ODI Specification Supplement: Frame Types and Protocol IDs

ODI Specification Version 4
Document Version 1.00
Part Number: 107–000055–001
24 November 1993

Disclaimer

Novell, Inc. makes no representations or warranties with respect to the contents or use of this manual, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.

Novell, Inc.
122 East 1700 South
Provo, Utah 84606

Trademarks

Novell has made every effort to supply trademark information about company names, products, and services mentioned in this document. Trademarks indicated below were derived from various sources.

Novell and NetWare are registered trademarks of Novell, Inc.

CHSM, C language Hardware Specific Module, CMSM, C language Media Support Module, CTSM, C language Topology Specific Module, Internetwork Packet Exchange, ODI, Open Data-Link Interface, LSL, Link Support Layer, MLID, Multiple Link Interface Driver, MLI, Multiple Link Interface, MPI, Multiple Protocol Interface, MSM, Media Support Module, TSM, Topology Support Module, HSM, Hardware Support Module, RX-Net, NE1000, NE2000, NE/2, NE2-32, and NTR2000 are trademarks of Novell, Inc.

Table Of Contents

List of Tables

List of Figures

Overview

This supplement provides information regarding the frame types currently supported by NetWare. This supplement illustrates each of the frame types and provides examples of some corresponding Protocol ID numbers. It also discusses 802.2 Type II frame support.

This supplement contains valuable reference material.

Note Byte positions are zero-based. ▲

Defined Frame Types and Protocol IDs

The following table shows the frame types we have currently defined and contains examples of their corresponding Protocol ID numbers for IPX and other protocols.

Table 1
Frame Types and Protocol IDs

Frame Types and Protocol IDs				
Frame ID	**Frame Type String**	**Protocol**	**Protocol ID**	**Description**
0	VIRTUAL_LAN	IPX/SPX	0	For use where no Frame ID/MAC envelope is necessary
1	LOCALTALK	AppleTalk	0	The Apple LocalTalk frame
2	ETHERNET_II	IPX/SPX	8137h	Ethernet using a DEC Ethernet II envelope
2	ETHERNET_II	XNS	600h	Ethernet using a DEC Ethernet II envelope
2	ETHERNET_II	AARP	80F3h	Ethernet using a DEC Ethernet II envelope
2	ETHERNET_II	AppleTalk	809Bh	Ethernet using a DEC Ethernet II envelope
2	ETHERNET_II	ARP	806h	Ethernet using a DEC Ethernet II envelope
2	ETHERNET_II	RARP	8035h	Ethernet using a DEC Ethernet II envelope
2	ETHERNET_II	IP	800h	Ethernet using a DEC Ethernet II envelope
3	ETHERNET_802.2	IPX/SPX	E0h	Ethernet (802.3) using an 802.2 envelope
3	ETHERNET_802.2	RPL	FCh	Ethernet (802.3) using an 802.2 envelope
3	ETHERNET_802.2	SNA	04h	Ethernet (802.3) using an 802.2 envelope
3	ETHERNET_802.2	NetBIOS	F0h	Ethernet (802.3) using an 802.2 envelope
4	Token-Ring	IPX/SPX	E0h	Token-Ring (802.5) using an 802.2 envelope
4	Token-Ring	RPL	FCh	Token-Ring (802.5) using an 802.2 envelope
4	Token-Ring	SNA	04h	Token-Ring (802.5) using an 802.2 envelope

Frame Types and Protocol IDs *(continued)*

Frame ID	Frame Type String	Protocol	Protocol ID	Description
4	Token-Ring	NetBIOS	F0h	Token-Ring (802.5) using an 802.2 envelope
5	ETHERNET_802.3	IPX/SPX	00h	IPX 802.3 raw encapsulation
6	802.4	IPX/SPX	N/A	Token-passing bus envelope
7	RESERVED			Reserved for future use.
8	GNET	IPX/SPX	E0h	Gateway's G/Net frame envelope
9	PRONET-10	IPX/SPX	N/A	Proteon's ProNET I/O frame envelope
10	ETHERNET_SNAP	IPX/SPX	8137h	Ethernet (802.3) using an 802.2 envelope with SNAP
10	ETHERNET_SNAP	XNS	600h	Ethernet (802.3) using an 802.2 envelope with SNAP
10	ETHERNET_SNAP	AARP	80F3h	Ethernet (802.3) using an 802.2 envelope with SNAP
10	ETHERNET_SNAP	AppleTalk	80007809Bh	Ethernet (802.3) using an 802.2 envelope with SNAP
10	ETHERNET_SNAP	ARP	806h	Ethernet (802.3) using an 802.2 envelope with SNAP
10	ETHERNET_SNAP	RARP	8035h	Ethernet (802.3) using an 802.2 envelope with SNAP
10	ETHERNET_SNAP	IP	800h	Ethernet (802.3) using an 802.2 envelope with SNAP
11	Token-Ring_SNAP	IPX/SPX	8137h	Token-Ring (802.5) using an 802.2 envelope with SNAP
12	LANPAC_II	IPX/SPX	N/A	Racore's frame envelope
13	ISDN	IPX/SPX	N/A	Integrated Services Digital Network (not available)
14	NOVELL_RX-NET	IPX/SPX	FAh	Novell's RX-Net envelope
17	OMNINET/4	IPX/SPX	N/A	Corvus's frame envelope
18	3270_COAXA	IPX/SPX	N/A	Harris Adacom's frame envelope
19	IP	IPX/SPX	N/A	IP Tunnel frame envelope
20	FDDI_802.2	IPX/SPX	E0h	FDDI using an 802.2 envelope
20	FDDI_802.2	RPL	FCh	FDDI using an 802.2 envelope
20	FDDI_802.2	SNA	04h	FDDI using an 802.2 envelope
21	IVDLAN_802.9	IPX/SPX	N/A	Commtex, Inc.'s frame envelope

Frame Types and Protocol IDs *(continued)*

Frame ID	Frame Type String	Protocol	Protocol ID	Description
22	DATACO_OSI	IPX/SPX	N/A	Dataco's frame envelope
23	FDDI_SNAP	IPX/SPX	8137h	FDDI using 802.2 with a SNAP envelope
23	FDDI_SNAP	XNS	600h	FDDI using 802.2 with a SNAP envelope
23	FDDI_SNAP	AARP	80F3h	FDDI using 802.2 with a SNAP envelope
23	FDDI_SNAP	AppleTalk	80007809Bh	FDDI using 802.2 with a SNAP envelope
23	FDDI_SNAP	ARP	806h	FDDI using 802.2 with a SNAP envelope
23	FDDI_SNAP	RARP	8035h	FDDI using 802.2 with a SNAP envelope
23	FDDI_SNAP	IP	800h	FDDI using 802.2 with a SNAP envelope
24	IBM_SDLC		Unknown	Novell frame type
25	PCO_FDDITP		Unknown	PC Office frame type
26	WAIDNET		Unknown	Hypercommunications frame type
27	SLIP		Unknown	Novell frame type
28	PPP		Unknown	Novell frame typye
29	RANGELAN		Unknown	Proxim frame type
30	X.25		Unknown	Novell frame type
31	Frame_Relay		Unknown	Novell frame type
32	IWI_BUS-NET_SNAP		Unknown	Integrated Workstations frame type
33	SNA_LINKS		Unknown	Novell frame type
34	WAN_Client_LAN		Unknown	Novell frame type

The 802.2 Frame Type

This section describes 802.2 Type I and Type II frames. The 802.2 frame is the default frame type for MLIDs and protocol stacks.

