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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
26 November 2020 (26.11.2020)

WIPO | PCT

(10) International Publication Number
WO 2020/236292 A1

(51) International Patent Classification:
H04L 12/931 (2013.01) *H04L 12/741* (2013.01)
H04L 12/947 (2013.01) *H04L 12/751* (2013.01)
H04L 12/935 (2013.01)

(21) International Application Number:
PCT/US2020/024276

(22) International Filing Date:
23 March 2020 (23.03.2020)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

62/852,203	23 May 2019 (23.05.2019)	US
62/852,273	23 May 2019 (23.05.2019)	US
62/852,289	23 May 2019 (23.05.2019)	US

(71) Applicant: **CRAY INC.** [US/US]; 901 Fifth Avenue, Suite 1000, Seattle, Washington 98164 (US).

(72) Inventors: **FROESE, Edwin L.**; 3404 E. Harmony Road, Mail Stop 79, Ft. Collins, Colorado 80528 (US). **ALVERSON, Robert**; 3404 E. Harmony Road, Mail Stop 79, Ft. Collins, Colorado 80528 (US). **FRAGKIADAKIS, Konstantinos**; 3404 E. Harmony Road, Mail Stop 79, Ft. Collins, Colorado 80528 (US).

(74) Agent: **FEBBO, Michael A.** et al.; c/o Hewlett Packard Enterprise, 3404 E. Harmony Road, Mail Stop 79, Ft. Collins, Colorado 80528 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

(54) Title: DEADLOCK-FREE MULTICAST ROUTING ON A DRAGONFLY

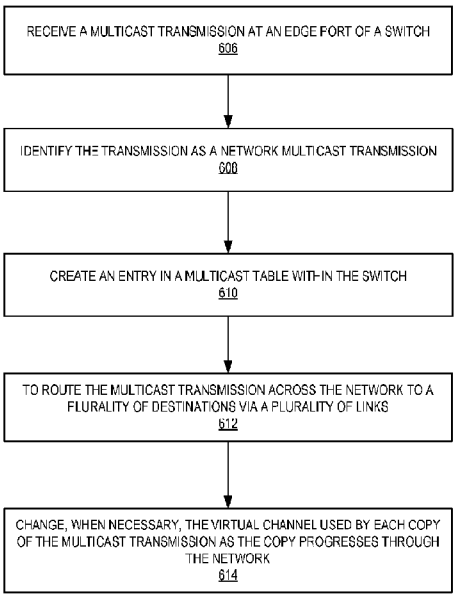

FIG. 6

(57) Abstract: Systems and methods are provided for managing multicast data transmission in a network having a plurality of switches arranged in a Dragonfly network topology, including: receiving a multicast transmission at an edge port of a switch and identifying the transmission as a network multicast transmission; creating an entry in a multicast table within the switch; routing the multicast transmission across the network to a plurality of destinations via a plurality of links, wherein at each of the links the multicast table is referenced to determine to which ports the multicast transmission should be forwarded; and changing, when necessary, the virtual channel used by each copy of the multicast transmission as the copy progresses through the network.

WO 2020/236292 A1

[Continued on next page]

Declarations under Rule 4.17:

— *as to the identity of the inventor (Rule 4.17(i))*
— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*
— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

Published:

— *with international search report (Art. 21(3))*

DEADLOCK-FREE MULTICAST ROUTING ON A DRAGONFLY

Statement of Government Rights

[0001] The invention(s) described herein were made with U.S. Government support under one or more of the contracts set forth below. The U.S. Government has certain rights in these inventions.

Contract Title	Customer / Agency	Contract Reference
FastForward-2	Lawrence Livermore National Security, LLC / Dept of Energy	Subcontract B609229 under prime contract DE-AC52-07NA27344
BeePresent	Maryland Procurement Office	H98230-15-D-0020; Delivery Order 003
SeaBiscuit	Maryland Procurement Office	II98230-14-C-0758
PathForward	Lawrence Livermore National Security, LLC / Dept of Energy	Subcontract B620872 under prime contract DE-AC52-07NA27344
DesignForward	The Regents of the University of California / Dept of Energy	Subcontract 7078453 under prime contract DE-AC02-05CII11231
DesignForward-2	The Regents of the University of California / Dept of Energy	Subcontract 7216357 under prime contract DE-AC02-05CII11231

Related Applications

[0002] This application claims the benefit of U.S. Provisional Patent Application No. 62/852273, filed May 23, 2019, entitled “Network Switch,” U.S. Provisional Patent Application No. 62/852203, filed May 23, 2019, entitled “Network Interface Controller,” and U.S.

Provisional Patent Application No. 62/852289, filed May 23, 2019, entitled "Network Computer System," the disclosures of which are incorporated herein by reference in their entirety for all purposes.

Description of Related Art

[0003] As network-enabled devices and applications become progressively more ubiquitous, various types of traffic as well as the ever-increasing network load continue to demand more performance from the underlying network architecture. For example, applications such as high-performance computing (HPC), media streaming, and Internet of Things (IOT) can generate different types of traffic with distinctive characteristics. As a result, in addition to conventional network performance metrics such as bandwidth and delay, network architects continue to face challenges such as scalability, versatility, and efficiency.

Brief Description of the Drawings

[0004] The present disclosure, in accordance with one or more various embodiments, is described in detail with reference to the following figures. The figures are provided for purposes of illustration only and merely depict typical or example embodiments.

[0005] FIG. 1 illustrates an example network in which various embodiments may be implemented.

[0006] FIG. 2 illustrates an example switching system that facilitates flow channels in accordance with various embodiments.

[0007] FIG. 3A illustrates crossbars implemented within an example crossbar switch in accordance with various embodiments.

[0008] FIG. 3B illustrates an example tile matrix corresponding to ports of the example edge switching system of FIG. 2 in accordance with various embodiments.

[0009] FIG. 3C illustrates an example tile making up the tile matrix of FIG. 3B in accordance with various embodiments.

[0010] FIG. 4A and FIG. 4B are block diagrams of an example FRF component implemented at each port of the example switching system of FIG. 2.

[0011] FIG. 5 illustrates an example of route selection in accordance with various embodiments.

[0012] FIG. 6 illustrates an example process for multicast routing on a dragonfly network in accordance with various embodiments.

[0013] FIG. 7 illustrates an example of multicast routing on a dragonfly network in accordance with various embodiments.

[0014] FIG. 8 illustrates an example of deadlock caused by unrestricted routing in accordance with various embodiments.

[0015] FIG. 9 illustrates an example of avoiding deadlock by restricting traffic to only route through intermediate switches that have connectivity with every other switch in the group in accordance with various embodiments.

[0016] FIG. 10 illustrates an example process for multicast traffic routing in accordance with some embodiments.

[0017] FIG. 11 is an example computing component that may be used to implement various features of embodiments described in the present disclosure.

[0018] The figures are not exhaustive and do not limit the present disclosure to the precise form disclosed.

Detailed Description

[0019] With the Dragonfly network topology, the network is subdivided into groups, with each group containing some number of switches and some number of endpoints. In the preferred system, each switch is an ASIC with many interconnected ports. The ports of each

switch are divided between edge ports, local ports, and global ports. Edge ports are ports in which traffic enters and leaves the network. For example, the ports connected to processing nodes are edge ports. Local ports connect to other switches within the same group. Global ports connect to other switches that are in a different group. In a fault-free Dragonfly network, every group is connected to every other group in the network by one or more global links and every switch within a group is connected to every other switch in the same group by one or more local links.

[0020] When a packet enters the network at one group (the source group) and leaves the network at a different group (the destination group), the packet may be globally routed (routed between its source and destination group) either minimally or non-minimally. The packet takes a global minimal path if it traverses one global link, directly connecting the source group to the destination group. With global non-minimal routing, some packets leaving the source group are routed to a group other than the destination group. This other group is chosen at random and is termed the intermediate group. On reaching an intermediate group, the packet is then routed directly to the destination group using a global link that directly connects the intermediate group to the destination group.

[0021] Multicast traffic is traffic that is intentionally replicated within the network. A multicast packet is received at an edge port of the network and copies of the packet are forwarded to a defined set of edge ports leaving the network. In effect, the multicast packet branches out in a tree-like fashion, where the root of the tree is the network edge port at which the packet is received and the leaves of the tree are the network edge ports at which copies of the packet leave the network. Multicast traffic is often used to facilitate one-to-many broadcast communication. Multicast traffic can also be used to facilitate any-to-all broadcast communication between the edge devices that are members of a multicast group.

[0022] To ensure endpoints do not receive duplicate copies of multicast packets, particular measures are taken in the switches. More specifically, the fabric routing techniques

of the switch include a multicast forwarding table and multicast forwarding filters. Multicast traffic is statically routed based on the configuration of the multicast forwarding tables. More specifically, the multicast table identifies the ports of the current switch to which copies of the multicast packet should be forwarded. This implicitly determines the number of copies generated at that point, as well, because for each multicast packet received at a port, only one copy is forwarded to each other port identified by the multicast table as requiring a copy. To prevent duplicate delivery of multicast packets, multicast forwarding filters conditionally prune copies of the multicast packet depending on the source group at which the multicast packet originated.

[0023] The physical topology of Dragonfly networks contains loops, thus the potential for deadlock exists. Deadlock is prevented for the multicast traffic by following the same routing rules for multicast traffic as are followed for unicast traffic. The multicast forwarding tables must be programmed to only send each copy of a multicast packet along a path that is permitted for Dragonfly routing, and that is a path that a unicast packet could also follow. To prevent the formation of flow control dependency loops that could lead to deadlock, the virtual channel (VC) in which a packet is flowing is advanced at specific points as a packet traverses its path across the network. Identical VC switching rules are employed for multicast traffic as are employed for unicast traffic. Applying the same routing rules to unicast and multicast traffic ensures unicast and multicast traffic can simultaneously coexist in the network.

[0024] Ethernet networks normally support multicast/broadcast and possibly floods in one form or another. The switches duplicate a packet arriving on one port so that it is sent out on many ports. While flow channels may serve as a point to point connection between fabric ingress and fabric egress, multicasts can be mapped onto flow channels with a flow tree created with one root at the fabric ingress and many branches leading to the fabric egress ports. The multicast packet is sent to all of the output ports and a flow is created or extended

in the Output Flow Channel Table (OFCT). However, with a multicast packet, all of these flows point back to the same Input Flow Channel Table (IFCT) entry, thus creating the tree. Each time a packet is copied to an additional output port an extra Ack will be returned to the input port. The IFCT can account for this with an additional Ack count value called the multicast_ack_flow. This count value is incremented by PktSize * (multicast_output_ports -1) each time a multicast packet is taken from the flow channel input queue. This count is decremented when new Acks are received from the Ack switch. Only when the count becomes zero is the master ack_flow incremented in the normal way and a new Ack sent back on the link.

[0025] FIG. 1 shows an example network 100 comprising a plurality of switches, which can also be referred to as a "switch fabric." As illustrated in FIG. 1, network 100 can include switches 102, 104, 106, 108, and 110. Each switch can have a unique address or ID within switch fabric 100. Various types of devices and networks can be coupled to a switch fabric. For example, a storage array 112 can be coupled to switch fabric 100 via switch 110; an InfiniBand (IB) based HPC network 114 can be coupled to switch fabric 100 via switch 108; a number of end hosts, such as host 116, can be coupled to switch fabric 100 via switch 104; and an IP/Ethernet network 118 can be coupled to switch fabric 100 via switch 102. For example, a switch, such as switch 102 may receive 802.3 frames (including the encapsulated IP payload) by way of Ethernet devices, such as network interface cards (NICs), switches, routers, or gateways. IPv4 or IPv6 packets, frames formatted specifically for network 100, etc. may also be received, transported through the switch fabric 100, to another switch, e.g., switch 110. Thus, network 100 is capable of handling multiple types of traffic simultaneously. In general, a switch can have edge ports and fabric ports. An edge port can couple to a device that is external to the fabric. A fabric port can couple to another switch within the fabric via a fabric link.

[0026] Typically, traffic can be injected into switch fabric 100 via an ingress port of an edge switch, and leave switch fabric 100 via an egress port of another (or the same) edge switch. An ingress edge switch can group injected data packets into flows, which can be identified by flow ID's. The concept of a flow is not limited to a particular protocol or layer (such as layer-2 or layer-3 in the Open System Interface (OSI) reference model). For example, a flow can be mapped to traffic with a particular source Ethernet address, traffic between a source IP address and destination IP address, traffic corresponding to a TCP or UDP port/IP 5-tuple (source and destination IP addresses, source and destination TCP or UDP port numbers, and IP protocol number), or traffic produced by a process or thread running on an end host. In other words, a flow can be configured to map to data between any physical or logic entities. The configuration of this mapping can be done remotely or locally at the ingress edge switch.

[0027] Upon receiving injected data packets, the ingress edge switch can assign a flow ID to the flow. This flow ID can be included in a special header, which the ingress edge switch can use to encapsulate the injected packets. Furthermore, the ingress edge switch can also inspect the original header fields of an injected packet to determine the appropriate egress edge switch's address, and include this address as a destination address in the encapsulation header. Note that the flow ID can be a locally significant value specific to a link, and this value can be unique only to a particular input port on a switch. When the packet is forwarded to the next-hop switch, the packet enters another link, and the flow-ID can be updated accordingly. As the packets of a flow traverse multiple links and switches, the flow IDs corresponding to this flow can form a unique chain. That is, at every switch, before a packet leaves the switch, the packet's flow ID can be updated to a flow ID used by the outgoing link. This up-stream-to-down-stream one-to-one mapping between flow ID's can begin at the ingress edge switch and end at the egress edge switch. Because the flow ID's only need to be unique within an incoming link, a switch can accommodate a large number of flows. For example, if a flow ID is 11 bits long, an input port can support up to 2048 flows.

Furthermore, the match pattern (one or more header fields of a packet) used to map to a flow can include a greater number of bits. For instance, a 32-bit long match pattern, which can include multiple fields in a packet header, can map up 2^32 different header field patterns. If a fabric has N ingress edge ports, a total number of N*2^32 identifiable flows can be supported.

[0028] A switch can assign every flow a separate, dedicated input queue. This configuration allows the switch to monitor and manage the level of congestion of individual flows, and prevent head-of-queue blocking which could occur if shared buffer were used for multiple flows. When a packet is delivered to the destination egress switch, the egress switch can generate and send back an acknowledgement (ACK) in the upstream direction along the same data path to the ingress edge switch. As this ACK packet traverses the same data path, the switches along the path can obtain the state information associated with the delivery of the corresponding flow by monitoring the amount of outstanding, unacknowledged data. This state information can then be used to perform flow-specific traffic management to ensure the health of the entire network and fair treatment of the flows. As explained in more detail below, this per-flow queuing, combined with flow-specific delivery acknowledgements, can allow the switch fabric to implement effective, fast, and accurate congestion control. In turn, the switch fabric can deliver traffic with significantly improved network utilization without suffering from congestion.

[0029] Flows can be set up and released dynamically, or "on the fly," based on demand. Specifically, a flow can be set up (e.g., the flow-ID to packet header mapping is established) by an ingress edge switch when a data packet arrives at the switch and no flow ID has been previously assigned to this packet. As this packet travels through the network, flow IDs can be assigned along every switch the packet traverses, and a chain of flow IDs can be established from ingress to egress. Subsequent packets belonging to the same flow can use the same flow IDs along the data path. When packets are delivered to the destination

egress switch and ACK packets are received by the switches along the data path, each switch can update its state information with respect to the amount of outstanding, unacknowledged data for this flow. When a switch's input queue for this flow is empty and there is no more unacknowledged data, the switch can release the flow ID (i.e., release this flow channel) and re-use the flow-ID for other flows. This data-driven dynamic flow setup and teardown mechanism can obviate the need for centralized flow management, and allows the network to respond quickly to traffic pattern changes.

[0030] Note that the network architecture described herein is different from software-defined networks (SDN's), which typically uses the OpenFlow protocol. In SDN, switches are configured by a central network controller, and packets are forwarded based one or more fields in the layer-2 (data link layer, such as Ethernet), layer-3 (network layer, such as IP), or layer-4 (transport layer, such as TCP or UDP) headers. In SDN such header-field lookup is performed at every switch in the network, and there is no fast flow ID-based forwarding as is done in the networks described herein. Furthermore, because the OpenFlow header-field lookup is done using ternary content-addressable memory (TCAM), the cost of such lookups can be high. Also, because the header-field mapping configuration is done by the central controller, the setup and tear-down of each mapping relationship is relatively slow and could require a fair amount of control traffic. As a result, an SDN network's response to various network situations, such as congestion, can be slow. In contrast, in the network described herein, the flows can be set up and torn down dynamically based on traffic demand; and packets can be forwarded by a fixed-length flow ID. In other words, flow channels can be data driven and managed (i.e., set up, monitored, and torn down) in a distributed manner, without the intervention of a central controller. Furthermore, the flow ID-based forwarding can reduce the amount of TCAM space used and as a result a much greater number of flows can be accommodated.

[0031] Referring to the example shown in FIG. 1, suppose that storage array 112 is to send data using TCP/IP to host 116. During operation, storage array 112 can send the first packet with host 116's IP address as the destination address and a predetermined TCP port specified in the TCP header. When this packet reaches switch 110, the packet processor at the input port of switch 110 can identify a TCP/IP 5-tuple of this packet. The packet processor of switch 110 can also determine that this 5-tuple currently is not mapped to any flow ID, and can allocate a new flow ID to this 5-tuple. Furthermore, switch 110 can determine the egress switch, which is switch 104, for this packet based on the destination (i.e., host 116's) IP address (assuming switch 110 has knowledge that host 116 is coupled to switch 104). Subsequently, switch 110 can encapsulate the received packet with a fabric header that indicates the newly assigned flow ID and switch 104's fabric address. Switch 110 can then schedule the encapsulated packet to be forwarded toward switch 104 based on a fabric forwarding table, which can be computed by all the switches in fabric 100 using a routing algorithm such as link state or distance vector.

[0032] Note that the operations described above can be performed substantially at line speed with little buffering and delay when the first packet is received. After the first packet is processed and scheduled for transmission, subsequent packets from the same flow can be processed by switch 110 even faster because the same flow ID is used. In addition, the design of the flow channels can be such that the allocation, matching, and deallocation of flow channels can have substantially the same cost. For example, a conditional allocation of a flow channel based on a lookup match and a separate, independent deallocation of another flow channel can be performed concurrently in nearly every clock cycle. This means that generating and controlling the flow channels can add nearly no additional overhead to the regular forwarding of packets. The congestion control mechanism, on the other hand, can improve the performance of some applications by more than three orders of magnitude.

