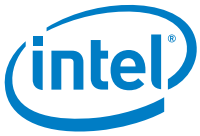


Term	Definition
TC	Traffic Class
TCI	For 802.1q, Tag Header field Tag Control Information (TCI); 2 octets.
TCO	Total Cost of Ownership (Management)
TCP/IP	Transmission Control Protocol/Internet Protocol
TDESC	Transmit Descriptor
TDP	Total Device Power
TDR	Time Domain Reflectometry
Teaming Mode	When the LAN is in Teaming mode, the 82599 is presented over the SMBus as one device and has one SMBus address.
TFCS	Transmit Flow Control Status
TLP	Transaction layer Packets
ToS	Type of Service
TPID	For 802.1q, Tag Header field Tag Protocol Identifier; 2 octets.
TPPAC	Transmit Packet Plane Arbitration Control
Transmit latency	Measured from Tail update until the packet is transmitted on the wire. It is assumed that a single packet is submitted for this traffic class and its latency is then measured in presence of traffic belonging to other traffic classes.
TS	Time Stamp
TSO	TCP or Transmit Segmentation offload — A mode in which a large TCP/UDP I/O is handled to the device and the device segments it to L2 packets according to the requested MSS.
TSS	Transmit Side Scaling
Tx, TX	Transmit
UBWG	User Bandwidth Group
ULP	Upper Layer Protocol
UP	User Priority
UR	Error Reporting Unsupported Request Error
VF	Virtual Function– A part of a PF assigned to a VI
VI	Virtual Image – A virtual machine to which a part of the I/O resources is assigned. Also known as a VM.
VM	Virtual Machine
VMM	Virtual Machine Monitor
VPD	Vital Product Data (PCI protocol).
VT	Virtualization
WB	Write Back
WC	Worst Case
WfM	Wired for Management was a primarily hardware-based system allowing a newly built computer without any software to be manipulated by a master computer that could access the hard disk of the new PC to paste the install program. It could also be used to update software and monitor system status remotely. Intel developed the system in the 1990s; it is now considered obsolete.
WoL	Wake-on-LAN Now called APM Wake up or Advanced power management Wake up.
WORD alignment	Implies that physical addresses must be aligned on even boundaries; i.e., the last nibble of the address may only end in 0, 2, 4, 6, 8, Ah, Ch, or Eh. For example, 0FECBD9A2h.

Term	Definition
WRR	Weighted Round-Robin
WSP	Weighted Strict Priority
XAUI	10 Gigabit Attachment Unit Interface
XFP	10 Gigabit Small Form Factor Pluggable modules
XGMII	10 Gigabit Media Independent Interface
XGXS	XGMII Extender Sub layer
XMT Frame Transmit	Most Recent Transmit Buffer Tail Register content

15.1 Register Attributes

Attribute	Description
RO	Read Only: Writes to this register setting do not affect the register value. Reads return either a constant or variable device state.
RW	Read Write: Writes to this register setting alter the register value. Reads return the value of the register.
RW1C	Read Write Clear: Register that is set to 1b by hardware, and cleared to 0b by software writing a 1b to the register
RWS	Read Write Set: Register that is set to1b by software by writing a 1b to the register, and cleared to 0b by hardware. Notation used in CSR sections
RWS	Read-write register: Register bits are read-write and can be either set or reset by software to the desired state. Bits are not cleared by reset and can only be reset with the PWRGOOD signal. Devices that consume AUX power are not allowed to reset sticky bits when AUX power consumption (either via AUX power or PME Enable) is enabled. Notation used in "PCI Express* Programming Interface" chapter.
WO	Write Only: Writes to this register alters the register value. Reads always return 0b.
RC	Read Clear. A register bit with this attribute is cleared after read. Writes have no effect on the bit value.
RW/RC	Read/Write and Read Clear.

Appendix A Packets and Frames

A.1 Legacy Packet Formats

A.1.1 ARP Packet Formats

A.1.1.1 ARP Request Packet

Offset	# of bytes	Field	Value (In Hex)	Action
0	6	Destination Address		Compare
6	6	Source Address		Stored
12	4	Possible VLAN Tag		Stored
12	8	Possible Len/LLC/SNAP Header		Stored
12	2	Type	0806	Compare
14	2	HW Type	0001	Compare
16	2	Protocol Type	0800	Compare
18	1	Hardware Size	06	Compare
19	1	Protocol Address Length	04	Compare
20	2	Operation	0001	Compare
22	6	Sender HW Address	-	Stored
28	4	Sender IP Address	-	Stored
32	6	Target HW Address	-	Ignore
38	4	Target IP Address	ARP IP address	Compare

A.1.1.2 ARP Response Packet

Offset	# of bytes	Field	Value
0	6	Destination Address	ARP Request Source Address
6	6	Source Address	Programmed from EEPROM or BMC
12	4	Possible VLAN Tag	From ARP Request
12	8	Possible Len/LLC/SNAP Header	From ARP Request
12	2	Type	0x0806
14	2	HW Type	0x0001
16	2	Protocol Type	0x0800
18	1	Hardware Size	0x06
19	1	Protocol Address Length	0x04
20	2	Operation	0x0002
22	6	Sender HW Address	Programmed from EEPROM or BMC
28	4	Sender IP Address	Programmed from EEPROM or BMC
32	6	Target HW Address	ARP Request Sender HW Address
38	4	Target IP Address	ARP Request Sender IP Address

A.1.1.3 Gratuitous ARP Packet

Offset	# of bytes	Field	Value
0	6	Destination Address	Broadcast address.
6	6	Source Address	
12	2	Type	0x0806
14	2	Hardware Type	0x0001
16	2	Protocol Type	0x0800
18	1	Hardware Size	0x06
19	1	Protocol Address Length	0x04
20	2	Operation	0x0001
22	6	Sender Hardware Address	

Offset	# of bytes	Field	Value
28	4	Sender IP Address	
32	6	Target Hardware Address	
38	4	Target IP Address	

A.1.2 IP and TCP/UDP Headers for TSO

This section outlines the format and content for the IP, TCP and UDP headers. the 82599 requires baseline information from the device driver in order to construct the appropriate header information during the segmentation process.

Header fields that are modified by the 82599 are highlighted in the figures below.

Note: The IP header is first shown in the traditional (i.e. RFC 791) representation, and because byte and bit ordering is confusing in that representation, the IP header is also shown in Little Endian format. The actual data will be fetched from memory in Little Endian format.

<table>
<tr><th colspan="4">0 1 2 3 4 5 6 7 | 1
8 9 0 1 2 3 4 5 | 2
6 7 8 9 0 1 2 3 | 3
4 5 6 7 8 9 0 1</th></tr>
<tr><td>Version</td><td>IP Hdr Length</td><td>TYPE of service</td><td>Total length (IP header + payload length)</td></tr>
<tr><td colspan="3">Identification</td><td>Flags | Fragment Offset</td></tr>
<tr><td colspan="2">Time to Live</td><td>Layer 4 Protocol ID</td><td>Header Checksum</td></tr>
<tr><td colspan="4">Source Address</td></tr>
<tr><td colspan="4">Destination Address</td></tr>
<tr><td colspan="4">Options</td></tr>
</table>

Figure A-1 IPv4 Header (Traditional Representation - most left byte first on the wire)

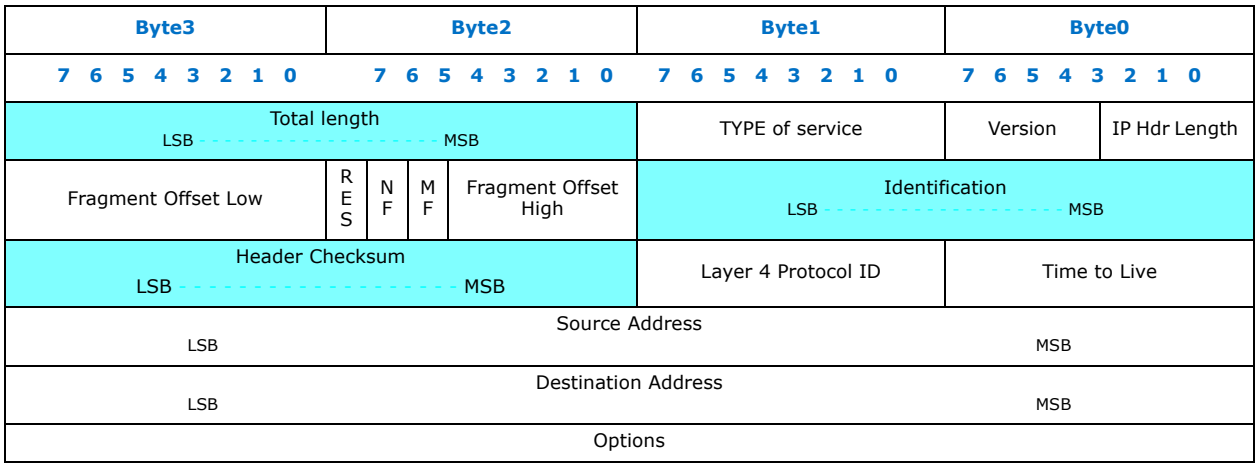


Figure A-2 IPv4 Header (Little Endian Order - Byte 0 first on the wire)

Identification is incremented on each packet.

Flags Field Definitions:

The Flags field is defined below. Note that hardware does not evaluate or change these bits.

- MF - More Fragments
- NF - No Fragments
- Reserved

the 82599 does TCP segmentation, not IP Fragmentation. IP Fragmentation may occur in transit through a network's infrastructure.

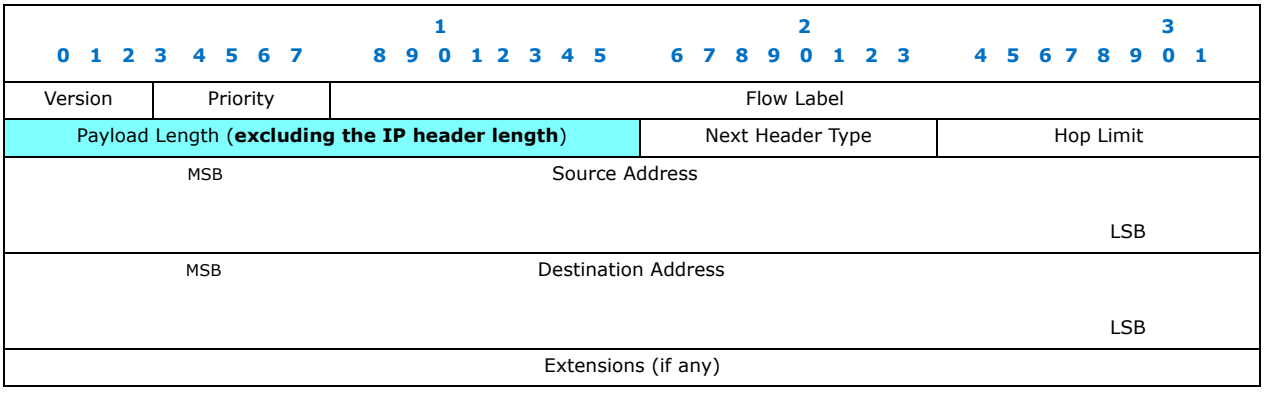


Figure A-3 IPv6 Header (Traditional Representation - most left byte first on the wire)

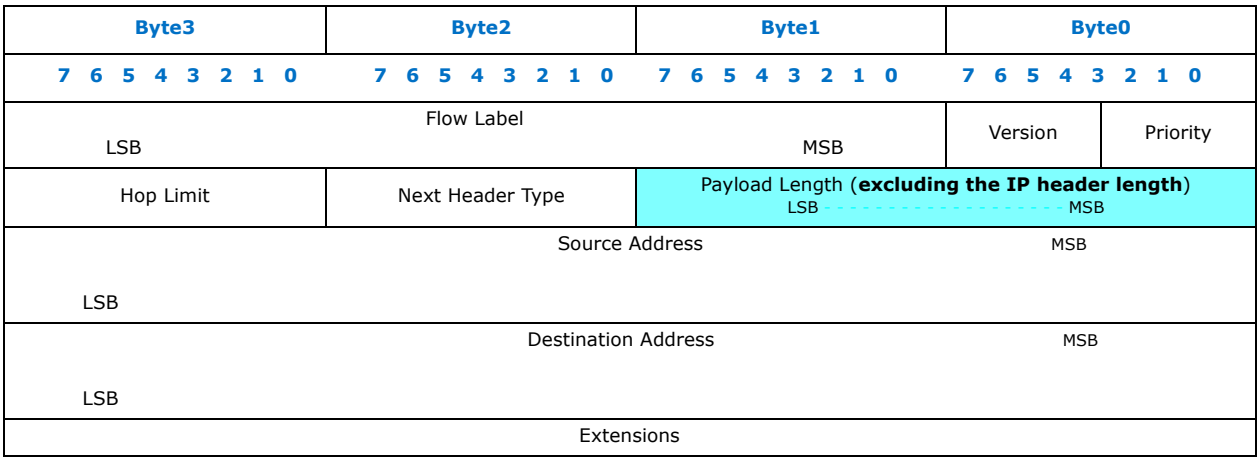


Figure A-4 IPv6 Header (Little Endian Order - byte 0 first on the wire)

A TCP or UDP frame uses a 16 bit wide one's complement checksum. The checksum word is computed on the outgoing TCP or UDP header and payload, and on the Pseudo Header. Details on checksum computations are provided in Section 7.2.4.6.

