LOG SYS$LOGIN:SDA.LOG
```

   a. Issue the command to look at the processes around at the time the crash was taken and the image they were executing:

   ```
   SDA> SHOW SUMMARY/IMAGE
   ```

   b. Issue the following command to look at pool:

   ```
   SDA> SHOW POOL/SUMMARY
   ```

   c. Isue the command to look at all NETn, MBAn, and RTAn devices:

   ```
   SDA> SHOW DEVICE NET
   SDA> SHOW DEVICE MB
   SDA> SHOW DEVICE RT
   ```

   d. Issue the command to look at network hardware devices:

   ```
   SDA> SHOW DEVICE XE
   SDA> SHOW DEVICE XM
   ```

SDA LAB SOLUTIONS

e. Set the output back to the terminal and print the file you created:

```
SDA> SET NOLOG
SDA> SPAWN/NOWAIT PRINT/NOTIFY/NOFEED SYS$LOGIN:SDA.LOG
```

4. Examine the output

   a. Track down what process is associated with what NETn device.

      Use the process ID and the UIC. To look at a process in more detail, issue SDA> SHOW PROCESS/INDEX=nn

      (Where nn is the index number under the SHOW SUMMARY command)

   b. Find a connection between a NETn device and a mailbox.

      Match a MBA address from the NETn device with the UCB address of the MBAn device.

   c. Find the address of the RCB (VCB of each NETn device).

5. Continue the data structures examination in SDA. You may send the output to a file or examine these structures on line.

6. FORMAT a UCB of one of the NETn devices.

7. FORMAT and get a printed copy of the RCB.

8. Look at the ADJ (RCB$L_PTR_ADJ) and get the index for several nodes. (Each index is a longword, so step through the table by 4 bytes.)

SDA LAB SOLUTIONS

9. FORMAT the ADJ of a node and check the PNA (Physical Node Address) against the list of known nodes as displayed above.

10. Examine the LPD table index (RCB$L_PTR_LPD) and look at an entry from ADJ$B_LPD_INX for a circuit used to a node (Most of these will be UNA_0.)

    Note the COST, ROUTER PRIORITY, and BUFFER SIZE among the values in the LPD.

11. Examine the Link Table (RCB$L_PTR_LTB) and look at one logical link.

12. FORMAT this LTB and get the pointer to the XWB for that link.

13. FORMAT and examine the XWB.

SDA LAB SOLUTIONS

14. [OPTIONAL] Using ANALYZE/SYSTEM

a. Establish a logical link with a node in the network.

```
$ COPY/LOG file.dat node::
$ OPEN REMFILE node::file.dat
```

NOTE

You may want to establish the link with your own node and examine both ends of the link.

You will be the source process talking to a FAL process (on node or the local node).

b. Issue the following NCP command and print out the file.