[0033] At each switch along the data path (which includes switches 110, 106, and 104), a dedicated input buffer can be provided for this flow, and the amount of transmitted but unacknowledged data can be tracked. When the first packet reaches switch 104, switch 104 can determine that the destination fabric address in the packet's fabric header matches its own address. In response, switch 104 can decapsulate the packet from the fabric header, and forward the decapsulated packet to host 116. Furthermore, switch 104 can generate an ACK packet and send this ACK packet back to switch 110. As this ACK packet traverses the same data path, switches 106 and 110 can each update their own state information for the unacknowledged data for this flow.

[0034] In general, congestion within a network can cause the network buffers to fill. When a network buffer is full, the traffic trying to pass through the buffer ideally should be slowed down or stopped. Otherwise, the buffer could overflow and packets could be dropped. In conventional networks, congestion control is typically done end-to-end at the edge. The core of the network is assumed to function only as "dumb pipes," the main purpose of which is to forward traffic. Such network design often suffers from slow responses to congestions, because congestion information often cannot be sent to the edge devices quickly, and the resulting action taken by the edge devices cannot always be effective in removing the congestion. This slow response in turn limits the utilization of the network, because to keep the network free of congestion the network operator often needs to limit the total amount of traffic injected into the network. Furthermore, end-to-end congestion control usually is only effective provided that the network is not already congested. Once the network is heavily congested, end-to-end congestion control would not work, because the congestion notification messages can be congested themselves (unless a separate control-plane network that is different from the data-plane network is used for sending congestion control messages).

[0035] In contrast, the flow channels can prevent such congestion from growing within the switch fabric. The flow channel mechanism can recognize when a flow is experiencing some degree of congestion, and in response can slow down or stop new packets of the same flow from entering the fabric. In turn, these new packets can be buffered in a flow channel queue on the edge port and are only allowed into the fabric when packets for the same flow leave the fabric at the destination edge port. This process can limit the total buffering requirements of this flow within the fabric to an amount that would not cause the fabric buffers to become too full.

[0036] With flow channels, the switches have a reasonably accurate state information on the amount of outstanding in-transit data within the fabric. This state information can be aggregated for all the flows on an ingress edge port. This means that the total amount of data injected by an ingress edge port can be known. Consequently, the flow channel mechanism can set a limit on the total amount of data in the fabric. When all edge ports apply this limit action, the total amount of packet data in the entire fabric can be well controlled, which in turn can prevent the entire fabric from being saturated. The flow channels can also slow the progress of an individual congested flow within the fabric without slowing down other flows. This feature can keep packets away from a congestion hot spot while preventing buffers from becoming full and ensuring free buffer space for unrelated traffic.

[0037] FIG. 2 illustrates an example switch 202 (which may be an embodiment of any one or more of switches 102, 104, 106, 108, and 110) that may be used to create a switch fabric, e.g., switch fabric 100 of FIG. 1. In this example, a switch 202 can include a number of communication ports, such as port 220. Each port can include a transmitter and a receiver. Switch 202 can also include a processor 204, a storage device 206, and a flow channel switching logic block 208. Flow channel switching logic block 208 can be coupled to all the

communication ports and can further include a crossbar switch 210, an EFCT logic block 212, an IFCT logic block 214, and an OFCT logic block 216.

[0038] Crossbar switch 210 includes crossbars, which can be configured to forward data packets and control packets (such as ACK packets) among the communication ports. EFCT logic block 212 can process packets received from an edge link and map the received packets to respective flows based on one or more header fields in the packets. In addition, EFCT logic block 212 can assemble FGFC Ethernet frames, which can be communicated to an end host to control the amount of data injected by individual processes or threads. IFCT logic block 214 can include the IFCT, and perform various flow control methods in response to control packets, such as endpoint-congestion-notification ACKs and fabric-link credit-based flow control ACKs. OFCT logic block 216 can include a memory unit that stores the OFCT and communicates with another switch's IFCT logic block to update a packet's flow ID when the packet is forwarded to a next-hop switch.

[0039] In one embodiment, switch 202 is an application-specific integrated circuit (ASIC) that can provide 64 network ports that can operate at either 100 Gbps or 200 Gbps for an aggregate throughput of 12.8 Tbps. Each network edge port may be able to support IEEE 802.3 Ethernet, and Optimized-IP based protocols as well as Portals, an enhanced frame format that provides support for higher rates of small messages. Ethernet frames can be bridged based on their L2 address or they can be routed based on their L3 (1Pv4//1Pv6) address. Optimized-IP frames may only have an L3 (1Pv4/1Pv6) header, and are routed. Specialized NIC support can be used for the Portals enhanced frame format, and can map directly onto the fabric format of network 100, e.g., a fabric format that provides certain control and status fields to support a multi-chip fabric when switches/switch chips, such as switches 102, 104, 106, 108, and 110 are connected and communicate with each other. As alluded to above, a congestion control mechanism based on flow channels can be used by

such switches, and can also achieve high transmission rates for small packets (e.g., more than 1.2 billion packets per second per port) to accommodate the needs of HPC applications.

[0040] Switch 202 can provide system-wide Quality of Service (QoS) classes, along with the ability to control how network bandwidth is allocated to different classes of traffic, and to different classes of applications, where a single privileged application may access more than one class of traffic. Where there is contention for network bandwidth, arbiters select packets to forward based on their traffic class and the credits available to that class. Network 100 can support minimum and maximum bandwidths for each traffic class. If a class does not use its minimum bandwidth, other classes may use the unused bandwidth, but no class can get more than its maximum allocated bandwidth. The ability to manage bandwidth provides the opportunity to dedicate network resources, as well as CPUs and memory bandwidth to a particular application.

[0041] In addition to support for QoS classes, switch 202 effectuates flow channel-based congestion control, and can reduce the number of network hops, e.g., in a network having a dragonfly topology, from five network hops to three. The design of switch 202, described in greater detail below, can reduce network cost and power consumption, and may further facilitate use of innovative adaptive routing algorithms that improve application performance. A fabric created by a plurality of switches, such as a plurality of switches 202 may also be used in constructing Fat-Tree networks, for example when building a storage subsystem for integration with third-party networks and software. Further still, the use of switch 202 enables fine-grain adaptive routing while maintaining ordered packet delivery. In some embodiments, switch 202 may be configured to send the header of a packet from an input port to an output port before the full data payload arrives, thereby allowing output port load metrics to reflect future loads, thereby improving adaptive routing decisions made by switch 202.

[0042] Crossbar switch 210 may comprise separate, distributed crossbars routing data/data elements between input and output ports. In some embodiments, and as illustrated in FIG. 3A, there are five distributed crossbars including a request crossbar 210a, a grant crossbar 210b, credit crossbar 210c, an Ack crossbar 210e, and a data crossbar 210d between input port 220b and output port 220c.

[0043] Request crossbar 210a is used to send requests from an input to a targeted output age queue. Grant crossbar 210b is used to return a grant back to the input to satisfy a request. In particular, grant crossbar 210b returns a pointer indicating where a packet is within an input buffer. It should be noted that a grant is returned when there is space in the output for the corresponding packet. Grant crossbar 210b may also optionally return a credit for requested space in the output. It should be noted that grants are returned when there is a landing spot for a packet at the output, e.g., an output port 220c, so packets cannot be blocked (though they can face transient contention for resources).

[0044] It should be understood that in accordance with various embodiments, a credit protocol may be used to guarantee that there is a landing space for a request at the output. Accordingly, a credit crossbar 210c may be used to return credit for requested space in the output.

[0045] A data crossbar 210d is used to move granted packets from an input buffer to a targeted output buffer. An Ack crossbar 210e is used to propagate Ack packets from output ports 220c to input ports 220b. Acks are steered in accordance with a state kept in an output flow channel table.

[0046] It should be understood that data crossbar 210d moves multi-clock packets with both headers and data, while the other four crossbars (request crossbar 210a, grant crossbar 210b, credit crossbar 210c, and Ack crossbar 210e) move only single-clock packet headers. All five crossbars use the same architecture with row buses and column buses within an 8 x 4 matrix of 32 dual-port tiles (as described below).

[0047] Referring back to FIG. 2, switch 202 may have a plurality of transmit/receive ports, e.g., port 220. The plurality of ports may be structured in a tile matrix. FIG. 3B illustrates an example of such a tile matrix 300. In one embodiment, tile matrix 300 comprises 32 tiles, each comprising two ports used to implement the crossbar switching between ports, and to provide the following: a serializer/de-serializer (SERDES) interface between the core of switch 202 and external high speed serial signals for driving the signals off switch 202; a media access control (MAC) sub-layer interface to the physical coding sublayer (PCS); a PCS interface between the SERDES and the Ethernet MAC function; a link level retry (LLR) function that operates on a per packet basis and uses ordered sets to deliver initialization sequences, Acks, and Nacks; and an Ingress Transforms block for converting between different frame fabric formats. Each tile contains a crossbar switch such as crossbar switch 210 for each of the crossbars (210a-210e).

[0048] Each crossbar switch 210 has sixteen input ports 220b, one for each port in its row, and eight output ports 220c, one for each port in its column. Row buses can be driven from each source in a row to all eight crossbars in that row (one-to-all). Arbitration can be performed at the crossbar from the sixteen row buses in that row to the eight column buses in a given column. Buffering can be provided at each 16 x 8 crossbar for each of the row buses in order to absorb packets during times when there is contention for a column bus. In some embodiments, a non-jumbo packet is kept off a row bus unless there is room for the entire packet in the targeted crossbar input buffer. Due to area constraints, jumbo packets are allowed to go even if there is not sufficient space (crossbar input buffer only sized to sink a non-jumbo packet) with the row bus being blocked until the packet wins arbitration and space is freed as it is moved onto a column bus.

[0049] Column buses are driven from a given crossbar to each destination port within a column (all-to-all). Each destination may have another level of arbitration between the column buses from the four rows. With sixteen row buses driving eight crossbars, each

feeding eight column buses, there is a 4x speedup between rows and columns. Each row has identical connections with the one-to-all row bus connections for a single row shown in row buses. Each tile will have a one (request, grant, credit) or a two (data, ack) clock delay per tile depending on the crossbar. This gives a maximum seven or fourteen clock delay to get between the leftmost and rightmost columns. Credit returns routed through credit crossbar 210c may have a one clock delay per tile, and therefore, can take a maximum of seven clocks to complete transmission.

[0050] It should be noted that each column may have identical connections with the all-to-all column bus connections for a single column, and there may be a two clock delay per tile, resulting in a six clock delay to get from the top row to the bottom row. It should also be understood that both row and column buses both use the aforementioned credit-based protocol to determine when they are able to send. In the case of row buses, the source port maintains credit counts for the input buffers of the crossbars within that row. For the data crossbar, care is needed to determine when a packet is allowed to go on a row bus. If grants targeting a particular crossbar input buffer all go through a single queue, space for the packet at the head of the queue is required before starting the packet transfer. If the grants are distributed across multiple queues, in order to prevent small packets from locking out large packets, a packet transfer does not start unless there is space for an entire max sized packet in the buffer. In this way, once a packet transfer on a row bus starts, it will not stop until the entire packet has been transferred. Accordingly, crossbar input buffers are configured to be large enough to handle the maximum packet size plus additional space to cover the worst case round trip (packet send to credit return). This will not be the case for jumbo packets. To save on buffering area, the crossbar input buffers are only deep enough to handle a non-jumbo sized MTU (1500 bytes) with a jumbo packet being allowed to block a row bus while waiting to gain access to the targeted column bus.

[0051] For column buses, each crossbar maintains credit counts for the input buffers at each destination port in that column. Unlike row buses, there is no requirement that credits be available for a maximum-sized packet before starting transfer of that packet on a column bus. Individual words of the packet will move as credits become available. Therefore, the input buffer at the destination for each column bus needs to only be big enough to cover the worst case round trip latency (packet to credit).

[0052] FIG. 3C illustrates, in greater detail, an example implementation of two ports, e.g., ports 0 and 1, handled by tile 0, along with crossbar switch 210 comprising a set of row buses and column channels with per tile crossbars. In this way, every port has its own row bus, which communicates across its row, and every tile has the aforementioned 16 x 8 crossbar, which is used to do corner turns, and a set of eight column channels that feed up to the eight ports that are contained in that column. In other words, each crossbar switch 210 has sixteen row bus input buffers and eight possible destinations. For example, for data to travel from, e.g., input port 17 to output port 52, data is routed along a row bus from input port 17, traverses a local crossbar which is a 16 to 8 arbitration, and then traverses up a column channel to output port 52. In terms of the total routing through all the set of distributed crossbars, there is four times more internal bandwidth than there is external bandwidth, resulting in an ability to keep up with ingress when routing nearly any arbitrary permutation of traffic through switch 202.

[0053] A fair round-robin arbitration may be used between the sixteen sources for each destination. For the data crossbar 210d, once a source wins arbitration, it keeps control of the destination column bus until the entire packet has been sent. Each output grants a limited amount of packet payload so it is expected that contention for a given column bus should be fairly limited when larger packets are involved. Because of this, a round-robin arbitration is expected to be sufficient even with possibly large differences in packet size among requesters.

[0054] Parts of switch 202 associated with output functions generally operate on frames within the switch fabric format, and have a fabric header, even, for example, for a frame arriving and leaning on an Ethernet port within a single switch 202.

[0055] Age queue output control is responsible for accepting requests from all of the input port, e.g., input ports 220b, via request crossbar 210a, buffering the requests, arbitrating between them by traffic class using a traffic shaper, and passing the requests to the OFCT 216 to be granted via grant crossbar 210b. Age queue buffering is managed to allow each input to have enough space to flow while also allowing an input with multiple flows targeting a given output to take more space. In particular, an age queue space is managed by output control. The age queue/output control may also be responsible for managing access to the link either using credit-based flow control for a connected input buffer or pause-based flow control for non-fabric links. When a packet is released by the age queue, it is committed to being put on the link. Additionally the age queue has a path allowing packets initiated on a given port e.g., one of input ports 220b (such as maintenance or reduction packets), to arbitrate for resources on the given port.

[0056] Requests come into the output control block via a column bus from each row of matrix 30. Each column bus feeds an independent FIFO (e.g., first-in-first-out shift register or buffer) with space in the FIFO managed via credits. The FIFOs may be sized (24 deep) to cover a round-trip plus additional space to allow requests to be moved out of the crossbars 210a-210e and prevent head-of-line blocking. Prior to writing into a FIFO, a request may be checked for a valid error correcting code (ECC). If the ECC check has either a multi bit error (MBE) or a single bit error (SBE) in the destination field (i.e. it has been routed to the wrong port), the request is considered to be an invalid request, and is discarded with an error being flagged.

[0057] Least recently used (LRU) arbitration may be performed between column bus FIFOs to choose which FIFO gets forwarded to age queue management. As requests are

removed from each FIFO, credits are returned to the corresponding crossbar. The row with which an incoming column bus corresponds can be dependent both on where in the matrix the tile is located, and as well as which half of the tile the block is in.

[0058] The output buffer (OBUF) makes requests to the output control block for sending reduction and maintenance packets across a link. These requests may be given the highest priority. A FIFO with 8 locations can be used to buffer these reduction/maintenance packet requests while they wait for resources. Reduction packets need not use flow channels, and maintenance packets may use loopback to create a flow so that checking for flow channel availability or flowing through the OFCT to create a grant is not needed. Reduction and maintenance packets also need not use any space in the output buffer so that no check of space is required. Rather, a check for the link partner input buffer may be performed. If allowed, a shaping queue (SQ) or virtual channel (VC) can be granted, blocking any grants from the age queue path from being granted during that cycle.

[0059] The size of the next request to be processed from the output buffer is checked against max_frame_size. If it exceeds this setting, the request is not processed and an error flag is set. This will result in the output buffer request path being blocked until a warm reset is performed. The error flag will stay set until the reset is done. The condition can also be released by increasing the setting of max_frame_size to a value above the size of the stuck output buffer request. The size used in the comparison may be the size indicated in the output buffer request (which may include a 4-byte frame checksum (FCS) used on the wire).

[0060] Each input may be given the same fixed allocation of age queue space. This age queue space is large enough to reserve a location for each SQ/VC with enough additional space to cover a request/credit round-trip. It is up to the input to manage the space it is given across its SQs/VCs. This allocation *(fixed_al/oc)* is programmable via a control and status register (CSR) in each input queue (INQ), and can be, e.g., in the range of 64-96 locations. The remaining age queue space (8K- *64*fixed_al/oc)* may be shared space that is available to all

inputs. The shared space can be managed by the output with it moving incoming requests from static to shared space as they arrive if there is room in the shared space, subject to per-input limits. When moving a request to the shared space, a credit is returned, e.g., immediately, via credit crossbar 210c, with the request marked in the age queue as being in the shared space.

[0061] When a request is granted, if it is marked as using the shared space, the shared space is credited. If it is not marked as using shared space, the request is considered to have used the static space, and a credit is returned to the input with the grant.

[0062] Due to conflicts in credit crossbar 210c, it is possible that credits may not be sent every clock period. Accordingly, a FIFO provides buffering for these transient disruptions. Space in this FIFO is required before taking a request from the request crossbar. A FIFO with a depth of 32 locations can be used to limit the chances of it ever backing up into request crossbar 210a. The shared space may have limits for how much space any input (from an input port 220b) can take. These limits can be set as a percentage of the available space. For instance, if the limit is set to 50%, if one input port is active, it has access to 50% of the buffer space, with two active input ports, each gets 37.5% ((space_used_by_l + pace_left*.5)/2 = (50%+50%*.5)/2), with three active input ports, each gets 29.2% ((space_used_by_2 + space_left*.5)/3 = (75%+25%*.5)/3), and so on. Additionally, the total space used by the active input ports can be limited to the given total (50%, 75%, 87.5%). Thus, the space allocated to each of input port 220b may vary dynamically by how many inputs ports are currently active. The addition of an active input port causes other active inputs ports to give up their space which is then taken by the new input.

