(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
7 October 2004 (07.10.2004)

PCT

(10) International Publication Number
WO 2004/086177 A2

(51) International Patent Classification[7]: G06F

(21) International Application Number:
PCT/US2004/008041

(22) International Filing Date: 16 March 2004 (16.03.2004)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
10/396,985 24 March 2003 (24.03.2003) US

(71) Applicant *(for all designated States except US)*: **EMULEX DESIGN & MANUFACTURING CORPORATION** [US/US]; 3333 Susan Street, Costa Mesa, CA 92626 (US).

(72) Inventor: **WILLIAMS, James, B.**; c/o Emulex Corporation, 3333 Susan Street, Costa Mesa, CA 92626 (US).

(74) Agents: **KUBOTA, Glenn, M.** et al.; Morrison & Foerster LLP, 555 W. Fifth Street, Suite 3500, Los Angeles, CA 90013 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: DIRECT DATA PLACEMENT

(57) Abstract: A system comprising a host and a network interface card or host bus adapter. The host is configured to perform transport protocol processing. The network interface card is configured to directly place data from a network into a buffer memory in the host.

WO 2004/086177 A2

DIRECT DATA PLACEMENT

BACKGROUND

[0001] Transmission Control Protocol (TCP)/Internet Protocol (IP) is a networking protocol that provides communication across interconnected networks, between computers with diverse hardware architectures and various operating systems. The TCP/IP family of protocols track Internet addresses of nodes, routes outgoing messages and recognizes incoming messages. TCP is a connection-oriented, end-to-end transport layer protocol used to transfer data in a network. IP addresses packets and is the messenger protocol of the TCP/IP family of protocols.

[0002] A conventional network interface card (NIC) receives TCP/IP packets from a network and stores the packets in a network interface card memory. A conventional host operating system (OS) copies packets from the network interface card memory to a host memory. A host TCP/IP stack handles TCP/IP protocol processing of the TCP/IP packets. Copying data from the network interface card memory to the host memory may consume a large amount of the host's processing power and is referred to as "overhead."

[0003] The host system may use TCP/IP headers to determine a "connection" associated with each packet. If the TCP/IP packets

are received from the network out of order, the host system may use TCP packet sequence numbers in TCP headers to put the TCP/IP packets in a proper order. The host system may also reassemble data (messages) that the network interface card receives in pieces.

[0004] As an alternative to the conventional host system, the host system may have a full TCP transport "offload," where the network interface card does all transport protocol processing instead of the host. This may enable zero copying of received data packets. The network interface card reassembles data packets, replaces out-of-place data packets, identifies the connection to which the packet belongs, and places the data in an appropriate buffer location in host memory. This full transport offload network interface card, however, may be fairly expensive, especially if the network interface card needs to handle a large number of connections and maintain context/connection state information for all connections. The network interface card needs to have high memory bandwidth to access context information to do transport processing. In addition, a host TCP/IP stack needs to be modified to account for the transport offload.

[0005] Direct Data Placement (DDP) is a developing protocol described in the "DDP Protocol Specification," published by an Internet Engineering Task Force (IETF) working group on October

21, 2002 (hereinafter "DDP Specification"). DDP may enable an Upper Layer Protocol (ULP) to send data to a Data Sink without requiring the Data Sink to place the data in an intermediate buffer. When data arrives at the Data Sink, a network interface can place the data directly into the ULP's receive buffer. This may enable the Data Sink to consume substantially less memory bandwidth than a buffered model because the Data Sink is not required to move the data from an intermediate buffer to the final destination. This can also enable the network protocol to consume substantially fewer CPU cycles than if the CPU was used to move data, and remove the bandwidth limitation of being only able to move data as fast as the CPU can copy the data.

SUMMARY

[0006] The present application relates to a network interface card (NIC) or host bust adapter (HBA) and a method for direct data placement (DDP) without transport protocol processing offload. The system may have a number of advantages, such as reducing host overhead for copying data, reducing the cost of a network interface card and improving host and network interface card performance.

[0007] The system and methods described herein may modify the host network stack with practical modifications that do not break any fundamental assumptions. In contrast, host stack

modifications that would support full, clean, and seamless TCP offload, i.e., for a network interface card to handle TCP/IP processing and data placement, may be significantly harder and more expensive to implement. A direct data placement-only network interface card may be considerably less expensive than a full TCP offload because context memory size and bandwidth is minimized for a direct data placement-only network interface card. Payload buffering is not required for a direct data placement-only network interface card or a full TCP offload network interface card.