802.2 Type I and Type II

802.2 packets may be Type I (one control byte) or Type II (two control bytes). The *ProtocolID* and the *DriverWorkspace* fields of the ECB contain the information that distinguishes 802.2 Type I and Type II packets.

Setting the ECB *ProtocolID* Field

In transmitted packets, the *ProtocolID* field of the ECB contains the information that is necessary to build the header for the correct 802.2 packet type. The table below summarizes the relationship between the *ProtocolID* contents and the header.

Table 2
***ProtocolID* Field Contents and the 802.2 Packet Header**

***ProtocolID* Field Contents and the 802.2 Packet Header**										
	***ProtocolID* Field Bytes**						**802.2 Header Bytes**			
Format Type	**0**	**1**	**2**	**3**	**4**	**5**	**1**	**2**	**3**	**4**
UI-Format	00	00	00	00	00	DSAP	DSAP	SSAP	03	
U-Format	02	00	00	DSAP	SSAP	Ctrl0	DSAP	SSAP	Ctrl0	
S-Format/ I-Format	03	00	DSAP	SSAP	Ctrl0	Ctrl1	DSAP	SSAP	Ctrl0	Ctrl1

Note If the packet is in UI-Format, the packet header *Ctrl0* field equals 03 and SSAP is assumed to equal DSAP. (For a more detailed description of the *DSAP, SSAP, Ctrl0,* and *Ctrl1,* fields, see *IEEE Std 802.2, 1989.* Reference number ISO 8802-2 : 1989 [E].)

In received packets, the *Ctrl0* field indicates whether the packet is a Type I or Type II packet. The following table shows the possible *Ctrl0* values and the actions that must be taken in filling in the ECB.

Table 3
***Ctrl0* Bits**

***Ctrl0* Bits**		
***Ctrl0* Bits**		**Action**
1	**0**	
x	0	The packet has an 802.2 Type II header, set byte 1 of *DriverWorkSpace* to 2. Point *PacketOffset* past the *Ctrl1* field of the header. Set the *ProtocolID* field according to the I-format row in Table 2.
0	1	The packet has an 802.2 Type II header, set byte 1 of *DriverWorkSpace* to 2. Point *PacketOffset* past the *Ctrl1* field of the header. Set the *ProtocolID* field according to the S-format row in Table 2.
1	1	The packet has an 802.2 Type I header, set byte 1 of *DriverWorkSpace* to 1. Point *PacketOffset* past the *Ctrl0* field of the header. If the packet header *Ctrl0* field contains a 03 (that is 00000011), set the *ProtocolID* field according to the UI-Format row in Table 2. Otherwise, set the *ProtocolID* field according to the U-Format row in Table 2.

MLID Support of 802.2 Frames

When an MLID that supports 802.2 frames transmits a packet, the MLID must parse the Protocol ID in order to determine the 802.2 header format. The header format must be consistent with the definitions given above. If the 802.2 header is not explicitly specified on a transmit operation (Byte 0 = 0), the MLID sets the SSAP equal to the given DSAP and the Ctrl0 byte to 03h (802.2 Type I). When the MLID receives a frame, the MLID uses the DSAP as the Protocol ID.

Frame Types

This section defines each of the frame types supported by NetWare. The following frame types should be supported for the standard topologies when running on the NetWare 3.*x* and 4.*x* operating systems. Default frame types are shown in **bold**. This list is for the standardized media types of Ethernet, Token-Ring, FDDI, and RX-Net. Proprietary frame types are not included here.

Table 4
Supported Frame Types

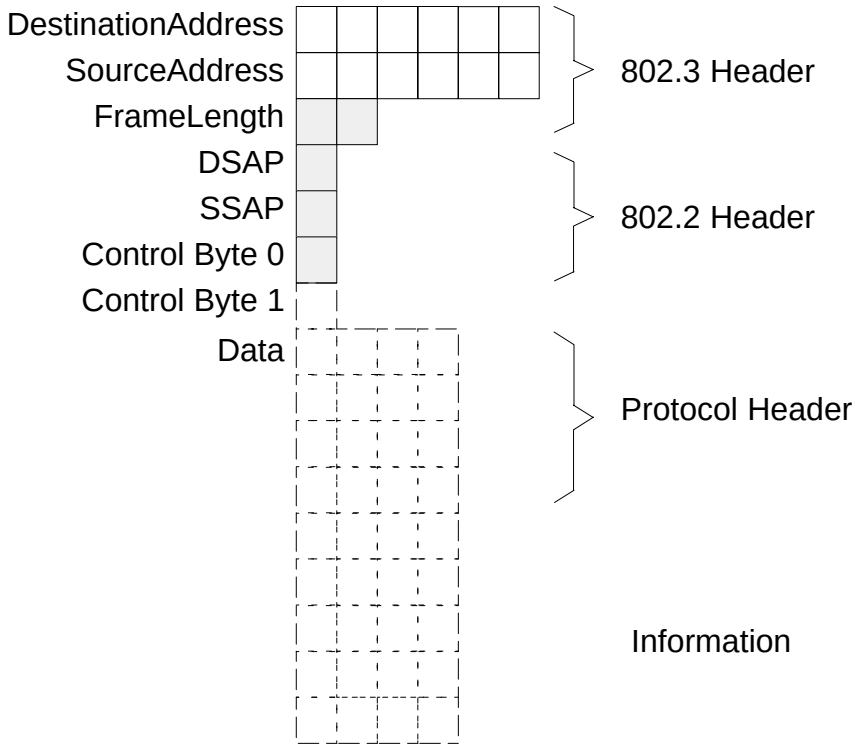
Supported Frame Types			
Ethernet	**Token-Ring**	**FDDI**	**RX–Net**
Ethernet 802.2	**Token-Ring 802.2**	**FDDI 802.2**	(short)
Ethernet SNAP	Token-Ring SNAP	FDDI SNAP	(long)
Ethernet II			(exception)

Ethernet 802.2

In Ethernet 802.2 packets, the word-length *FrameLength* field (bytes 12 and 13) contains a value less than or equal to 1500 (decimal). If the following three bytes––*DSAP*, *SSAP*, and *Control Byte 0*––do *not* contain the values AAh, AAh, and 03h, respectively, the packet is Ethernet 802.2, Type I or Type II.

