US 20090279441A1

(19) United States

(12) Patent Application Publication

(10) Pub. No.: US 2009/0279441 A1

(43) Pub. Date: Nov. 12, 2009

Wong et al.

(54) **TECHNIQUES FOR TRANSMITTING FAILURE DETECTION PROTOCOL PACKETS**

(75) Inventors: **Yuen Wong**, San Jose, CA (US); **Pedman Moobed**, San Jose, CA (US)

Correspondence Address:
TOWNSEND AND TOWNSEND AND CREW, LLP
TWO EMBARCADERO CENTER, EIGHTH FLOOR
SAN FRANCISCO, CA 94111-3834 (US)

(73) Assignee: **Foundry Networks, Inc.**, Santa Clara, CA (US)

(21) Appl. No.: **11/953,745**

(22) Filed: **Dec. 10, 2007**

Related U.S. Application Data

(60) Provisional application No. 60/880,074, filed on Jan. 11, 2007.

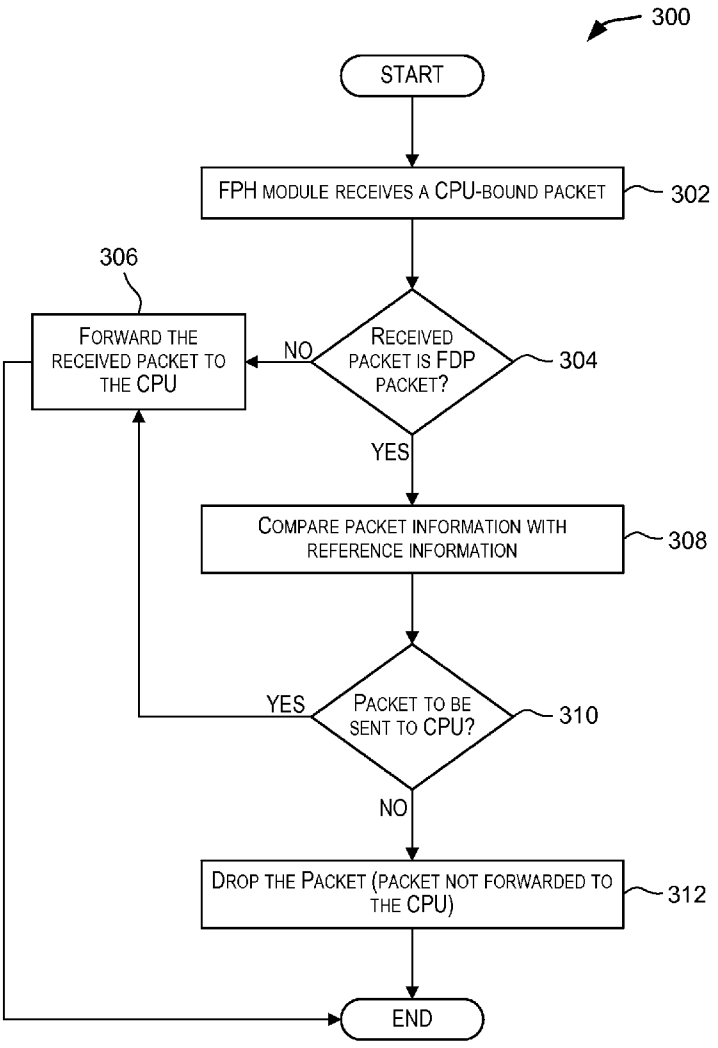
Publication Classification

(51) **Int. Cl.**
G01R 31/08 (2006.01)

(52) **U.S. Cl.** **370/242**

(57) **ABSTRACT**

Techniques are provided for processing of failure detection protocol (FDP) packets. Techniques are provided that assist a CPU of a network device in processing incoming FDP packets. The task of transmitting FDP packets from a network device is offloaded from the CPU of the network device and instead handled by another module of the network device. In this manner, the processing that the CPU of the network device has to perform for transmitting FDP packets for the various FDP sessions of the network device is reduced. This enables the network device to support newer FDPs with shorter periodic interval requirements.



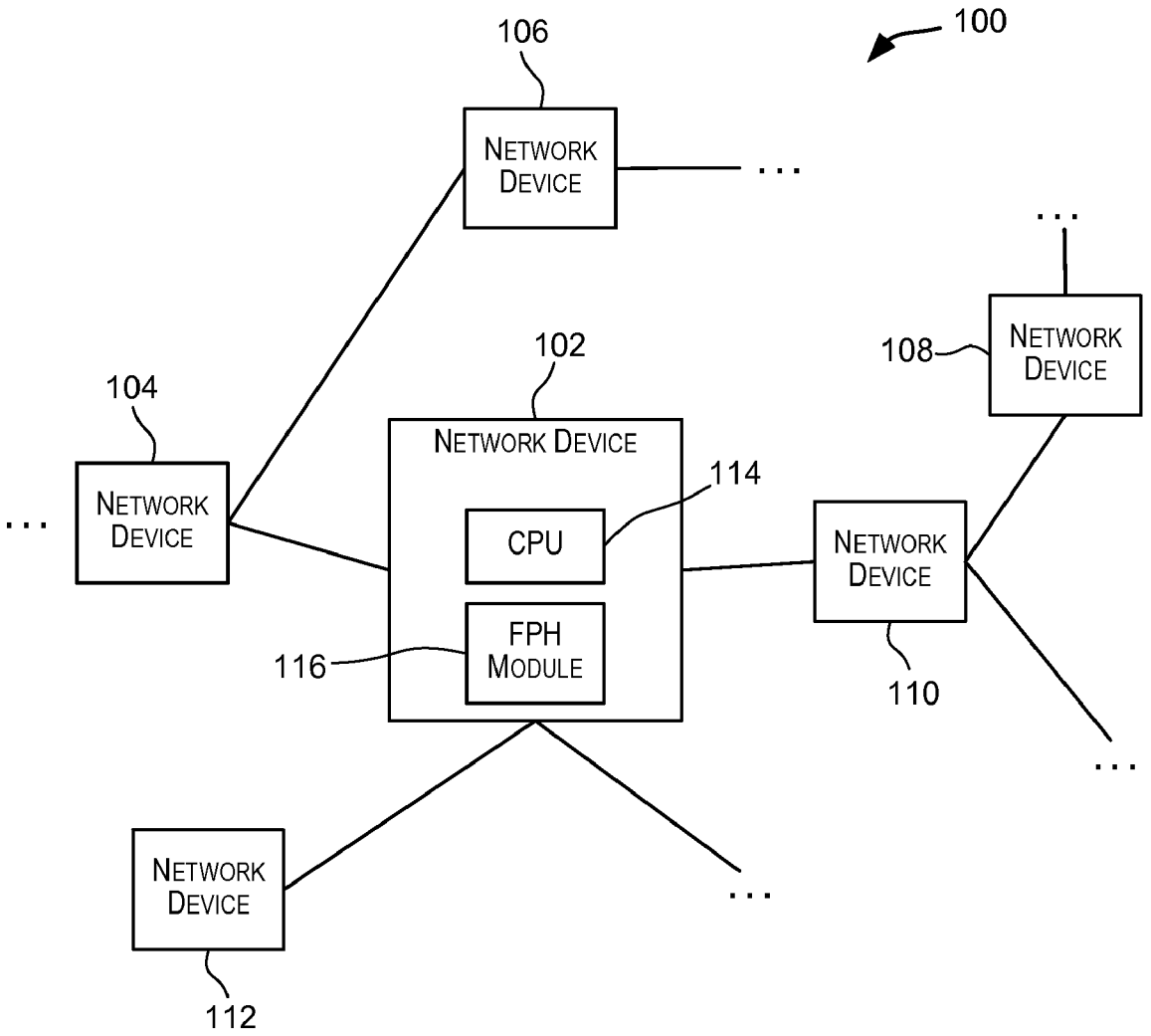

100
106
NETWORK DEVICE
102
NETWORK DEVICE
CPU
FPH MODULE
114
116
104
NETWORK DEVICE
108
NETWORK DEVICE
NETWORK DEVICE
110
NETWORK DEVICE
112

FIG. 1

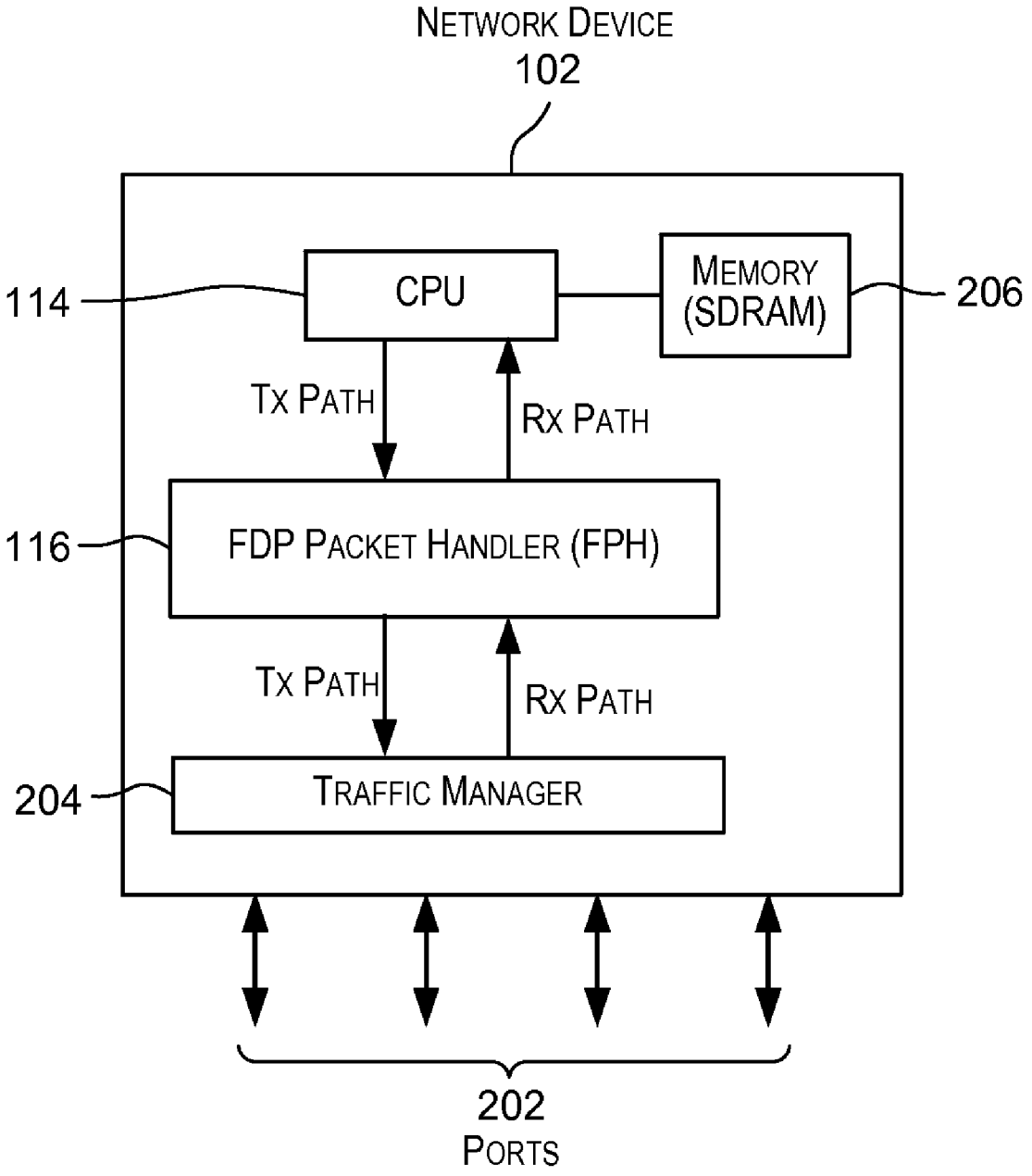

NETWORK DEVICE
102
114
CPU
MEMORY
(SDRAM)
206
TX PATH
RX PATH
116
FDP PACKET HANDLER (FPH)
TX PATH
RX PATH
204
TRAFFIC MANAGER
202
PORTS

FIG. 2

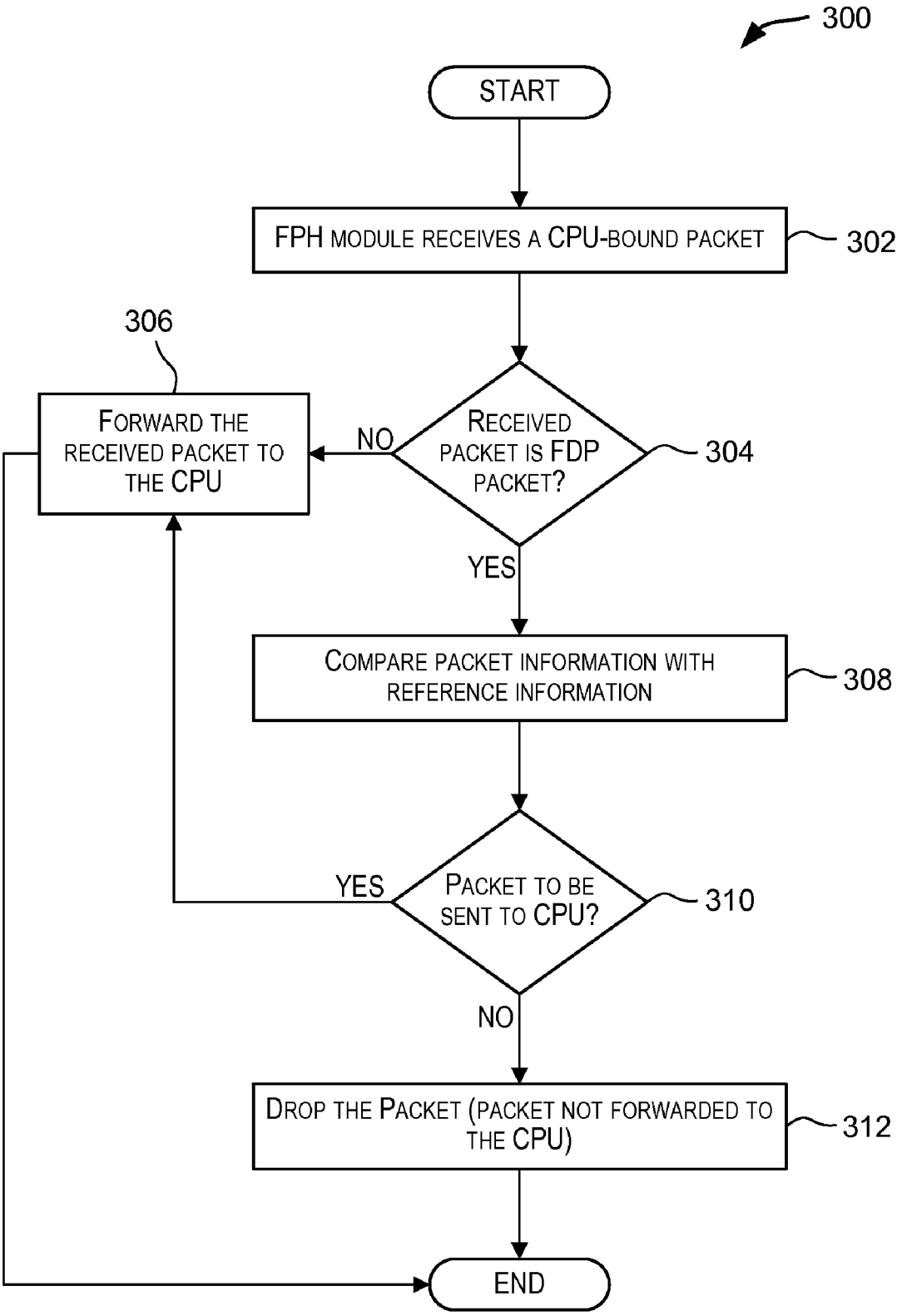

300
START
FPH MODULE RECEIVES A CPU-BOUND PACKET
302
306
FORWARD THE RECEIVED PACKET TO THE CPU
NO
RECEIVED PACKET IS FDP PACKET?
304
YES
COMPARE PACKET INFORMATION WITH REFERENCE INFORMATION
308
YES
PACKET TO BE SENT TO CPU?
310
NO
DROP THE PACKET (PACKET NOT FORWARDED TO THE CPU)
312
END

FIG. 3

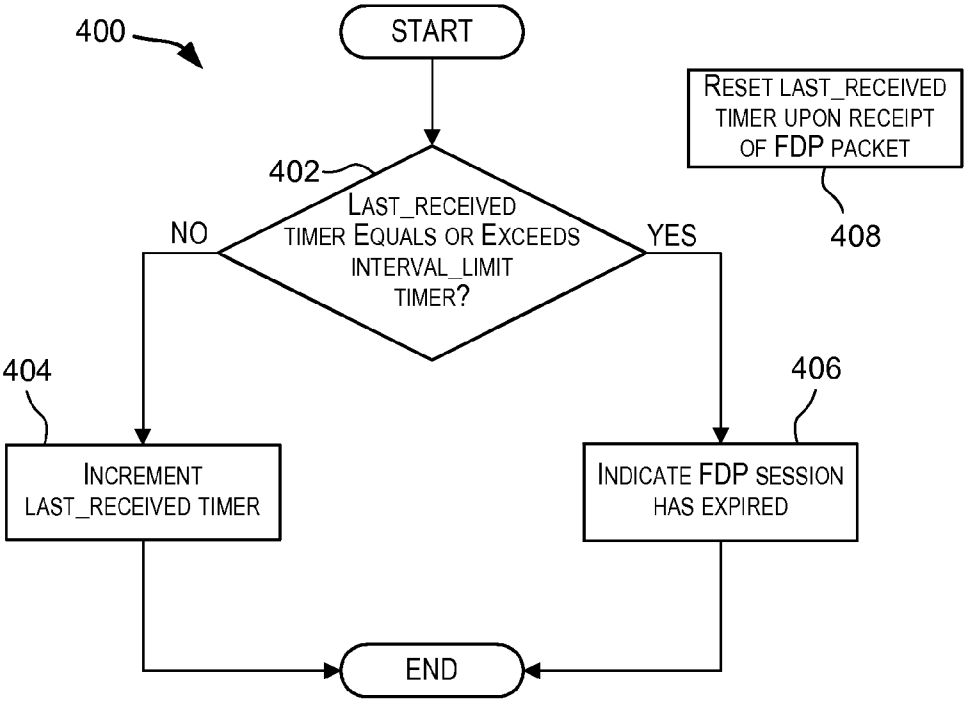

400
START
RESET LAST_RECEIVED TIMER UPON RECEIPT OF FDP PACKET
408
402
LAST_RECEIVED TIMER EQUALS OR EXCEEDS INTERVAL_LIMIT TIMER?
NO
YES
404
406
INCREMENT LAST_RECEIVED TIMER
INDICATE FDP SESSION HAS EXPIRED
END