```
$ MCR NCP SHOW KNOWN LINKS STATUS TO LINKS.DAT
$ MCR NCP TELL node SHOW KNOW LINKS STASTUS TO LINKS.DAT
```

c. Examine the current system with the command ANALYZE/SYSTEM.

d. Repeat enough of the above exercises so that you can trace down the structures related to your logical link. Information from the NCP display will match what you find in XWB$REMLINK, XWB$_LOCLNK, XWB$_REMNOD, XWB$_PID, and other fields.

APPENDIX

LISTING OF DECnet-VAX MODULES

To look up on Microfiche

1 MODULES OF NETACP

1.1 NETTRN - Major NETACP Work Dispatching Loop

WQE$RESET_TIM - Cancel and reset timer
WQE$CANCEL_TIM - Cancel work timer
WQE$TIMER_AST - Work timer AST
WQE$INSQUE - Insert WQE into work queue
WQE$REMQUE - Dispatch next work element
WQE$ALLOCATE - Allocate a work element
WQE$DEALLOCATE - Deallocate a work element
WQE$FORK - Switch to work queue level
NET$GETUTLBUF - Get use of utility buffer
NET$BIN2ASC - Convert binary to ASCII
NET$JNX_CO - Journalling routine
Pool allocation routines

1.2 NETCONECT - Routines to Process User Connect Requests

NET$CONNECT - IO$_ACCESS $QIO Procesing
PRS_NCB - Parse Network Connect Block
PRS_NODE - Parse NCB nodename
PRS_ACCESS - Parse NCB access control fields
PRS_OBJECT - Parse NCB target task identifier
PRS_END - Parse the remainder of the NCB
DFLT_ACCESS - Get default access control
GET_STR_NUM - Get next numeric token
GET_TOKEN - Get next token

1.3 NETPROCRE - Process Creation Routines

NET$PROC_XWB	- Process returned XWB
NET$CREATE_MBX	- Create ACP mailbox
NET$KILL_MBX	- Delete ACP mailbox
NET$MBX_QIO	- Issue mailbox read
NET$SET_MBX_AST	- Process mailbox AST
NET$CONNECT_FAIL	- Notify NETDRIVER of failed link
NET$SERVER_FAIL	- Notify NETDRIVER of terminated server
NET$SCAN_FOR_ZNA	- Send pending connects to declared object
NET$RESEND_SERVER	- Re-send initial connect to server
NET$STARTUP_OBJ	- Startup privileged process
NET$STARTUP_OBJ_NAM	- Startup process by name
NET$DELIVER_CI	- Process and Deliver Inbound Connect
BUILD_NCB	- Build NCB for incoming connect
GET_PROC	- Locate process to accept connect
SEND_TO_SERVER	- Send connect to waiting server
CREATE_SPI	- Create SPI database entry
GET_PR_NAM	- Get name of object procedure
GET_PR_ZNA	- Construct ZNA string for an object
TELL_DRV	- Call NETDRIVER
UP_CASE	- Upcase the LOGINOUT strings

1.4 NETACPTRN - Control Network Local Node State Transition Routines

MOUNT	- Mount the NET device
INIT_NETDRIVER	- Tell NETDRIVER to initialize
CALL_NETDRIVER	- Call NETDRIVER entry point
NET$DEC_TRANS	- Decrement transaction count
DISMOUNT	- Dismount the NET device
START_TIMER	- Start up ACP activity timer
PROC_EVT	- Process ACP event
MBX_NET_SHUT	- Broadcast shutdown message
NET$DECR_MCOUNT	- Decrement ACP mount count
NET$UPD_LOCAL	- Update local state
NET$DECLARE_PSI	- Declare ourselves as a PSI process
UNDECLARE_PSI	- Remove PSI declaration
UPDATE_DATABASE	- Update non-paged control blocks
BUILD_LTB	- Build logical link table
BUILD_LPD	- Build the LPD vector
BUILD_ADJ	- Build the ADJ vector
BUILD_NDC	- Build the node counter vector
BUILD_OA	- Build output adjacency vector
BUILD_AOA	- Build area output adjacency vector
COM_BLD_CO	- Common build vector coroutine

1.5 NETCNF - Configuration Database Access Routines

CNF$PRE_SHOW	- Pre-SHOW processing
CNF$PRE_QIO	- Pre-QIO processing
CNF$DELETE	- Delete a CNF entry
CNF$PURGE	- Drain CNF entries marked for delete
CNF$INSERT	- Insert/replace a CNF entry
CNF$COPY	- Copy a CNF to another
CNF$CLONE	- Compress a CNF entry
CNF$INIT	- Initialize CNF entry
CNF$KEY_SEARCH	- Search for selected CNFs
CNF$SEARCH	- Search for CNFs by list of keys
COMPARE	- Compare CNF against keys
CNF$GET_FIELD	- Get field from CNF entry
CNF$PUT_FIELD	- Store field into CNF entry
CNF$CLR_FIELD	- Clear a CNF field
CNF$VERIFY	- Check if field exists
GET_RT_FIELD	- Call action routine to get value
PUT_RT_FIELD	- Call action routine to store value
GET_DSC	- Get descriptor of CNF field

1.6 NETCLUSTR - Cluster Node Name Routines

NET$START_ALIAS	- Initialize alias participation
MRL_BLKAST	- Blocking AST for master registration lock
READ_MRL_VALBLK	- Put registration data in MRL value block
PLR_AST	- Blocking AST on participant's lock registration
PDL_AST	- Completion AST on participant departure lock
NET$STOP_ALIAS	- Disable this node's participation in alias
NET$ALIAS_XONOF	- Perform flow control for alias
RELEASE_LOCKS	- Release all locks to coordinate participation
ADJUST_NUMRTR	- Count the number of participating routers

1.7 NETCNFACT - Configuration Database Access Action Routines