[0063] Given that division is not something easily done in hardware, the aforementioned age queue credit management function can be implemented as a lookup table 310 with 64 entries 312. The number of inputs currently active in the age queues 320 indexes 315 the lookup table 310. The values 314 in the lookup table 310 reflect the limit of

the number of shared space locations any input can take along with the total space they can consume as a whole. Thus, it is up to software to program the values 314 in the lookup table 310 according to how much total shared space there is and what percentage each input port is allowed to take. As more input ports 220b become active, each input port 220b is allowed less space, and the total space available increases. Incoming requests from input ports 220b that are above this limit, or in total, exceed the total space limit, are not allowed to take more shared space. In order to track the number of active input ports 220b in the age queues, a set of 64 counters 316 (one for each input port) is used. These count up when a request is put in the age queues 320 and count down as they are taken out (i.e., granted). A count of the number of non-zero counts 319 is used as an index into the lookup table 310. In addition, in order to manage the shared space, an additional set of 64 counters 318 may be used to track the current usage of the shared space by each input. There may also be a single counter 334 that can be used to track overall shared space usage. These counters are compared against the current quotas to determine if a request is allowed to use the shared space or not. Counters 316, 318 can be, e.g., 13 bits wide, to provide sufficient coverage of the maximum value of an object that may be somewhat less than 8K.

[0064] Age queues 320 may use a single storage RAM 321 that has 8K locations in it. These locations can be dynamically allocated to 32 separate queues (one for each SQ/VC) with each consisting of a linked-list of locations within the storage RAM 321. This gives each SQ/VC the ability to take more space as needed.

[0065] An age queue 320 can be created with a front pointer 322 pointing to the front of the queue, and a next pointer 324 for each location pointing the next item in the queue. The last location in the queue may be indicated by a back pointer 326. Items are taken from the front of the queue and inserted at the back of the queue. In addition to the above data structures, each queue has a FIFO 328 of entries at its head. These FIFOs 328 may ensure that a queue can sustain a request every clock with a multi-clock read access time from the request

RAM 321. When a new request arrives, if the head FIFO 328 for that queue is not full, it bypasses the request RAM 321, and can be written directly into the head FIFO 328. Once requests for a given age queue are being written to the request RAM 321, subsequent requests are also written to the request RAM 321 to maintain order. The bypass path can be used again once there are no more requests for that age queue in the request RAM 321, and there is room are the head FIFO 328. When a request is read from a head FIFO 328, and there are corresponding requests queued in the request RAM 321, a dequeue is initiated. One head FIFO 328 may be read at a time, such that a single dequeue operation can be initiated each clock period. Logic may be included to handle the various race conditions between an ongoing or imminent enqueue operation and a head FIFO 328 being read.

[0066] The aforementioned ECC protection used in the age queue RAM 321 can be extended to the FIFOs 328 to protect the data path flops. The resulting structure may include 8K flops (32 queues x 5 deep x SQ-bits wide). When generating the ECC, the age queue number can be included in the calculation (but not stored) as an extra check of the free list management. When the ECC is checked, the request can be considered to be in error if there is an MBE or there is an SBE in the queue number bits.

[0067] A free list RAM can be a simple FIFO which is initialized with pointers to all 8K entries whenever a reset is performed. A count can be maintained to keep track of how may entries are valid within the free list. As entries are taken, they are popped off the front of the FIFO and used. As entries are returned, they are pushed onto the back of the FIFO. Some number of entries, e.g., three entries,) at the head of the free list can be kept in flops so they are available for quick access. As with the head FIFOs for the age queues, ECC is carried through the flops to provide protection. The resulting structure may have minimal flops (57 = 3 deep x 19-bits wide).

[0068] In order to support full performance for small packets, age queues support both an enqueue and a dequeue every clock period. The operations across the data

structures for an enqueue operation are discussed below, and can differ depending on whether the queue being written is empty or not.

[0069] In some cases, a simultaneous enqueue and dequeue to a specific queue is easily handled as they are using and updating separate fields. Some specialized scenarios may arise, e.g., when a dequeue operation empties the age queue. In order to handle this scenario, a dequeue occurs first logically, followed by an enqueue operation. Accordingly, an empty flag is seen as being set when the queue is emptied by the dequeue operation, and then cleared due to the enqueue operation.

[0070] The arbitration alluded to above can be performed among requests that are permitted to be granted subject to input buffer management, output buffer management, and flow channel quotas. Arbitration can also be halted if there are no credits for the OFCT input FIFO. In some embodiments, arbitration may be performed at two levels. First, traffic shaping arbitration can be used to arbitrate between the SQs. A Deficit Round-robin arbitration can be used to arbitrate between VCs within a given SQ. Traffic shaping arbitration may use a series of token buckets to control the bandwidth of each SQ as follows: eight leaf buckets, one for each SQ; four branch buckets; and a single head bucket.

[0071] Arbitration can be divided into three groups with a first group having the highest priority, followed by a second group, which in turn is followed by a third group. For the first and second groups, arbitration may be handled in the same way among eligible SQs. A x8 round-robin arbitration can be performed between the SQs for each of the eight priority levels (eight parallel round-robin arbitrations). A fixed arbitration can be performed between priority levels. For example, group 3 arbitration has no priorities, and therefore is simply a single x8 round-robin arbitration.

[0072] For arbitration in the first group, the priority for each comes from the setting in the leaf buckets. For arbitration in the second group, priority comes from the setting in the branches of the leaf buckets. In all cases, the buckets which are checked to be eligible for

that group, are also the buckets from which packet size tokens are obtained if that request wins arbitration.

[0073] Regarding age queue 320 selection, packets can be classified in order to select the SQ to which their request is forwarded. This allows traffic associated with an application to be shaped differently from traffic originating from a different application or a different traffic class. This can be useful on the edge ports which connect to a NIC in that the applications will have been configured to use a share of the resources on the node, and similarly will be granted a proportion of the network bandwidth. In accordance with one embodiment, this classification is performed by classifying the packets into a traffic class identifier (FTAG), e.g., a 4-bit code that is part of the fabric frame header, and a VLAN ID (VNI) as the packet ingresses into the fabric. The FTAG and VNI may then be used as the packet egresses the fabric to select the shaping queue.

[0074] A configuration register can be used to map FTAGs to SQs. This configuration matches the corresponding configuration in the in queue. When the output buffer requests or returns link partner credits, it converts a given FTAG to an SQ. For packet injection, the FTAG is found in R_TF _OBUF _CFG_PFG_TX_CTRL. For test generation, the FTAG is found in the test control register. When the reduction engine (RED) requests a credit return, the FTAG is found in *ret_cdtJtag.* When a reduction frame is removed from the output stream and link partner credits need to be returned, the FTAG is found in the frame header.

[0075] Regarding the SQs discussed herein, each age queue 320 may have 32 SQs 330 that are addressed by {SQ, VC}. The 3-bit SQ 330 can be considered a shaping function, and the VC selects one of four queues within that shaping function. For Ethernet egress (edge) ports, the VC is not needed for deadlock avoidance. Accordingly, all 32 SQs 330 can be available. In such a scenario, the SQ 330 can be selected by adding the SQ base from R_TF _OBUF _CFG_FTAG_SQ_MAP to the lower bits of the VNI. The 5-bit sum defines the {SQ,VC} to send to the age queue. It should be noted that when injecting frames on an egress port, a

VNI is not available, and therefore, an SQ base can be directly used. For fabric links, the SQ 330 is taken from the upper three bits of the SQ base. The VC can be taken from the frame header when returning credits for reduction frames, or from the appropriate control CSR (R_TF _OBUF _CFG_TEST_CTRL or R_TF _OBUF _CFG_PFG_TX_CTRL) when injecting frames.

[0076] A link partner input buffer management can depend on the type of device to which the link is attached. Devices such as switch 202 may use credit-based flow control where each credit represents a cell of storage in the input buffer. Other devices may use standard Ethernet pause or priority pause-based flow control. Requests which are marked to terminate locally *(lac term* set) need not consider link partner input buffer flow control and need not update any associated counters. Link partner space need not be considered when the link is in the draining state.

[0077] For credit-based flow control, the link partner input buffer can be divided into eight buffer classes. Each SQ 330 can be assigned to one of these 8 buffer classes. Credits are maintained for each of the buffer classes with each credit representing 32 bytes of storage in the link partner input buffer. In order to allow credit-based flow control to work with various devices (switch, enhanced NIC), each of which may have different cell sizes, the cell size is a programmable value in units of 32 bytes.

[0078] There may be two sets of VCs with each SQ 330 assigned to one set. A maximum frame size worth of space can be reserved for each VC, and each VC set can have a different maximum frame size. The remainder of the link partner input buffer is shared dynamic space usable by any SQ/VC, subject to per VC and buffer class limits.

[0079] The size that comes with the request represents the size of the packet on the wire which includes a 4-byte FCS. This gets converted to an internal 2-byte FCS at the link partner before writing the packet to the link partner input buffer so the crediting needs to account for this difference, which can be a factor at the boundary of the cell size. For instance, for a 96 byte cell, a size that is 97 or 98 will take a single cell. In order to know when this

happens, the request includes a correction term which is calculated as: *req.len_correct*= (byte_len % 16) == 1 or 2.

[0080] Further validation of this term is required to convert it to whatever the cell size boundary may be. It will be valid when the length just exceeds the cell size. With this, the validated *fen _correct* term can be determined by: *len_correct* = (((16-byte size)% (2*32-byte cell size))== 1) & *req. len correct*

[0081] An example of how these values work for a few cell and packet sizes is illustrated in the table below:

Length Correct Calculation

Size (bytes)	Req len_correct	Size (16B units)	Cell Size (32B units)	len_correct Modulo result	len_correct	Credit Taken
64	0	4	2	0	0	2
65	1	5	2	1	1	2
66	1	5	2	1	1	2
67	0	5	2	1	0	3
96	0	6	3	0	0	3
97	1	7	3	1	1	3
9B	1	7	3	1	1	3
99	0	7	3	1	0	4
128	0	8	4	0	0	4
129	1	9	4	1	I	4
130	1	9	4	1	1	4
131	0	9	4	1	0	5

[0082] The size that comes with the request uses 8-byte units and the link partner input buffer cell size is a multiple of 32 bytes (32 * y where y = cell size from CSR). First, the 8-byte size is converted to a 16-byte size (ROUNDUP((8-byte size)/2)). Also, the cell size is converted to 16 byte units (2*y). Mathematically, the number of cells a request will use can be calculated by: ROUNDDN(((l6-byte size)+ 2*y - 1 - *len_correct)/(2*y))* = # of cells

[0083] While a divide operation is possible in hardware, due to timing reasons, a divide operation cannot be done in the critical path of the arbitration. Instead, an alternate credit management is used. That is, credits are maintained in units of 32 bytes. When a request wins arbitration, the number of credits taken is adjusted by the maximum error term (2 * y-1) using the calculation: ROUNDDN(((l6-byte size)+ 2*y - 1)/2) = Maximum 32 byte credits needed. Because this calculation overestimates the credit required for the packet, on the following clock, a modulo operation (X = (16-byte size) MOD 2*y, y = 32-byte cell size from CSR) van be performed to determine what the actual remainder is. This value along with the len _ correct term are used to adjust the credit counter. The formula used to create the adjustment value *(adf_val)* for Xis: If (X == 0) *adj_val* = y – 1 else if (X == 1 and *fen _correct) adj_val* = y else *adj_val* = ROUNDDN((X-1)/2)

[0084] The table below illustrates a request credit example for 96 byte cells showing the values used across several packet lengths for the 96 byte cells of the switch input buffer (y = 3).

Request Credit Example For 96-byte Cells

Packet Size (bytes)	Packet Size (16-byte units)	Credit Taken	Modulo Result	len_correct	adj_val	Corrected Credit Taken
48	3	4	3	0	1	3
64	4	4	4	0	1	3
80	5	5	5	0	2	3
96	6	5	0	0	2	3
97	7	6	1	1	3	3
98	7	6	1	1	3	3
99	7	6	1	0	0	6
128	8	6	2	0	0	6

[0085] If a request is filtered before being forwarded to the link partner input buffer, the output buffer logic returns the SQ and VC so they can be used to return the credits to the

appropriate credit counters. No size is required since the packet size is always the same, the length of a reduction frame (69-byte or 16-byte size= 5).

[0086] The local (master) side of the link maintains a count of the number of packets sent from each VC across both sets (8 total), a count of amount of packet (in 3 2-byte quantities) sent to each VC (4), and a count of the amount of packet (in 32-byte quantities) sent for each buffer class(8). The link partner (slave) side of the link maintains the same set of counts with them being sent over the link periodically. The difference between the master and slave counts is a count of the number of packets in the link partner input buffer from each VC across both sets and a count of the amount of space (in 32-byte quantities) currently occupied by each VC and each buffer class. A count is also maintained of the total amount of space used across all packets. A summary of the counters is as follows: master_vcx_cnt[4]/slave_vcx_cnt[4] - master and slave counts of the number of packets sent to each VC in set X; master_vcy_cnt[4]/slave_vcy_cnt[4] - master and slave counts of the number of packets sent to each VC in set Y; master_bc_cnt[8]/slave_bc_cnt[8] - master and slave counts of the amount of space occupied by each buffer class in units of 32-bytes; master_vc_cnt[4]/slave_vc_cnt[4] - master and slave counts of the amount of space occupied by each VC in units of32-bytes; master- tot- cnt/slave- tot - cnt - master and slave counts of the total amount of space occupied in units of32-bytes.

[0087] All counters are set to zero on a warm reset. They are also forced to zero when the link is in the draining state or when the DBG_RESET CSR bit to clear their state is set. The output buffer filter will steer a reduction packet to something other than the path to the link partner input buffer. In this case, a signal can be returned along with the SQ and VC of the packet. Again, the length is not required as the size of these packets is fixed. This information is used to adjust the appropriate master credit counts.

[0088] A request is allowed to participate in arbitration if either its VC count is 0 (indicating its one statically assigned slot is available) or there is space for a max sized frame

in the dynamic space (subject to the targeted buffer class and VC limits). There can be a single programmable value for max frame size which is used across all VCs and SQs. The request validation for input buffer space can be addressed using credit-based flow control.

[0089] Credit-based flow control can be used to divide a dynamic space in two ways, each independent of each other: first, based on a limit of how much dynamic space each of the four VCs can take; and second, based on a limit to how much dynamic space each of the eight buffer classes can take. In both cases, the limits are set as a percentage of the available space. For a given packet, space should be made available in both its targeted VC and buffer class. For instance, if each space has its limit set to 50%, if one is active, it has access to 50% of the buffer space, with two active, each space gets 37.5% ((50+50*.5)/2), with three active, each space gets 29.2% ((75+25*.5)/3), and so on. Also, the total space used by those spaces that are active can be limited to the given total (50%, 75%, 87.5%). Accordingly, the space allocated to each varies dynamically by how many are currently active. When an additional one goes active it causes others that are active to give up some of their space which is then taken by the new one.

[0090] Like the division function discussed above, this function is implemented as a lookup table. For the VC space in this example, there are 16 entries with each entry specifying the space available to each VC along with the total space available across all VCs. For the buffer classes, there may be 256 entries with each entry specifying the space available to each buffer class along with the total space available across all buffer classes. Space for each is expressed in 2048-byte units. The depth of each table is sufficient to cover all combinations of active members (VCs or buffer classes), with each being able to have an independent setting for their percentages. With this, it is up to software to program the values in the table according to how much total dynamic space there is and what percentage each is allowed to take across all possible combinations. As more become active, each is allowed less space and

the total available increases. Requests for spaces that are above this limit, or in total above the total limit, are not allowed to take more dynamic space.

[0091] A VC or buffer class is considered active either if it has a request in an age queue, or if it has outstanding credits for link partner input buffer space. As an example, consider there are only 4 spaces (16 entry table) with percentages set as SPACE0(50%), SPACE1(40%), SPACE2(30%), SPACE3(10%), and a total dynamic space of 16KB. This results in the values, in quantities of 16-bytes presented in the buffer space example table below.

Buffer Space Example

Index	SPACE3	SPACE2	SPACE1	SPACE0	Total
0	N/A	N/A	N/A	N/A	N/A
1	N/A	N/A	N/A	512	512
2	N/A	N/A	410	N/A	410
3	N/A	N/A	319	398	717
4	N/A	307	N/A	N/A	307
5	N/A	250	N/A	416	666
6	N/A	255	339	N/A	594
7	N/A	202	270	337	809
8	102	N/A	N/A	N/A	102
9	94	N/A	N/A	469	563
10	94	N/A	377	N/A	471
11	75	N/A	299	374	748
12	95	284	N/A	N/A	379
13	78	234	N/A	389	701
14	80	239	319	N/A	638
15	79	236	315	394	1024

[0092] As an example, the values in the row for index 7 are calculated as: Total%= 0.5 + (1-0.5)*0.4 + (1-0.5-(1-0.5)*0.4)*0.3 = 0.79; SPACEO = (0.5/(0.5+0.4+0.3))*0.79*1024 = 337; SPACEl = (0.4/(0.5+0.4+0.3))*0.79*1024 = 270; SPACE2 = (0.3/(0.5+0.4+0.3))*0.79*1024 = 202; Total= 337 + 270 + 202 = 809

[0093] As noted above, and referring back to FIG. 2, switches, such as switch 202 may be used to create a switch fabric, where the switch ports 220 may be configured to operate as either edge ports or fabric ports. As also noted above, switch 202 can support various network topologies including but not limited to, e.g., dragonfly and fat-tree topologies. Networks can be thought of as comprising one or more slices, each having the same overall topology, although slices may differ with respect to how each is populated. Nodes are connected to one or more ports on each slice. When a network has multiple slices, and a node is connected to more than one slice, the node is assumed to be connected at the same location in each slice.

[0094] Routing in the switch fabric may be controlled by a fabric routing function (FRF) implemented in switch 202. An example FRF component 400 is illustrated in FIGS. 4A and 4B. It should be understood that a separate instance of FRF component 400 may be implemented within input logic for each port of switch 202. Routing decisions made by FRF component 400 can be applied to those frames that are not already part of an established flow. It should be noted that FRF component 400 does not necessarily know whether or not a particular frame is associated with a flow, but rather makes an independent forwarding decision for each fame presented at an input port. FRF component 400 may comprise filters, tables, circuitry, and/or logic such as selection circuitry/logic to effectuate routing of data throughout a switch fabric as described herein. As illustrated, FRF component 400 includes at least: a minimal ports selection component 402(which includes a minimal tables component 402A), various ports filters (permitted ports filters, operational ports filters, busy ports filters); a preferred ports discrimination component 402B; pseudo-random down selection components/logic 402C; exception tables 404 (including an exception list table 404A); operational ports component 406 that includes a global fault table 406A; and a routing algorithm table 408. As illustrated in FIG. 4B, FRF component 400 may further comprise: a non-minimal ports selection component 410 that includes local non-minimal selection

component 410A and global non-minimal selection component 410B); and output logic component 412, which includes an adaptive selection component or logic 412A. FRF component 400 includes other components, and are described herein.

