US 20020003795A1

(19) United States

(12) Patent Application Publication (10) Pub. No.: US 2002/0003795 A1

OSKOUY et al. (43) Pub. Date: Jan. 10, 2002

(54) IN-LINE PACKET PROCESSING

(76) Inventors: **RASOUL MIRZAZADEH OSKOUY**, FREMONT, CA (US); **DENNIS C. FERGUSON**, MOUNTAIN VIEW, CA (US); **HANN-HWAN JU**, SAN JOSE, CA (US); **RAYMOND MARCELINO MANESE LIM**, MOUNTAIN VIEW, CA (US); **PRADEEP S. SINDHU**, MOUNTAIN VIEW, CA (US); **SREERAM VEERAGANDHAM**, SUNNYVALE, CA (US); **JEFF ZIMMER**, SCOTTS VALLEY, CA (US); **MICHAEL M.Y. HUI**, BEDFORD, MA (US)

Correspondence Address:
JUNIPER NETWORKS, INC.
1194 N. MATHILDA AVENUE
SUNNYVALE, CA 94089 (US)

(*) Notice: This is a publication of a continued prosecution application (CPA) filed under 37 CFR 1.53(d).

(21) Appl. No.: **09/129,137**

(22) Filed: **Aug. 4, 1998**

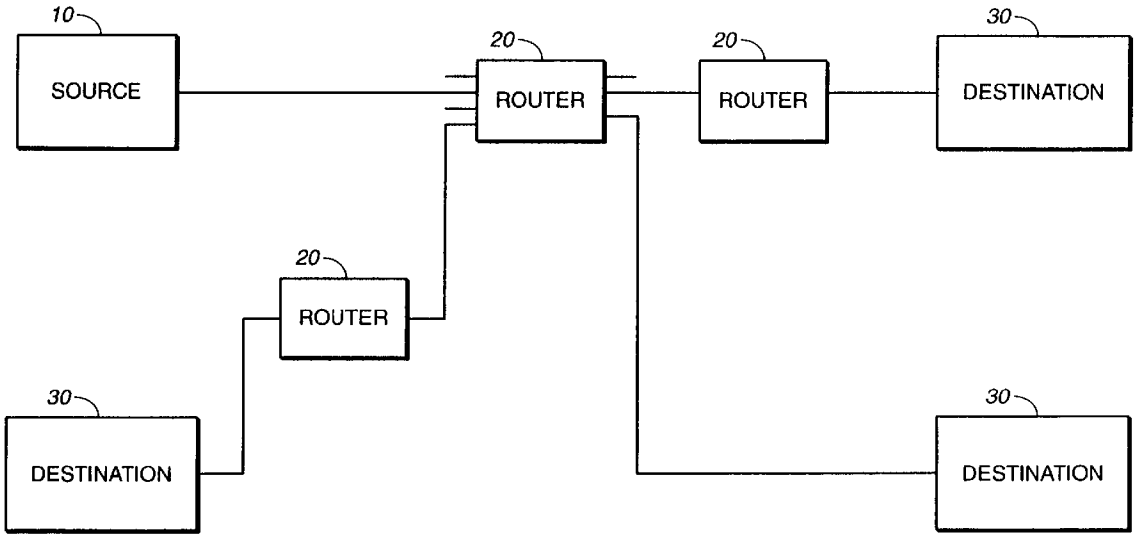
Publication Classification

(51) **Int. Cl.**[7] **H04L 12/28**

(52) **U.S. Cl.** **370/389**; 370/386

(57) **ABSTRACT**

A method and apparatus for in-line processing a data packet while routing the packet through a router in a system transmitting data packets between a source and a destination over a network including the router. The method includes receiving the data packet and pre-processing layer header data for the data packet as the data packet is received and prior to transferring any portion of the data packet to packet memory. The data packet is thereafter stored in the packet memory. A routing through the router is determined including a next hop index describing the next connection in the network. The data packet is retrieved from the packet memory and a new layer header for the data packet is constructed from the next hop index while the data packet is being retrieved from memory. The new layer header is coupled to the data packet prior to transfer from the router.



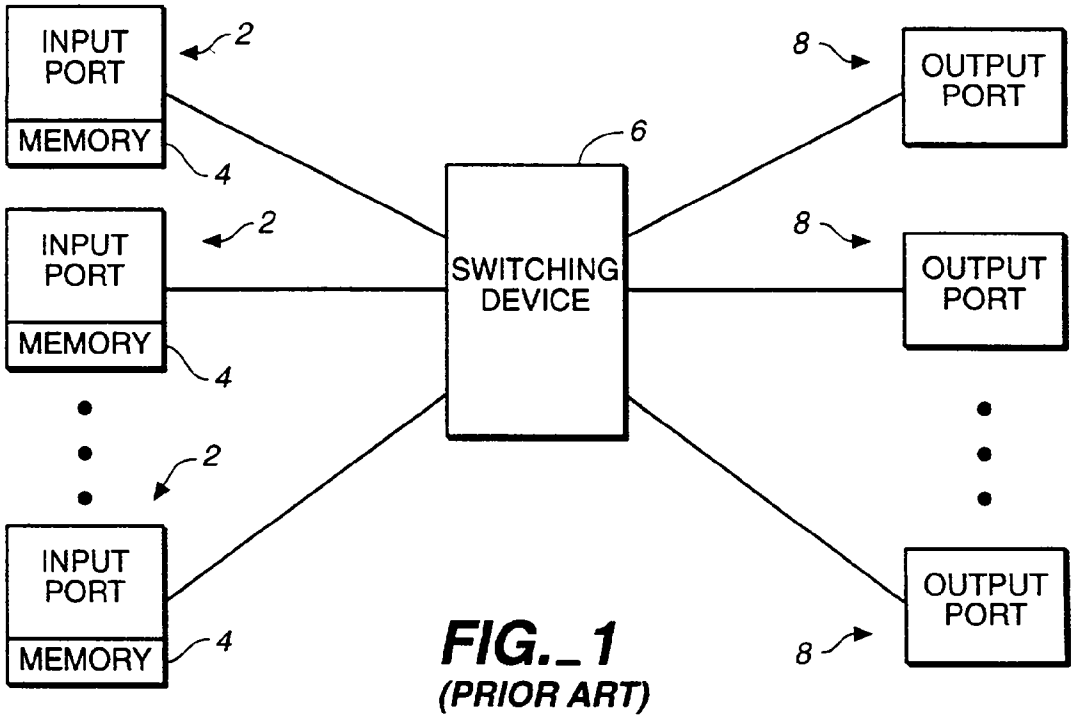


FIG._1
(PRIOR ART)

