D

Media Frame Types

Frame and Protocol ID Table

The table below shows the media frame types currently defined by Novell and the corresponding Protocol ID number on IPX/SPX.

Frame ID	Frame Type String	Protocol ID on IPX/SPX	Description / Company
0	VIRTUAL_LAN	00h	For use when no Media ID / MAC envelope is necessary.
1	LOCALTALK	00h	The Apple LocalTalk media
2	ETHERNET_II	8137h	Ethernet using a DEC Ethernet II envelope
3	ETHERNET_802.2	E0h	Ethernet (802.3) using an 802.2 envelope
4	TOKEN-RING	E0h	Token-Ring (802.5) using an 802.2 envelope
5	ETHERNET_802.3	00h	IPX 802.3 raw encapsulation
6	802.4		Token-passing bus envelope
7	NOVELL_PCN2	1111h	Novell's IBM PC Network II envelope
8	GNET	E0h	Gateway's GNET media envelope
9	PRONET-10		Proteon's Pronet I/O media envelope
10	ETHERNET_SNAP	8137h	Ethernet (802.3) using an 802.2 envelope with SNAP
11	TOKEN-RING_SNAP	8137h	Token-Ring (802.5) using an 802.2 envelope with SNAP
12	LANPAC_II		Racore's media envelope
13	ISDN		Integrated Services Digital Network (Not available)
14	NOVELL_RX-NET	FAh	Novell's RX-Net envelope
15	IBM_PCN2_802.2	E0h	IBM PCN2 using 802.2 envelope
16	IBM_PCN2_SNAP	8137h	IBM PCN2 using 802.2 with SNAP envelope
17	OMNINET/4		Corvus frame envelope
18	3270_COAXA		Harris Adacom frame envelope
19	IP		IP Tunnel Frame envelope
20	FDDI_802.2	E0h	FDDI using an 802.2 envelope
21	IVDLAN_802.9		Commtex, Inc. frame envelope
22	DATACO_OSI		Dataco frame envelope
23	FDDI_SNAP	8137h	FDDI using 802.2 SNAP envelope
24	IBM_SDLC		SDLC tunnel envelope
25	PCO_FDDITP		PC Office frame envelope
26	WAIDNET		Hypercommunications
27	SLIP		Novell frame envelope
28	PPP		Novell frame envelope
29	RANGELAN		Proxim
30	X.25		X.25
31	Frame_Relay		Novell
32	IWI_BUS-NET_SNAP		Integrated Workstations
33	SNA_LINKS		Novell

Frame Types

This section provides frame related information and illustrates several of the different frame formats.

802.2 Type I and Type II

802.2 packets may be Type I (one control byte) or Type II (two control bytes). Previously 802.2 Type II packets were handled by placing the additional control byte in the first byte of the data field. This is no longer the case. The *ProtocolID* and the *DriverWorkspace* fields of the ECB now contain information required to distinguish 802.2 Type I and Type II packets.

Note: Drivers written using the MSM and TSM Modules do not need to be concerned with the 802.2 frame Type I and II situation. These support modules handle the identification on incoming packets and build the headers for outgoing packets.

Packet Transmission

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

ProtocolID Bytes						802.2 Header Bytes			
0	**1**	**2**	**3**	**4**	**5**	**1**	**2**	**3**	**4**
00	00	00	00	00	DSAP	DSAP	DSAP	03	
02	00	00	DSAP	SSAP	Ctrl0	DSAP	SSAP	Ctrl0	
03	00	DSAP	SSAP	Ctrl0	Ctrl1	DSAP	SSAP	Ctrl0	Ctrl1

Packet Reception

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 action that must be taken in filling in the receive ECB.

Ctrl0 Bits		Action
0	**1**	
0	0	The packet has an 802.2 Type II header, set byte 1 of *DriverWorkSpace* to 2. Make sure *PacketOffset* points past the Ctrl1 field of the header.
0	1	The packet has an 802.2 Type II header, set byte 1 of *DriverWorkSpace* to 2. Make sure *PacketOffset* points past the Ctrl1 field of the header.
1	0	The packet has an 802.2 Type II header, set byte 1 of *DriverWorkSpace* to 2. Make sure *PacketOffset* points past the Ctrl1 field of the header.
1	1	The packet has an 802.2 Type I header, set byte 1 of *DriverWorkSpace* to 1.

Default Frame Types

The following frame types should be supported for the standard topologies when running on the NetWare v4.x operating system. Default frame types are shown in **bold**. This list is for the standardized media types of Ethernet, Token-Ring, FDDI, PCN2, and RX-Net. Proprietary frame types are not included here.

Ethernet	Token-Ring	FDDI	PCN2	RX-Net
Ethernet 802.2 Ethernet Snap Ethernet 802.3 Ethernet II	**Token-Ring 802.2** Token-Ring Snap	**FDDI 802.2** FDDI Snap	**PCN2 802.2** PCN2 Snap	(short) (long) (exception)

Frame Formats

The remainder of this appendix illustrates several of the standard frame formats. The white fields in the frame diagrams on the following pages are used to determine the frame type.

Ethernet 802.2

In Ethernet 802.2 packets, the *FrameLength* field contains a value less than or equal to 1500 (decimal). If the next three bytes (*DSAP, SSAP,* and *Control*) **do not** contain the Ethernet SNAP byte sequence: AAh, AAh, 03h, the packet is Ethernet 802.2.