FIG. 4

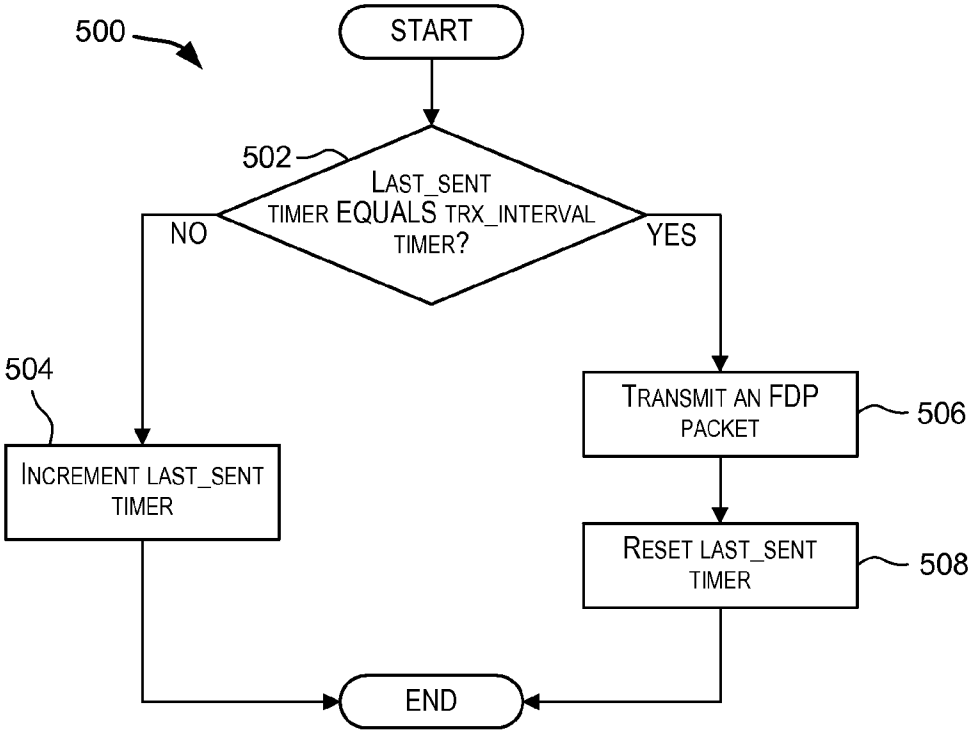

500
START
502
LAST_SENT TIMER EQUALS TRX_INTERVAL TIMER?
NO
YES
504
TRANSMIT AN FDP PACKET
506
INCREMENT LAST_SENT TIMER
RESET LAST_SENT TIMER
508
END

FIG. 5

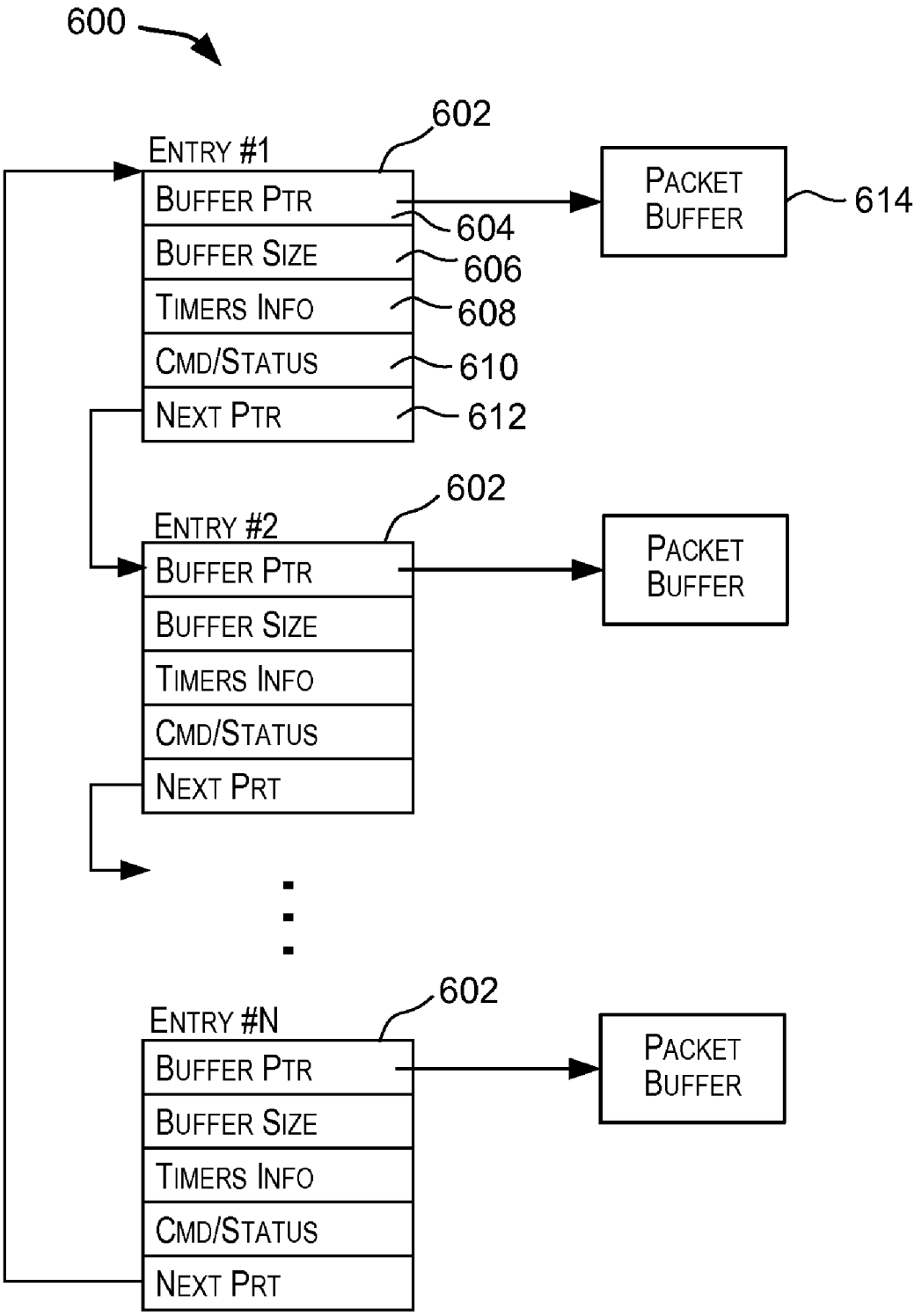

600
602
ENTRY #1
BUFFER PTR
BUFFER SIZE
TIMERS INFO
CMD/STATUS
NEXT PTR
604
606
608
610
612
PACKET BUFFER
614
602
ENTRY #2
BUFFER PTR
BUFFER SIZE
TIMERS INFO
CMD/STATUS
NEXT PRT
PACKET BUFFER
602
ENTRY #N
BUFFER PTR
BUFFER SIZE
TIMERS INFO
CMD/STATUS
NEXT PRT
PACKET BUFFER

FIG. 6

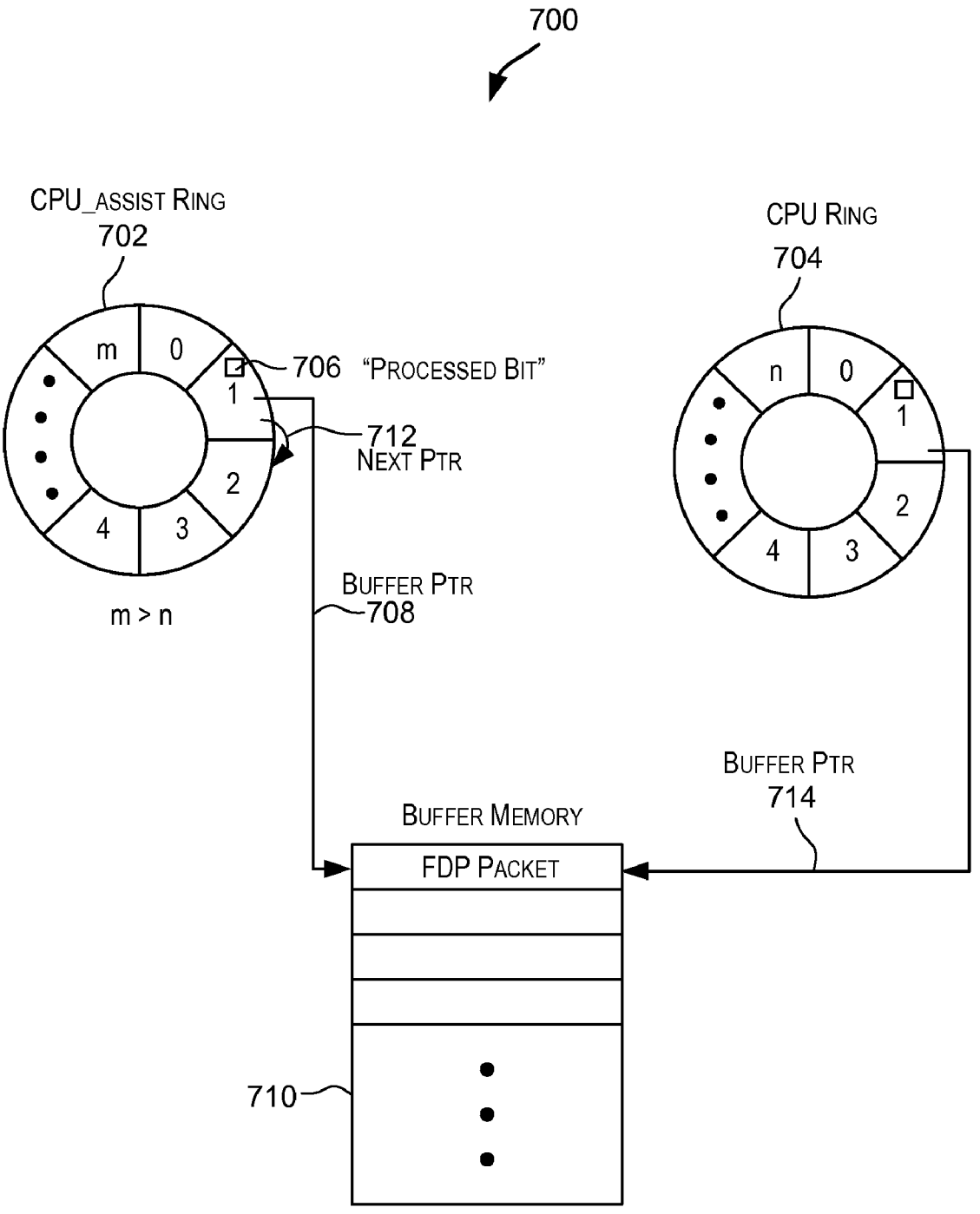

700
CPU_ASSIST RING
702
m
0
1
2
3
4
706 "PROCESSED BIT"
712
NEXT PTR
BUFFER PTR
708
m > n
CPU RING
704
n
0
1
2
3
4
BUFFER PTR
714
BUFFER MEMORY
FDP PACKET
710

FIG. 7

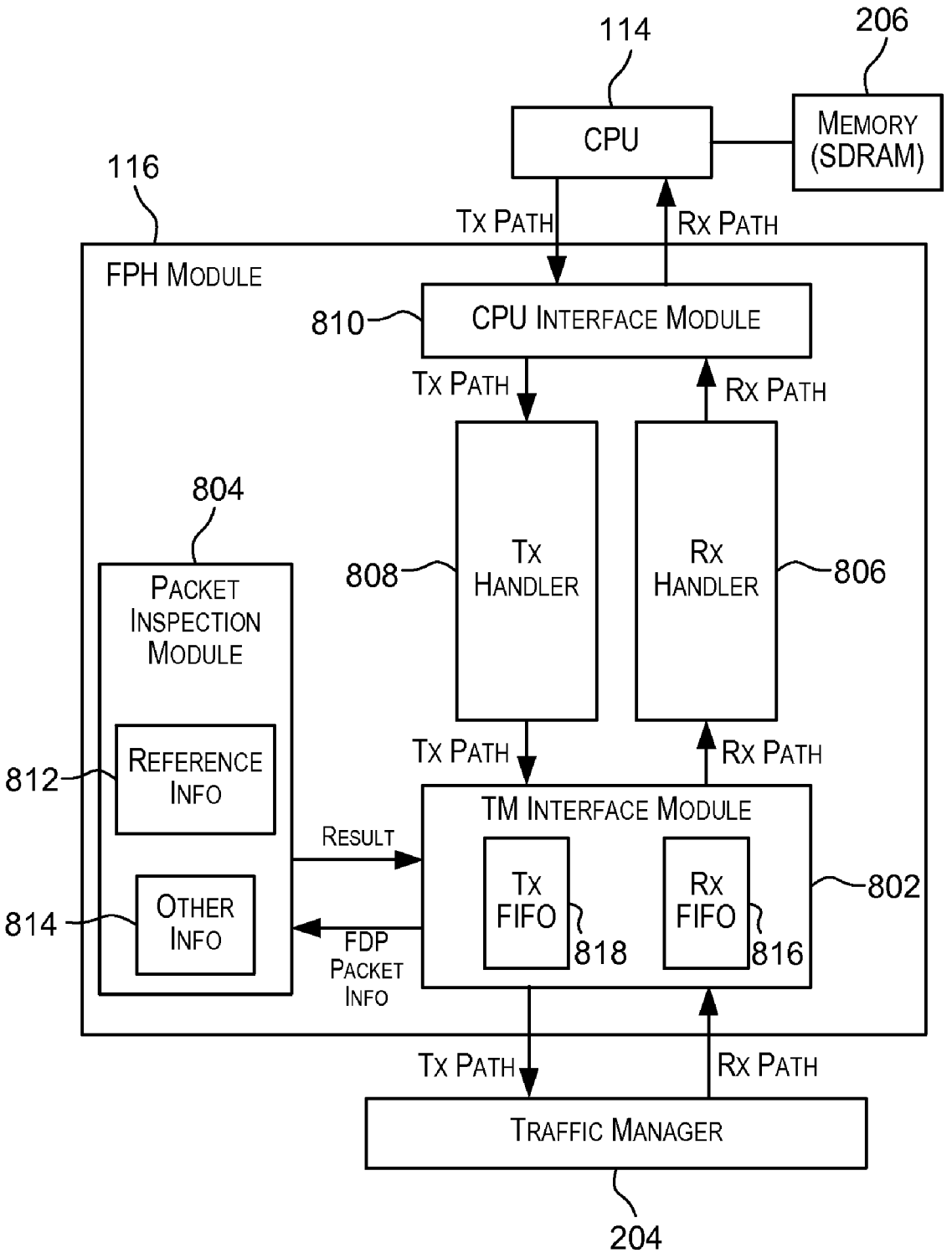

114
206
CPU
MEMORY (SDRAM)
116
TX PATH
RX PATH
FPH MODULE
810
CPU INTERFACE MODULE
TX PATH
RX PATH
804
808
TX HANDLER
RX HANDLER
806
PACKET INSPECTION MODULE
812
REFERENCE INFO
TX PATH
RX PATH
TM INTERFACE MODULE
RESULT
814
OTHER INFO
TX FIFO
818
RX FIFO
816
802
FDP PACKET INFO
TX PATH
RX PATH
TRAFFIC MANAGER
204

FIG. 8

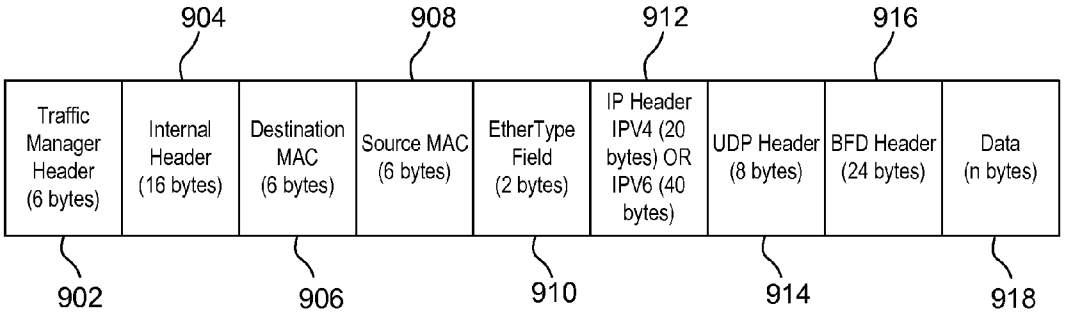

904
908
912
916
Traffic Manager Header (6 bytes)
Internal Header (16 bytes)
Destination MAC (6 bytes)
Source MAC (6 bytes)
EtherType Field (2 bytes)
IP Header IPV4 (20 bytes) OR IPV6 (40 bytes)
UDP Header (8 bytes)
BFD Header (24 bytes)
Data (n bytes)
902
906
910
914
918

FIG. 9

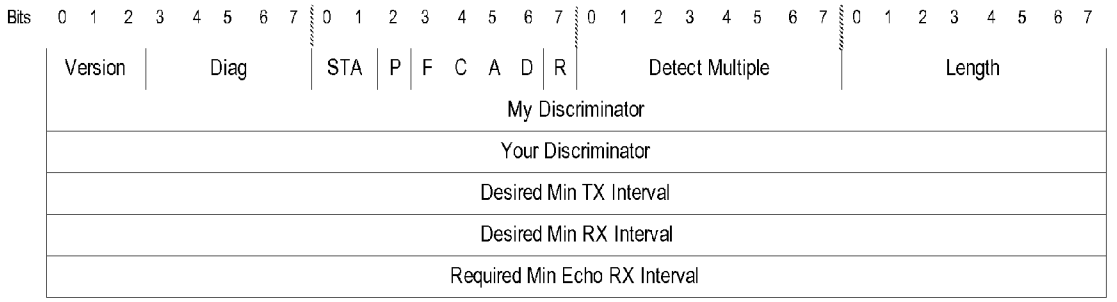

Bits 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
Version
Diag
STA
P
F
C
A
D
R
Detect Multiple
Length
My Discriminator
Your Discriminator
Desired Min TX Interval
Desired Min RX Interval
Required Min Echo RX Interval