Note Ethernet MLIDs should default to the 802.2 frame type. ▲

Figure 1
Ethernet 802.2 Packet Type



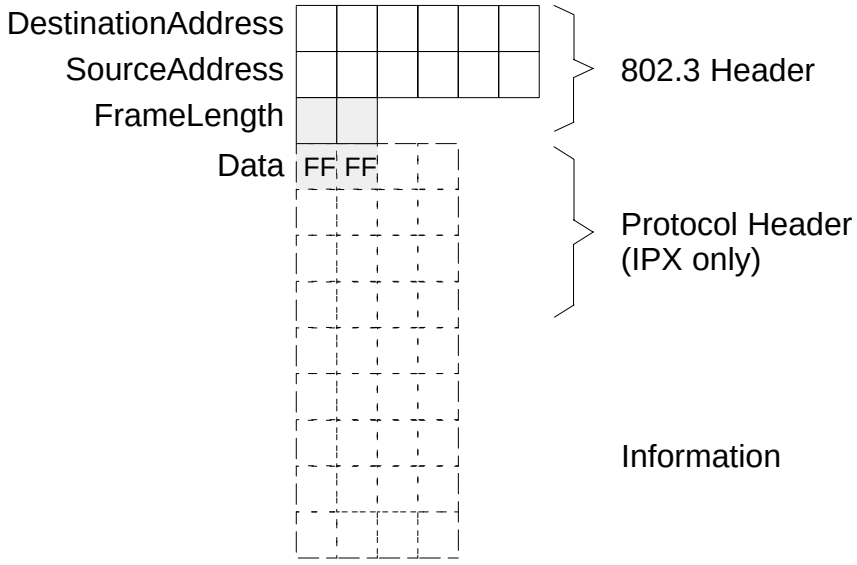
Bits 0 and 1 of the *ControlByte0* field indicate whether an 802.2 header is Type I or Type II. If both bit 0 and bit 1 are set, the header is 802.2 Type I. Any other combination (00, 01, 10) indicates an 802.2 Type II header.

Ethernet Raw 802.3

Important MLIDs written in C language do not support Ethernet Raw 802.3.

In Ethernet raw 802.3 packets, the word-length *FrameLength* field (bytes 12 and 13) contains a value less than or equal to 1500 (decimal). In addition, the first two bytes of the *Data* area (bytes 14 and 15 of the packet) must contain the values FFh and FFh.

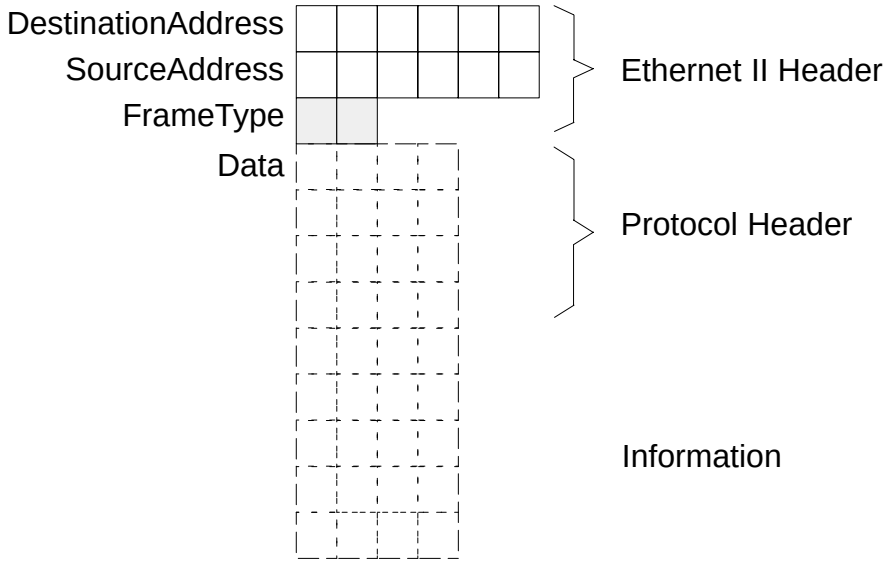
Figure 2
Ethernet Raw 802.3 Packet Type



Ethernet II

In Ethernet II packets, the word-length field *FrameType* (bytes 12 and 13) contains a value greater than 1500 (decimal).

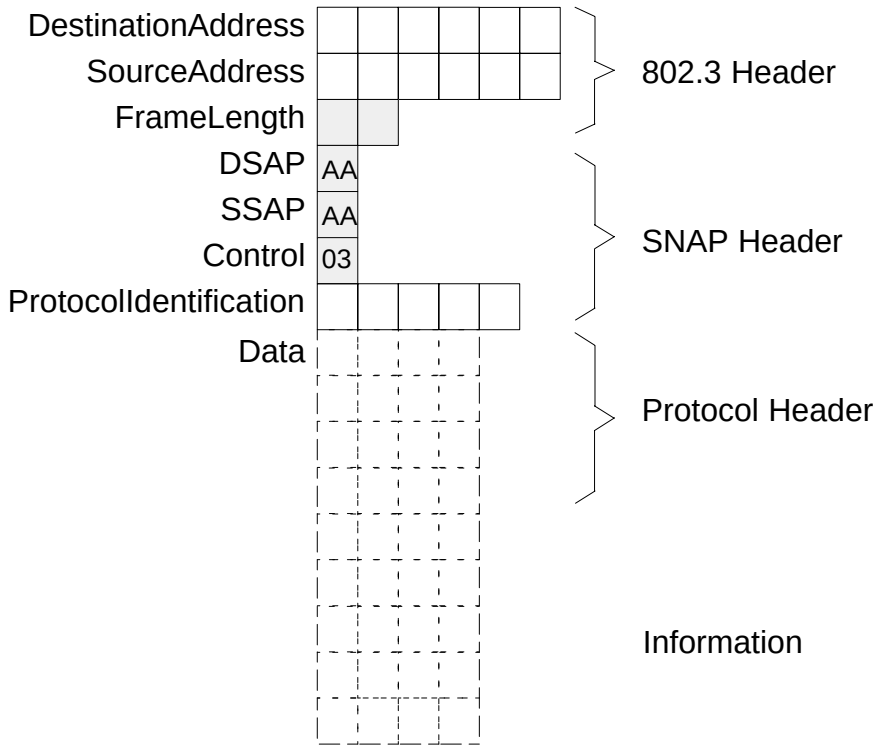
Figure 3
Ethernet II Packet Type



Ethernet SNAP

In Ethernet SNAP packets, the word-length *FrameLength* field (bytes 12 and 13) contains a value less than or equal to 1500 (decimal). In addition, the following three bytes––*DSAP*, *SSAP*, and *Control*––contain the values AAh, AAh, and 03h, respectively.