Note: TCP and UDP over IPv6 requires the use of checksum, where it is optional for UDP over IPv4.

The TCP header is first shown in the traditional (i.e. RFC 793) representation, and because byte and bit ordering is confusing in that representation, the TCP header is also shown in Little Endian format. The actual data will be fetched from memory in Little Endian format.

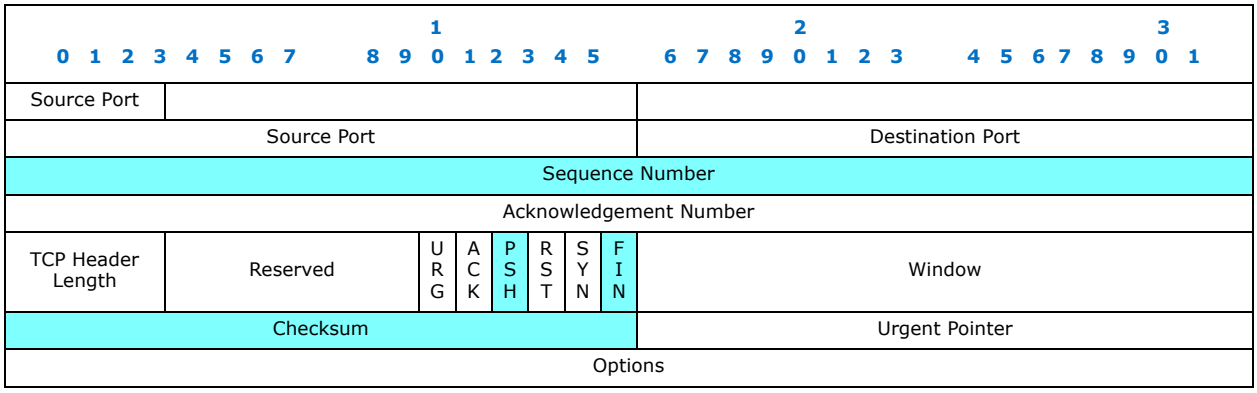


Figure A-5 TCP Header (Traditional Representation)

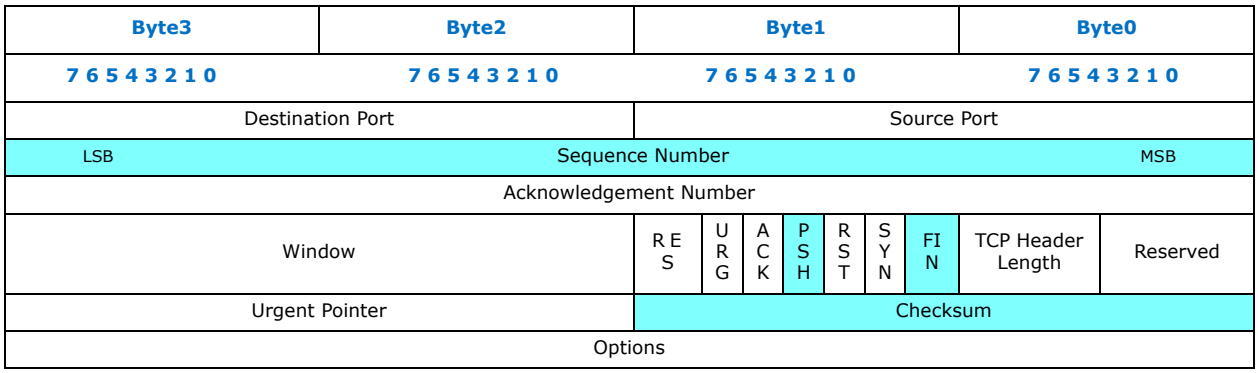


Figure A-6 TCP Header (Little Endian)

The TCP header is always a multiple of 32 bit words. TCP options may occupy space at the end of the TCP header and are a multiple of 8 bits in length. All options are included in the checksum.

The checksum also covers a 96-bit pseudo header prefixed to the TCP Header (see Figure A-7 below). For IPv4 packets, this pseudo header contains the IP Source Address, the IP Destination Address, the IP Protocol field, and TCP Length. Software pre-calculates the partial pseudo header sum, that includes IPv4 SA, DA and protocol types, but NOT the TCP length, and stores this value into the TCP checksum field of the packet. For both IPv4 and IPv6, hardware needs to factor in the TCP length to the software supplied pseudo header partial checksum.

Note: When calculating the TCP pseudo header, the byte ordering can be tricky. One common question is whether the Protocol ID field is added to the "lower" or "upper" byte of the 16- bit sum. The Protocol ID field should be added to the least significant byte (LSB) of the 16-bit pseudo header sum, where the most significant byte (MSB) of the 16-bit sum is the byte that corresponds to the first checksum byte out on the wire.

The TCP Length field is the TCP Header Length including option fields plus the data length in bytes, which is calculated by hardware on a frame-by-frame basis. The TCP Length does not count the 12 bytes of the pseudo header. The TCP length of the packet is determined by hardware as:

TCP Length = min (MSS,PAYLOADLEN) + L5_LEN

The two flags that may be modified are defined as:

- PSH: receiver should pass this data to the app without delay
- FIN: sender is finished sending data

The handling of these flags is described in Section 7.2.4.7.

"Payload" is normally MSS except for the last packet where it represents the remainder of the payload.

IPv4 Source Address		
IPv4 Destination Address		
Zero	Layer 4 Protocol ID	TCP/UDP Length

Figure A-7 TCP/UDP Pseudo Header Content for IPv4 (Traditional Representation)

IPv6 Source Address	
IPv6 Final Destination Address	
TCP/UDP Packet Length	
Zero	Next Header

Figure A-8 TCP/UDP Pseudo Header Content for IPv6 (Traditional Representation)

Note: From RFC2460:

- If the IPv6 packet contains a Routing header, the Destination Address used in the pseudo-header is that of the final destination. At the originating node, that address will be in the last element of the Routing header; at the recipient(s), that address will be in the Destination Address field of the IPv6 header.
- The Next Header value in the pseudo-header identifies the upper-layer protocol (e.g., 6 for TCP, or 17 for UDP). It will differ from the Next Header value in the IPv6 header if there are extension headers between the IPv6 header and the upper-layer header.
- The Upper-Layer Packet Length in the pseudo-header is the length of the upper-layer header and data (e.g., TCP header plus TCP data). Some upper-layer protocols carry their own length information (e.g., the Length field in the UDP header); for such protocols, that is the length used in the pseudo- header. Other protocols (such as TCP) do not carry their own length information, in which case the length used in the pseudo-header is the Payload Length from the IPv6 header, minus the length of any extension headers present between the IPv6 header and the upper-layer header.
- Unlike IPv4, when UDP packets are originated by an IPv6 node, the UDP checksum is not optional. That is, whenever originating a UDP packet, an IPv6 node must compute a UDP checksum over the packet and the pseudo-header, and, if that computation yields a result of zero, it must be changed to hex FFFF for placement in the UDP header. IPv6 receivers must discard UDP packets containing a zero checksum, and should log the error.

A type 0 Routing header has the following format:

Next Header	Hdr Ext Len	Routing Type "0"	Segments Left "n"
Reserved			
Address[1]			
Address[2]			
...			
Final Destination Address [n]			

Figure A-9 IPv6 Routing Header (Traditional Representation)

- Next Header - 8-bit selector. Identifies the type of header immediately following the Routing header. Uses the same values as the IPv4 Protocol field [RFC-1700 et seq.].
- Hdr Ext Len - 8-bit unsigned integer. Length of the Routing header in 8-octet units, not including the first 8 octets. For the Type 0 Routing header, Hdr Ext Len is equal to two times the number of addresses in the header.
- Routing Type - 0.
- Segments Left - 8-bit unsigned integer. Number of route segments remaining, i.e., number of explicitly listed intermediate nodes still to be visited before reaching the final destination. Equal to "n" at the source node.

Reserved - 32-bit reserved field. Initialized to zero for transmission; ignored on reception.

- Address[1...n] - Vector of 128-bit addresses, numbered 1 to n.

The UDP header is always 8 bytes in size with no options.

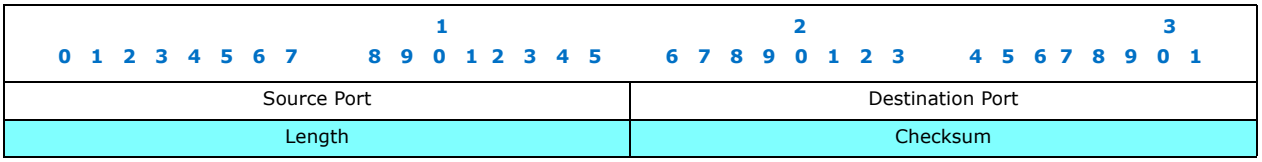


Figure A-10 UDP Header (Traditional Representation)

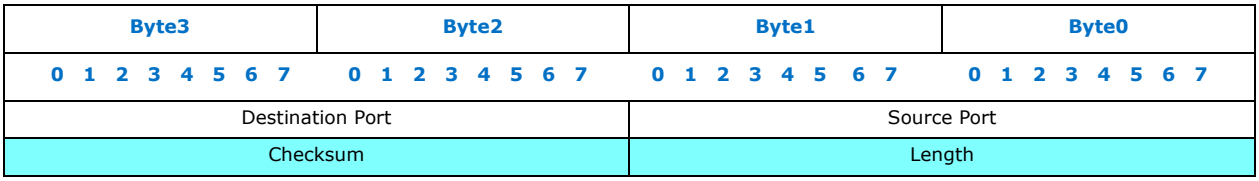


Figure A-11 UDP Header (Little Endian Order)

UDP pseudo header has the same format as the TCP pseudo header. The pseudo header prefixed to the UDP header contains the IPv4 source address, the IPv4 destination address, the IPv4 protocol field, and the UDP length (same as the TCP Length discussed above). This checksum procedure is the same as is used in TCP.

Unlike the TCP checksum, the UDP checksum is optional (for IPv4). Software must set the TXSM bit in the TCP/IP Context Transmit Descriptor to indicate that a UDP checksum should be inserted. Hardware will not overwrite the UDP checksum unless the TXSM bit is set.

A.1.3 Magic Packet

The Magic Packet is a broadcast frame, but could also be a Multicast or Unicast Ethernet MAC Addresses. the 82599 accepts this packet if it matches any of its pre-programmed Ethernet MAC Addresses. Magic packet can be sent over a variety of connectionless protocols (usually UDP or IPX). The Magic packet pattern is composed of the following sequences:

- Synchronization stream composed of 6 bytes equal to 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
- Unique pattern composed of 16 times the end node Ethernet MAC Address. the 82599 expect for the Ethernet MAC Address stored in the RAL[0] and RAH[0] registers.

the 82599 looks for the synchronization pattern and the sequence of 16 Ethernet MAC Addresses. It does not check the packet content and the length of the header that precedes the magic pattern nor any data that follows it.

A.1.4 SNAP Packet Format

the 82599 supports SNAP packets for all offload capabilities as described in this document. The device identifies SNAP packet by the following fields highlighted in “light green” color in the diagram below.

<table>
<tr><td rowspan="2">L2 Destination & Source MAC Addresses</td><td rowspan="2">L2 Length must be less equal than 1500</td><td colspan="4">SNAP Header</td><td rowspan="2">Payload</td><td rowspan="2">CRC</td></tr>
<tr><td>DSAP
0xAA</td><td>SSAP
0xAA</td><td>CTRL
0x03</td><td>ORG
0x00 0x00 0x00</td><td>EtherType</td></tr>
<tr><td></td><td colspan="5"><----------- Compared by the Hardware ----------></td><td></td><td></td></tr>
</table>

The packet may carry VLAN header(s). In that case, the EtherType in the SNAP header has the EtherType value of the first VLAN header.

The packet may be encapsulated by LinkSec. In that case, the LinkSec header appears before the L2 length field.

A.2 Packet Types for Packet Split Filtering

The following packet types are supported by the 'Packet Split' feature in the 82599. This section describes the packets from the 'split-header' point of view. This means that when describing the different fields that are checked and compared, it emphasizes only the fields that are needed to calculate the header length. This document describes the checks that are done after the decision to pass the packet to the host memory was done.

Terminology:

- Compare - The field values are compared and must be exactly equal to the value specified in this document.
- Checked - The field values are checked for calculation (header length …).
- Ignore - The field values are ignored but the field is counted as part of the header.

A.2.1 Type 1.1: Ethernet (VLAN/SNAP) IP Packets

A.2.1.1 Type 1.1: Ethernet, IP, Data

This type contains only Ethernet header and Ipv4 header while the payload header of the IP is not Ipv6/TCP/UDP.