```
NET$SCAN_xxx       - Default database scanner
NDIDEF_SCAN        - Default NDI database scanner
NET$SCAN_NDI       - Scan NDI database
NET$SCAN_AJI       - Scan AJI database
NET$SCAN_SDI       - Scan SDI database
NET$SCAN_ARI       - Scan ARI database
NET$SPCSCAN_XXX    - Special database scan routines
NET$SPCSCAN_NDI    - Special scan of NDI database
NET$PRE_QIO_xxx    - Pre-QIO processing
NET$SHOW_xxx       - Pre-SHOW processing
NET$DEFAULT_xxx    - Apply default values
NET$DEFAULT_NDI    - Apply default values to NDI CNF
NET$INSERT_LNI     - Pre-insertion processing
NDI_MARKER         - Insert executor NDI marker
NET$INSERT_NDI     - Pre-insertion processing
NET$INSERT_OBI     - Pre-insertion processing
NET$INSERT_xxx     - Pre-insertion processing
CHK_LOGIN_xxx      - Check login string length
NET$SPCINS_XXX     - Special database insertion routines
NET$SPCINS_DEF     - Default database insertion routine
NET$SPCINS_NDI     - Insert NDI database into binary tree
NET$DELETE_xxx     - Pre-delete processing
NET$REMOVE_xxx     - Process the remove request
NET$REMOVE_DEF     - Default processing of the remove request
SCAN_XWB           - Scan XWB list
SUPPRESS_AREA      - Suppress area from node address
NET$TEST_REACH     - Test node reachability
NET$AREA_REACH     - Test area reachability
NET$GET_LOC_STA    - Get executor state
NET$NDI_BY_ADD     - Find NDI CNF by node address
NET$LOCATE_NDI     - Find phantom or real NDI CNF
FMT_CNT            - FORMAT COUNTERS
LOG_COUNTERS       - LOG ZERO COUNTER EVENT
LNI PARAMETER ACTION ROUTINES
NDI PARAMETER ACTION ROUTINES
OBI PARAMETER ACTION ROUTINES
ESI PARAMETER ACTION ROUTINES
EFI PARAMETER ACTION ROUTINES
LLI PARAMETER ACTION ROUTINES
SPI PARAMETER ACTION ROUTINES
AJI PARAMETER ACTION ROUTINES
SDI PARAMETER ACTION ROUTINES
ARI PARAMETER ACTION ROUTINES
MOVE PARAMETER SUBROUTINES
```

1.8 NETCNFDLL - Datalink Database Action Routines