[0008] The system and methods described above may also enable a desired usage mode, which would otherwise be difficult with a full TCP offload. The usage mode allows an initial connection of legacy protocol (such as Small Computer System Interface (SCSI), Small Computer System Interface over Internet Protocol (iSCSI) or Network File System (NFS)) to be established to an associated well-known port number. Then in-band negotiation is performed to upgrade the connection to use direct data placement. After successful negotiation of direct data placement, the connection may transition to DDP mode. Backward compatibility may be difficult to maintain without this capability.

[0009] An aspect of the application relates to a system comprising a host system and a network interface card (NIC).

The host system comprises a host transport protocol processing stack and a memory. The network interface card is configured to receive packets from a network, send a header of each packet to the host transport protocol processing stack, and directly place a payload of each packet in the host memory.

[0010] Another aspect relates to a network interface card comprising a direct data placement engine and a memory storing connection state information. The direct data placement engine is configured to read a header of a packet received from a network, access the connection state information, determine whether a payload of the packet can be directly placed in a host memory, send the header to a host protocol processing stack, and directly place the payload in the host memory.

[0011] Another aspect relates to a method comprising: reading a header of a packet received from a network; determining whether packet data is authorized to be directly placed in a host memory; if the packet data is authorized to be directly placed in the host memory, placing the packet data directly in the host memory and sending a packet header to a host transport protocol processing stack; and if the packet data is not authorized to be directly placed in the host memory, sending the packet to the host transport protocol processing stack.

[0012] The details of one or more embodiments are set forth in the accompanying drawings and the description below. Other

features and advantages will be apparent from the description and drawings, and from the claims.

DESCRIPTION OF DRAWINGS

[0013] Fig. 1 illustrates a host system and a network interface card (NIC), a bus and a network connection.

[0014] Fig. 2 illustrates a packet that the network interface card of Fig. 1 may receive from the Ethernet connection.

[0015] Fig. 3 illustrates a method of direct data placement with the system of Fig. 1.

[0016] Fig. 4 illustrates a method of determining whether packets are in order, handling out-of-order packets and recovering after packets are back in order.

DETAILED DESCRIPTION

[0017] Fig. 1 illustrates a host system 100 and a network interface card (NIC) or host bus adapter (HBA) 102, a bus 104 and a network connection 106. The bus 104 may be a Peripheral Component Interface (PCI) bus, a PCI-X bus, a Small Computer System Interface (SCSI) bus or some other type of bus. The network connection 106 may be coupled to an Ethernet network with servers, user computers, storage devices, network attached storage (NAS), storage area networks (SANs), routers and other devices.

[0018] The host system 100 may include a direct data placement (DDP) stack 110, a Transmission Control Protocol/Internet Protocol (TCP/IP) stack 112, an offload detection module 116 and a memory 118. A host CPU or processor may execute the direct data placement (DDP) stack 110, TCP/IP stack 112, and offload detection module 116. The DDP stack 110 and offload detection module 116 may be separate from the TCP/IP stack 112 or may be combined with the TCP/IP stack 112.

[0019] The host memory 118 has a context storage 114 that stores context or connection state information, which is described below. The context storage 114 may be a part of the memory 118 or separate from the memory 118.

[0020] The host memory 118 may have a buffer 119 that stores packet data payloads 200 (Fig. 2 described below) that are associated with a particular host application or connection, i.e., source and destination Internet Protocol (IP) addresses and port numbers. The memory 118 may have a set of buffers for a set of connections.

[0021] The network interface card 102 may include a direct data placement (DDP) engine 120, a memory 124 and a network (e.g., Ethernet) connection 106. The DDP engine 120 may include firmware and/or hardware, such as a processor. The network interface card memory 124 may include a context cache or storage 122, tables, and buffer lists.

[0022] Packet

[0023] Fig. 2 illustrates a packet 200 that the network interface card 102 may receive from the network connection 106. The packet 200 may include an IP header 202, a TCP header 204, a DDP header 206 and payload data 208. The IP header 202 may include a source IP address and a destination IP address. The TCP header 204 may include a source port ID/number, a destination port ID/number, a checksum, a packet sequence number and other control information. The IP and TCP headers 202 and 204 provide sufficient information on where the packet payload 208 is supposed to be stored in the host buffer 119.

