ULTRIX

digital

Guide to the Data Link Interface

Order Number: AA–PBKZA–TE

ULTRIX

Guide to the Data Link Interface

Order Number: AA-PBKZA-TE
June 1990

Product Version: ULTRIX Version 4.0 or higher

This manual tells you how to use the Data Link Interface (DLI) to write applications at the data link layer. The manual explains programming procedures and provides DLI programming examples.

digital equipment corporation
maynard, massachusetts

Restricted Rights: Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause of DFARS 252.227-7013.

© Digital Equipment Corporation 1990
All rights reserved.

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

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

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

The following are trademarks of Digital Equipment Corporation:

digital	DECUS	ULTRIX Worksystem Software
	DECwindows	VAX
CDA	DTIF	VAXstation
DDIF	MASSBUS	VMS
DDIS	MicroVAX	VMS/ULTRIX Connection
DEC	Q-bus	VT
DECnet	ULTRIX	XUI
DECstation	ULTRIX Mail Connection	

UNIX is a registered trademark of AT&T in the USA and other countries.

Ethernet is a registered trademark of Xerox Corporation.

Contents

Preface vii

Chapter 1 Introduction to the Data Link Interface

1.1 What Is DLI? 1–1
1.1.1 DLI Services 1–1
1.1.2 Hardware Support 1–2

1.2 Using DLI to Access the Ethernet 1–2

1.3 Prerequisites for DLI Programming 1–2

1.4 Including Higher-Level Services 1–2

1.5 DLI Concepts 1–3

Chapter 2 Understanding the DLI Socket Address Data Structure

2.1 How the sockaddr_dl Structure Works 2–1
2.1.1 Specifying Values Within the Structure 2–1
2.1.2 Using Ethernet and 802 Substructures 2–2

2.2 Using the Ethernet Substructure 2–3
2.2.1 How Ethernet Frames Work 2–4
2.2.2 Defining Ethernet Substructure Values 2–4
2.2.2.1 Target Node Physical Address 2–4
2.2.2.2 Protocol Type 2–4
2.2.2.3 I/O Control Flag 2–4

2.3 Using the 8802-3 Substructure 2–5
2.3.1 Defining 802 Substructure Values 2–6
2.3.1.1 Target Node Physical Address 2–6
2.3.1.2 Service Class 2–6
2.3.2 Destination Service Access Point 2–7
2.3.2.1 Source Service Access Point 2–7
2.3.2.2 Control Field 2–8

Chapter 3 Writing DLI Programs

3.1 Supplying Data Link Services 3–1

3.2 Using ULTRIX System Calls 3–1

3.3 Creating a Socket 3–2

3.4 Setting Socket Options 3–3

3.5 Binding the Socket 3–4
3.5.1 Filling in the sockaddr_dl Structure 3–4
3.5.1.1 Specifying the Address Family 3–4
3.5.1.2 Specifying the I/O device ID 3–4
3.5.1.3 Specifying the Substructure Type 3–5
3.5.2 Calculating the Buffer Size 3–7

3.6 Transferring Data 3–7

3.7 Deactivating the Socket 3–7

Appendix A DLI Data Structures

A.1 DLI Socket Address Structure A–1

A.2 DLI Device Identification Structure A–1

A.3 DLI Ethernet Socket Address Structure A–2

A.4 DLI 8802-3 Socket Address Structure A–2

A.5 8802-3 Packet Header Format A–2

Appendix B DLI Programming Examples

B.1 Sample DLI Client Program Using Ethernet Format Packets B–2

B.2 Sample DLI Server Program Using Ethernet Format Packets B–6

B.3 Sample DLI Client Program Using 8802-3 Format Packets B–10

B.4 Sample DLI Server Program Using 8802-3 Format Packets B–14

B.5 Sample DLI Program Using getsockopt and setsockopt B–18

Index

Figures

2–1 The Ethernet Frame Format 2–3
2–2 The 8802-3 Frame Format 2–6

Tables

3–1 Calling Sequence for DLI Programs 3–2
3–2 Data Transfer System Calls Used with DLI 3–7

Preface

The *Guide to the Data Link Interface (DLI)* tells you how to use the Data Link Interface to write application programs that run at the data link layer.

Audience

This guide is intended for anyone responsible for application programming using the Data Link Interface. It assumes the following:

- You have superuser (root) privileges.
- You are familiar with an editor, such as **vi** or **ed**.
- You have a thorough knowledge of the C programming language, with experience writing system or network programs.
- You are familiar with ULTRIX socket calls.
- You are familiar with the 8802-2 and 8802-3 protocols.
- You are familiar with ULTRIX naming conventions.

Organization

This guide has three chapters and two appendixes:

Chapter 1: Introduction to the Data Link Interface
Describes the Data Link Interface (DLI) capabilities and prerequisites for using it.

Chapter 2: Understanding the DLI Socket Address Data Structure
Describes format and input arguments used in the DLI socket address (**sockaddr_dl**) structure, as well as the Ethernet and 8802-3 substructures.

Chapter 3: Writing DLI Programs
Gives procedures for writing DLI programs.

Appendix A: DLI Data Structures
Shows DLI data structures.

Appendix B: DLI Programming Examples
Shows DLI programming examples.

Related Documents

For additional information, see the following documents:

- *ULTRIX Version 4.0 Release Notes*

  This manual gives information and updates not included in the ULTRIX documentation set.

- *ULTRIX Reference Pages*

  These volumes provide ULTRIX system reference pages, which describe commands, system calls, subroutines, file formats, and special files.

- *Guide to Network Programming*

  This manual describes network programming concepts for programming in the ULTRIX environment.

To obtain a detailed description of the Digital Network Architecture, refer to the following document:

- *DECnet Digital Network Architecture (Phase IV), General Description*
- *DECnet Digital Network Architecture, CSMA/CD Data Link Functional Specification, v.1.0.1, Nov. 1985*

For more information about the IEEE 8802-3 frame format, refer to the following two standards:

- *Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*, ANSI/IEEE Std 8802-3–1985, ISO Draft International Standard 8802/3.
- *Logical Link Control*, ANSI/IEEE Std 802.2–1985, ISO Draft International Standard 8802/2.

Both standards are published by the Institute of Electrical and Electronics Engineers, Inc. and are distributed in cooperation with Wiley-Interscience, a division of John Wiley & Sons, Inc.

Terms and Acronyms

The following terms are used in this manual:

802
: Refers to both ISO 8802-2 and ISO 8802-3. ISO 8802-2 is equivalent to IEEE 802.2 and ISO 8802-3 is equivalent to IEEE 802.3.

Ethernet
: A physical transmission media (cables and controllers) and data link software that provides access to the physical media according to the CSMA/CD protocol. DLI programs can transfer data over Ethernet using the usual Ethernet frame format or the Open Systems Interconnect (OSI) 8802-3 frame format.

The following acronyms are used in this manual:

DLI
: Data Link Interface

DNA	Digital Network Architecture
DSAP	Destination Service Access Point
IEEE	Institute of Electrical and Electronics Engineers
LLC	Logical Link Control
SAP	Service Access Point
SSAP	Source Service Access Point
SNAP	Subnetwork Access Protocol
XID	Exchange Identification

Graphic Conventions

Convention	Meaning
Special type	In text, each mention of a specific command, option, partition, pathname, directory, or file is presented in this type.
command(x)	In text, cross-references to the command documentation include the section number in the reference manual where the commands are documented. For example: See the **cat(1)** command. This indicates that you can find the material on the **cat** command in Section 1 of the reference pages.
italics	In syntax descriptions, this type indicates terms that are optional.
[ ]	In syntax descriptions, square brackets indicate terms that are optional.
...	In syntax descriptions, a horizontal ellipsis indicates that the preceding item can be repeated one or more times.
UPPERCASE	The ULTRIX system differentiates between lowercase and uppercase characters. Enter uppercase characters only where specifically indicated by an example or a syntax line.
`example`	In examples, computer output text is printed in this type.

All Ethernet addresses are hexadecimal; all other numbers are decimal unless otherwise noted.

New and Changed Information

This is a new manual.

Chapter 1

Introduction to the Data Link Interface

This chapter describes the capabilities of the Data Link Interface (DLI) and explains some basic DLI concepts.

1.1 What Is DLI?

DLI is network software that lets programs on an ULTRIX Ethernet node communicate with data link programs on another Ethernet node. DLI lets a user program interact directly with the broadcast network device drivers of an ULTRIX system.

DLI programs can transfer data over an Ethernet using the standard Ethernet frame format or the Open Systems Interconnect (OSI) 8802-3 frame format. Your ULTRIX node can run DECnet, Internet, and DLI programs concurrently. Each program can communicate with programs executing on a remote Ethernet node.

1.1.1 DLI Services

DLI provides the following services at the data link layer:

- Datagram service.
- Ethernet or 802 packets.
- ISO 8802-2 Class I, Type I service.
- Ability to use a multicast address mode to send and receive messages over the network. Multicast address mode sends simultaneous messages to a group of nodes.

1.1.2 Hardware Support

DLI supports the following hardware:

- DEBNA
- DEBNI (XNA)
- DEBNT
- DELQA
- DEQNA
- DESVA
- DEUNA

1.2 Using DLI to Access the Ethernet

An Ethernet data link on a single Ethernet controller supports multiple concurrent users. Each station represents an available port onto the Ethernet channel.

Because multiple users simultaneously access the Ethernet channel, your program must use addressing mechanisms that ensure delivery of messages to the correct recipient. Any message you transmit on the Ethernet must include an Ethernet address that identifies the target node. The message must also include an additional identifier that directs the message to the correct user on the target node; this identifier varies according to the frame format you choose to use. DLI builds frames according to the Ethernet or IEEE 8802-3 standard.

1.3 Prerequisites for DLI Programming

ULTRIX allows only superusers to run DLI applications. If you do not have the required privileges to run a DLI program, see your system manager.

DLI programming requires both a thorough knowledge of the C programming language and experience writing system programs. If you intend to use the 802 substructure, you should also be familiar with 802 protocols.

1.4 Including Higher-Level Services

DLI provides only datagram service. Because DLI is a direct interface to the Data Link layer, it does not offer higher-level services normally provided by Internet and DECnet. Therefore, your application should provide the following kinds of services:

- Packet routing and guaranteed delivery
- Flow control
- Error recovery
- Data segmentation

1.5 DLI Concepts

Familiarity with the following concepts will help you effectively use the DLI programming procedures described in this manual:

- Datagram Sockets—Your application uses sockets to send and receive both Ethernet and 8802-3 frames. DLI uses datagram sockets only.

  For more information about sockets and how to use them, see the *Guide to Network Programming*.