```
NET$SCAN_xxx       - Scan database
NET$PRE_QIO_xxx    - Pre-QIO processing
NET$SHOW_xxx       - Pre-SHOW processing
JAM_CNF            - Store driver values into CNF
GET_PSI_xxx        - Get current PSI parameter values
CLEAR_VOLATILE     - Clear list of volatile parameters
NET$DEFAULT_CRI    - Apply default values
NET$DEFAULT_PLI    - Apply default values
NET$INSERT_CRI     - Pre-insertion processing
CRI_TO_PSI         - Send CRI parameters to PSIACP
NET$INSERT_PLI     - Pre-insertion processing
ALLOC_PLVEC        - Set up PLVEC entry for new line
NET$SET_QIOW       - Issue datalink SETMODE function
PLI_TO_PSI         - Send PLI parameters to PSIACP
SEND_TO_PSI        - Send control QIO to PSIACP
NET$DELETE_CRI     - Pre-delete processing
NET$DELETE_PLI     - Pre-delete processing
NET$REMOVE_xxx     - Pre-remove processing
CRI parameter action routines
PLI parameter action routines
BUILD_DEVBUF       - Build DLLQIO buffer
NET$CVT_NMA_INT    - Convert NMA to NFB code
TRAN_DEVNAM        - Translate device name
PRS_MNEMONIC       - Parse device mnemonic
PRS_DECIMAL        - Parse decimal number
DEV_CNT_QIO        - Get device counters
```

1.9 NETDLLTRN - Routing and Datalink Control Layer Routines

```
NET$INIT_ROUTING     - Initialize routing database
NET$DLLUPDLNI        - Process modified LNI parameters
FORCE_FULL_DECISION  - Force full decision algorithm
NET$DLL_ALL_OFF      - Turn off all circuits
NET$DLL_OPR_SET      - Process operator-generated event
ALLOC_LPD            - Allocate LPD
ALLOC_COSTHOPS       - Allocate a cost/hops buffer
DEAL_LPD             - Deallocate LPD
CHECK_REQ_PARAMS     - Check that required parameters are
                       set
NET$DLL_X25_CALL     - Process incoming X.25 call
NET$DLL_X25_RESET    - X.25 reset detected
NET$DLL_RCV          - Process message received from driver
Received message pre-processing routines
    RCV_STR2         - Received Phase II start message
    RCV_STR3         - Received Phase III start message
    RCV_STR4         - Received Phase IV start message
    RCV_VRF          - Received routing verification message
    RCV_RHEL         - Received Phase IV Router Hello
                       message
    RCV_EHEL         - Received Phase IV Endnode Hello
                       message
    RCV_RT3          - Received Phase III routing message
    RCV_RT4          - Received Phase IV routing message
    RCV_ART          - Area Routing message received

Check for routing update loss
Parse phase II/III/IV address
SET_DLL_EVT          - Schedule event transition
NET$DLL_PRC_WQE      - Process work queue element
PROC_EVT             - Process an event
FIND_WQE_CTX         - Find context for a new WQE
Simple transition routines
ACT_RCV_STR          - Received start message
ADAPT_TO_PARTNER     - Adapt to partner's node type
ACT_RCV_VRF          - Received verification message
ACT_RCV_RHEL         - Received Router Hello message
ACT_ELECT            - Resolve election after waiting
ACT_RCV_EHEL         - Received Endnode Hello message
ACT_RCV_RT           - Receive routing message
UPDATE_MATRIX        - Update the routing matrix
ACT_RCV_ART          - Receive area routing message
REQUEST_UPDATE       - Request update of routing database
UPDATE               - Update database and neighbors
DECISION             - Update forwarding database
FIND_PATH_TO_NODE    - Find least cost path to node
AREA_DECISION        - Update area forwarding database
FIND_PATH_TO_AREA    - Find least cost path to area
UPD_NEIGHBORS        - Schedule routing messages
TIMER_XRT            - Automatic routing update timer
```

```
Start automatic routing update timer
ENDNODE_DECISION         - Endnode decision algorithm
ACT_ENT_MOP              - Enter MOP state
ACT_DLL_UP               - Datalink has initialized
DLE-related state changes
ACT_RUN_DOWN             - Run down a circuit
ACT_SET_OPER             - Restart a "stalled" circuit
ACT_TST_DL               - Circuit acceptance algorithm
ACT_ENT_RUN              - Enter RUN state
ACT_BC_UP                - Broadcast datalink has initialized
BRA_UP                   - Set up new adjacency for BRA
LOWEST_PRIO_BRA          - Find lowest priority BRA
BEA_UP                   - Set up new adjacency for BEA