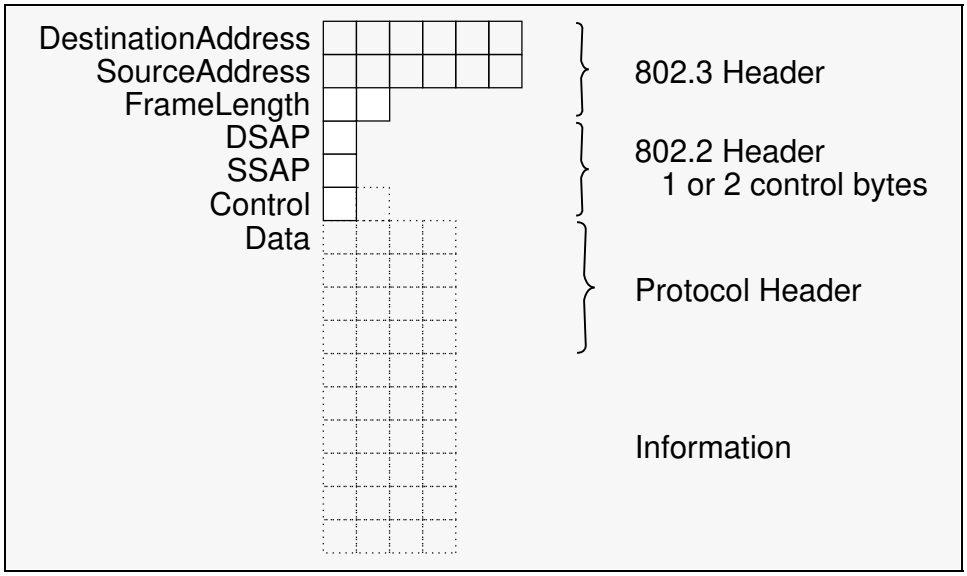

Figure D.1 Ethernet 802.2 Frame Type

Ethernet Raw 802.3

In Ethernet Raw 802.3 packets, the *FrameLength* field contains a value less than or equal to 1500 (decimal). In addition, the first two bytes of the *Data* area must contain the values FFh and FFh.

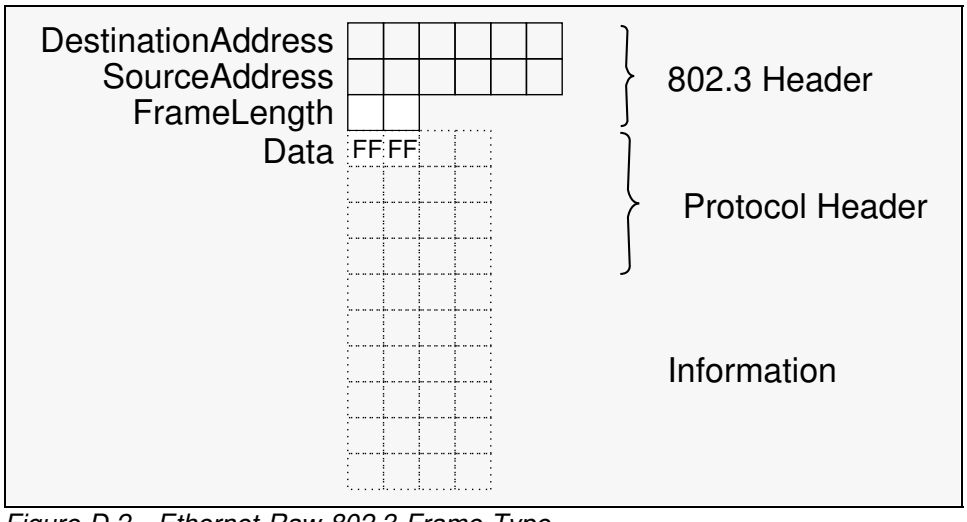

Figure D.2 Ethernet Raw 802.3 Frame Type

Ethernet SNAP

In Ethernet SNAP packets, the *FrameLength* field contains a value less than or equal to 1500 (decimal). In addition, the next three bytes (*DSAP*, *SSAP*, and *Control*) will contain the values AAh, AAh, and 03h respectively.