[0095] In particular, FRF component 400 determines a preferred port to forward each frame presented at the input port based on: a received frame's destination fabric address (DFA); the frame's current routing state (where the frame is along its path, and the path(s) it took to reach its current routing state); the switch fabric routing algorithm and configuration; and load metrics associated with the possible output ports to using busy ports filters.

[0096] FRF component 400 may include a routing algorithm table 408 that may be embodied as a software configurable table that determines valid choices based on the frame's current routing state. Valid choices are decisions such as whether a local minimal, global minimal, local non-minimal, or global non-minimal path is allowed to be chosen for the frame's next hop. The routing state includes information such as the VC the frame was received on, and whether it is in the source, the destination, or an intermediate group. The routing algorithm table 408, along with the adaptive selection function or logic 412A (described below), also determines the VC to be used for the frame's next hop.

[0097] Frame routing with unicast DFAs will be described as an example. However, it should be noted that the DFA of the routing request can either be in unicast or multicast format. The unicast format can include a 9-bit global ID field (global_id), a 5-bit switch ID field (switch_id), and a 6-bit endpoint ID field (endpoint_id). The global ID can uniquely identify a group within the network. Specifically, it identifies the final group to which the frame must be delivered. The switch ID uniquely identifies a switch within the group identified by global ID. The endpoint ID field, together with the global ID and switch ID identify the endpoint, connected to the edge of the network fabric, to which the frame is to be delivered. This field is mapped to a port or set of ports on the switch identified by global ID and switch ID.

[0098] The multicast format includes a 13-bit multicast ID field (multicast_id). This field is mapped by FRF component 400 to a set of ports on the current switch to which the frame is to be forwarded.

[0099] From this information, FRF component 400 determines an updated routing state for the frame, which is then carried within the frame. For example, to effectuate routing in a dragonfly topology, a frame's current state may be gleaned from the frame's VC (discussed above). Based on algorithmic switch fabric routing rules specified for the switch fabric (the selection of which is described below), FRF component 400 determines a particular VC to be used for the frame's next hop to avoid any deadlocks. Additional routing state information can be provided depending on where the frame is along its path, e.g., whether the frame is in its source group, in an intermediate group, or in its destination group. It should be noted that FRF component 400 performs port filtering (described in greater detail below) using permitted ports filter, operational ports filter, busy ports filters, etc. to determine if a preferred port to which a frame is to be forwarded is currently faulty, busy, absent, etc.

[00100] Switch 202 distributes load information between switches. The FRF component 400 receives the load measurement of and from its associated output port. The FRF component 400 receives summary load information from its associated input port for a neighboring switch. Each FRF component 400 exchanges load information with all other FRF instances within the same switch. FRF component 400 provides summary load information to its associated output port for delivery to a neighboring switch. Through the load distribution mechanism, each FRF component 400 learns the load measured at each output port of its switch. As well, each FRF learns the summary load information for all neighboring switches.

[00101] It should be noted that FRF component 400 can support frame multicasting. When a multicast DFA is received, FRF component 400 determines a set of ports to which the frame associated with the multicast DFA should be forwarded. The set of ports can be determined by accessing a lookup table that maps software-configured multicast fabric

addresses to output ports. This can help avoid problems associated with duplicate multicast frame copies.

[00102] As noted, multicast traffic is statically routed based on the configuration of the multicast forwarding tables. More specifically, the multicast table identifies the ports of the current switch to which copies of the multicast packet should be forwarded. This implicitly determines the number of copies generated at that point, as well, because for each multicast packet received at a port, only one copy is forwarded to each other port identified by the multicast table as requiring a copy. To prevent duplicate delivery of multicast packets, multicast forwarding filters conditionally prune copies of the multicast packet depending on source group at which the multicast packet originated.

[00103] FIG. 5 illustrates an example route selection process involving down-selection of candidate ports and adaptive route selection based on load. FRF component 400 considers three categories of candidate ports to which a frame may be forwarded: preferred minimal path candidate ports 502; non-preferred minimal path candidate ports 504; and non-minimal path candidate ports 506. Depending on where a frame may be along its path, non-minimal path candidate ports are either global non-minimal path candidate ports or local non-minimal path candidate ports.

[00104] Filtering may be applied to the three categories of candidate ports, e.g., operational port filtering, useable port filtering, and busy port filtering. Port filtering as applied herein can be used to reduce the set of valid ports considered as path candidate ports by identifying and removing absent and/or faulty ports from consideration.

[00105] Operational port filtering (or non-operational port filtering) can refer to the removal of non-operational ports from sets of ports being considered as candidates for routing, e.g., preferred minimal path candidate ports 502, non-preferred minimal path candidate ports 504, and non-minimal path candidate ports 506. That is, switch 202 may identify certain ports as being non-operational. These non-operational ports may be reported

in a non-operational port mask. It should be noted that in some embodiments, software can force additional ports of switch 202 to be considered as non-operational using a non-operational port CSR, e.g., when a port(s) is to become disconnected as a result of planned maintenance.

[00106] Usable (or unusable) port filtering may involve filtering out candidate ports for consideration that would normally have been acceptable, but due to faults within network 100, for example, the candidate ports have become unacceptable/unusable for reaching one or more destination switches, destination groups (of switches), etc., but remain acceptable or usable for reaching one or more other destination switches. In some embodiments, global fault table 406A can be used to block global minimal path port candidates and global non-minimal path port candidates depending on a destination group of the frame being routed. For example, candidate ports that lead to an intermediate group (of switches) without connectivity to a particular destination group (of switches) can be excluded from consideration when routing frames to that destination group, although the same candidate ports may not necessarily be blocked out for other destination groups. The global fault table 406A can be determined or indexed by the global_id field of the frame's DFA.

[00107] In some embodiments, an exception list maintained by exception list table 404A may be used to conditionally exclude port candidates depending on the destination group or switch to which the frame is being routed. It should be noted that that exception list table 440A may be used to identify preferred global minimum path ports. Accordingly, use of the exception list table 404A to exclude port candidates is done when it is not being used to identify preferred global minimum path ports.

[00108] It should be noted that knowledge regarding which ports are busy in a neighboring switch can be used to determine if the ports that connect to a neighboring switch are poor candidates to receive a forwarded frame based on whether the neighboring switch will subsequently need to forward the frame to a port that is already busy. For example, when

considering candidate ports for use in global minimal routing, the ports connected to a neighboring switch are poor candidates if the neighboring switch's global ports that connect to the frame's destination group are all busy, global ports referring to ports that connect to other switches belonging to a different group of switches. Similarly, when in the destination group and considering candidate ports for use in local nonminimal routing, the ports connected to a neighboring switch are poor candidates if the neighboring switch's local ports that connect to the frame's destination switch are all busy.

[00109] Accordingly, busy port filtering can be performed by FRF component 400 by using busy port masks to remove heavily loaded ports from being considered candidate ports. It should be noted that in some embodiments, heavily loaded ports are removed from consideration when other candidate ports that are not heavily loaded exist. Otherwise, when non-heavily loaded ports do not exist, busy port filtering will not remove the heavily loaded ports from consideration. FRF component 400 maintains four masks of busy ports, that is, ports whose load exceeds a software-defined threshold: local switch busy port mask; global non-minimal busy global port mask; global non-minimal busy local port mask; remote switch busy port mask. Information from these masks is communicated between switches to populate the remote switch.

[00110] A local switch busy port mask can be applied to minimal path candidate ports as well as to local non-minimal path candidate ports. The FRF generates a 64-bit ls_busy_port_mask by comparing each port's local_load to a software defined threshold. Ports with loads higher than this threshold are marked as busy in this mask.

[00111] A global non-minimal busy port global mask can be applied to global ports of global non-minimal path candidate ports. The FRF generates a 64-bit gnmbgp_mask by comparing each port's gnmgp_load to a software defined threshold. Ports with loads higher than this threshold are marked as busy in this mask.

[00112] A global non-minimal busy local port mask can be applied to local ports of global non-minimal path candidate ports. The FRF generates a 64-bit gnmblp_mask by comparing each port's gnmlp_load to a software defined threshold. Ports with loads higher than this threshold are marked as busy in this mask.

[00113] A destination group dependent busy-port mask, obtained from a remote switch busy global port table can be applied to global minimal path candidate ports. Correspondingly, when a frame is being routed in its destination group, a destination switch-dependent busy-port mask, obtained from a remote switch busy local port table can be applied to local non-minimal path candidate ports.

[00114] Upon applying the aforementioned filtering or down-selection stage, a set of surviving path candidate ports 508 can result. That is, a reduced number of candidate ports can be determined after removing non-operational and unusable port candidates, heavily loaded, a set of path candidate ports remains. In some embodiments, a pseudo-random selection process is used to further reduce the number of surviving path candidate ports 508 to a determined threshold number of ports associated with each category of candidate ports (preferred minimal path candidate ports, non-preferred minimal path candidate ports, and non-minimal path candidate ports). In some embodiments, that threshold number of candidate ports may be four candidate ports per category. If the minimum threshold number of candidate ports is not met, no candidate ports from that category are removed from consideration.

[00115] In some embodiments, this pseudo-random selection (or down selection) of candidate ports can be weighted. That is, weights, e.g., weights between 0 and 15, can be assigned to each port per the CSR configuration. This weighting can be used to influence the probability with which individual candidate ports are chosen such that higher-weighted ports have a great chance of being chosen. For example, a weight of 15 results in a port having 15 times greater likelihood of being selected in the pseudo-random selection process. In some

embodiments, candidate ports may be filtered into four groups (GW1, GW2, GW4, GW8) based on their assigned weights, where a candidate port can belong to multiple groups depending on the assigned weight (e.g., a candidate port with weight 1 belongs only a one weight group, while a candidate port with weight 5 belongs to two groups (GW1 and GW4, i.e., 1 + 4 = 5), and a candidate port with weight 15 belongs to all four groups (Gw1, GW2, GW4, GW8, i.e., 1 + 2 + 4 + 8 = 15). The number of candidate ports in each group can be determined (nW1, nW2, nW4, nW8), and pseudo-random selection is applied to each group to select one candidate port from each group (cW1, cW2, cW4, cW8). The weight of each group can be computed, along with their total weight: wW1 = nW1; wW2 = 2*nW2; wW4 = 4*nW4; wW8 = 8*nws; wtotal= wW1+wW2+wW4+wW8. A fifth pseudo-random selection can be performed to choose a number j in the range O ... Wtotal - 1. One of the candidates, cW1, cW2, cW4, cW8 is chosen as the down selected candidate port based on the value of j as follows: If j < wW1, choose cW 1; else If j < wW1 + wW2, choose cW2; else If j < wW1 + wW2 + wW4, choose cW4; else choose cW8.

[00116] The number of ports in each group can vary from request to request due to changing operational status and load, and may also be dependent on the configuration of the global fault table 406A. It may also vary depending on the type of port at which the routing request is being performed, i.e., edge port versus local port, for example. On the assumption that operational status and load do not change too quickly, and that the configuration of global fault table 406A should not vary greatly for different global_id values, each pseudo-random generator's previously generated value is simply used as the offset for biasing its next value. If an offset value is out of range (> n - 1), it is brought within range through truncation of upper bits. mgnm = 4 candidate ports are produced by the pseudo-random down selection process. Each candidate can be produced through a separate copy of the weighted pseudo-random down selection logic 410C, described above.

[00117] It should be noted that the same candidate port may be chosen by more than one of the instances/iterations of the weighted pseudo-random down selection logic 410C, in effect reducing the number of candidate ports that are chosen. The probability of the same candidate port being chosen by more than one of the mgnm = 4 global non-minimal, weighted pseudo-random selectors decreases with increasing numbers of candidate ports to choose from. In the context of a dragonfly topology, for example, and a network with full global bandwidth, at an edge port in the source group there are potentially about 48 possible global non-minimal candidate ports: 16 global ports and 32 local ports. If a local hop has been taken, the next hop is a global hop, thereby reducing the number of candidate ports to about 16. However, if the network is tapered such that it only supports one quarter of full global bandwidth, there may only be 4 global candidates to choose from after a local hop has been taken. The probability of selecting four unique global non-minimal candidate ports shows the probability of four unique candidate ports being chosen for varying numbers of possible candidates to choose from. The probability of selecting n unique global non-minimal candidate ports shows the probability of varying numbers of the four selected candidates being unique when there are only 16, 8, or 4 candidate ports to choose from.

[00118] FRF component 400 can use received remote switch busy port masks to generate the aforementioned Remote Switch Busy Global Port Table of busy port masks that identify ports connected to neighboring switches that should be avoided as global minimal path candidates based on the destination group to which the frame is being routed. Similarly, the received remote switch busy port masks can also be used to generate the aforementioned remote switch busy local port table of busy port masks that identify ports connected to neighboring switches that should be avoided as local non-minimal path candidates, when routing in the destination group, based on the destination switch to which the frame is being routed.

[00119] The rs_busy_port_masks are used in assessing the suitability of neighboring switches (whether ports of neighboring switches are busy or quiet) for reaching specific destination groups via global minimal paths and specific destination switches via local nonminimal paths or via local minimal paths. Each FRF instance corresponding to a local port or a global port can be configured to generate a 64-bit rs_busy_port_mask. The generated mask is delivered to the partner switch connected to that port. Similarly, the partner switch can also generate and return an rs_busy_port_mask.

[00120] Each FRF instance communicates the rs_busy_port_mask, that it received from its partner switch, to all other FRF instances in the switch using the port status ring (which connects the tiles of a switch and communicates status and load information amongst the ports on the switch). Each FRF instance captures all rs_busy_port_masks such that all FRF instances learn the remote busy port status being provided by all neighboring switches. Each FRF instance uses the rs_busy_port_masks that it receives to generate the busy port tables described in the Remote Switch Busy Global Port (RSBGP) Table and Remote Switch Busy Local Port (RSBLP) Table.

[00121] Generation of the rs_busy_port_mask is a two-step process. The first step is to compare each port's local_load to a software configurable generating an intermediate mask of all ports that are individually busy. This intermediate mask is formed as the status of each port is received from the port status ring interface. Ports that are classified as non-operational are also recorded as busy in the intermediate mask. The second step takes link bundling into account such that a port is only marked as busy in the rs_busy_port_mask if it and all other ports, that are part of the same bundle, are marked as busy in the intermediate mask. Either all ports that are members of the same bundle are marked as busy in the rs_busy_port_mask, or none are. Global ports that are part of the same bundle all connect to the same remote group. Local ports that are part of the same bundle all connect to the same remote switch within the current group.

[00122] As the rs_busy_port_masks are used to determine whether the switch that generated the mask is a good candidate for routing a frame to another group or to another switch in the current group, bundling is used to provide a consistent view of the generating switch's suitability when the busy status across its links, that connect to the destination group or to the destination switch in the current group, is inconsistent. The rationale for the treatment of bundling described here is that the switch generating the rs_busy_port_mask remains a candidate for reaching the destination group, or the destination switch in the current group as long as it has at least one link to the destination group or switch that is not busy; adaptive routing at the switch that generated the rs_busy port_mask should direct the frame to the non-busy link.

[00123] Ports must be included in either the bundled ports mask CSR or the unbundled ports mask CSR (both of which are part of the static description of the wiring) in order for them to be marked as busy in the rs_busy port_mask. The second step is performed as each frame is received from the port status ring. The bundled port masks are scanned to identify the bundles and the ports that they contain. In addition, the unbundled port mask is consulted to identify any other ports, that are not members of a bundle, but whose busy status should also be included in the generated rs_busy_port_mask.

[00124] A different software-defined threshold is used in computing the rs_busy_port_mask because of the larger latency involved in communicating and processing the rs_busy_port_mask and in delivering a frame subject to this mask to the remote switch that generated the mask. Because of the larger latency, it may be useful to require a port to be more loaded before it is considered to be so busy that it is not a good candidate for receiving additional frames from a remote switch. A busy remote port should be sufficiently loaded such that it remains loaded throughout the time it would take to receive a frame that has been subject to the mask.

[00125] The aforementioned RSBGP table stores busy port masks indexed by destination group (global_id). Again, the RSBGP table is used in the evaluation of global minimal paths consisting of a hop to a neighboring switch which has a global port connected to the destination group to filter out ports of the current switch which are poor choices for use in reaching the destination group because the corresponding global port or ports, of the neighboring switch reached by the filtered-out ports of the current switch, are too heavily loaded.

[00126] The RSBLP table stores busy port masks indexed by destination switch (switch_id), and again, can be used in the evaluation of local non-minimal paths consisting of a local hop to a neighboring switch followed by another local hop to the destination switch. For topologies, such as fat-tree, where a local minimal path can consist of a local hop to a neighboring switch followed by another local hop to the destination switch, the RSBLP Table can also be used in the evaluation of local minimal paths. The RSBLP table is used to filter out ports on the current switch which are poor choices for use in indirectly reaching the destination switch because the neighboring switch's port or ports that connect to the destination switch are too heavily loaded.

[00127] It should be noted that the RSBGP table and the RSBLP table are never both accessed for the same routing request. The former is accessed when routing a frame that is not in the destination group, and the latter is accessed only when routing a frame that is in the destination group. Therefore, both are implemented within the same memory, termed the Remote Switch Busy Port

[00128] Provided that there is at least one valid candidate port, the various busy port filters (Busy Ports Filter, Local Non-Minimal (LN) Busy Port Filter, Global Non-Minimal (GN) Busy Port Filter) may not be allowed to collectively block all candidate ports. If there are viable candidate port choices, they are allowed, despite being “poor” choices, if there are no

better choices. Otherwise, an empty route response will be generated for the routing request when routes are actually available.

[00129] To prevent an incorrect empty route response from being generated, the first stage of the preferred and non-preferred minimal path busy port filters (FIG. 4A) and the first stage of the local non-minimal path busy port filter (FIG. 4B) are all disabled if the following conditions are all true: No candidates survive the first stage of the preferred minimal path busy port filter (Busy Ports Filter); No candidates survive the first stage of the non-preferred minimal path busy port filter (Busy Ports Filter); No candidates survive the first stage of the local non-minimal busy port filter (Local Non-Minimal (LN) Busy Port Filter); and No candidates survive the global non-minimal busy port filter (Global NonMinimal (GN) Busy Port Filter).