Error action routines for RUN state
EXIT_RUN_STATE           - Exit the RUN state
ADJ_DOWN                 - Mark adjacency as shutdown
BRA_DOWN                 - Mark BRA down
BUILD_RTR_LIST           - Rebuild NI router/state list
ELECT_ROUTER             - Elect designated router
ACT_QIO_SHUT             - Shut down the datalink
ACT_QIO_STRT             - Start the datalink
ACT_PVC_START            - Start an X.25 PVC in multiple
                           steps
ACT_X25_CALL             - Accept incoming X.25 call
CHK_CIRC_START           - Check if circuit can be started
TOGGLE_LINE              - Shut down and start up line
ACT_XMT                  - Transmit pending messages
XMT_DALLY                - Dally before sending start message
XMT_STR                  - Transmit start message
XMT_VRF                  - Transmit verification message
XMT_RT                   - Transmit a routing message
XMT_RT4                  - Transmit a Phase IV routing
                           message
XMT_ART                  - Transmit a Phase IV area routing
                           message
CHK_IO                   - Check for multiple transmits
NET$DLL_QIO_CO           - Common QIO routine
SET_IOTIM                - Set I/O timer
RESET_CHAN               - Cancel all device I/O
NET$GET_LPD_CRI          - Locate CNF given LPD index
NET$ADJ_LPD_CRI          - Locate CNF given ADJ index
NET$LOCATE_LPD           - Locate LPD given CNF
NET$FIND_LPD             - Find LPD given LPD index
NET$FIND_ADJ             - Find LPD and ADJ given ADJ index
NET$GET_PLVECLPD         - Find next active LPD
TEST_SWITCH_LINE         - See if this line is SWITCH ENABLE
SCHED_SWITCH_WQE         - Schedule KILL_SWITCHED_DATABASE
                           routine
KILL_SWITCHED_DATABASE - Dissolve switched circuit database
TELL_NETDRIVER           - Inform NETDRIVER of an event
```

1.10 NETCTLALL - Process ACP Control QIO Routines

```
DISPATCHING
Declare Name or Object
Declare server process available for new connect
Cancel I/O
CTL_DATABASE     - Process database QIOs
GET_P2_KEY       - Get next P2 value
PROCESS_CNF      - Process each CNF block
```

1.11 NETEVTLOG - Process Event Logging Needs Routines

```
Event timer action routine
Internal inbound raw event processing
Inbound raw event processing
STARTUP_EVL      - Start EVL process
Event logging database changes
Outbound raw event processing
NET$SET_CTR_TIMER - Reset automatic counter timer
```

1.12 NETGETLIN - Check for DECnet License Routines

```
Check if DECnet license installed
Routing software key
Endnode software key
MORE ROUTINES TO CHECK LICENSE
```

1.13 NETCONFIG - Local Configuration Database

1.14 NETOPCOM - Operator Communications Routines