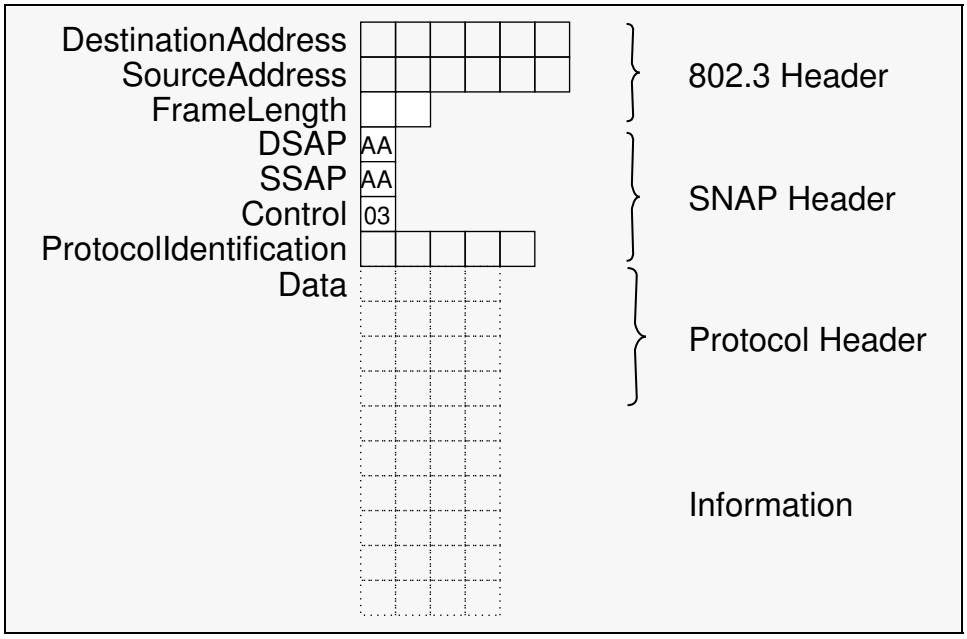

Figure D.3 Ethernet SNAP Frame Type

Ethernet II

In Ethernet II packets, the *FrameType* field will contain a value greater than 1500 (decimal).

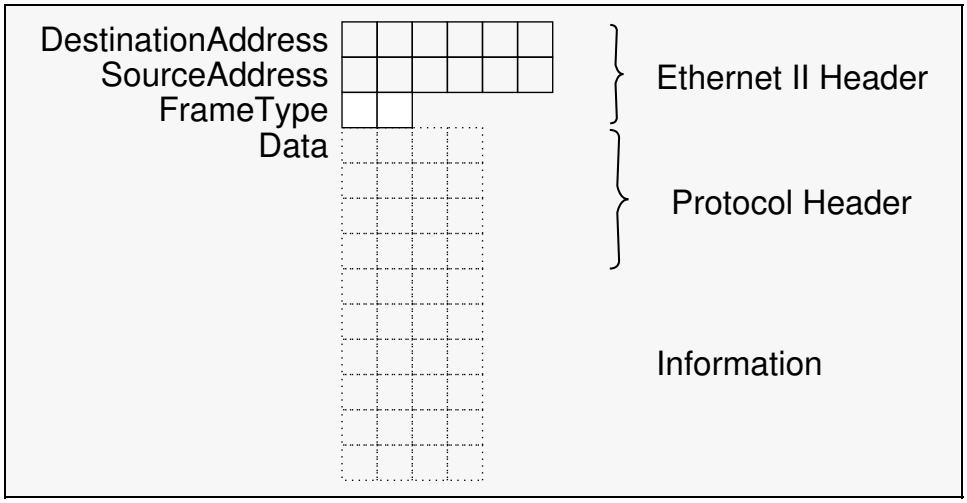

Figure D.4 Ethernet II Frame Type

Token-Ring 802.2

In Token-Ring 802.2 packets, the *DSAP, SSAP,* and *Control* fields **will not** contain the byte sequence: AAh, AAh, 03h. See the description of 802.2 Type I and Type II packet handling at the beginning of this section.

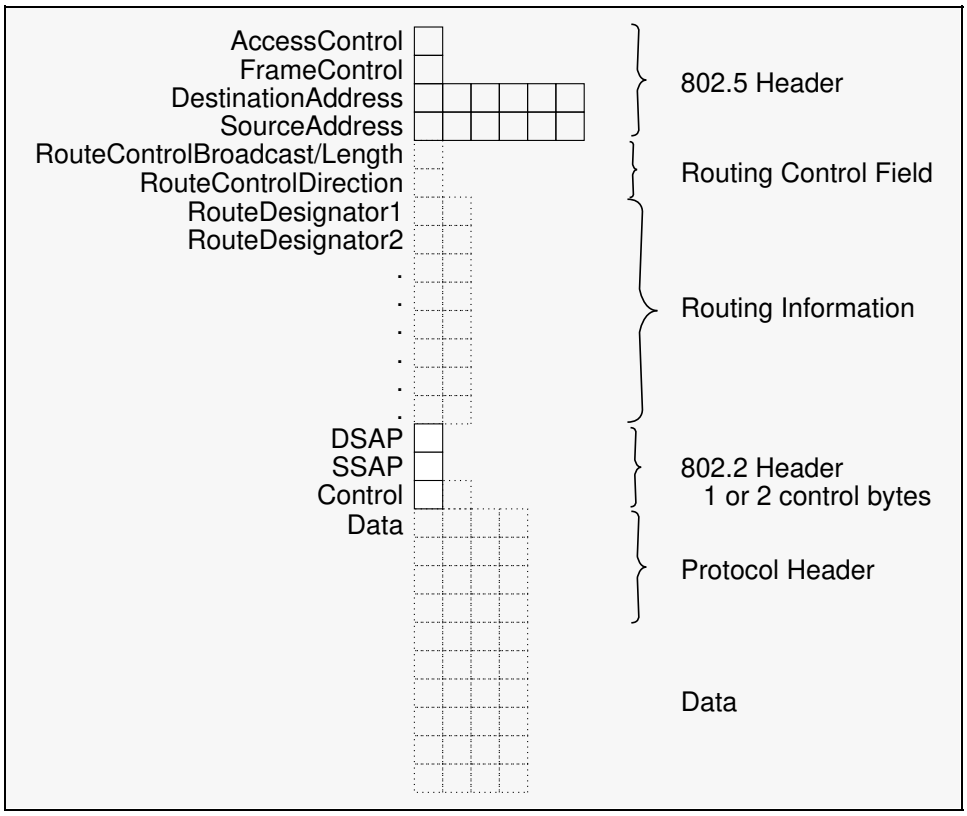

Figure D.5 Token-Ring 802.2 Frame Type

Token-Ring SNAP

In Token-Ring SNAP packets, the *DSAP, SSAP,* and *Control* fields will contain the byte sequence: AAh, AAh, 03h