- Logical Link Control (LLC)—LLC is a sublayer of DLI that provides a set of services determined by a value in the 8802-2 frame format.

- Physical and Multicast Addressing—You can send and receive messages over the network using physical or multicast addresses. You can send physical addresses to a single destination node. Multicast addresses are not associated with any specific node; instead, a packet sent to a multicast address is received by all nodes with the multicast address enabled.

- Standard Frame Formats—The Ethernet format is a proprietary standard that belongs to Digital Equipment Corporation, Intel Corporation, and Xerox Corporation. The IEEE 8802-3 format is a standard for multivendor networking. Any single application can send and receive both types of frames.

Chapter 2

Understanding the DLI Socket Address Data Structure

This chapter describes the function of the DLI Socket Address data structure (**sockaddr_dl**). It explains how you use **sockaddr_dl** to specify the domain address, the network device, and the Ethernet and 8802-3 substructures.

2.1 How the sockaddr_dl Structure Works

DLI provides a socket address data structure through which you can configure the set of services required for communication at the data link layer. The data structure **sockaddr_dl** is used to convey information to DLI when an application binds to the network, or when it transmits a packet to the network. DLI also uses it to convey information to the application when it receives a packet from the network. This includes network device information, the packet format to be used, and addressing information.

The following example shows the DLI socket address structure:

```
#define DLI_ETHERNET    0
#define DLI_802         2
struct sockaddr_dl
{
    u_short dli_family;            /* address family (AF_DLI) */
    struct dli_devid dli_device;   /* id of communication device */
    u_char dli_substructype;       /* id to interpret following union */
                                   /* set to DLI_ETHERNET or DLI_802 */
    union
    {
        struct sockaddr_edl dli_eaddr;   /* Ethernet support */
        struct sockaddr_802 dli_802addr; /* OSI 802 support */
    } choose_addr;
};
```

2.1.1 Specifying Values Within the Structure

You can use system calls to specify values within the socket address structure by using either the Ethernet or 802 substructures. The fields within the substructures are updated as a function of the system call. For example, the **bind** call is used to specify the domain, network device, and most of the substructure. The **sendto** call is used to specify the domain, network device, and part of the substructure. The **recvfrom** call is used to specify the domain and network device, and DLI fills in part of the substructure.

For further information, see **bind(2)**, **recvfrom(2)**, and **sendto(2)** in the *ULTRIX Reference Pages*.

The **dli_econn** and **dli_802_3_conn** user-written subroutines open a socket and bind the associated domain, network device name, protocol type, and other substructure information to the socket.

The following code shows how the **dli_econn** subroutine can be used to fill in the **sockaddr_dl** data structure:

```
dli_econn(devname, devunit, ptype, taddr, ioctl)
char *devname;
u_short devunit;
u_short ptype;
u_char *taddr;
u_char ioctl;
{


    int i, sock;
    struct sockaddr_dl out_bind;

    if ( (i = strlen(devname)) >
          sizeof(out_bind.dli_device.dli_devname) )
    {
         fprintf(stderr, "dli_eth: bad device name");
         return(-1);
    }

    if ((sock = socket(AF_DLI, SOCK_DGRAM, DLPROTO_DLI)) < 0)
    {
         perror("dli_eth, can't open DLI socket");
         return(-1);
    }
  }
```

Section B-1 contains the complete code for the **dli_econn** subroutine.

2.1.2 Using Ethernet and 802 Substructures

Any single application can send and receive both Ethernet and 802 substructures. The Ethernet substructure enables applications to communicate across an Ethernet. The 802 substructure enables applications to use 8802-2 and 8802-3 protocols to communicate with each other.

The following sections describe the functions the Ethernet and 8802-3 substructures provide within the DLI **sockaddr_dl** data structure.

2.2 Using the Ethernet Substructure

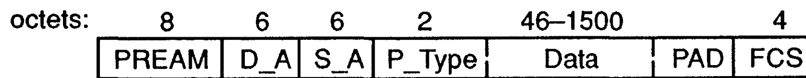
The Ethernet substructure lets you specify the target node's physical address, the client layer protocol type, and an I/O control flag for the protocol type. This information is used to create the Ethernet frame format.

The following example shows the DLI Ethernet socket address structure:

```
#define DLI_EADDRSIZE   6
struct sockaddr_edl
{
    u_char dli_ioctlflg;              /* filter on incoming packets */
    u_short dli_protype;              /* Ethernet protocol type */
    u_char dli_target[DLI_EADDRSIZE]; /* Ethernet address of target node */
    u_char dli_dest[DLI_EADDRSIZE];   /* Ethernet address used to address */
                                      /*  the local node; DLI places */
                                      /*  the destination address of an */
                                      /*  incoming packet here to be */
                                      /*  used in the recvfrom call. */
};
```

Figure 2-1 illustrates the Ethernet frame format.

Figure 2–1: The Ethernet Frame Format



2.2.1 How Ethernet Frames Work

All Ethernet frames contain a 16-bit identification number called an Ethernet protocol type. When a message arrives at the controller, the protocol type is issued to identify which port receives the frame. DLI applications that communicate across the Ethernet must always enable the same Ethernet protocol type. In addition to using protocol types to select a user for an incoming packet, you can configure DLI to select a user as a function of both the protocol type and the physical address of the remote node. This allows several applications in the same system to use the same type, which can make input/output simpler for the application.

2.2.2 Defining Ethernet Substructure Values

The following sections define the possible values for all members in the Ethernet substructure.

2.2.2.1 Target Node Physical Address

The target node physical address is a 48-bit unique value assigned by Xerox Corporation to a station on the Ethernet. For example, AA-00-03-22-14-55 is a valid Ethernet address and it is the address of the remote node.

If you do not specify the target node physical address with the **bind** call, you must specify this address when sending data by using the **sendto** call. In addition, you should use the **recvfrom** call to determine the source of a data message.

You can use either the physical address or a multicast address to send messages.

2.2.2.2 Protocol Type

The protocol type is a 16-bit value in the Ethernet frame following the source address. The Ethernet driver passes the protocol type to DLI for use in determining the recipient of the data in the frame. With the exception of reserved values, you can use any Ethernet protocol type assigned to you by Xerox Corporation and not used elsewhere in your system.

The following hexadecimal values are reserved for use by the system:

- 0X 800
- 0X 806
- 0X 1000 <*ptype*> 0X 1010, where *ptype* is the protocol type

2.2.2.3 I/O Control Flag

The I/O control flag is a value that DLI uses to determine how your program reserves a protocol type. That is, it is used by DLI to determine whether to select a user as a function of the protocol type alone, or as a function of the combination of the protocol type and the target audience. The following list defines the possible I/O control flags and describes the conditions for their use:

- **NORMAL** - Allows your program to exchange messages with one target node, using only the specified protocol type. When using the **NORMAL** flag, you must specify the target node physical address in the **bind** call, and you can use any of the data transfer calls to send and receive data. DLI forwards to the user all messages containing the specified protocol type from the specified target.
- **EXCLUSIVE** - Gives your program exclusive use of the specified protocol type and allows the program to exchange data with any other node using this protocol type. In other words, the program receives all messages with the specified protocol type. When you use the **EXCLUSIVE** flag, do not specify the target address with the **bind** call. You must use the **sendto** and **recvfrom** calls to exchange data with other nodes, and you must specify the target address with the **sendto** call. In the address structure (returned with **recvfrom**), DLI fills in the target address with the source address in the Ethernet frame. It also fills in the destination address with the destination address in the Ethernet frame.
- **DEFAULT** - Allows your program to receive messages that contain the specified protocol type and that are meant for no other program on the system. If no other program is bound exclusively to the protocol type or the protocol type/address pair in the message, the socket bound to the protocol type gets the message by default. This mode of operation is recommended for use in programs that listen for messages but do not necessarily send them. When you use the **DEFAULT** flag, do not specify the target address with the **bind** call. Use the **recvfrom** call to receive data from other nodes. If you are using the **DEFAULT** flag, DLI fills in the target node's physical address and the destination address used for the incoming message on this call.

2.3 Using the 8802-3 Substructure

The 8802-3 substructure enables applications to communicate with each other using the 8802-2 and 8802-3 protocol. It uses two basic modes of operation: Class I, Type 1 service, and the services supplied by your application using the 8802-2 protocol.

The following example shows the DLI 8802-3 socket address structure:

```
struct sockaddr_802                 /* 8802-3 sockaddr struct */
{
    u_char ioctl;                   /* filter on incoming packets */
                                    /* addressed to the SNAP SAP */
    u_char svc;                     /* service class for this portal */
    struct osi_802hdr eh_802;       /* OSI 802 header format */
};
```

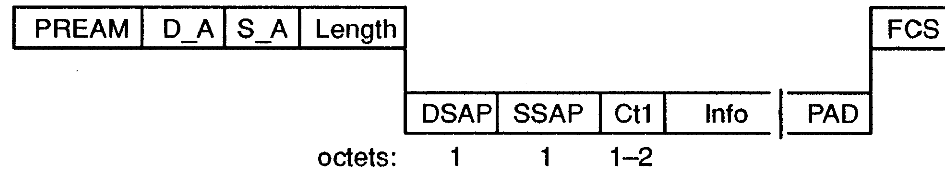
The 8802-3 substructure lets you specify values for the following field in the 8802-3 frame format:

- Target node physical address.
- Source service access point (SSAP). The protocol identifier and I/O control field may be required, depending on the type of SSAP you enable.
- Destination service access points (DSAPs).
  - Individual
  - Group

- Control field

Figure 2-2 illustrates the 8802-3 frame format.

Figure 2–2: The 8802-3 Frame Format



NOTE

> The Institute of Electrical and Electronics Engineers (IEEE) has defined the 802 frame format for communicating over the Ethernet. Before using the 8802-3 frame format, be sure to familiarize yourself with the IEEE standard.

2.3.1 Defining 802 Substructure Values

The following sections define the possible values for all members in the 802 substructure.

2.3.1.1 Target Node Physical Address

The target node physical address is a 48-bit unique value assigned by Xerox Corporation to a station on the Ethernet. For example, AA-00-03-22-14-55 is a valid Ethernet address. This is the address of the remote node, with which the application will attempt to exchange packets. It must be specified in the **bind** call, except when the I/O control field is ether **EXCLUSIVE** or **DEFAULT** and the service access point (SAP) is a **SNAP_SAP** type. The SAP must be specified in the **sendto** call.

2.3.1.2 Service Class

The service class is a value in the 8802-3 substructure that determines the capabilities and features provided by the Logical Link Control (LLC) sublayer of the data link layer. The possible service classes are:

- **TYPE1** - This value causes DLI to interpret all header information and provide Class I, Type 1 service.