[0024] The DDP header 206 may indicate permission or authorization for the network interface card 102 to directly write data payloads 208 associated with a connection to the buffer 119. The packet 200 shows a Direct Data Placement (DDP) protocol run over a TCP/IP network.

[0025] NIC Receives Packets In Order

[0026] Fig. 3 illustrates a method of direct data placement with the system of Fig. 1. When the network interface card 102 receives packets in order via the Ethernet connection 106, the network interface card 102 has sufficient context information in storage 122 to directly place the packet payloads 208 in the host buffer 119. The network interface card 102 may transfer two types of packets from the network connection 106 to the host

TCP/IP stack 112: header-only packets; and header and payload packets.

[0027] The DDP engine 120 reads the DDP header 206 in 300 of Fig. 3 and determines whether the DDP engine 120 has permission to directly place the payload 208 (associated with a particular connection) into the host's buffer 119 via a bus or line 105. If not, the DDP engine 120 may pass the whole packet 200 to the host TCP/IP stack 112 in 302.

[0028] If direct placement is permitted, the DDP engine 120 of the network interface card 102 passes the IP header 202, TCP header 204 and some DDP header information to the host network TCP/IP stack 112 for processing. The DDP engine 120 separates payload data 208 from headers 202, 204, 206. The DDP engine 120 places the payload data 208 directly into the buffer 119 in the memory 118 of the host system 100 according to information in the IP header 202, TCP header 204 and DDP header 206. Thus, the network interface card 102 may do direct data placement (DDP) in the host buffer 119, but not transport (e.g., TCP) offload.

[0029] The network interface card DDP engine 120 may set a flag bit in the DDP header 206 sent to the host TCP/IP stack 112. The host's offload detection module 116 may detect the flag bit for a header-only packet transferred from the network interface card 102 to the host system 100 and acknowledge that

the network interface card 102 directly placed data in the host memory buffer 119.

[0030] The network interface card DDP engine 120 may check the TCP sequence number in the TCP header 204 to determine if the packet 200 is a "valid" packet, which means the packet is "in sequence" (in order). If the packet 200 is a retransmission of an old packet, the packet may be invalid and invalid packets will be dropped. If the packet 200 is out of sequence, the DDP engine 120 sends the entire packet to the host TCP/IP stack 112.

[0031] Context Information stored in Network Interface Card

[0032] The DDP engine 120 may identify a connection to which the packet is associated by accessing minimal TCP context information stored in the network interface card context storage 122. The context storage 122 may maintain a total of, for example, 16 to 32 bytes of context information (described below) per connection, in contrast to 256 to 512 bytes of context information per connection for full TCP offload (if TCP processing is handled by a network interface card).

[0033] The minimal TCP "context" or "connection state" information may include for each DDP connection: (a) a local IP address and port number; (b) a remote IP address and port number; (c) a sequence number of the next TCP packet expected by the connection; and (d) a protection ID (e.g., PTag). The protection ID is a mechanism for protecting the host memory

buffer 119 and checking all accesses to the host memory buffer 119 for permission. The protection ID indicates whether the network interface card 102 has permission to write data directly to the host buffer 119.

[0034] The DDP Specification describes a PTag (protection tag), which is an example of a protection ID for protecting the memory buffer 119 from unauthorized writes. There is a PTag associated with each buffer 119 in the host memory 118, and another PTag associated with the connection. If the two PTags match, then the connection is associated with that buffer 119.

[0035] The context information may include a STag (steering tags) for each buffer in the memory 118. STags are described in the DDP Specification. The DDP header 206 (Fig. 2) may include a STag. The STag identifies a Data Sink's tagged ULP buffer, such as the host buffer 119. The STag directs the DDP engine 120 to write data to the buffer 119 referenced by the STag.

[0036] The context information may further include a next_expected_sequence, a recovery_point and an in-order flag, as described below with Fig. 4.

[0037] The memory 124 may store a source IP address, a destination IP address, a source port number, and a destination port number (collectively called a "four-tuple") for each connection for which the network interface card 102 is doing direct data placement. Four-tuples are used to identify

incoming packets and associate each packet with a connection. An implementation has a plurality of four-tuples stored in a hash table 126 or other associative lookup structure in the network interface card memory 124, instead of the network interface card context cache 122. The output of the hash table 126 may be a pointer or other means to access the context in the context cache 122 for the identified connection. Thus, the four-tuple is stored in the network interface card memory 124 and is part of the "context" associated with a connection. But a different mechanism, such as hash tables 126, is used to store the four-tuple, so the four-tuple is not part of what is referred to as the "connection context."