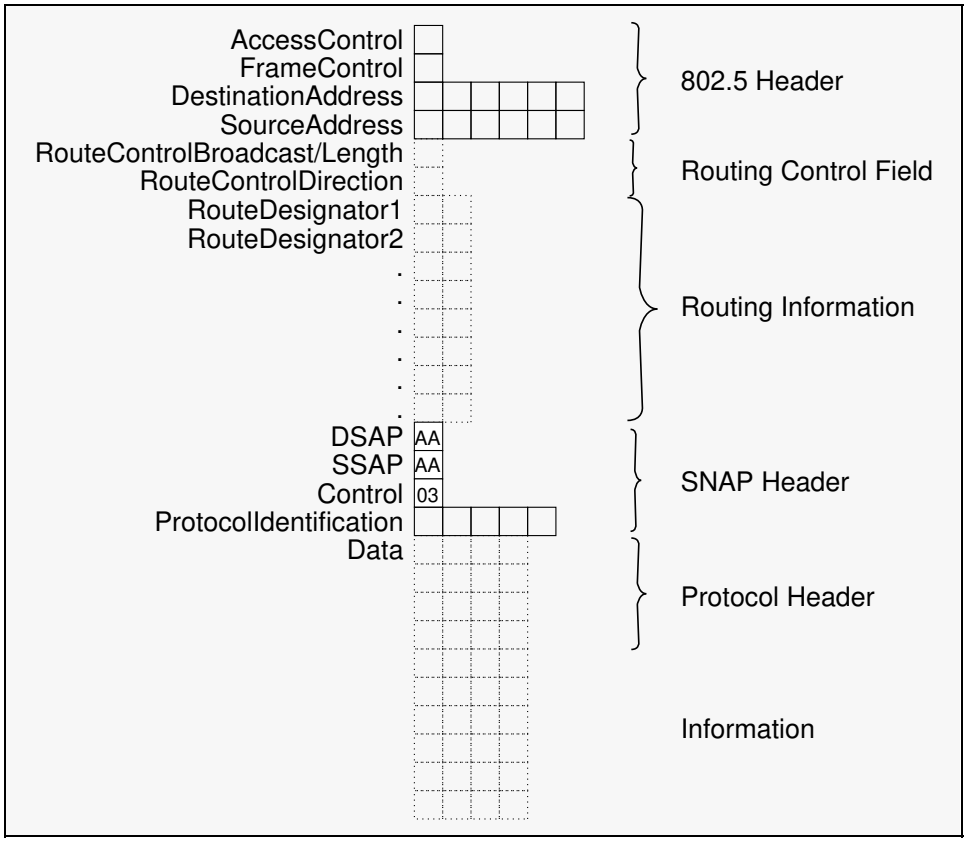

Figure D.6 Token-Ring SNAP Frame Type

FDDI 802.2

In FDDI 802.2 packets, the *DSAP, SSAP,* and *Control* fields **will not** contain the byte sequence: AAh, AAh, 03h. See the description of 802.2 Type I and Type II packet handling at the beginning of this section.