When Type 1 service is used, the DLI software handles the **XID** and **TEST** packets. This is transparent to the application. DLI uses the source and destination service access points to determine who should receive the message; it interprets the control field on behalf of the user. Whether DLI passes the data field to the user depends on the value of the control field.

- **USER** - This value provides few services. The user must, therefore, implement most of the 8802-2 protocol. In other words, the application must handle the **XID** and **TEST** packets. DLI uses the source and destination service access points, but it passes the control field with the data to the user. The user must interpret the control field. This mode should be selected if the application needs to implement Class II, Type 2 service.

2.3.2 Destination Service Access Point

The destination service access point (DSAP) is a field in the 8802-3 structure that identifies the application for which the message is intended. You can use individual or group DSAPs to identify one user or a group of users. (You can use group DSAPs only when the service class is set to **USER**.) The possible values for this field are:

- Individual DSAPs:

  NULL_SAP - A DSAP consisting of all zeros. You can send **TEST** and **XID** commands and responses, but no data, to a **NULL_SAP**. (**TEST** and **XID** are explained later in this section.) The data link layer uses the **NULL_SAP** to talk to another data link layer, primarily for testing.

  SNAP_SAP - The 8802-3 Subnetwork Access Protocol. When you send data, you should enable a source **SNAP_SAP** and send to a remote destination **SNAP_SAP**.

  Only unnumbered information (UI) packets can be sent or received when using the **SNAP_SAP**.

  User-defined DSAP - Identifies one user for whom the message is intended. The user-defined individual DSAP must be an even number greater than or equal to 2 and less than or equal to 254.

- Group DSAP (user-defined) - Identifies more than one user for whom the message is intended. You can send data to a maximum of 127 group DSAPs on one socket. The user-defined group DSAP must be an odd number greater than or equal to 3 and less than or equal to 255. Note that the 255 number is the global SAP and must be enabled like any other group SAP. You can use Group SAPs only when the service class is set to **USER**.

2.3.2.1 Source Service Access Point

The source service access point (SSAP) is a field in the 8802-3 frame that identifies the address of the application that sent the message. You can enable only one SSAP on a socket. The possible values for this packet are:

- **SNAP_SAP** - The 8802-3 Subnetwork Access Protocol. When you enable a source **SNAP_SAP** you must specify the protocol identifier and control field. The protocol identifier is five bytes. The control field is one byte. Enabling the **SNAP_SAP** is allowed only when the service class is **TYPE1**.

Only unnumbered information (UI) packets can be sent or received when using the **SNAP_SAP**.

- User-defined - The individual SSAP must be an even number greater than or equal to 2 and less than or equal to 254.

The IEEE 8802-2 Standard reserves for its own definition all SAP addresses with the second least significant bit set to 1. It is suggested that you use these SAP values for their intended purposes, as defined in the IEEE 8802-2 Standard.

2.3.2.2 Control Field

The control field specifies the packet type. The following values are defined for Type 1, Class I service, and can also be used in user-supplied mode to provide Type 2, Class II service.

An application using this user mode is responsible for providing the correct services. For other operations supported by CLASS II service, see the *IEEE Standards for Local Area Networks: Logical Link Control*, published by the Institute of Electrical and Electronics Engineers, Inc.

- Exchange Identification – The value **XID** identifies the exchange identification command or response. An 8-bit format identifier and a 16-bit parameter follow the **XID** control field. The 16-bit parameter identifies the supported LLC services and the receive window size. The LLC is the top sublayer in the Data Link layer of the IEEE/Std 802 Local Area Network Protocol. The following values of **XID** are defined in the DLI header file, **dli_var.h**:

  XID_PCMD - Exchange identification command with the poll bit set. The exchange identification command conveys the types of LLC services supported and the receive window size to the destination LLC. This command causes the destination LLC to reply with the **XID** response Protocol Data Unit (PDU) at the earliest opportunity. The poll bit is set to 1, soliciting a response PDU.
  XID_NPCMD - Exchange identification command with no poll bit set. This command is identical to the previous command, except that you clear the poll bit. No response is expected.
  XID_PRSP - Exchange identification response with the poll bit set. The Data Link layer uses the exchange identification response to reply to an **XID** command at the earliest opportunity. The **XID** response PDU identifies the responding LLC and includes an information field like that defined for the **XID** command PDU, regardless of what information is present in the information field of the received **XID** command PDU. The final bit is set to 1, indicating that this response is sent by the LLC as a reply to a soliciting command PDU.
  XID_NPRSP - Exchange identification response with no poll bit set. This response is identical to the previous one, except that the final bit is cleared.

- LLC Protocol Data Unit Test – The value **TEST** identifies the LLC PDU command or response test. The **TEST** control field can be followed by a data field. The following values of **TEST** are defined in the DLI header file, **dli_var.h**:

  TEST_PCMD - Test command with the poll bit set. This command tests the LLC-to-LLC transmission path by causing the destination LLC to respond with the **TEST** response Protocol Data Unit (PDU) at the earliest opportunity. An information field is optional with this control field value. If used, the receiving LLC returns the information rather than passing it to the user. The poll bit is set to 1, soliciting a response PDU.

TEST_NPCMD - Test command with no poll bit set. This command is identical to the previous command, except that the poll bit is cleared.
TEST_PRSP - Test response with the poll bit set. This response Protocol Data Unit (PDU) is a reply to the **TEST** command PDU. An information field, if present in the **TEST** command PDU, is returned in the corresponding **TEST** response PDU. The final bit is set to 1, indicating that this response is sent by the LLC as a reply to a soliciting command PDU.
TEST_NPRSP - Test response with no poll bit set. This response is identical to the previous one, except that the final bit is cleared.

- Unnumbered Information Command – The Unnumbered information command with no poll set (**UI_NPCMD**) sends information to one or more LLCs. The **UI_NPCMD** command does not have an LLC response PDU. This is usually passed up to the application.

  Applications generally send and receive data using this command.

Chapter 3

Writing DLI Programs

This chapter explains how to use ULTRIX system calls to write DLI programs and describes procedures for specifying values within the Ethernet and 8802-3 substructures. Appendix B contains DLI programming examples of procedures described in this chapter.

For additional information on how to use sockets and system calls to write application programs, refer to the *Guide to Network Programming*.

3.1 Supplying Data Link Services

Your DLI application should contain other services that the higher levels of network software normally provide:

- Flow control—DLI programs running on different nodes must synchronize data transfer or they will lose data.
- Error recovery—DLI reports errors, but your application must recover from them.
- Data segmentation—Your application must segment data during transmission. (See Section 3.5.2 for instructions on how to calculate the buffer size for Ethernet and 8802-3 packets.)

3.2 Using ULTRIX System Calls

Your DLI program uses standard ULTRIX system calls with input arguments, structures, and substructures specific to DLI. For example, when issuing the **socket** call, your program uses the the address format **AF_DLI** and the protocol **DLPROTO_DLI**.

At the beginning of any DLI program, you must include the header file, **/sys/netdnet/dli_var.h**. Then, follow the calling sequence shown in Table 3-1.

Table 3–1: Calling Sequence for DLI Programs

Function	System Call
Create a socket.	**socket**
Bind the socket to a device by specifying the address family, the frame format type, and the device over which the program will send the data using the **sockaddr_dl** structure.	**bind**
Set socket options. This call is optional.	**setsockopt**
Transfer data.	**write**
	send
	sendto
	read
	recv
	recvfrom
Deactivate the socket descriptor.	**close**

The following sections describe DLI functions, input arguments, and structures.

3.3 Creating a Socket

Your DLI application must create a socket by using the **socket** system call with the following input arguments:

Address family: **AF_DLI**
Socket type: **SOCK_DGRAM**
Protocol: **DLPROTO_DLI**

The value **AF_DLI** specifies the DLI address family. **SOCK_DGRAM** creates a datagram socket, which is the only type of socket that DLI allows. DLI does not supply the services necessary for connecting to other programs and for using other socket types. The value **DLPROTO_DLI** specifies the DLI protocol module.

The following example shows how the **socket** call is used to open a socket to DLI:

```
int so;
   .
   .
   .
if ( (so = socket(AF_DLI,SOCK_DGRAM,DLPROTO_DLI))<0)
  {
      perror("cannot open DLI socket");
      return (-1);
  }
```

3.4 Setting Socket Options

Use the **setsockopt** call to set the following socket options within the **sockaddr_dl** structure:

DLI_ENAGSAP: Enables a Group Service Access Point (GSAP)
DLI_DISGSAP: Disables a Group Service Access Point (GSAP)
DLI_SET802CTL: Sets the 802 control field
DLI_MULTICAST: Enables the reception of all messages addressed to a multicast address

The following code examples show how to use the **setsockopt** call to set the socket options.

This example shows how the **setsockopt** call is used to enable the GSAP option:

```
/* enable GSAPs supplied by user */
 j = 3;
 i = 0;
 while (j < argc ) {
     sscanf(argv[j++], "%x", &k);
     out_opt[i++] = k;
 }
 optlen = i;
 if
 (setsockopt(sock,DLPROTO_DLI,DLI_ENAGSAP,&out_opt[0],optlen) < 0){
     perror("dli_setsockopt: Can't enable gsap");
     exit(1);
 }
```

This example shows how the **setsockopt** call is used to disable the GSAP option:

```
/* disable all but the last 4 or all GSAPs, */
/* whichever is smallest */
if ( optlen > 4 )
    optlen -= 4;
if
(setsockopt(sock,DLPROTO_DLI,DLI_DISGSAP,&out_opt[0],optlen) < 0){
    perror("dli_setsockopt: Can't disable gsap");
}
```

This example shows how the **setsockopt** call is used to set the 802 control field:

```
/* set 802 control field */
out_opt[0] = TEST_PCMD;
optlen = 1;
if
(setsockopt(sock,DLPROTO_DLI,DLI_SET802CTL,
           &out_opt[0],optlen)<0){
    perror("dli_setsockopt: Can't set 802 control");
    exit(1);
}
```

This example shows how the **setsockopt** call is used to enable two multicast addresses:

```
/* enable two multicast addresses */
bcopy(mcast0, out_opt, sizeof(mcast0));
bcopy(mcast1, out_opt+sizeof(mcast0), sizeof(mcast1));

if ( setsockopt(sock, DLPROTO_DLI, DLI_MULTICAST, &out_opt[0],
                (sizeof(mcast0) + sizeof(mcast1))) < 0 ) {
    perror("dli_setsockopt: can't enable multicast");
}
```

See Section B.5 for more detailed code examples.

3.5 Binding the Socket

After you create the socket, your application must bind the socket to a network device. At this point, you specify the type of format for the message. You assign a name to the socket, where the variable *name* is a pointer to a structure of the type **sockaddr_dl**. Then, you must fill in the **sockaddr_dl** data structure and include the substructures (Ethernet, 802, or both).