Offset	# of bytes	Field	Value	Action	Comment
0	6	Destination Address		Ignore	MAC Header – processed by main address filter, or broadcast
6	6	Source Address		Ignore	
12	S=(0/4)	Possible VLAN Tag	8100 ****	Compare	
12+S	D=(0/8)	Possible Len/LLC/SNAP Header		Compare	
12+D+S	2	Type	0800h	Compare	IP
IPv4 header					
14+D+S	1	Version/ HDR length	4Xh	Compare	Check IPv4 and header length.
15+D+S	1	Type of Service	-	Ignore	
16+D+S	2	Packet Length	-	Ignore	
18+D+S	2	Identification	-	Ignore	
20+D+S	2	Fragment Info	>0 or MF bit is set	Check	Check that the packet is fragmented.
22+D+S	1	Time to live	-	Ignore	
23+D+S	1	Protocol		Ignore	Has no meaning if the packet is fragmented.
24+D+S	2	Header Checksum	-	Ignore	
26+D+S	4	Source IP Address	-	Ignore	
30+D+S	4	Destination IP Address	-	Ignore	
34+D+S	N	Possible IP Options		Ignore	

A.2.1.2 Type 1.2: Ethernet (SNAP/VLAN), Ipv4, UDP

This type contains only Ethernet header, Ipv4 header, and UDP header.

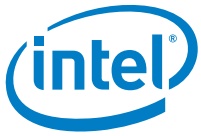
Offset	# of bytes	Field	Value	Action	Comment
0	6	Destination Address		Ignore	MAC Header – processed by main address filter, or broadcast filter.
6	6	Source Address		Ignore	
12	S=(0/4)	Possible VLAN Tag	8100 ****	Check	
12+S	D=(0/8)	Possible Len/LLC/SNAP Header		Check	
12+D+S	2	Type	0800h	Compare	IP
IP header					
14+D+S	1	Version/ HDR length	4Xh	Compare	Check IPv4 and header length.
15+D+S	1	Type of Service	-	Ignore	
16+D+S	2	Packet Length	-	Ignore	
18+D+S	2	Identification	-	Ignore	
20+D+S	2	Fragment Info	(xx00) 000h	Compare	
22+D+S	1	Time to live	-	Ignore	
23+D+S	1	Protocol	0x11	Compare	UDP header
24+D+S	2	Header Checksum	-	Ignore	
26+D+S	4	Source IP Address	-	Ignore	
30+D+S	4	Destination IP Address	-	Ignore	
34+D+S	N	Possible IP Options		Ignore	
UDP header					
34+D+S+N	2	Source Port	Not (0x801)	Check	Not NFS packet
36+D+S+N	2	Destination Port	Not (0x801)	Check	Not NFS packet
38+D+S+N	2	Length	-	Ignore	
40+D+S+N	2	Checksum	-	Ignore	

In this case, the packet is split after (42+D+S+N) bytes.

A.2.1.3 Type 1.3: Ethernet (VLAN/SNAP) Ipv4 TCP

This type contains only Ethernet header, Ipv4 header, and TCP header.

Offset	# of bytes	Field	Value	Action	Comment
0	6	Destination Address		Ignore	MAC Header – processed by main address filter, or broadcast filter.
6	6	Source Address		Ignore	
12	S=(0/4)	Possible VLAN Tag	8100 ****	Check	
12+S	D=(0/8)	Possible Len/LLC/SNAP Header		Check	
12+D+S	2	Type	0800h	Compare	IP
IPv4 header					
14+D+S	1	Version/ HDR length	4Xh	Compare	Check IPv4 and header length.
15+D+S	1	Type of Service	-	Ignore	
16+D+S	2	Packet Length	-	Ignore	
18+D+S	2	Identification	-	Ignore	
20+D+S	2	Fragment Info	0x00	Compare	
22+D+S	1	Time to live	-	Ignore	
23+D+S	1	Protocol	0x06	Compare	TCP header
24+D+S	2	Header Checksum	-	Ignore	
26+D+S	4	Source IP Address	-	Ignore	
30+D+S	4	Destination IP Address	-	Ignore	
34+D+S	N	Possible IP Options		Ignore	
TCP header					
34+D+S+N	2	Source Port	Not (0x801)	Check	Not NFS packet
36+D+S+N	2	Destination Port	Not (0x801)	Check	Not NFS packet
38+D+S+N	4	Sequence number	-	Ignore	
42+D+S+N	4	Acknowledge number	-	Ignore	
46+D+S+N	1/2	Header Length		Check	
46.5+D+S+N	1.5	Different bits	-	Ignore	



Offset	# of bytes	Field	Value	Action	Comment
48+D+S+N	2	Window size	-	Ignore	
50+D+S+N	2	TCP checksum	-	Ignore	
52+D+S+N	2	Urgent pointer	-	Ignore	
54+D+S+N	F	TCP options	-	Ignore	

In this case, the packet is split after (54+D+S+N+F) bytes.

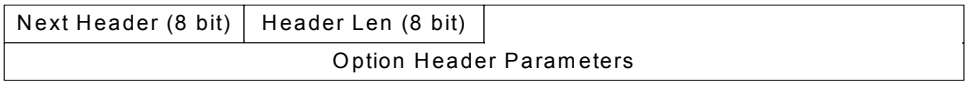
- N = (IP HDR length -5) * 4.
- F = (TCP header length - 5) * 4.

A.2.1.4 Type 1.4: Ethernet Ipv4 Ipv6

A.2.1.4.1 IPv6 header options processing

This type of processing looks at the next-header field and header length in order to determine the identity of the next-header processes, the Ipv6 options, and it's length.

If the next header in the Ipv6 header is equal to 0x00/0x2B/0x2C/0x3B/0x3c it means that the next header is an Ipv6 option header and this is its structure:



Header Len determines the length of the header while the next header field determines the identity of the next header (should be any IPv6 extension header for another Ipv6 header option).

A.2.1.4.2 IPv6 Next Header Values

When parsing an IPv6 header, the 82599 does not parse all possible extension headers and if there is an extension header that is not supported by the 82599 then the packet is treated as an unknown payload after the IPv6 header.

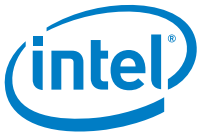
Value	Header type
0x00	Hop by Hop
0x2B	Routing
0x2C	Fragment
0x3B	No next header (EOL)
0x3C	Destination option header

- The next header in a fragment header is ignored and this extension header is expected to be the last header.

A.2.1.4.3 Type 1.4.1: Ethernet Ipv4 Ipv6 data

This type contains only Ethernet header, Ipv4 header, and IPv6 header.

Offset	# of bytes	Field	Value	Action	Comment
0	6	Destination Address		Ignore	MAC Header – processed by main address filter, or broadcast filter.
6	6	Source Address		Ignore	
12	S=(0/4)	Possible VLAN Tag	8100	Check	
12+S	D=(0/8)	Possible Len/LLC/SNAP Header		Check	
12+D+S	2	Type	0800h	Compare	IP
Ipv4 header					
14+D+S	1	Version/ HDR length	4Xh	Compare	Check IPv4 and header length.
15+D+S	1	Type of Service	-	Ignore	
16+D+S	2	Packet Length	-	Ignore	
18+D+S	2	Identification	-	Ignore	
20+D+S	2	Fragment Info	0x00	Compare	
22+D+S	1	Time to live	-	Ignore	
23+D+S	1	Protocol	0x29	Compare	Ipv6
24+D+S	2	Header Checksum	-	Ignore	
26+D+S	4	Source IP Address	-	Ignore	
30+D+S	4	Destination IP Address	-	Ignore	
34+D+S	N	Possible IP Options		Ignore	
Ipv6 header					
34+D+S+N	1	Version/ Traffic Class	6Xh	Compare	Check IPv6
35+D+S+N	3	Traffic Class/Flow Label	-	Ignore	
38+D+S+N	2	Payload Length	-	Ignore	
40+D+S+N	1	Next Header	IPv6 extension headers	Check	
41+D+S+N	1	Hop Limit	-	Ignore	
42+D+S+N	16	Source Address	-	Ignore	



Offset	# of bytes	Field	Value	Action	Comment
48+D+S+N	16	Destination Address		Ignore	
74+D+S+N	B	Possible IPv6 Next Headers	-	Ignore	

In this case the packet is split after (74+D+S+N+B) bytes.

- N = (IP HDR length - 5) * 4.
- One of the extension headers of the IPv6 packets must be a “fragment header” in order for the packet to be parsed.

A.2.1.4.4 Type 1.4.2: Ethernet (VLAN/SNAP) Ipv4 Ipv6 UDP

This type contains only Ethernet header, Ipv4 header, IPv6 header and UDP header.

Offset	# of bytes	Field	Value	Action	Comment
0	6	Destination Address		Ignore	MAC Header – processed by main address filter, or broadcast filter.
6	6	Source Address		Ignore	
12	S=(0/4)	Possible VLAN Tag	8100	Check	
12+S	D=(0/8)	Possible Len/LLC/SNAP Header		Check	
12+D+S	2	Type	0800h	Compare	IP
Ipv4 header					
14+D+S	1	Version/ HDR length	4Xh	Compare	Check IPv4 and header length
15+D+S	1	Type of Service	-	Ignore	
16+D+S	2	Packet Length	-	Ignore	
18+D+S	2	Identification	-	Ignore	
20+D+S	2	Fragment Info	0x00	Compare	
22+D+S	1	Time to live	-	Ignore	
23+D+S	1	Protocol	0x29	Compare	Ipv6
24+D+S	2	Header Checksum	-	Ignore	
26+D+S	4	Source IP Address	-	Ignore	
30+D+S	4	Destination IP Address	-	Ignore	

Offset	# of bytes	Field	Value	Action	Comment
34+D+S	N	Possible IP Options		Ignore	
Ipv6 header					
34+D+S+N	1	Version/ Traffic Class	6Xh	Compare	Check IPv6
35+D+S+N	3	Traffic Class/Flow Label	-	Ignore	
38+D+S+N	2	Payload Length	-	Ignore	
40+D+S+N	1	Next Header	IPv6 extension header or 0x11	Check	IPv6 extension headers:
41+D+S+N	1	Hop Limit	-	Ignore	
42+D+S+N	16	Source Address	-	Ignore	
58+D+S+N	16	Destination Address		Ignore	
74+D+S+N	B	Possible IPv6 Next Headers	-	Ignore	
UDP header					
74+D+S+N+B	2	Source Port	Not (0x801)	Check	Not NFS packet
76+D+S+N+B	2	Destination Port	Not (0x801)	Check	Not NFS packet
78+D+S+N+B	2	Length	-	Ignore	
80+D+S+N+B	2	Checksum	-	Ignore	

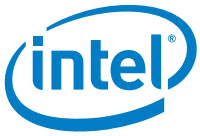
In this case the packet is split after (82+D+S+N+B) bytes.

N = (IP HDR length - 5) * 4.

A.2.1.4.5 Type 1.4.3: Ethernet (VLAN/SNAP) Ipv4 Ipv6 TCP

This type contain only Ethernet header, Ipv4 header, IPv6 header and TCP header.

Offset	# of bytes	Field	Value	Action	Comment
0	6	Destination Address		Ignore	MAC Header – processed by main address filter, or broadcast filter.
6	6	Source Address		Ignore	
12	S=(0/4)	Possible VLAN Tag	8100	Check	
12+S	D=(0/8)	Possible Len/LLC/SNAP Header		Check	



Offset	# of bytes	Field	Value	Action	Comment
12+D+S	2	Type	0800h	Compare	IP
Ipv4 header					
14+D+S	1	Version/ HDR length	4Xh	Compare	Check IPv4 and header length
15+D+S	1	Type of Service	-	Ignore	
16+D+S	2	Packet Length	-	Ignore	
18+D+S	2	Identification	-	Ignore	
20+D+S	2	Fragment Info	0x00	Compare	
22+D+S	1	Time to live	-	Ignore	
23+D+S	1	Protocol	0x2A	Compare	Ipv6
24+D+S	2	Header Checksum	-	Ignore	
26+D+S	4	Source IP Address	-	Ignore	
30+D+S	4	Destination IP Address	-	Ignore	
34+D+S	N	Possible IP Options		Ignore	
Ipv6 header					
34+D+S+N	1	Version/ Traffic Class	6Xh	Compare	Check IPv6
35+D+S+N	3	Traffic Class/Flow Label	-	Ignore	
38+D+S+N	2	Payload Length	-	Ignore	
40+D+S+N	1	Next Header	Ipv6 extension header Or 0x06	Check	IPv6 extension headers
41+D+S+N	1	Hop Limit	-	Ignore	
42+D+S+N	16	Source Address	-	Ignore	
58+D+S+N	16	Destination Address		Ignore	
74+D+S+N	B	Possible IPv6 Next Headers	-	Ignore	
TCP header					
74+T	2	Source Port	Not (0x801)	Check	Not NFS packet
76+T	2	Destination Port	Not (0x801)	Check	Not NFS packet
78+T	4	Sequence number	-	Ignore	
82+T	4	Acknowledge number	-	Ignore	

Offset	# of bytes	Field	Value	Action	Comment
86+T	1/2	Header Length		Check	
86.5+T	1.5	Different bits	-	Ignore	
88+T	2	Window size	-	Ignore	
90+T	2	TCP checksum	-	Ignore	
92+T	2	Urgent pointer	-	Ignore	
94+T	F	TCP options	-	Ignore	

In this case the packet is split after (94+D+S+N+B+F) bytes.

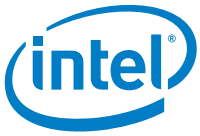
- T = D+S+N+B
- N = (IP HDR length - 5) * 4.
- F = (TCP HDR length - 5)*4

A.2.2 Type 2: Ethernet, Ipv6

A.2.2.1 Type 2.1: Ethernet, Ipv6 data

This type contains only an Ethernet header and an Ipv6 header while the packet should be a fragmented packet. If the packet is not fragmented and the Next header is not supported then the header is not split. The supported packet types for header split are programmed in PSRTYPE register (per VF).