Figure 4
Ethernet SNAP Packet Type

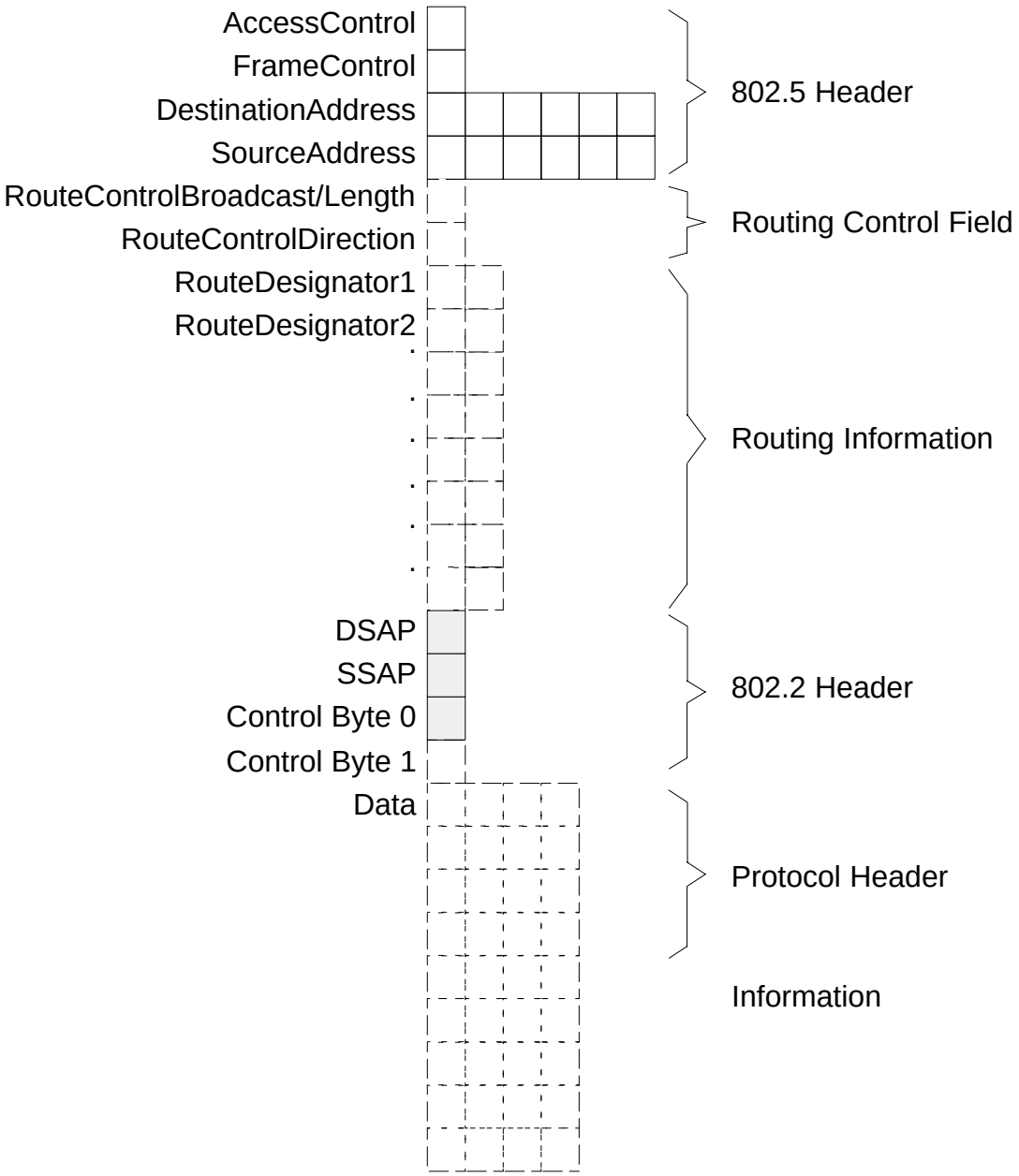


Token-Ring 802.2

In Token-Ring packets, the *DSAP*, *SSAP*, and *Control Byte 0* fields of the packet will *not* be AAh, AAh, and 03h, respectively. The packet is 802.2, Type I or II.

Note Token-Ring MLIDs should default to the 802.2 frame type. ▲

Figure 5
Token-Ring 802.2 Packet Type



Token-Ring 802.2 *continued*

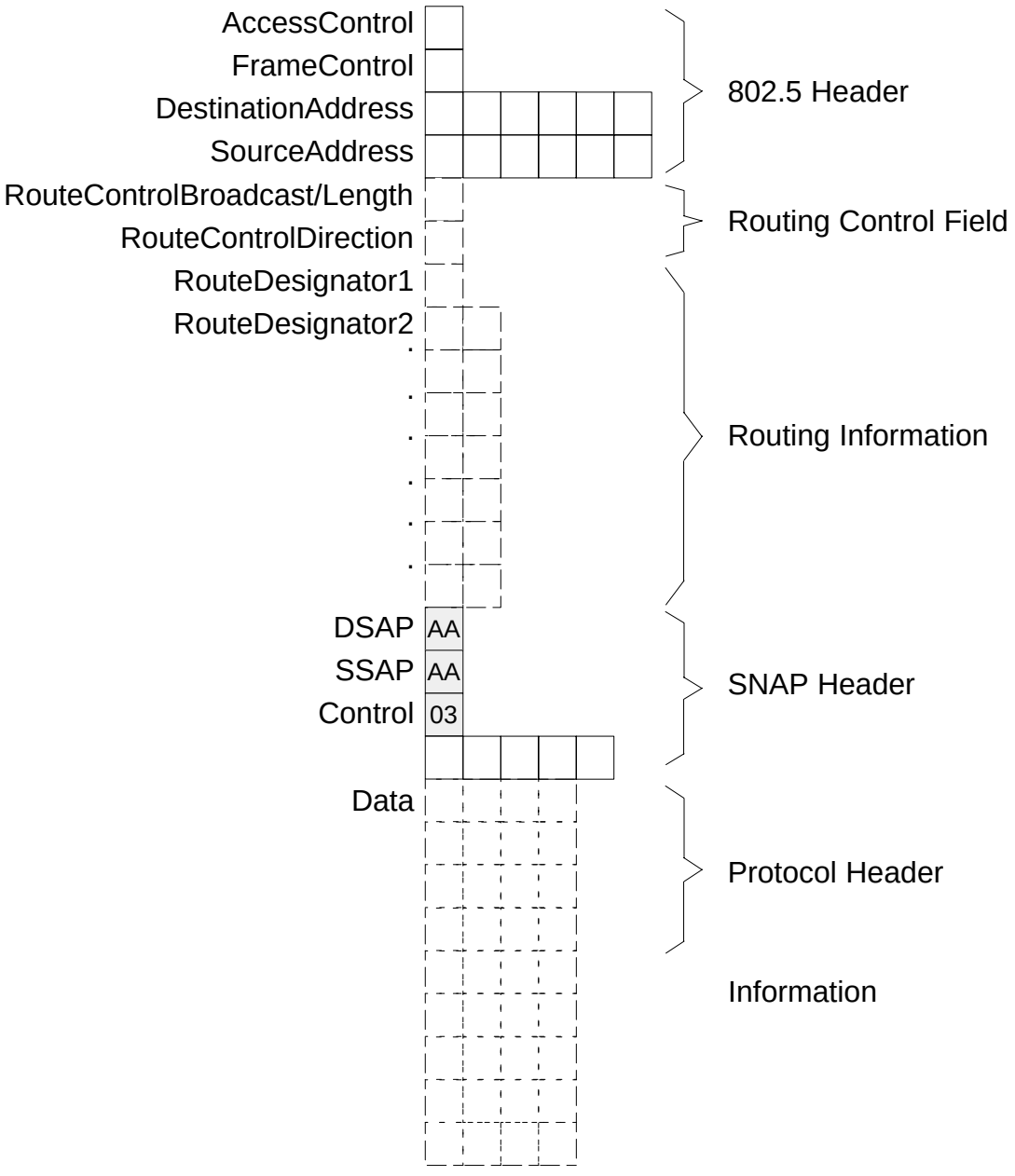
MLIDs previously supported only the 802.2 Type I frame. When a protocol stack received a Type II packet, the stack considered the second control byte of the header as part of the

packet's data. Bits 0 and 1 of the *ControlByte0* field indicate whether an 802.2 header is Type I or Type II. If both bit 0 and bit 1 are set, the header is 802.2 Type I. Any other combination (00, 01, 10) indicates an 802.2 Type II header.