```
NETWORK OPERATOR MESSAGE FORMATTING
```

1.15 NETTREE - Subroutines for Processing Binary Trees

1.16 NETDEFS - Various NETACP Symbol Definitions

1.17 NETDLE - NETACP DLE Processing Routines

DLE$DISPATCH	- Dispatch newly received DLE IRP
DLE$ACCESS	- Handle IO$_ACCESS function
DLE$LPD_STATUS	- Check completion of MOP transition
BC_ACCESS	- Handle DLE access to broadcast circuit
DLE$SETMODE	- Process IO$_SETMODE request
DLE$DEACCESS	- Process IO$_DEACCESS request
LEAVE_MOP_STATE	- Leave MOP state
DLE$CANCEL	- Process DLE cancel request
DLE$BC_UP	- Initialize DLE on broadcast circuit
DLE$BC_DOWN	- Clean up DLE on broadcast circuit
INIT_UNSOL_CHAN	- Initialize channel for unsolicited messages
ISSUE_NI_READ	- Issue read request to NI driver
RCV_DLE_MSG	- Receive unsolicited DLE message
DLE$MOP_REQUEST	- Partner has requested MOP mode
STARTUP_MOM	- Start MOM process
ATTACH_UNSOL_MSG	- Attach unsolicited message
DLE$PRC_EXIT	- Handle MOM process termination

1.18 NETLLICNT - Node and Logical Link Counter Support Routines

NET$INIT_NDCOU	- Initialize NDC queues
NET$ACQUIRE_NDCOU	- Acquire Node Counter block
NET$RELEASE_NDCOU	- Release claim on NDCOU block
NET$FLUSH_LLI_CNT	- Flush logical-link counters
NET$READ_LLI_CNT	- Read logical-link counters
NET$READ_NDI_CNT	- Read node counters
COPY_NDC	- Copy NDC counters
ADD_NDC	- Add NDC counters in NDC format
ADD_NDC_TEMP	- Add NDC counters and copy to temp area
SUB_NDC	- Subtract NDC counters in NDC format
LOG_NDCOU	- Log NDC counters
GET_HASH_ADDR	- Get the table entry address
NET$LOOKUP_NDCOU	- Find NDCOU in Hash Table

2 MODULES OF NETDRIVER

2.1 NETDRVSES - DECnet Session Control Module for NETDRIVER

NET$AZ_DR_TABLE	- Disconnect Reason Code Mapping
NET$FORK	- Fork the XWB to do new work
NET$END_EVENT	- Abort current event without changing state
NET$COMPLEX_EV	- Change state and process new event
NET$PRE_EMPT	- Process new event without changing state
NET$EVENT	- Event dispatcher
NET$SCH_MSG	- Schedule message transmission
ACT$NOP	- Null action routine
ACT$BUG	- BUG_CHECK action routine
ACT$LOG	- Log-event action routine
ACT$NOLINK	- Report "SS$_FILNOTACC"
ACT$SSABORT	- Abort QIO since link was disconnected
NET$STARTIO	- Start I/O operation
NET$FDT_SETMODE	- Process IO$_SETMODE request
NET$FDT_CONTROL	- IO$_ACPCONTROL FDT processing
NET$CONTROL	- IO$_ACPCONTROL "startio" processing
NET$FDT_ACCESS	- IO$_ACCESS FDT processing
NET$ACCESS	- IO$_ACCESS "startio" processing
ACT$INITIATE	- Connect Initiate action routine
ACT$CONFRIM	- Connect Confirm action routine
NET$CMPL_ACC	- Complete IO$_ACCESS, fill in window
ACT$ENT_RUN	- Enter RUN state action routine
NET$FDT_DEACCESS	- IO$_DEACCESS FDT processing
NET$DEACCESS	- IO$_DEACCESS "startio" processing