FIG. 10

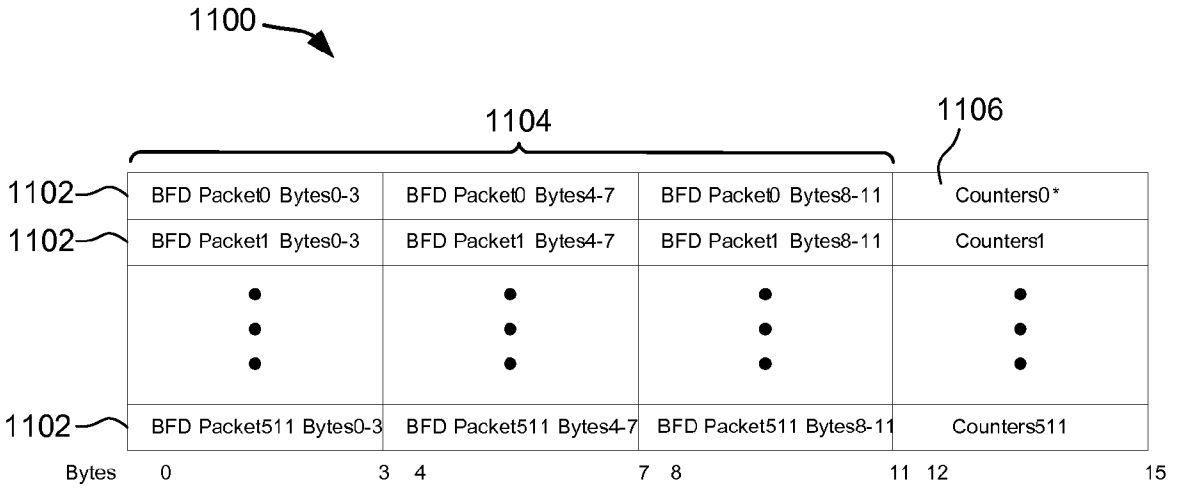

1100
1104
1106
1102
1102
1102
BFD Packet0 Bytes0-3
BFD Packet0 Bytes4-7
BFD Packet0 Bytes8-11
Counters0*
BFD Packet1 Bytes0-3
BFD Packet1 Bytes4-7
BFD Packet1 Bytes8-11
Counters1
BFD Packet511 Bytes0-3
BFD Packet511 Bytes4-7
BFD Packet511 Bytes8-11
Counters511
Bytes 0 3 4 7 8 11 12 15

FIG. 11

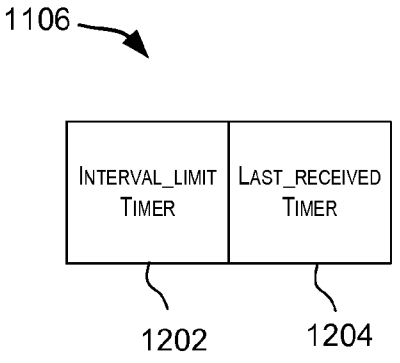

1106
INTERVAL_LIMIT TIMER
LAST_RECEIVED TIMER
1202
1204

FIG. 12

PACKETS FROM VPLS/VLL UPLINK

HEADER	INTERNAL HEADER 16 BYTES	DEST MAC 6 BYTES	SRC MAC 6 BYTES	ETYPE 2 BYTES 0x8847	MPLS STACK 4-8 BYTES	INNER DST MAC 6 BYTES	INNER SRC MAC 6 BYTES	ETYPE 2 BYTES 0x8902	802.1AG DATA

1302

FIG. 13A

PACKET FROM REGULAR LINK

HEADER	INTERNAL HEADER 16 BYTES	DEST MAC 6 BYTES	SRC MAC 6 BYTES	ETYPE 2 BYTES 0x8847	802.1 AG DATA

1304

FIG. 13B

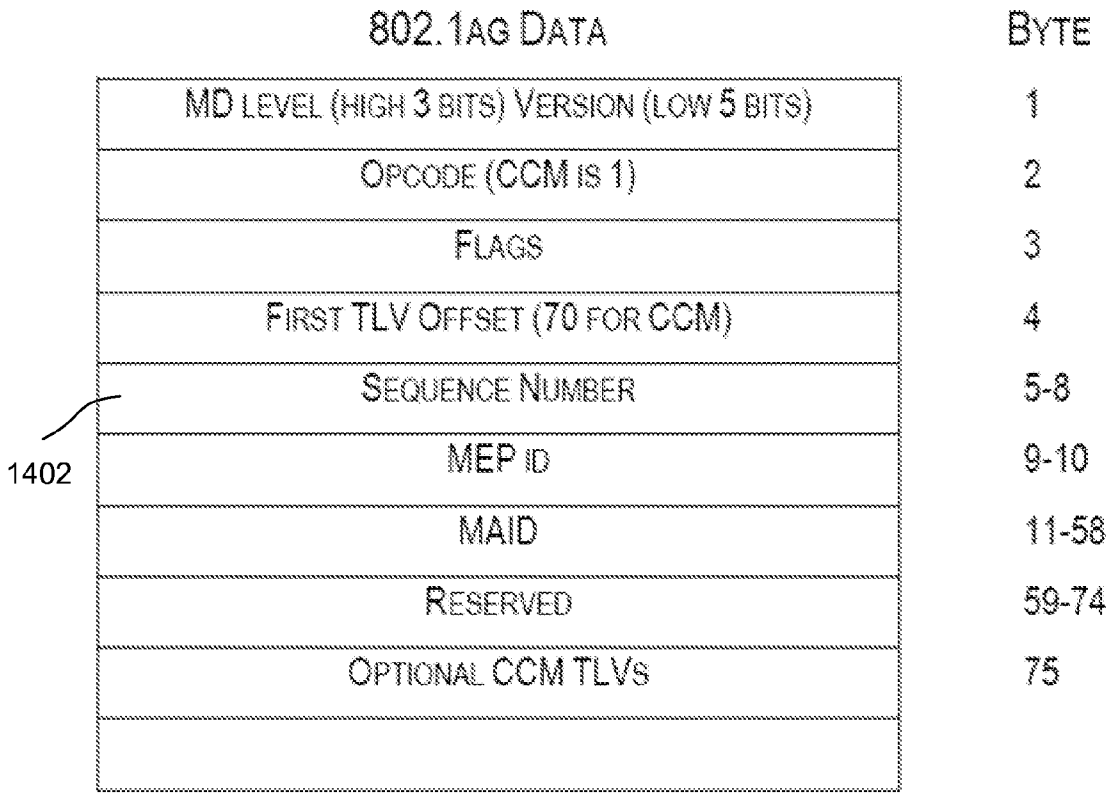

802.1AG DATA
BYTE
MD LEVEL (HIGH 3 BITS) VERSION (LOW 5 BITS)
1
OPCODE (CCM IS 1)
2
FLAGS
3
FIRST TLV OFFSET (70 FOR CCM)
4
SEQUENCE NUMBER
5-8
MEP ID
9-10
MAID
11-58
RESERVED
59-74
OPTIONAL CCM TLVS
75
1402

FIG. 14

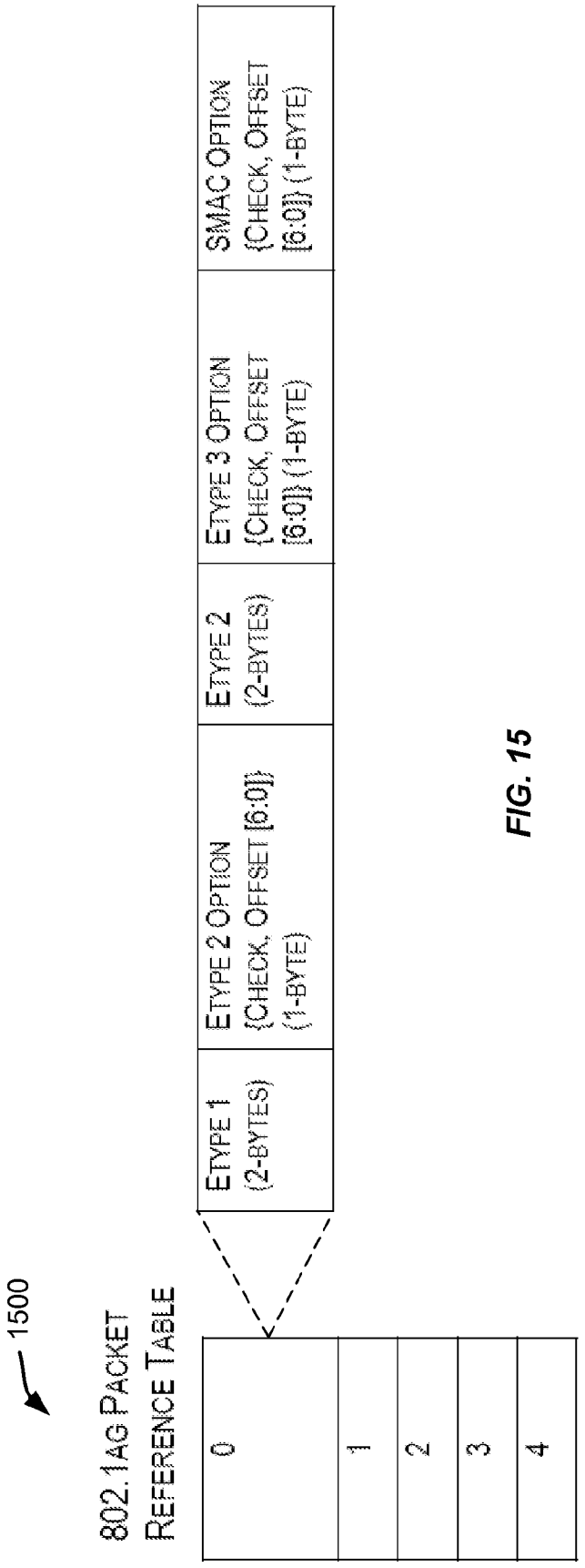

1500
802.1AG PACKET REFERENCE TABLE
0
1
2
3
4
ETYPE 1 (2-BYTES)
ETYPE 2 OPTION {CHECK, OFFSET [6:0]} (1-BYTE)
ETYPE 2 (2-BYTES)
ETYPE 3 OPTION {CHECK, OFFSET [6:0]} (1-BYTE)
SMAC OPTION {CHECK, OFFSET [6:0]} (1-BYTE)

FIG. 15

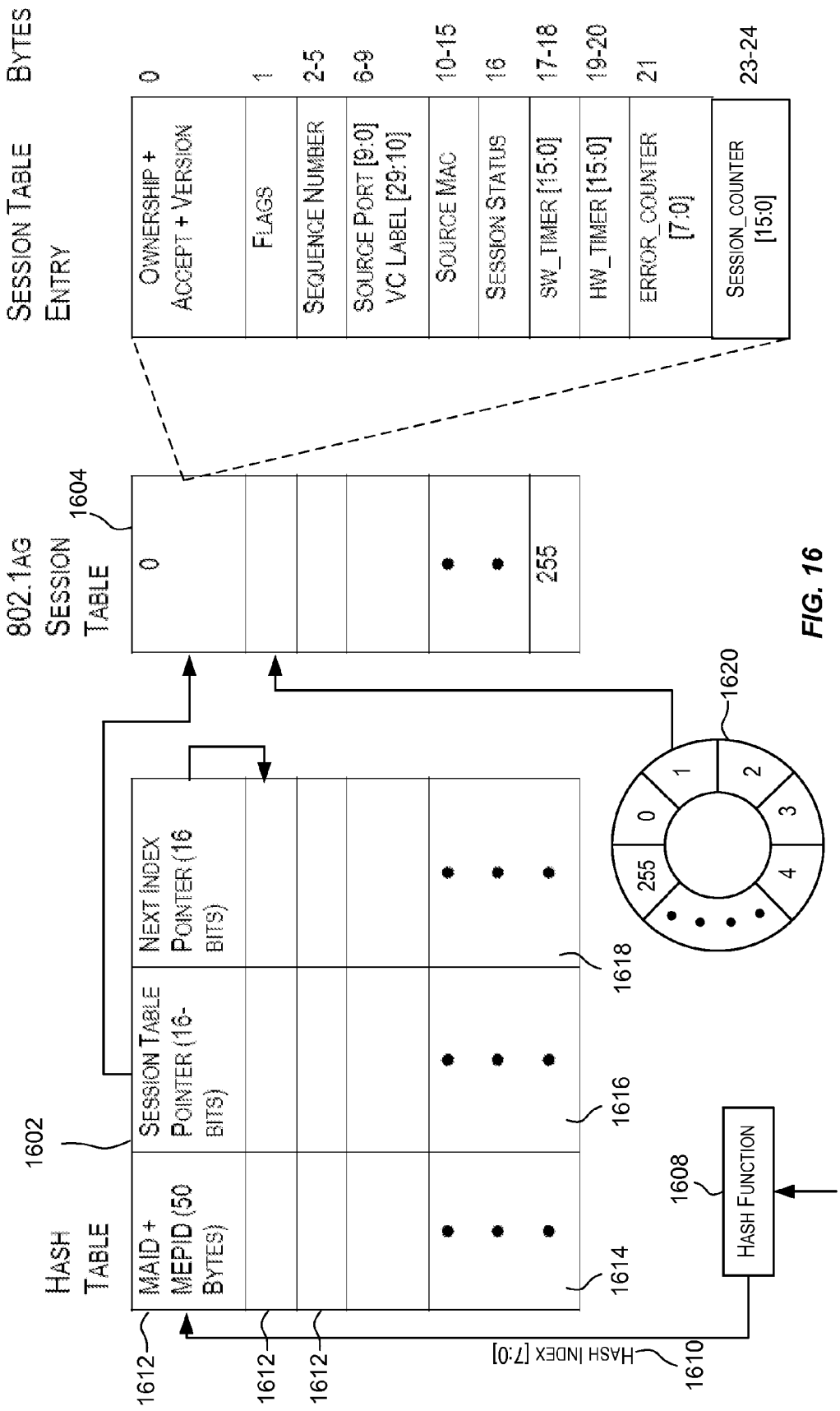

SESSION TABLE ENTRY
BYTES
OWNERSHIP + ACCEPT + VERSION
0
FLAGS
1
SEQUENCE NUMBER
2-5
SOURCE PORT [9:0] VC LABEL [29:10]
6-9
SOURCE MAC
10-15
SESSION STATUS
16
SW_TIMER [15:0]
17-18
HW_TIMER [15:0]
19-20
ERROR_COUNTER [7:0]
21
SESSION_COUNTER [15:0]
23-24
802.1AG SESSION TABLE
1604
0
255
1620
0
1
2
3
4
255
HASH TABLE
1602
MAID + MEPID (50 BYTES)
SESSION TABLE POINTER (16-BITS)
NEXT INDEX POINTER (16 BITS)
1618
1616
1614
1612
1612
1612
HASH INDEX [7:0]
1610
1608
HASH FUNCTION
802.1AG PACKET PORTION
1606

FIG. 16

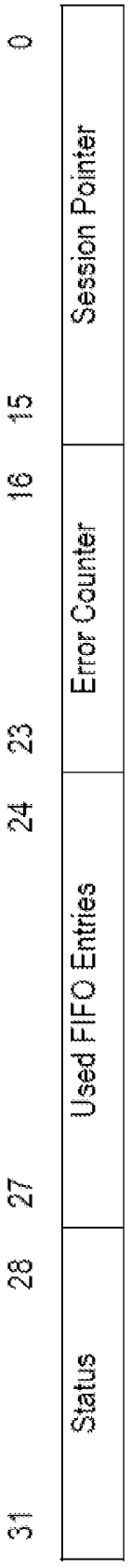

31
28
27
24
23
16
15
0
Status
Used FIFO Entries
Error Counter
Session Pointer

FIG. 17

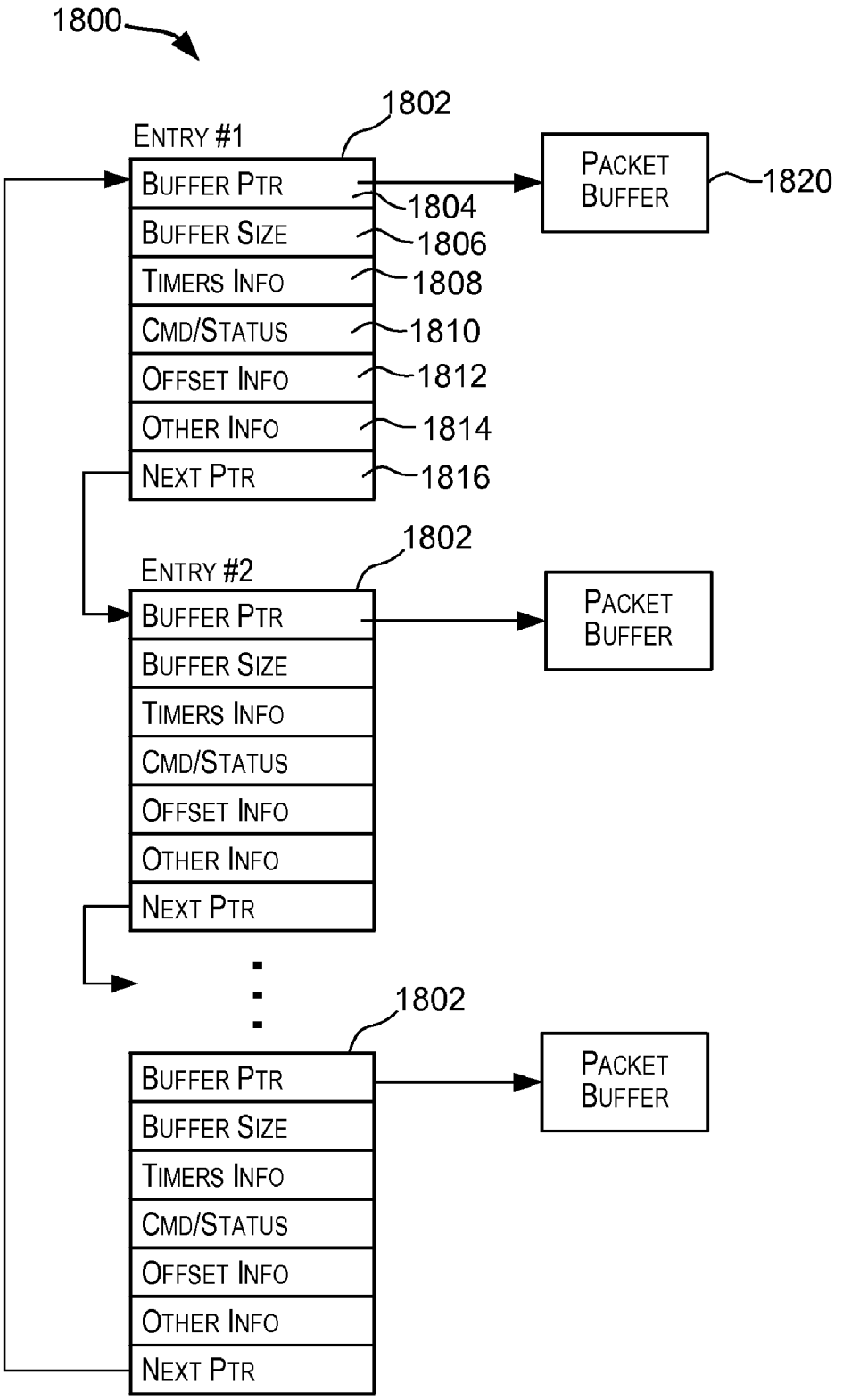

1800
1802
ENTRY #1
BUFFER PTR
BUFFER SIZE
TIMERS INFO
CMD/STATUS
OFFSET INFO
OTHER INFO
NEXT PTR
1804
1806
1808
1810
1812
1814
1816
PACKET BUFFER
1820
1802
ENTRY #2
BUFFER PTR
BUFFER SIZE
TIMERS INFO
CMD/STATUS
OFFSET INFO
OTHER INFO
NEXT PTR
PACKET BUFFER
1802
BUFFER PTR
BUFFER SIZE
TIMERS INFO
CMD/STATUS
OFFSET INFO
OTHER INFO
NEXT PTR
PACKET BUFFER