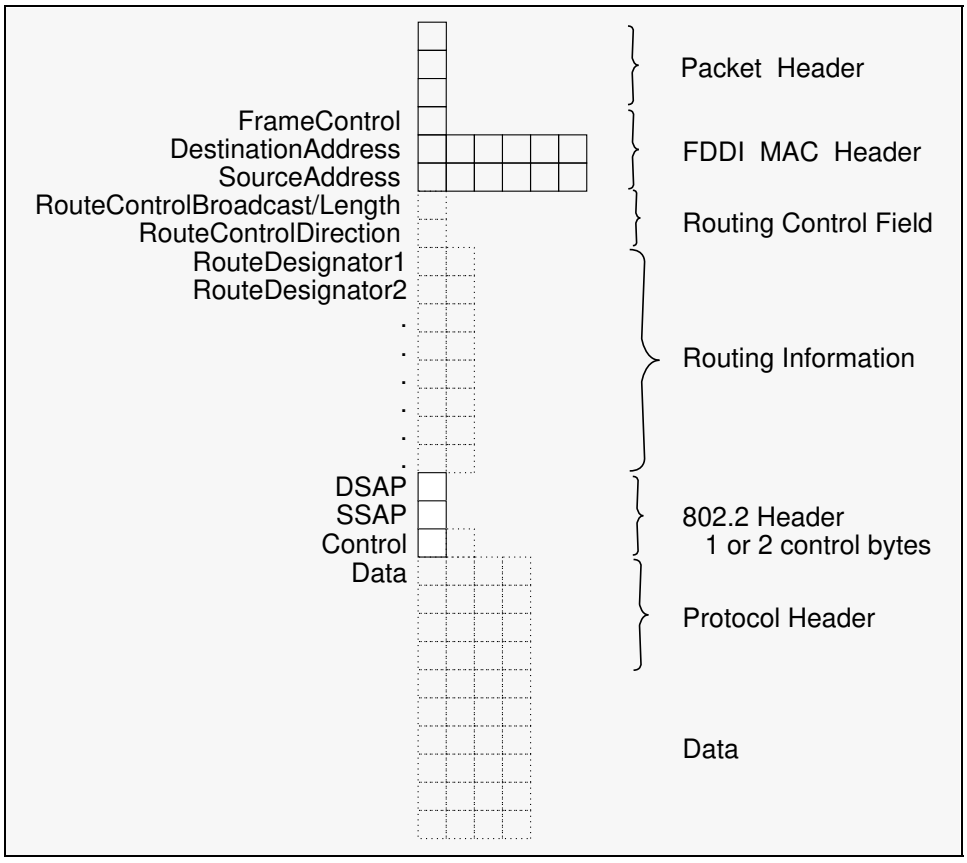

Figure D.7 FDDI 802.2 Frame Type

FDDI SNAP

In FDDI SNAP packets, the *DSAP, SSAP,* and *Control* fields will contain the byte sequence: AAh, AAh, 03h

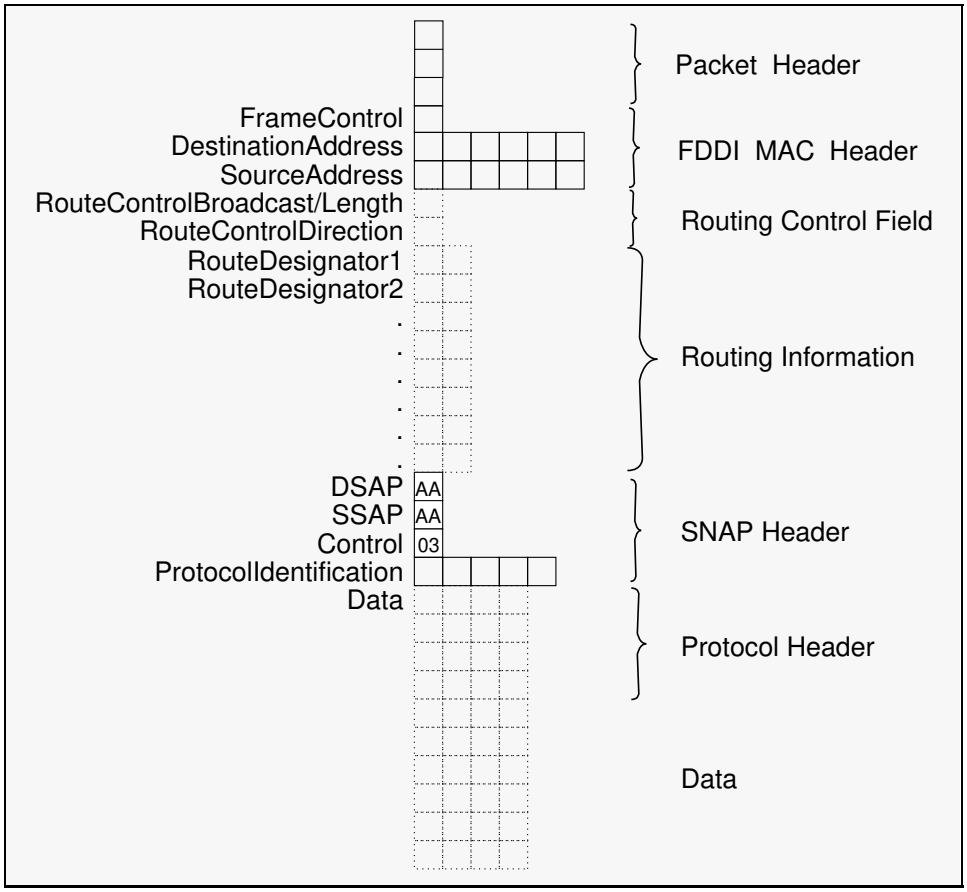

Figure D.8 FDDI SNAP Frame Type

PCN2 802.2

In PCN2 802.2 packets, the *DSAP, SSAP,* and *Control* fields **will not** contain the byte sequence: AAh, AAh, 03h. See the description of 802.2 Type I and Type II packet handling at the beginning of this section.