To bind the socket, use the following format:

```
bind (s,name,namelen)
int s;
struct sockaddr_dl *name;
int namelen;
```

For more information about the **bind** call, see **bind(2)** in the *ULTRIX Reference Pages*.

3.5.1 Filling in the sockaddr_dl Structure

Fill in the **sockaddr_dl** structure with the following information:

- Address family
- I/O device ID
- Substructure type

3.5.1.1 Specifying the Address Family

To specify the Address family, use the value **AF_DLI** in the **socket** call.

3.5.1.2 Specifying the I/O device ID

The I/O device is the controller over which your program sends and receives data to and from the target node. The I/O device ID consists of the device name (*dli_devname*) and the device number (*dli_devnumber*). Definitions for each variable follow:

- *dli_devname*: The possible device names are: de, qe, ln, xna, and ni.
- *dli_devnumber*: The device number is set up in the system configuration file.

3.5.1.3 Specifying the Substructure Type

The substructure specifies the type of frame format that the program will use. Definitions for each variable follow:

- *dli_eaddr*: Ethernet frame format **(DLI_ETHERNET)**
- *dli_802addr*: 8802-3 frame format **(DLI_802)**

A program can send and receive both Ethernet and 8802-3 frames, as long as it has a socket for each type. For example, your DLI program might communicate with one node using the Ethernet frames and another node using 8802-3 frames. Your choice of frame formats depends on the frame types used by the target program.

NOTE

Only one type of frame is allowed for each socket.

Your program must specify the packet header for sending your message after filling in the substructure of your choice. The following examples show how to fill the **sockaddr_dl** structure.

This example shows how to fill the **sockaddr_dl** structure for the Ethernet protocol:

```
 /*
     * fill out the sockaddr_dl structure for the bind call
     */
    bzero(&out_bind, sizeof(out_bind));
    out_bind.dli_family = AF_DLI;
    out_bind.dli_substructype = DLI_ETHERNET;
    bcopy(devname, out_bind.dli_device.dli_devname, i);
    out_bind.dli_device.dli_devnumber = devunit;
    out_bind.choose_addr.dli_eaddr.dli_ioctlflg = ioctl;
    out_bind.choose_addr.dli_eaddr.dli_protype = ptype;
    if ( taddr )
         bcopy(taddr, out_bind.choose_addr.dli_eaddr.dli_target,
               DLI_EADDRSIZE);

    if ( bind(sock, &out_bind, sizeof(out_bind)) < 0 )
    {
         perror("dli_eth, can't bind DLI socket");
         return(-1);
    }

    return(sock);
}
```

This example shows how to fill the **sockaddr_dl** substructure for the 802 protocol:

```
/*
     * fill out sockaddr_dl structure for the bind call.
     * note that we need to determine whether the
     * control field is 8 bits (unnumbered format) or
     * 16 bits (informational/supervisory format).  We do this
     * by checking the low order 2 bits, which are both 1 only
     * for unnumbered control fields.
     */
    bzero(&out_bind, sizeof(out_bind));
    out_bind.dli_family = AF_DLI;
    out_bind.dli_substructype = DLI_802;
    bcopy(devname, out_bind.dli_device.dli_devname, i);
    out_bind.dli_device.dli_devnumber = devunit;
    out_bind.choose_addr.dli_802addr.ioctl = ioctl;
    out_bind.choose_addr.dli_802addr.svc = svc;
    if(ctl & 3)
        out_bind.choose_addr.dli_802addr.eh_802.ctl.U_fmt=(u_char)ctl;
    else
        out_bind.choose_addr.dli_802addr.eh_802.ctl.I_S_fmt = ctl;
    out_bind.choose_addr.dli_802addr.eh_802.ssap = sap;
    out_bind.choose_addr.dli_802addr.eh_802.dsap = dsap;
    if ( ptype )
        bcopy(ptype,out_bind.choose_addr.dli_802addr.eh_802.osi_pi,5);
    if ( taddr )
         bcopy(taddr, out_bind.choose_addr.dli_802addr.eh_802.dst,
               DLI_EADDRSIZE);
    if ( bind(sock, &out_bind, sizeof(out_bind)) < 0 )
    {
         perror("dli_802, can't bind DLI socket");
         return(-1);
    }

    return(sock);
}
```

3.5.2 Calculating the Buffer Size

The buffer size should be no larger than the controllers on the communicating nodes can handle, or you will lose data. The maximum buffer size for Ethernet packets is 1500 bytes. The maximum buffer size for 8802-3 packets is calculated as follows:

```
bytes = 1500 - [ 2 + (control field == UI? 1:2) +
                  (Source SAP == SNAP SAP ? 5:0)]
```

The number of bytes in the control field and in the Source SAP are specified in the **bind** call.

3.6 Transferring Data

The DLI program can use the **write**, **send**, or **sendto** calls to send data and the **read**, **recv**, or **recvfrom** calls to receive data. The "X" in Table 3-2 indicates the conditions under which you can use the system calls with the **bind** call.

NOTE

You must set the target address in the **bind** call when using the Normal control flag, not when using the Exclusive or Default control flags.

Table 3–2: Data Transfer System Calls Used with DLI

System Calls	Normal Control	Exclusive Control	Default Control
write	X		
send	X		
sendto	X	X	X
read	X		
recv	X		
recvfrom	X	X	X

When you set the control flag to **NORMAL**, set the target address in the **bind** call. Then use any of the following calls to transfer data: **write**, **send**, **sendto**, **read**, **recv**, **recvfrom**.

When you set the control flag to **EXCLUSIVE**, make the value of the target address in the **bind** call zero. Then, set the target address in the **sendto** call. Use only the **sendto** and **recvfrom** calls to transfer data.

When you set the control flag to **DEFAULT**, make the value of the target address in the **bind** call zero. Then, use the **sendto** call to send data and set the target address in that call. Use the **recvfrom** call to determine the source address of any data.

3.7 Deactivating the Socket

When you have finished sending or receiving data, you can deactivate the socket by issuing the **close** call. For more information about the **close** call, see **close(2)** in the *ULTRIX Reference Pages*.

Appendix A

DLI Data Structures

This chapter contains the DLI data structures that are defined in the header file, **/usr/include/netdnet/dli_var.h**. For guidelines on specifying these data structures, see the relevant system calls and the header file, **dli_var.h**.

A.1 DLI Socket Address Structure

The following example shows the DLI socket address structure:

```
#define DLI_ETHERNET    0
#define DLI_802         2
struct sockaddr_dl
{
    u_short dli_family;             /* address family (AF_DLI) */
    struct dli_devid dli_device;    /* id of communication device */
    u_char dli_substructype;        /* id to interpret following union */
                                    /* set to DLI_ETHERNET or DLI_802 */
    union
    {
        struct sockaddr_edl dli_eaddr;   /* Ethernet support */
        struct sockaddr_802 dli_802addr; /* OSI 802 support */
    } choose_addr;
};
```

A.2 DLI Device Identification Structure

The following example shows the DLI device identification structure:

```
#define DLI_DEVSIZE     16
struct dli_devid
{
    u_char dli_devname[DLI_DEVSIZE+1];    /* device name */
    u_short dli_devnumber;                /* device unit number */
};
```

A.3 DLI Ethernet Socket Address Structure

The following example shows the DLI Ethernet socket address structure:

```
#define DLI_EADDRSIZE   6
struct sockaddr_edl
{
    u_char dli_ioctlflg;              /* filter on incoming packets */
    u_short dli_protype;              /* Ethernet protocol type */
    u_char dli_target[DLI_EADDRSIZE]; /* Ethernet address of target node */
    u_char dli_dest[DLI_EADDRSIZE];   /* Ethernet address used to address */
                                      /*  the local node; DLI places */
                                      /*  the destination address of an */
                                      /*  incoming packet here to be */
                                      /*  used in the recvfrom call. */
};
```

A.4 DLI 8802-3 Socket Address Structure

The following example shows the DLI 8802-3 socket address structure:

```
struct sockaddr_802                 /* 8802-3 sockaddr struct */
{
    u_char ioctl;                   /* filter on incoming packets */
                                    /* addressed to the SNAP SAP */
    u_char svc;                     /* service class for this portal */
    struct osi_802hdr eh_802;       /* OSI 802 header format */
};
```

A.5 8802-3 Packet Header Format

The following example shows the 8802-3 packet header format:

```
#define DLI_EADDRSIZE   6
struct  osi_802hdr
{
    u_char dst[DLI_EADDRSIZE];  /* Ethernet address of remote node */
    u_char src[DLI_EADDRSIZE];  /* Ethernet address of local node */
    u_short len;                /* length of user data and 802 header */
    u_char dsap;                /* SAP used by the remote node */
    u_char ssap;                /* SAP used by the local node */
    union {                     /* control field */
           u_char U_fmt;               /* unnumbered information format */
           u_short I_S_fmt;            /* information/supervisory format */
    }ctl;
    u_char osi_pi[5];m          /* 8802-3 protocol identifier used */
                                /*  when SNAP SAP is used */
};
```

Appendix B

DLI Programming Examples

This appendix presents the following DLI programming examples:

B.1: A sample DLI client program using Ethernet format packets
B.2: A sample DLI server program using Ethernet format packets
B.3: A sample DLI client program using 8802-3 format packets
B.4: A sample DLI server program using 8802-3 format packets
B.5: A sample DLI program using **getsockopt** and **setsockopt** system calls

These programming examples are also available on-line in **/usr/examples/dli**.