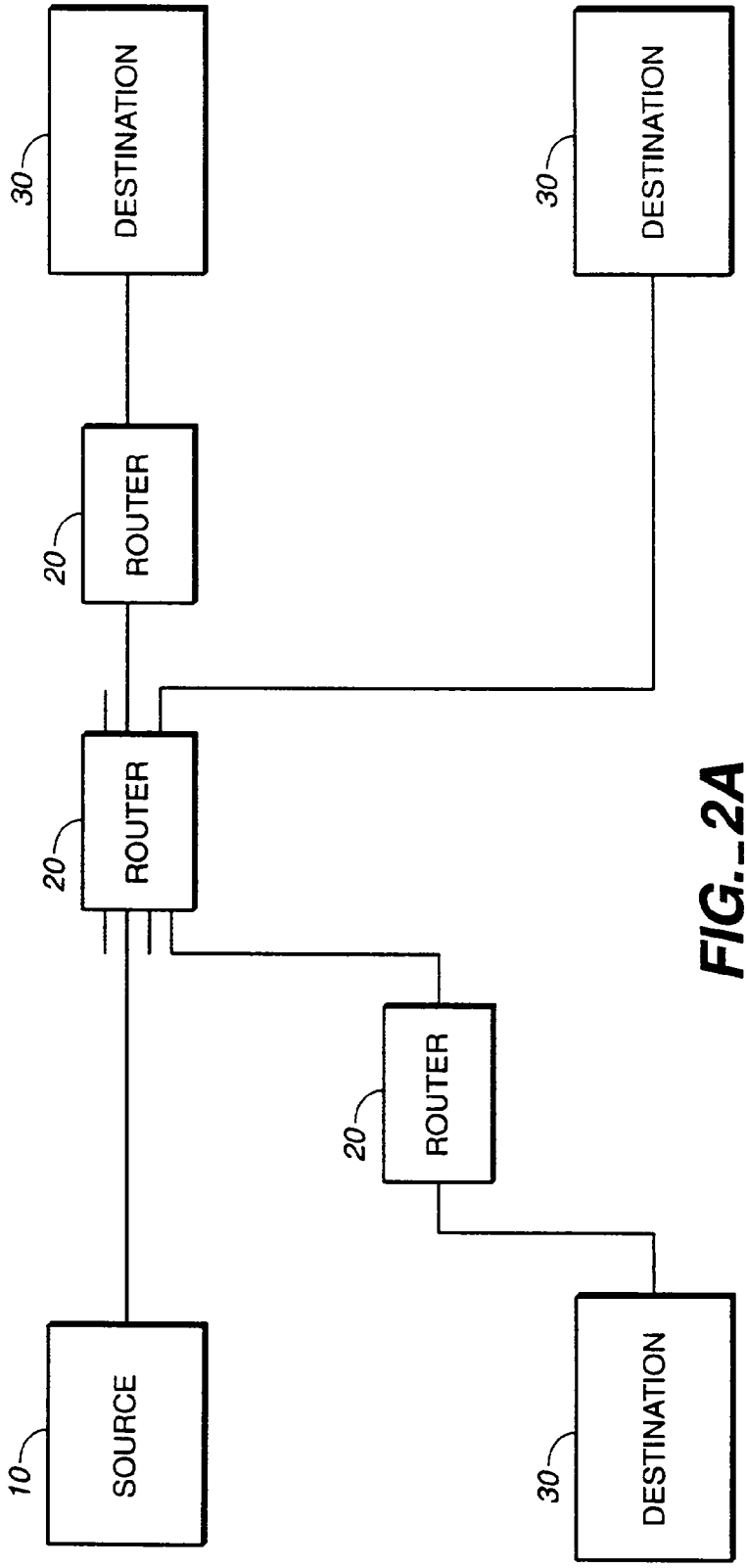

SOURCE
10
ROUTER
20
ROUTER
20
ROUTER
20
DESTINATION
30
DESTINATION
30
DESTINATION
30

FIG._2A

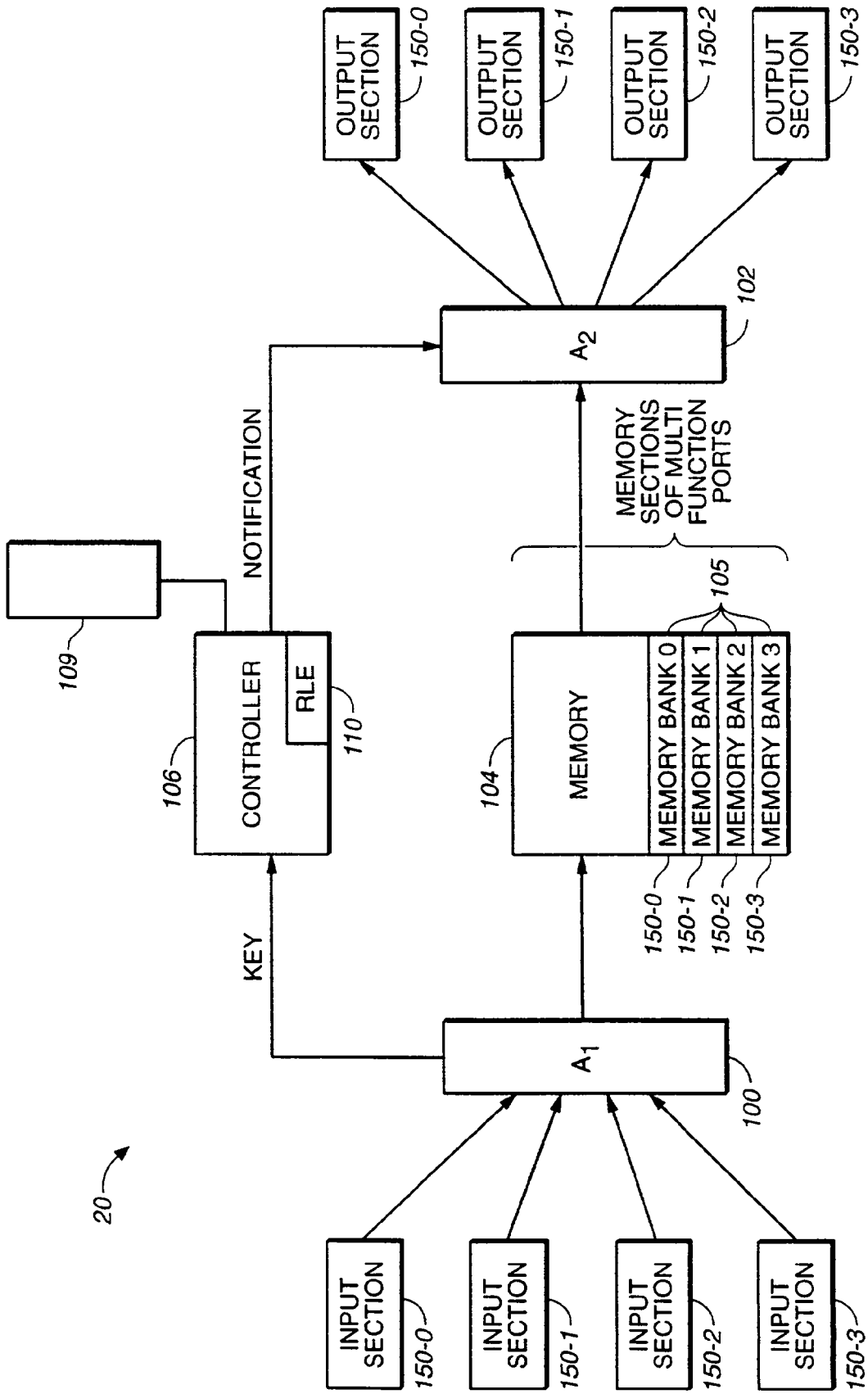

20
INPUT SECTION
150-0
INPUT SECTION
150-1
INPUT SECTION
150-2
INPUT SECTION
150-3
A1
100
KEY
CONTROLLER
106
RLE
110
109
NOTIFICATION
MEMORY
104
MEMORY BANK 0
MEMORY BANK 1
MEMORY BANK 2
MEMORY BANK 3
105
150-0
150-1
150-2
150-3
MEMORY SECTIONS OF MULTI FUNCTION PORTS
A2
102
OUTPUT SECTION
150-0
OUTPUT SECTION
150-1
OUTPUT SECTION
150-2
OUTPUT SECTION
150-3

FIG._2B

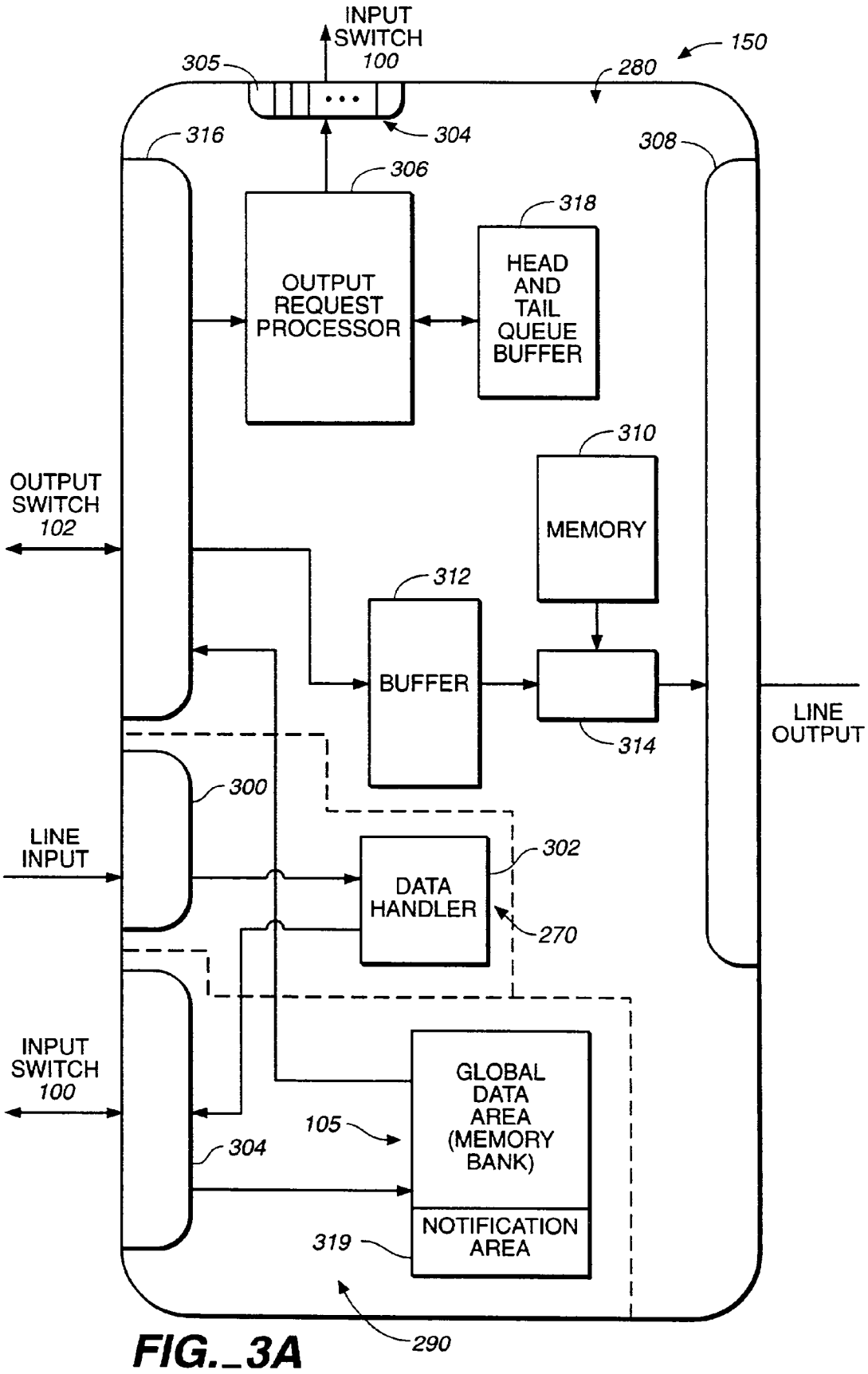

INPUT SWITCH 100
305
280
150
316
304
308
306
318
OUTPUT REQUEST PROCESSOR
HEAD AND TAIL QUEUE BUFFER
310
OUTPUT SWITCH 102
MEMORY
312
BUFFER
314
LINE OUTPUT
300
LINE INPUT
302
DATA HANDLER
270
INPUT SWITCH 100
GLOBAL DATA AREA (MEMORY BANK)
105
304
NOTIFICATION AREA
319
290

FIG._3A

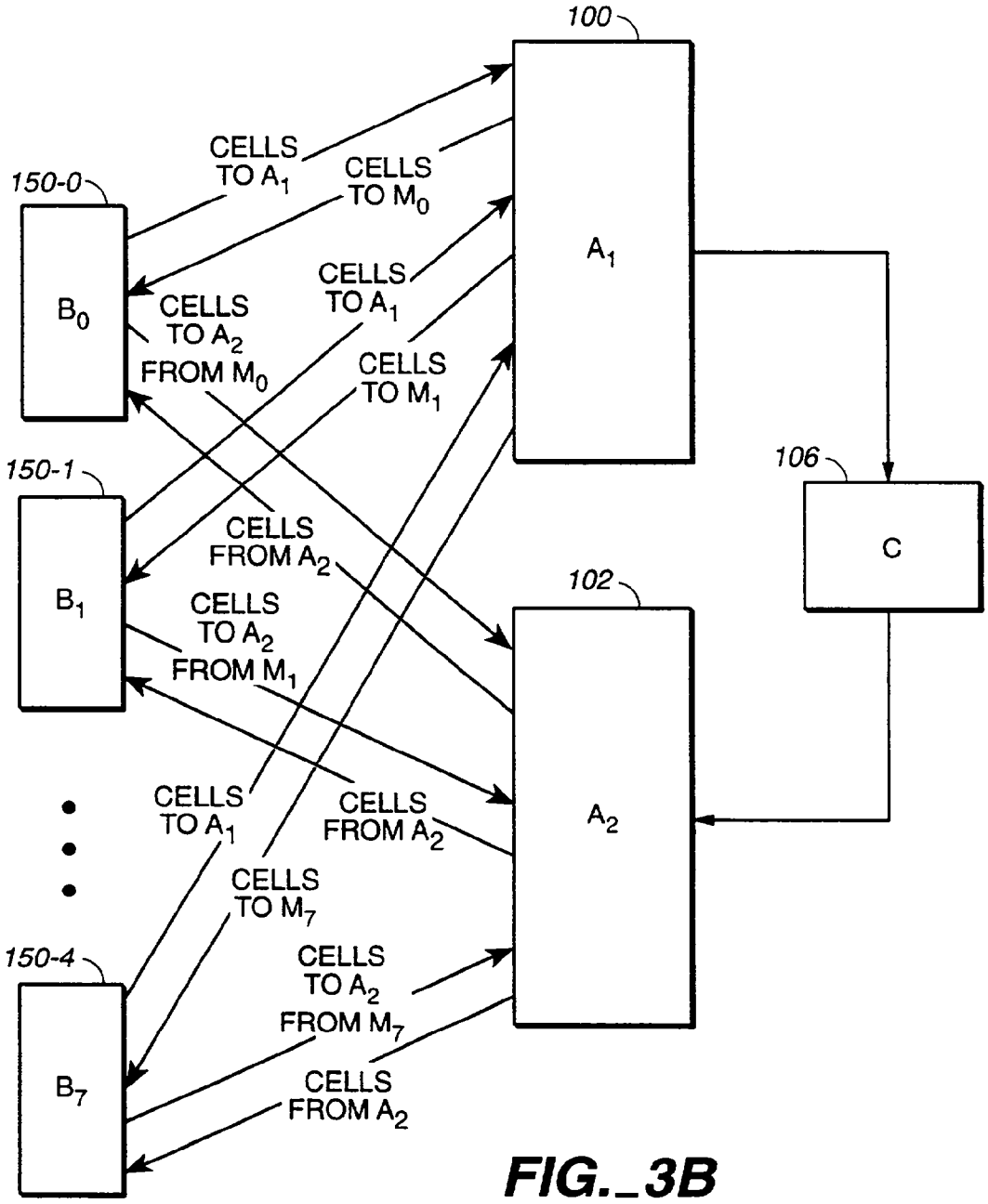

100
150-0
CELLS TO A_1
CELLS TO M_0
A_1
B_0
CELLS TO A_2 FROM M_0
CELLS TO A_1
CELLS TO M_1
106
150-1
C
CELLS FROM A_2
102
B_1
CELLS TO A_2 FROM M_1
CELLS TO A_1
CELLS FROM A_2
A_2
CELLS TO M_7
150-4
CELLS TO A_2 FROM M_7
B_7
CELLS FROM A_2