Token-Ring SNAP

In Token-Ring SNAP packets, the *DSAP, SSAP,* and *Control* fields of the packet will be AAh, AAh, and 03h respectively.

Figure 6
Token-Ring SNAP Packet Type

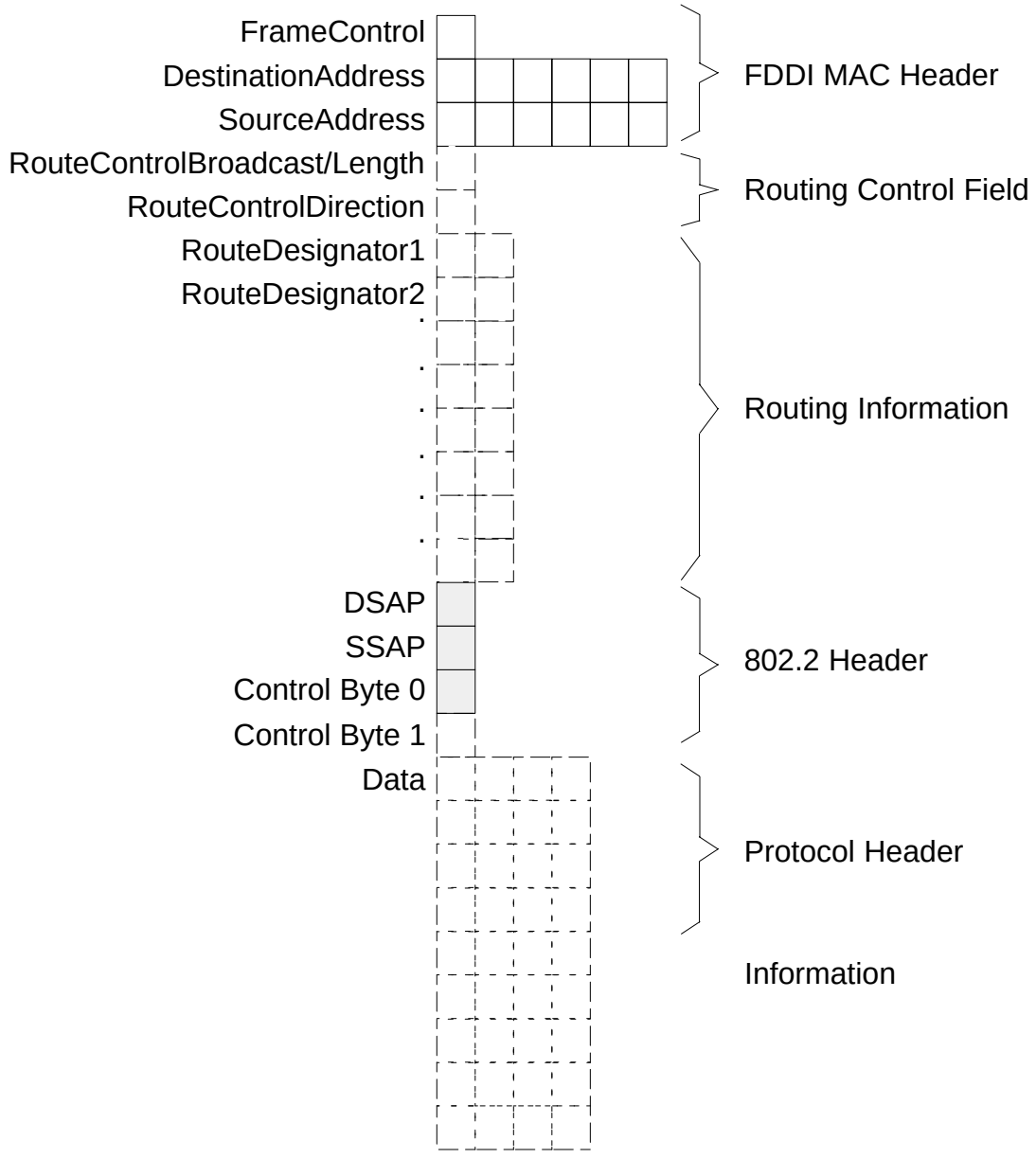


FDDI 802.2

In FDDI packets, the *DSAP*, *SSAP*, and *Control Byte 0* fields of the packet will *not* be AAh, AAh, and 03h, respectively. The packet is 802.2 Type I or II.

Note FDDI MLIDs should default to the 802.2 frame type. ▲

Figure 7
FDDI 802.2 Packet Type



MLIDs previously supported only the 802.2 Type I frame. When a protocol stack received a Type II packet, the stack considered the second control byte of the header as part of the