FIG. 18

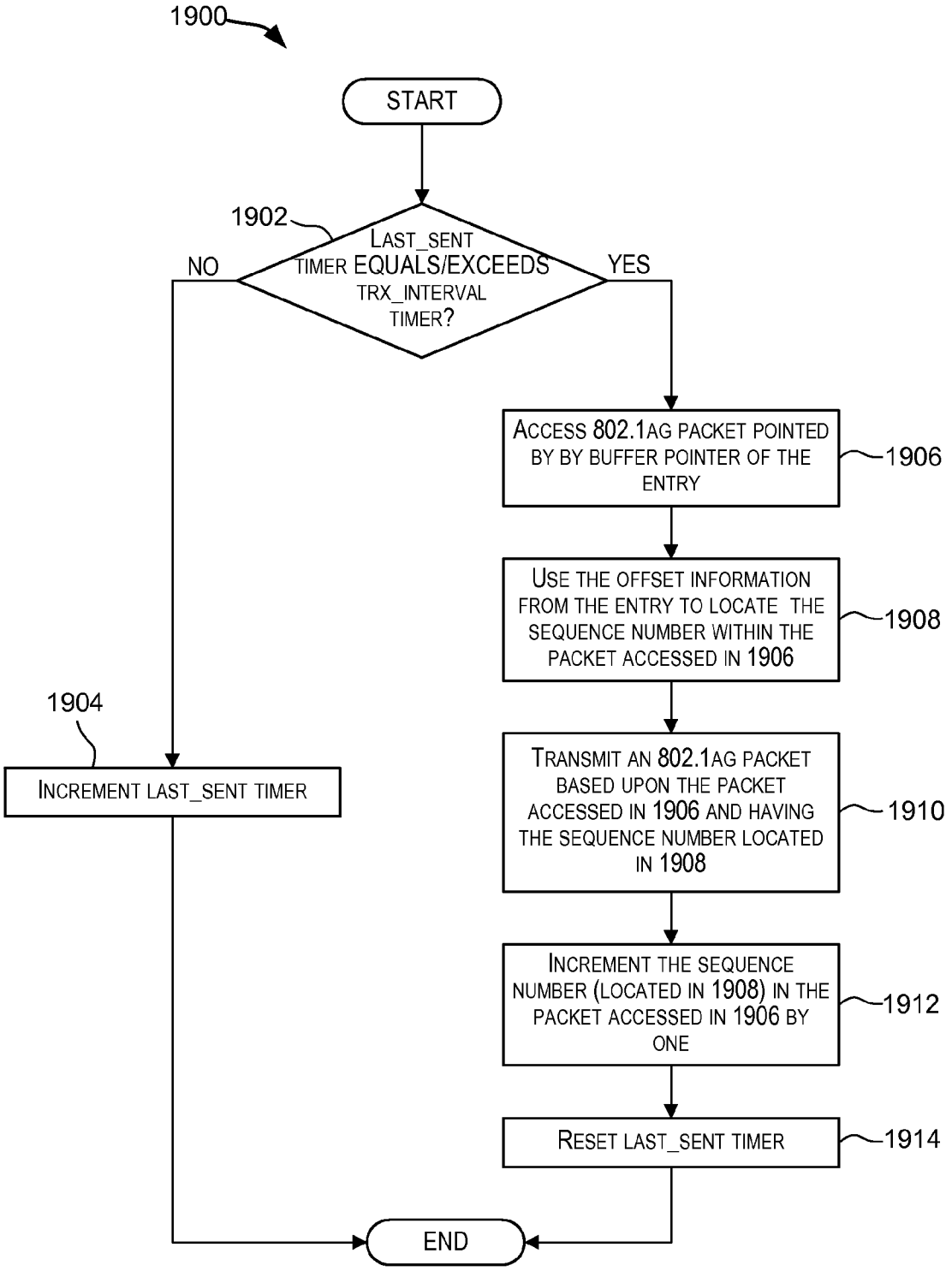

1900
START
1902
LAST_SENT TIMER EQUALS/EXCEEDS TRX_INTERVAL TIMER?
NO
YES
1904
INCREMENT LAST_SENT TIMER
ACCESS 802.1AG PACKET POINTED BY BY BUFFER POINTER OF THE ENTRY
1906
USE THE OFFSET INFORMATION FROM THE ENTRY TO LOCATE THE SEQUENCE NUMBER WITHIN THE PACKET ACCESSED IN 1906
1908
TRANSMIT AN 802.1AG PACKET BASED UPON THE PACKET ACCESSED IN 1906 AND HAVING THE SEQUENCE NUMBER LOCATED IN 1908
1910
INCREMENT THE SEQUENCE NUMBER (LOCATED IN 1908) IN THE PACKET ACCESSED IN 1906 BY ONE
1912
RESET LAST_SENT TIMER
1914
END

FIG. 19

TECHNIQUES FOR TRANSMITTING FAILURE DETECTION PROTOCOL PACKETS

CROSS-REFERENCES TO RELATED APPLICATIONS

[0001] The present application claims the benefit and priority under 35 U.S.C. 119(e) from U.S. Provisional Application No. 60/880,074 filed Jan. 11, 2007 entitled TIMING SENSITIVE PROTOCOL PACKET HARDWARE ASSIST, the entire contents of which are herein incorporated by reference for all purposes.

[0002] The present application also incorporates by reference for all purposes the entire contents of the following applications filed concurrently with the present application:

[0003] (1) U.S. Non-Provisional Application No. _____ (Atty. Docket No. 019959-003910US) titled TECHNIQUES FOR PROCESSING INCOMING FAILURE DETECTION PROTOCOL PACKETS;

[0004] (2) U.S. Non-Provisional Application No. _____ (Atty. Docket No. 019959-003920US) titled TECHNIQUES FOR DETECTING NON-RECEIPT OF FAULT DETECTION PROTOCOL PACKETS; and

[0005] (3) U.S. Non-Provisional Application No. _____ (Atty. Docket No. 019959-003940US) titled TECHNIQUES FOR USING DUAL MEMORY STRUCTURES FOR PROCESSING FAILURE DETECTION PROTOCOL PACKETS.

BACKGROUND OF THE INVENTION

[0006] The present application relates to networking technologies and more particularly to techniques for transmitting failure detection protocol packets from a network device.

[0007] The ability to detect communication failures is an important aspect of any networking environment. Networks use several different mechanisms to detect failures. For example, several different failure detection protocols (FDP) are used that enable detection of failures in a networking environment. Examples of FDPs include "hello" protocols, "keep alive" protocols, various Organization Administration and Maintenance (OAM) protocols, and others.

[0008] Network devices (e.g., routers, switches) in a network using a failure detection protocol are generally configured to continuously transmit FDP packets at regular intervals. A network device in the network receives FDP packets transmitted by other network devices in the network and uses the periodically received packets to ascertain the health of the other devices and the network connections. For example, if a network device does not receive an FDP packet within a period of time associated with the FDP packet, then the network device may assume that there is a network failure somewhere in the network that prevented the expected FDP packet from reaching the network device. The network device itself also transmits FDP packets on a periodic basis.

[0009] A network device may receive and transmit different types of FDP packets and may be involved in one or more FDP sessions at a time. Each transmitted FDP packet comprises an identifier identifying a unique FDP session for which the packet has been transmitted.

[0010] Traditionally, FDP-related processing in a network device is performed by software executed by a CPU or processor of the network device. For example, a processor of a network device executing software configured for FDP packets processing is configured to process FDP packets received by the network device from other network devices and handle transmission of FDP at regular intervals from the network device. In older failure detection protocols, the periodic time intervals associated with FDP protocols were generally in the range of seconds such as 1 second, 5 seconds, 10 seconds, 20 seconds, and the like. Such a time interval allowed sufficient time for the software running on the CPU to handle processing of the incoming FDP packets and also to process transmission of the FDP packets within the periodic time interval without detrimentally affecting the performance of the CPU or overwhelming the CPU. However, due to the large periodic interval time values, the time required to detect network failures is also quite large (usually several seconds). While this was acceptable in the past, it is no longer acceptable in today's larger and faster networks wherein a long failure detection time translates to large amounts of data being lost at today's fast networking speeds (e.g., at gigabit speeds).

[0011] In order to reduce failure detection times, today's networks typically use newer fault detection protocols with significantly shorter periodic time intervals that dramatically reduce the failure detection times. Examples of such newer FDPs include OAM protocols such as Bidirectional Forwarding (BFD) protocol which is used to detect router link failures and the 802.1ag standard that specifies protocols, procedures, and managed objects to support transport fault management. The periodic time intervals associated with these new FDPs is usually in the order of milliseconds (msecs) or even faster.

[0012] While these new protocols reduce failure detection times, they create an undue burden on a network device that is configured to handle processing of the FDP packets. As a result of the dramatically shorter periodic time intervals, a network device has to periodically transmit FDP packets in the order of milliseconds (msecs) or even faster, which is much faster than transmission processing done previously by network device for older FDPs. Due to the faster transmission rates, the number and rate at which FDP packets are received by a network device is also much faster than in the past. As a result, more CPU cycles per unit time are needed on the network device to perform FDP packets processing, including transmission of FDP packets and processing of incoming FDP packets. However, processors in conventional network devices executing software for processing the FDP packets are unable to cope up with the processing of newer FDP packets. As a result, conventional network devices are unable to handle and support the newer failure detection protocols.

BRIEF SUMMARY OF THE INVENTION

[0013] Embodiments of the present invention provide techniques that assist in processing of failure detection protocol (FDP) packets. Techniques are provided that assist a CPU of a network device in processing incoming FDP packets. The task of transmitting FDP packets from a network device is offloaded from the CPU of the network device and instead handled by another module of the network device. In this manner, the processing that the CPU of the network device has to perform for transmitting FDP packets for the various FDP sessions of the network device is reduced. This enables the network device to support newer FDPs with shorter periodic interval requirements.

[0014] According to an embodiment of the present invention, techniques are provided for transmitting packets according to an FDP from a network device. The network device stores timer information specifying a periodic time interval

for transmitting an FDP packet for an FDP. FDP packets for the FDP are transmitted periodically at times determined based upon the periodic time interval stored for the FDP. The transmitting is performed by a module of the network device other than a processor configured to execute software for processing FDP packets. The module may be a field-programmable logic device in one embodiment.

[0015] In one embodiment, the transmitting comprises determining a time for transmitting an FDP packet for the FDP based upon the periodic time interval stored for the FDP, and transmitting an FDP packet at the determined time.

[0016] In one embodiment, the timer information for an FDP may comprise a first timer specifying the periodic time interval for the FDP and a second timer identifying when an FDP packet was last transmitted by the network device for the FDP. An FDP packet may be transmitted when the second timer equals the first timer. The first timer may be set by the software executed by the processor of the network device and the second timer is updated by the module.

[0017] In one embodiment, FDP packets for an FDP may comprise a field that is updated with every successive FDP packet transmission. In this scenario, transmitting an FDP packet may comprise transmitting a first FDP packet based upon the periodic time interval stored for the FDP, wherein the field of the first failure is set to a first value, and transmitting, after transmitting the first FDP packet, a second FDP packet based upon the periodic time interval stored for the FDP, wherein the field of the second FDP packet is set to a second value that is different from the first value. In one embodiment this may done by storing a parameter for the FDP. Prior to transmitting the first FDP packet, the field of the first FDP packet may be set to the first value based upon the value of the parameter. The value of the parameter may be updated upon transmission of the first FDP packet. Prior to transmitting the second FDP packet, the field of the second FDP packet may be set to the second value based upon the updated value of the parameter.

[0018] Examples of FDPs include 802.1ag, Bidirectional Forwarding (BFD) protocol, and others. The periodic time intervals associated with the protocols may even be less than one second.

[0019] The foregoing, together with other features, embodiments, and advantages of the present invention, will become more apparent when referring to the following specification, claims, and accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[0020] FIG. **1** is a simplified diagram of a portion of a network that may incorporate an embodiment of the present invention;

[0021] FIG. **2** depicts a simplified block diagram of a network device incorporating an embodiment of the present invention;

[0022] FIG. **3** depicts a simplified flowchart showing a method performed by an FDP Packet Handler (FPH) module for processing a CPU-bound packet in the receive (Rx) path according to an embodiment of the present invention;

[0023] FIG. **4** depicts a simplified flowchart showing a method performed by an FPH module for detecting non-receipt of FDP packets for a session according to an embodiment of the present invention;

[0024] FIG. **5** depicts a simplified flowchart showing a method performed by an FPH module for transmitting FDP packets for an FDP session from a network device according to an embodiment of the present invention;

[0025] FIG. **6** depicts an example of a linked list priority queue that may be used by an FPH module to facilitate transmission of FDP packets according to an embodiment of the present invention;

[0026] FIG. **7** depicts a two ring structure that may be used by an FPH module to facilitate processing of FDP packets in the receive (Rx) path according to an embodiment of the present invention;

[0027] FIG. **8** is simplified block diagram of an FPH module according to an embodiment of the present invention;

[0028] FIG. **9** depicts a format for a BFD packet;

[0029] FIG. **10** depicts the format for a BFD header field;

[0030] FIG. **11** depicts a memory structure storing BFD reference information for multiple BFD sessions according to an embodiment of the present invention;

[0031] FIG. **12** depicts contents of counter information for a BFD session entry according to an embodiment of the present invention;

[0032] FIGS. **13**A and **13**B depict formats for two types of 802.1ag packets that may be processed by an embodiment of the present invention;

[0033] FIG. **14** depicts contents of an 802.1ag packet data section;

[0034] FIG. **15** depicts an 802.1ag packet reference table according to an embodiment of the present invention;

[0035] FIG. **16** depicts contents of a hash table and sessions table storing reference information according to an embodiment of the present invention;

[0036] FIG. **17** depicts a format of a Session Status FIFO according to an embodiment of the present invention;

[0037] FIG. **18** depicts a linked list that may be used by an FPH module to facilitate transmission of 802.1ag packets according to an embodiment of the present invention; and

[0038] FIG. **19** depicts a simplified flowchart showing a method performed by an FPH module for transmitting 802.1ag packets according to an embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

[0039] In the following description, for the purposes of explanation, specific details are set forth in order to provide a thorough understanding of the invention. However, it will be apparent that the invention may be practiced without these specific details.

[0040] Embodiments of the present invention provide techniques that assist in processing of failure detection protocol (FDP) packets. Techniques are provided that assist a CPU of a network device in processing incoming FDP packets. A failure detection protocol (or FDP) is intended to include any protocol, standard, procedure, or method in which packets are transmitted at periodic intervals for purposes of monitoring, detecting, or identifying a failure in a network. A packet transmitted according to an FDP is referred to as an FDP packet. Examples of FDPs that may be supported by embodiments of the present invention include OAM protocols such as Bidirectional Forwarding (BFD) protocol which is used to detect router link failures and the 802.1ag standard that specifies protocols, procedures, and managed objects to support transport fault management.

[0041] FIG. **1** is a simplified diagram of a portion of a network **100** that may incorporate an embodiment of the present invention. The portion of network **100** depicted in

FIG. **1** comprises a number of network devices **102**, **104**, **106**, **108**, **110**, and **112** coupled to one another via communication links. A network device may be any device capable of receiving and/or transmitting data in a network. The communication links may be wired links or wireless links. Different protocols may be used to communicate data between the various network devices.

[0042] The network devices depicted in FIG. **1** may use one or more types of failure detection protocols (FDPs) to facilitate detection of failures in network **100**. The FDPs used may include OAM protocols such as BFD and 802.1ag, and others. A network device may be involved in one or more FDP sessions. For each FDP session, the network device may be configured to continuously transmit FDP packets at a periodic time interval associated with that FDP session. A network device may also receive FDP packets transmitted by other devices in the network. The periodic time intervals at which FDP packets are received or transmitted may vary from one FDP session to another depending on the type of FDP session. The periodic time intervals may be in the order of one or more milliseconds (msecs), one or more seconds, or other faster or slower time intervals.

[0043] In accordance with an embodiment of the present invention, a network device **102** may be configured to receive FDP packets for one or more FDP sessions from other network devices. Network device **102** may also transmit FDP packets at periodic intervals for one or more FDP sessions to other devices in the network. The time intervals at which the packets are received and transmitted may vary from session to session and may be in the order of one or more milliseconds (as required by the newer FDPs) to seconds, or even faster or slower intervals. Network device **102** may be configured to ascertain the health of the other devices and the network connections based upon the FDP packets received by network device **102**. For example, if network device **102** does not receive an FDP packet for an FDP session within a preconfigured time interval for that session, then network device **102** may assume that there is a network failure somewhere in the network that prevented the expected FDP packets from reaching network device **102** for the session. Network device itself may transmit FDP packets at periodic intervals for one or more FDP sessions.

[0044] As depicted in FIG. **1**, network device **102** comprises a processor or CPU **114** and an FDP Packet Handler (FPH) module **116**. CPU **114** is configured to execute software for performing various tasks performed by network device **102**. In one embodiment, CPU **114** executes software (e.g., program, code, instructions, etc.) that is configured to handle processing of FDP packets. According to an embodiment of the present invention, FPH module **116** assists CPU **114** in FDP packets-related processing, including processing of incoming FDP packets and transmission of FDP packets. FPH module **116** may be implemented as a field programmable logic device (FPLD) such as a programmed field-programmable gate array (FPGA) device or an ASIC.