FIG._3B

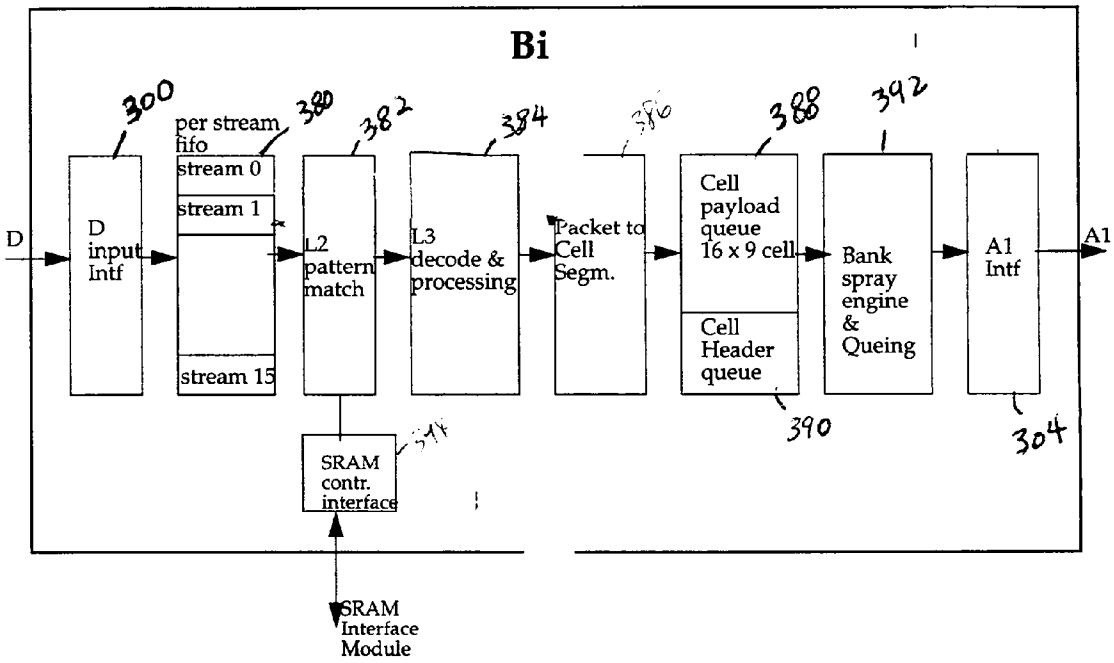

Bi
300
per stream fifo
380
382
384
386
388
392
D
D input Intf
stream 0
stream 1
stream 15
L2 pattern match
L3 decode & processing
Packet to Cell Segm.
Cell payload queue 16 x 9 cell
Cell Header queue
Bank spray engine & Queing
A1 Intf
A1
390
304
SRAM contr. interface
SRAM Interface Module

FIG. 3c

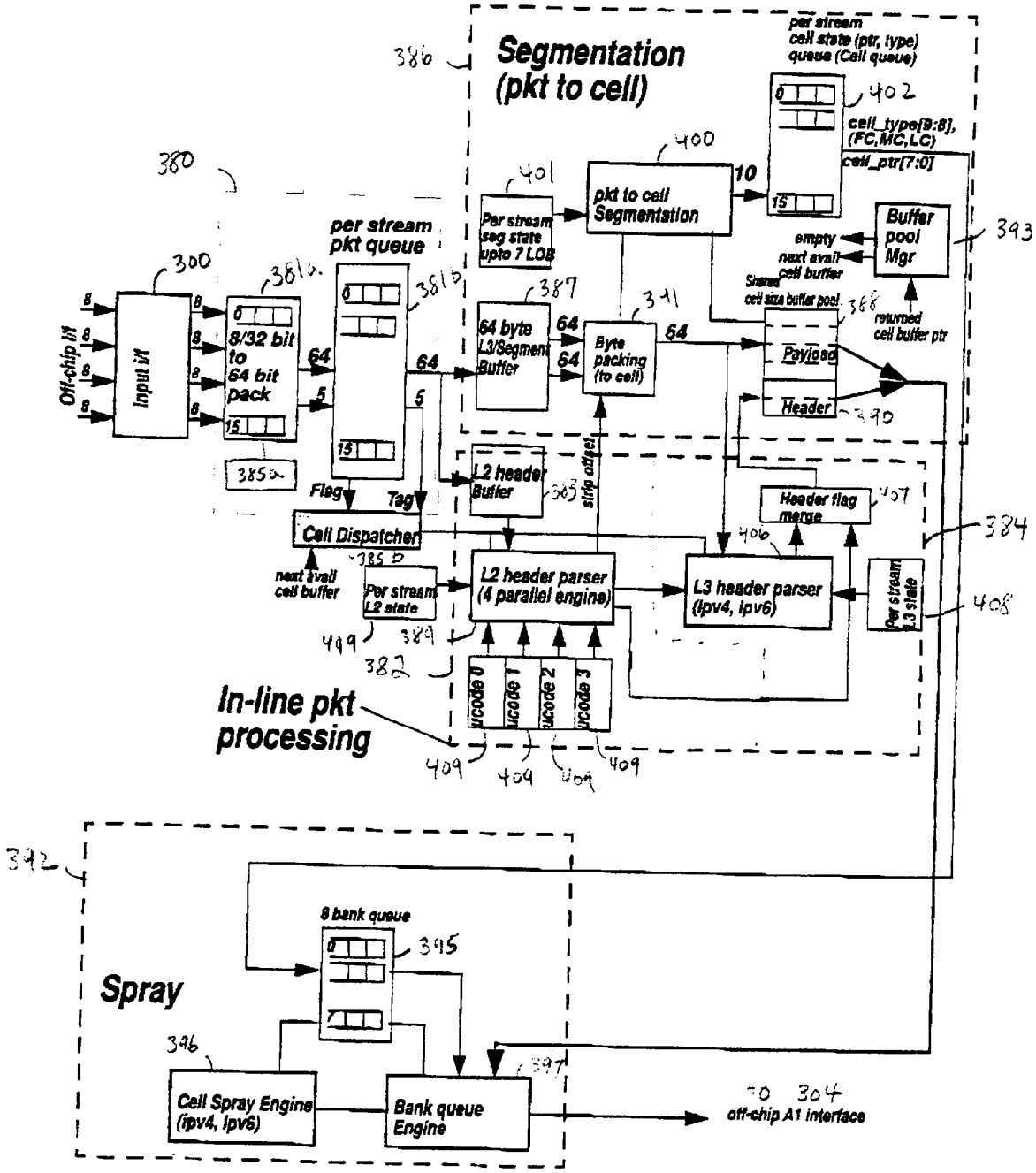

Segmentation (pkt to cell)
386
380
300
Off-chip I/f
Input I/f
8/32 bit to 64 bit pack
385a
per stream pkt queue
381a
381b
Flag
Tag
Cell Dispatcher
385 D
next avail cell buffer
Per stream L2 state
389
382
In-line pkt processing
per stream seg state upto 7 LOB
401
pkt to cell Segmentation
400
64 byte L3/Segment Buffer
387
Byte packing (to cell)
391
per stream cell state (ptr, type) queue (Cell queue)
402
cell_type[9:8], (FC,MC,LC)
cell_ptr[7:0]
Buffer pool Mgr
393
empty
next avail cell buffer
Shared cell size buffer pool
388
returned cell buffer ptr
Payload
Header
390
L2 header Buffer
strip offset
Header flag merge
407
384
L2 header parser (4 parallel engine)
L3 header parser (ipv4, ipv6)
Per stream L3 state
408
ucode 0
ucode 1
ucode 2
ucode 3
409
392
Spray
8 bank queue
395
396
Cell Spray Engine (ipv4, ipv6)
Bank queue Engine
397
TO 304
off-chip A1 interface