[0038] The network interface card's context storage 122 may also maintain a small amount of DDP context information for each connection, including a current buffer (start address of a host buffer 119), a current offset (memory address offset from the buffer start address), and bytes remaining to be transferred to the buffer 119, since DDP "blocks" may span multiple TCP "segments."

[0039] Host Stacks

[0040] The host network stack 112 may continue to do all TCP/IP protocol processing, other than copying payloads 208 of connections authorized for network interface card 102 to handle DDP to destination buffers 119. Copying, however, may be a

majority of the host network stack's work if the host does not have a network interface card 102 that does direct data placement. As an example, 50% host CPU offload may be achieved by having the network interface card 102 do direct data placement.

[0041] If the network interface card DDP engine 120 stops doing DDP because of an invalid packet, the host TCP/IP stack 112 may send a signal to instruct the network interface card DDP engine 120 to resume DDP.

[0042] The host DDP stack 110 may process information in the DDP header 206 and handles DDP protocol functions, as described in the DDP Specification.

[0043] Dropped Packet

[0044] If the network drops a packet, the network interface card 102 may (a) stop DDP and note where in a sequence a packet is missing or (b) resume direct placement on a subsequent packet. When the missing packet is retransmitted across the Ethernet network and reaches the network interface card 102, the network interface card 102 may send the retransmitted packet to the host stack 112, which would copy the retransmitted packet to its intended destination in the memory 118.

[0045] Alternatively, the network interface card 102 may be able to process and transport the retransmitted packet if the network interface card 102 has context information devoted to

tracking gaps in a packet sequence and TCP acknowledgements (ACKs) for packets. A TCP ACK notifies the network interface card 102 that everything in a sequence up to a particular packet has been received by a destination buffer 119. The network interface card 102 may watch the TCP ACKs to determine when DDP may be resumed.

[0046] NIC Receives Packets Out of Order

[0047] The network interface card context storage 122 may maintain the following state information for each connection:

[0048]	next_expected_sequence	32 bit sequence number
[0049]	recovery_point	32 bit sequence number
[0050]	in_order	flag: TRUE or FALSE

[0051]

[0052] Fig. 4 illustrates a method of determining whether packets are in order, handling out-of-order packets and recovering after packets are back in order. The following algorithm may be used on each received packet 200. Each incoming TCP packet 200 contains a sequence number SEQ and a length LEN. Next_expected_sequence represents the sequence number the network interface card 102 expects to see in the next packet and indicates if the next packet is in order. The next_expected_sequence may always be set to the sequence number of the last packet received on a connection plus the length of that packet in 400.

[0053] next_expected_sequence = SEQ + LEN;

[0054]

[0055] The network interface card 102 compares the sequence number of each packet with next_expected_sequence in 402. If the sequence number of the next packet is not equal to next_expected_sequence, the packet is out of order in 404. The recovery_point is set to the sequence number of the last incoming packet that was out of order, and is not changed when subsequent packets are received in order. The in_order flag is set to false in 404.

[0056] if (SEQ != next_expected_sequence)

[0057] // this is an out-of-order packet

[0058] recovery_point = SEQ

[0059] in_order = FALSE

[0060] On each packet transmitted to the host 100 by the network interface card 102, the network interface card 102 checks an ACK field to determine if the network interface card 102 can restart "in order" processing in 406. When the network interface card 102 sees an outgoing packet containing an ACK greater than or equal to the recovery_point in 410, the network interface card 102 knows that all packets up to that sequence number have been received. The network interface card 102 also knows that it has seen all packets since that point in order. Therefore, if the network interface card 102 had stopped direct

placement due to out of order packets being received, the network interface card 102 may now resume direct placement in 410.

[0061] if (ACK >= recovery_point)

[0062] in_order = TRUE

[0063] if (in_order is TRUE)

[0064] do direct placement

[0065] else

[0066] pass full packet to host for processing in 408.

[0067] A number of embodiments have been described. Nevertheless, it will be understood that various modifications may be made without departing from the spirit and scope of the application. For example, the description above assumes an underlying TCP/IP network, but other types of protocols, standards, packet types and networks may be used. For example, the systems and methods described herein may be applied to Simple Computer Telephony Protocol (SCTP), Virtual Interface (VI) over TCP/IP, Fibre Channel or iSCSI.

