Open Transport 802.2 Developer Note

PRELIMINARY
Revision 1.1b14
1/12/96

Table of Contents

Revision History

01/12/96	Updated
11/15/95	Updated and added Raw Mode
7/18/94	Updated for 1.0d13 build
4/29/94	First compiled

Related Documents

Apple Shared Library Manager Developer's Guide, by ESD Publications, October 4, 1993, Apple Computer, Inc.

Open Transport Client Developer Note

802.2 for Open Transport

This document describes how client applications may directly use Ethernet or TokenRing endpoints. This document should be used along with the *Open Transport Client Developer Note* , which describes general information about Open Transport endpoint libraries.

This document describes 802.2, Ethernet and IPX address formats and options specific to link level endpoints.

Limitations

- Only Type 1(Unacknowleged connectionless) LLC's are supported.
- There is currenly no provision for attaching to 802.2 Group SAPs.

Technical Specifications

This section describes the address formats that are used in the Open Transport Endpoint functions (like *SndUData()*, *Bind()*, etc.) for IEEE 802.2 link layers.

The file OpenTptLinks.h contains the declarations of the necessary constants and data structures needed. This file may be included by both 'C' and 'C++' source files.

Other Ethernet Protocols

In addition to the 802.2 packets, an Ethernet endpoint can handle Ethernet, and Novell "Raw 802.3"(IPX).

The type of packets the driver will handle is specified at bind time by the SAP passed to the *Bind()* call.

Address Formats

Addresses are either 10 or 15 bytes long. They have the following format:

2-byte	address family (AF_8022)
6-byte	hardware address
2-byte	SAP
5-byte	SNAP (only if SAP = 0xAA)

The structure T8022Address defined in OpenTptLinks.h can be used to access the fields in an 802.2 address.

The fields

The first field in an 802.2 address is a two byte address family specifier. This must always be set to AF_8022.

The next field is the 6 byte hardware address. An address of all ones is the broadcast address. For Ethernet, a multicast address is a non-broadcast address that has a one in the low bit of the first byte. For Token Ring, a multicast address is a non-broadcast address with ones in the two high bits of the first byte.

The third field is the SAP (Service Access Point). The values this field can take depend on the underlying protocol:

802.2 — For the 802.2 protocol, this field must be an even number in the range 0x00 to 0xFE, and corresponds to the 802.2 SAP.

Ethernet — For the classic Ethernet protocol, this field must be in the range 0x5DD to 0xFFFF, and corresponds to the protocol type.

IPX — For IPX, this field should be 0xFF.

Note that SAPs in the range 0x100 to 0x5DC, and odd valued SAP's less then 0xFE are illegal.

If the value of the SAP is 0xAA, then it must be followed by a 5 byte SNAP.

Using the 802.2 Endpoints

The 802.2 endpoints are connectionless datagram endpoints. They do not support any of the connection-oriented calls, nor any of the transaction calls.

An Ethernet endpoint can be created using *OpenEndpoint()* passing the Ethernet identification string constant *kEnetName*. Similarly, a Token Ring endpoint can be created by passing the string constant *kTokenRingName* to `OTpenEndpoint()`. In either of these cases, the driver will be opened on the default slot (*i.e.*, the built in Ethernet or the card in the lowest numbered slot);

Endpoint Information

Binding

The bind operation associates an endpoint with an protocol address.

The hardware address field of the 802.2 address passed into the bind must be all zeros. If the bind completes successfully, the hardware address field of the returned address will contain the actual hardware address.

See the *Open Transport Client Developer Note* , for details on the specific calls to *Bind()*.

Sending data

When sending a packet, it is generally only necessary to specify the hardware address of the destination, unless it is being sent to a different SAP than that to which the sending endpoint is bound. In the latter case, the destination address must be fully specified.

Options

The 802.2 endpoints support four options: OPT_ADDMCAST, OPT_DELMCAST, OPT_SETRAWMODE, and OPT_SETPROMISCUOUS.

The OPT_ADDMCAST Option

The OPT_ADDMCAST option is used to enable a multicast address. The value must be a valid 6-byte multicast address. Only one multicast address can be enabled per *OptionManagement()* call. All added multicast addresses are use-counted so that deleting a multicast address does not really remove it until the last client that added it has removed it.

The OPT_DELMCAST Option

The OPT_DELMCAST option is used to disable a multicast address. The value must be a valid 6-byte multicast address that was enabled by a prior OPT_DELMCAST option. Only one multicast address can be disabled per *OptionManagement()* call. All added multicast addresses are use-counted so that deleting a multicast address does not really remove it until the last client that added it has removed it.

The OPT_SETRAWMODE Option

In normal use, the endpoint's client sends and receives only the data portion of the packet. In some circumstances the client may wish to send and receive the entire packet including the MAC and protocol headers. The OPT_SETRAMMODE option allows the client to do this. The option takes a 4-byte value which should be *kOTRawRcvOn* or *kOTRawRcvOnWithTimeStamp* to enable raw mode or *kOTRawRcvOff* to disable it.

If raw mode is enabled, packets are delivered to the client with the headers intact. Packets from the client are delivered to the transport unmodified, except that an 802.3 length field of zero will be replaced with the actual data length.

If *kOTRawRcvOnWithTimeStamp* is uses as the option value, then all incoming packets will be preceded by a 64-bit timestamp (obtained from the `OTGetTimeStamp` function).

The OPT_SETPROMISCUOUS Option

An endpoint normally sends and receives data only to a given SAP or SNAP. In some cases, it is desired to receive all of the traffic on the wire. The OPT_SETPROMISCUOUS option allows this. The value of the option is a 32-bit entity, indicating the desired level of packet reception:

The DLPI specification defines three levels of promiscuous mode: DL_PROMISC_PHYS, DL_PROMISC_SAP and DL_PROMISC_MULTI. The specification is notably vague as to exactly what these levels mean. Working with other Streams developers, we have come up with the following definitions

DL_PROMISC_PHYS

If the DLPI provider is in DL_UNBOUND state, the DLPI user receives all traffic on the wire regardless of MAC address or SAP.

If the DLPI provider is in DL_IDLE state, the DLPI user receives all traffic on the wire destined for the bound SAP(s), regardless of MAC address.

DL_PROMISC_SAP

If the DLPI provider is in DL_UNBOUND state, the DLPI user receives all traffic destined for this interface (physical address match, multicast address match or broadcast address) which match any SAP bound by any DLPI user of this interface.

If the DLPI provider is in DL_IDLE state, the DLPI user receives all traffic destined for this interface (physical address match, multicast address match or broadcast address) and which match the bound SAP.

DL_PROMISC_MULTI

If the DLPI provider is in DL_UNBOUND state, the DLPI user receives all multicast packets on the wire, regardless of SAP.

If the DLPI provider is in DL_IDLE state, the DLPI user receives all multicast packets on the wire destined for the bound SAP(s).

DL_PROMISC_OFF

Turn off promiscuous mode.

Index