FIG. 3d

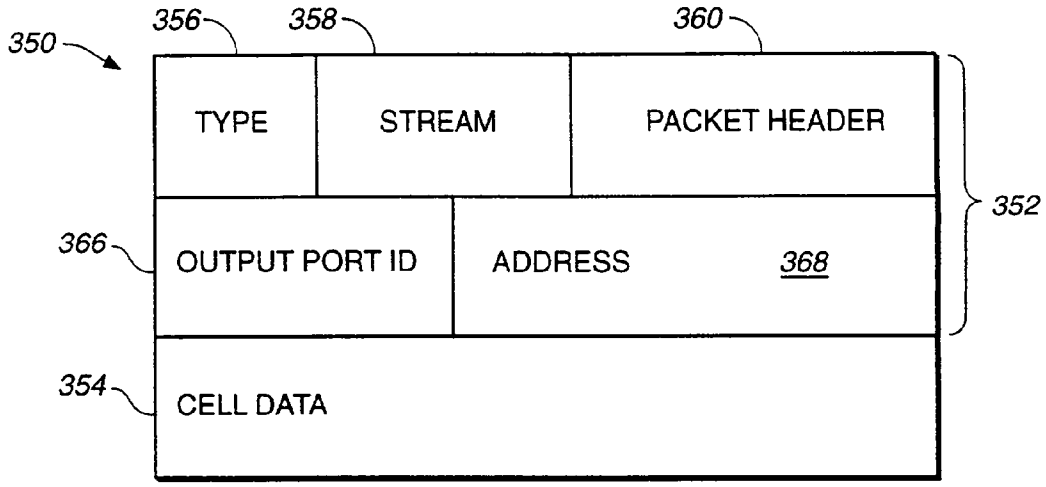

350
356
358
360
TYPE
STREAM
PACKET HEADER
352
366
OUTPUT PORT ID
ADDRESS
368
354
CELL DATA

FIG._3

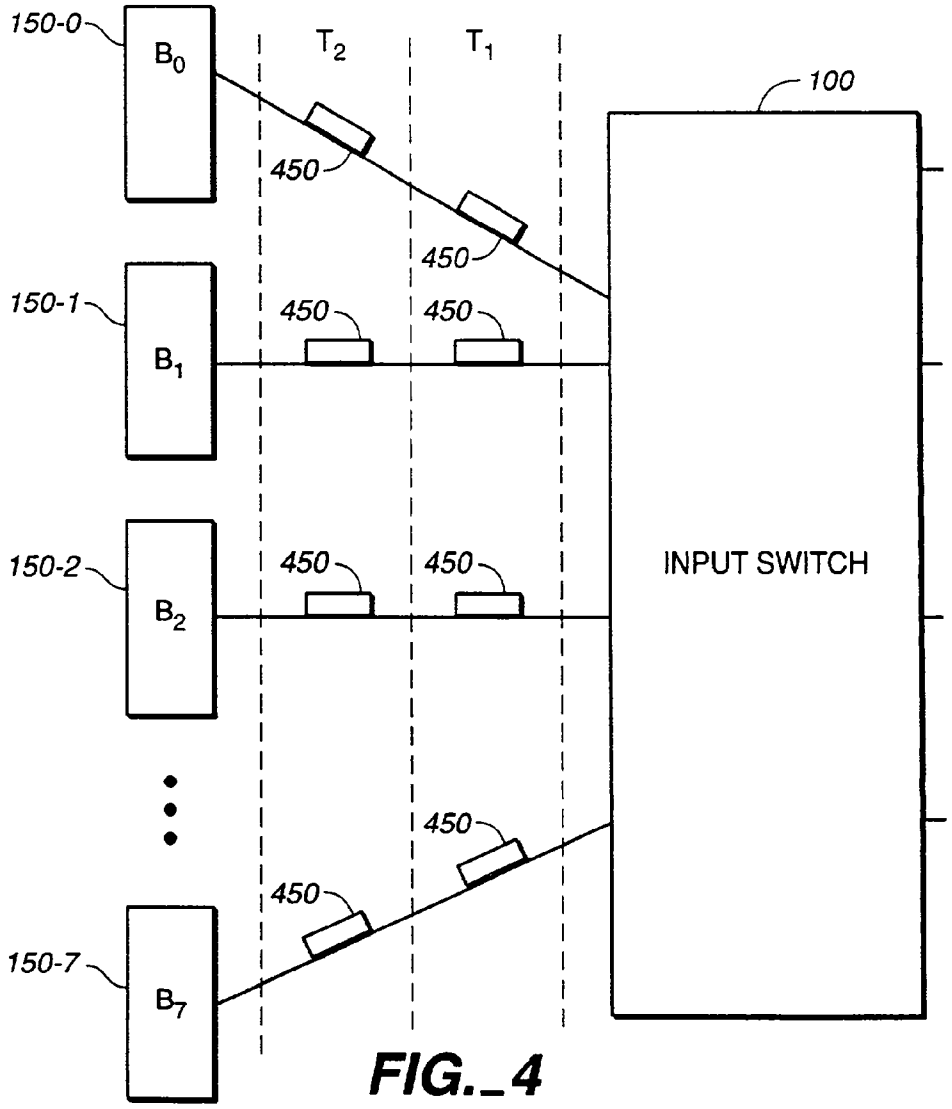

150-0
B0
T2
T1
100
450
450
150-1
B1
450
450
150-2
B2
450
450
INPUT SWITCH
450
450
150-7
B7

FIG._4

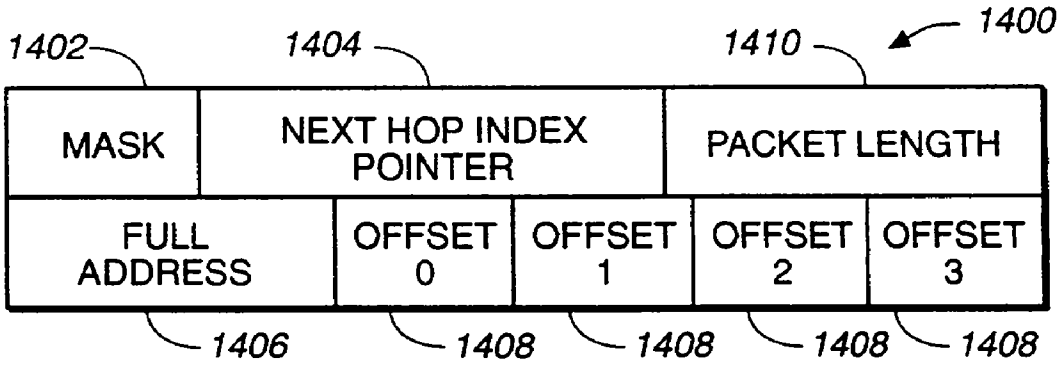

1400
1402
1404
1410
MASK
NEXT HOP INDEX POINTER
PACKET LENGTH
FULL ADDRESS
OFFSET 0
OFFSET 1
OFFSET 2
OFFSET 3
1406
1408
1408
1408
1408

FIG._ 5

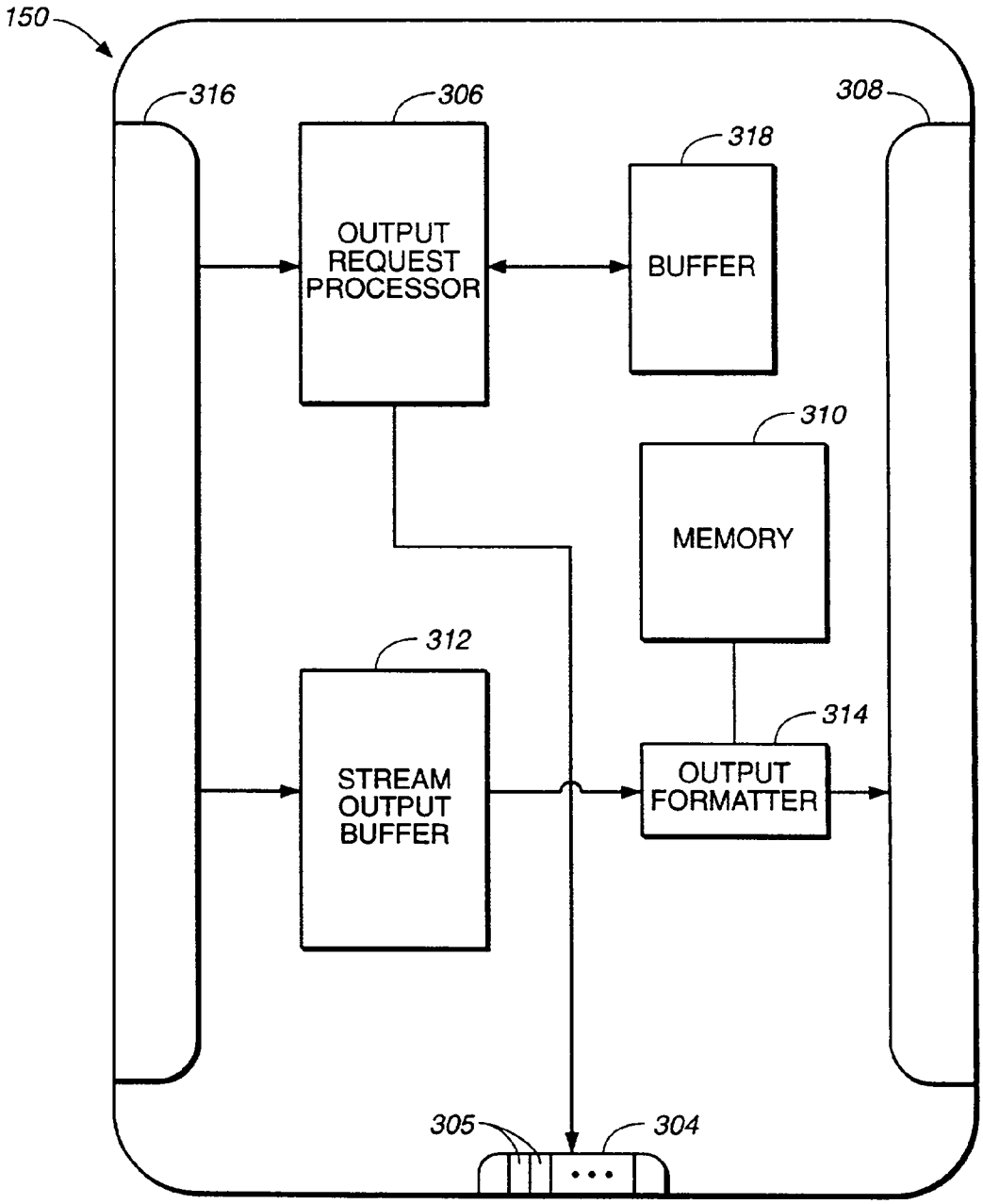

150
316
306
308
318
OUTPUT REQUEST PROCESSOR
BUFFER
310
MEMORY
312
314
STREAM OUTPUT BUFFER
OUTPUT FORMATTER
305
304

FIG._ 6

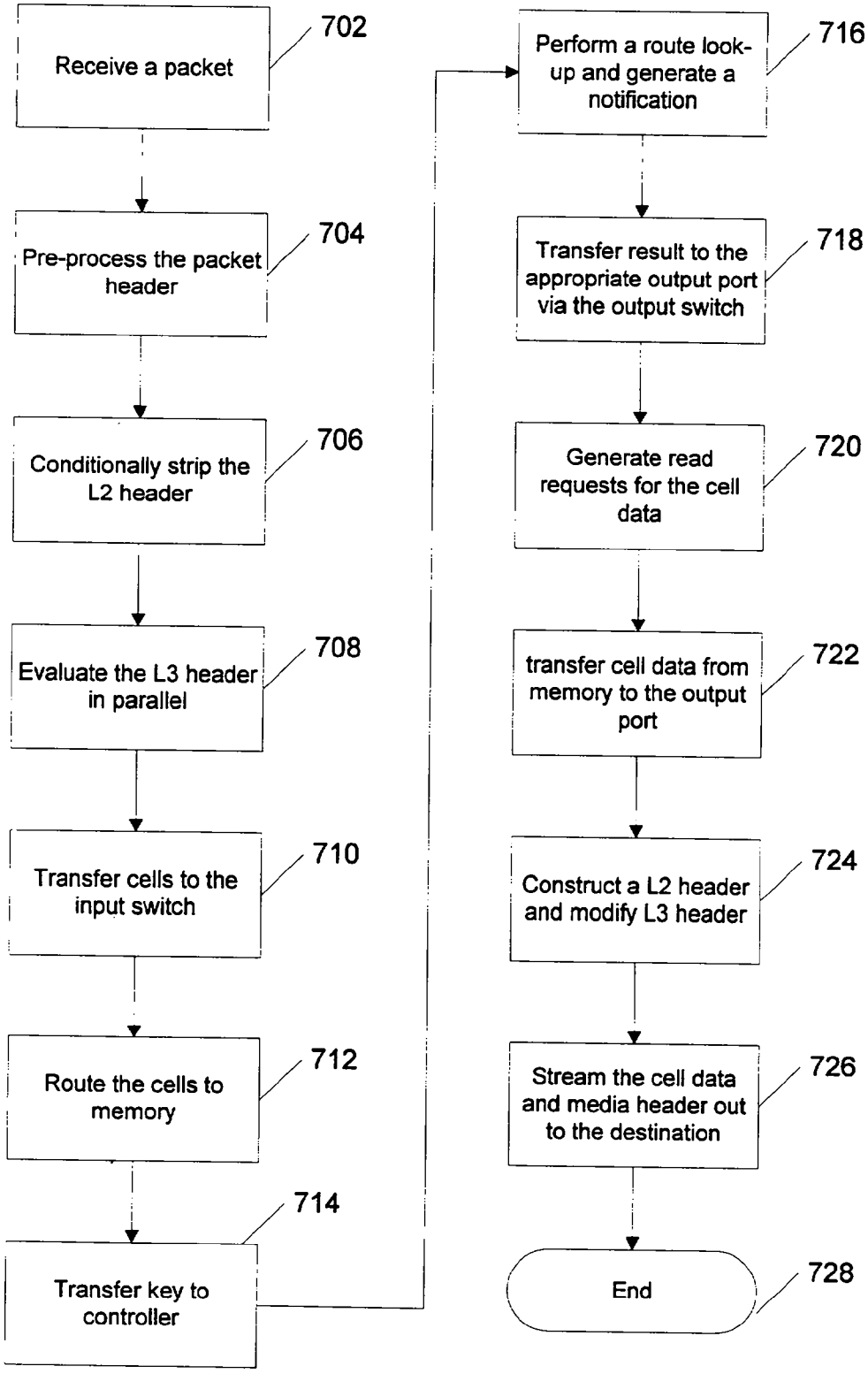

Receive a packet
702
Pre-process the packet header
704
Conditionally strip the L2 header
706
Evaluate the L3 header in parallel
708
Transfer cells to the input switch
710
Route the cells to memory
712
Transfer key to controller
714
Perform a route look-up and generate a notification
716
Transfer result to the appropriate output port via the output switch
718
Generate read requests for the cell data
720
transfer cell data from memory to the output port
722
Construct a L2 header and modify L3 header
724
Stream the cell data and media header out to the destination
726
End
728

Figure 7

IN-LINE PACKET PROCESSING

BACKGROUND

[0001] The present invention relates generally to data routing systems, and more particularly to methods and apparatus for efficiently routing packets through a network.

[0002] In packet switched communication systems, a router is a switching device that receives packets containing data or control information on one port, and based on destination information contained within the packet, routes the packet out another port to a destination (or an intermediary destination). Conventional routers perform this switching function by evaluating header information contained within a first data block in the packet. The header includes destination information that can be used in determining the proper output port for a particular packet.

[0003] Efficient switching of packets through the router is of paramount concern. Referring now to **FIG. 1, a** conventional router includes a plurality of input ports **2** each including an input buffer (memory) **4**, a switching device **6** and a plurality of output ports **8**. Data packets received at input port **2** are stored at least temporarily, in memory **4** while destination information associated with each packet is decoded to determine the appropriate switching through switching device **6**.

[0004] Data packets include both header and data fields and can be of variable lengths. The header includes both format information related to the type of packet and layer data. Networking protocols are defined in terms of layers, with each layer responsible for different portions of the communications in the network. For example TCP/IP is a standard internet network protocol that includes an application layer, transport layer, IP layer (network layer or L3), link layer (L2) and physical layer (L1). A data packet can include a plurality headers, one for each relevant layer that is to handle the packet during routing. Each layer header can include source and destination information as well as other layer specific information. To transfer the packet out of the switch, portions of the header are required to be examined (e.g., Layer 2 and Layer 3, L2 and L3 respectively, header information). Typically, packets are stored in memory **4** while header processing is performed.

[0005] Complicating the examination process, the format of the packet headers can vary. Destination or other information (e.g., L3 information) may not always be located at the same position in a header.