[0068] In addition, although DDP from the "DDP Specification" is described herein, the host and network interface card may use other information in a packet to enable the network interface card to directly place packet payloads into appropriate locations in a host buffer and reduce the amount of copying by the host of data from a network interface card intermediate

buffer. Accordingly, other embodiments are within the scope of the following claims.

WHAT IS CLAIMED IS:

1. A system comprising:

a host system comprising a host transport protocol processing stack and a memory; and

a network interface card configured to receive packets from a network, send a header of each packet to the host transport protocol processing stack, and directly place a payload of each packet in the host memory.

2. The system of Claim 1, wherein the transport protocol processing stack is a Transmission Control Protocol/Internet Protocol processing stack.

3. The system of Claim 1, wherein the host system memory stores connection state information accessible to the transport protocol processing stack.

4. The system of Claim 1, wherein the host system further comprises a direct data placement stack configured to process direct data placement headers sent by the network interface card.

5. The system of Claim 1, wherein the host system further comprises an offload detection module configured to detect whether the network interface card directly placed a packet payload in the host memory.

6. The system of Claim 1, wherein the host memory comprises a plurality of buffers, each buffer being associated with an Internet Protocol address and port connection.

7. The system of Claim 1, wherein the network interface card is configured to send an entire packet to the host transport protocol processing stack if the network interface card is not authorized to directly place a payload of the packet in the host memory.

8. The system of Claim 1, wherein the network interface card is configured to send an entire packet to the host transport protocol processing stack if the network interface card determines that the packet is out of order.

9. The system of Claim 1, further comprising a network coupled to the network interface card.

10. A network interface card comprising:

a memory storing connection state information; and

a direct data placement engine capable of receiving a packet from a network, reading a header of a packet received from the network, accessing the connection state information, determining whether a payload of the packet can be directly placed in a host memory, sending the header to a host protocol processing stack, and directly placing the payload in the host memory.

11. The network interface card of Claim 10, wherein the packet comprises a Transmission Control Protocol header, an Internet Protocol header and a direct data placement header.

12. The network interface card of Claim 10, wherein the connection state information comprises:

a local Internet Protocol address and port number;

a remote Internet Protocol address and port number;

a sequence number of the next packet expected by the connection; and

a protection identification that authorizes the network interface card to directly place the payload in the host memory.

13. The network interface card of Claim 12, wherein the connection state information further comprises an steering tag.

14. The network interface card of Claim 10, wherein the memory further stores a source Internet Protocol address, a destination Internet Protocol address, a source port number, and a destination port number for each connection for which the network interface card directly places data into the host memory.

15. The network interface card of Claim 14, further comprising a hash table storing the source Internet Protocol address, destination Internet Protocol address, source port number, and destination port number.

16. A host bus adapter comprising firmware and a memory configured to store connection state information, the firmware being configured to read a header of a packet from a network, access the connection state information, determine whether a payload of the packet can be directly placed in a host memory, send the header to a host transport processing stack, and directly place the payload in the host memory.

17. A host system comprising:

a memory configured to store packet payloads; and

a Transmission Control Protocol/Internet Protocol stack configured to process Transmission Control Protocol/Internet Protocol headers and detect whether a network interface card has directly placed packet payloads in the memory.

18. The host system of Claim 17, wherein the memory further stores connection state information.

19. A method comprising:

reading a header of a packet received from a network;

determining whether packet data is authorized to be directly placed in a host memory;

if the packet data is authorized to be directly placed in the host memory, placing the packet data directly in the host memory and sending a packet header to a host transport protocol processing stack; and

if the packet data is not authorized to be directly placed in the host memory, sending the packet to the host transport protocol processing stack.

20. The method of Claim 19, further comprising identifying a connection associated with a packet.

21. The method of Claim 19, further comprising accessing connection state information to determine whether packet data is authorized to be directly placed in a host memory.

22. The method of Claim 19, further comprising accessing connection state information to find a location for directly placing packet data in the host memory.

23. The method of Claim 19, further comprising:

determining whether the packet is in sequence by comparing a sequence number of the packet with an expected sequence number;

if the packet is out of sequence, sending the packet to the host transport processing stack; and

if the packet payload is in sequence, placing the payload directly in the host memory and sending a packet header to a host transport processing stack.