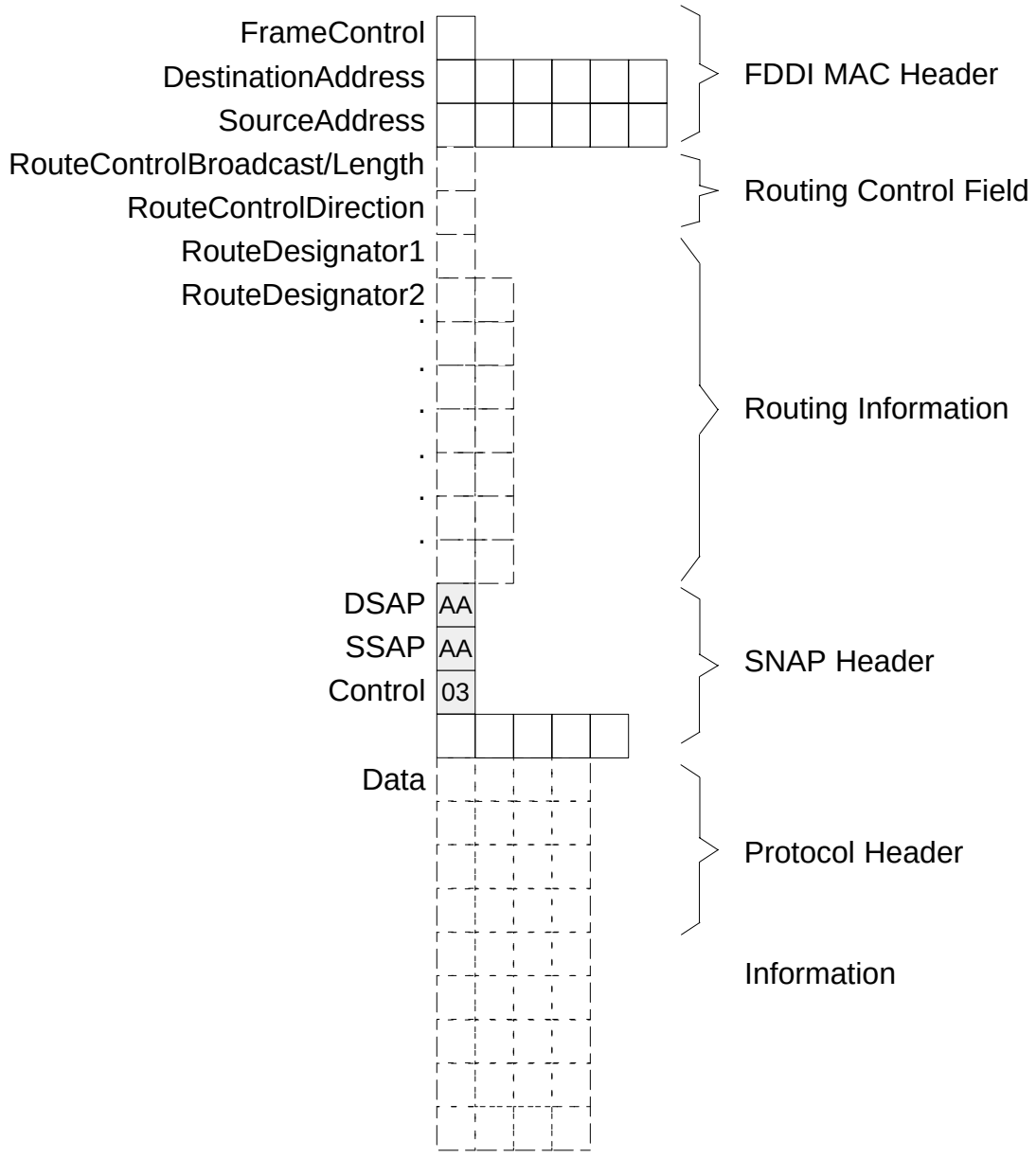
FDDI 802.2 *continued*

packet's data. Bits 0 and 1 of the *ControlByte0* field indicate whether an 802.2 header is Type I or Type II. If both bit 0 and bit 1 are set, the header is 802.2 Type I. Any other combination (00, 01, 10) indicates an 802.2 Type II header.

FDDI SNAP

In FDDI SNAP packets, the *DSAP*, *SSAP*, and *Control* fields of the packet will be AAh, AAh, and 03h respectively.

Figure 8
FDDI SNAP Packet Type



RX-Net

RX-Net drivers must recognize and handle three packet types:

- Short
- Long
- Exception

Novell RX-Net drivers do not use logical boards to handle these different packet types. Instead, they examine the packet to determine the type and fill out the RCB accordingly.

Each packet type has a 3-byte RX-Net header that includes the source and destination addresses, and an offset byte that points to the packet data and also indicates the data length. It is the length of the data, in bytes, that determines the packet type as shown below:

0 – 249 Short packet or fragment

250 – 252 Exception packet

253 – 504 Long packet or fragment

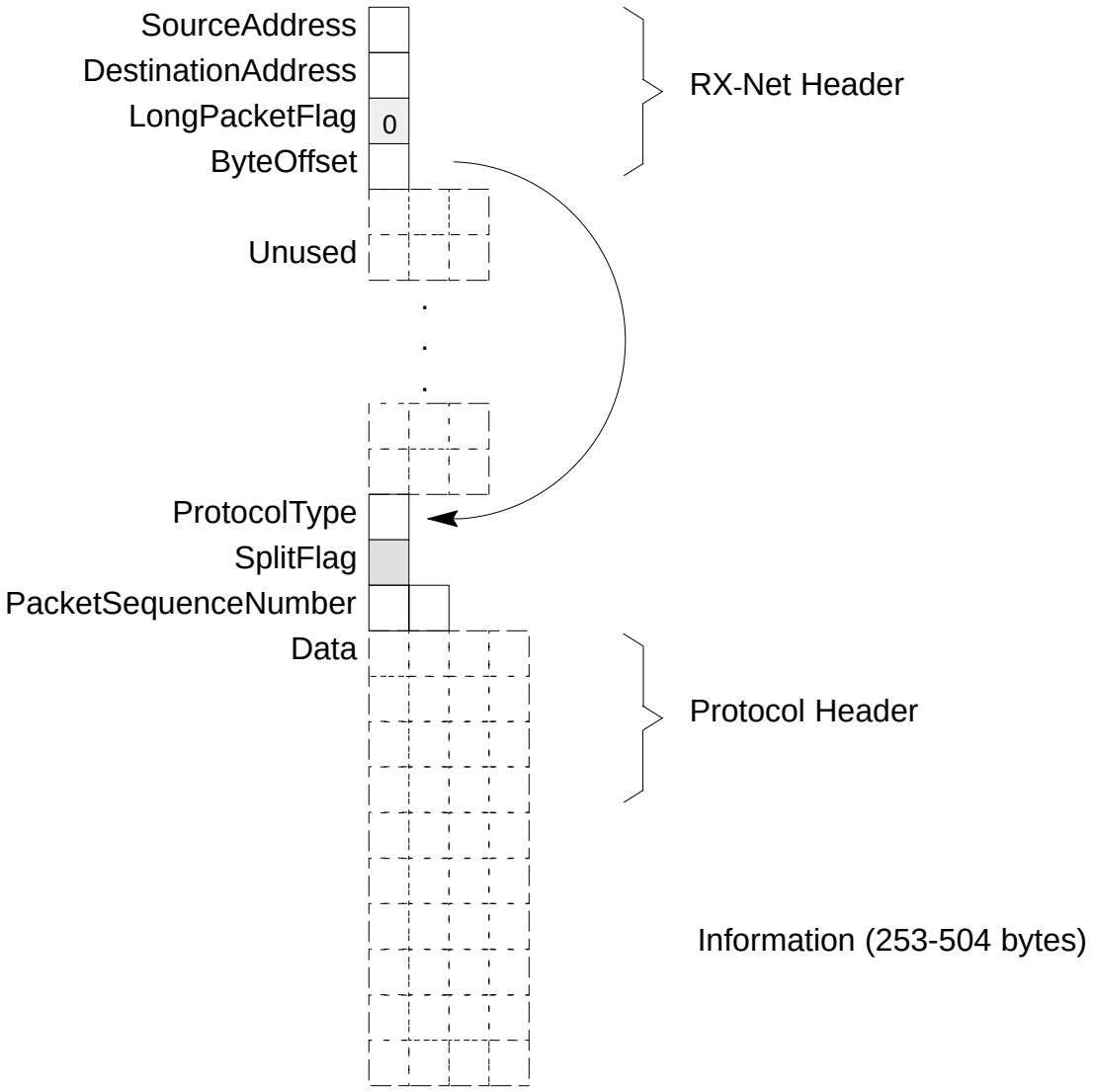
Each packet type also contains a *ProtocolType* byte that contains the unique number that Datapoint Corporation has assigned to the adapter manufacturer to identify the company's packets on the wire. For example, Novell packets use FAh, and AppleTalk packets use DDh.

RX-Net *continued*

Long Packet

In long packets, the third byte of the packet (*LongPacketFlag*) will be 00h, and the byte after the *ProtocolType* byte (*SplitFlag*) *will not* be FFh. This packet format is for a single packet or fragment containing 253 to 504 bytes of data. The fields are defined following the figure:

Figure 9
RX-Net Long Packet Type



RX-Net *continued*

Source Address. This is the single byte address of the sending board inserted by the RX-Net hardware.

Destination Address. This is the single-byte address of the destination card.

Long Packet Flag. This is a single byte of 00h indicating that this packet is a long packet.