[0006] Unfortunately, conventional routers are inefficient in a number of respects. Conventional routers are incapable of processing packets in-line. All of the packet header typically must be received prior to the beginning of header processing. Packets are not screened early on for faults necessitating additional bandwidth and resources for handling these bad packets.

SUMMARY OF THE INVENTION

[0007] In general, in one aspect, the invention provides a method of in-line processing a data packet while routing the packet through a router in a system transmitting data packets between a source and a destination over a network including the router. The method includes receiving the data packet and pre-processing layer header data for the data packet as the data packet is received and prior to transferring any portion of the data packet to packet memory. The data packet is thereafter stored in the packet memory. A routing through the router is determined including a next hop index describing the next connection in the network. The data packet is retrieved from the packet memory and a new layer header for the data packet is constructed from the next hop index while the data packet is being retrieved from memory. The new layer header is coupled to the data packet prior to transfer from the router.

[0008] Aspects of the invention include numerous features. The pre-processing step includes screening header layer data associated with the data packet for errors and dropping a bad data packet prior to transferring any portion of the data packet to packet memory. The screening includes screening Layer 2 (L2) and Layer 3 (L3) headers for errors. The L2 header can be examined to detect errors arising from unrecognized L2 header formats and unconfigured L2 connections. The L3 header can be examined to detect data packets with checksum errors, packet length errors and L3 header errors.

[0009] The step of storing the data packet in memory includes dividing the data packet into cells of a fixed size and storing the cells in a distributed memory. The step of retrieving the data packet from memory includes reconstructing the packet from cells stored in the memory.

[0010] The pre-processing includes stripping L2 header data from a data packet prior to storage the packet memory, identifying the beginning of the L3 header and examining the L3 header for errors prior to the storage of the data packet in the packet memory. The next hop index is a pointer to a sequence stored in a storage device within the router and the step of constructing a new layer header includes executing the sequence. The execution of the sequence includes retrieving a common template for constructing a common portion of an L2 header to be attached to the data packet and a custom template for constructing a unique portion of the L2 header. The common and unique templates can be executable code operable to construct and associated portion of an L2 header for the data packet.

[0011] The step of receiving the data packet includes receiving a plurality of data packets for processing from a plurality of input ports representing a plurality of streams of data to be routed through the router. The step of pre-processing the data packet includes dividing the data packet into fixed length cells and parsing the L2 header associated with the first cell of the data packet prior to receipt of the entire data packet. The step of parsing the L2 header includes examining the L2 header for errors and identifying the start of a next layer header in the data packet.

[0012] The cells for a data packet can be temporarily stored in a queue after L2 header parsing is completed. Consecutive cells in a data packet can be sprayed to a different bank in the packet memory.

[0013] The method can include snooping while the cells are being written to the queue and parsing the L3 header including examining the L3 header for errors. A data packet can be dropped if errors are detected in the L2 header during L2 header parsing without storing a cell associated with the data packet in the queue.

[0014] In another aspect, the invention provides a router for in-line processing a data packet while routing the packet

in a system transmitting data packets between a source and a destination over a network including the router. The router includes a packet memory for storing portions of the data packet, an input port for receiving a data packet including a header processing engine for evaluating header layer information upon receipt and prior to the storage of the data packet in the packet memory, a controller operable to determine packet routing through the router and output a next hop index indicating the next connection in the network for each data packet to be routed through the router and an output port for transferring the data packet to the destination. The output port includes a output formatter for constructing a layer header for the data packet to facilitate the transfer of the packet to the destination.

[0015] Aspects of the invention include numerous features. The header processing engine is operable to screen header layer data associated with the data packet for errors and drop a bad data packet prior to transferring any portion of the data packet to packet memory. The header processing engine screens Layer 2 (L2) and Layer 3 (L3) headers for errors. The header processing engine examines the L2 header to detect errors arising from unrecognized L2 header formats and unconfigured L2 connections. The header processing engine examine the L3 header to detect data packets with checksum errors, packet length errors and L3 header errors.

[0016] The router can include a cell packetizer operable to divide the data packet into cells of a fixed size prior to transfer to packet memory. The packet memory can be a distributed memory and the router can include a spray engine for distributing cells across the distributed memory.

[0017] The header processing engine strips L2 header data from a data packet prior to storage the packet memory, identifies the beginning of an L3 header and examines the L3 header for errors prior to the storage of the data packet in the packet memory. The next hop index is a pointer to a sequence stored in a storage device within the router and the output formatter executes the sequence to construct a new layer header. The storage device includes a common template for constructing a common portion of an L2 header to be attached to the data packet and a custom template for constructing a unique portion of the L2 header. The storage device is distributed and includes a local portion within the output port and a remote portion elsewhere in the routing device and where the common template is stored in local portion and the custom template is stored in the remote portion. The common and unique templates are executable code operable to construct and associated portion of an L2 header for the data packet. The router can further include a plurality of input ports and a plurality of output ports, each of the input ports and output ports configurable to receive and process a plurality of data packets representing a plurality of streams of data to be routed through the router.

[0018] The router can further include a cell packetizer operable to divide the data packet into fixed length cells and a L2 parsing engine operable to examine the L2 header associated with the first cell of the data packet prior to receipt of the entire data packet. The L2 parsing engine is operable to examine the L2 header for errors and identify the start of a next layer header in the data packet.

[0019] The router can further include a queue operable for temporarily storing the cells for a data packet after L2 header parsing is completed and a spray engine operable to spray consecutive cells in a data packet to a different bank in the packet memory.

[0020] The router can further include a L3 parsing engine operable to snoop while the cells are being written to the queue and parse the L3 header including examining the L3 header for errors. The L2 parser engine is operable to drop a data packet if errors are detected in the L2 header during L2 header parsing without storing a cell associated with the data packet in the queue.

[0021] In another aspect the invention provides a method of routing a data packet through a router in a system transmitting data packets between a source and a destination over a network including the router. The method includes receiving the data packet and dividing the data packet into cells of a fixed size. While dividing the data packet, layer header data for the packet is pre-processed to locate the beginning of a next layer header. The cells are stored in a distributed memory. Prior to storing a first cell in the distributed memory associated with the packet, header layer data is screened for errors and a bad packet is dropped prior to transferring any cells into the distributed memory. A routing through the router is determined including a next hop index describing the next connection in the network. The packet is reconstructed from cells stored in the memory and new layer header data for the packet is constructed from the next hop index.

[0022] Among the advantages of the invention are one or more of the following. Packets are processed in-line and input buffering is minimized by the use of shared buffers. Packet pre-processing is accelerated by using multiple embedded micro-code engines to perform L2 header processing for received packets. Pre-processing includes segmentation of the packets into cells and distribution of the cells across memory within the router while processing L3 header data in parallel. Packet header screening is accomplished early on prior to the receipt of all of the data associated with the header of a given packet. Early packet screening reduces bandwidth and frees resources in the router.

[0023] Other advantages and features will be apparent from the following description and claims.

BRIEF DESCRIPTION OF THE DRAWINGS

[0024] **FIG. 1** is a block diagram of a conventional router device.

[0025] **FIG. 2***a* is a schematic block diagram of a data routing system according to one embodiment of the present invention.

[0026] **FIG. 2***b* is a schematic block diagram of a router according to one embodiment of the present invention.

[0027] **FIG. 3***a* is a schematic block diagram of an multi-function port according to one embodiment of the present invention.

[0028] **FIG. 3***b* is a schematic block diagram showing data transfers between components of the router of **FIG. 2***b* according to one embodiment of the present invention.

[0029] **FIG. 3***c* is a schematic block diagram of the input section of a multi-function multiport according to one embodiment of the invention.

[0030] **FIG. 3***d* is a schematic block diagram of the data handler of a multi-function multiport according to one embodiment of the invention.

[0031] **FIG. 3***e* is a data structure for a cell transferred between a multi-function port and an input switch according to one embodiment of the present invention.

[0032] **FIG. 4** is a schematic block diagram of a router showing the timing and ordering of transfers from a multi-function multiport to the input switch according to one embodiment of the present invention.

[0033] **FIG. 5** is a data structure for an output request transferred from the controller to the output switch according to one embodiment of the present invention.

[0034] **FIG. 6** is a schematic block diagram for an output section of a multi-function port according to one embodiment of the present invention.

[0035] **FIG. 7** is a flow diagram for a process of routing a packet through a router according to one embodiment of the present invention.

DETAILED DESCRIPTION

[0036] Referring to **FIG. 2***a*, in a packet switching system, a source **10** is connected to one or more routers **20** for transmitting packets to one or more destinations **30**. Each router includes a plurality of multi-function multiports that are connected to various sources and destinations. A packet from source **10** can pass through more than one router **20** prior to arriving at its destination.

[0037] Referring to **FIG. 2***b*, each router **20** includes an input switch **100**, an output switch **102**, a global data buffer **104** including one or more memory banks **105**, a controller **106** and a plurality of multi-function multiports **150** (**150-0** through **150-3**, respectively). Associated with the controller **106** is controller memory **109** for storing a routing table. Input switch **100** and output switch **102** are connected to each multi-function multiport **150** in router **20**. In one embodiment, router **20** includes plug-and-play multi-function multiports which allows for easy expansion capability. The present invention will be described with reference to a system including eight multi-function multiports **150** (even though **FIG. 2***b* only shows four), with each multi-function multiport including up to sixteen input ports and sixteen output ports. Other configurations can be used depending on user load conditions. Each multi-function multiport includes one or more input ports, one or more output ports and a memory. The configuration and operation of the multi-function multiports will be described in greater detail below.

[0038] In operation, packets are received at a multi-function multiport **150**, transferred to input switch **100** and stored temporarily in global data buffer **104**. When the packet is received by input switch **100**, a key is read from the packet and transferred to controller **106**. The key contains destination information which is derived from a header field associated with the first block of data in a packet and other information (such as source ID, priority data and flow ID).

[0039] A route look-up engine **110** in controller **106** performs a tree based search based on the key information and returns a result that includes the output multiport associated with the destination. The result is coupled with other information (such as source ID, flow ID and packet length) for routing the packet through router **20** and provided as a notification from controller **106** to output switch **102**. Output switch **102** transfers the notification to the identified multi-function multiport **150**. Upon receiving the notification information, the multi-function multiport **150** initiates the transfer of the packet from global data buffer **104** through output switch **102** to the appropriate multi-function multiport **150**.

[0040] Multi-function Multiports

[0041] Referring to **FIG. 3***a*, each multi-function multiport **150** includes an input section **270**, an output section **280** and a memory section **290**.

[0042] Input section **270** includes a line input interface **300**, a data handler **302** and an input switch interface **304**.

[0043] Output section **280** includes an output request processor **306**, a line output interface **308**, a storage device **310**, stream output buffers **312** (one for each output stream), output formatter **314**, an output switch interface **316** and head and tail queue buffer **318**. In addition, the output section includes a portion of input switch interface **304**. Specifically, input switch interface **304** includes read request queues **305**, one for each memory bank. The use and operation of the read request queues, stream output buffers, and head and tail queue will be discussed in greater detail below in association with **FIG. 6**.

[0044] Memory section **290** includes a memory bank **105** (which represents a portion of the global data buffer **104**) and a notification queue body **319**.

[0045] The multi-function multiport is used in conjunction with the input switch, output switch and controller as is shown in **FIG. 3***b*. The various piece components of the input section, output section and memory section will be described in greater detail below. The combination of the input, output and memory sections into a single unit simplifies the interfaces between the components.

[0046] Referring again to **FIG. 3***a*, packets are received at line input interface **300**. As the packets are received, data handler **302** divides the packets received into fixed lengths cells. In one embodiment, the length of each cell is 80 bytes, with 16 bytes of internal header (control information) and 64 bytes of cell data. As the data handler divides the incoming packets into fixed length cells, it synchronously outputs the cells to input switch **100** through input switch interface **304**.