24. The method of Claim 19, further comprising:

checking an acknowledgement field of a packet sent to the host transport processing stack;

if the acknowledgement field is greater than or equal to a recovery sequence number, then begin placing payloads directly in the host memory and sending packet headers to the host transport processing stack; and

if the acknowledgement field is less than a recovery sequence number, then send the packet to the host transport processing stack.

25. The method of Claim 19, further comprising maintaining a next expected sequence variable, a recovery sequence number an in order flag.

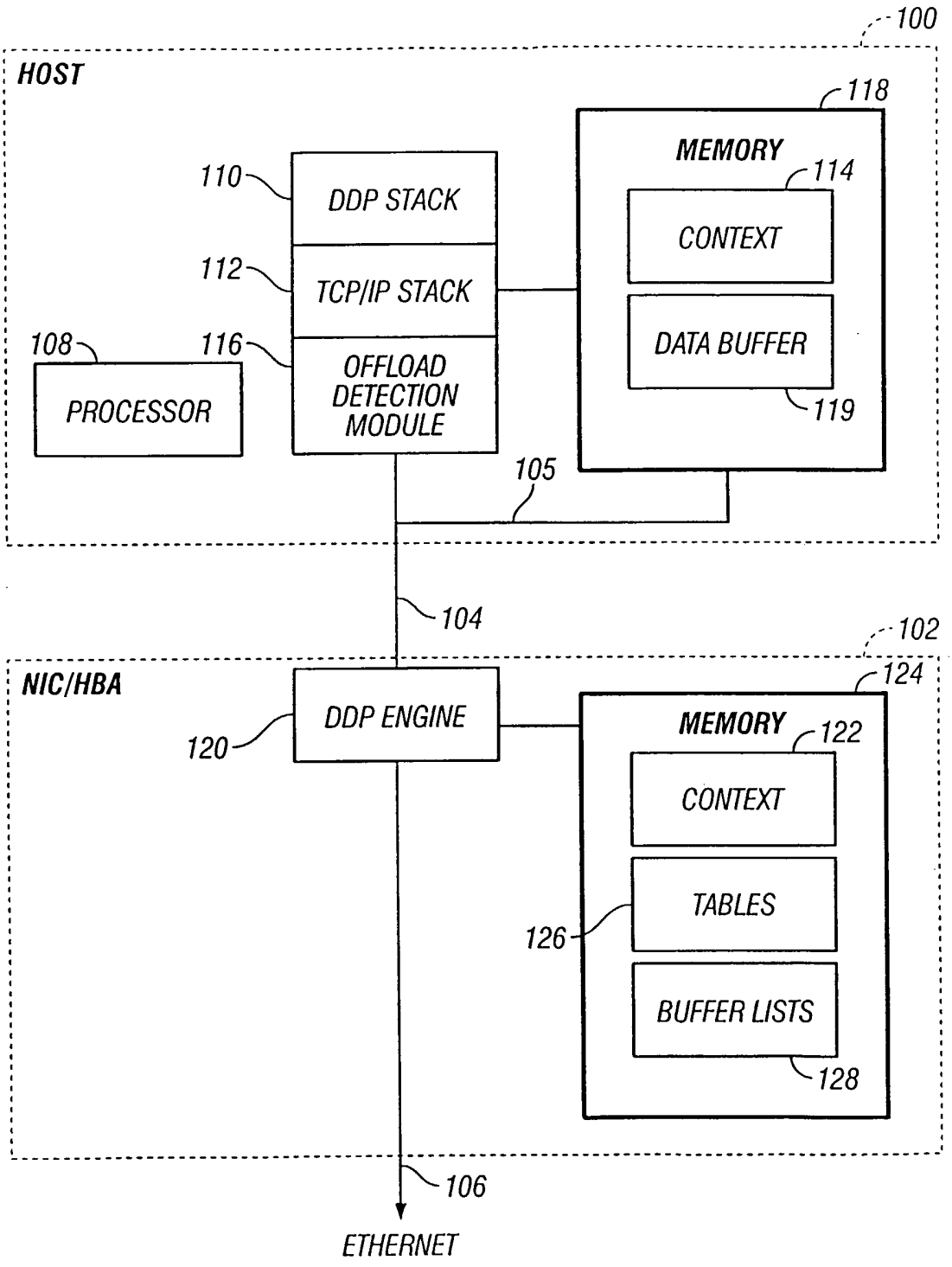
100
HOST
118
MEMORY
114
CONTEXT
DATA BUFFER
119
110
DDP STACK
112
TCP/IP STACK
108
PROCESSOR
116
OFFLOAD DETECTION MODULE
105
104
102
NIC/HBA
124
DDP ENGINE
120
MEMORY
122
CONTEXT
TABLES
126
BUFFER LISTS
128
106
ETHERNET