CLEANUP_ACCESS	- Clean up XWB for terminated IO$_ACCESS
NET$CANCEL	- Cancel I/O routine
NET$PURG_RUN	- Clean up XWB to exit RUN state
NET$ACP_COMM	- Entry for ACP communication
NET$SEND_CS_MBX	- Send counted string to mailbox
NET$SEND_MBX	- Co-routine to send mailbox message
NET$CREATE_XWB	- Create XWB for logical-link
XWB_LOCLNK	- Get XWB via local link number
NET$XWB_LOCLNK	- Get XWB via local link number
NET$RET_SLOT	- Return logical-link XWB slot if done
NET$QUE_XWB	- Queue XWB to NETACP's AQB
NET$DRAIN_FREE_CXB	- Drain CXB free queue
NET$ALONPGD_Z	- Allocate and zero from system pool
NET$ALONONPAGED	- Allocate from system pool
NET$DEALLOCATE	- Deallocate non-paged pool
NET$MOV_TO_XWB	- Move counted string to XWB$B_DATA
NET$MOV_CSTR	- Move counted string with count field
NET$MOV_USTR	- Move counted string without count field
NET$POST_IO	- Send IRP to COM$POST

2.2 NETDRVNSP - DECnet NSP Module for NETDRIVER

NET$SETUP_RUN	- Set up XWB for the RUN state
NET$ALTENTRY	- Driver alternate entry point
NET$FDT_RCV	- Process IO$_READxBLK requests
NET$FDT_XMT	- Process IO$_WRITExBLK requests
NET$UNSOL_INTR	- Receive from Transport layer
ACT$RTS_NLT	- Return to sender as "no-link-terminate"
ACT$RCV_CC	- Respond to a received Connect Confirm
ACT$RCV_CA	- Respond to Connect Acknowledge
ACT$RCV_CI	- Process received Connect Initiate message
PRS_CHR	- Get characteristics from Connect message
ACT$RCV_RTS	- Receive CI message - "returned to sender"
ACT$RCV_Dx	- Receive DI/DC message
ACT$ABORT	- Disconnect or abort a link
ACT$CANLNK	- Disconnect link due to user's $CANCEL
ACT$RCV_DTACK	- DATA ACK message processing
ACT$RCV_LIACK	- INT/LI ACK message processing
NET$PIG_ACK	- Common piggy-backed ACK processing
PROC_DTACK	- Process of DATA ACK
PROC_LIACK	- Process INT/LS ACK
NET$ACK_XMT_SEGS	- ACK Xmt Segs, Complete User Xmt IRPs
ACT$RCV_LI	- Receive INT/LS message
CHK_INT_AVL	- Conditionally set XWB$V_FLG_IAVL
SHRINK_XPW	- Shrink the DATA transmit-packet-window
NEW_DATA_FLOW	- React to flow control msg
CALC_HXS...	- Calc 'highest xmt seg sendable'
ACT$RCV_DATA	- Process received DATA message
CLONE_RCV_CXB	- Clone a copy of a received CXB
NSP$SOLICIT	- Solicit permission to transmit
BLD_DISPATCH	- Dispatch to build message
BLD_CD	- Build Connect/Disconnect messages
BLD_CI	- Build a CI msg from XWB contents
BLD_CA	- Build a CA msg from XWB contents
BLD_CC	- Build a CC msg from XWB contents
BLD_DI	- Build a DI msg from XWB contents
BLD_DC	- Build A DC msg from XWB contents
BLD_LIACK	- Build a INT/LS ACK message
BLD_DTACK	- Build a DATA ACK message
BLD_LI	- Build INT/LS message
BLD_DAT	- Build DATA message
GET_XMT_CXB	- Get xmt CXB while in FDT context
GET_XMT_BUF	- Get xmt buffer while in fork context
NET$IO_STATUS	- Receive xmit status from Transport layer