[0047] As described above, packets are divided into data cells and transferred into the global data buffer **104** for temporary storage during the look-up process. Data handler **302** of input section **270** divides the packets into data cells and examines data format information (L2/L3 information) from each packet prior to storage of the packet in the global data buffer portion of memory **104** (**FIG. 2***b*). In one implementation shown in **FIG. 3***c*, data handler **302** includes a packing queue **380**, L2 pattern match decoder **382**, L3 decoder **384**, cell segmentation engine **386**, cell payload queue **388**, cell header queue **390** and bank spray engine **392**.

[0048] Each multi-function multiport includes a line input interface **300** that can include up to 16 input ports for receiving data from up to 16 active streams. Each multi-function multiport is configurable to accommodate streams of various formats and in one implementation supports a 2.4

Gbps (OC-48) full duplex interface. Other input configurations are available with an accumulated bandwidth of up to 2.4 Gbps, including up to 16 streams of OC-3 bandwidth. Other configurations include **12** streams of OC-3 bandwidth and one stream of OC-12 bandwidth. Packing queue **380** receives the various input stream data on the input ports and packs the stream data into data words for transfer to segmentation buffer **387** (up to 64 bytes in one implementation). Packing queue **380** includes per stream queues for accumulating data from each stream.

[**0049**] Referring to **FIG. 3***d*, the packing process is performed in two phases. In a first phase, 8 bit or 32 bit data is accumulated in a bit packing queue **381***a*. Bit packing queue **381***a* includes stream bit packing queues (**381***a*-**1** thru **381***a*-**16**), one for each stream. In one implementation, bit packing queue **381***a* includes 16 stream bit packing queues (when supporting 16 OC-3 streams). Bit packing queue **381***a* includes a controller **385***a* for streaming data words from the bit packing queue to a byte packing queue **381***b*. Each stream bit packing queue can be sized to hold two or more words of data, and in one implementation are sized to hold three 64 bit words of data (8 byte data words). Associated with each data word stored in the stream bit packing queues are control flags which are received along with the stream data. In one embodiment, five (5) bits of flag data are stored with each 64 bit data word in each stream bit packing queue. The five flag data bits are passed with each 64 bit word from the bit packing queue **381***a* to the byte packing queue **381***b*.

[**0050**] In one embodiment the control flags include an end of packet without error flag (1 bit), a end of packet with error flag (1 bit), and a last byte pointer (3 bits). The size of the last byte pointer indicates the last byte in the eight (8) byte word of data that contains data in the transfers between the bit packing queue **381***a* and byte packing queue **381***b*. In one implementation, line input interface **300** screens the incoming packets and generates the control flag data. The control flag data can be advantageously used in screening packets prior to transfer to the global memory. More specifically, errors detected by the line input interface **300** are reconciled during L2 header processing. A packet that is written to segmentation buffer **387** that has an error flag set is never transferred by the packetizer **391** to payload buffer **388**. Data associated with the packet is overwritten in time by a next 64 byte data word from byte packing queue **381***b*. Other early error detection methods are described in greater detail below in association with L2 and L3 processing.

[**0051**] Based on the availability of data in the individual stream bit packing queues, a single 64 bit word is transferred per clock cycle by controller **385***a* from bit packing queue **381***a* to byte packing queue **381***b*. Controller **385***a* cycles through the individual bit packing queues in a round-robin fashion to transfer available data words to byte packing queue **381***b*.

[**0052**] In a second phase of the packing process, byte packing queue **381***b* accumulates eight (8) byte portions of data (64 bit data words) prior to segmentation. Byte packing queue **381***b* includes stream byte packing queues (**381***b*-**0** thru **381***b*-**15**), one for each stream. In one implementation, byte packing queue **381***b* includes 16 stream byte packing queues to support 16 streams of OC-3 data. Depending on the format of the stream data received, a lesser number of the stream byte packing queues can be used. Byte packing queue **381***b* is sized to support the input bandwidth, which in one implementation is 2.4 Gbps. Byte packing queue **381***b* is configurable to support a variety of input stream configurations. Byte packing queue **381***b* can be a flexible buffer whose resources are dynamically allocated at start-up depending on the input configuration.

[**0053**] Byte packing queue **381***b* includes a cell dispatcher **385***b* for streaming data words from byte packing queue **381***b* to segmentation buffer **387**. Each stream byte packing queue can be sized to hold N or more words of data, and in one implementation each is sized to hold eighteen (18) eight (8) byte data portions (64 bit data words). Associated with each data word stored in the stream byte packing queues are control flags which are received along with the stream data.

[**0054**] Depending on the input configuration of the stream data received, a lesser number of divisions for byte packing queue **381***b* may be required. For example, when supporting four OC-12 streams, byte packing queue **381***b* is configured with four byte packing queues to accumulate 64 bit data words for transfer to segmentation buffer **387**.

[**0055**] Cell dispatcher **385***b* operates in a round-robin fashion to cycle through the various individual stream byte packing queues to extract data words for transfer to segmentation buffer **387**. In one embodiment, eight (8) byte read cycles are used to transfer data from the individual byte packing queues to segmentation buffer **387**.

[**0056**] Each byte packing queue signals to the cell dispatcher when an appropriate number of data words have been accumulated and thus the respective byte packing queue is ready for servicing. A stream byte packing queue initiates a signal to cell dispatcher **385***b* indicating data is ripe for transfer upon the occurrence of one of three trigger conditions: upon receipt of the first 96 bytes of a new packet; upon receipt of an end of packet flag; or upon receipt of 64 bytes of data which do not comprise the beginning or the end of a packet. The flags received along with the data words from the bit packing queue **381***a* are used to evaluate the data words received from byte packing queue **381***a* to determine when the various conditions have been satisfied.

[**0057**] The individual stream byte packing queues are sized to ensure that entries received from bit packing queue **381***a* are not lost while waiting for service by cell dispatcher **385***b*. In an embodiment that includes 16 OC-3 input streams, all of the stream byte packing queues are identically sized to include eighteen 64 bit words.

[**0058**] L2 Header Processing

[**0059**] As described above, data words are transferred from byte packing queue **381***b* to segmentation buffer **387** as they become available. In addition, coincident with the transfer, the first 32 bytes of data packet (four 8 byte data words) are also transferred to L2 header buffer **383** of L2 pattern match decoder **382**. For middle packets which include up to 64 bytes of data in the middle of a packet, cell dispatcher **385***b* disables the transfer of data to the L2 header buffer and no L2 header processing is required. The present invention processes packets in-line. In-line processing includes two components, one at the input prior to storage of a packet (or portion of the packet in packet memory, i.e. memory **104** of **FIG. 2***b*) and the other at the output after packets are read from packet memory. The input component conditionally strips the L2 header, finds the start of the L3

header and checks the L3 header for errors. In addition, other early error verifications can be performed to assure that bad packets are dropped prior to storage in packet memory. The output component attaches a new L2 header to a packet and updates certain L3 header fields as required. In-line processing significantly increases throughput in the router allowing for the early dumping of packets prior to storage in packet memory. The input component is described immediately below. The output component is described later in the specification.

[**0060**] While the first 32 bytes of a packet are transferred to L2 header buffer **383**, a flag is set and passed to L2 header parser **389** indicating a packet is available to process. Associated with L2 header parser **389** is a decoder **409** for processing L2 header information for each packet. Decoder **409** can include a micro-coded engine or subroutine operating to decode header information (L2 header information). In the implementation shown, the micro-coded engine includes four sub-processes **409-0** through **409-3**. The number of sub-processes can be selected to minimize the number of clock cycles required to examine the L2 header information. L2 header parser **389** under the control of decoder **409** consumes bytes stored in L2 header buffer **383** to locate a header offset. The header offset indicates the end of the L2 header and the beginning of the next layer header (e.g., L3 header) for a given packet.

[**0061**] In addition, the L2 header parser examines portions of the L2 header for errors. Errors arising form unrecognized L2 headers, unconfigured L2 connections, or other L2 errors are immediately identified and dropped prior to being written to memory. In one implementation packets are dropped by never transferring the packet out of the segmentation buffer **387**. The dropped packet is overwritten in time by a next 64 byte data word from byte packing queue **381***b*.

[**0062**] Associated with the L2 header parser is a per stream L2 state queue **419**. Per stream state queue **499** stores flags associated with a micro-code starting address, priority (precedence) flag for the stream, an interface index mapping to one or more logical interfaces, virtual connection stream state information and channel stream state information. The per stream state queue stores information associated with each stream so as to assure continuity in steam processing.

[**0063**] Segmentation buffer **387** is sized to accommodate up to eight data word (64 bytes) in one implementation. Associated with the segmentation buffer is a selectable start read pointer that indicates the location in the segmentation buffer to begin read operations (when reading data from the segmentation engine by packetizer **391**). Data bytes are read from byte packing queue **381***b* and stored in segmentation buffer **387**. Subsequent byte transfers (beyond the first 64 bytes) from byte packing queue **381***b* can result in a wrap around condition. Wrap around results in the overwriting of a portion of the contents of the segmentation buffer. Wrap around may arise when a portion of the L2 header is overwritten in accordance with the strip offset determined as part of L2 header processing. A circular buffer can be used to minimize the size of the buffer required to support the line rate processing.

[**0064**] The segmentation buffer provides temporary storage of the data words transferred from the byte packing queue while the L2 processing completes. Transfer from segmentation buffer **387** by cell packetizer **391** to cell payload queue **388** is initiated upon completion of the L2 processing and delivery of the offset information to cell packetizer **391**. Cell dispatcher **385***b* triggers the transfer of middle data words in a packet (all data words after the first data word associated with a packet).

[**0065**] In one implementation, segmentation buffer **387** can be a dual output ported device that includes two output ports for providing data bytes to cell packetizer **391** at twice the input rate. The dual porting allows for the reading of up to sixteen bytes of data from segmentation buffer **387** during a single read cycle by cell packetizer **391**.

[**0066**] Cell Formation

[**0067**] Cell packetizer **391** is an engine that services cell segmentation buffer **387** providing a cell sized amount of data to cell payload queue **388**. As described above, cells are the preferred storage mechanism for storing a fixed portion of a packet. Each cell includes a data portion and a header portion. In one implementation, cell packetizer **391** transfers 64 bytes of data to cell payload queue **388**. Cell packetizer receives as an input cell data from segmentation buffer **387** and from the segmentation state queue **401** as well as offset information from L2 pattern match decoder **382**. The operation of the segmentation state queue **401** is described in detail below.

[**0068**] As described above, the offset information is derived from the L2 packet header. The offset information can be provided in the form of a pointer pointing to a particular location in the data stored in segmentation buffer **387**. The pointer can be used to indicate the particular byte in the data transferred from the segmentation buffer that marks the beginning the next layer header. Cell packetizer **391** can discard those portions of the data that belong to the L2 header necessitating further reads from the byte packing queue **381***b* to fill a data portion of a cell (64 byte) in cell payload queue **388**.

[**0069**] The offset may not arise exactly on an eight byte boundary necessitating the storage of overflow data. To facilitate block transfers from byte packing queue **381***b* to segmentation buffer **387** (8 byte blocks), an overflow queue is provided. Segmentation state queue **401** includes a queue sized to contain N-1 bytes of data for each stream, where N is equal to the size of the block transfers from byte packing queue **381***b* to segmentation buffer **387**. Extra bytes that are required to be read from byte packing queue **381***b* to facilitate the filling of a cell in cell payload queue **388** are stored in segmentation state queue **401**.

[**0070**] At the beginning of a cell packing operation, cell packetizer **391** first checks to determine if extra bytes for the stream being processed are present in segmentation state queue **401**. If so, the extra bytes are loaded first prior to the loading of bytes from the segmentation buffer **387**. A cell segmentation engine **400** oversees the transfer of data from segmentation state queue **401** to cell packetizer **391**. In one implementation, each queue in the segmentation state queue **401** stores up to seven leftover bytes of data that may have been required to be read in order to fill the previous cell associated with the given stream.