FIG. 1

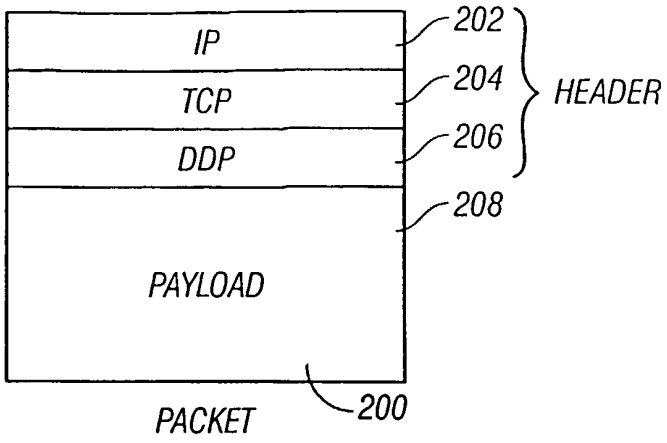
IP
TCP
DDP
PAYLOAD
PACKET
202
204
206
208
200
HEADER

FIG. 2

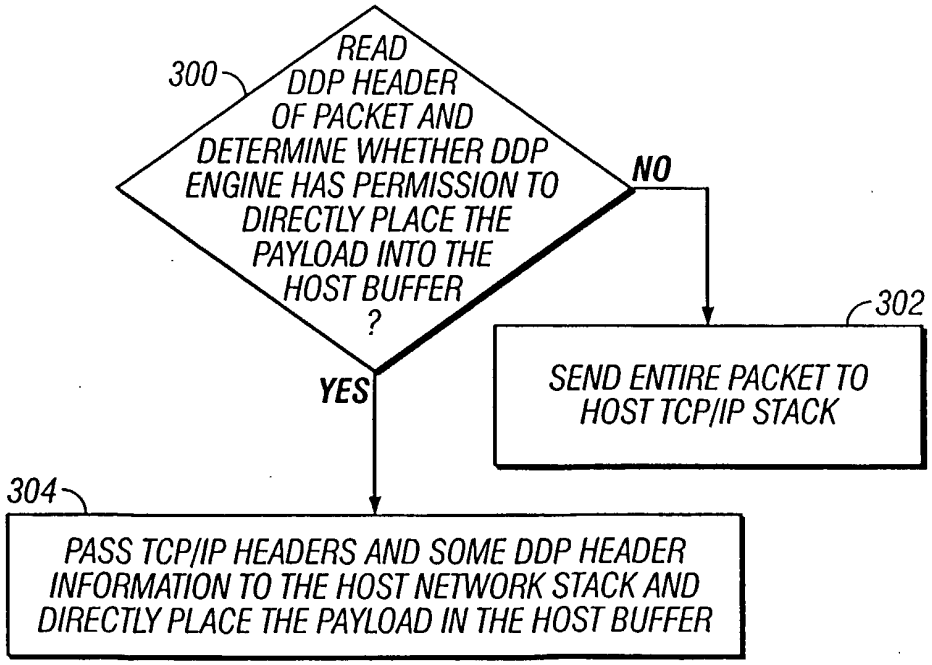
300
READ DDP HEADER OF PACKET AND DETERMINE WHETHER DDP ENGINE HAS PERMISSION TO DIRECTLY PLACE THE PAYLOAD INTO THE HOST BUFFER ?
NO
YES
302
SEND ENTIRE PACKET TO HOST TCP/IP STACK
304
PASS TCP/IP HEADERS AND SOME DDP HEADER INFORMATION TO THE HOST NETWORK STACK AND DIRECTLY PLACE THE PAYLOAD IN THE HOST BUFFER