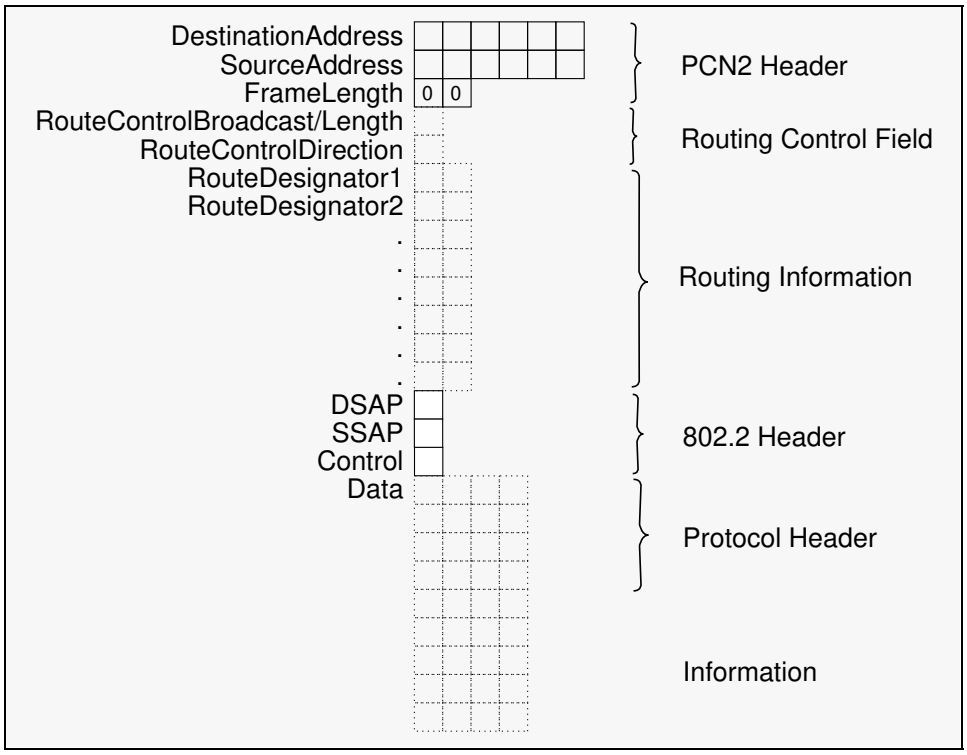

Figure D.9 PCN2 802.2 Frame Type

PCN2 SNAP

In PCN2 SNAP packets, the *DSAP, SSAP,* and *Control* fields will contain the byte sequence: AAh, AAh, 03h

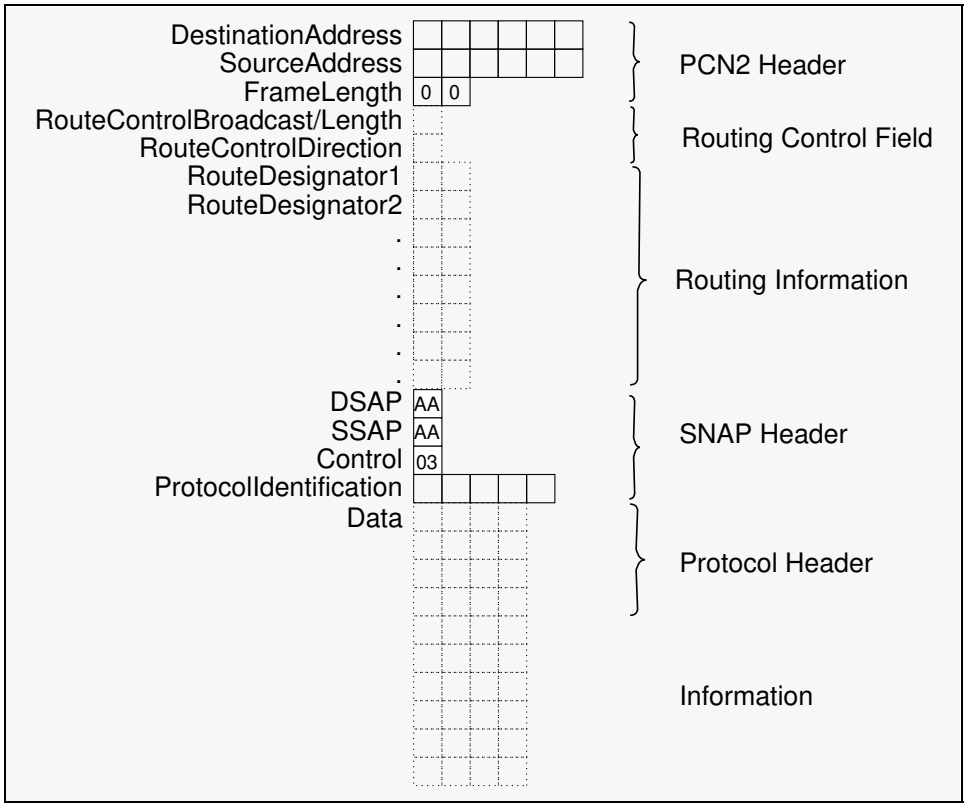

Figure D.10 PCN2 SNAP Frame Type

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 three 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 Datapoint has assigned to the adapter manufacturer to identify the company's packets on the wire. For example, Novell packets use FAh, AppleTalk packets use DDh.

Short Packet

In short packets, the third byte of the packet (*ByteOffset*) will contain a non-zero value. This packet format is for a single packet or fragment containing 0 to 249 bytes of data. The fields indicated in Figure D.11 are defined here.

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

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

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

$$\text{Byte Offset} = 256 - (N + 4)$$

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.

Split Flag - This is a single byte entry identifying which fragment of a total data packet is contained within this packet. Refer to Figure D.14 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.

N Bytes of Data - This is the actual data.