[**0071**] The position in cell payload queue **388** to which the cell is written is controlled by buffer pool manager **393**. Cell dispatcher **385***b* receives feedback from buffer pool manager **393** as entries are extracted from cell payload queue **388**. A

slot must be available in cell payload queue **388** prior to the packetizing of the cell data and extraction from byte packing queue **381***b*. Buffer pool manager includes a pointer that indicates the next available cell in the cell payload queue that can be written to by cell packetizer **391**. As each cell is written into cell payload queue **388** an associated header is written into the cell header queue **390**.

[**0072**] Ordering of cells at the stream level is accomplished through use of cell state queue **402**. Cells in the same stream must be extracted from cell payload queue **388** in sequential order. Ordering is achieved by writing a code to a per stream queue **388** as each cell is transferred into cell payload queue **388**. More specifically, for each write of a data portion of a cell by cell packetizer **391** into cell payload queue **388**, cell segmentation engine **400** writes a code that is stored in cell state queue **402**. Cell state queue **402** includes a queue for each stream in the input. As each cell is written to cell payload queue **388**, a code including address and state information is written to cell state queue **402**. The address information includes a pointer to the location of the data portion of the cell in cell payload queue **388**. The state information includes information indicating whether the cell is the first cell, middle cell or last cell in a packet. In one implementation, the code is 10 bits in length and includes two (2) bits indicating whether the cell is a first cell, middle cell or last cell as well as eight (8) bits of address information.

[**0073**] L3 Header Processing

[**0074**] The next layer header processing can be performed in parallel to the data transfers to the cell payload queue. In one implementation, L3 header processing is performed in parallel to the packing of data by cell packetizer **391**. L3 header parser **406** snoops on the bus between the cell packetizer and cell payload queue **388** examining the L3 header data to derive a header to be stored in an associated entry in cell header queue **390**.

[**0075**] Associated with L3 header parser **406** is a L3 state buffer **408**. The L3 state buffer **408** includes a plurality of queues, one for each stream, and operates to store L3 header information derived from a previous portion of a packet by L3 header parser **406**. L3 header data can extend beyond the first 64 bytes of a data packet. Accordingly, L3 header processing can begin and then be delayed due to insufficient data (not all of the L3 header information is part of the first 64 byte data word associated with a packet). Alternatively, L3 processing can be delayed until receipt of all of the L3 header data. In either case, L3 state buffer **408** is used to store L3 state information while waiting for a second data portion for a packet in the same stream to be processed by cell packetizer **391**.

[**0076**] L3 header processing includes error checking for checksum errors, packet length errors, or other L3 header errors. If an error is detected, then the pointer associated with a cell that is written to the cell payload queue **388** is not committed. Pointers associated with cells in the cell payload queue are stored in cell state queue **402**. The operation of cell state queue **402** is described in greater detail below.

[**0077**] Cell headers are constructed and stored in cell header queue **390**. The format for a cell header is described in greater detail below in association with **FIG. 3***e*. L3 header parser **406** examines L3 header data while snooping on the bus to derive values for the various L3 flags stored in a cell header. L2 header parser **389** derives a series of L2 flags while processing the L2 header. These flags are used by the look-up engine and are based on the type of L2 packet received. L2/L3 header merge **407** collects L2 flag data from L2 header parser **389** and stores the flag data temporarily while the L3 header is processed. L3 flags detected by the L3 header parser are combined with the L2 flags and stored in the cell header.

[**0078**] The flags store information required for the efficient down-stream processing of a given packet. In one implementation, the L2 flags derived during L2 processing include a packet loss priority flag, a send packet to processor flag, a sample packet flag and a physical multicast flag. The L3 flags derived during L3 header processing include an option flag, packet priority flag, transmission control protocol (TCP) flag, protocol type flag, and DF (don't fragment) flag.

[**0079**] The packet loss priority flag is used to determine a priority for packets that are dropped due to congestion in an output port of the router. The dropping of packets due to congestion is described in greater detail in co-pending patent application entitled "HIGH SPEED SWITCHING DEVICE", filed Apr. 18, 1997, assigned Ser. No. 08/844,171, which is hereby expressly incorporated by reference.

[**0080**] The send packet to processor flag indicates that a packet is not to be forwarded out from the router and only sent to the controller for processing. The sample packet flag indicates that a packet is to be forwarded (to the appropriate output port) and also sent to the controller for processing. The physical multicast flag identifies multicast packets.

[**0081**] The option flag indicates that the packet has an option header (indicates that the L3 header is greater than the minimal size, e.g., 20 bytes for IPV4 packets). The packet priority flag indicates the priority of the packet and is used by the output portion of the multi-function multiport. The TCP flag indicates the packet is a TCP packet. The protocol flag indicates the L3 protocol type. The protocol flag is used by the input switch to determine the length of the key to be passed to the controller. The DF flag indicates whether the packet is to be fragmented. Fragmentation may occur at the output interface depending on the configuration of the particular interface. Setting the DF flag will result in the dropping of the packet at the output interface rather than fragment.

[**0082**] The header of a cell is generated from data associated with the cell type (indirect cell, direct cell, and data cells) along with header flags extracted during L2 and L3 header processing as described above. In one implementation, the position in the cell header queue **389** that the header is written to is derived from the location in the cell payload queue **388**. The parallelism of the addressing for headers and cell data portions simplifies cell processing.

[**0083**] Spraying of Packets across Memory Banks

[**0084**] Bank spray engine **392** includes a spray bank queue **395**, cell spray engine **396** and bank queue engine **397**. The queues in cell state queue **402** are serviced by cell spray engine **396**. Spray engine **396** sequences through the queues and delivers addresses associated with cells to spray bank queue **395**. Spray bank queue **395** includes a plurality of queues, one for each active memory bank in the router. In

one implementation, spray bank queue **395** includes eight queues corresponding to eight memory banks. Entries in the cell state queues are serviced sequentially and provided as an input to a queue in spray bank queue **395**. Consecutive entries in a queue associated with a particular stream in cell state queue **402** are sprayed sequentially to the queues of spray bank queue **395**.

[**0085**] Spray bank queue **395** is serviced in a strict round-robin fashion by bank queue engine **397**. Bank queue engine **397** receives as an input the next entry from spray bank queue **395**. Each entry indicates both an address in the cell payload queue **388** for the next cell to be processed as well as an indication of whether or not the cell is a first cell, middle cell or last cell to be processed. Bank engine queue **397** extracts the indicated cell data from payload queue **388** and the associated header information from cell header queue **390**. The complete cell is provided as an output to input switch interface **304**.

[**0086**] The format for transfers between the multi-function multiport and the input switch is shown in **FIG. 3***e*. A cell **350** transferred from a multi-function multiport **150** to the input switch contains a cell header **352** and cell data **354**. Cell header **352** includes a type field **356**, stream field **358**, and packet header fields **360**. In addition, cell header **352** includes an independent read request in the form of a multi-function multiport identifier **366** and address **368**.

[**0087**] The type field **356** indicates the type of cell to be transferred from the multi-function multiport. At each cell slot (20 clock cycles in one embodiment), a multi-function multiport can transfer either a data cell, an indirect cell placeholder, or a delayed indirect cell placeholder. Data cells contain data associated with an incoming packet. An indirect cell placeholder is an empty cell, and is used in conjunction with indirect addressing for the storage of the cells in the global data buffer **104**. Delayed indirect cell placeholders arise when a data stream that requires indirect addressing terminates at a time prior to the designated time for writing the last indirect addressing cell associated with the data stream to global data buffer **104**. The generation and operation of indirect placeholders and delayed indirect placeholders will be discussed in greater detail below in conjunction with **FIG. 7**.

[**0088**] Stream field **358** indicates the stream to which the cell data belongs. In one embodiment of the present invention, each multi-function multiport is capable of handling up to sixteen separate streams of data at a time, one on each of its respective 16 input ports.

[**0089**] Packet header field **360** contains header information associated with a given packet and includes start offset information, packet length, interface index information and L2 and L3 flags generated as part of the L2 and L3 decoding processes recited above.

[**0090**] Multi-function multiport identifier **366** identifies the multi-function multiport which is sourcing the read request. Address **368** indicates the address in global data buffer **104** to be read.

[**0091**] Referring now to **FIG. 4, a** single cell **450** is transferred from a multi-function multiport **150** to input switch **100** at each cell (time) slot "T". For a given cell slot "T", input switch **100** receives a total of "N" cells, where "N" is equal to the number of multi-function multiports.

[**0092**] In one embodiment, cells from a given stream can be written to memory in an order that is different from the arrival order. These out of order writes are performed to make efficient use of scarce bandwidth between the multi-function multiports and the input switch. When a packet comes in to the multi-function multiport, it is broken up into cells as described above as the bytes arrive and the cells are placed in per-bank output queues on the way to the input switch. These queues are designed to share scarce interconnect bandwidth between the streams of a multi-functional multiport in the most efficient way possible, but they have the detrimental effect of reordering cells at the interface between the multi-function multiport and the input switch. Thus the cells from a given stream can arrive at the input switch out of order. The multi-function multiport marks the data cells of a stream with one of four codes: first cell (FC); intermediate data cell (DC); last cell (LC); or first cell which happens to be also a last cell (FLC).

[**0093**] Input Switch

[**0094**] Referring again to **FIG. 2***b*, input switch **100** routes cells to global data buffer **104**. The architecture of input switch **100** and the routing of cells through the input switch **100** is described in greater detail in co-pending patent application entitled "HIGH SPEED SWITCHING DEVICE", filed Apr. 18, 1997, assigned Ser. No. 08/844,171.

[**0095**] Memory Architecture

[**0096**] The memory architecture and routing of cells through global data buffer **104** is described in greater detail in co-pending patent application entitled "HIGH SPEED SWITCHING DEVICE", filed Apr. 18, 1997, assigned Ser. No. 08/844,171.

[**0097**] Incoming packets are broken up into as many cells as needed and the cells are written to the global data buffer as they arrive. The global data buffer is treated as a single large circular buffer. The input switch maintains an array of write pointers, one per active bank, to keep track of where to write the next cell. The pointers start out at one end of the buffer and advance until they eventually wrap around and overwrite packets that were written a long time ago. An ageing mechanism is used to guard against reading cells that may have been overwritten by subsequent packets. The cells of packets arriving on a given stream are interleaved strictly across the active banks to spread the bandwidth load.

[**0098**] Controller

[**0099**] Controller **106** includes controller memory **109**, route look-up engine **110**, an input switch interface and output switch interface. Controller **106** receives a route look-up request from input switch **100** at the input switch interface. Controller **106** performs a best match look-up and outputs a notification through the output switch interface to output switch **102**. The process of matching keys retrieved from a packet in order to determine a best match route through the router is described in greater detail in co-pending patent application entitled "HIGH SPEED VARIABLE LENGTH BEST MATCH LOOK-UP IN A SWITCHING DEVICE", filed on Dec. 16, 1996, by Fergusen et al., Ser. No. 08/767,576, which is hereby expressly incorporated by reference.

[**0100**] The notification includes a result which indicates the multi-function multiport to be used in the transfer of the packet to its destination.

[0101] Referring now to **FIG. 5**, the data structure associated with the notification outputted by the controller **106** to the output switch **102** is shown. Th data structure **1400** for the notification includes a mask **1402**, a next hop index pointer **1404**, full address **1406**, offsets **1408** and packet length **1410**.

[0102] The mask field **1402** is used to indicate which multi-function multiport connected to output switch **102** is to transfer the packet. In one embodiment, the notification can be sent to more than one multi-function multiport resulting in the broadcast of the associated packet.

[0103] Associated with each multi-function multiport **150** is a storage **310**. The next hop index pointer points to a location in storage **310**. Storage **310** is used to store media header information associated with a particular type of packet transfer. Next hop addresses, media headers and storage **310** will be described in greater detail below in association with the output section of multi-function multiport **150**.

[0104] The full address **1406** indicates the starting address in the global data buffer where the first cell in the packet is stored. As was described above, offsets **1408** provide linking information for retrieving cells or an indirect cell associated with the packet. Packet length filed **1410** indicates the length of the associated packet and can be used to determine if indirect cells will have to be retrieved.