FIG. 3

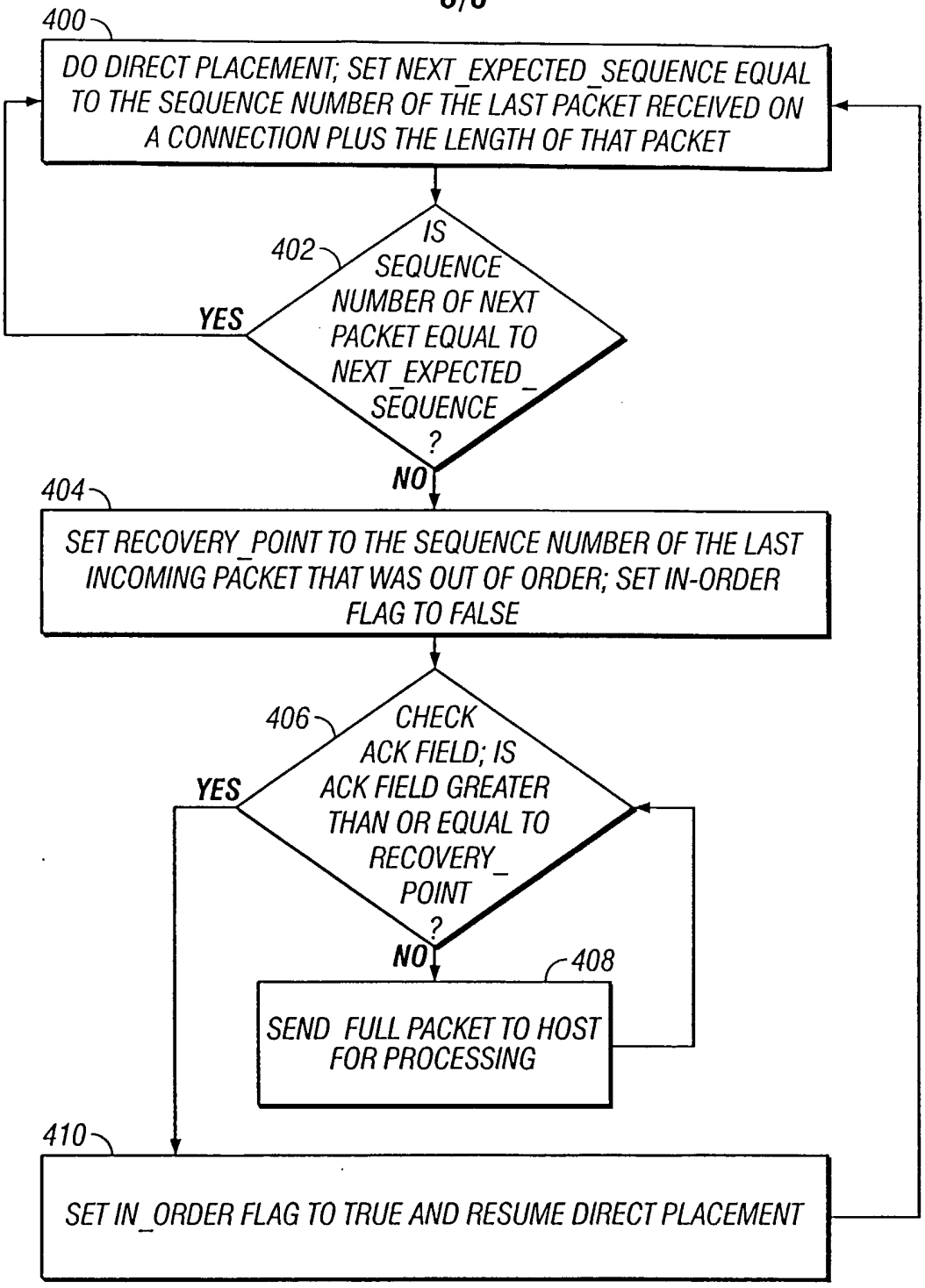
400
DO DIRECT PLACEMENT; SET NEXT_EXPECTED_SEQUENCE EQUAL TO THE SEQUENCE NUMBER OF THE LAST PACKET RECEIVED ON A CONNECTION PLUS THE LENGTH OF THAT PACKET
402
IS SEQUENCE NUMBER OF NEXT PACKET EQUAL TO NEXT_EXPECTED_SEQUENCE ?
YES
NO
404
SET RECOVERY_POINT TO THE SEQUENCE NUMBER OF THE LAST INCOMING PACKET THAT WAS OUT OF ORDER; SET IN-ORDER FLAG TO FALSE
406
CHECK ACK FIELD; IS ACK FIELD GREATER THAN OR EQUAL TO RECOVERY_POINT ?
YES
NO
408
SEND FULL PACKET TO HOST FOR PROCESSING
410
SET IN_ORDER FLAG TO TRUE AND RESUME DIRECT PLACEMENT

FIG. 4