[0045] In one embodiment, FPH module **116** is configured to filter FDP packets received by network device **102** and bound for CPU **114** such that only a subset of received FDP packets are forwarded to CPU **114** for processing, the other FDP packets are dropped by FPH module **116** and not forwarded to CPU **114**. In one embodiment, FPH module **116** receives CPU-bound packets and identifies FDP packets from other CPU-bound packets. For a packet identified as an FDP packet, FPH module **116** determines whether the packet needs to be sent to CPU **114**. If FPH module **116** determines that the FDP packet need not be forwarded to CPU **114**, the FDP packet is dropped and not sent to CPU **114**, thereby relieving CPU **114** from having to process the packet. An FDP packet is forwarded to CPU **114** only if FPH module **116** determines that the packet cannot be consumed by FPH module **116** and needs to be forwarded to CPU **114** for inspection. In this manner, only a small subset of FDP packets received by network device **102** is forwarded to CPU **114** for processing. This reduces the amount of processing that CPU **114** has to do to process FDP packets received by network device **102**.

[0046] FPH module **116** is also configured to assist in transmission of FDP packets from network device **102**. For each FDP session in which network device **102** participates, FDP packets for that session are transmitted from network device **102** at periodic time intervals associated with the FDP for that session. The time intervals may be different for different FDP sessions. In this manner, the task of transmitting FDP packets is partially or completely offloaded from CPU **114** of network device **102**.

[0047] By assisting in processing of incoming FDP packets, FPH module **116** reduces the number of incoming FDP packets that CPU **114** has to process, thereby freeing CPU cycles for other tasks performed by CPU **114**. FPH module **116** also offloads the task of transmitting FDP packets from CPU **114**. In this manner, the amount of FDP packets-related processing that CPU **114** has to perform is reduced. This enables network device **102** to support various FDPs including FDPs with shorter periodic interval requirements (e.g., periodic time intervals measured in milliseconds or even faster). Network device **102** comprising a FPH module **116** may coexist in a network with other network devices that may or may not comprise FPH modules.

[0048] FIG. **2** depicts a simplified block diagram of a network device **102** incorporating an embodiment of the present invention. FIG. **2** is merely illustrative of an embodiment incorporating the present invention and does not limit the scope of the invention as recited in the claims. One of ordinary skill in the art would recognize other variations, modifications, and alternatives. Network device **102** may be embodied as a switch or router, such as routers and switches provided by Foundry Networks®, Inc. of Santa Clara, Calif.

[0049] As depicted in FIG. **2**, network device **102** comprises one or more ports **202**, a traffic manager (TM) module **204**, an FDP packet handler (FPH) module **116**, and a CPU **114** with associated memory **206** (e.g., SDRAM). Network device **102** receives data, including FDP packets, via one or more ports **202**. Network device **102** may receive multiple streams of FDP packets concurrently from one or more sources for one or more FDP sessions. An FDP packet received from a source may comprise an FDP session identifier identifying the FDP session for which the packet is received. Ports **202** are also used to transmit data, including FDP packets, from network device **102**.

[0050] Network device **102** may support two data processing paths including a receive path (Rx path) and a transmit path (Tx path). Processing for the two paths may be performed in parallel. The Rx path is a path traversed by a packet from a port of network device **102** towards CPU **114** of network device **102**. The Tx path is a path traversed by a packet from CPU **114** towards a port of network device **102**.

[0051] In one embodiment, in the Rx path, packets received by network device **102** via one or more ports **202** (or a subset of the received packets) are forwarded to TM module **204**.

These packets may include CPU-bound packets (i.e., packets to be forwarded to CPU **114** for processing) and other packets. TM module **204** is configured to forward the CPU-bound packets to FPH module **116**. The CPU-bound packets may include FDP packets, packets that enable CPU **114** to learn network topologies, and other types of packets. In the Tx path, TM module **204** is configured to receive packets, including FDP packets, from FPH module **116**. The packets are then forwarded to their destinations, which may include one or more ports **202** of network device **102**. The packets, including FDP packets, are then transmitted from network device **102** using the destination ports.

[0052] FPH module **116** is configured to assist with processing related to FDP packets. In one embodiment, FPH module **116** is implemented in hardware. For example, FPH module **116** may be implemented as a field programmable logic device (FPLD) such as a programmed field-programmable gate array (FPGA) device. FPH module **116** may also be implemented as an ASIC. As depicted in FIG. **2**, FPH module **116** is positioned in the Rx path between TM module **204** and CPU **114** and in the Tx path between CPU **114** and TM module **204**. This enables FPH module **116** to receive all CPU-bound packets in the Rx path, including all FDP packets received by network device **102**, prior to the packets being forwarded to CPU **114**.

[0053] Processing of Incoming FDP Packets

[0054] In the Rx path, FPH module **116** receives CPU-bound packets from TM Interface module **204**. From the packets received from TM module **204**, FPH module **116** is configured to identify FDP packets. FPH module **116** may use different techniques to identify FDP packets. For example, in one embodiment, contents of a packet, including the header and/or the payload of the packet, may be examined to determine if the packet is an FDP packet. For example, a BSD packet comprises a UDP header specifying a destination port that identifies the packet as a BSD packet and accordingly an FDP packet. For an 802.1ag packet, fields in the packet are used to identify if the packet is an 802.1ag packet.

[0055] For a packet identified as an FDP packet, FPH module **116** determines if the FDP packet needs to be forwarded to CPU **114** or whether the FDP packet can be dropped without forwarding to CPU **114**. Various different techniques may be used by FPH module **116** to determine if a packet needs to be forwarded to CPU **114**. The techniques may be different for different FDPs. According to one technique, a portion of the FDP packet is compared with preconfigured reference information and the results of the comparison are used to determine if the FDP packet needs to be sent to CPU **114** or if the packet can be dropped without sending it to CPU **114**. As part of the comparison, FPH module **116** is configured to determine if a portion of an FDP packet received for a session matches a corresponding entry in the reference information for that session. In one embodiment, a match indicates that the FDP packet need not be forwarded to the CPU. In such a case, the FDP packet for that session is dropped and not forwarded to CPU **114**. As a result, CPU **114** does not receive the FDP packet and consequently does not have to process the dropped packet. In one embodiment, if the portion of the FDP packet being compared does not match information in the reference information, then FPH module **116** forwards the FDP packet to CPU **114**. In such a scenario, FPH module **116** may also raise an interrupt signaling to CPU **114** that a packet is being forwarded to CPU **114**.

[0056] Different techniques may be used to forward an FDP packet from FPH module **116** to CPU **114**. In one embodiment, a direct memory access (DMA) technique may be used to forward the packet to CPU **114**. FPH module **116** may DMA the packet to a buffer stored in memory **106** associated with CPU **114**. CPU **114** may then read the packet from memory **206**. Various other techniques may also be used to forward FDP packets to CPU **114**.

[0057] The reference information that is used by FPH module **116** to determine whether or not an FDP packet for a session needs to be forwarded to CPU **114** may be preconfigured and programmed by software executed by CPU **114**. The reference information may comprise various rules configured by the software for various types of FDP sessions. The reference information may be stored in different formats. In one embodiment, the reference information is stored as a table with each row entry in the table corresponding to an FDP session. In one embodiment, the reference information is stored by FPH module **116**. The reference information may also be stored in other memory locations accessible to FPH module **116**. For example, in one embodiment, the reference information may be stored in one or more memories **206** associated with CPU **114**.

[0058] FIG. **3** depicts a simplified flowchart **300** showing a method performed by FPH module **116** for processing a CPU-bound packet in the receive (Rx) path according to an embodiment of the present invention. As depicted in FIG. **3**, processing is initiated when FPH module **116** receives a CPU-bound packet (step **302**). For the embodiment depicted in FIG. **2**, processing may be initiated when a CPU-bound packet is received by FPH module **116** from TM module **204**.

[0059] FPH module **116** then determines if the packet is an FDP packet (step **304**). As previously described, various different techniques may be used to determine if a packet is an FDP packet. For example, information stored in the header and/or payload of the packet may be used to determine if the packet is an FDP packet. As part of **304**, FPH module **116** may also determine a particular FDP (e.g., BFD or 802.1ag, or other) to which the packet belongs. If it is determined in **304** that the packet is not an FDP packet, then the packet is forwarded to CPU **114** (step **306**) and processing ends. Various different techniques such as DMA techniques and others may be used to forward the packet to CPU **114**.

[0060] If it is determined in **304** that the packet is an FDP packet, then a portion of the packet is compared with information stored in reference information (step **308**). The reference information may be stored by FPH module **116** or may be stored in some memory location accessible to FPH module **116**. The portion of the packet being compared may include a portion of the header of the packet and/or a portion of the payload of the packet. In one embodiment, the packet information is compared with reference information configured for the FDP session for which the packet is received.

[0061] Different types of reference information may be stored and used for the comparison for different FDPs. Accordingly, in **308**, the reference information that is used for the comparison may depend on the FDP corresponding to the FDP packet. For example, the reference information used for a BFD packet is different from the reference information used for an 802.1ag packet.

[0062] Based upon the results of the comparison performed in **308**, a determination is made if the packet is to be forwarded to CPU **114** (step **310**). In one embodiment, if the portion of the FDP packet being compared matches informa-

tion in the reference information for that FDP then this indicates that the packet is not to be forwarded to CPU **114**. If however the portion of the packet being compared does not match information in the reference information then the packet is identified to be forwarded to CPU **114**. If it is determined in **310** that the packet is to be forwarded to CPU, then the FDP packet is forwarded to CPU **114** according to step **306** and processing terminates. If it is determined in **310** that the packet is not to be forwarded to CPU, then the FDP packet is dropped and not forwarded to CPU **114** (step **312**) and processing terminates.

[0063] The processing depicted in FIG. **3** and described above may be repeated for each packet received by FPH module **116**. As a result of the processing depicted in FIG. **3** and described above, only a subset of CPU-bound FDP packets received by FPH module **116** may need to be forwarded to CPU **114** for processing. In this manner, CPU **114** may not receive each FDP packet received by network device **102** thereby reducing the number of incoming FDP packets that need to be processed by CPU **114**.

[0064] Determining Non-Receipt of FDP Packets

[0065] FPH module **116** is also configured to monitor non-receipt of FDP packets. FPH module **116** is configured to monitor and take appropriate actions in situations where an FDP packet for an FDP session is not received by network device **102** within a periodic time interval for that FDP session. As previously described, an FDP specifies a periodic time interval in which packets for that FDP are to be transmitted and received. When an FDP packet is not received within the expected time interval, FPH module **116** is configured to signal such an occurrence to CPU **114** since it may indicate a network failure.

[0066] As previously described, a network device such as network device **102** depicted in FIG. **2** may be involved in one or more FDP sessions. Each FDP session may have its own associated periodic time interval within which FDP packets should be received for that session. FPH module **116** is configured to monitor and track the receipt of FDP packets for each of the sessions and raise a signal if an FDP packet is not received within the time interval for a session.

[0067] FPH module **116** may use different techniques to monitor receipt and non-receipt of FDP packets information for various FDP sessions. In one embodiment, FPH module **116** maintains a pair of timers for each session. The pair of timers includes a first timer (interval_limit timer) that indicates a time interval_limit within which an FDP packet should be received for that session. This timer is typically programmed by software executed by the CPU of the network device. The pair of timers also includes a second timer (last_received timer) that is used by FPH module **116** to monitor the time that it has waited to receive an FDP packet for the session since the last_receipt of an FDP packet for that session. When an FDP packet for that session is received, the last_received timer is reset by FPH module **116**. FPH module **116** iteratively checks the values of the interval_limit timer and the last_received timer for each of the FDP sessions in which the network device participates. During each iteration for a session, FPH module **116** checks if the last_received timer for a session has reached the interval_limit timer for that session. When the last_received timer reaches or exceeds the interval_limit timer, it indicates that the FDP packet for the session was not received in the expected time interval and FPH module **116** signals this to CPU **114**. The FDP session may be considered to have expired due to the non-receipt of an FDP packet. In one embodiment, an interrupt is generated by FPH module **116** to flag that the particular FDP session has expired possibly due to some failure in the network (e.g., a link is down). If the last_received timer has not reached he interval_limit timer, then the last_received timer is incremented. In this manner, the timers are used to monitor receipt and non-receipt of FDP packets for each FDP session. The timers information may be stored by FPH module **116** or in some memory location accessible to FPH module **116**.

[0068] FIG. **4** depicts a simplified flowchart **400** showing a method performed by FPH module **116** for detecting non-receipt of FDP packets for an FDP session according to an embodiment of the present invention. FIG. **4** depicts processing performed at each iteration by FPH module **116**. As depicted in FIG. **4**, at each iteration, FPH module **116** checks if the last_received timer equals or exceeds the interval_limit timer (step **402**). If it is determined in **402**, that the last_received timer has not reached the interval_limit timer (i.e., the last_received timer is less than the interval_limit timer), the last_received timer is incremented (step **404**). The amount by which the last_received timer is incremented depends upon the time frequency at which FPH module **116** performs the iterations. For example, if FPH module **116** checks the timers for a session every 1 msec, then the last_received timer is incremented by 1 msec, if the check is performed every 50 msecs then the last_received timer is incremented by 50 msecs, and so on. If it is determined in **402**, that the last_received timer has reached or exceeded the interval_limit timer (i.e., the last_received timer is equal to or greater than the interval_limit timer) then this indicates that an FDP packet for the session has not been received within the time interval window for that particular FDP session and a signal is raised to indicate that the FDP session has expired (step **406**). As also shown in FIG. **4**, the last_received timer for an FDP session is reset to zero upon receipt of an FDP packet for that session within the interval_limit timer period.

[0069] The processing depicted in FIG. **4** is repeated by FPH module **116** for each of the FDP sessions at periodic intervals such as every one millisecond, every 50 milliseconds, etc. In one embodiment, the frequency at which the timers information is checked for a session is programmable. As previously indicated, FPH module **116** may monitor timers for multiple FDP sessions. The frequency with which FPH module **116** checks the timers information for the sessions may also be automatically determined based upon the interval_limit timers for the various sessions. For example, in one embodiment, the iteration frequency may be set to the least common denominator of the various interval_limit timers being tracked for the various sessions by FPH module **116**. For example, if three FDP sessions are being tracked having interval_limit timers of 4 msecs, 6 msecs, and 12 msecs, then the frequency at which the iterations are performed may be set to 2 msecs, which is the least common denominator for the three interval_limit timers. In this manner, the frequency is programmable and/or may be automatically determined from the interval_limit timers.

[0070] As described above, in the Rx path, FPH module **116** handles processing of FDP packets received by a network device and also non-receipt of FDP packets. In the Tx path, FPH module **116** may receive packets from CPU **114** and forward the packets to TM module **204**. TM module **204** may then forward the packets to the appropriate destinations for the packets.

[0071] Transmission of FDP Packets

[0072] FPH module **116** is also configured to assist in transmission of FDP packets for the various FDP sessions in which network device **102** participates. In this manner, the FDP packets transmission task is offloaded from CPU **114**. In order to facilitate transmission of FDP packets, in one embodiment, FPH module **116** maintains a pair of timers for each FDP session. The pair of timers includes a first timer (trx_interval timer) that indicates the periodic transmission interval for transmitting an FDP packet for that FDP session. The trx_interval timer may be different for the different FDP sessions handled by the network device. This timer is typically programmed by software executed by CPU **114**. The pair of timers also includes a second timer (last_sent timer) that is used by FPH module **116** to monitor the time when an FDP packet was last transmitted by network device **102** for the FDP session. The two timers for each session are iteratively checked at periodic intervals to determine when to transmit an FDP packet for that session. In one embodiment, when the last_sent timer is equal to the trx_interval timer, FPH module **116** transmits an FDP packet for that session and the last_sent timer is reset to zero to restart the count. The FDP packet transmitted by FPH module **116** for a session is forwarded to TM module **204** and then to a destination port of network device **102**. The FDP packet is then forwarded from network device **102** using the destination port. In this manner, FPH module **116** facilitates transmission of FDP packets from network device **102** for the various FDP sessions at periodic intervals associated with the FDP sessions.

[0073] FIG. **5** depicts a simplified flowchart **500** showing a method performed by FPH module **116** for transmitting FDP packets for an FDP session from network device **102** according to an embodiment of the present invention. FIG. **5** depicts processing performed at each iteration by FPH module **116**. As depicted in FIG. **5**, at each iteration, FPH module **116** checks if the last_sent timer has reached the trx_interval timer (step **502**). If it is determined in **502**, that the last_sent timer has not reached the trx_interval timer (i.e., the last_sent timer is less than the trx_interval timer), the last_sent timer is incremented (step **504**). The amount by which the last_sent timer is incremented depends upon the time frequency at which FPH module **116** checks the timers for the session. For example, if FPH module **116** checks the timers for a session very 1 msec, then the last_received timer is incremented by 1 msec, if the iteration is performed every 50 msecs then the last_received timer is incremented by 50 msecs, and so on. If it is determined in **502** that the last_sent timer has reached the trx_interval timer (i.e., the last_sent timer is equal to the trx_interval timer), this indicates that it is time to transmit an FDP packet for the session and an FDP packet is transmitted (step **506**). In one embodiment, as part of **506**, FPH module **116** transmits an FDP packet for the session to TM module **204**. The FDP packet is then forwarded to a port of the network device and transmitted from the network device via the port. After an FDP packet transmission, the last_sent timer is reset to zero to restart the countdown for the next time an FDP packet is to be sent for the session (step **508**). The processing depicted in FIG. **5** is repeated at each iteration.

[0074] The processing depicted in FIG. **5** and described above may be performed by FPH module **116** at periodic intervals for each FDP session of network device **102**. The frequency at which the processing is performed may be every one millisecond, every 50 milliseconds, etc. In one embodiment, the frequency at which FPH module **116** performs the processing depicted in FIG. **5** is programmable. The frequency may also be automatically determined based upon the trx_interval timers for the various sessions. For example, the processing frequency may be set to the least common denominator of the various trx_interval timers for the various FDP sessions of network device **102**. For example, if three FDP sessions are being handled having trx_interval timers of 20 msecs, 40 msecs, and 100 msecs, then the frequency at which the iterations are performed may be set to 20 msecs. In this manner, the frequency is programmable and/or may be automatically determined from the trx_interval timers.

[0075] As depicted in FIG. **5** and described above, FPH module **116** handles transmission of FDP packets at regular intervals for the various FDP sessions of network device **102**. The sessions may correspond to different FDPs. Software executed by CPU **114** typically pre-configures the periodic intervals at which FDP packets are to be sent for the FDP sessions and FPH module **116** handles the transmission of FDP packets for the sessions. The periodic time interval for a session at which FDP packets are transmitted for that session may be measured in seconds (e.g., every 1 second, every 5 seconds, etc.) or even faster than one second such as measured in milliseconds (e.g., every 1 msec, every 5 msecs, etc.), or some other faster or slower time period. In this manner, the FDP packets transmission task is offloaded from CPU **113** by FPH module **116**.