[0105] Output Switch

[0106] Referring again to **FIG. 2***b*, output switch **102** processes notifications received from controller **106** for transfer to an appropriate multi-function multiport **150**. The architecture and routing of cells through output switch **102** is described in greater detail in co-pending patent application entitled "HIGH SPEED SWITCHING DEVICE", filed Apr. 18, 1997, assigned Ser. No. 08/844,171, which is hereby expressly incorporated by reference.

[0107] Output Section of a Multi-function Multiport

[0108] Referring now to **FIG. 6**, each multi-function multiport **150** includes an output switch interface **316**, an input switch interface **304** including read request queues, head and tail queue buffer **318**, an output request processor **306**, an line output interface **308**, storage device (memory) **310**, stream output buffers **312** and output formatter **314**.

[0109] A multi-function multiport **150** receives notification that a packet is to be processed in the form of a notification received at the output switch interface **316**. Output request processor **306** processes notifications, storing each in an appropriate location in head and tail queue buffer **318** and servicing notification requests as they make their way through the various priority queues in head and tail queue buffer **318**. The servicing of requests results in the generation of a read request to input switch **100** associated with the first address in memory where the packet (associated with the particular notification) is stored in the global data buffer. The architecture and operation of the notification queues is described in greater detail in co-pending patent application entitled "HIGH SPEED SWITCHING DEVICE", filed Apr. 18, 1997, assigned Ser. No. 08/844, 171.

[0110] As cells are received back at the multi-function multiport **150** (responsive to the read requests), they are stored in an associated stream output buffer **312**. Stream output buffer **312** includes a plurality of FIFOs, one for each stream. Each cell received for a stream is placed in the streams associated FIFO. For given packet, the multi-function multiport stores a fixed number of cells (in the FIFO) required to provide a streamed output prior to initializing the output of the stream to line output interface **308**. In one embodiment of the present invention, twelve cells are stored prior to beginning output (stream data) from the output port. The selection of the number of cells for storage in output buffer **312** is based on the latency in the read process (number of clock cycles between a read request from an multi-function multiport and the arrival of the cell associated with the read request to the output port).

[0111] Output formatter **314** receives the cells from output buffer **312** and couples the data with media header information stored in memory **310**. Each request (notification) received from output switch **102** includes a next hop index. The next hop index indicates the starting address in memory **310** of the media header information associated with a given type of transmission (derived from the destination of the packet). Media header information stored in memory **310** can be loaded in memory **310** upon initialization of the router and updated by the controller as required. The media header information includes all pertinent L2 construction parameters as well as L3 protocol type information for L2/L3 in-line header re-writing operations.

[0112] As described above, "in-line" packet processing includes an output component that constructs a new L2 header for a given packet prior to transfer out of the router. Output formatter **314** constructs a new L2 header based on the media header information. In one implementation, the next hop index points to a sequence capable of producing the desired L2 header. The sequence can be of the form of a data structure that includes references to common components of L2 headers which can be stored on-chip in an on-chip template as well as references to specific L2 header templates which are unique to a given next hop index. The unique templates can be stored off-chip in SRAM. Ethernet, Frame Relay, and other L2 header types are created and stored in templates.

[0113] The L2 header data structure includes three parts: a program header, L2 header byte generation template and one or more templates. The program header includes flags for initiating error checking functions and is used to define the size of the sequence. Error checking functions include calculating a checksum for the L2 header.

[0114] The L2 byte generation template stores flags indicating which bytes of the data structure are generated from internal or external templates.

[0115] A template includes instructions for generating particular L2 header data for a given next hop index. In addition to generating a new L2 header based on the next hop index, output formatter **314** can be required to manipulate portions of the L3 header that have been stored along with the packet in packet memory (memory **104** in **FIG. 2***b*). For example, output formatter **314** calculates new checksum data for the L3 header prior to transmission from the router.

[0116] Output formatter **314** couples the cell data returned from memory with the appropriate media header (L2 header) to generate a complete packet for transfer out of router **20** on the line output interface **308**.

[0117] In-line Packet Processing Overview

[0118] Referring now to **FIG. 7**, in a method for in-line processing packets for routing through a switch, a packet is received at a multi-function multiport (**702**). Packet headers are pre-processed to the evaluate L2 header data for errors and locate the start of the next layer header (**704**). L2 header data is conditionally stripped as part of the pre-processing (**706**). L3 header data is evaluated in parallel for errors (**708**). The cells are transferred to an input switch (**710**). Input switch removes the key information from the first cell in a packet and routes the cells to memory banks resident in the multi-function multiports in a time division multiplexed manner (**712**). The key, full address of the first cell and offset information is transferred as a look-up request to a controller (**714**). The controller performs a best match look-up and generates a result of the look-up (**716**). The result includes the destination port (multi-function multiport), address, offset information and next hop index. A notification including the result is transferred to the output switch for transfer to the appropriate multi-function multiport (**718**).

[0119] Upon receipt of a notification, the multi-function multiport generates read requests a cell at a time to the input switch for the data associated with the packet (**720**). When the memory bank receives the request from the input switch, cell data are transferred to an appropriate multi-function multiport through the output switch (**722**). Upon receipt, the multi-function multiport constructs a L2 header from templates indicated by the next hop index and modifies the L3 header as appropriate (**724**). Finally, the cell data and media header information is streamed to the destination (**726**) and the process ends (**728**).

[0120] The present invention has been described in terms of specific embodiments, which are illustrative of the invention and not to be construed as limiting. Other embodiments are within the scope of the following claims.

What is claimed is:

1. A method of in-line processing a data packet while routing the packet through a router in a system transmitting data packets between a source and a destination over a network including the router, the method comprising:

receiving the data packet;

pre-processing layer header data for the data packet as the data packet is received and prior to transferring any portion of the data packet to packet memory;

storing the data packet in the packet memory;

determining a routing through the router including a next hop index describing the next connection in the network;

retrieving the data packet from the packet memory;

constructing a new layer header for the data packet from the next hop index while the data packet is being retrieved from memory; and

coupling the new layer header to the data packet prior to transfer from the router.

2. The method of claim 1 where the pre-processing step includes screening header layer data associated with the data packet for errors and dropping a bad data packet prior to transferring any portion of the data packet to packet memory.

3. The method of claim 2 where the screening includes screening Layer 2 (L2) and Layer 3 (L3) headers for errors.

4. The method of claim 3 where the L2 header is examined to detect errors arising from unrecognized L2 header formats and unconfigured L2 connections.

5. The method of claim 3 where the L3 header is examined to detect data packets with checksum errors, packet length errors and L3 header errors.

6. The method of claim 1 where the step of storing the data packet in memory includes dividing the data packet into cells of a fixed size and storing the cells in a distributed memory.

7. The method of claim 6 where the step of retrieving the data packet from memory includes reconstructing the packet from cells stored in the memory.

8. The method of claim 1 where the pre-processing includes stripping L2 header data from a data packet prior to storage the packet memory.

9. The method of claim 8 where the pre-processing step includes identifying the beginning of the L3 header and examining the L3 header for errors prior to the storage of the data packet in the packet memory.

10. The method of claim 1 where the next hop index is a pointer to a sequence stored in a storage device within the router and the step of constructing a new layer header includes executing the sequence.

11. The method of claim 10 where the execution of the sequence includes retrieving a common template for constructing a common portion of an L2 header to be attached to the data packet and a custom template for constructing a unique portion of the L2 header.

12. The method of claim 11 where the common and unique templates are executable code operable to construct and associated portion of an L2 header for the data packet.

13. The method of claim 1 where the step of receiving the data packet includes receiving a plurality of data packets for processing from a plurality of input ports representing a plurality of streams of data to be routed through the router.

14. The method of claim 1 where the step of pre-processing the data packet includes dividing the data packet into fixed length cells and parsing the L2 header associated with the first cell of the data packet prior to receipt of the entire data packet.

15. The method of claim 14 where the step of parsing the L2 header includes examining the L2 header for errors and identifying the start of a next layer header in the data packet.

16. The method of claim 15 including temporarily storing the cells for a data packet in a queue after L2 header parsing is completed and spraying consecutive cells in a data packet to a different bank in the packet memory.

17. The method of claim 16 including snooping while the cells are being written to the queue and parsing the L3 header including examining the L3 header for errors.

18. The method of claim 16 including dropping a data packet if errors are detected in the L2 header during L2 header parsing without storing a cell associated with the data packet in the queue.

19. A router for in-line processing a data packet while routing the packet in a system transmitting data packets between a source and a destination over a network including the router, the router comprising:

a packet memory for storing portions of the data packet;

an input port for receiving a data packet including a header processing engine for evaluating header layer information upon receipt and prior to the storage of the data packet in the packet memory;

a controller for determining packet routing through the router including a next hop index indicating the next connection in the network for each data packet to be routed through the router; and

an output port for transferring the data packet to the destination including an output formatter for constructing a layer header for the data packet to facilitate the transfer of the packet to the destination.

20. The router of claim 19 where the header processing engine is operable to screen header layer data associated with the data packet for errors and drop a bad data packet prior to transferring any portion of the data packet to packet memory.

21. The router of claim 20 where the header processing engine screens Layer 2 (L2) and Layer 3 (L3) headers for errors.

22. The router of claim 21 where the header processing engine examines the L2 header to detect errors arising from unrecognized L2 header formats and unconfigured L2 connections.

23. The router of claim 21 where header processing engine examine the L3 header to detect data packets with checksum errors, packet length errors and L3 header errors.

24. The router of claim 19 includes a cell packetizer operable to divide the data packet into cells of a fixed size prior to transfer to packet memory.

25. The router of claim 24 where the packet memory is a distributed memory and the router includes a spray engine for distributing cells across the distributed memory.

26. The router of claim 19 where the header processing engine strips L2 header data from a data packet prior to storage the packet memory.

27. The router of claim 26 where the header processing engine identifies the beginning of an L3 header and examines the L3 header for errors prior to the storage of the data packet in the packet memory.

28. The router of claim 19 where the next hop index is a pointer to a sequence stored in a storage device within the router and the output formatter executes the sequence to construct a new layer header.

29. The router of claim 28 where the storage device includes a common template for constructing a common portion of an L2 header to be attached to the data packet and a custom template for constructing a unique portion of the L2 header.

30. The router of claim 29 where the storage device is distributed and includes a local portion within the output port and a remote portion elsewhere in the routing device and where the common template is stored in local portion and the custom template is stored in the remote portion.

31. The router of claim 29 where the common and unique templates are executable code operable to construct and associated portion of an L2 header for the data packet.

32. The router of claim 19 further including a plurality of input ports and a plurality of output ports, each of the input ports and output ports configurable to receive and process a plurality of data packets representing a plurality of streams of data to be routed through the router.

33. The router of claim 19 further including a cell packetizer operable to divide the data packet into fixed length cells and a L2 parsing engine operable to examine the L2 header associated with the first cell of the data packet prior to receipt of the entire data packet.

34. The router of claim 33 where the L2 parsing engine is operable to examine the L2 header for errors and identify the start of a next layer header in the data packet.

35. The router of claim 34 further including a queue operable for temporarily storing the cells for a data packet after L2 header parsing is completed and a spray engine operable to spray consecutive cells in a data packet to a different bank in the packet memory.

36. The router of claim 35 further including a L3 parsing engine operable to snoop while the cells are being written to the queue and parse the L3 header including examining the L3 header for errors.

37. The router of claim 36 where the L2 parser engine is operable to drop a data packet if errors are detected in the L2 header during L2 header parsing without storing a cell associated with the data packet in the queue.

38. A method of routing a data packet through a router in a system transmitting data packets between a source and a destination over a network including the router, the method comprising:

receiving the data packet;

dividing the data packet into cells of a fixed size;

while dividing the data packet, pre-processing layer header data for the packet to locate the beginning of a next layer header;

storing the cells in a memory;

prior to storing a first cell associated with the packet in the memory, screening header layer data for errors and dropping a bad packet prior to transferring any cells into the memory;

determining a routing through the router including a next hop index describing the next connection in the network;

reconstructing the packet from cells stored in the memory; and

constructing new layer header data for the packet from the next hop index.

* * * * *