Offset	# of bytes	Field	Value (hex)	Action	Comment
0	6	Destination Address		Ignore	MAC Header – processed by main address filter, or broadcast filter.
6	6	Source Address		Ignore	
12	S=(0/4)	Possible VLAN Tag	8100	Check	
12+S	D=(0/8)	Possible Len/LLC/SNAP Header		Check	
IPv6 header					
12+D+S	2	Type	86DDh	Compare	IP
14+D+S	1	Version/ Traffic Class	6Xh	Compare	Check IPv6
15+D+S	3	Traffic Class/Flow Label	-	Ignore	
18+D+S	2	Payload Length	-	Ignore	



Offset	# of bytes	Field	Value (hex)	Action	Comment
20+D+S	1	Next Header	IPv6 next header types	Check	The last header must be fragmented header in order for the header to be split.
21+D+S	1	Hop Limit	-	Ignore	
22+D+S	16	Source Address	-	Ignore	
38+D+S	16	Destination Address		Ignore	
54+D+S	N	Possible IPv6 Next Headers	-	Ignore	

In this case the packet is split after (54+D+S+N) bytes.

- The last next header field of the IP section field should not be 0x11/0x06 (TCP/UDP).

A.2.2.1.1 Type 2.2: Ethernet (VLAN/SNAP) Ipv6 UDP

This type contains only Ethernet header, Ipv6 header, and UDP header.

Offset	# of bytes	Field	Value (hex)	Action	Comment
0	6	Destination Address		Ignore	MAC Header – processed by main address filter, or broadcast filter.
6	6	Source Address		Ignore	
12	S=(0/4)	Possible VLAN Tag		Check	
12+S	D=(0/8)	Possible Len/LLC/SNAP Header		Check	
IPv6 header					
12+D+S	2	Type	86DDh	Compare	IP
14+D+S	1	Version/ Traffic Class	6Xh	Compare	Check IPv6
15+D+S	3	Traffic Class/Flow Label	-	Ignore	
18+D+S	2	Payload Length	-	Ignore	
20+D+S	1	Next Header	IPv6 next header types Or 0x11	Check	
21+D+S	1	Hop Limit	-	Ignore	
22+D+S	16	Source Address	-	Ignore	
38+D+S	16	Destination Address		Ignore	

Offset	# of bytes	Field	Value (hex)	Action	Comment
54+D+S	N	Possible IPv6 Next Headers	-	Ignore	
UDP header					
54+D+S+N	2	Source Port	Not (0x801)		Not NFS packet
56+D+S+N	2	Destination Port	Not (0x801)		Not NFS packet
58+D+S+N	2	Length	-		
60+D+S+N	2	Checksum	-		

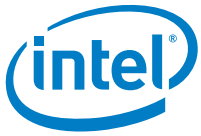
In this case the packet is split after (62+D+S+N) bytes.

- The last 'next-header' field of the last header of the IP section must be 0x06.

A.2.2.2 Type 2.3: Ethernet (VLAN/SNAP) Ipv6 TCP

This type contains only Ethernet header, Ipv6 header, and TCP header.

Offset	# of bytes	Field	Value (hex)	Action	Comment
0	6	Destination Address		Ignore	MAC Header – processed by main address filter, or broadcast filter.
6	6	Source Address		Ignore	
12	S=(0/4)	Possible VLAN Tag		Check	
12+S	D=(0/8)	Possible Len/LLC/SNAP Header		Check	
IPv6 header					
12+D+S	2	Type	86DDh	Compare	IP
14+D+S	1	Version/ Traffic Class	6Xh	Compare	Check IPv6
15+D+S	3	Traffic Class/Flow Label	-	Ignore	
18+D+S	2	Payload Length	-	Ignore	
20+D+S	1	Next Header	IPv6 next header types Or TCP	Check	
21+D+S	1	Hop Limit	-	Ignore	
22+D+S	16	Source Address	-	Ignore	
38+D+S	16	Destination Address		Ignore	



Offset	# of bytes	Field	Value (hex)	Action	Comment
54+D+S	N	Possible IPv6 Next Headers	-	Ignore	
TCP header					
54+D+S+N	2	Source Port	Not (0x801)	Check	Not NFS packet
56+D+S+N	2	Destination Port	Not (0x801)	Check	Not NFS packet
58+D+S+N	4	Sequence number	-	Ignore	
62+D+S+N	4	Acknowledge number	-	Ignore	
66+D+S+N	1/2	Header Length		Check	
66.5+D+S+N	1.5	Different bits	-	Ignore	
68+D+S+N	2	Window size	-	Ignore	
70+D+S+N	2	TCP checksum	-	Ignore	
72+D+S+N	2	Urgent pointer	-	Ignore	
74+D+S+N	F	TCP options	-	Ignore	

In this case the packet is split after (54+D+S+N+F) bytes.

- F = (TCP header length - 5) * 4.
- The last 'next-header' field of the last header of the IP section must be 0x11.

A.2.3 Type 3: Reserved

Type 3 used to be iSCSI packets (Header split is not supported for iSCSI packets in the 82599).

A.2.4 Type 4: NFS Packets

NFS headers can come in all the frames that contain UDP/TCP header. The NFS (and RPC headers) are extensions to these types of packets: 1.2, 1.3, 1.4.2, 1.4.3, 2.2, 2.3 that were presented in the previous sections. In this section only the NFS (and RPC) header is described and to its length should be added the length of the primary type of the packet.

the 82599 starts looking within the UDP/TCP payload to check whether it contain an NFS header if either the source or destination port of the TCP/UDP equal to 0x801.

```
Destination port equal to 0x801 => NFS write request => NFS write
request (as received by the NFS server).

Source port equal to 0x801       => NFS read request => read reply (as
received by the NFS client).
```

The VSZ/CSZ fields are 4 bytes long but there actual values are less than 2 words by definition, so hardware only checks the lower 2 bytes of these size fields.

RPC read requests are not described in this document since they contain only headers and no data -> no need to split.

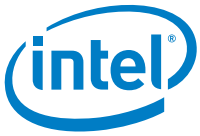
NFS over TCP is problematic - in fact, RPC header may appear in the middle of the frame. It remains to be checked if the software can support always putting RPC right next to the UDP/TCP header, but it doesn't have to.

A.2.4.1 Type 4.1: NFS Write Request

In all the write requests the destination port of the TCP/UDP header must be 0x801.

A.2.4.1.1 Type 4.1.1: NFS Write Request (NFSv2)

Offset	# of bytes	Field	Value (hex)	Action	Comment
RPC header					
0	D =(0/4)	Record Header	-	Ignore	If the previous header was a TCP header then this field contains 4 bytes.
0+D	4	Message type	0x00	Compare	
4+D	4	RPC version	0x02	Compare	
8+D	4	RPC program	0x18A63	Compare	
12+D	4	Program version	0x02	Compare	
16+D	4	Procedure	0x08	Compare	
20+D	4	Credentials Size (CSZ)	<400	Check	
24+D	B	Credentials Data	-	Ignore	B = (CSZ pad 4)
24+D+B	4	Verifier Flavor	-	Ignore	
28+D+B	4	Verifier Size (VSZ)	<400	Check	
32+D+B	F	Verifier Data		Ignore	F = (VSZ pad 4)
NFS header					
32+D+B+F	32	handle		Ignore	
64+D+B+F	4	begin offset		Ignore	
68+D+B+F	4	Offset		Ignore	
72+D+B+F	4	Total count		Ignore	
76+D+B_F	4	Data len		Ignore	



In this case the packet is split after (80+D+B+F) bytes should be added to the UDP/TCP type that was already parsed.

A.2.4.1.2 Type 4.1.2: NFS Write Request (NFSv3)

Offset	# of bytes	Field	Value (hex)	Action	Comment
RPC header					
0	D =(0/4)	Record Header	-	Ignore	If the previous header was a TCP header than this field contains 4 bytes.
0+D	4	Message type	0x00	Compare	
4+D	4	RPC version	0x02	Compare	
8+D	4	RPC program	0x18A63	Compare	
12+D	4	Program version	0x03	Compare	
16+D	4	Procedure	0x07	Compare	
20+D	4	Credentials Size (CSZ)	<400	Check	
24+D	B	Credentials Data	-	Ignore	B = (CSZ padded to 4)
24+D+B	4	Verifier Flavor	-	Ignore	
28+D+B	4	Verifier Size (VSZ)	<400	Check	
32+D+B	F	Verifier Data		Ignore	F = (VSZ padded to 4)
NFS header					
32+D+B+F	4	Fhandle_size	<64	Check	
36+D+B+F	S	fhandle		Ignore	S = (Fhandle_size padded to 4)
36+D+B+F+S	8	Offset		Ignore	
44+D+B+F+S	4	Count		Ignore	
48+D+B+F+S	4	Stable_how		Ignore	
52+D+B+F+S	4	Data len		Ignore	

In this case the packet is split after (56+D+B+F+S) bytes should be added to the UDP/TCP type that was already parsed.

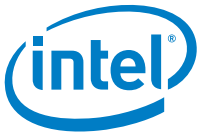
A.2.4.1.3 Type 4.1.3: NFS Write Request (NFSv4)

Offset	# of bytes	Field	Value (hex)	Action	Comment
RPC header					
0	D =(0/4)	Record Header	-	Ignore	If the previous header was a TCP header than this field contains 4 bytes.
0+D	4	Message type	0x00	Compare	
4+D	4	RPC version	0x02	Compare	
8+D	4	RPC program	0x18A63	Compare	
12+D	4	Program version	0x04	Compare	
16+D	4	Procedure	0x26	Compare	
20+D	4	Credentials Size (CSZ)	<400	Check	
24+D	B	Credentials Data	-	Ignore	B = (CSZ pad 4)
24+D+B	4	Verifier Flavor	-	Ignore	
28+D+B	4	Verifier Size (VSZ)	<400	Check	
32+D+B	F	Verifier Data		Ignore	F = (VSZ pad 4)
NFS header					
32+D+B+F	8	State id		Ignore	
40+D+B+F	8	Offset		Ignore	
48+D+B+F	4	Stable_how		Ignore	
52+D+B+F	4	Data len		Ignore	

In this case the packet is split after (100+D+F) bytes should be added to the UDP/TCP type that was already parsed.

A.2.4.1.4 Type 4.2.1: NFS Read Response (NFSv3)

Offset	# of bytes	Field	Value (hex)	Action	Comment
RPC header					
0	D =(0/4)	Record Header	-	Ignore	If the previous header was a TCP header than this field contains 4 bytes.
0+D	4	XID		Ignore	
4+D	4	Message type	0x01	Compare	



Offset	# of bytes	Field	Value (hex)	Action	Comment
8+D	4	Reply status	0x00	Ignore	'0' means OK and only if this value is '0' is there additional data.
12+D	4	Verifier Flavor	-	Ignore	
16+D	4	Verifier Size (VSZ)	<400	Check	
20+D	F	Verifier Data	-	Ignore	F = (VSZ pad 4)
20+D+F	4	Accept status	0x00	Ignore	'0' means OK
NFS header					
24+D+F	4	Status	0x00	Ignore	
28+D+F	68	Attributes	-	Ignore	
96+D+F	4	Data len	-	Ignore	

In this case the packet is split after (40+D+F+S) bytes should be added to the UDP/TCP type that was already parsed.

A.2.4.1.5 Type 4.2.1: NFS Read Response (NFSv4)

Offset	# of bytes	Field	Value (hex)	Action	Comment
RPC header					
0	D =(0/4)	Record Header	-	Ignore	If the previous header was a TCP header than this field contains 4 bytes.
0+D	4	XID	-	Ignore	
4+D	4	Message type	0x01	Compare	
8+D	4	Reply status	0x00	Ignore	'0' means OK and only if this value is '0' is there additional data.
12+D	4	Verifier Flavor	-	Ignore	
16+D	4	Verifier Size (VSZ)	<400	Check	
20+D	F	Verifier Data	-	Ignore	F = (VSZ pad 4)
20+D+F	4	Accept status	0x00	Ignore	'0' means OK
NFS header					
24+D+F	4	Status	0x00	Ignore	
	4	Attr_follow	-	Check	

Offset	# of bytes	Field	Value (hex)	Action	Comment
28+D+F	S	Attributes	-	Ignore	Attr_flow=1? S=84: S=0
28+D+F+S	4	Count	-	Ignore	
32+D+F+S	4	Eof	-	Ignore	
36+D+F+S	4	Data len	-	Ignore	

In this case the packet is split after (36+D+F) bytes should be added to the UDP/TCP type that was already parsed.

A.3 IPsec Formats Run Over the Wire

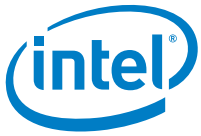
This section describes the IPsec packet encapsulation formats run over the wire by IPsec packets concerned with the off load in either Tx or Rx direction.

The following legend is valid for the figures Figure A-12 through Figure A-17 of this appendix.

Shaded fields correspond to the portion of the data that is protected by the integrity check.
Yellow colored fields are mutable fields that might be changed when traveling between the source and the destination and shall thus be zeroed when computing ICV or when encrypting/decrypting.
Cyan colored fields correspond to the portion of data that is protected for both integrity and confidentiality.
Non-colored fields are not protected either for integrity or for confidentiality.