[00130] It should be noted that there will be no minimal path candidate ports if minimal routing is disabled (Permitted Ports Filter). There will be local non-minimal path candidate ports only if local non-minimal routing is enabled (Candidate Local Non-Minimal Path Ports). There will be global non-minimal path candidates only if global non-minimal routing is enabled (Candidate Global Non-Minimal Path Ports). Local and global non-minimal routing are generally not both enabled simultaneously. When the first stage of the preferred and non-preferred minimal path busy port filters and the local non-minimal busy port filter are disabled due to the conditions described above, the only candidate ports that will be seen at an adaptive selection stage (described below) will be poor candidates because they will all be ports that lead to other switches whose ports (that connect to the destination group or to the destination switch) are heavily loaded. This is because these are the only candidate ports that were being blocked by the filters that are being disabled and, without these filters being disabled, there are no other candidates.

[00131] The adaptive selection stage will choose between these remaining/surviving candidate ports, which are all poor, based on their biased local loads (Local Load and Load

Value Selection), although their local loads will not necessarily reflect the reason why they are poor. Their poor character can be the result of high downstream load on certain ports of the other switches reached by these candidate ports. It is because the adaptive selection stage may not be able to see how poor these candidates are that the coordination between the different busy port filters, described herein may be used. If each busy port filter decides independently whether or not to disable its RSBGP Table and RSBLP Table-based filters, situations such as the following could occur. The non-preferred minimal path busy port filter might produce one or more candidates, which are not poor, without any of its filter stages disabled. The preferred minimal path busy port filter might only be able to produce one or more candidate ports by disabling both of its filter stages. Thus, all of the candidate ports it is able to produce are poor. At the adaptive select stage, the down-selected, not poor, non-preferred minimal path candidate ports, are compared against the down-selected, poor, preferred minimal path candidate ports. However, the adaptive selection stage lacks visibility into how poor the preferred minimal path candidates are, so it may select a poor preferred minimal path candidate over a not poor non-preferred minimal path candidate.

[00132] An alternative to the busy port filter coordination mechanism described herein, would be for all of the busy port filters to act independently, but for the minimal path and the local non-minimal path busy port filters to each forward a signal through to the adaptive select stage to indicate if their respective candidate ports are poor choices due to busy ports at downstream switches. If they are, the adaptive selection function may de-prioritize their candidate ports in favor of other ports. The result would be the same as is achieved by the coordination, described herein, between the various busy port filters.

[00133] As illustrated in FIG. 5, load-based adaptive selection can be performed on the surviving path candidate ports 510 that remain after the pseudo-random selection process is performed by FRF component 400. The adaptive selection stage will result in a single, least loaded candidate port 512 to which a frame can be routed, where the current load present

on the candidate ports surviving pseudo-random down selection (surviving path candidate ports 508) are compared to determine the least loaded candidate port among this remaining set of candidate ports.

[00134] In some embodiments, preferred minimal path candidate ports are preferentially selected over non-preferred minimal path candidate ports, and minimal path candidate ports are to be preferentially selected over non-minimal path candidate ports. To accomplish this preferential selection, a bias value can be added to each candidate port's load before the adaptive selection comparison is performed. The bias value used can be configured using CSRs, and can vary depending on the type of path to which it is being applied (i.e., non-preferred minimal, preferred minimal, and non-minimal), the traffic class of the frame being routed, and where the frame is along its path. For example, frames belonging to a low-latency traffic class can be more strongly biased towards minimal paths versus frames in other traffic classes to have a greater likelihood of achieving/comporting with low-latency requirements or needs. Frames may also be increasingly biased towards minimal path routes the closer the frames are to their destination.

[00135] It should be understood that each biased load value (9-bit wide) is obtained by adding an unsigned (8-bit wide) bias value to an unsigned (8-bit wide) port load value. To determine the bias value and the type of load value that applies to each candidate, the preceding minimal port selection and non-minimal port selection stages separate the candidate ports into three different categories. Bias value are obtained from a software configurable table, i.e., bias table 414A (described in greater detail below). The type of load value to be used for each port is determined, and result in minimal preferred candidate ports (which include edge port candidates, if any exist), non-minimal candidate ports (either global or local non-minimal candidates depending on the configuration of routing algorithm table 408), where a set of global non-minimal candidate ports can include both global and local ports, while a set of local non-minimal candidate ports include only local ports. Depending

on the point where the routing request is being performed, along the path between the source and destination edge ports, a category might not contain any candidates. For example, when at the destination switch, with typical configuration only the minimal non-preferred category will contain any candidates for unicast traffic.

[00136] To enable the bias to change as a frame progresses toward its destination as well as to allow the bias to be dependent on the frame's FTag, the bias table 414A's entries are indexed by both the frame's point along its path (e.g. at the source group, at an intermediate group, at the destination group) and by its FTag bias class. The FTag bias class is a value derived from the frame's FTag value. Each entry in the bias table provides three bias values: one used for minimal preferred candidate ports, one used for non-preferred minimal candidate ports, and one used for non-minimal candidate ports.

[00137] In some embodiments, biased load override is also implemented, where if a candidate category's bias value, obtained from the bias table 414A, is the maximum possible bias value, that category is handled differently than it would be handled for other bias values. In particular, ports, to which the maximum bias value is being applied, are never chosen, regardless of their load, unless there are no other choices available. However, if the maximum possible bias value is being applied to all available ports, then the available port with the lowest biased load is chosen; the different handling, associated with the maximum possible bias value, is overridden in this case.

[00138] In particular, load values represent the busyness of switch 202's ports and are used in evaluating the load-based port masks and in comparing candidate ports during the adaptive route selection process. Load-based port masks are used in the busy port filters to remove ports that are poor candidates, based on current load, from the set of candidates being considered. There are a number of different types of load values used within the switch and some are communicated to neighboring switch devices.

[00139] Several load metrics are computed and used in determining which port to route a frame when there is more than one port to which the frame can be routed. The load metrics are also used in generating busy-port masks, which as described above, are used to remove heavily loaded ports from consideration.

[00140] There are five load metrics described herein: local load; group load; global non-minimal global port load; mean global load; and global non-minimal local port load.

[00141] Regarding local load, the load of each of a switch's output ports (e.g., output ports 220c of switch 202) is continuously being evaluated and provided to the corresponding FRF instance as the 8-bit value local_load. Larger values represent higher load. The current load present at each output port is measured by the output control age queue block. The output port load is provided by each age queue instance to the FRF instance (of FRF component 400) that is associated with the input side of the same port. The load value provided to the FRF is an 8-bit value termed the local_load. The age queue determines the local load based on a combination of the amount of traffic enqueued, waiting to go out that port and the amount of traffic enqueued at the opposite side of the link in the link-partner Switch device's input buffer. The calculation and configuration of local_load is performed later. Each FRF instance distributes the local_load value it receives from its associated age queue instance to all other FRF instances. In this way, each FRF instance learns the current local_load of every output port.

[00142] When the port loads of candidate ports are being compared to determine the best port to route a frame to, it is the port's local_load value that is used for ports being considered for minimal path routing, and for ports being considered for local non-minimal path routing.

[00143] Group load is a measure of how suitable a dragonfly group is for use as the intermediate group in a global non-minimal path. The 8-bit group load value is not computed by a switch, such as switch 202, but is software-configurable. Software might use a measure

of the network injection load present across the input side of the group's edge ports in deriving the group_load value, or might determine the group_load value based on a policy of discouraging use of certain groups as non-minimal intermediate groups, perhaps based on the jobs or services that are running in the groups. That is, the group_load value is intended to be representative of the amount of local traffic within a group.

[00144] A network management stack sets the group_load value by periodically writing to a CSR. The software-configured group load value is communicated across global links. FRF instances associated with global links forward the group_load value that they receive from their link-partner in the group at the opposite side of the link, to all other FRF instances in the switch. In this way, each FRF instance learns the group load values of the groups at the opposite end of each of the global links terminated by the switch.

[00145] In terms of global non-minimal global port load (gnmgp_load), this load is a metric used in assessing a global port's suitability for use in directing a frame to the intermediate group reached by the global link connected to the global port. The gnmgp_load is nominally equal to the maximum of the global port's local_load and the group_load value being received from the group reached by the global link. However, through field G-NMGP _EN_ GRP LD in CSRR _ TF __ FRF_CFG-_LOAD_CTRL, the group_load component can be excluded.

[00146] When the port loads of candidate ports are being compared to determine the best choice port to route a frame to, it is the port's gnmgp_load value that is used for global ports being considered for global non-minimal path routing.

[00147] Mean global load (mean_global_load) is intended for use in assessing a switch's suitability for use in reaching any intermediate group that is directly connected to that switch. The mean_global_load value is an 8-bit value equal to the arithmetic mean of the gnmgp_load values of all of the switch's global ports. Ports that are classified as non-operational by either hardware or software are excluded from the calculation. The ports to

include in the mean_global_load computation are determined from CSR R TF FRF CFG GNM global ports.

[00148] For any port whose load is included in the mean_global_load computation, if a group_load value is not being received for that port from the port status ring, either because link partner data is not being received for that port or because the link partner data that is being received is not global link data, the contribution of that port to the mean_global_load is based solely on that port's local load. It should be understood that the aforementioned port status ring communicates status and load information amongst the ports on a switch (e.g., input and output ports 220b and 220c, respectively, of switch 202). The computed mean_global_load value is communicated across local links. FRF instances associated with local links forward the mean_global_load value,that they receive from their link-partner in the local switch at the opposite side of the link, to all other FRF instances in a switch. In this way, each FRF instance learns the mean_global_load values of the local switches at the opposite end of each of the local links terminated by the switch, and can use these values in global non-minimal path selection.

[00149] Global non-minimal local port load metrics are computed by each FRF instance, distributed between ports in a switch using the port status ring, and distributed between switches. The global non-minimal local port load is a metric for use in assessing a local port's suitability for use in directing a frame to an intermediate group of a global non-minimal path. The global non-minimal local port load takes into account the load on the local port as well as the suitability of the local group switch, to which the port connects, for use in reaching an intermediate group. A port's gnmlp_load value is equal to the maximum of the port's local_load and the mean_global_load reported by the port's partner switch. Through software configuration it is possible to remove the mean_global_load component such that a port's gnmlp_load becomes simply equal to its local_load.

[00150] The gnmlp_load value is an 8-bit value. The computed gnmlp_load value is based on the local_load and mean_global_load values distributed via the port status ring. Each FRF instance computes the gnmlp_load value for all of its switch's ports at which link partner data for a local link is being received. If load status information is not being received from a port's partner switch, the gnmlp_load value for that port is set equal to the port's local_load.

[00151] For ports at which link partner data for a global link is being received, the port's gnmlp_load value is set to the value computed for gnmgp_load. This is a side-effect of an implementation optimization in which the same storage is used for gnmgp_load and gnmlp_load since, for any given port, at most one of the two is valid. The global non-minimal local port load metric is not used on ports where global link partner data is being received.

[00152] FIG. 6 illustrates an example process for deadlock-free multicast routing in accordance with one embodiment of the disclosed technology. At operation 606, a switch (e.g., switch 202) receives a multicast transmission at an edge port of a switch. As noted, a multicast packet is received at an edge port of the network and copies of the packet are forwarded to a defined set of edge ports leaving the network. At operation 608, the switch identifies the transmission as a network multicast transmission.

[00153] At operation 610 an entry is created in a multicast table within the switch to identify ports of the current switch to which copies of the multicast packet should be forwarded. This implicitly determines the number of copies generated at that point, as well, because for each multicast packet received at a port, only one copy is forwarded to each other port identified by the multicast table as requiring a copy.

[00154] If the format of the DFA of the routing request is multicast, the set of ports to be returned in response to the routing request is not generated through the adaptive routing process; instead, it is simply looked up in the multicast tables. Accordingly, operation 612 routes the multicast transmission across the network to a plurality of destinations via a

plurality of links. At each of the links, the multicast table is referenced to determine to which ports the multicast transmission should be forwarded. A two stage process is used to perform the look-up. First the multicast_id, obtained from the DFA, is looked-up in the Multicast Address Map Table (mam_table) to produce a port_set_id. The port_set_id is then looked up in the Multicast Port Table (mcast_port_table) to determine the specific ports that are included in the port set identified by the port_set_id. Regardless of the configuration of the multicast tables, multicast frames are not forwarded to ports that are classified as non-operational by either hardware or software.

[00155] At operation 614, the virtual channel used by each copy of the multicast transmission is changed, as necessary, as the copy progresses through the network. A base virtual channel, to be used for the forwarded multicast frames, is looked up in a VC remapping table indexed by the virtual channel on which the frame was received.

[00156] FIG. 7 illustrates an example computing component 700 that may be used to effectuate deadlock-free multicast routing in accordance with one embodiment of the disclosed technology. Computing component 700 may be, for example, a server computer, a controller, or any other similar computing component capable of processing data. In the example implementation of FIG. 7, the computing component 700 includes a hardware processor 702, and machine-readable storage medium 704. Hardware processor 702 may be one or more central processing units (CPUs), semiconductor-based microprocessors, and/or other hardware devices suitable for retrieval and execution of instructions stored in machine-readable storage medium 704. Hardware processor 702 may fetch, decode, and execute instructions, such as instructions 706-714, to control processes or operations for merging local parameters to effectuate deadlock-free multicast routing. As an alternative or in addition to retrieving and executing instructions, hardware processor 702 may include one or more electronic circuits that include electronic components for performing the functionality

of one or more instructions, such as a field programmable gate array (FPGA), application specific integrated circuit (ASIC), or other electronic circuits.

[00157] A machine-readable storage medium, such as machine-readable storage medium 704, may be any electronic, magnetic, optical, or other physical storage device that contains or stores executable instructions. Thus, machine-readable storage medium 704 may be, for example, Random Access Memory (RAM), non-volatile RAM (NVRAM), an Electrically Erasable Programmable Read-Only Memory (EEPROM), a storage device, an optical disc, and the like. In some embodiments, machine-readable storage medium 704 may be a non-transitory storage medium, where the term "non-transitory" does not encompass transitory propagating signals. As described in detail below, machine-readable storage medium 704 may be encoded with executable instructions, for example, instructions 706-714.

[00158] Hardware processor 702 may execute instruction 706 to receive a multicast transmission at an edge port of a switch. As noted, a multicast packet is received at an edge port of the network and copies of the packet are forwarded to a defined set of edge ports leaving the network. Hardware processor 702 may execute instruction 708 to identify the transmission as a network multicast transmission.

[00159] Hardware processor 702 may execute instruction 710 to create an entry in a multicast table within the switch. The multicast table identifies the ports of the current switch to which copies of the multicast packet should be forwarded. This implicitly determines the number of copies generated at that point, as well, because for each multicast packet received at a port, only one copy is forwarded to each other port identified by the multicast table as requiring a copy.

[00160] If the format of the DFA of the routing request is multicast, the set of ports to be returned in response to the routing request is not generated through the adaptive routing process; instead, it is simply looked up in the multicast tables. Accordingly, hardware processor 702 may execute instruction 712 to route the multicast transmission across the

network to a plurality of destinations via a plurality of links. At each of the links, the multicast table is referenced to determine to which ports the multicast transmission should be forwarded. A two stage process is used to perform the look-up. First the multicast_id, obtained from the DFA, is looked-up in the Multicast Address Map Table (mam_table) to produce a port_set_id. The port_set_id is then looked up in the Multicast Port Table (mcast_port_table) to determine the specific ports that are included in the port set identified by the port_set_id. Regardless of the configuration of the multicast tables, multicast frames are not forwarded to ports that are classified as non-operational by either hardware or software.

[00161] Hardware processor 702 may execute instruction 714 to change, when necessary, the virtual channel used by each copy of the multicast transmission as the copy progresses through the network. A base virtual channel, to be used for the forwarded multicast frames, is looked up in a VC remapping table indexed by the virtual channel on which the frame was received.

[00162] The physical topology of Dragonfly networks contains loops, thus the potential for deadlock exists. If done in an unrestricted manner, routing through an intermediate switch without advancing the VC can lead to deadlock if there is more than one pair of switches without connectivity between them. Fig. 8 illustrates deadlock caused by unrestricted routing. Because of the lack of connectivity between switches S1 and S5, traffic between these switches, arriving at the group on VC2, is routed through intermediate switches on VC3 without advancing the VC when passing through the intermediate switch. The situation is the same for traffic between S3 and S7 arriving at the group on VC2. Path P1's progress depends on P3's progress, which depends on P5's progress, which depends on P7's progress. And, because P7's progress is dependent on P1's progress, a cyclic dependency exists, so deadlock is possible.

[00163] If the traffic for each of these paths arrives at the group on VC1 (or VCO), deadlock will not occur. For example, if traffic arrives at P1 on VCO, it will use VC1 between S1 and S3 and VC2 between S3 and S5. If traffic arrives at P3 on VC1, it will use VC2 between S3 and S5 and VC3 between S5 and S7. If traffic arrives at P5 on VC1 it will use VC2 between S5 and S7 and VC3 between S7 and S1. P1 is dependent on P3 as both share VC2 between S3 and S5. However, P3 is not dependent on P5 because, between S5 and S7, P3 is on VC3 while P5 is on the lower VC, VC2. There is no cyclic dependency.

[00164] In the case where traffic does arrive at the group on VC2, and must route indirectly through an intermediate switch because of a lack of connectivity between the ingress and egress switch, the risk of deadlock can be avoided by restricting the traffic arriving on VC2 to only route through intermediate switches that have connectivity with every other switch in the group. An example of this is shown at FIG. 9.

[00165] Because of this complete connectivity, traffic entering the group at the intermediate switch on VC2 will only ever take a single hop before exiting the group, so no cyclic dependency can form. In FIG. 9, traffic arrives at P1 on VC2 and uses VC3 between S1 and S2 and between S2 and S5. Traffic arrives at P2 on VC2 and uses VC3 between S2 and S5. P1 is dependent on P2 as both share VC3 between S2 and S5. However, P2 exits the group at S5 so it does not have a dependency on any other paths that carry on to other switches after S5. Because P1 and P2 are on the highest VC (VC3) between S2 and S5, they are not dependent on other traffic between S2 and S5 using lower VCs. If there are N switches in the group, at least N/2 links must fail for there to be no switches left that have full connectivity with every other switch in the group; more if there is more than one link between each pair of switches.

[00166] Deadlock is prevented for the multicast traffic by following the same routing rules for multicast traffic as are followed for unicast traffic. The multicast forwarding tables

must be programmed to only send each copy of a multicast packet along a path that is permitted for Dragonfly routing, and that is a path that a unicast packet could also follow. To prevent the formation of flow control dependency loops that could lead to deadlock, the virtual channel (VC) in which a packet is flowing is advanced at specific points as a packet traverses its path across the network. Identical VC switching rules are employed for multicast traffic as are employed for unicast traffic. Applying the same routing rules to unicast and multicast traffic ensures unicast and multicast traffic can simultaneously coexist in the network.