Byte Offset. This is a single byte entry, the value of which is calculated as follows:

Byte Offset = 512 – (*N* + 4)

where *N* represents the number of data bytes, 4 represents the information bytes defined below, and 512 represents the length of the RX-Net long buffer.

Protocol Type. This is the single-byte type number issued by Datapoint Corporation.

Split Flag. This is a single byte entry identifying which fragment of a total data packet is contained within this packet. Refer to Figure 12 in the Split Flag description for more information on this byte.

Sequence Number. This is a word value (low-high) set equal to a counter kept by the sending machine. All fragments of a particular data packet must have the same sequence number. This number is incremented after all fragments of a data packet have been transmitted. When the sequence number has reached FFFFh, it should be incremented to 0000h.

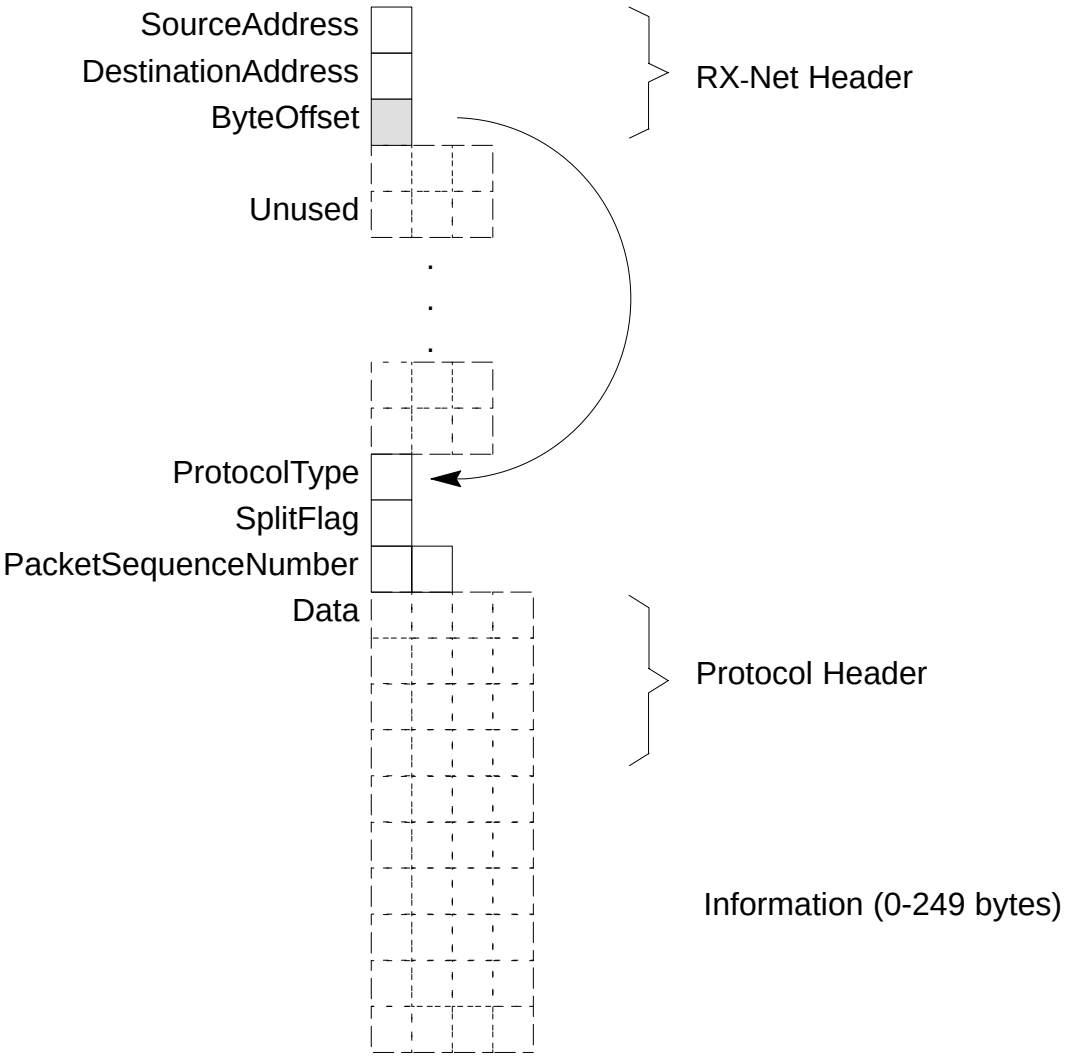
Data. This is the actual data.

RX-Net *continued*

Short Packet

In short packets, the third byte of the packet (*ByteOffset*) will contain a nonzero value. This packet format is for a single packet or fragment containing 0 to 249 bytes of data. The fields are defined following the figure.

Figure 10
RX-Net Short Packet Type



Source Address. This is the single-byte address of the sending board inserted by the RX-Net hardware.

Destination Address. This is the single-byte address of the destination board.

RX-Net *continued*

Byte Offset. This is a single byte entry, the value of which is calculated as follows:

Byte Offset = 256 – (*N* + 4)

where *N* represents the number of data bytes, 4 represents the information bytes defined below, and 256 represents the length of the RX-Net short buffer.

Protocol Type. This is the single-byte type number issued by Datapoint Corporation.

Split Flag. This is a single-byte entry identifying which fragment of a total data packet is contained within this packet. Refer to Figure 12 in the Split Flag description for more information on this byte.

Sequence Number. This is a word value (low-high) set equal to a counter kept by the sending machine. All fragments of a particular data packet must have the same sequence number. This number is incremented after all fragments of a data packet have been transmitted. When the sequence number has reached FFFFh, it should be incremented to 0000h.