A.3.1 AH Formats

- IPv4 header:
  - IP total length (2 bytes) - Total IP packet length in bytes, including IP header, AH header, TCP/UDP header, and TCP/UDP payload.
  - Protocol (1 byte) - AH protocol number, i.e. value 51.
- IPv6 header:
  - IP payload length (2 bytes) - IP payload length in bytes, including AH header, TCP/UDP header, and TCP/UDP payload.
  - Next header (1 byte) - AH protocol number, i.e. value 51.
- AH header:
  - Next header (1 byte) - Layer4 protocol number, 6 for TCP, 17 for UDP, etc.
  - AH length (1 byte) - Authentication Header length in 32-bits Dwords units, minus “2”, i.e. for AES-128 its value is 7 for IPv4 and 8 for IPv6.



  - — Reserved (2 bytes) - must be set to zero.
  - — SPI (4 bytes) - arbitrary 32-bits Security Parameters Index allocated by the receiver to identify the SA to which the incoming packet is bound. It is required that the local OS will allocate SPIs in a unique manner per local IP address.
  - — SN (4 bytes) - unsigned 32-bit Sequence Number that contains a counter value that increases by one for each Ethernet frame sent. It is initialized to 0 by the sender (and the receiver) when the SA is established, i.e. the first packet sent using a given SA will have a sequence number of 1.
  - — IV (8 bytes) - Initialization Vector to be used 'as is' in the nonce input to AES-128 crypto engine, but it must be zeroed prior to using it in the AAD input to the engine.
  - — ICV (16 bytes) - Integrity Check Value for this packet, authentication tag output of the AES-128 crypto engine. As being part of the AH header, this field is also included in the AAD input to the crypto engine, and it should be zeroed prior to the computation.
  - — ICV Padding (4 bytes) - *explicit* padding bytes appended to the ICV field in IPv6, as it is required to maintain the (Authentication) extension header length as a multiple of 64-bits. By *explicit* we mean that these bytes are sent over the wire. It is formed by 4 arbitrary bytes that need not be random to achieve security. For TSO, it will be replicated from the header provided by the driver in every frame.
- L4 header (for example - TCP/UDP): Length (in bytes) depend on the protocol.
- L4 payload (for example - TCP/UDP): Can be any length in bytes

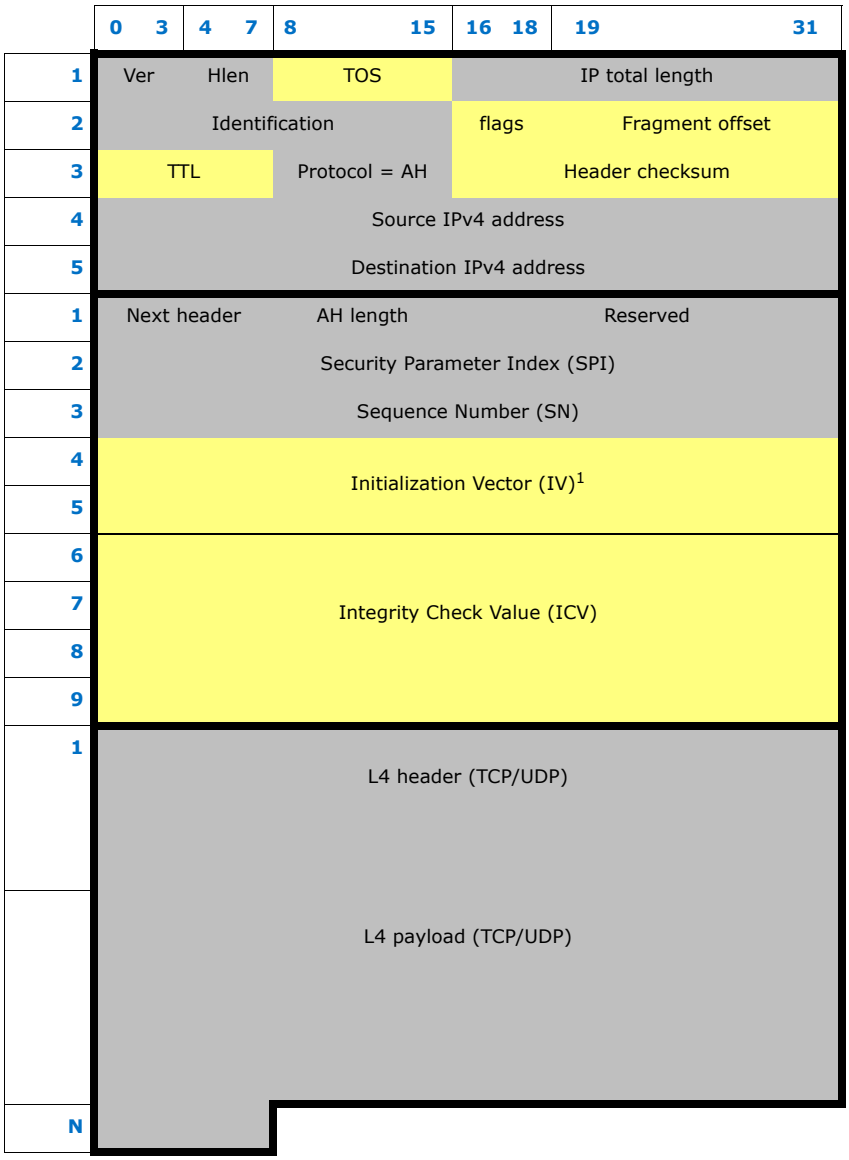


Figure A-12 AH packet over IPv4

1. IV field has been colored in Yellow as it must be zeroed in the AAD input to AES-128 crypto engine, in spite of this it is NOT zeroed in the nonce input to the engine.

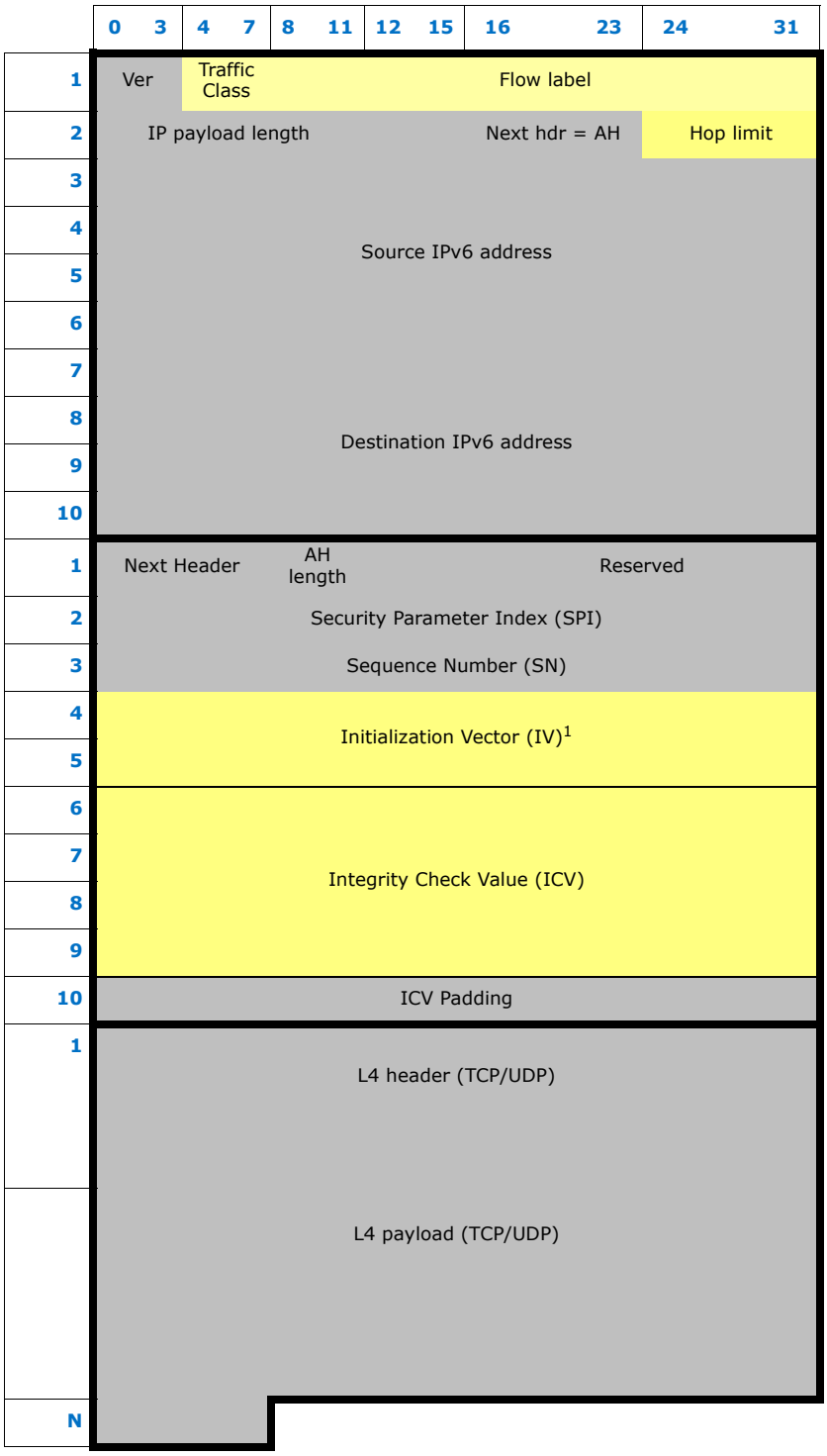


Figure A-13 AH packet over IPv6

1. IV field has been colored in Yellow as it must be zeroed in the AAD input to AES-128 crypto engine, in spite of this, it is NOT zeroed in the nonce input to the engine.

A.3.2 ESP Formats

- IPv4 header:
  - IP total length (2 bytes) - Total IP packet length in bytes, including IP header, ESP header, TCP/UDP header, TCP/UDP payload, ESP trailer, and ESP ICV if present.
  - Protocol (1 byte) - ESP protocol number, i.e. value 50.
- IPv6 header:
  - IP payload length (2 bytes) - IP payload length in bytes, including ESP header, TCP/UDP header, TCP/UDP payload, ESP trailer, and ESP ICV if present.
  - Next header (1 byte) - ESP protocol number, i.e. value 50.
- ESP header:
  - SPI (4 bytes) - arbitrary 32-bit Security Parameters Index allocated by the receiver to identify the SA to which the incoming packet is bound. It is required that the local OS will allocate SPIs in a unique manner per local IP address.
  - SN (4 bytes) - unsigned 32-bit Sequence Number that contains a counter value that increases by one for each Ethernet frame sent. It is initialized to 0 by the sender (and the receiver) when the SA is established, i.e. the first packet sent using a given SA will have a sequence number of 1.
  - IV (8 bytes) - Initialization Vector to be used in the nonce input field of the AES-128 crypto engine, and for authenticated-only ESP packets it is used also in the AAD input.
- L4 header (for example - TCP/UDP):
  - Length (in bytes) depend on the protocol.
  - TCP/UDP checksum computed from the TCP/UDP header up to the end of TCP payload, excluding ESP trailer.
  - TCP/UDP header is encrypted if ESP encryption is required.
- L4 payload (for example - TCP/UDP): Can be any length in bytes. It is encrypted if ESP encryption is required.
- ESP trailer:
  - Padding (0-255 bytes) - unsigned 1-byte integer values, with its content started by 1 and making up a monotonically increasing sequence: 1, 2, 3,... Though in Tx it will only be 0-15 bytes long, in Rx it might be longer, if the sender's policy is to hide the packet length.
  - Padding length (1 byte) - Number of explicit padding bytes (Padding length and Next header bytes excluded) required to get 4-bytes alignment of the ESP header, TCP/UDP header, TCP/UDP payload, and ESP trailer. By explicit we mean that these bytes are sent over the wire. A remote IPsec implementation may also add more padding bytes (up to 255-bytes) than the minimum required for getting the 4-bytes alignment with the aim of hiding the packet length.
  - Next header (1 byte) - Layer4 protocol number, 6 for TCP, 17 for UDP, etc.
  - ESP trailer will be encrypted if ESP encryption is required.

- ESP ICV (16 bytes) - Integrity Check Value for this packet, authentication tag output of the AES-128 crypto engine.