B.1 Sample DLI Client Program Using Ethernet Format Packets

```
#ifndef lint
static  char  *sccsid = "@(#)dli_eth.c  1.5  3/27/90";
#endif lint

#include <stdio.h>
#include <ctype.h>
#include <errno.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <net/if.h>
#include <netinet/in.h>
#include <netinet/if_ether.h>
#include <netdnet/dli_var.h>
#include <sys/ioctl.h>

extern int errno;

/*
 *      d l i _ e x a m p l e : d l i _ e t h
 *
 * Description: This program sends out a message to a node where a
 *              companion program, dli_ethd, echoes the message.
 *              The Ethernet packet format is used.  The Ethernet
 *              address of the node where the companion program is
 *              running, the protocol type, and the message are
 *              supplied by the user.  The companion program should
 *              be started before executing this program.
 *
 * Inputs:      device, target address, protocol type, short message.
 *
 * Outputs:     Exit status.
 *
 * To compile:  cc -o dli_eth dli_eth.c
 *
 * Example:     dli_eth qe0 08-00-2b-02-e2-ff 6006 "Echo this"
 *
 * Comments:    This example demonstrates the use of the "NORMAL" I/O
 *              control flag.  The use of the "NORMAL" flag means that
 *              we can communicate only with a single specific node
 *              whose address is specified during the bind.  Because
 *              of this, we can use the normal write and read system
 *              calls on the socket, because the source/destination of
 *              all data that is read/written on the socket is fixed.
 *
 */

/*
 * Digital Equipment Corporation supplies this software example on
 * an "as-is" basis for general customer use.  Note that Digital
 * does not offer any support for it, nor is it covered under any
 * of Digital's support contracts.
 */

main(argc, argv, envp)
int argc;
char **argv, **envp;

{
```

```
u_char inbuf[1500], outbuf[1500];
u_char target_eaddr[6];
u_char devname[16];
int rsize, devunit;
char *cp;
int i, sock;
u_short ptype, obsiz;

if ( argc < 5 )
{
    fprintf(stderr, "%s %s %s\n",
       "usage:",
       argv[0],
       "device ethernet-address hex-protocol-type short-message");
    exit(1);
}

/* get device name and unit number. */
bzero(devname, sizeof(devname));
i = 0;
cp = argv[1];
while ( isalpha(*cp) )
    devname[i++] = *cp++;
sscanf(cp, "%d", &devunit);

/* get phys addr of remote node */
bzero(target_eaddr, sizeof(target_eaddr));
i = 0;
cp = argv[2];
while ( *cp ) {
    if ( *cp == '-' ) {
        cp++;
        continue;
    }
    else {
        sscanf(cp, "%2x", &target_eaddr[i++]);
        cp += 2;
    }
}

/* get protocol type */
sscanf(argv[3], "%hx", &ptype);

/* get message */
bzero(outbuf, sizeof(outbuf));
if ( (obsiz = strlen(argv[4])) > 1500 ) {
    fprintf(stderr, "%s: message is too long\n", argv[0]);
    exit(1);
}
strcpy(outbuf, argv[4]);

/* open dli socket */
if ( (sock = dli_econn(devname, devunit, ptype,
                       target_eaddr, DLI_NORMAL)) < 0 ) {
    perror("dli_eth, dli_econn failed");
    exit(1);
}

/* send message to target.  minimum message size is 46 bytes. */
if ( write(sock, outbuf, (obsiz < 46 ? 46 : obsiz)) < 0 ) {
    sprintf(outbuf, "%s: DLI transmission failed", argv[0]);
    perror(outbuf);
    exit(2);
}
```

```
    /* wait for response */
    if ((rsize = read(sock, inbuf, sizeof(inbuf))) < 0 ) {
        sprintf(inbuf, "%s: DLI reception failed", argv[0]);
        perror(inbuf);
        exit(2);
    }
    if ( ! rsize ) {
        fprintf(stderr, "%s, no data returned\n", argv[0]);
        exit(2);
    }

    /* print results */
    printf("%s\n", inbuf);

    close(sock);

}

/*
 *              d l i _ e c o n n
 *
 *
 *
 * Description:
 *      This subroutine opens a dli socket, then binds an associated
 *      device name and protocol type to the socket.
 *
 * Inputs:
 *      devname         = ptr to device name
 *      devunit         = device unit number
 *      ptype           = protocol type
 *      taddr           = target address
 *      ioctl           = io control flag
 *
 *
 * Outputs:
 *      returns         = socket handle if success, otherwise -1
 *
 *
 */

dli_econn(devname, devunit, ptype, taddr, ioctl)
char *devname;
u_short devunit;
u_short ptype;
u_char *taddr;
u_char ioctl;
{
    int i, sock;
    struct sockaddr_dl out_bind;

    if ( (i = strlen(devname)) >
          sizeof(out_bind.dli_device.dli_devname) )
    {
         fprintf(stderr, "dli_eth: bad device name");
         return(-1);
    }

    if ((sock = socket(AF_DLI, SOCK_DGRAM, DLPROTO_DLI)) < 0)
    {
         perror("dli_eth, can't open DLI socket");
         return(-1);
    }
```

```
    /*
     * fill out bind structure
     */
    bzero(&out_bind, sizeof(out_bind));
    out_bind.dli_family = AF_DLI;
    out_bind.dli_substructype = DLI_ETHERNET;
    bcopy(devname, out_bind.dli_device.dli_devname, i);
    out_bind.dli_device.dli_devnumber = devunit;
    out_bind.choose_addr.dli_eaddr.dli_ioctlflg = ioctl;
    out_bind.choose_addr.dli_eaddr.dli_protype = ptype;
    if ( taddr )
         bcopy(taddr, out_bind.choose_addr.dli_eaddr.dli_target,
               DLI_EADDRSIZE);

    if ( bind(sock, &out_bind, sizeof(out_bind)) < 0 )
    {
         perror("dli_eth, can't bind DLI socket");
         return(-1);
    }

    return(sock);
}
```

B.2 Sample DLI Server Program Using Ethernet Format Packets

```
#ifndef lint
static  char  *sccsid = "@(#)dli_ethd.c          1.6  3/27/90";
#endif lint

#include <stdio.h>
#include <ctype.h>
#include <errno.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <net/if.h>
#include <netinet/in.h>
#include <netinet/if_ether.h>
#include <netdnet/dli_var.h>
#include <sys/ioctl.h>

extern int errno;

/*
 *
 *        d l i   e x a m p l e : d l i _ e t h d
 *
 * Description: This daemon program transmits any message it receives
 *              to the originating node, that is, it echoes the message.
 *              The device and protocol type are supplied by
 *              the user.  The program uses Ethernet format packets.
 *
 * Inputs:      device, protocol type.
 *
 * Outputs:     Exit status.
 *
 * To compile:  cc -o dli_ethd dli_ethd.c
 *
 * Example:     dli_ethd de0 6006
 *
 * Comments:    This example demonstrates the use of the "DEFAULT" I/O
 *              control flag, and the recvfrom & sendto system calls.
 *              By specifying "DEFAULT" when binding the DLI socket to
 *              the device we inform the system that this program will
 *              receive any Ethernet format packet with the given
 *              protocol type that is not meant for any other program
 *              on the system.  Since packets may arrive from
 *              different nodes, we use the recvfrom call to read the
 *              packets.  This call gives us access to the packet
 *              header information so that we can determine where the
 *              packet came from.  When we write on the socket we must
 *              use the sendto system call to explicitly give the
 *              destination of the packet.
 */

/*
 * Digital Equipment Corporation supplies this software example on
 * an "as-is" basis for general customer use.  Note that Digital
 * does not offer any support for it, nor is it covered under any
 * of Digital's support contracts.
 */

main(argc, argv, envp)
int argc;
char **argv, **envp;

{
```

```
u_char inbuf[1500], outbuf[1500];
u_char devname[16];
u_char target_eaddr[6];
char *cp;
int rsize, devunit;
int i, sock, fromlen;
u_short ptype, obsiz;
struct sockaddr_dl from;

if ( argc < 3 )
{
    fprintf(stderr,
            "usage: %s device hex-protocol-type\n", argv[0]);
    exit(1);
}

/* get device name and unit number. */
bzero(devname, sizeof(devname));
i = 0;
cp = argv[1];
while ( isalpha(*cp) )
    devname[i++] = *cp++;
sscanf(cp, "%d", &devunit);

/* get protocol type */
sscanf(argv[2], "%x", &ptype);

/* open dli socket */
if
((sock = dli_econn(devname, devunit, ptype, NULL, DLI_DEFAULT))<0)
{
    perror("dli_ethd, dli_econn failed");
    exit(1);
}

while ( 1 ) {
    /* wait for message */
    from.dli_family = AF_DLI;
    fromlen = sizeof(struct sockaddr_dl);
    if ((rsize = recvfrom(sock, inbuf, sizeof(inbuf),
                          NULL, &from, &fromlen)) < 0 ) {
        sprintf(inbuf, "%s: DLI reception failed", argv[0]);
        perror(inbuf);
        exit(2);
    }

    /* check header */
    if ( fromlen != sizeof(struct sockaddr_dl) ) {
        fprintf(stderr,"%s, incorrect header supplied\n",argv[0]);
        continue;
    }

    /* any data? */
    if ( ! rsize )
        fprintf(stderr, "%s, NO data received from ", argv[0]);
    else
        fprintf(stderr, "%s, data received from ", argv[0]);
    for ( i = 0; i < 6; i++ )
        fprintf(stderr, "%x%s",
                from.choose_addr.dli_eaddr.dli_target[i],
                ((i<5)?"-":" "));
    fprintf(stderr, "on protocol type %x\n",
            from.choose_addr.dli_eaddr.dli_protype);
```

```
        /* send response to originator. */
        if ( sendto(sock, inbuf, rsize, NULL, &from, fromlen) < 0 ) {
            sprintf(outbuf, "%s: DLI transmission failed", argv[0]);
            perror(outbuf);
            exit(2);
        }
    }
}
/*
 *              d l i _ e c o n n
 *
 *
 *
 * Description:
 *      This subroutine opens a dli socket, then binds an associated
 *      device name and protocol type to the socket.
 *
 * Inputs:
 *      devname         = ptr to device name
 *      devunit         = device unit number
 *      ptype           = protocol type
 *      taddr           = target address
 *      ioctl           = io control flag
 *
 *
 * Outputs:
 *      returns         = socket handle if success, otherwise -1
 *
 *
 */
dli_econn(devname, devunit, ptype, taddr, ioctl)
char *devname;
u_short devunit;
u_short ptype;
u_char *taddr;
u_char ioctl;
{
    int i, sock;
    struct sockaddr_dl out_bind;

    if ( (i = strlen(devname)) >
          sizeof(out_bind.dli_device.dli_devname) )
    {
         fprintf(stderr, "dli_ethd: bad device name");
         return(-1);
    }

    if ((sock = socket(AF_DLI, SOCK_DGRAM, DLPROTO_DLI)) < 0)
    {
         perror("dli_ethd, can't open DLI socket");
         return(-1);
    }

    /*
     * fill out bind structure
     */
    bzero(&out_bind, sizeof(out_bind));
    out_bind.dli_family = AF_DLI;
    out_bind.dli_substructype = DLI_ETHERNET;
    bcopy(devname, out_bind.dli_device.dli_devname, i);
    out_bind.dli_device.dli_devnumber = devunit;
    out_bind.choose_addr.dli_eaddr.dli_ioctlflg = ioctl;
    out_bind.choose_addr.dli_eaddr.dli_protype = ptype;
    if ( taddr )
         bcopy(taddr, out_bind.choose_addr.dli_eaddr.dli_target,
                DLI_EADDRSIZE);
```

```
    if ( bind(sock, &out_bind, sizeof(out_bind)) < 0 )
    {
         perror("dli_ethd, can't bind DLI socket");
         return(-1);
    }

    return(sock);
}
```