[0076] Various different memory structures may be used to facilitate automated transmission of FDP packets. In one embodiment, CPU **114** may provide multiple priority queues for transmitting packets. Each priority queue is implemented as a linked list that contains a set of descriptor entries. In one embodiment, one or more such linked list priority queues are assigned to FPH module **116** to facilitate transmission of FDP packets. FIG. **6** depicts an example of a linked list priority queue **600** that may be used by FPH module **116** to facilitate transmission of FDP packets according to an embodiment of the present invention. As depicted in FIG. **6**, linked list **600** comprises a set of descriptor entries **602**. Each entry **602** corresponds to and stores information for an FDP session for which an FDP packet is to be transmitted. Each entry **602** comprises: (1) a buffer pointer **604** pointing to a memory location **614** storing the corresponding FDP packet; (2) buffer size information **606** identifying the size of the corresponding FDP packet; (3) timers information **608**; (4) command/status information **610**; and (5) a pointer **612** pointing to the next entry in the linked list. Pointer **612** is used to traverse the entries in linked list **600**. In one embodiment, linked list **600** may be implemented as a circular linked list wherein pointer **612** of the last entry in the linked list points to the first entry in the linked list. For a session, FPH module **116** uses the information stored in the entry **602** for the session to transmit FDP packets for that session in an automated manner that does not require CPU processing.

[0077] Timers information **608** in an entry stores the trx_interval timer and the last_sent timer values for the FDP session corresponding to the entry. The trx_interval timer value for the session is initialized by software executed by CPU **114**. As previously described, the trx_interval and last_sent timers are used by FPH module **116** to determine when to send an FDP packet for the session.

[0078] According to an embodiment of the present invention, a base timer may be associated with linked list **600** used by FPH module **116** to transmit FDP packets. The base timer for a linked list determines the interval at which the entries in

the linked list are visited and checked by FPH module **116**. For example, if the base timer for linked list **600** depicted in FIG. **6** is 5 msecs, then FPH module **116** visits each entry in the linked list every 5 msecs. FPH module **116** may start with one entry in the linked list and then use next pointer **612** to traverse through the various entries in the linked list. For a linked list with an associated base timer, the trx_interval and last_sent timer values stored for each entry in the linked list may be expressed as multiples of the base timer value. For example, if the base timer value associated with linked list **600** is 5 msecs, then the trx_interval and last_sent timer values in entries **602** may be expressed as a multiple of the base timer 5 msecs. For example, if the periodic time interval for transmitting an FDP packet for a session is 20 msecs, then trx_interval for that session may be expressed as (4* base timer). In one embodiment, the base timer for a linked list is determined based upon the trx_interval timers for the various session entries in the linked list as the least common denominator of the trx_interval timer values.

[0079] Command/status information **610** may store other information related to the FDP session. For example, if the FDP session requires any special processing then that information may be stored in information **610**.

[0080] Various other types of data structures may also be used to facilitate FDP packets transmission in alternative embodiments. For example, in one embodiment multiple linked lists may be used by FPH module **116** to facilitate transmission of FDP packets, each with its own associated base timer. In one embodiment using two linked lists, one linked list may have an associated base timer of 1 msec and the other may have an associated base timer of 50 msecs. A session may be allocated to one of the two linked lists based upon the trx_interval timer values associated with the session. For example, an FDP session having a trx_interval timer of 100 msecs may be allocated to the linked list having an associated base timer of 50 msecs whereas an FDP session having a trx_interval timer of 6 msecs may be allocated to the linked list having an associated base timer of 1 msec. In one embodiment, FPH module **116** may also be configured to transmit FDP packets for all the FDP session entries in a liked list at once (referred to as a "one-shot" transmission).

[0081] New entries may be added to a transmission linked list as more FDP sessions are initiated. In one embodiment, when a new entry is to be added to a linked list, the transmit functionality is disabled for the linked list to which the entry is to be added. In one embodiment, a one-shot transmission may be first performed for the linked list prior to the disabling. The new entry is then added to the linked list. The transmit operations for the linked list, now with the new entry for a new session, are then enabled. The trx_interval timer information for a session entry in the linked list may also be changed by software executed by the CPU of a network device.

[0082] As described above, FPH module **116** offloads some of the FDP packets-related processing that was conventionally performed by software executed by a CPU of a network device. In the Rx path, FPH module **116** determines if an FDP packet received by network device **102** needs to be provided to CPU **114** for processing. If it is determined that the FDP packet does not need to be forwarded to CPU **114** then the FDP packet is dropped. If instead, it is determined that the FDP packet needs to be forwarded to CPU **114** then FPH module **116** forwards the packet to CPU **114**.

[0083] Processing Using Dual Ring Structures

[0084] Various data structures may be used by FPH module **116** to facilitate processing of FDP packets in the Rx path. According to an embodiment of the present invention, a dual ring structure is used to facilitate the processing. A dual ring structure may comprise two rings, with each ring being a circular linked list of entries. FIG. **7** depicts a two ring structure **700** that may be used by FPH module **116** to facilitate processing of FDP packets in the receive (Rx) path according to an embodiment of the present invention. As depicted in FIG. **7**, structure **700** comprises a first ring **702** (referred to as a CPU_assist ring) and a second ring **704** (referred to as the CPU ring). CPU_assist ring **702** comprises a number of entries storing information related to FDP packets received by FPH module **116**. The processing of CPU_assist ring **702** is handled by FPH module **116**. CPU ring **704** comprises several entries storing information for FDP packets (and possibly for other packets) that are to be processed by CPU **114**. CPU **114** handles the processing of CPU ring **704**.

[0085] The number of entries in the two rings may be user-configurable. The entries are sometimes referred to as descriptor entries as they store information describing FDP packets. The number of entries in CPU_assist ring **702** is generally greater than the number of entries in CPU ring **704**.

[0086] CPU_assist ring **702** is used by FPH module **116** to process FDP packets received by the network device and by FPH module **116** in the Rx path. CPU_assist ring **702** comprises a number of entries ("m" entries depicted in FIG. **7**) for storing information used for processing FDP packets. As depicted in FIG. **7**, CPU_assist ring **702** may be implemented as a circular linked list of entries storing information related to FDP packets. The FDP packets themselves may be buffered in buffer memory **710**. When an FDP packet is received by a network device, the packet is stored in buffer memory **710** that is accessible to FPH module **116** and information corresponding to the buffered FDP packet is stored in an entry of CPU_assist ring **702**. FPH module **116** then manages processing of the FDP packets using the entries in CPU_assist ring **702**.

[0087] Buffer memory **710** used for buffering the FDP packets may be located in FPH module **116** or in some other location accessible to FPH module **116**. In embodiments where the memory resources of FPH module **116** are limited, buffer memory **710** may be stored for example in a memory (e.g., SDRAM **206**) associated with CPU **114**.

[0088] In one embodiment, each entry in CPU_assist ring **702** for an FDP packet buffered in memory **710** comprises the following information: (1) a buffer pointer **708** pointing to the location in buffer memory **710** storing the FDP packet corresponding to the entry; (2) a "processed bit" **706** indicating if the FDP packet corresponding to the entry has been processed by FPH module **116**; and (3) a next pointer **712** pointing to the next descriptor entry in CPU_assist ring **702**.

[0089] Processed bit **706** in an entry is used to identify the processing status of the FDP packet corresponding to the entry. In one embodiment, if the bit is set to 0 (zero), it indicates that the FDP packet corresponding to the entry needs to be processed. If the bit is set to 1 (one), it indicates that the FDP packet for the entry has already been processed and the entry is available for storing information for a new FDP packet. The bit is set to 1 (one) after the FDP packet has been processed.

[0090] A process_start_address pointer and a dma_start_addr pointer may also be provided (not shown in FIG. **7**) and

function as read and write pointers for CPU_assist ring **702** respectively. The dma_start_addr points to the entry in ring **702** that is available for storing information for an incoming FDP packet. The process_start_address points to the next entry in ring **702** that is available for storing information for an incoming FDP packet. FPH module **116** uses these pointers to traverse CPU_assist ring **702** and process entries corresponding to buffered FDP packets.

[0091] FPH module **116** traverses CPU_assist ring **702** at regular time intervals to process FDP packets corresponding to entries in CPU_assist ring **702**. For an unprocessed entry (as indicated by processed bit set to 0 in the entry), FPH module **116** uses the buffer pointer of the entry to access the corresponding FDP packet stored in buffer memory **710**. A portion of the FDP packet is then selected and compared to information stored in reference information for the FDP. As described above, if there is a match, it indicates that the FDP packet need not be provided to CPU **114** and can be dropped. In this event, processed bit **706** of the entry in CPU_assist ring **702** is set to 1 to indicate that the FDP packet corresponding to the entry has been processed and the FDP packet is dropped.

[0092] If there is no match, it indicates that the FDP packet is to be provided to CPU **114**. In this case, a buffer swap is performed between the buffer pointed to by the entry in CPU_assist ring **702** and a free entry in CPU ring **704**. In one embodiment, as a result of the swap, a buffer pointer in a previously available entry in CPU ring **704** is made to point to a buffer memory location pointed to by the buffer pointer in the entry in CPU_assist ring **702**. In this manner, after the buffer swap, a buffer pointer in an entry in CPU ring **704** now points to the location of the buffered FDP packet. For example, in FIG. **7**, buffer pointer **714** of CPU ring **704** points to the FDP packet stored in buffer memory **710**. CPU **114** may then access the FDP packet from buffer memory **710** and process the FDP packet. After the buffer swap, processed bit **706** in the entry in CPU_assist ring **702** is set to 1 to indicate that the entry is available for storing information for a new FDP packet and the buffer pointer for the entry is freed.

[0093] There may be situations where there are no available entries in CPU ring **704** for performing the buffer swap. This may occur for example when CPU **114** is backed up in its processing and is unable to process the FDP packets pointed to by entries in CPU ring **704** in a timely manner. This scenario may arise due to the rate at which FDP packets are received by the network device exceeding the rate at which CPU **114** is able to process the FDP packets. In such a scenario, FPH module **116** drops the buffered FDP packet corresponding to the entry in CPU_assist ring **702** whose pointer is to be swapped. FPH module **116** then continues processing of the next entry in CPU_assist ring **702** corresponding to the next unprocessed FDP packet. In this manner, FPH module **116** is able to continue processing the incoming FDP packets even if CPU **114** is backed up. This minimizes the number of incoming FDP packets that are dropped due to CPU **114** being busy.

[0094] The dual ring structure depicted in FIG. **7** and described above decouples receipt of FDP packets by network device **102** from processing of FDP packets by CPU **114** of the network device. FDP packets received by a network device are buffered in buffer memory **710** and corresponding entries stored in CPU_assist ring **702** which is handled by FPH module **116**. Buffering of FDP packets and processing of the packets by FPH module **116** is done separately from the processing of FDP packets performed by CPU **114** using CPU ring **704**. In this manner, CPU **114** may continue to process FDP packets (or perform other functions) using CPU ring **704** while FDP packets are being received and buffered by the network device. The decoupling enables FDP packets to be received without being concerned about the status of CPU **114**. Accordingly, FDP packets may be received by a network device at a rate that is faster than the rate at which the CPU of the network device can process the FDP packets. Even if CPU **114** is backed up processing FDP packets or performing other tasks, FDP packets may continue to be received and processed by FPH module **116** using CPU_assist ring **702**. As a result, incoming FDP packets do not have to be dropped due to CPU **114** being tied up with other processing activities (including processing of previously received FDP packets). This is particularly useful given the bursty nature of FDP packets. The decoupling also enables CPU **114** to process FDP packets without being hindered by the frequency at which the FDP packets are received by the network device. Further, only those FDP packets that need to be sent to CPU **114** are sent to CPU ring **704** from CPU_assist ring **702**. In this manner, CPU **114** does not see or process FDP packets that do not need to be sent to CPU **114**.

[0095] FIG. **8** is simplified block diagram of a FPH module **116** according to an embodiment of the present invention. FPH module **116** includes a number of modules including a TM Interface module **802**, a Packet Inspection module **804**, a Receive (Rx) Handler module **806**, a Transmit (Tx) Handler module **808**, and a CPU interface module **810**. FIG. **8** is merely illustrative of an embodiment incorporating the present invention and does not limit the scope of the invention as recited in the claims. One of ordinary skill in the art would recognize other variations, modifications, and alternatives. FPH module **116** may be incorporated in a network device such as a switch or router, such as routers and switches provided by Foundry Networks®, Inc. of Santa Clara, Calif.

[0096] TM Interface module **802** provides an interface for receiving packets from and transmitting packet to TM module **204**. In the Rx path, TM Interface module **802** receives CPU-bound packets, including FDP packets, from TM module **204**. The incoming packets may be buffered in Rx FIFO **816** for analysis. TM Interface module **802** identifies FDP packets from the CPU-bound packets received from TM module **204**. Packets that are not FDP packets are forwarded to Rx handler module **806** for forwarding to CPU **114**. For a packet identified as an FDP packet, TM Interface module **802** presents a portion of the FDP packet to Packet Inspection module **804** for analysis. In one embodiment, this is done by presenting an offset into the FDP packet to Packet Inspection module **804**. The offset is programmable and may be different for different FDPs. The portion of the FDP packet presented to Packet Inspection module **804** may include a portion of the header of the FDP packet, a portion of the payload of the FDP packet, or even the entire FDP packet. The portion of the FDP packet generally includes the session identifier for the packet.

[0097] Packet Inspection module **804** is configured to take the portion of the FDP packet received from TM Interface module **802** and compare information in the portion with reference information that has been programmed by software running on CPU **114**. The reference information may be stored by Packet Inspection module **804** (e.g., reference information **812** depicted in FIG. **8**) or alternatively may be stored in a memory location accessible to Packet Inspection module

804. In one embodiment, the reference information may store one or more session entries for different FDP sessions.

[0098] Packet Inspection module **804** may use an indexing scheme to perform the compare operation. In one embodiment, a part of the portion of the FDP packet received from TM Interface module **802** is used as an index into the reference information to identify an entry in the reference information corresponding to a particular session. The size of the index may vary based upon the number of session entries in the reference information. For example, a 9-bit index is needed for indexing 512 reference information entries. The information stored in a particular session entry identified using the index is then compared to the information in the portion of the FDP packet received from TM Interface module **802** to determine if there is a match. Results of the match are provided to TM Interface module **802**. The results identify whether or not the information in the portion of the FDP packet matched the information in the particular session entry indexed by the FDP packet portion.

[0099] As described above, TM Interface module **802** receives a result response from Packet Inspection module **804** indicating whether or not the FDP packet information matched the corresponding information in the reference information. If the received result indicates a match, then this indicates to TM Interface module **802** that the particular FDP packet can be dropped and need not be forwarded to CPU **114**. TM Interface module **802** then drops the FDP packet and flushes Rx FIFO **816** buffers corresponding to the FDP packet. The FDP packet is dropped without notifying CPU **114** about the packet. If the result received from Packet Inspection module **804** indicates that the FDP packet information did not match information in the session entry in the reference information, TM Interface module **802** forwards the FDP packet to Rx handler module **806** for forwarding to CPU **114**.

[0100] TM Interface module **802** is also configured to flag an error when an FDP packet for an FDP session is not received within a periodic time interval corresponding to the FDP session. In one embodiment, for each FDP session handled by the network device, TM Interface module **802** stores information (e.g., timers information) tracking when the last FDP packet for the session was received and when the next FDP packet is due to be received. If the next FDP packet for that session is not received within the time interval for that FDP session, then an error is flagged. CPU **114** may be notified about the error.

[0101] Rx Handler **806** is configured to receive CPU-bound packets, including FDP packets, from TM Interface module **802** and provide the packets to CPU Interface module **810** for forwarding to CPU **114**. Rx Handler **806** may also comprise a FIFO for storing the CPU-bound packets before being forwarded to CPU **114**.

[0102] CPU Interface module **810** is configured to forward packets to CPU **114**. In one embodiment, a DMA technique is used to forward packets to CPU **114**. In such an embodiment, CPU Interface Module **810** acts as a DMA engine that DMAs the packets to CPU **114**. In one embodiment, the packet is written to a memory **206** associated with CPU **114** from where the packet can be accessed by CPU **114**. Different interfaces may be used to forward packets from FPH module **116** to CPU **114**. For example, in one embodiment, a PCI bus interface may be used to forward packets to CPU **114**. In such an embodiment, CPU Interface Module **810** may comprise PCI-related modules for forwarding packets to CPU **114**. In one embodiment, the DMA engine is part of Rx handler **806** and CPU interface module **810** initiates the DMA process.

[0103] CPU Interface Module **810** is also configured to receive packets from CPU **114**. These packets are then forwarded to Tx Handler module **808**. Tx Handler module **808** may comprise a FIFO for storing the packets. The packets are then forwarded to TM Interface module **802**. In one embodiment, Tx Handler module **808** comprises a DMA engine that retrieves FDP packets from the CPU SDRAM. TM Interface module **802** may comprise a Tx FIFO **818** for storing the packets prior to transmission. A descriptor entries scheme may be used for retrieving and storing the packets. TM Interface module **204** then forwards the packets to TM module **204**. The packets may then be forwarded to the appropriate destination ports and transmitted from network device **102** via the destination ports.

[0104] According to an embodiment of the present invention, Tx Handler module **808** is configured to handle transmission of FDP packets from network device **102** for various FDP sessions. In one embodiment, Tx Handler module **808** maintains a pair of timers for each FDP session handled by the network device. As previously described, the pair of timers may include a trx_interval timer that indicates that transmission interval for transmitting an FDP packet for that FDP session and a last_sent timer that is used to monitor the time when an FDP packet was last transmitted by the network device for the FDP session. The two timers for each session are iteratively checked at periodic intervals to determine when to transmit an FDP packet for each session.

[0105] Tx Handler module **808** may use different structures to facilitate automated transmission of FDP packets. For example, in one embodiment, one or more circular linked lists (such as linked list **600** depicted in FIG. **6** and described above) may be provided to facilitate the transmission. Multiple linked lists may also be used, each with an associated base timer. The base timer for a linked list determines the frequency at which Tx Handler **808** visits and checks the entries in the linked list. For example, a list having an associated base timer of 1-msec is checked every 1-msec, a list having an associated base timer of 50 msecs is checked every 50 msecs, a list having an associated base timer of 200 msecs is checked every 200 msecs, and so on. In one embodiment, two linked lists are used: a first linked list having a 1 millisecond based timer and a second linked list having a 50 milliseconds base timer may be used. The entries in the 1-msec linked list are checked by Tx Handler module **808** every 1 msec. This linked list may store entries for FDP sessions whose periodic transmission intervals are multiples of 1 msec, e.g., 4 msecs, 15 msecs, etc. The 50-msec linked list entries are checked by Tx Handler module **808** every 50 msecs. This linked list may store entries for FDP sessions whose periodic transmission intervals are multiples of 50 msecs, e.g., 100 msecs, 250 msecs, etc.