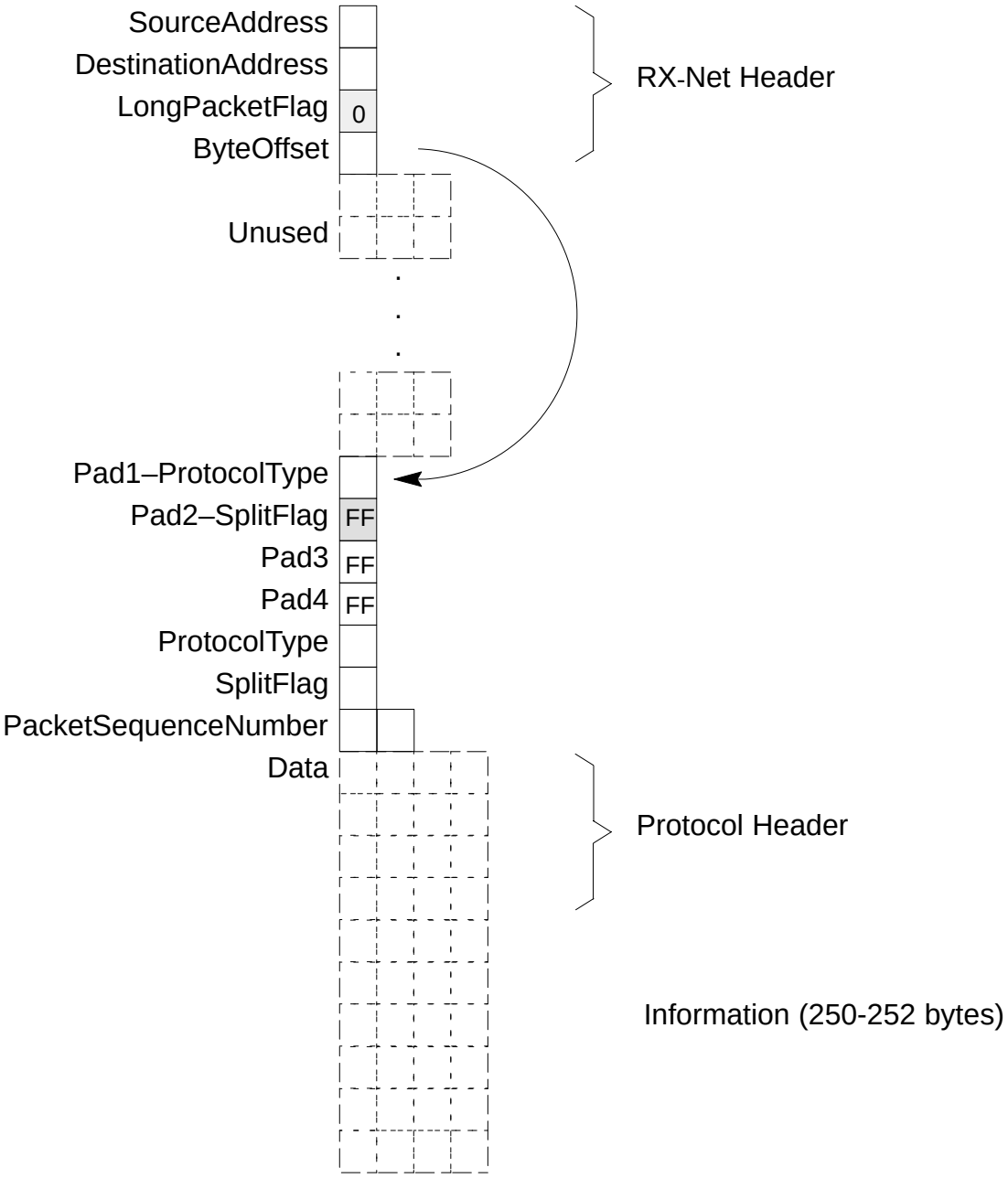
Data. This is the actual data.

RX-Net *continued*

Exception Packet

In exception packets, the third byte of the packet (*LongPacketFlag*) will be 00h, and the byte after the *ProtocolType* byte (*Pad2–SplitFlag*) will be FFh. The fields are defined following the figure.

Figure 11
RX-Net Exception Packet Type



RX-Net *continued*

This packet format is for a single packet or fragment containing 250 to 252 bytes of data. The format is similar to the long packet, but because of the single byte representing the byte offset, these lengths require that padding be added to the data packet.

Source Address. This is the single-byte address of the sending board inserted by the RX-Net hardware.

Destination Address. This is the single-byte address of the destination board.

Long Packet Flag. This is a single-byte of 00h indicating that this packet is a long packet.

Byte Offset. This is a single-byte entry, the value of which is calculated as follows:

Byte Offset = 512 – (*N* + 8)

where *N* represents the number of data bytes, 8 represents the bytes defined below including the 4 padding bytes, and 512 represents the length of the RX-Net long buffer.

Pad1–Protocol Type. This is the single-byte type number issued by Datapoint Corporation.

Pad2–Split Flag. This is a single byte entry, FFh, identifying this packet as an exception packet. Refer to Figure 12 in the Split Flag description for more information on this byte.

Pad3–Padding Byte. This is a single-byte of FFh.

Pad4–Padding Byte. This is a single-byte of FFh.

Protocol Type. This is the single-byte type number issued by Datapoint Corporation.

Split Flag. This is a single-byte entry identifying which fragment of a total data packet is contained within this packet (Refer to Figure 12).

Sequence Number. This is a word value (low-high) set equal to a counter kept by the sending machine. All fragments of a particular data packet must have the same sequence number. This number is incremented after all fragments of a data packet have been transmitted. When the word value has reached FFFFh, it should be incremented to 0000h.

Data. This is the actual data.

RX-Net *continued*

RX-Net Split Flag

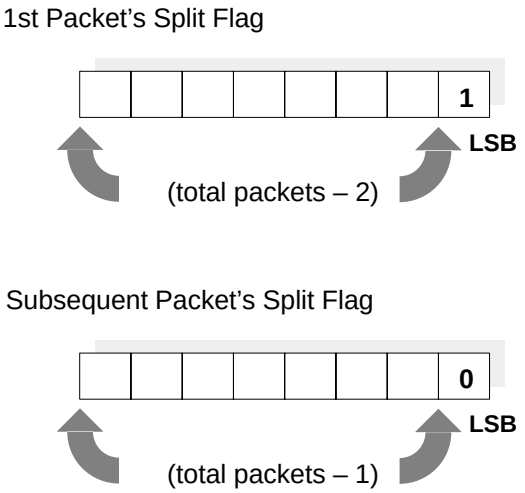
Splitting data into multiple packets allows data sizes to exceed the limit set by the physical network. The RX-Net *SplitFlag* provides a method for tracking and reassembling the packets that make up a frame.

The *SplitFlag* allows data to be split between multiple packets. This method allows a maximum of 120 fragments per frame, which yields a maximum of 60,480 bytes per frame:

(120 frags/frame) x (504 bytes/frags) = 60,480 bytes/frame

When data is fragmented, the *SplitFlag* byte contains two flags: the *more flag* and a *fragment number*. The *more flag* is the least significant bit. The seven most significant bits contain the *fragment number*. Figure 12 illustrates the *SplitFlag* byte.

Figure 12
RX-Net Split Flag



First Fragment. The first fragment of a split data frame has its *more flag* set to indicate that there are more fragments to follow. The *fragment number* indicates the total number of fragments minus 2.

Subsequent Fragments. The *more flag* is 0 for all remaining fragments. The *fragment number* indicates the current fragment number minus 1.

Example 1

A complete, unfragmented packet has a *SplitFlag* value of 00h.

Example 2

If packet data is split into two fragments, the *SplitFlag* values for each fragment are:

Fragment 1	0000000	1	= 01h
Fragment 2	0000001	0	= 02h

The first packet is prepared with the *SplitFlag* byte's *fragment number* set to 0000000 (two total fragments minus 2), and the *more flag* is set to indicate more fragments to come. The resulting *SplitFlag* value is 01h.

When the second packet is prepared, the *fragment number* is set to 0000001 (current fragment number minus 1), and the *more flag* is set to 0, indicating this is the second of two fragments. The resulting *SplitFlag* byte value is 02h.

Example 3

If packet data is split into four fragments, the *SplitFlag* byte values are:

Fragment 1	0000010	1	= 05h
Fragment 2	0000001	0	= 02h
Fragment 3	0000010	0	= 04h
Fragment 4	0000011	0	= 06h

The last fragment in a series has a *SplitFlag* value equal to the first fragment's *SplitFlag* byte value plus 1. The MLID can use this information to determine when the last fragment of data has arrived. Calculating the number of fragments that make a complete frame should not seriously affect the performance of the driver.

Note *SplitFlag* codes from F0h to FEh are reserved for future use. ▲

Index

Numbers

D

E

F

M

P

R

S

T