B.3 Sample DLI Client Program Using 8802-3 Format Packets

```
#ifndef lint
static  char  *sccsid = "@(#)dli_802.c  1.7  3/27/90";
#endif lint

#include <stdio.h>
#include <ctype.h>
#include <errno.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <net/if.h>
#include <netinet/in.h>
#include <netinet/if_ether.h>
#include <netdnet/dli_var.h>
#include <sys/ioctl.h>

extern int errno;

#define PROTOCOL_ID      {0x00, 0x00, 0x00, 0x00, 0x5}
u_char protocolid[] = PROTOCOL_ID;

/*
 *      d l i _ e x a m p l e : d l i _ 8 0 2
 *
 * Description: This program sends out a message to a node where a
 *              companion program, dli_802d, echoes the message.
 *              The 8802-3 packet format is used.  The Ethernet
 *              address of the node where the companion program is
 *              running, the sap, and the message are supplied by the
 *              user.  The companion program should be started before
 *              executing this program.
 *
 * Inputs:      device, target address, sap, short message.
 *
 * Outputs:     Exit status.
 *
 * To compile:  cc -o dli_802 dli_802.c
 *
 * Example:     dli_802 qe0 08-00-2b-02-e2-ff ac "Echo this"
 *
 * Comments:    This example demonstrates the use of 802 "TYPE1"
 *              service.  With TYPE1 service, the processing of
 *              XID and TEST messages is handled transparently by
 *              DLI, that is, this program doesn't have to be concerned
 *              with handling them.  If the SNAP SAP (0xAA) is
 *              selected, a 5-byte protocol id is also required.
 *              This example automatically uses a protocol id
 *              of PROTOCOL_ID when this SNAP SAP is used.  Also,
 *              note the use of DLI_NORMAL for the i/o control flag.
 *              DLI makes use of this only when that SNAP_SAP/Protocol
 *              ID pair is used. DLI filters all incoming messages
 *              by comparing the Ethernet source address and Protocol
 *              ID against the target address and Protocol ID set up
 *              in the bind call.  Only if a match occurs will DLI
 *              pass the message up to the application.
 */

/*
 * Digital Equipment Corporation supplies this software example on
 * an "as-is" basis for general customer use.  Note that Digital
 * does not offer any support for it, nor is it covered under any
 * of Digital's support contracts.
 */
```

```
main(argc, argv, envp)
int argc;
char **argv, **envp;

{
    u_char inbuf[1500], outbuf[1500];
    u_char target_eaddr[6];
    u_char devname[16];
    int rsize, devunit;
    char *cp;
    int i, sock, fromlen;
    struct sockaddr_dl from;
    u_short obsiz;
    u_int sap;
    u_char  *pi = 0;

    if ( argc < 5 )
    {
        fprintf(stderr, "%s %s %s\n",
                "usage:",
                argv[0],
                "device ethernet-address hex-sap short-message");
        exit(1);
    }

    /* get device name and unit number. */
    bzero(devname, sizeof(devname));
    i = 0;
    cp = argv[1];
    while ( isalpha(*cp) )
        devname[i++] = *cp++;
    sscanf(cp, "%d", &devunit);

    /* get phys addr of remote node */
    bzero(target_eaddr, sizeof(target_eaddr));
    i = 0;
    cp = argv[2];
    while ( *cp ) {
        if ( *cp == '-' ) {
            cp++;
            continue;
        }
        else {
            sscanf(cp, "%2x", &target_eaddr[i++]);
            cp += 2;
        }
    }

    /* get sap */
    sscanf(argv[3], "%x", &sap);

    /* get message */
    bzero(outbuf, sizeof(outbuf));
    if ( (obsiz = strlen(argv[4])) > 1500 ) {
        fprintf(stderr, "%s: message is too long\n", argv[0]);
        exit(2);
    }
    strcpy(outbuf, argv[4]);

    /* open dli socket. notice that if (and only if) the snap sap */
    /* was selected then a protocol id must also be provided. */
    if ( sap == SNAP_SAP )
        pi = protocolid;
    if ( (sock = dli_802_3_conn(devname, devunit, pi, target_eaddr,
                    DLI_NORMAL, TYPE1, sap, sap, UI_NPCMD)) < 0 ) {
        perror("dli_802, dli_econn failed");
        exit(3);
    }
```

```
    /* send message to target. minimum message size is 46 bytes. */
    if ( write(sock, outbuf, (obsiz < 46 ? 46 : obsiz)) < 0 ) {
        sprintf(outbuf, "%s: DLI transmission failed", argv[0]);
        perror(outbuf);
        exit(4);
    }

    /* wait for response from correct address */
    while (1) {
        bzero(&from, sizeof(from));
        from.dli_family = AF_DLI;
        fromlen = sizeof(struct sockaddr_dl);
        if ((rsize = recvfrom(sock, inbuf, sizeof(inbuf),
                              NULL, &from, &fromlen)) < 0 ) {
                sprintf(inbuf, "%s: DLI reception failed", argv[0]);
                perror(inbuf);
                exit(5);
        }
        if ( fromlen != sizeof(struct sockaddr_dl) ) {
                fprintf(stderr,"%s, invalid address size\n",argv[0]);
                exit(6);
        }
        if ( bcmp(from.choose_addr.dli_802addr.eh_802.dst,
                  target_eaddr, sizeof(target_eaddr)) == 0 )
                break;
    }

    if ( ! rsize ) {
        fprintf(stderr, "%s, no data returned\n", argv[0]);
        exit(7);
    }
    /* print message */
    printf("%s\n", inbuf);

    close(sock);
}

/*
 *              d l i _8 0 2 _ 3 _ c o n n
 *
 *
 *
 * Description:
 *      This subroutine opens a dli 8802-3 socket and then binds an
 *      associated device name and protocol type to the socket.
 *
 * Inputs:
 *      devname         = ptr to device name
 *      devunit         = device unit number
 *      ptype           = protocol type
 *      taddr           = target address
 *      ioctl           = io control flag
 *      svc             = service class
 *      sap             = source sap
 *      dsap            = destination sap
 *      ctl             = control field
 *
 *
 * Outputs:
 *      returns         = socket handle if success, otherwise -1
 *
 *
 */
```

```
dli_802_3_conn (devname,devunit,ptype,taddr,ioctl,svc,sap,dsap,ctl)
char *devname;
u_short devunit;
u_char *ptype;
u_char *taddr;
u_char ioctl;
u_char svc;
u_char sap;
u_char dsap;
u_short ctl;

{
    int i, sock;
    struct sockaddr_dl out_bind;

    if ( (i = strlen(devname)) >
         sizeof(out_bind.dli_device.dli_devname) )
    {
         fprintf(stderr, "dli_802: bad device name");
         return(-1);
    }

    if ((sock = socket(AF_DLI, SOCK_DGRAM, DLPROTO_DLI)) < 0)
    {
         perror("dli_802, can't open DLI socket");
         return(-1);
    }

    /*
     * fill out bind structure.  note that we need to determine
     * whether the ctl field is 8 bits (unnumbered format) or
     * 16 bits (informational/supervisory format).  We do this
     * by checking the low order 2 bits, which are both 1 only
     * for unnumbered control fields.
     */
    bzero(&out_bind, sizeof(out_bind));
    out_bind.dli_family = AF_DLI;
    out_bind.dli_substructype = DLI_802;
    bcopy(devname, out_bind.dli_device.dli_devname, i);
    out_bind.dli_device.dli_devnumber = devunit;
    out_bind.choose_addr.dli_802addr.ioctl = ioctl;
    out_bind.choose_addr.dli_802addr.svc = svc;
    if(ctl & 3)
        out_bind.choose_addr.dli_802addr.eh_802.ctl.U_fmt=(u_char)ctl;
    else
        out_bind.choose_addr.dli_802addr.eh_802.ctl.I_S_fmt = ctl;
    out_bind.choose_addr.dli_802addr.eh_802.ssap = sap;
    out_bind.choose_addr.dli_802addr.eh_802.dsap = dsap;
    if ( ptype )
        bcopy(ptype,out_bind.choose_addr.dli_802addr.eh_802.osi_pi,5);
    if ( taddr )
         bcopy(taddr, out_bind.choose_addr.dli_802addr.eh_802.dst,
                DLI_EADDRSIZE);
    if ( bind(sock, &out_bind, sizeof(out_bind)) < 0 )
    {
         perror("dli_802, can't bind DLI socket");
         return(-1);
    }

    return(sock);
}
```