[00167] The multicast address map table is sized to support 8192 multicast_id values. To reduce ECC overhead, each word within the multicast address map table contains four 10-bit port_set_ids corresponding to four consecutive multicast_ids. The multicast port table is sized to support 1024 unique port sets. Each entry in the multicast port table contains a 64 bit field, used to indicate the ports to which the multicast frame should be forwarded, and a 20-bit waitcount field used to support the arming of reduction operations.

[00168] The multicast address map table is implemented as an ECC protected memory of 2048 words by 40 bits (4 * 10) (plus ECC) per word. The mcast_port_ table is implemented as an ECC protected memory of 1024 words by 84 bits (64 + 20) (plus ECC) per word.

[00169] A base VC, to be used for the forwarded multicast frames, is looked up in a VC remapping table indexed by the VC the frame was received on. A CSR-based VC increment mask is also included in the FRF's response to the multicast routing request. The mask indicates ports for which the VC, to be used for the copies of the forwarded frame, should be the base VC+1 instead of just the base VC. This can be used, for example, to increment the VC when forwarding from a local port to another local port. This is achieved by configuring the VC increment mask of local ports to identify the local ports while not identifying any ports in the VC increment mask of edge and global ports.

[00170] A maximum VC value is included in the VC remapping table. If the VC on which any copy of the multicast frame is to be forwarded exceeds the specified maximum VC, that copy of the frame is dropped and an error is reported. The occurrence of the error indicates that there may be a configuration error within the network that has caused a multicast frame to be routed incorrectly.

[00171] With network topologies that can contain loops, like Dragonfly, and with multicast configurations that can allow multicast frames using the same multicast_id to originate at different places within the network, there is a risk of multicast frames being caught in cyclic loops and of copies of the same multicast frame taking more than one path to arrive at the same destination, resulting in duplicate delivery of the frame at the destination.

[00172] This document now describes multicast filtering functionality provided by the FRF to handle problematic scenarios. As well, the FRF's functionality to drop multicast frames if their VC value becomes too high, can help to squelch multicast frames should they become caught in an endless cyclic loop.

[00173] A guideline for how multicast traffic can be routed includes that multicast traffic should always be routed minimally except where use of a non-minimal path is necessary to work around a fault in the network. For multicast traffic originating from a particular source group and entering another group, there should be only one global link from which multicast traffic from that particular source group is accepted. If the traffic is attempting to enter the group on a different global link, it can be dropped. Similarly, traffic that has been looped through one or more other groups and arrived back at its source group can also be dropped. Dropping multicast traffic in this manner can prevent duplicate delivery and can prevent frames from looping.

[00174] FIG. 10 illustrates an example process for multicast traffic routing in accordance with some embodiments. At operation 812 a frame is received at a port. At

operation 814, its header information passes through the EIQ (Ethernet Ingress Queues) block prior to being forwarded to the INQ (Input Queue) block. At operation 816, the Fabric Routing Function (FRF) receives the routing request for the frame after it has arrived at the INQ block. However, when the frame's header is being processed by the EIQ, at operation 818 the EIQ interrogates the FRF with the frame's Source Fabric Address (SFA) and DFA. The FRF responds to the EIQ with two pieces of information: whether the frame is still within its source group, and whether the frame should be dropped.

[00175] At operation 822, the FRF uses the SFA provided by the EIQ to determine whether the frame is still in its source group. When the routing request for the frame is subsequently received by the FRF, the frame's SFA is not available to the FRF but this status of whether the frame is still in its source group is included within the information provided in the routing request. The FRF requires this status to determine how to route the frame.

[00176] At operation 824, the FRF uses the DFA provided by the EIQ to determine whether the frame is a multicast frame and, if it is, the FRF looks up the frame's SFA in the Multicast Filter Table to determine whether multicast frames originating in the source group identified by the SFA should be accepted or dropped at the port where the frame has been received. The accept or drop indication is returned to the EIQ. At operation 826, if the frame is to be dropped, the FRF will not, subsequently, receive a routing request for it. The Multicast Filter Table is implemented as an 8 word by 64 bits (plus ECC) per word memory. The 64 bits in each word represent the accept or drop status for 64 consecutive source groups.

[00177] The FRF receives routing requests from the INQ for each received frame, and for each routing request returns a routing response to the IFCT. The routing response indicates which port, or ports, the frame should be forwarded to, and the VC on what it should be forwarded. For non-multicast requests, the response indicates both a preferred port and a set of acceptable ports to which the frame can be forwarded, thereby allowing the IFCT to use the preferred port for a new flow, or a rerouted flow or to maintain the current path,

using a port that may not be the FRF's current preferred choice, for an existing flow. In the presence of errors, the FRF may also indicate to the IFCT that there is no legal port the frame can be forwarded to. When this occurs, the frame is discarded.

[00178] The FRF's routing decisions are based on a combination of software configurable table based rules, dynamic load information and pseudorandom selection. Rules take into account factors including the frame's destination, where it is along its path (source group, intermedia group, destination group, destination switch), the VC it was received on, and the type of port (edge, local, or global) at which it was received. The AGEQ provides the FRF with the current load present at the output side of the port that the FRF instance is associated with. Each FRF instance communicates with every other FRF instance within the Rosetta device to learn the current load present at each output port, and the linkup/down status of each of the device's ports. FRF instances also communicate with FRF instances in neighboring Rosetta devices to obtain load -related status of the neighboring devices. The FRF is sufficiently configurable to be able to support multiple network topologies.

[00179] While still in its source group, group A, a multicast frame may need to be delivered to edge ports of group A and to global ports so that the frame can also be delivered to edge ports of other groups. Another multicast frame, with the same multicast_id, but arriving at group A from another group is delivered to the same edge ports of group A. However, unless, because of network failures, group A is required to act as an intermediate group of a non-minimal global path, the frame that originated outside of group A should, ideally, not also be forwarded to any other groups; to do so will result in duplicates copies of the frame arriving at its destination group - one which took a direct path from its source group and one which was routed indirectly through group A.

[00180] The multicast frame that originated in group A and the one that arrived at group A from another group are both routed using the same entries in the FRF's multicast forwarding tables because both have the same multicast_id. To allow multicast frames to be

conditionally forwarded to global ports, or not, depending on whether the frame is still within its source group, the result returned by the Multicast Port Table, when the frame is being routed, is subject to the Multicast Enabled Ports Filter to determine the final set of ports to which the multicast frame should be forwarded. The Multicast Enabled Ports Filter enables either one set of ports or another depending on whether or not the multicast frame is still in its source group.

[00181] To allow the IFCT block to return discard acks if a multicast request is not forwarded to all of the ports that, based on FRF configuration it was intended that it should be forwarded to, the FRF provides a multicast_port_dropped signal to the IFCT. This signal is valid for all multicast requests and is asserted only if the FRF has dropped one or more ports – possibly all – from the set of ports to which the request was intended to be forwarded. This signal is only asserted if ports are dropped due to errors. Ports dropped due to the pruning actions associated with the multicast port table, described herein, do not cause the multicast_port_dropped signal to be asserted. The conditions that can cause multicast_port_dropped to be asserted are shown in the table below.

Conditions That Can Cause Multicast_Port_Dropped To Be Asserted

An uncorrectable error detected in the entry referenced for the request in either the multicast address map table or the multicast port table.
A valid port type (edge, local, global) not being configured in CSR R_TF_FRF_CFG_BASIC_ROUTING, field LINK_TYPE, Section 9.7.5.20 in the SDG.
The multicast request being unexpected based on the multicast_id value in its DFA. The definition of the MCAST_EMPTY_ROUTE_UMID error flag, in Section 9.7.5.7 in the SDG,

describes the conditions that result in a multicast request being considered to be unexpected.
Any ports being dropped from the multicast request because the ports are either in the non-operational state (Section 9.7.5.30 in the SDG, Section 9.7.5.60 in the SDG), or because the VC that the request would need to be forwarded on, for that port, would be too high (Virtual Channel Control).

[00182] It should be noted that the terms "optimize," "optimal" and the like as used herein can be used to mean making or achieving performance as effective or perfect as possible. However, as one of ordinary skill in the art reading this document will recognize, perfection cannot always be achieved. Accordingly, these terms can also encompass making or achieving performance as good or effective as possible or practical under the given circumstances, or making or achieving performance better than that which can be achieved with other settings or parameters.]

[00183] FIG. 11 depicts a block diagram of an example computer system 1100 in which various of the embodiments described herein may be implemented. The computer system 1100 includes a bus 1102 or other communication mechanism for communicating information, one or more hardware processors 1104 coupled with bus 1102 for processing information. Hardware processor(s) 1104 may be, for example, one or more general purpose microprocessors.

[00184] The computer system 1100 also includes a main memory 1106, such as a random access memory (RAM), cache and/or other dynamic storage devices, coupled to bus 1102 for storing information and instructions to be executed by processor 1104. Main memory 1106 also may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor 1104. Such instructions, when stored in storage media accessible to processor 1104, render computer

system 1100 into a special-purpose machine that is customized to perform the operations specified in the instructions.

[00185] The computer system 1100 further includes a read only memory (ROM) 1108 or other static storage device coupled to bus 1102 for storing static information and instructions for processor 1104. A storage device 1110, such as a magnetic disk, optical disk, or USB thumb drive (Flash drive), etc., is provided and coupled to bus 1102 for storing information and instructions.

[00186] The computer system 1100 may be coupled via bus 1102 to a display 1112, such as a liquid crystal display (LCD) (or touch screen), for displaying information to a computer user. An input device 1114, including alphanumeric and other keys, is coupled to bus 1102 for communicating information and command selections to processor 1104. Another type of user input device is cursor control 1116, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 1104 and for controlling cursor movement on display 1112. In some embodiments, the same direction information and command selections as cursor control may be implemented via receiving touches on a touch screen without a cursor.

[00187] The computing system 1100 may include a user interface module to implement a GUI that may be stored in a mass storage device as executable software codes that are executed by the computing device(s). This and other modules may include, by way of example, components, such as software components, object-oriented software components, class components and task components, processes, functions, attributes, procedures, subroutines, segments of program code, drivers, firmware, microcode, circuitry, data, databases, data structures, tables, arrays, and variables.

[00188] In general, the word "component," "engine," "system," "database," and the like, as used herein, can refer to logic embodied in hardware or firmware, or to a collection of software instructions, possibly having entry and exit points, written in a programming

language, such as, for example, Java, C or C++. A software component may be compiled and linked into an executable program, installed in a dynamic link library, or may be written in an interpreted programming language such as, for example, BASIC, Perl, or Python. It will be appreciated that software components may be callable from other components or from themselves, and/or may be invoked in response to detected events or interrupts. Software components configured for execution on computing devices may be provided on a computer readable medium, such as a compact disc, digital video disc, flash drive, magnetic disc, or any other tangible medium, or as a digital download (and may be originally stored in a compressed or installable format that requires installation, decompression or decryption prior to execution). Such software code may be stored, partially or fully, on a memory device of the executing computing device, for execution by the computing device. Software instructions may be embedded in firmware, such as an EPROM. It will be further appreciated that hardware components may be comprised of connected logic units, such as gates and flip-flops, and/or may be comprised of programmable units, such as programmable gate arrays or processors.

[00189] The computer system 1100 may implement the techniques described herein using customized hard-wired logic, one or more ASICs or FPGAs, firmware and/or program logic which in combination with the computer system causes or programs computer system 1100 to be a special-purpose machine. According to one embodiment, the techniques herein are performed by computer system 1100 in response to processor(s) 1104 executing one or more sequences of one or more instructions contained in main memory 1106. Such instructions may be read into main memory 1106 from another storage medium, such as storage device 1110. Execution of the sequences of instructions contained in main memory 1106 causes processor(s) 1104 to perform the process steps described herein. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions.

[00190] The term "non-transitory media," and similar terms, as used herein refers to any media that store data and/or instructions that cause a machine to operate in a specific fashion. Such non-transitory media may comprise non-volatile media and/or volatile media. Non-volatile media includes, for example, optical or magnetic disks, such as storage device 1110. Volatile media includes dynamic memory, such as main memory 1106. Common forms of non-transitory media include, for example, a floppy disk, a flexible disk, hard disk, solid state drive, magnetic tape, or any other magnetic data storage medium, a CD-ROM, any other optical data storage medium, any physical medium with patterns of holes, a RAM, a PROM, and EPROM, a FLASH-EPROM, NVRAM, any other memory chip or cartridge, and networked versions of the same.

[00191] Non-transitory media is distinct from but may be used in conjunction with transmission media. Transmission media participates in transferring information between non-transitory media. For example, transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus 1102. Transmission media can also take the form of acoustic or light waves, such as those generated during radio-wave and infra-red data communications.

[00192] The computer system 1100 also includes a communication interface 1118 coupled to bus 1102. Network interface 1118 provides a two-way data communication coupling to one or more network links that are connected to one or more local networks. For example, communication interface 1118 may be an integrated services digital network (ISDN) card, cable modem, satellite modem, or a modem to provide a data communication connection to a corresponding type of telephone line. As another example, network interface 1118 may be a local area network (LAN) card to provide a data communication connection to a compatible LAN (or WAN component to communicated with a WAN). Wireless links may also be implemented. In any such implementation, network interface 1118 sends and

receives electrical, electromagnetic or optical signals that carry digital data streams representing various types of information.

[00193] A network link typically provides data communication through one or more networks to other data devices. For example, a network link may provide a connection through local network to a host computer or to data equipment operated by an Internet Service Provider (ISP). The ISP in turn provides data communication services through the world wide packet data communication network now commonly referred to as the "Internet." Local network and Internet both use electrical, electromagnetic or optical signals that carry digital data streams. The signals through the various networks and the signals on network link and through communication interface 1118, which carry the digital data to and from computer system 1100, are example forms of transmission media.

[00194] The computer system 1100 can send messages and receive data, including program code, through the network(s), network link and communication interface 1118. In the Internet example, a server might transmit a requested code for an application program through the Internet, the ISP, the local network and the communication interface 1118.

[00195] The received code may be executed by processor 1104 as it is received, and/or stored in storage device 1110, or other non-volatile storage for later execution.

[00196] Each of the processes, methods, and algorithms described in the preceding sections may be embodied in, and fully or partially automated by, code components executed by one or more computer systems or computer processors comprising computer hardware. The one or more computer systems or computer processors may also operate to support performance of the relevant operations in a "cloud computing" environment or as a "software as a service" (SaaS). The processes and algorithms may be implemented partially or wholly in application-specific circuitry. The various features and processes described above may be used independently of one another, or may be combined in various ways. Different combinations and sub-combinations are intended to fall within the scope of this disclosure,

and certain method or process blocks may be omitted in some implementations. The methods and processes described herein are also not limited to any particular sequence, and the blocks or states relating thereto can be performed in other sequences that are appropriate, or may be performed in parallel, or in some other manner. Blocks or states may be added to or removed from the disclosed example embodiments. The performance of certain of the operations or processes may be distributed among computer systems or computers processors, not only residing within a single machine, but deployed across a number of machines.

[00197] As used herein, a circuit might be implemented utilizing any form of hardware, software, or a combination thereof. For example, one or more processors, controllers, ASICs, PLAs, PALs, CPLDs, FPGAs, logical components, software routines or other mechanisms might be implemented to make up a circuit. In implementation, the various circuits described herein might be implemented as discrete circuits or the functions and features described can be shared in part or in total among one or more circuits. Even though various features or elements of functionality may be individually described or claimed as separate circuits, these features and functionality can be shared among one or more common circuits, and such description shall not require or imply that separate circuits are required to implement such features or functionality. Where a circuit is implemented in whole or in part using software, such software can be implemented to operate with a computing or processing system capable of carrying out the functionality described with respect thereto, such as computer system 700.

[00198] As used herein, the term "or" may be construed in either an inclusive or exclusive sense. Moreover, the description of resources, operations, or structures in the singular shall not be read to exclude the plural. Conditional language, such as, among others, "can," "could," "might," or "may," unless specifically stated otherwise, or otherwise understood within the context as used, is generally intended to convey that certain

embodiments include, while other embodiments do not include, certain features, elements and/or steps.

[00199] Terms and phrases used in this document, and variations thereof, unless otherwise expressly stated, should be construed as open ended as opposed to limiting. Adjectives such as "conventional," "traditional," "normal," "standard," "known," and terms of similar meaning should not be construed as limiting the item described to a given time period or to an item available as of a given time, but instead should be read to encompass conventional, traditional, normal, or standard technologies that may be available or known now or at any time in the future. The presence of broadening words and phrases such as "one or more," "at least," "but not limited to" or other like phrases in some instances shall not be read to mean that the narrower case is intended or required in instances where such broadening phrases may be absent.

Claims

What is claimed is:

1. A method of managing multicast data transmission in a network having a plurality of switches arranged in a Dragonfly network topology, comprising:

receiving a multicast transmission at an edge port of a switch and identifying the transmission as a network multicast transmission;

creating an entry in a multicast table within the switch;

routing the multicast transmission across the network to a plurality of destinations via a plurality of links, wherein at each of the links the multicast table is referenced to determine to which ports the multicast transmission should be forwarded; and

changing, when necessary, the virtual channel used by each copy of the multicast transmission as the copy progresses through the network.

2. The method of claim 1, further comprising pruning copies of the multicast transmission based on a source group at which the multicast transmission originated.

3. The method of claim 1, wherein routing the multicast transmission comprises statically routing multicast traffic based on a configuration of the multicast forwarding table, wherein the multicast forwarding table identifies ports of a current switch to which copies of a multicast packet should be forwarded.

4. The method of claim 3, wherein identifying the ports of the current switch to which copies of the multicast packet should be forwarded, further determines the number of copies generated at the current switch.

5. The method of claim 1, wherein routing comprises following unicast traffic rules for the multicast transmission to avoid deadlock.

6. The method of claim 1, wherein routing further comprises only sending each copy of a multicast packet along a path that is legal for Dragonfly routing and that is a path that a unicast packet is permitted to follow.

7. The method of claim 1, further comprising advancing a virtual channel in which a multicast transmission is flowing as the multicast transmission traverses across the Dragonfly network.

8. The method of claim 7, wherein identical virtual channel switching rules are employed for multicast traffic as are employed for unicast traffic.

9. The method of claim 1, wherein routing the multicast transmission comprises routing only one copy of the multicast transmission to each other port identified by the multicast table.

10. The method of claim 1, wherein determining to which ports the multicast transmission should be forwarded, comprises:

looking up a multicast_id in a multicast address map table to produce a port_set_id; and

looking up the port_set_id in a multicast port table to determine specific ports that are included in a port set identified by the port_set_id.