[0106] The FDP packets transmitted by Tx Handler **808** are forwarded to TM Interface module **802** and then to TM module **204**. The FDP packets are then forwarded to one or more ports of the network device and then transmitted from the network device using the one or more ports.

[0107] As described above, embodiments of the present invention reduce the amount of FDP packets-related processing that a CPU of a network device has to perform. For incoming FDP packets, FPH module **116** assists the CPU by reducing the number of incoming FDP packets that a CPU has to process. FPH module **116** is also able to flag when an FDP

packet for an FDP session is not received within the periodic time interval for the session. FPH module **116** also handles transmission of FDP packets for various sessions at regular time intervals. The FDP packets transmission task is thus offloaded from the CPU of the network device. This further reduces the processing cycles that the CPU of the network device has to spend on FDP-packets related processing. This enables the network device to be able to support newer FDPs such as 802.1ag and BFD having very short periodic time intervals for transmission of FDP packets (e.g., faster than 1 second, more typically in milliseconds such as 1 millisecond, 5 milliseconds, or even shorter) without adversely affecting CPU performance. Accordingly, embodiments of the present invention enable a network device to process reception and transmission of FDP packets that may be received and transmitted at a rate faster than 1 FDP packet per second. Embodiments of the present invention are able to handle FDPs having periodic time intervals that may be one or more milliseconds (msecs), one or more seconds, or other shorter or longer time intervals.

[0108] As previously indicated, there are several different types of FDPs. Examples include BFD and 802.1ag. The following sections of the application describe embodiments of present invention for BFD and 802.1ag packets processing.

[0109] Processing of Bidirectional Forwarding (BFD) Protocol Packets

[0110] As previously described, BFD is a type of FDP. FIG. **9** depicts a format for a BFD packet. A BFD packet is delineated by a Start of Packet (SOP) and an End of Packet (EOP) field. A BFD packet is generally transmitted in a unicast, point-to-point mode. As depicted in FIG. **9**, a BFD packet comprises a Unicast header **902**, an internal header **904**, a Destination MAC **906**, a source MAC **908**, an EtherType field **910**, an IP header (IPv4 or IPv6) **912**, a UDP header **914**, a BFD header **916**, and data **918**.

[0111] EtherType Field **910** indicates whether the IP Header is IPv4 or IPv6. For example, a value of 0x0800 indicates IPv4 while 0x86DD indicates IPv6. In case of an IPv4 packet, IP header field **912** comprises 20 bytes of an IPv4 header (as specified by the IPv4 protocol). In case of an IPv6 packet, IP header field **912** comprises 40 bytes of an IPv6 header (as specified by the IPv6 protocol).

[0112] UDP header section **914** of the packet is used by FPH module **116** to identify a packet as a BFD packet. The following Table A shows the contents of UDP header section **914**.

TABLE A

UDP Header Format

Field	Size (bits)	Purpose
Checksum	16	UDP checksum.
Source Port	16	A free port on the sender's machine where any responses should be sent.
Destination Port	16	This field identifies the destination program on the server this packet should be directed to. This value is (decimal) 3784 for BFD control and (decimal) 3785 for BFD echo packets.
Message Length	16	Total size of UDP header plus data payload (but not the IP header) in 8-bit chunks (aka bytes or octets).

A BFD echo packet is addressed to the router who is sending it, so that the next-hop router will send the packet back to the initiating router. FPH module **116** uses the "Destination Port" field to identify a packet as a BFD packet.

[0113] FIG. **10** depicts the format for BFD header field **916**. The various fields in the BFD header field include:

[0114] (1) Version (3-bit): The version number of the protocol.

[0115] (2) Diagnostic (Diag) (5-bit): A diagnostic code specifying the local system's reason for the last session state change. This field allows remote systems to determine the reason that the previous session failed. Values are: 0—No Diagnostic; 1—Control Detection Time Expired; 2—Echo Function Failed; 3—Neighbor Signaled Session Down; 4—Forwarding Plane Reset; 5—Path Down; 6—Concatenated Path Down; 7—Administratively Down; 8—Reverse Concatenated Path Down; 9-31—Reserved for future use.

[0116] (3) State (STA) (2-bit): The current BFD session state as seen by the transmitting system. Values are: 0—AdminDown; 1—Down; 2—Init; 3—UpPoll (P) (1-bit): If set, the transmitting system is requesting verification of connectivity, or of a parameter change. If clear, the transmitting system is not requesting verification.

[0117] (4) Final (F) (1-bit): If set, the transmitting system is responding to a received BFD Control packet that had the Poll (P) bit set. If clear, the transmitting system is not responding to a Poll.

[0118] (5) Control Plane Independent (C) (1-bit): If set, the transmitting system's BFD implementation does not share fate with its control plane. If clear, the transmitting system's BFD implementation shares fate with its control plane.

[0119] (6) Authentication Present (A) (1-bit): If set, the Authentication Section is present and the session is to be authenticated.

[0120] (7) Demand (D) (1-bit): If set, the transmitting system wishes to operate in Demand Mode. If clear, the transmitting system does not wish to or is not capable of operating in Demand Mode.

[0121] (8) Reserved (R) (1-bit): This bit must be zero on transmit, and ignored on receipt. Detect Multiple (8-bit): Detect time multiplier. The negotiated transmit interval, multiplied by this value, provides the detection time for the transmitting system in Asynchronous mode.

[0122] (9) Length (8-bit): Length of the BFD Control packet, in bytes.

[0123] (10) My Discriminator (32-bit): A unique, nonzero discriminator value generated by the transmitting system, used to demultiplex multiple BFD sessions between the same pair of systems.

[0124] (11) Your Discriminator (32-bit): The discriminator received from the corresponding remote system. This field reflects back the received value of My Discriminator, or is zero if that value is unknown.

[0125] (12) Desired Min TX Interval (32-bit): This is the minimum interval (in microseconds) that the local system would like to use when transmitting BFD Control packets.

[0126] (13) Required Min RX Interval (32-bit): This is the minimum interval, in microseconds, between received BFD Control packets that this system is capable of supporting.

[0127] (14) Required Min Echo RX Interval (32-bit): This is the minimum interval, in microseconds, between received BFD Echo packets that this system is capable of supporting. If this value is zero, the transmitting system does not support the receipt of BFD Echo packets.

[0128] In the receive (Rx) path, TM Interface module **802** determines if an incoming packet is a BFD packet based upon the "Destination Port" information in the UDP header portion of the packet. TM Interface module **802** then determines if the BFD packet should be dropped (or terminated) or otherwise should be sent to CPU **114** for inspection. As part of this determination, TM Interface module **802** presents a portion of the BFD packet to Packet Inspection module **804**. Generally, an offset into the BFD packet is provided to Packet Inspection module **804**.

[0129] Packet Inspection module **804** then compares the information in the BFD packet portion received from TM Interface module **802** to information stored in the reference information to see if there is a match. In one embodiment, the reference information for BFD packets comparison is stored by FPH module **116** and comprises 512 BFD session entries, each entry 12-bytes long. FIG. **11** depicts a memory structure **1100** storing BFD reference information for multiple BFD sessions according to an embodiment of the present invention. As depicted in FIG. **11**, each entry **1102** stores information for a BFD session and has a 12-byte header **1104** and counters information **1106**. The BFD reference information for each session may be configured by software executed by CPU **114** and used by TM Interface module **802** for the comparison.

[0130] Packet Inspection module **804** uses an index to select which one of the 512 reference information session entries of structure **1100** is to be selected for comparison with BFD packet information. As previously described, a BFD packet has a 24-byte BFD header section **916**. The 24-byte header section or a portion thereof (e.g., a 12-byte section of the BFD header) may be used as an index. In one embodiment, the least significant 9 bits of the "Your Discriminator" of the BFD header section of a BFD packet that are unique per BFD session are used by the TM Interface module **802** as an index to the BFD reference memory structure. The size of the index depends upon the number of entries stored in the BFD reference information. In alternative embodiments, any unique portion of the BFD packet may be used as an index.

[0131] The index, which is based upon a portion of the BFD packet, is then used to identify a particular session entry in the BFD reference information. The reference information from the selected session entry is then compared to the information in the portion of the BFD packet received by Packet Inspection module **804** to see if there is a match. For example, for an entry in memory structure **1100** depicted in FIG. **11**, the 12 header bytes of the entry are used for the comparison. Packet Inspection module **804** then sends a signal to TM Interface module **802** indicating the result of the comparison.

[0132] If TM Interface module **802** receives a signal from Packet Inspection module **804** indicating a match, then the particular BFD packet is dropped or terminated. In this manner, the BFD packet is not forwarded to CPU **114**. If the signal received from Packet Inspection module **804** indicates that there was no match, it indicates to TM Interface module **802** that the BFD packet needs to be forwarded to CPU **114** for inspection and processing. TM Interface module **802** then forwards the BFD packet to CPU **114** via Rx Handler **806** and CPU Interface Module **810**.

[0133] FPH module **116** is also configured to flag non-receipt of BFD packets for a BFD session. In one embodiment, this may be performed by TM Interface module **802**. This is facilitated by counter information **1006** that is included in each session entry in the BFD reference information **1100** as depicted in FIG. **11**. FIG. **12** depicts contents of counter information **1106** for a BFD session entry according to an embodiment of the present invention. As depicted in FIG. **12**, counter information **1106** includes an interval_limit timer **1202** and a last_received timer **1204**. Field **1202** is typically programmed by software executed by CPU **114**. Interval_timer **1202** indicates the interval_limit within which FPH module **116** should receive an FDP packet for that session. Last_received timer **1204** is used by FPH module **116** to monitor the time that it has waited to receive an FDP packet for the session. When a BFD packet is received for a session (i.e., when the information for a received BFD packet matches the reference information in the session entry) the last_received timer in the entry is reset. Other timers may also be provided in alternative embodiments. For example, a timer may be provided to count the number of times that the interval_limit timer value has been exceeded.

[0134] FPH module **116** iteratively checks the counters information for the various BFD session entries in memory structure **1100**. When FPH module **116** determines for a session that last_received timer **1204** for the session reaches or exceeds interval_limit timer **1202**, it indicates that a BFD packet for the session was not received in the expected time interval and FPH module **116** signals this error condition to CPU **114**. In one embodiment, an interrupt is generated by FPH module **116** to flag that the particular BFD session has expired and there may be some failure in the network (e.g., a link is down). In this manner, non-reception of a BFD packet for a session is detected and flagged.

[0135] Counters information **1106** is monitored and checked by FPH module **116** on a periodic basis for the BFD entries. During a check, FPH module **116** walks through the session entries in the BFD reference information **1100** and checks the timers for each entry. In one embodiment, the frequency at which the checks are repeated is user-programmable. The frequency may also be determined automatically based upon the interval_limit timers for the sessions. For example, in one embodiment, the iteration frequency is set to the least common denominator of the various interval_limit timers being monitored for the various sessions by FPH module **116**. In this manner, the frequency is programmable and/or may be automatically determined from the interval_limit timers information for the BFD sessions in the reference information.

[0136] In the transmit (Tx) path, FPH module **116** is configured to perform automated transmission of BFD packets for the various BFD sessions. In one embodiment, this may be performed by Tx handler **808**. In one embodiment, a memory structure such as linked list **600** depicted in FIG. **6** and described above may be used to facilitate the automated transmission of BFD packets for different BFD sessions. In alternative embodiments, other types of memory structures may be used. The BFD packets transmitted by Tx Handler module **808** are forwarded to TM Interface module **802** and then to TM module **204**. The BFD packets are then forwarded to one or more ports of the network device. The BFD packets are then transmitted from the network device using the one or more ports of the network device.

[0137] Processing of 802.1ag Packets

[0138] As previously indicated, 801.1ag packets are a type of FDP packets. The 802.1ag standard specifies protocols, procedures, and managed objects to support transport fault management. These allow discovery and verification of the path, through bridges and LANs, taken for frames addressed to and from specified network users, detection, and isolation of a connectivity fault to a specific bridge or LAN. 802.1ag Connectivity Fault Management (CFM) provides capabilities for detecting, verifying and isolating connectivity failures in multi-vendor networks. FPH module **116** assists the CPU of a network device in processing 802.1ag packets, both in processing of incoming 802.1ag packets (e.g., determining if an 802.1ag packet should be forwarded to the CPU) and transmission of 802.1ag packets.

[0139] FPH module **116** is capable of supporting multiple types (e.g., five different types in one embodiment) of 802.1ag packets. FIGS. **13**A and **13**B depict formats for two types of 802.1ag packets that may be processed by an embodiment of the present invention. The format depicted in FIG. **13**A is for an 802.1ag packet received from VPLS/VLL uplink. The format depicted in FIG. **13**B is for an 802.1ag packet received from a regular link.

[0140] Each 802.1ag packet has a data section (e.g., section **1302** depicted in FIG. **13**A and section **1304** depicted in FIG. **13**B) that is used by FPH module **116** for analysis. FPH module **116** determines the offset within an 802.1ag packet to access the data section of the packet. FIG. **14** depicts contents of an 802.1ag data section. The data section comprises a "Sequence Number" field **1402** that is incremented each time an 802.1ag packet is transmitted. Accordingly, sequence number **1402** changes with each transmission of an 802.1ag packet. This changing field has to be taken into account when performing comparisons to determine whether an 802.1ag packet it to be forwarded to a CPU and also while transmitting 802.1ag packets.

[0141] FPH module **116** identifies a packet as an 802.1ag packet using an 802.1ag packet reference information table. FIG. **15** depicts an 802.1ag packet reference table **1500** according to an embodiment of the present invention. Table **1500** may be stored by FPH module **116** or in some location accessible to FPH module **116**. As depicted in FIG. **15**, each entry in table **1500** has the following content:

[0142] (1) Etype 1 (2-bytes): This is the first Etype field after the Source MAC in the 802.1ag packet header. This field is 0x8847 for VPLS/VLL uplink and 0x8902 for a regular link. This field is used in the comparison performed by FPH module **116** to determine if an 802.1ag packet needs to be sent to the CPU for processing and whether it can be dropped.

[0143] (2) Etype 2 Option (1-byte): This field consists of a 1-bit check field and a 7-bit offset field. If the check bit is set, the offset field indicates the number of bytes after Etype 1 where Etype 2 can be found. If the incoming packet is from a VPLS/VLL, FPH module **116** checks the MPLS label stack for the S bit. If the bit is 0, 4-bytes will get added to the offset field.

[0144] (3) Etype 2 (2-bytes): This is compared against an incoming packet's second Etype field if the check bit is set in the Etype 2 Option's field.

[0145] (4) Etype 3 Option (1-byte): This field consists of a 1-bit check field and a 7-bit offset field. If the check bit is set, the offset field indicates the number of bytes after Etype 1 where Etype 3 can be found. Etype 3 is considered to be 0x8902 and the 802.1ag Data always starts after it.

[0146] (5) SMAC Option (1-byte): This field consists of a 1-bit check field and a 7-bit offset field. If the check bit is set, the offset field indicates the number of bytes from the last Etype where SMAC can be found.

[0147] In one embodiment, FPH module **116** uses the 802.1ag packet reference table to determine if an incoming packet is an 802.1ag packet. If the packet is determined to be an 802.1ag packet, then the packet is buffered and an entry for the packet created in the CPU_assist ring depicted in FIG. **7**. The 802.1ag packet is then processed as previously described with regards to FIG. **7**. A portion of the 802.1ag packet is used to perform a comparison with reference information. As described above, if there is a match, it indicates that the 802.1ag packet need not be provided to CPU **114** and can be dropped. If there is no match, it indicates that the 802.1ag packet is to be provided to CPU **114**. In this case, a buffer swap is performed between the buffer pointed to by the entry in CPU_assist ring and a free entry in CPU ring, as previously described. CPU **114** may then access the 802.1ag packet from buffer memory **710** and process the packet.

[0148] The use of the dual ring structure depicted in FIG. **7** and described above may be used for processing incoming 802.1ag packets. This enables the network device to receive 802.1ag packets even when the CPU of the network device is unable to keep up with the processing of the packets. The incoming 802.1ag packets stored in CPU_assist ring **702** are processed by FPH module **116** to determine whether the packets need to be forwarded to the CPU of the network device for further processing.

[0149] In one embodiment, various checks are made to determine whether an 802.1ag packet needs to be forwarded to the CPU for processing. FPH module **116** first checks the Opcode field (depicted in FIG. **14**) of the packet. If Opcode is 1, the 802.1ag packet is a Continuity Check Message (CCM) and further comparisons are performed. Else, if the Opcode is not 1, then the packet is forwarded to the CPU for further processing.

[0150] As indicated above, if the 802.1ag packet is determined to be a CCM packet, then further comparisons are performed to determine if the packet needs to be forwarded to the CPU of the network device. In one embodiment, the reference information against which the comparisons are performed includes a hash table and a sessions table. The hash table and memory table may be stored in a memory location accessible to FPH module **116** and are initialized by software executed by the CPU of the network device. In one embodiment, the hash table and sessions table may be stored in RAM (e.g., SDRAM) associated with the CPU.

[0151] FIG. **16** depicts contents of a hash table **1602** and sessions table **1604** storing reference information according to an embodiment of the present invention. As depicted in FIG. **16**, as part of the processing, a portion **1606** of an 802.1ag packet is fed to a hash function **1608** to generate a hash index **1610**. In one embodiment, the portion of the 802.1ag packet that is fed to hash function **1608** includes bytes **9-10** (MEPID), and bytes **13-17** and bytes **36-58** (MAID) (see FIG. **14**) of the header of a CCM 802.1ag packet. In one embodiment, hash function **1608** yields an 8-bit hash value **1610** that represents an index to an entry **1612** within hash table **1602**. FPH module **116** then uses information in the entry within hash table **1602** to find an index to an entry in sessions table **1604**.

[0152] According to an embodiment of the present invention, each entry **1612** in hash table **1602** comprises packet reference information **1614** to be used for comparison, a session table pointer **1616** pointing to an entry in sessions table **1604**, and a next pointer **1618** pointing to the next entry in hash table **1602**. In one embodiment, packet reference information **1614** comprises 50 bytes corresponding to bytes **9-10** of MEPID, and bytes **13-17** and **36-58** of MAID from a CCM 802.1ag packet. In this embodiment, after an entry in hash table **1602** has been identified by hash index **1610** for a received 802.1ag packet, bytes **9-10** of MEPID, and bytes **13-17** and **36-58** of MAID from the received 802.1ag packet are compared with the 50 bytes **1614** stored in the hash table entry indexed by the 8-bit hash result. If the received packet information matches the packet reference information **1614** stored in the hash table entry, then session table pointer **1616** of the hash table entry is used to identify an entry in sessions table **1604**. If there is no match, then the entries in hash table **1602** may be traversed using next pointer **1618** until a matching packet reference information is identified or until all entries have been traversed. The session table pointer **1616** of the entry comprising the matching packet reference information is then used to identify an entry in sessions table **1604**. If no matching information is found in the hash table, then the received 802.1ag packet is forwarded to the CPU of the network device.