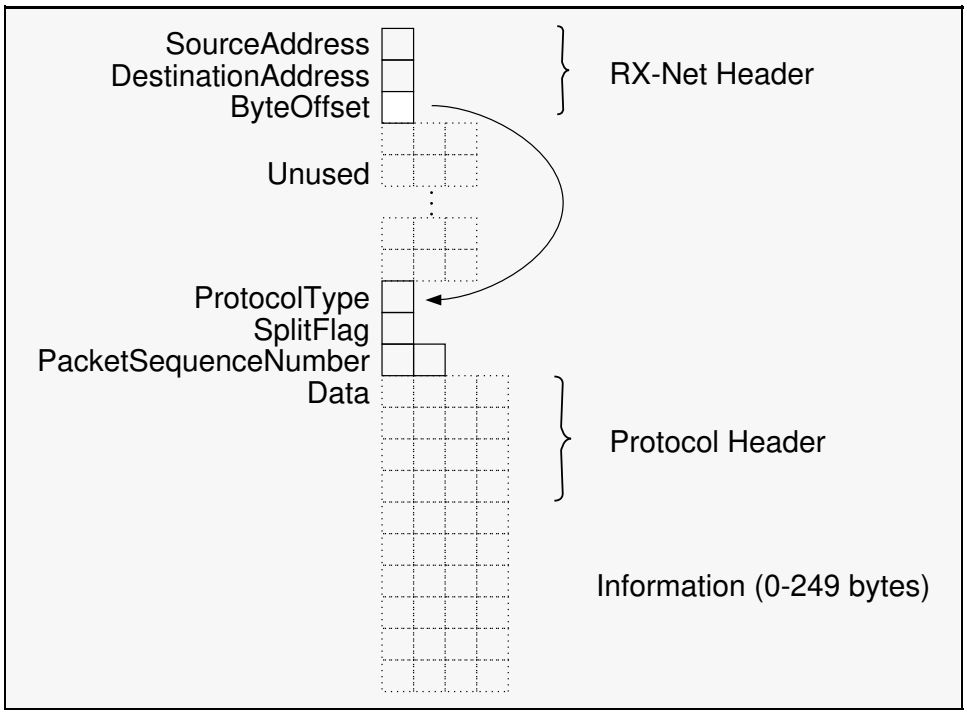

Figure D.11 RX-Net Short Packet Type

Long Packet

In long packets, the third byte of the packet (*LongPacketFlag*) will be 0, 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:

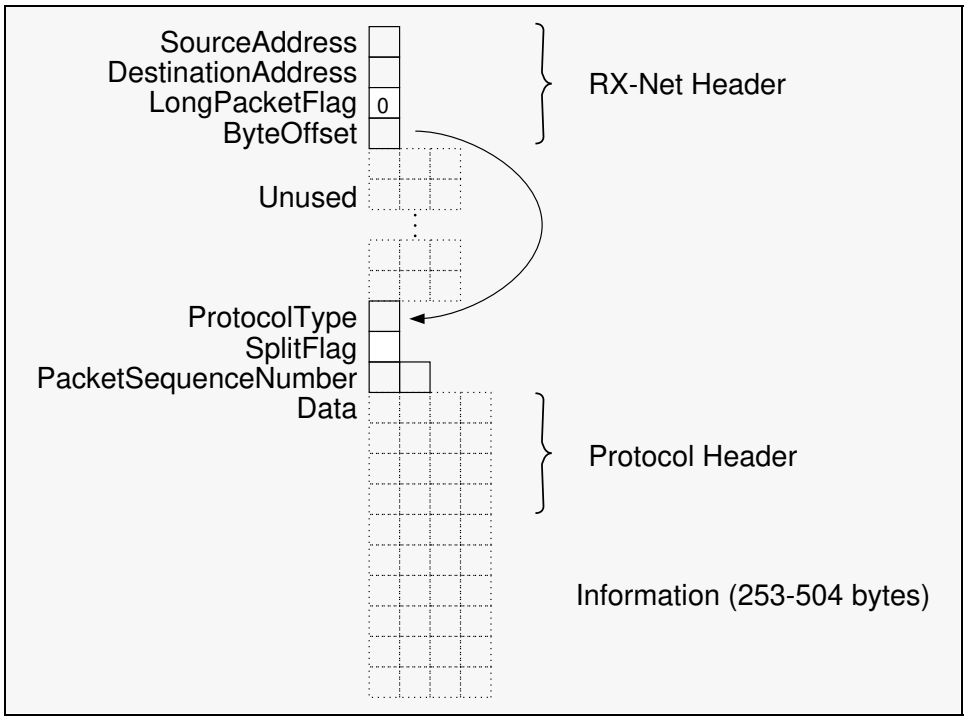

Figure D.12 RX-Net Long Packet Type

Source Address - This is the single byte address of the sending card 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:

$$\text{Byte Offset} = 512 - (N + 4)$$

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.

Split Flag - This is a single byte entry identifying which fragment of a total data packet is contained within this packet. Refer to Figure D.14 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.

N Bytes of Data - This is the actual data.

Exception Packet

In exception packets, the third byte of the packet (*LongPacketFlag*) will be 0, and the byte after the *ProtocolType* byte (*Pad2-SplitFlag*) will be FFh. 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. The fields are defined following the figure.