11. The method of claim 1, wherein changing the virtual channel comprises looking up a base virtual channel to be used for the forwarded multicast transmission in a VC remapping table indexed by the virtual channel on which the frame was received.

12. The method of claim 1, further comprising using a virtual channel increment mask for copies of the forwarded frame to increment the VC when forwarding from a local port to another local port.

13. The method of claim 12, further comprising configuring the virtual channel increment mask of local ports to identify the local ports while not identifying any ports in the virtual channel increment mask of edge and global ports.

14. The method of claim 1, wherein routing the multicast transmission comprises routing the multicast transmission minimally except where use of a non-minimal path is necessary to work around a fault in the network.

15. The method of claim 1, wherein routing the multicast transmission comprises accepting the multicast transmission from only one global link for a particular source group.

16. The method of claim 16, further comprising checking a global link from which the multicast transmission is entering a source group, and if the multicast transmission is attempting to enter the source group on a second global link that is different from the one global, dropping the multi-cast transmission link

17. A switch, comprising:

an input port of a switch to receive a multicast transmission;

a plurality of output ports to send a copy of the multicast transmission;

a multicast forwarding table within the switch to identify the output ports of the switch to which copies of the multicast transmission should be forwarded;

a crossbar switch to route copies of the multicast transmission to the identified output ports; and

a virtual channel remapping table to identify a virtual channel to be used for the multicast transmission;

wherein the multicast transmission is routed across a network to a plurality of destinations via a plurality of links, wherein at each of the links a multicast table is referenced to determine to which ports within each link the multicast transmission should be forwarded;

and wherein the virtual channel used by the multicast transmission changes as the multicast transmission traverses a network of multiple switches.

18. The switch of claim 17, wherein the multicast forwarding table identifies ports of a current switch to which copies of a multicast packet should be forwarded.

19. The switch of claim 17, wherein the virtual channel used by a copy of the multicast transmission changes as the copy traverses a network of multiple switches.

20. The switch of claim 17, further comprising virtual channel increment mask to increment the VC when forwarding from a local port to another local port.

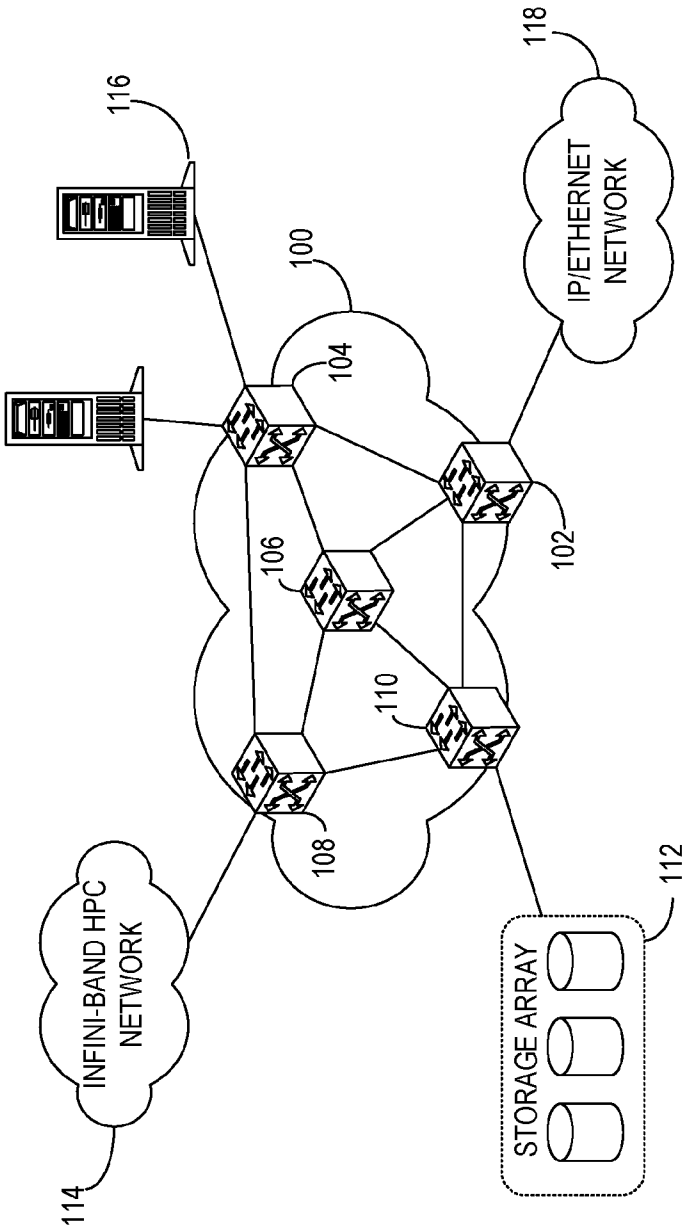
116
118
IP/ETHERNET NETWORK
100
104
102
106
110
108
INFINI-BAND HPC NETWORK
114
STORAGE ARRAY
112

FIG. 1

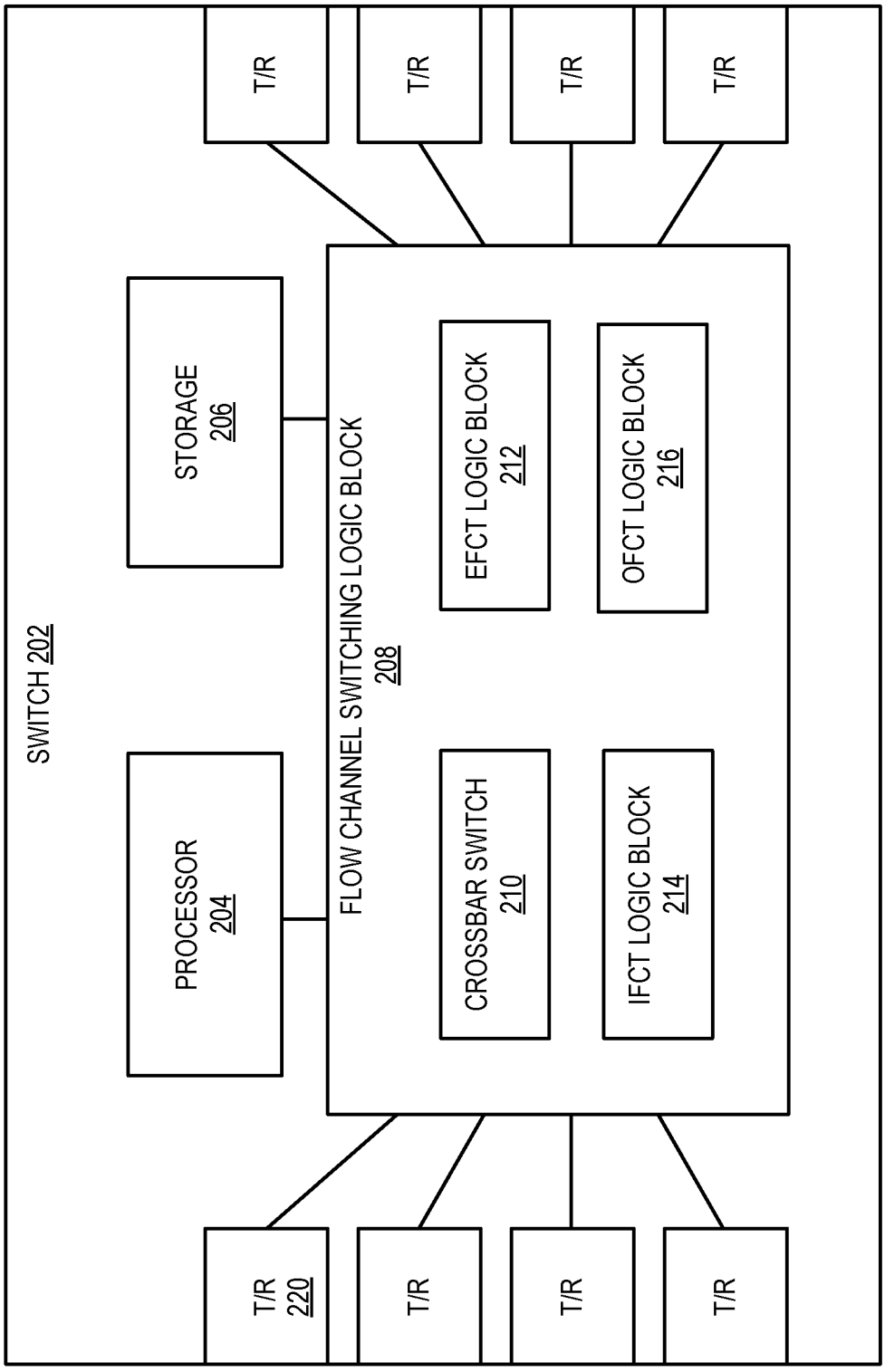
SWITCH 202
T/R
T/R
T/R
T/R
STORAGE 206
FLOW CHANNEL SWITCHING LOGIC BLOCK 208
EFCT LOGIC BLOCK 212
OFCT LOGIC BLOCK 216
PROCESSOR 204
CROSSBAR SWITCH 210
IFCT LOGIC BLOCK 214
T/R 220
T/R
T/R
T/R

FIG. 2

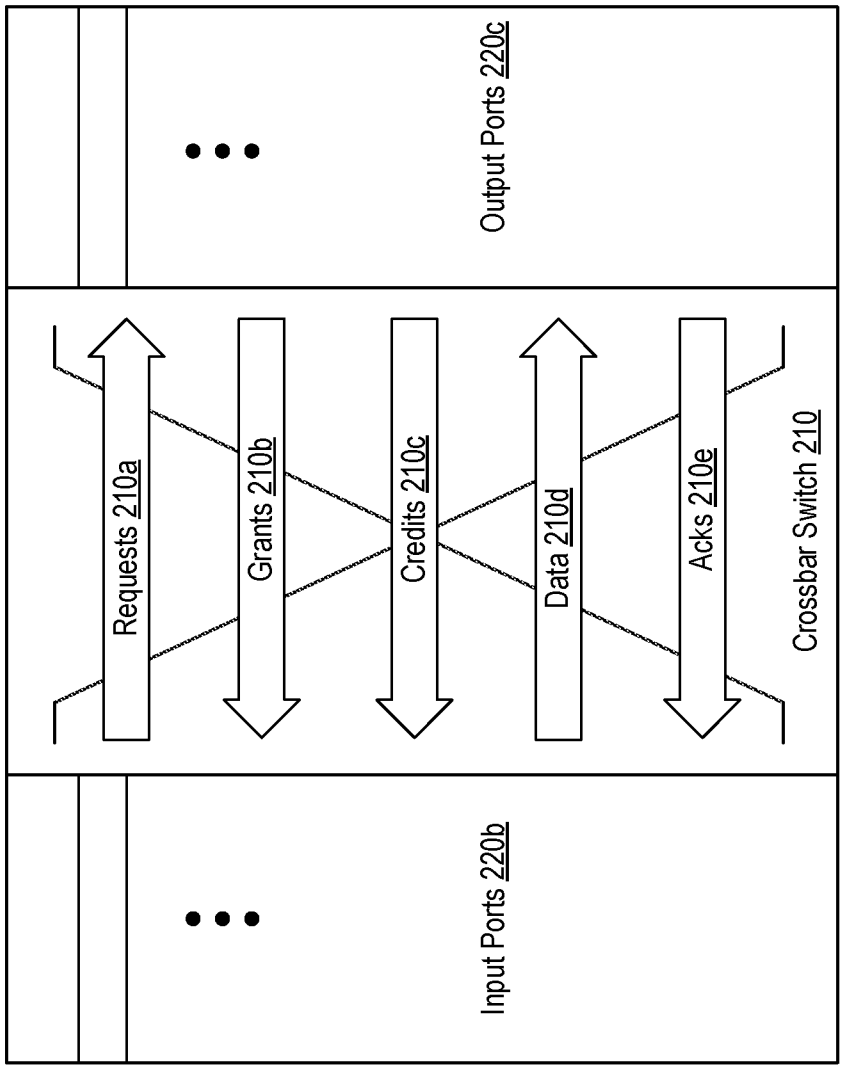
Output Ports 220c
Requests 210a
Grants 210b
Credits 210c
Data 210d
Acks 210e
Crossbar Switch 210
Input Ports 220b