B.4 Sample DLI Server Program Using 8802-3 Format Packets

```
#ifndef lint
static  char  *sccsid = "@(#)dli_802d.c         1.6  3/27/90";
#endif lint

#include <stdio.h>
#include <ctype.h>
#include <errno.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <net/if.h>
#include <netinet/in.h>
#include <netinet/if_ether.h>
#include <netdnet/dli_var.h>
#include <sys/ioctl.h>

extern int errno;

#define PROTOCOL_ID      {0x00, 0x00, 0x00, 0x00, 0x5}
u_char protocolid[] = PROTOCOL_ID;

/*
 *       d l i  e x a m p l e : d l i  8 0 2 d
 *
 * Description: This daemon program transmits any message it receives
 *              to the originating node, that is, it echoes the message.
 *              The device and sap are supplied by the user.
 *              The program uses 8802-3 format packets.
 *
 * Inputs:      device, sap.
 *
 * Outputs:     Exit status.
 *
 * To compile:  cc -o dli_802d dli_802d.c
 *
 * Example:     dli_802d de0 ac
 *
 * Comments:    This example demonstrates the recvfrom and sendto system
 *              calls.  Since packets may arrive from different nodes,
 *              we use the recvfrom call to read the packets.  This
 *              call gives us access to the packet header information
 *              so that we can determine where the packet came from.
 *              When we write on the socket, we must use the sendto
 *              system call to explicitly give the destination of
 *              the packet.  The use of the "DEFAULT" I/O control flag
 *              only has an affect when the SNAP SAP is used.
 *              When the SNAP SAP is used, any arriving packets that
 *              have the specified protocol id and that are not
 *              destined for some other program will be given to this
 *              program.
 */

/*
 * Digital Equipment Corporation supplies this software example on
 * an "as-is" basis for general customer use.  Note that Digital
 * does not offer any support for it, nor is it covered under any
 * of Digital's support contracts.
 */

main(argc, argv, envp)
int argc;
char **argv, **envp;

{
```

```
u_char inbuf[1500], outbuf[1500];
u_char devname[16];
u_char target_eaddr[6];
char *cp;
int rsize, devunit;
int i, sock, fromlen;
u_short obsiz;
u_char tmpsap, sap;
struct sockaddr_dl from;
u_char *pi = 0;

if ( argc < 3 )
{
    fprintf(stderr, "usage: %s device hex-sap\n", argv[0]);
    exit(1);
}

/* get device name and unit number. */
bzero(devname, sizeof(devname));
i = 0;
cp = argv[1];
while ( isalpha(*cp) )
    devname[i++] = *cp++;
sscanf(cp, "%d", &devunit);

/* get sap */
sscanf(argv[2], "%x", &sap);

/* open dli socket. note that if (and only if) the snap sap */
/* was selected then a protocol id must also be specified. */
if ( sap == SNAP_SAP )
    pi = protocolid;
if ((sock = dli_802_3_conn(devname, devunit, pi, target_eaddr,
                DLI_DEFAULT, TYPE1, sap, sap, UI_NPCMD)) < 0) {
    perror("dli_802d, dli_conn failed");
    exit(1);
}

/* listen and respond */
while ( 1 ) {
    /* wait for message */
    from.dli_family = AF_DLI;
    fromlen = sizeof(struct sockaddr_dl);
    if ((rsize = recvfrom(sock, inbuf, sizeof(inbuf), NULL,
                          &from, &fromlen)) < 0 ) {
        sprintf(inbuf, "%s: DLI reception failed", argv[0]);
        perror(inbuf);
        exit(2);
    }

    /* check header */
    if ( fromlen != sizeof(struct sockaddr_dl) ) {
        fprintf(stderr,"%s, incorrect header supplied\n",argv[0]);
        continue;
    }

    /*
     * Note that DLI swaps the source and destination saps and lan
     * addresses in the sockaddr_dl structure returned by the
     * recvfrom call.  That is, it places the DSAP in eh_802.ssap
     * and the SSAP in eh_802.dsap; it also places the destination
     * lan address in eh_802.src and the source lan address in
     * eh_802.dst.  This allows for minimal manipulation of
     * the address structure for subsequent sendto or dli
     * connection calls.
     */
```

```
        /* any data? */
        if ( ! rsize )
            fprintf(stderr, "%s: NO data received from ", argv[0]);
        else
            fprintf(stderr, "%s: data received from ", argv[0]);
        for ( i = 0; i < 6; i++ )
            fprintf(stderr, "%x%s",
                    from.choose_addr.dli_802addr.eh_802.dst[i],
                    ((i<5)?"-":" "));
        fprintf(stderr, "\n     on dsap %x ",
                from.choose_addr.dli_802addr.eh_802.ssap);
        if ( from.choose_addr.dli_802addr.eh_802.dsap == SNAP_SAP )
            fprintf(stderr,
                    "(SNAP SAP), protocol id = %x-%x-%x-%x-%x\n    ",
                    from.choose_addr.dli_802addr.eh_802.osi_pi[0],
                    from.choose_addr.dli_802addr.eh_802.osi_pi[1],
                    from.choose_addr.dli_802addr.eh_802.osi_pi[2],
                    from.choose_addr.dli_802addr.eh_802.osi_pi[3],
                    from.choose_addr.dli_802addr.eh_802.osi_pi[4]);
        fprintf(stderr, " from ssap %x ",
                from.choose_addr.dli_802addr.eh_802.dsap);
        fprintf(stderr, "\n\n");

        /* send response to originator. */
        if ( from.choose_addr.dli_802addr.eh_802.dsap == SNAP_SAP )
            bcopy(protocolid,
                  from.choose_addr.dli_802addr.eh_802.osi_pi, 5);
        if ( sendto(sock, inbuf, rsize, NULL, &from, fromlen) < 0 ) {
            sprintf(outbuf, "%s: DLI transmission failed", argv[0]);
            perror(outbuf);
            exit(2);
        }
    }
}

/*
 *              d l i _8 0 2 _ 3 _ c o n n
 *
 *
 *
 * Description:
 *      This subroutine opens a dli 8802-3 socket and then binds an
 *      associated device name and protocol type to the socket.
 *
 * Inputs:
 *      devname         = ptr to device name
 *      devunit         = device unit number
 *      ptype           = protocol type
 *      taddr           = target address
 *      ioctl           = io control flag
 *      svc             = service class
 *      sap             = source sap
 *      dsap            = destination sap
 *      ctl             = control field
 *
 *
 * Outputs:
 *      returns         = socket handle if success, otherwise -1
 *
 *
 */
```

```
dli_802_3_conn (devname,devunit,ptype,taddr,ioctl,svc,sap,dsap,ctl)
char *devname;
u_short devunit;
u_char *ptype;
u_char *taddr;
u_char ioctl;
u_char svc;
u_char sap;
u_char dsap;
u_short ctl;
{
    int i, sock;
    struct sockaddr_dl out_bind;

    if ( (i = strlen(devname)) >
         sizeof(out_bind.dli_device.dli_devname) )
    {
         fprintf(stderr, "dli_802d: bad device name");
         return(-1);
    }

    if ((sock = socket(AF_DLI, SOCK_DGRAM, DLPROTO_DLI)) < 0)
    {
         perror("dli_802d, can't open DLI socket");
         return(-1);
    }

    /*
     * fill out bind structure.  note that we need to determine
     * whether the ctl field is 8 bits (unnumbered format) or
     * 16 bits (informational/supervisory format).  We do this
     * by checking the low order 2 bits, which are both 1 only
     * for unnumbered control fields.
     */
    bzero(&out_bind, sizeof(out_bind));
    out_bind.dli_family = AF_DLI;
    out_bind.dli_substructype = DLI_802;
    bzero(&out_bind, sizeof(out_bind));
    out_bind.dli_family = AF_DLI;
    out_bind.dli_substructype = DLI_802;
    bcopy(devname, out_bind.dli_device.dli_devname, i);
    out_bind.dli_device.dli_devnumber = devunit;
    out_bind.choose_addr.dli_802addr.ioctl = ioctl;
    out_bind.choose_addr.dli_802addr.svc = svc;
    if(ctl & 3)
        out_bind.choose_addr.dli_802addr.eh_802.ctl.U_fmt=(u_char)ctl;
    else
        out_bind.choose_addr.dli_802addr.eh_802.ctl.I_S_fmt = ctl;
    out_bind.choose_addr.dli_802addr.eh_802.ssap = sap;
    out_bind.choose_addr.dli_802addr.eh_802.dsap = dsap;
    if ( ptype )
        bcopy(ptype,out_bind.choose_addr.dli_802addr.eh_802.osi_pi,5);
    if ( taddr )
         bcopy(taddr, out_bind.choose_addr.dli_802addr.eh_802.dst,
               DLI_EADDRSIZE);

    if ( bind(sock, &out_bind, sizeof(out_bind)) < 0 )
    {
         perror("dli_802d, can't bind DLI socket");
         return(-1);
    }

    return(sock);
}
```