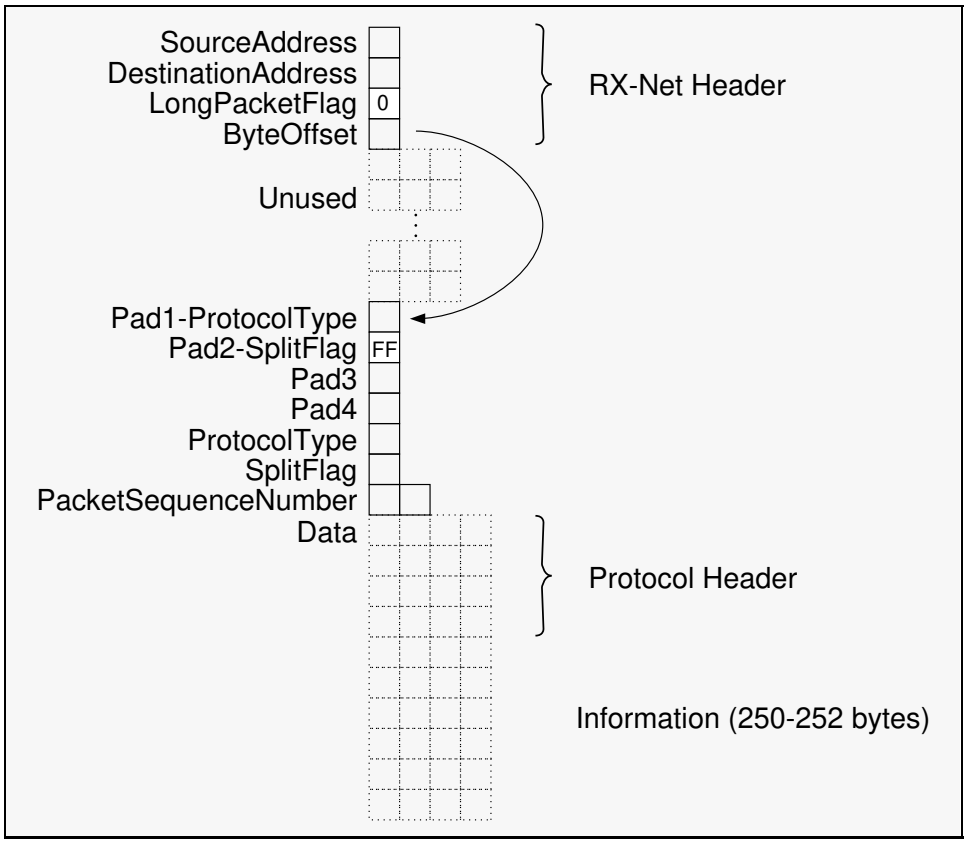

Figure D.13 RX-Net Exception Packet Type

Source Address - This is the single byte address of the sending card 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 + 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.

Pad2-Split Flag - This is a single byte entry, FFh, identifying this packet as an exception packet. Refer to Figure D.14 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.

Split Flag - This is a single byte entry identifying which fragment of a total data packet is contained within this packet (Refer to Figure D.14).

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.

N Bytes of Data - This is the actual data.

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 D.14 illustrates the SplitFlag byte.

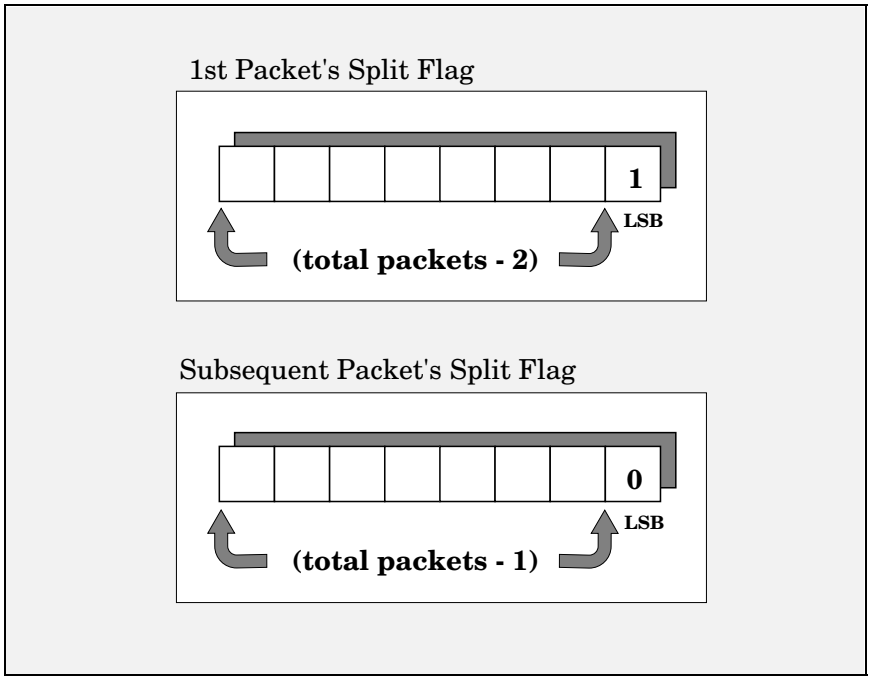

Figure D.14 Split Frame Flags

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 two.

Subsequent Fragments - The *more flag* is zero for all remaining fragments. The *fragment number* indicates the current fragment number minus one.

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 (2 total fragments - 2), and the *more flag* is set to indicate more fragments to come. The resulting flag value is 01h.

When the second packet is prepared, the *fragment number* is set to 0000001 (current fragment number - 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 one. The TSM 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: Split flag codes from F0h-FEh are reserved for future use.