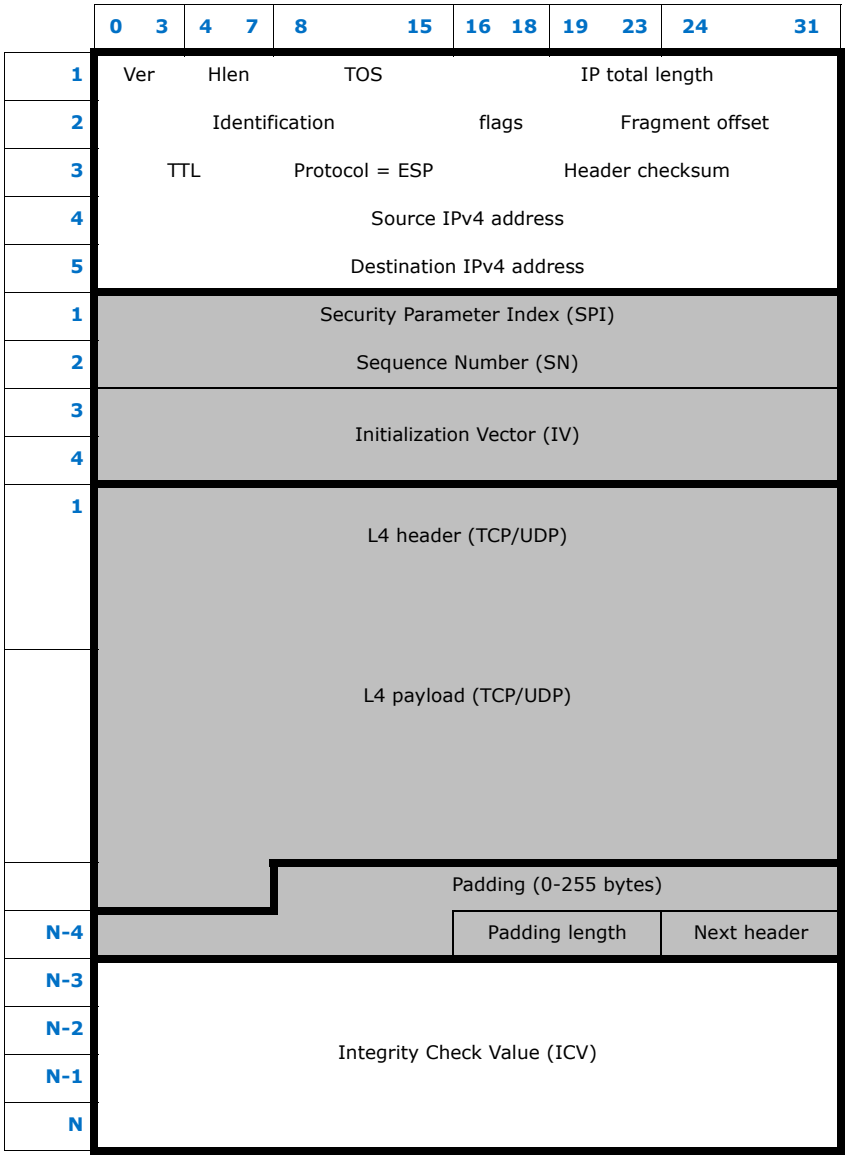


Figure A-14 Authenticated only ESP packet over IPv4

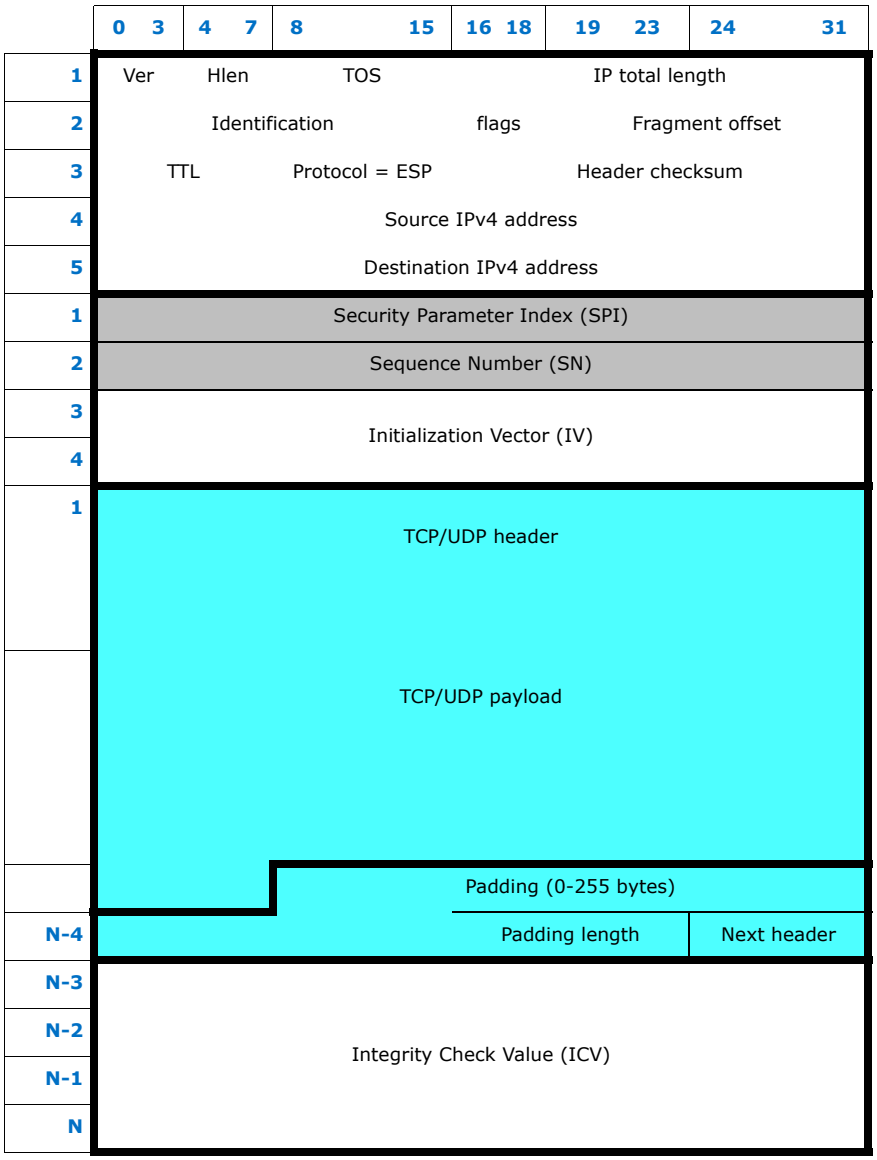


Figure A-15 Authenticated & encrypted ESP packet over IPv4

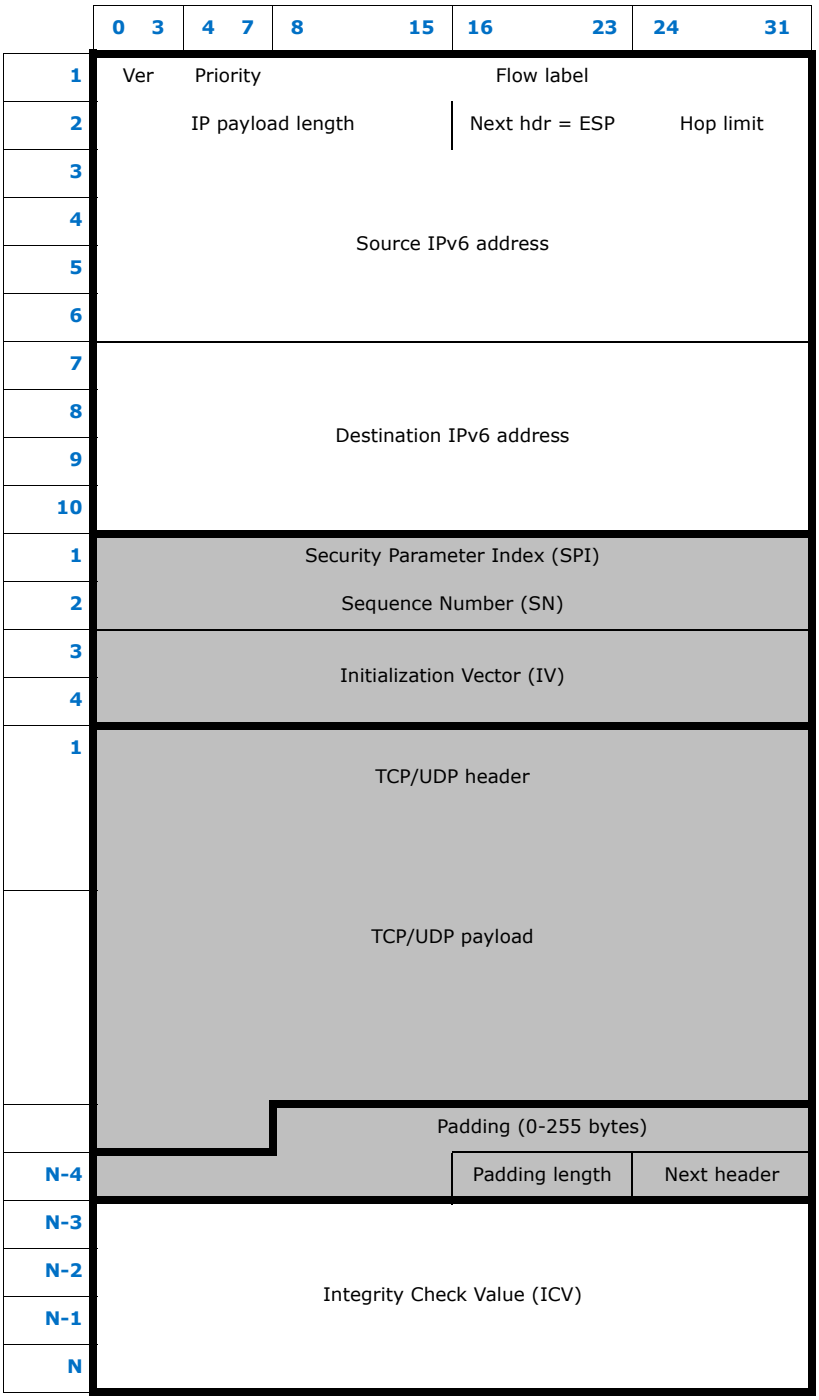


Figure A-16 Authenticated only ESP packet over IPv6

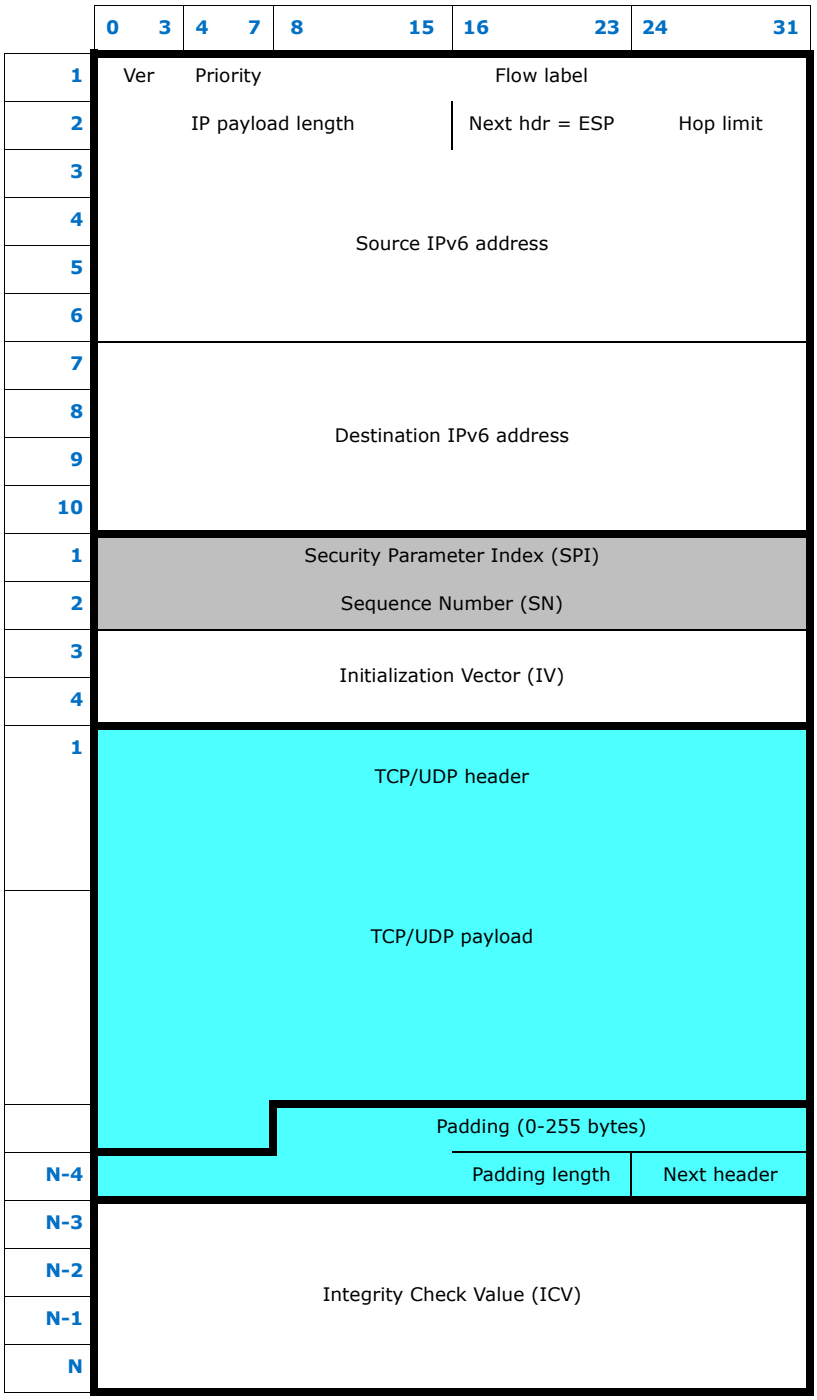


Figure A-17 Authenticated and encrypted ESP packet over IPv6

For authenticated and encrypted ESP packets, though it is used in the Nonce input to the AES-128 crypto engine, the IV field was left non-colored because it is not a part of the ADD or the Plaintext input fields. Refer to Section 7.12.7.