[0153] As depicted in FIG. **16**, sessions table **1604** may store a number of entries, each entry corresponding to an 802.1ag session. In one embodiment, sessions table **1604** stores 256 entries. According to an embodiment of the present invention, each entry in sessions table **1604** comprises the following information:

[0154] (1) Ownership (1-bit): The ownership bit is used to indicate if FPH module **116** can modify the sequence number of the entry. When the bit is set, the session can be used by FPH module **116**.

[0155] (2) Accept (1-bit): When this bit is set, FPH module **116** disregards the sequence number of the first 802.1ag packet matching the session. FPH module **116** saves the incremented CCM packet sequence number in the session entry and resets the accept bit.

[0156] (3) Version (5-bits): If the version field in the session table entry does not match the corresponding information in the received 802.1ag packet, the packet is forwarded to the CPU.

[0157] (4) Flags (8-bits): If this field in the session table entry does not match the corresponding information in the received 802.1ag packet, the received packet is sent to the CPU.

[0158] (5) Sequence number (32-bits): Software executing on the CPU initializes a value. When the Accept bit is set, FPH module **116** accepts whatever sequence number it sees and stores it in the session entry and then resets the Accept bit. When the Accept bit is not set (i.e., is zero), FPH module **116** checks to see if the sequence number of an incoming 802.1ag packet is 1 greater than the stored sequence number in the session table entry. If true, the sequence number of the incoming packet is correct. If the ownership bit is 1, the sequence number value is incremented and saved into the sessions table entry, so that the right value is available for comparison for that session for the next received 802.1ag packet that will also have an incremented sequence number. If sequence number of an incoming 802.1ag packet is not 1 greater than the stored sequence number in the session table entry, there is a potential problem and a packet with the correct expected sequence number may have been dropped. In this case, the new sequence number of the packet is stored in the session entry in the session status FIFO (described below), but the received 802.1ag packet is not forwarded to the CPU. The updating of the sequence number in the session table entry prevents every packet, after a mismatch of sequence numbers, of being treated as a mismatch and being forwarded to the CPU.

[0159] (6) Source Port (10-bits) and VC Label (20-bits): Source port of a packet is internal header bits **64-73** and VC label is MPLS stack bits **0-19**. An incoming 802.1ag packet is sent to the CPU if the field checks below against the session entry do not match:

[0160] i) For packets coming from VPLS/VLL link (outer Etype is 0x8847 and Inner Etype is 0x8902) FPH module **116** checks the VC label.

[0161] ii) For packets coming from regular link (Etype is 0x8902), FPH module **116** checks the Source Port.

[0162] (7) Source MAC (6-bytes): It can be outer or inner Source MAC.

[0163] (8) Session Status (8-bits): These bits are set by FPH module **116** and encode conditions related to the session corresponding to the sessions table entry. The conditions are:

8'h01=Session timeout

8'h02=Sequence number mismatch

Others=Reserved.

[0164] (9) SW timer (16-bits) (referred to above as the interval_limit timer): This field is set by software executed by the CPU and indicates the time interval that an 802.1ag packet is expected to be received in for that session. This may be expressed as a multiple of some base timer. This is same as the interval_limit timer previously described.

[0165] (10) HW timer (16-bits) (referred to above as the last-received timer): This field is used by FPH module **116** to keep track of aging, i.e., the time that an 802.1ag packet matching the session has not been received. It is reset by FPH module **116** every time it processes a matching 802.1ag packet whether it is dropped or sent to the CPU. This is same as the trx_interval timer previously described.

[0166] (11) Error counter (8-bits): This field is incremented by FPH module **116** every time there is a Sequence number mismatch. At a programmable interval, FPH module **116** writes the Session pointer of a session entry and the non-zero Error counter into the Session Status FIFO.

[0167] (12) Session counter (16-bits): This field keeps track of how many packets matching a session have been received.

This field is incremented by FPH module **116** every time a packet matching the session is received. Software executing on the CPU of the network device may reset the timer when it takes over the ownership of the session.

[0168] As previously described, if the received packet information matches the packet reference information **1614** stored in the hash table entry, session table pointer **1616** of the hash table entry is used to identify an entry in session table **1604**. The reference information in the session table entry is then compared to information in the packet. In one embodiment, one or more fields of the particular session table entry are compared to corresponding fields of the received packet. If the compared information matches, then the received packet is dropped and not forwarded to the CPU. If the compared information does not match, then the packet is forwarded to CPU. The fields of a session table entry that are compared to the corresponding information in a received packet may differ based upon the type of the received packet, for example, the type of 802.1ag packet. In this manner, a received 802.1ag packet is dropped if information from the packet matches reference information **1614** of an entry **1612** in hash table **1602** and information from the packet also matches reference information in a session table entry pointed to by session table pointer **1616** of the matching hash table entry **1612**—else, the packet is not dropped and forwarded to the CPU for processing.

[0169] As depicted in FIG. **16**, a linked list **1620** is provided that enables FPH module **116** to walk through the entries in sessions table **1604** in order to determine if an 802.1ag packet has not been received within the expected time interval for each of the sessions. For each session entry, FPH module **116** compares the hw_timer value for the entry with the sw_timer value for the entry. If the sw_timer and hw_timer are the same for a session entry, it indicates that an 802.1ag packet was not received within the time interval for that session and an error condition is flagged by FPH module **116**. In one embodiment, FPH module **116** writes the session pointer into the Session Status FIFO that stores session pointers of the sessions that have had issues such as session timeouts or session sequence number mismatch, etc. and generates an interrupt.

[0170] FIG. **17** depicts a format of a Session Status FIFO according to an embodiment of the present invention. As depicted in FIG. **17**, the session status FIFO entry comprises:

[0171] (1) Session Pointer (16-bits): Indicates the list significant 16 bits of the pointer where there is a sequence number mismatch.

[0172] (2) Error Counter (8-bits): It is the same value as the Error Counter field in the Session Table entry. Relevant for Sequence number mismatch.

[0173] (3) Used FIFO Entries (4-bits): Indicates the number of used FIFO entries in the Session Status FIFO in 32-bit increments i.e., 4'h0: 0-32 entries, 4'h1: 33-64 entries, . . 4'hF: 481-512.

[0174] (4) Status (4-bits). This field indicates the condition that the entry was recorded for. The values are: 4'h1: Session timeout; 4'h2: Sequence number mismatch; Others: Reserved.

[0175] In one embodiment, an 802.1ag packet is also sent to the CPU for processing if the following conditions below hold true: (1) The "First TLV Offset" (byte **4** of the 802.1ag data section of a packet) is less than 70; or (2) If the "First TLV Offset" is equal to or more than 70 and the "Optional CCM TLVs" (byte **75** of CCM data) is not zero.

[0176] As indicated in FIG. **14**, 802.1ag packets comprise a sequence number whose value is incremented with each transmitted 802.1ag packet for a session. Accordingly, when 802.1ag packets are transmitted, the sequence numbers of the packets have to be incremented with each packet transmission. According to an embodiment of the present invention, FPH module **116** performs processing to update the sequence number prior to transmission of **802.1** ag packets.

[0177] FIG. **18** depicts a linked list **1800** that may be used by FPH module **116** to facilitate transmission of 802.1ag packets according to an embodiment of the present invention. Linked list **1800** comprises a set of entries **1802** with each entry storing information for an 802.1ag session. Each entry comprises a buffer pointer **1804** pointing to a memory location **1820** storing the corresponding 802.1ag packet, buffer size information **1806** identifying the size of the corresponding 802.1ag packet, timers information **1808**, command/status information **1810**, offset information **1812**, other information **1814**, and a pointer **1816** pointing to the next entry in linked list **1800**. Pointer **1816** is used to traverse the linked list. In one embodiment, linked list **1800** is a circular linked list wherein pointer **1816** of the last entry in the linked list points to the first entry in the linked list.

[0178] Timers information **1808** in an entry stores the trx_interval timer and the last_sent timer values for the 802.1ag session corresponding to the entry. The trx_interval timer value for the session is initialized by software executed by CPU **114**. As previously described, the trx_interval and last_sent timers are used to determine when to send an 802.1ag packet for the session.

[0179] According to an embodiment of the present invention, a base timer may be associated with linked list **1800**. The base timer determines the interval at which the entries in linked list **1800** are visited and checked by FPH module **116**. For example, if the base timer for linked list **1800** depicted in FIG. **18** is 5 msecs, then FPH module **116** visits each entry in linked list **1800** every 5 msecs. For a linked list with an associated base timer, the trx_interval and last_sent timer values are expressed as multiples of the base timer value. For example, if the base timer value associated with linked list **1800** is 5 msecs and the periodic time interval for transmitting an 802.1ag packet for a session is 20 msecs, then trx_interval may be represented as (4* base timer). In one embodiment, the base timer value for a linked list may be determined based upon the trx_interval timers for the various session entries in the linked list.

[0180] Command/status information **1810** may store other information related to the 802.1ag session. For example, if the 802.1ag session requires any special processing then that information may be stored in information **1810**.

[0181] Offset information **1812** provides an offset into the 802.1ag packet pointed to by buffer pointer **1804** pointing to information in the packet that needs to be changed prior to periodic transmission of the packet. Offset information **1812** thus identifies the location within an 802.1ag packet that needs to be changed prior to transmission. For example, as previously described, the sequence number within an 802.1ag needs to be incremented with each transmitted 802.1ag packet. In such an embodiment, offset information **1812** may provide an offset to a location in the 802.1ag packet storing sequence number information that needs to be incremented with every transmitted packet. In other types of FDP packets, other one or more offsets may be provided that may be used to

access one or more sections or portions of the packet that need to be changed prior to transmission of the packet.

[0182] The sequence number information in an 802.1ag packet is incremented by FPH module **116** with each transmission of an 802.1ag packet for that session. The initial sequence number value may be set by software executed by the CPU of a network device. The sequence number information **1814** is then updated (e.g., incremented) after each 802.1ag packet transmission such that the updated value may subsequently be used for the next transmitted 802.1ag packet. In this manner, a parameter within an FDP packet may be updated with transmission of each FDP packet such that the correct parameter value is used for the next transmission.

[0183] FIG. **19** depicts a simplified flowchart **1900** showing a method performed by FPH module **116** for transmitting 802.1ag packets according to an embodiment of the present invention. The processing depicted in FIG. **19** is performed by FPH module **116** for each entry in a transmission linked list (e.g., linked list **1800** depicted in FIG. **18**) for every iteration when the entry is checked. For a session entry in the linked list, from timers information **1808** in the session entry, FPH module **116** checks if the last_sent timer has reached the trx_interval timer (step **1902**). If it is determined in **1902**, that the last_sent timer has not reached the trx_interval timer (i.e., the last_sent timer is less than the trx_interval timer), it is not time yet to transmit the 802.1ag packet and the last_sent timer is incremented (step **1904**). The amount by which the last_sent timer is incremented depends upon the base timer associated with the linked list. For example, if the base timer is 5 msecs, then the last_received timer is incremented by 5 msecs.

[0184] If it is determined in **1902** that the last_sent timer has reached or exceeded the trx_interval timer for the session, FPH module **116** prepares an 802.1ag packet for transmission. As part of this process, FPH module **116** first accesses the 802.1ag packet pointed to by buffer pointer **1804** of the entry corresponding to the session (step **1906**). Offset information **1812** of the linked list entry is then used to locate the sequence number field within the 802.1ag packet accessed in **1906** (step **1908**). An 802.1ag packet is then transmitted based upon the packet accessed in **1906** and having the sequence number located in **1908** (step **1910**). As part of **1910**, FPH module **116** may transmit the 802.1ag packet to TM module **204**. The 802.1ag packet may then be forwarded to a port of the network device and transmitted from the network device via the port.

[0185] The sequence number located in **1908** in the packet accessed in **1906** is then incremented by one (step **1912**). In this manner, an incremented sequence number is available for the next transmission of an 802.1ag packet for that session. The last_sent timer in timer information **1808** for the entry is then reset to zero (step **1914**) to restart the countdown to the next packet transmission.

[0186] As described above with respect to FIG. **19**, the sequence number of an 802.1ag packet stored in the buffer is incremented with each transmission of an 802.1ag packet. In this manner, 802.1ag packets with the correct sequence number are transmitted. The technique described above with respect to FIG. **19** may also be used to change one or more fields of FDP packets prior to transmission, as necessitated by the FDPs. In alternative embodiments of the present invention, multiple linked lists may be used by FPH module **116** to facilitate transmission of 802.1ag packets. Each linked list may have its own associated base timer.

[0187] In the examples provided above, a linked list memory structure was used to facilitate transmission of FDP packets. Embodiments of the present invention are however not restricted to using linked lists. Other types of memory structures may also be used in alternative embodiments.

[0188] Although specific embodiments of the invention have been described, various modifications, alterations, alternative constructions, and equivalents are also encompassed within the scope of the invention. The described invention is not restricted to operation within certain specific data processing environments, but is free to operate within a plurality of data processing environments. Additionally, although the present invention has been described using a particular series of transactions and steps, it should be apparent to those skilled in the art that the scope of the present invention is not limited to the described series of transactions and steps.

[0189] Further, while the present invention has been described using a particular combination of hardware and software, it should be recognized that other combinations of hardware and software are also within the scope of the present invention. The present invention may be implemented only in hardware, or only in software, or using combinations thereof.

[0190] The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense. It will, however, be evident that additions, subtractions, deletions, and other modifications and changes may be made thereunto without departing from the broader spirit and scope of the invention as set forth in the claim.

What is claimed is:

1. A method of transmitting packets according to a failure detection protocol, the method comprising:

storing, at a network device, timer information specifying a periodic time interval for transmitting a failure detection protocol packet for a failure detection protocol; and

periodically transmitting failure detection protocol packets for the failure detection protocol at times determined based upon the periodic time interval stored for the failure detection protocol;

wherein the transmitting is performed by a module of the network device other than a processor configured to execute software for processing failure detection protocol packets.

2. The method of claim **1** wherein the transmitting comprises:

determining a time for transmitting a failure detection protocol packet for the failure detection protocol based upon the periodic time interval stored for the failure detection protocol; and

transmitting a failure detection protocol packet at the determined time.

3. The method of claim **1** wherein the timer information comprises a first timer specifying the periodic time interval for the failure detection protocol and a second timer identifying when a failure detection protocol packet was last transmitted by the network device for the failure detection protocol.

4. The method of claim **3** wherein the transmitting comprises transmitting a failure detection protocol packet when the second timer equals the first timer.

5. The method of claim **3** wherein the first timer is set by the software executed by the processor of the network device and the second timer is updated by the module.

6. The method of claim 1 wherein each failure detection protocol packet for the failure detection protocol comprises a field, and wherein the transmitting comprises:

transmitting a first failure detection protocol packet based upon the periodic time interval stored for the failure detection protocol, wherein the field of the first failure is set to a first value; and

transmitting, after transmitting the first failure detection protocol packet, a second first failure detection protocol packet based upon the periodic time interval stored for the failure detection protocol, wherein the field of the second failure detection protocol packet is set to a second value that is different from the first value.

7. The method of claim 6 further comprising:

storing a parameter for the failure detection protocol;

prior to transmitting the first failure detection protocol packet, setting the field of the first failure detection protocol packet to the first value based upon the value of the parameter;

updating the value of the parameter upon transmission of the first failure detection protocol packet; and

prior to transmitting the second failure detection protocol packet, setting the field of the second failure detection protocol packet to the second value based upon the updated value of the parameter.

8. The method of claim 1 wherein the failure detection protocol is 802.1ag.

9. The method of claim 1 wherein the failure detection protocol is Bidirectional Forwarding (BFD) protocol.

10. The method of claim 1 wherein the periodic time interval is less than one second.

11. A system for transmitting packets according to a failure detection protocol, the system comprising:

a memory configured to store timer information specifying a periodic time interval for transmitting a failure detection protocol packet for a failure detection protocol;

a processor configured to execute software for processing failure detection protocol packets.; and

a module configured to periodically transmit failure detection protocol packets for the failure detection protocol at times determined based upon the periodic time interval stored for the failure detection protocol.

12. The system of claim 11 wherein the module is configured to:

determine a time for transmitting a failure detection protocol packet for the failure detection protocol based upon the periodic time interval stored for the failure detection protocol; and

transmit a failure detection protocol packet at the determined time.

13. The system of claim 11 wherein the timer information comprises a first timer specifying the periodic time interval for the failure detection protocol and a second timer identifying when a failure detection protocol packet was last transmitted by the system for the failure detection protocol.

14. The system of claim 13 wherein the module is configured to transmit a failure detection protocol packet when the second timer equals the first timer.

15. The system of claim 13 wherein the first timer is set by the software executed by the processor of the network device and the second timer is updated by the module.

16. The system of claim 11 wherein each failure detection protocol packet for the failure detection protocol comprises a field, and wherein the module is configured to:

transmit a first failure detection protocol packet based upon the periodic time interval stored for the failure detection protocol, wherein the field of the first failure is set to a first value; and

transmit, after transmission of the first failure detection protocol packet, a second first failure detection protocol packet based upon the periodic time interval stored for the failure detection protocol, wherein the field of the second failure detection protocol packet is set to a second value that is different from the first value.

17. The system of claim 16 wherein the module is configured to:

store a parameter for the failure detection protocol;

prior to transmission of the first failure detection protocol packet, set the field of the first failure detection protocol packet to the first value based upon the value of the parameter;

update the value of the parameter upon transmission of the first failure detection protocol packet; and

prior to transmission of the second failure detection protocol packet, set the field of the second failure detection protocol packet to the second value based upon the updated value of the parameter.

18. The system of claim 11 wherein the failure detection protocol is 802.1ag.

19. The system of claim 11 wherein the failure detection protocol is Bidirectional Forwarding (BFD) protocol.

20. The system of claim 11 wherein the periodic time interval is less than one second.

21. The system of claim 11 wherein the module is a field-programmable logic device.

22. A network device comprising:

a processor configured to execute software for processing failure detection protocol packets, the processor configured to set a periodic time interval for transmitting a failure detection protocol packet for a failure detection protocol;

a module configured to periodically transmit failure detection protocol packets for the failure detection protocol at times determined based upon the periodic time interval stored for the failure detection protocol; and

one or more ports configured to transmit the failure detection protocol packets from the network device.

* * * * *