B.5 Sample DLI Program Using getsockopt and setsockopt

```
#ifndef lint
static  char  *sccsid = "@(#)dli_setsockopt.c   1.5  3/27/90";
#endif lint

#include <stdio.h>
#include <ctype.h>
#include <errno.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <net/if.h>
#include <netinet/in.h>
#include <netinet/if_ether.h>
#include <netdnet/dli_var.h>
#include <sys/ioctl.h>

extern int errno;
int debug = 0;

#define PROTOCOL_ID       {0x00, 0x00, 0x00, 0x00, 0x5}
#define CUSTOMER0         {0xab, 0x00, 0x04, 0x00, 0x00, 0x00}
#define CUSTOMER1         {0xab, 0x00, 0x04, 0x00, 0x00, 0x01}

u_char mcast0[] = CUSTOMER0;
u_char mcast1[] = CUSTOMER1;
u_char protocolid[] = PROTOCOL_ID;

/*
 *
 *      d l i   e x a m p l e : d l i   s e t s o c k o p t
 *
 * Description: This program demonstrates the use of the DLI get
 *              and setsockopt calls.  It opens a socket, enables
 *              2 multicast addresses, changes the 802 control
 *              field, enables a number of group saps supplied by
 *              the user, and reads the group saps that are enabled.
 *
 * Inputs:      device, sap, group-saps.
 *
 * Outputs:     Exit status.
 *
 * To compile:  cc -o dli_setsockopt dli_setsockopt.c
 *
 * Example:     dli_setsockopt qe0 ac  5 9 d
 *
 * Comments:    When a packet arrives with a group dsap, all dli
 *              programs that have that group sap enabled will
 *              receive copies of that packet.  Group saps are
 *              those with the low order bit set.  Group sap 1
 *              is currently not allowed for customer use. Group
 *              saps with the second bit set (eg 3,7,etc) are
 *              reserved by IEEE.
 */

/*
 * Digital Equipment Corporation supplies this software example on
 * an "as-is" basis for general customer use.  Note that Digital
 * does not offer any support for it, nor is it covered under any
 * of Digital's support contracts.
 */

main(argc, argv, envp)
int argc;
char **argv, **envp;

{
```

```
u_char inbuf[1500], outbuf[1500];
u_char devname[16];
u_char target_eaddr[6];
char *cp;
int rsize, devunit;
int i, j, k, sock, fromlen;
u_short obsiz;
u_char tmpsap, sap;
struct sockaddr_dl from;
u_char *pi = 0;
u_char out_opt[1000], in_opt[1000];
int optlen, ioptlen = sizeof(in_opt);

if ( argc < 4 )
{
    fprintf(stderr, "usage: %s device hex-sap hex-groupsaps\n",
    argv[0]);
    exit(1);
}

/* get device name and unit number. */
bzero(devname, sizeof(devname));
i = 0;
cp = argv[1];
while ( isalpha(*cp) )
devname[i++] = *cp++;
sscanf(cp, "%d", &devunit);

/* get protocol type */
sscanf(argv[2], "%x", &sap);

/* open dli socket */
if ( sap == SNAP_SAP ) {
    fprintf(stderr,
            "%s: can't use SNAP_SAP in USER mode\n", argv[0]);
    exit(1);
}
if ( (sock = dli_802_3_conn(devname, devunit, pi, target_eaddr,
                 DLI_DEFAULT, USER, sap, sap, UI_NPCMD)) < 0 ) {
    perror("dli_setsockopt: dli_conn failed");
    exit(1);
}

/* enable two multicast addresses */
bcopy(mcast0, out_opt, sizeof(mcast0));
bcopy(mcast1, out_opt+sizeof(mcast0), sizeof(mcast1));

if ( setsockopt(sock, DLPROTO_DLI, DLI_MULTICAST, &out_opt[0],
                (sizeof(mcast0) + sizeof(mcast1))) < 0 ) {
    perror("dli_setsockopt: can't enable multicast");
}

/* set 802 control field */
out_opt[0] = TEST_PCMD;
optlen = 1;
if
(setsockopt(sock,DLPROTO_DLI,DLI_SET802CTL,&out_opt[0],optlen)<0)S (){
    perror("dli_setsockopt: Can't set 802 control");
    exit(1);
}
```

```
    /* enable GSAPs supplied by user */
    j = 3;
    i = 0;
    while (j < argc ) {
        sscanf(argv[j++], "%x", &k);
        out_opt[i++] = k;
    }
    optlen = i;
    if
    (setsockopt(sock,DLPROTO_DLI,DLI_ENAGSAP,&out_opt[0],optlen) < 0){
        perror("dli_setsockopt: Can't enable gsap");
        exit(1);
    }

    /* verify all gsaps are enabled */
    bzero(in_opt, (ioptlen = sizeof(in_opt)));
    if
    (getsockopt(sock,DLPROTO_DLI,DLI_GETGSAP,in_opt,&ioptlen) < 0){
        perror("dli_setsockopt: DLI getsockopt 2 failed");
        exit(1);
    }
    printf("number of enabled GSAPs = %d, GSAPS:", ioptlen);
    for(i = 0; i < ioptlen; i++) {
        if ( ! (i % 10) )
            printf("\n");
        printf("%2x ",in_opt[i]);
    }
    printf("\n");

    /* disable all but the last 4 or all GSAPs, */
    /* whichever is smallest */
    if ( optlen > 4 )
        optlen -= 4;
    if
    (setsockopt(sock,DLPROTO_DLI,DLI_DISGSAP,&out_opt[0],optlen) < 0){
        perror("dli_setsockopt: Can't disable gsap");
    }

    /* verify some gsaps still enabled */
    bzero(in_opt, (ioptlen = sizeof(in_opt)));
    if
    (getsockopt(sock,DLPROTO_DLI,DLI_GETGSAP,in_opt,&ioptlen) < 0){
        perror("dli_setsockopt: getsockopt 3 failed");
        exit(1);
    }
    printf("number of enabled GSAPs = %d, GSAPS:", ioptlen);
    for(i = 0; i < ioptlen; i++) {
        if ( ! (i % 10) )
    printf("\n");
        printf("%2x ",in_opt[i]);
    }
    printf("\n");

}
```

```
/*
 *              d l i _8 0 2 _ 3 _ c o n n
 *
 *
 *
 * Description:
 *      This subroutine opens a dli 8802-3 socket and then binds an
 *      associated device name and protocol type to it.
 *
 * Inputs:
 *      devname         = ptr to device name
 *      devunit         = device unit number
 *      ptype           = protocol type
 *      taddr           = target address
 *      ioctl           = io control flag
 *      svc             = service class
 *      sap             = source sap
 *      dsap            = destination sap
 *      ctl             = control field
 *
 *
 * Outputs:
 *      returns         = socket handle if success, otherwise -1
 *
 *
 */

dli_802_3_conn (devname,devunit,ptype,taddr,ioctl,svc,sap,dsap,ctl)
char *devname;
u_short devunit;
u_char *ptype;
u_char *taddr;
u_char ioctl;
u_char svc;
u_char sap;
u_char dsap;
u_short ctl;
{
    int i, sock;
    struct sockaddr_dl out_bind;

    if ( (i = strlen(devname)) >
         sizeof(out_bind.dli_device.dli_devname) )
    {
        fprintf(stderr, "dli_setsockopt: bad device name");
        return(-1);
    }

    if ((sock = socket(AF_DLI, SOCK_DGRAM, DLPROTO_DLI)) < 0)
    {
        perror("dli_setsockopt: can't open DLI socket");
        return(-1);
    }
```

```
    /*
     * fill out bind structure
     */
    bzero(&out_bind, sizeof(out_bind));
    out_bind.dli_family = AF_DLI;
    out_bind.dli_substructype = DLI_802;
    bcopy(devname, out_bind.dli_device.dli_devname, i);
    out_bind.dli_device.dli_devnumber = devunit;
    out_bind.choose_addr.dli_802addr.ioctl = ioctl;
    out_bind.choose_addr.dli_802addr.svc = svc;
    if(ctl & 3)
        out_bind.choose_addr.dli_802addr.eh_802.ctl.U_fmt=(u_char)ctl;
    else
        out_bind.choose_addr.dli_802addr.eh_802.ctl.I_S_fmt = ctl;
    out_bind.choose_addr.dli_802addr.eh_802.ssap = sap;
    out_bind.choose_addr.dli_802addr.eh_802.dsap = dsap;
    if ( ptype )
        bcopy(ptype,out_bind.choose_addr.dli_802addr.eh_802.osi_pi,5);
    if ( taddr )
        bcopy(taddr, out_bind.choose_addr.dli_802addr.eh_802.dst,
              DLI_EADDRSIZE);
    if ( bind(sock, &out_bind, sizeof(out_bind)) < 0 )
    {
        perror("dli_setsockopt: can't bind DLI socket");
        return(-1);
    }

    return(sock);
}
```

Index

A

Address family,
  specifying the, 3–2

B

bind system call,
  using, 2–1, 2–4
Buffer size,
  calculating, 3–7

C

Concurrent programs,
  running, 1–1
Control field,
  function of, 2–8

D

Datagram socket,
  creating a, 3–2
  using, 1–3
Data Link Interface,
  definition of, 1–1
Data segmentation,
  providing, 1–2, 3–1
Data Transfer,
  using DLI program, 3–7
Destination Service Access Point,
  definition of, 2–7
DLI address family,
  specifying the, 3–2
DLI program,
  transferring data, 3–7
  writing, 3–1
DLI protocol module,
  specifying the, 3–2
DLI Services,
  examples of, 1–1
dli_802_3_conn subroutine,
  example, B–17
  using, 2–2
dli_econn subroutine,
  example, B–5
  using, 2–2
Domain,
  specifying the, 2–1
DSAP,
  definition of, 2–7

E

Error recovery,
  providing, 1–2, 3–1
Ethernet,
  accessing, 1–2
  address, 1–2
  multiple users, 1–2
  transmitting messages on, 1–2
Ethernet frame structure,
  example of, 2–3
  function of, 2–4
  specifying target node information, 2–3
Ethernet substructure,
  filling the, 3–5
  frame structure, 2–3
  sending and receiving, 2–2
  values, 2–4
Exchange Identification,
  definition of, 2–8
  function of, 2–8

F

Flow control,
  providing, 1–2, 3–1
Frame format,
  802, 1–3
  description of 8802-3, 2–5
  Ethernet, 1–3, 2–3
  example of 8802-3, 2–6
  processing 8802-3, 2–6
  standard, 1–3
Frames,
  building, 1–2

G

Guaranteed delivery,
  providing, 1–2

H

Hardware support,
  examples of, 1–2
High-Level services,
  providing, 1–2, 3–1

I

I\O control flags,
  functions of, 2–4

L

LLC,
  sublayer of DLI, 2–6
LLC Protocol Data Unit Test,
  definition of, 2–8
  function of, 2–8
Logical Link Control (LLC),
  definition, 1–3
  sublayer of DLI, 2–6

M

Multicast address,
  using, 1–3, 2–4
Multiple users,
  on Ethernet, 1–2

N

Network device,
  specifying the, 2–1

P

Packet routing,
  providing, 1–2
Physical address,
  using, 1–3, 2–4
Prerequisites,
  for DLI programming, 1–2
Privileges,
  superuser, 1–2
Protocol type,
  definition, 2–4

R

recvfrom system call,
  using, 2–4

S

sendto system call,
  using, 2–1, 2–4
Service class,
  definition of, 2–6
  values, 2–6
Services,
  providing high-level, 3–1
SNAP_SAP,
  using, 2–7
sockaddr_dl data structure,
  explanation of, 2–1
  filling in, 2–1
  function of, 2–1
  using system calls to fill, 2–1
Socket,
  binding the, 3–4
  creating a, 3–2
  deactivating, 3–7
Socket address data structure,
  explanation of, 2–1
  filling in, 2–1, 3–2
  function of, 2–1
  using system calls to fill, 2–1
Socket options,
Socket options, (Cont.)
  setting, 3–3
socket system call,
  using, 3–2
SSAP,
  definition of, 2–7
Standard frame formats,
  802, 1–3
  Ethernet, 1–3
Subroutines,
  using, 2–2
802 substructure,
  filling in, 2–1
  sending and receiving, 2–2
8802-3 substructure,
  filling the, 3–6
Substructures,
  8802-3, 2–5
  Ethernet frame structure, 2–3
  filling in, 2–1
  sending and receiving, 2–2
8802-3 substructure values,
  control field, 2–8
  destination service access point, 2–7
  exchange identification, 2–8
  LLC Protocol Data Unit Test, 2–8
  Service class, 2–6
  source service access point, 2–7
  target node physical address, 2–6
  Unnumbered Information Command, 2–9
  XID, 2–8
System calls,
  bind, 3–2
  calling sequence, 3–2
  close, 3–2
  read, 3–2
  recv, 3–2
  recvfrom, 3–2
  send, 3–2
  sendto, 3–2
  socket, 3–2
  specifying values with, 2–1
  summary of, 3–2
  used to transfer data, 3–7
  using, 3–1
  write, 3–2

T

Target node,
  specifying information, 2–3
Target node physical address,
  definition of, 2–4, 2–6
  specifying, 2–4

U

Unnumbered Information Command,
  definition of, 2–9
  function of, 2–9

X

XID,
  definition of, 2–8
  function of, 2–8

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-234-1998 using a 1200- or 2400-baud modem from anywhere in the USA, Canada, or Puerto Rico. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local Digital Subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	————	Local Digital subsidiary or approved distributor
Internal*	————	SSB Order Processing - WMO/E15 *or* Software Supply Business Digital Equipment Corporation Westminster, Massachusetts 01473

* For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Reader's Comments

ULTRIX
Guide to the Data Link Interface
AA-PBKZA-TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

Please rate this manual:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

What would you like to see more/less of? ______________________________

What do you like best about this manual? ______________________________

What do you like least about this manual? ______________________________

Please list errors you have found in this manual:

Page	Description

Additional comments or suggestions to improve this manual:

____

What version of the software described by this manual are you using? __________

Name/Title ______________________ Dept. __________

Company ______________________ Date __________

Mailing Address ______________________

______________ Email ______________ Phone __________

Do Not Tear – Fold Here and Tape

digital™

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST–CLASS MAIL PERMIT NO. 33 MAYNARD MA

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
OPEN SOFTWARE PUBLICATIONS MANAGER
ZKO3–2/Z04
110 SPIT BROOK ROAD
NASHUA NH 03062–9987

Do Not Tear – Fold Here

Cut
Along
Dotted
Line