```
NET$CCS_IOSTAT      - Receive xmit status for Phase II CC
                      message
NET$TIMER           - Process NETDRIVER clock tick
TIMED_SEG_ACKED     - Timed segment has been ACKed
```

2.3 NETDRVXPT - NETDRIVER Transport (Routing) Layer Routines

```
TR$UPDATE           - Initiate receive sequence on data link
TR$KILL_LOC_LPD     - Attempt to shut down local LPD
TR$TIMER            - Process Transport layer clock tick
TR$SOLICIT          - Process ECL request to xmit into the
                      net
TR$DENY             - Deny solicitor permission to transmit
TR$GRANT            - Grant solicitor permission to transmit
TR$TEST_REACH       - Check if node is reachable
TR$GET_ADJ          - Get output ADJ and LPD
TR$RCV_DIO_DATA     - Rcv Direct I/O from datalink layer
TR$RCV_BIO_DATA     - Rcv Buffered I/O from datalink layer
RCV_DIO_BIO         - Common Receive IRP processing
DISP_RCV_MSG        - Dispatch received message
TR_RTHDR            - Process received msg's route header
TR_ECL              - Pass received packet to ECL
Packet Errors       - Process miscellaneous packet errors
TR_RTHRU            - Process packet for route-thru
FINISH_XMT_HDR      - Finish building HDR and transmit it
UPDATE_CACHE        - Update the BC cache table
TR$RTRN_XMT_RTH     - End-action routine for route-thru IRPs
TR$RTRN_XMT_ECL     - End-action routine for ECL IRPs
TR$RTRN_XMT_TLK     - End-action routine for TALKER IRPs
TR_RTRN_IRP         - Recycle IRP Xmit IRP pool
TR_LPD_DOWN         - Process "LPD down" event
TR$GIVE_TO_ACP      - ECL entry to queue a buffer to the ACP
TR$QUE_WQE_AQB      - Queue WQE to AQB
TR$QUE_IRP_AQB      - Queue "LPD down" IRP to AQB
TR$LOC_DLL_XMT      - "Local" datalink driver transmit
TR$LOC_DLL_RCV      - "Local" datalink driver receive
TR$ADJUST_IRP       - Adjust the number of IRPs in the pool
TR$ALLOC_IRP        - Allocate IRP
TR$ALLOCATE         - Allocate and initialize buffer
TR_FILL_JNX         - Conditionally fill journal record
```

2.4 NETDRVQRL - DECnet 'Quick Routing Layer' Module for NETDRIVER

```
QRL$SOLICIT         - Process ECL request to xmit into the
                      net
QRL$DENY            - Deny solicitor permission to transmit
QRL$GRANT           - Grant solicitor permission to transmit
QRL$SETUP_CHAN      - Set up channel to specified node
QRL$SETUP_RTHDR     - Build route-header
QRL$SETUP_X_IRP     - Allocate, init, queue IRP
```

3 NDDRIVER - DECnet DLE DRIVER MODULES

```
DRIVER PROLOGUE TABLE
DRIVER DISPATCH TABLE
FUNCTION DECISION TABLE
DLE$STARTIO         - Start I/O operation
DLE$FDT_ACCESS      - IO$_ACCESS FDT processing
DLE$ACCESS          - IO$_ACCESS "startio" processing
DLE$FDT_DEACCESS    - IO$_DEACCESS FDT processing
DLE$DEACCESS        - IO$_DEACCESS "startio" processing
DEALLOC_DWB         - Deallocate DWB
RESTORE_QUOTA       - Restore "access" quota
DLE$FDT_SETMODE     - Process IO$_SETMODE request
SETMODE_ACPBUF      - Build SETMODE ACP complex buffer
GET_STATUS          - Refresh DLE status flags for IOSB
DLE$FDT_CONTROL     - IO$_ACPCONTROL FDT processing
DLE$CONTROL         - IO$_ACPCONTROL "startio" processing
DLE$CANCEL          - Cancel I/O routine
CANCEL_ALL          - Cancel all outstanding I/O
DLE$LPD_DOWN        - The circuit has gone away
DLE$FDT_RCV         - FDT for IO$_READxBLK requests
DLE$FDT_XMT         - FDT for IO$_WRITExBLK requests
DLE$FDT_RW          - DLE FDT read/write processing
DLE$FDT_BYTQUO      - Get non-paged pool quota
DLE$XMT_MSG         - Send message over direct-accessed
                      circuit
DLE$XMT_DONE        - Transmit I/O post-processing
INIT_RCV_IRP        - Initialize datalink receive IRP
ISSUE_DLL_RCV       - Issue datalink receive request
DLE$RCV_MSG         - Receive a message from datalink
RCV_DONE            - Complete User Receive IRP
UNIT_INIT           - Unit initialization
DLE$ALONPGD_Z       - Allocate and zero from system pool
DLE$ALONONPAGED     - Allocate from system pool
```