A.4 BCN Frame Format

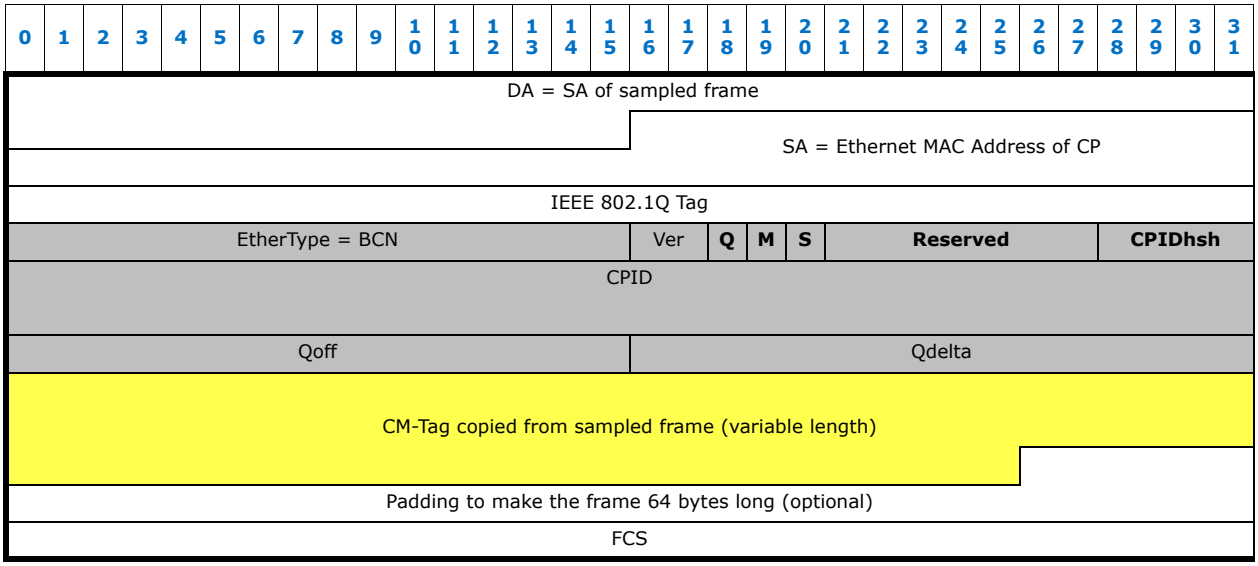


Figure A-18 BCN frame format

- 802.1Q Tag is copied from the sampled frame. The 802.1p priority field is set either to the priority of the sampled frame or to a configurable priority. It is preferable to use highest priority or the priority value corresponding to the network management/ control traffic in order to minimize the latency experienced by BCN frames.
- EtherType = TBD, is programmable via the ETQF register, because IEEE802.1au Standard has not assigned a value yet. It is different from the EtherType included in the CM-tags.
- Ver stands for the Version of the BCN protocol.
- Special congestion conditions, Q,M, and S bits:
  - Q-bit indicates that the Qdelta field has saturated.
  - M-bit indicates that a Mild congestion threshold was passed.
  - S-bit indicates that a Severe congestion threshold was passed.
- CPIDhsh is a hash tag computed by the Congestion Point over its own CPID. It will be used by the CP when returned in a rate-limited frame to determine whether a positive BCN control frame shall be sent for that flow - if the congested conditions disappeared. If the CPIDhsh hash tag does not match with the congestion point's CPID, for sure the rate-limited frame was not associated with that congestion point, and thus no positive BCN frame will be sent for that flow by the congested point.
- CPID stands for Congestion Point IDentifier, which should include the congestion point Ethernet MAC Address, as well as a local identifier for the local congestion entity, usually a queue in the switch.

- Qoff and Qdelta contains the queue filling status at the congestion point, where Qoff is the algebraic offset of the current queue length with respect to the equilibrium threshold Qeq, and Qdelta is the change in length of the queue since the last sampled frame. Both are expressed in pages of 64-bytes.
- FCS is the Frame Check Sequence of Ethernet frames.

Note: The shaded fields shall be inserted just after the 802.1Q VLAN tag, and thus it can be located as follow:

1. MAC DA, SA
2. LinkSec tag - all data after this tag is encrypted.
3. Double VLAN - follows the same rules described above for 802.1Q tag.
4. 802.1Q VLAN
5. Shaded fields

A.5 FCoE Framing

Related Standards

- FC-FS-2 - FRAMING AND SIGNALING-2 (FC-FS-2) Rev 1.00
- FCoE - Fibre Channel over Ethernet Draft Presented at the T11 on May 2007

A.5.1 FCoE Frame Format

FC over Ethernet packets encapsulate FC frames as shown in Figure A-19 and Figure A-20. Maximum expected FCoE frame size is 2164 bytes. This size does not include FC extension headers (not expected in FCoE), without optional LinkSec encapsulation (expected to be off-loaded by the hardware) and without BCN tag (not expected in receive).

All fields in the FCoE frame are treated as any other field in the network. The MS Byte is first on the wire and the LS bit on each byte is first on the wire. This rule applies for all fields including those ones that span on non complete byte boundaries such as the FCoE VER field.

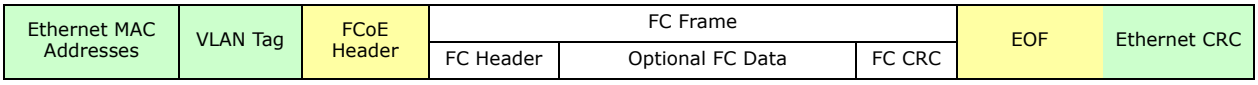
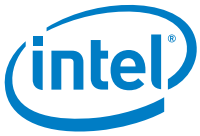


Figure A-19 Ethernet Encapsulation to FC frames



	msb 31 lsb 24 First byte on the wire	msb 23 lsb 16	msb 15 lsb 12	11 lsb 8	msb 7 ... . . lsb 0 Last byte on the wire
0	Destination Ethernet MAC Address				
4					
8	Source Ethernet MAC Address				
12	802.1Q Tag (VLAN + Priority)				
16	FCoE Ethernet Type = 0x8906		Ver	Reserved	
20	Reserved				
24	Reserved				
28	Reserved				SOF
32	Routing Control (R_CTL)	Destination Identification (D_ID)			
36	Class Specific Control (CS_CTL)	Source Identification (S_ID)			
40	TYPE	Frame Control (F_CTL)			
44	Sequence ID (SEQ_ID)	Data Field Control (DF_CTL)	Sequence Count (SEQ_CNT)		
48	Originator Exchange ID (OX_ID)		Responder Exchange ID (RX_ID)		
52	Parameter (PARAM)				
0...N	Optional FC Data... always 4 byte align (including optional FC padding)				
56 +N	Fibre Channel CRC (FC_CRC)				
N+8	EOF	Reserved			
M	Ethernet CRC				

Figure A-20 FCoE Packet Structure

A.5.1.1 Ethernet MAC Addresses

L2 destination and Source Ethernet MAC Addresses (each of them is 6 bytes long). The Ethernet MAC Address of the target is assumed to be assigned by the network. It could be done by the FCoE repeater or LAN administrator. The mechanism that is used for Ethernet MAC Address assignment and Ethernet MAC Address detection is outside of the scope of this document.

A.5.1.2 802.1Q Tag

802.1Q tagging is mandatory for FCoE usage. FCoE assumes Reedtown functionality which provides class-based FC and BCN. These functions require 802.1Q tag presence to define packet priority.

A.5.1.3 FCoE Header

The FCoE header is composed of a new FCoE Ethernet type, FCoE version tag and the Start Of Frame tag.

FCoE Ethernet Type

Equals 0x8906

Version (Ver)

A 4 bit field that indicates the FCoE protocol version number. the 82599 supports FCoE version as defined by FCRXCTRL.FCOEVER.

Start of Frame (SOF)

The FCoE Start of frame is a subset of the FC-FS-2 SOF codes as defined in the Table below:

Table A-1 E_SOF Mapping

FC SOF	FCoE SOF Code	FC Traffic Class	Comment
SOFf	0x28	F	Fabric start of frame. Not expected in an FCoE.
SOFi2	0x2D	2	used in the first frames in a sequence
SOFn2	0x35	2	used in all but first frame in a sequence
SOFi3	0x2E	3	used in the first frames in a sequence
SOFn3	0x36	3	used in all but first frame in a sequence

A.5.1.4 FCoE Packet Encapsulation Trailer

The FCoE trailer is composed of an End of frame, optional padding and Ethernet CRC.

End of Frame (EOF):

The FCoE End of frame maps the FC-FS-2 EOF codes as defined in Table A-2:

Table A-2 EOF Mapping

FC EOF	FCoE EOF Code	FC Traffic Class	Comment
EOFn	0x41	2, 3, 4, F	normal EOF
EOFt	0x42	2, 3, 4, F	EOF Terminate used to close a sequence
EOFni	0x49	2, 3, 4, F	EOF Invalid indicating that the frame content is invalid.
EOFa	0x50	2, 3, 4, F	EOF Abort

Ethernet CRC:

The IEEE 802.3 CRC as defined by the following polynomial:
$X_{32}+X_{26}+X_{23}+X_{22}+X_{16}+X_{12}+X_{11}+X_{10}+X_8+X_7+X_5+X_4+X_2+X+1$

A.5.2 FC Frame Format

Note: This section is provided as a background on FC and is not required for the HW implementation. For a complete description of the FC fields please refer to FC-FS-2 specification.

The FC frame as defined in FC-FS-2 specification is shown in the Figure A-21 below while relevant fields are detailed in this section.

Figure A-21 FC Frame Format

A.5.2.1 FC SOF and EOF

FC Start of frame (SOF) delimiter and End of frame (EOF) delimiter. In FCoE frames, the SOF and EOF fields in the FC frame are extracted and reflected in the FCoE encapsulation. The SOF and EOF codes that are reflected in the FCoE framing are shown in Table and Table .

A.5.2.2 FC CRC

The Cyclic Redundancy Check (CRC) is a four byte field that follows the Data Field. It enables end to end integrity checking on the whole FC frame. The HW adds this field if the FCoE bit is set in the transmit context descriptor. The FC CRC off load is described in more detail in Section 7.13.3.2.

A.5.2.3 FC Header

The FC header fields are provided in the header buffer by the FCoE driver. The FC header includes fields that are modified by the HW as part of Large Send off load. These fields are indicated in this section as “Dynamic” while fields that are not modified by the HW are indicated as “Static”.

Routing Control (R_CTL)

The R_CTL is a one-byte Static field that contains routing and information bits to categorize the frame function.

Class Specific Control (CS_CTL)

This is a 1 byte Static field that defines either the Class specific control or priority according to bit 17 in the Frame control (F_CTL) field.

Destination Identification (D_ID)

The D_ID is a 3 byte Static field that defines the FC destination address.

Source Identification (S_ID)

The S_ID is a 3 byte Static field that defines the FC source address.

Data Structure Type (TYPE)

The TYPE is a 1 byte Static field that identifies the protocol of the frame content for data frames.

Frame Control (F_CTL)

The F_CTL is a 3 byte Dynamic field that contains control information relating to the frame content. The F_CTL is further described in Section A.5.2.3.1. Section 7.13.2.7 describes how the F_CTL field is modified during large send.

Data Field Control (DF_CTL)

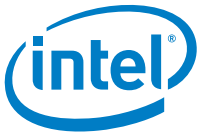
The DF_CTL is a 1 byte Dynamic field that specifies the presence of optional headers at the beginning of the Data_Field. The Optional headers supported by large send are present only on the first frame in the FC sequence. Section 7.13.2.7.1 describes how the DF_CTL field is modified during large send.

Sequence ID (SEQ_ID)

The SEQ_ID is a 1 byte Static number associated with a sequence. A sender must assign SEQ_ID numbers so that the recipient would always be able to distinguish between consecutive sequences. SEQ_ID do not have to be sequential and do not have to be unique even within the same IO exchange as long as it is guaranteed that the recipient would be able to distinguish between the them.

Sequence Count (SEQ_CNT)

The SEQ_CNT is a 2 byte Dynamic field that indicates the sequential order of Data frame transmission within a single Sequence or multiple consecutive Sequences for the same Exchange. The SEQ_CNT of the first Data frame of the first Sequence of the Exchange transmitted by either the Originator or Responder is ‘0’. The SEQ_CNT of subsequent Data frames in the Sequence is incremented by one for each data frame. The SEQ_CNT of the first Data frame in each sequence other than the first one can start at ‘0’ or be incremented by 1 from the last used SEQ_CNT.



Originator Exchange ID (OX_ID)

The OX_ID is a two-byte **S**tatic field that identifies the Exchange_ID assigned by the Originator. If the Originator is enforcing uniqueness via the OX_ID mechanism, it shall set a unique value for OX_ID other than FFFFh. A value of FFFFh indicates that the OX_ID is unassigned and that the Originator is not enforcing uniqueness via the OX_ID. the 82599 supports large receive and direct data placement only if OX_ID is used to identify uniqueness.

Responder Exchange ID (RX_ID)

The RX_ID is a two byte Static field assigned by the Responder that shall provide a unique, locally meaningful exchange identifier at the Responder. The Responder of the Exchange shall set a unique value for RX_ID other than FFFh.

Parameter (PARAM)

The Parameter field is a Dynamic field which is based on frame type. For Data frames with the relative offset present bit set to 1, the Parameter field specifies relative offset. The offset defines the relative displacement of the first byte of the Payload of the frame from the base address as specified by the ULP. Relative offset is expressed in terms of bytes.

A.5.2.3.1 Frame Control (F_CTL)

The Frame Control (F_CTL) is a three-byte field that contains control information relating to the frame content. If an error in bit usage is detected, the SW initiates a reject frame (P_RJT) in response with an appropriate reason code (FCoE SW driver responsibility). The F_CTL format is shown in Table A-3 below.

When a bit(s) is designated as “Static” it is provided by the FCoE driver as part of the large send header. The HW keeps this bit(s) as is in all preceding frames of the large send.

When a bit(s) is designated as “Dynamic” it is provided by the FCoE driver as part of the large send header. The HW may change it in some of the packets in the large send as described in Section 7.13.2.7. Exact setting of these bit(s) is described in the Table A-3 below.

When a bit(s) is designated as meaningful under a set of conditions, that bit shall be ignored if those conditions are not present.

Table A-3 F_CTL Format

Control Field	bit	Type	Description.
Exchange Context	23	Static	0b = Originator of Exchange 1b = Responder of Exchange
Sequence Context	22	Static	0b = Sequence Initiator 1b = Sequence Recipient
First Sequence	21	Static	0b = Sequence other than first of Exchange 1b = First Sequence of Exchange

Table A-3 F_CTL Format (Continued)

Control Field	bit	Type	Description.
Last Sequence	20	Static	0b = Sequence other than last of Exchange 1b = Last Sequence of Exchange Must be set on the last data frame of the last sequence. However it can be set on any preceding frames. Once it is set, it must be set on all frames till the last one of that exchange.
End Sequence	19	Dynamic	0b = Data frame other than last of Sequence 1b = Last Data frame of Sequence
End Connection	18	Static	Relevant for Class 1 or 6
CS CTL / Priority Enable	17	Static	Defines the meaning of the CS_CTL byte in the FC header to be either CS_CTL or Priority indication. 0b = 0Word 1, Bits 31-24 = CS_CTL 1b = Word 1, Bits 31-24 = Priority
Sequence Initiative	16	Dynamic	This bit is used to transfer initiative from a sender to a recipient. This bit is meaningful only on the last frame of a sequence (when bit 19 - "End Sequence" is set). 0b = Hold Sequence initiative 1b = Transfer Sequence initiative
X_ID reassigned	15	Static	Obsolete.
Invalidate X_ID	14	Static	Obsolete.
ACK Form	13:12	Static	ACK Form is meaningful on all Class 1, Class 2, or Class 6 Data frames of a Sequence and on all connect request frames. ACK_Form is not meaningful on Class 1, Class 2, or Class 6 Link_Control frames, or any Class 3 frames. 00b = No assistance provided 10b = Reserved 01b = Ack_1 Required 11b = Ack_0 Required
Data Compression	11	Static	Obsolete.
Data Encryption	10	Static	Obsolete.
Retransmitted Sequence	9	Static	Meaningful in Class 1 and 6 and only. 0b = Original Sequence transmission 1b = Sequence retransmission
Unidirectional Transmit	8	Static	Relevant for Class 1.
Continue Sequence Condition	7:6	Dynamic	Last Data frame - Sequence Initiator 00b = No information 01b = Sequence to follow-immediately 10b = Sequence to follow-soon 11b = Sequence to follow-delayed It is meaningful only when the "End Sequence" is set to '1' and the "Sequence Initiative" bit is cleared '0'.

Table A-3 F_CTL Format (Continued)

Control Field	bit	Type	Description.
Abort Sequence Condition	5:4	Static	ACK frame - Sequence Recipient 00b = Continue sequence 01b = Abort Sequence, Perform ABTS 10b = Stop Sequence 11b = Immediate Sequence re-XMT requested Data frame (1st of Exchange) - The Abort Sequence shall be set to a value by the Sequence Initiator on the first Data frame of an Exchange to indicate that the Originator is requiring a specific error policy for the Exchange. 00b = Abort, Discard multiple Sequences 01b = Abort, Discard a single Sequence 10b = Process policy with infinite buffers 11b = Discard multiple Seq with immediate re-XMT
Relative offset present	3	Static	0b = Parameter field defined for some frames 1b = Parameter Field = relative offset
Exchange reassembly	2	Static	Reserved for Exchange reassembly
Fill Bytes	1:0	Dynamic	Defines the number of FC padding bytes to fill in the last frame in the sequence. The value of these bytes is 0x00. When FCoE offload is enabled (TUCMD.FCoE bit is set in the transmit context descriptor), the hardware can pad the FC payload according to the buffer size indicated by software (by the PAYLEN field in the transmit data descriptor).

A.5.2.4 FC Extended Headers

Note: The following section is provided as a background on FC and is not required for the HW implementation or the SW implementation since in FCoE frames, extended headers are not expected. The following quote is from the FCoE spec Rev 0.7 "No Extended Headers are relevant for the operations of a Reedtown NIC".

The diagrams in Figure A-22, Figure A-23 and Table A-4 show the Fibre Channel frame structure with Extended headers and lists the existing headers. Extended headers are identified by the R_CTL value as shown in the diagrams below. The LAN controller does not support these Extended headers since they are not expected in FCoE usage.

FC Frame				
Extended Headers [Optional]	FC Header	Optional FC Header	[optional] FC Data (including optional padding)	FC CRC

Figure A-22 FC Frame format with Extended Headers

	31 24 Last bytes on the wire	23 16	15 8	7 0 First bytes on the wire
0	Extended Header Specific Fields			R_CTL
4...				

Figure A-23 Extended Header Format

Table A-4 FC Extended Headers

R_CTL	Extended Header	Length
0x50	VFT_Header (Virtual Fabric Tagging Header)	8 Bytes
0x51	IFR_Header (Inter-Fabric Routing Header)	8 Bytes
0x52	Enc_Header (Encapsulation Header)	24 Bytes
0x53…0x5F	Reserved	-

A.5.2.5 FC Optional Headers

Note: Most of the following section is provided as a background on FC and is not required for the HW implementation. The reader may skip the detailed explanation of the Optional headers provided below and concentrate in the tables and figures that follow the text.

FCoE frames may include FC Optional headers. These headers (if they exist) would always show in the first frame in a sequence. While FC implementation may include the FC Optional headers only on the first frame in a sequence. In Large send functionality, the FC Optional headers may show only on the first frame as shown in Figure 7-47 and Table A-5 (in Section 7.13.2.7.1).

The following diagrams Table A-5, Figure A-24 and Figure A-25 show the Fibre Channel frame structure with Optional headers and lists the Optional headers. The Optional headers (that are present) are always ordered as shown in Figure A-24 and Figure A-25. Their presence is indicated in the Data Field Control (DF_CTL) field in the FC Header as indicated in Table A-5

Maximum FC frame size: The sum of the length in bytes of the FC Payload, the number of fill bytes, and the lengths in bytes of all optional headers shall not exceed 2112.

Table A-5 FC Optional Headers

DF_CTL	Optional Header	Length
bit 6	ESP Header / ESP Trailer	Variable
bit 5	Network Header	16 Bytes
bit 4	Association Header	32 Byte
bits 1:0	Device Header	0, 16, 32, or 64 Bytes

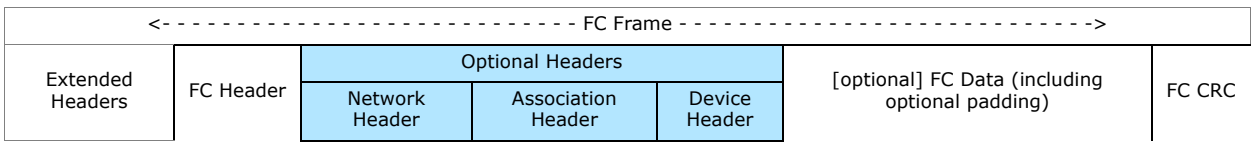


Figure A-24 FC Frame format with Optional Headers (without ESP Header)

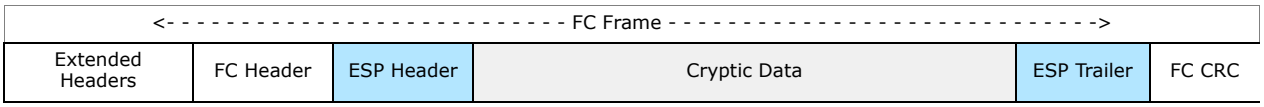


Figure A-25 FC Frame format with Optional Headers (with ESP Header)

ESP Header:

This is the first Optional header which covers the whole FC frame other than the FC header which is transmitted on the clear (as plain text). When an ESP header is present there is also the ESP trailer. If required, the SW is responsible for the cryptic calculation and preparing the ESP header and trailer. Its presence is indicated by bit 6 in the DF_CTL field being set to one. The hardware does not support Large Send off load when ESP Optional header is used. When present, the ESP header and trailer are present in all frames of the exchange.

Network Header:

The Network Header, if used, shall be present only in the first Data frame of a Sequence. A bridge or a gateway node that interfaces to an external Network may use the Network Header. The Network Header, is an optional header 16 Bytes long within the FC Data Field content. Its presence is indicated by bit 5 in the DF_CTL field being set to one. The Network Header may be used for routing between Fibre Channel networks of different Fabric address spaces, or Fibre Channel and non-Fibre Channel networks. The Network Header contains Name Identifiers for Network Destination Address and Network Source Address.

Association Header:

The Association Header, if used, shall be present only in the first Data frame of a Sequence. The Association Header is an optional header of 32 Bytes long within the Data Field content. Its presence is indicated by bit 4 in the DF_CTL field being set to one. The Association Header may be used to identify a specific Process or group of Processes within a node associated with an Exchange. When an Nx_Port has indicated during Login that an Initial Process Associator is required to communicate with it, the Association Header should be used by that Nx_Port to identify a specific Process or group of Processes within a node associated with an Exchange. the 82599 does not use the Association for any filtering purposes but rather uses the OX_ID.

Device header:

The Device Header, if present, shall be present either in the first Data frame or in all Data frames of a Sequence. If Large Send off load is used then the Device header, if present, is present only in the first frame of the same large send. The Device Header, if present, shall be 16, 32, or 64 bytes in size as defined by bits 1:0 in the DF_CTL field. The contents of the Device Header are controlled at a level above FC-2. Upper layer protocol (ULP) may use a Device Header, requiring the Device Header to be supported. The Device Header may be ignored and skipped, if not needed. If a Device Header is present for a ULP that does not require it, the related FC-4 may reject the frame with the reason code of “TYPE not supported”.

NOTE: ***This page intentionally left blank.***

Appendix B LESM - Link Establishment State Machine for the 82599

B.1 Background

"Legacy" XAUI-based switches developed prior to the IEEE 802.3ap standard TX only in one lane (Lane 0) during link detection. Typically, these devices only transition to a XAUI-like 10 GbE link when all four pairs of their receivers is active.

Additionally, IEEE 802.3ap compliant devices such as the Intel 82599 controller are required to transmit auto-negotiation only on Lane 0 per Clause 73.3, and the Intel device only parallel-detects a XAUI-like 10 GbE link when all four pairs of their receivers are active. Therefore, a speedlock condition can occur when the 82599 device is connected to a legacy XAUI-based switch, since both devices are capable of 10 GbE XAUI-like parallel detection, but only the lane 0 transmitters on each device are active -- one device needs to turn on all four transmitters for the other device to see 10 GbE XAUI-like mode. Otherwise, either no link, or a 1 GbE link is observed in the system, depending on the specific behavior of the switch link state machine.

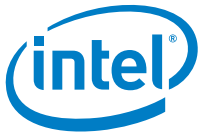
The LESM (link establishment state machine) was developed by Intel to break the speedlock condition described above. The feature can be implemented in the 82599 controller with on-chip firmware, and is used to switch the link-mode-select setting in the AUTOC register to try a different configuration after timeout. For example, after trying CL 73 AN and Parallel Detect, it may change to XAUI-mode (which turns on all four lane transmitters) and check link status.

B.2 Location in the NVM

The LESM module in NVM includes the parameters used by LESM and is pointed by the Firmware Module offset 0x2 (see Section 6.4) and is placed instead the unused No Manageability Patch.

Table B-1 Firmware module

Global MNG Word Offset	Description
0x0	Test Configuration Pointer - Section 6.4.1
0x1	Reserved
0x2	LESM Module Pointer
0x3	Common Firmware Parameters - Section 6.4.2
0x4	Pass Through Patch Configuration Pointer (Patch structure identical to the Loader Patch) - Section 6.4.3
0x5	Pass Through LAN 0 Configuration Pointer - Section 6.4.3
0x6	SideBand Configuration Pointer - Section 6.4.4
0x7	Flexible TCO Filter Configuration Pointer - Section 6.4.5
0x8	Pass Through LAN 1 Configuration Pointer - Section 6.4.3
0x9	NC-SI Microcode Download Pointer - Section 6.4.6
0xA	NC-SI Configuration Pointer - Section 6.4.7



NOTE: ***This page intentionally left blank.***

LEGAL