FIG. 3A

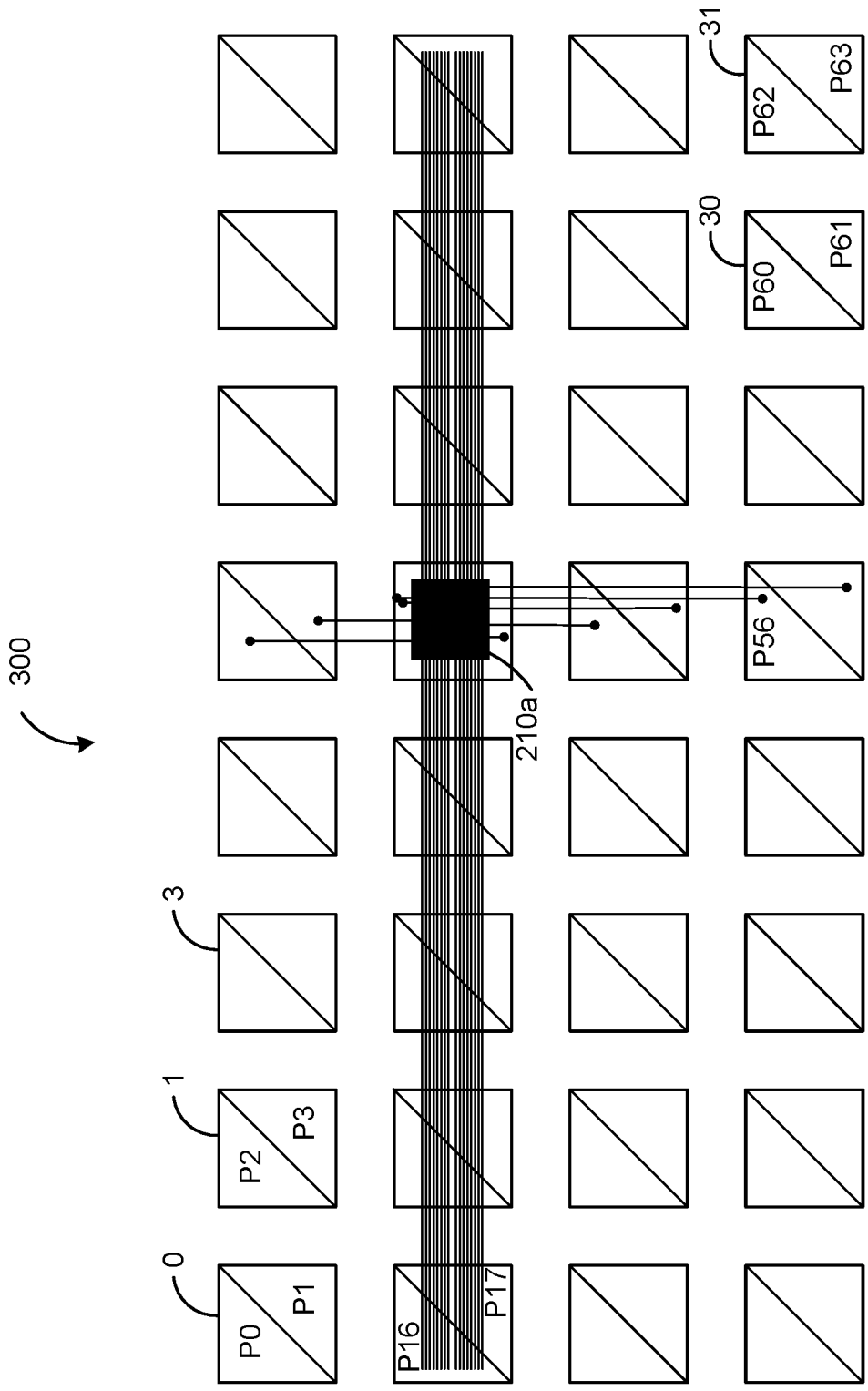

FIG. 3B

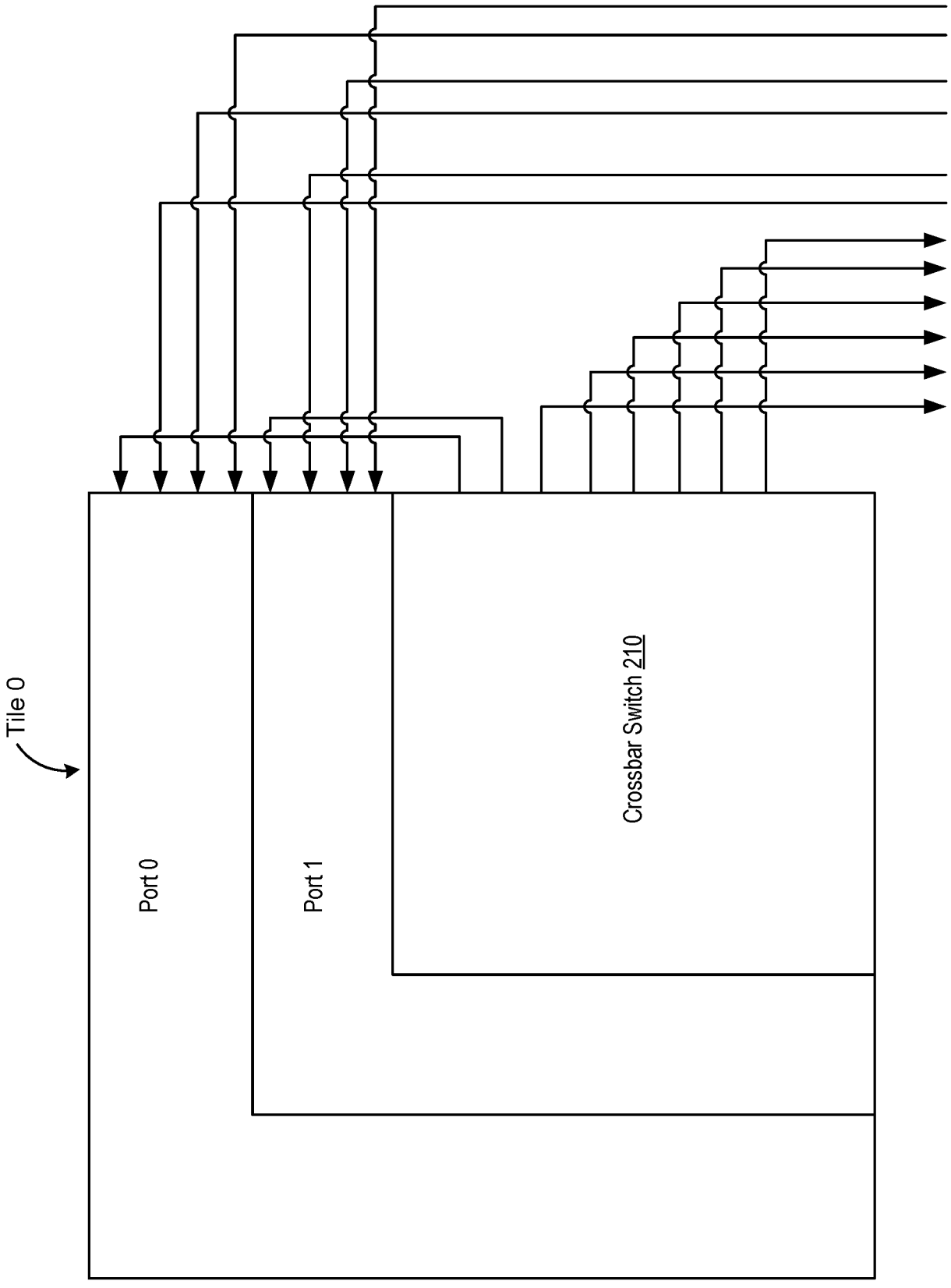

FIG. 3C

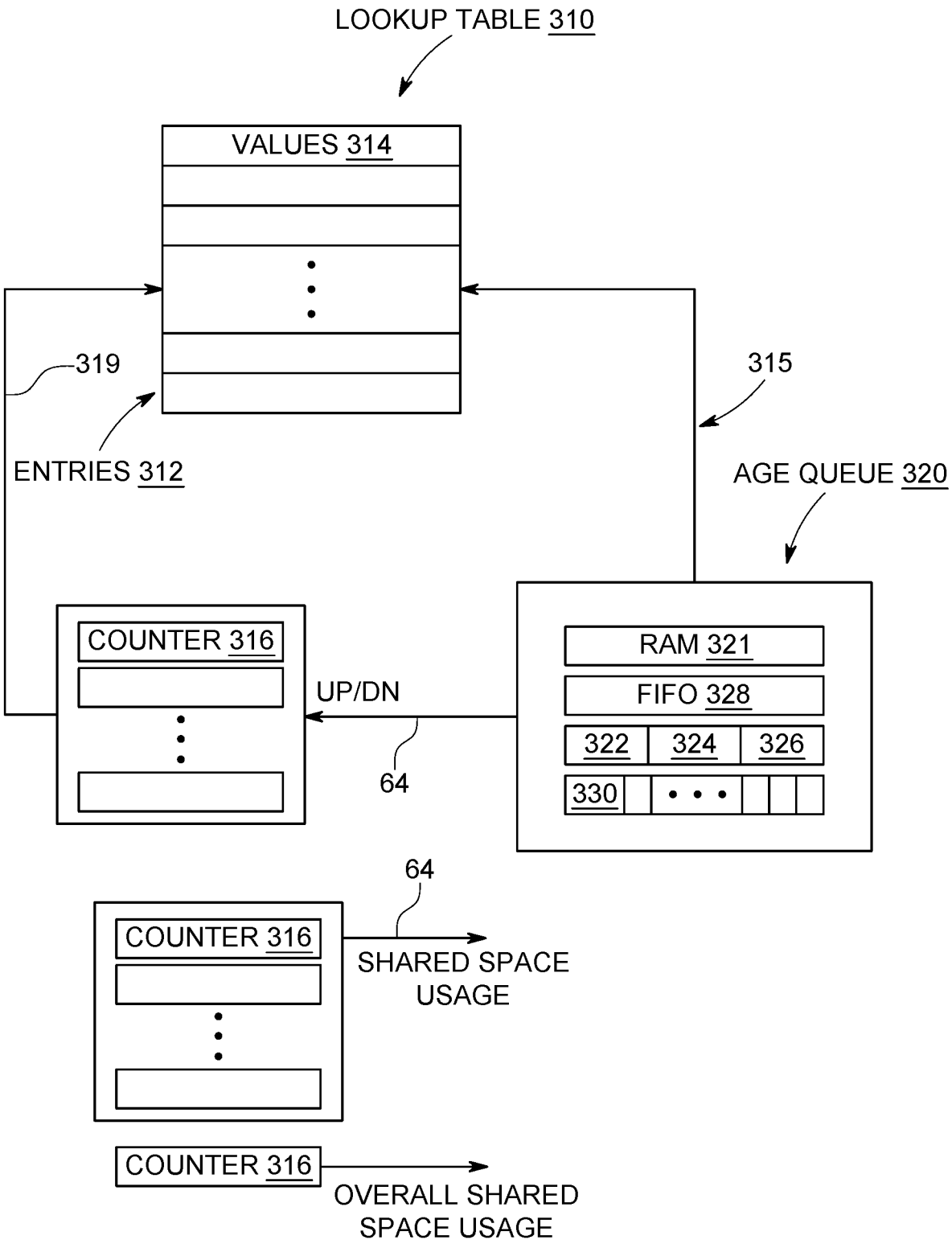
LOOKUP TABLE 310
VALUES 314
319
315
ENTRIES 312
AGE QUEUE 320
COUNTER 316
UP/DN
RAM 321
FIFO 328
322
324
326
330
64
64
COUNTER 316
SHARED SPACE USAGE
COUNTER 316
OVERALL SHARED SPACE USAGE

FIG. 3D

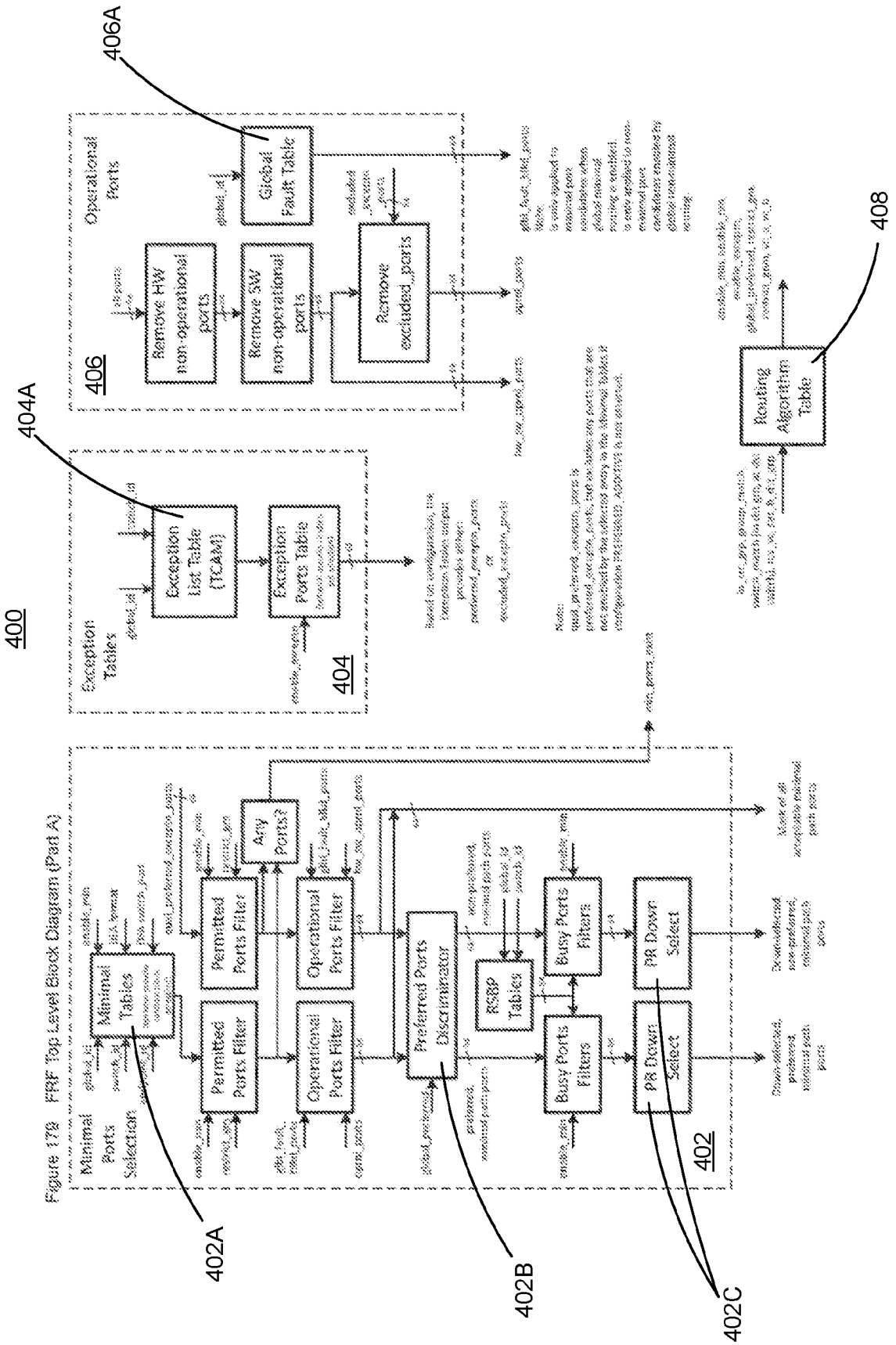
400
Figure 17a FRF Top Level Block Diagram (Part A)
Minimal Ports Selection
Minimal Tables
Permitted Ports Filter
Operational Ports Filter
Any Ports?
Preferred Ports Discriminator
RSBP Tables
Busy Ports Filters
PR Down Select
402
402A
402B
402C
Exception Tables
Exception List Table (TCAM)
Exception Ports Table
404
404A
Operational Ports
406
Remove HW non-operational ports
Remove SW non-operational ports
Remove excluded_ports
Global Fault Table
406A
Routing Algorithm Table
408

FIG. 4A

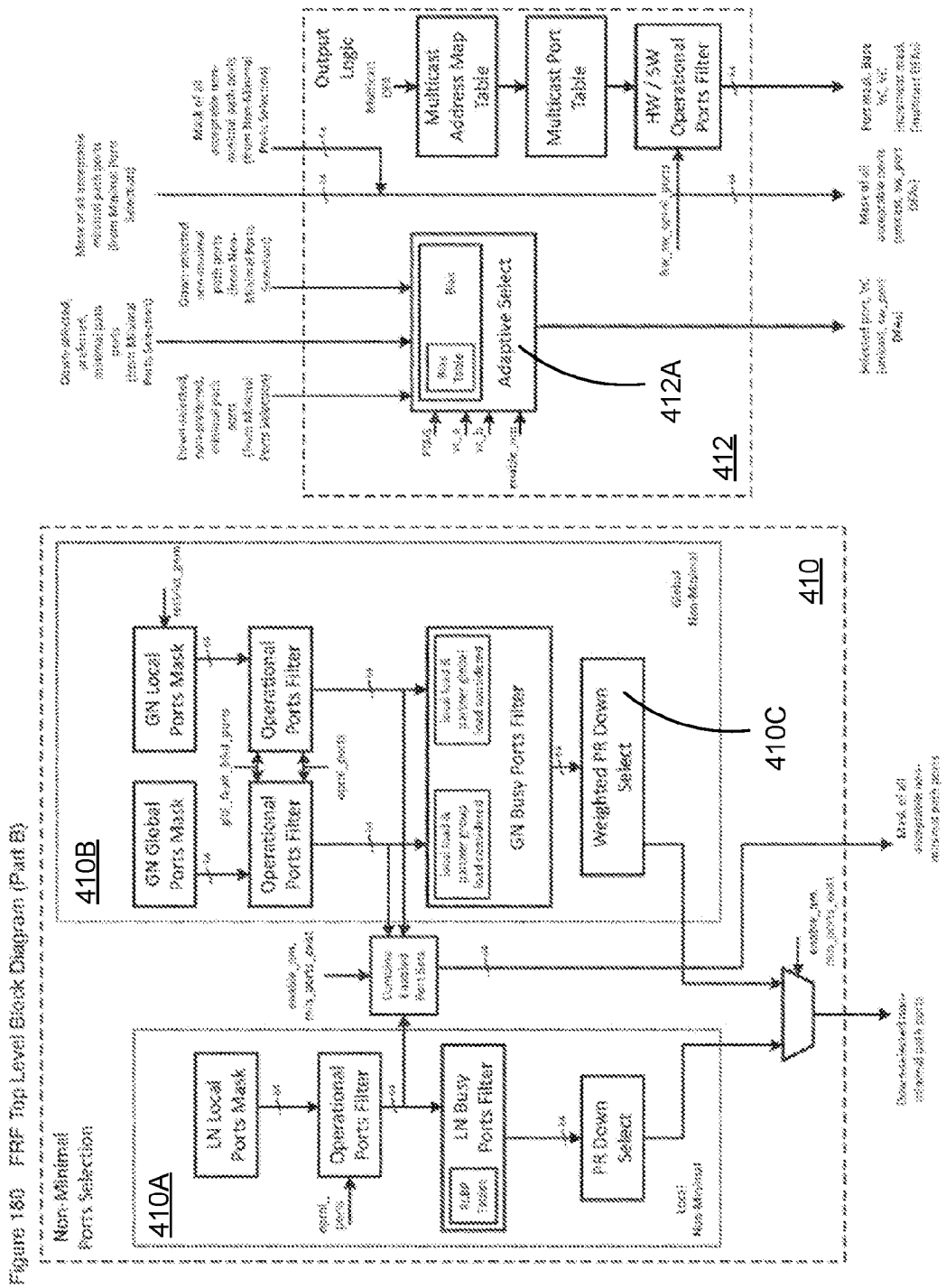
400
Figure 18B FRF Top Level Block Diagram (Part B)
Non-Minimal Ports Selection
410A
LN Local Ports Mask
Operational Ports Filter
LN Busy Ports Filter
PR Down Select
410B
GN Global Ports Mask
GN Local Ports Mask
Operational Ports Filter
Operational Ports Filter
GN Busy Ports Filter
Weighted PR Down Select
410C
410
412
Output Logic
Multicast Address Map Table
Multicast Port Table
HW / SW Operational Ports Filter
Adaptive Select
412A

FIG. 4B

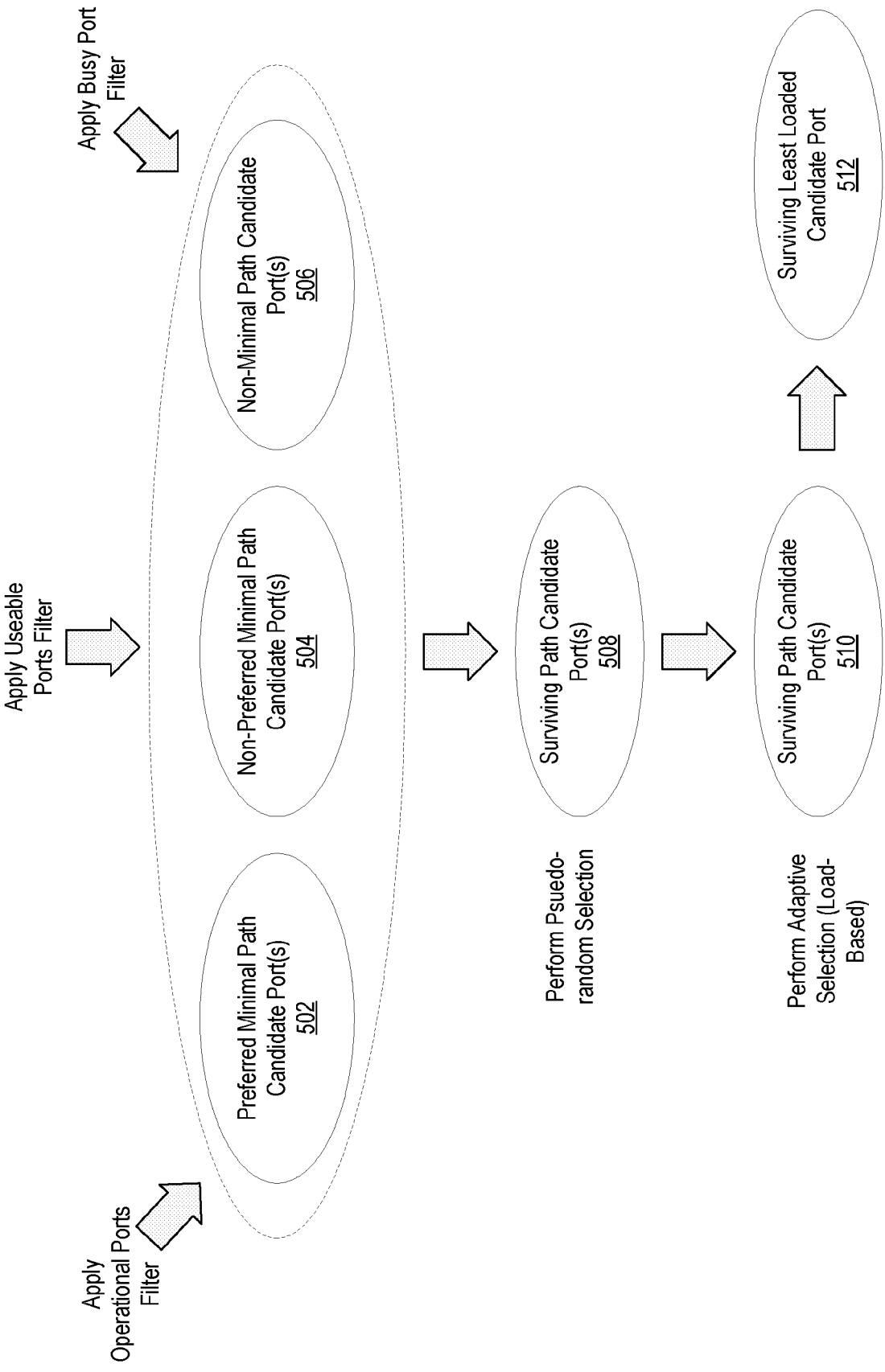
Apply Busy Port Filter
Apply Useable Ports Filter
Apply Operational Ports Filter
Preferred Minimal Path Candidate Port(s) 502
Non-Preferred Minimal Path Candidate Port(s) 504
Non-Minimal Path Candidate Port(s) 506
Perform Psuedo-random Selection
Surviving Path Candidate Port(s) 508
Perform Adaptive Selection (Load-Based)
Surviving Path Candidate Port(s) 510
Surviving Least Loaded Candidate Port 512

FIG. 5

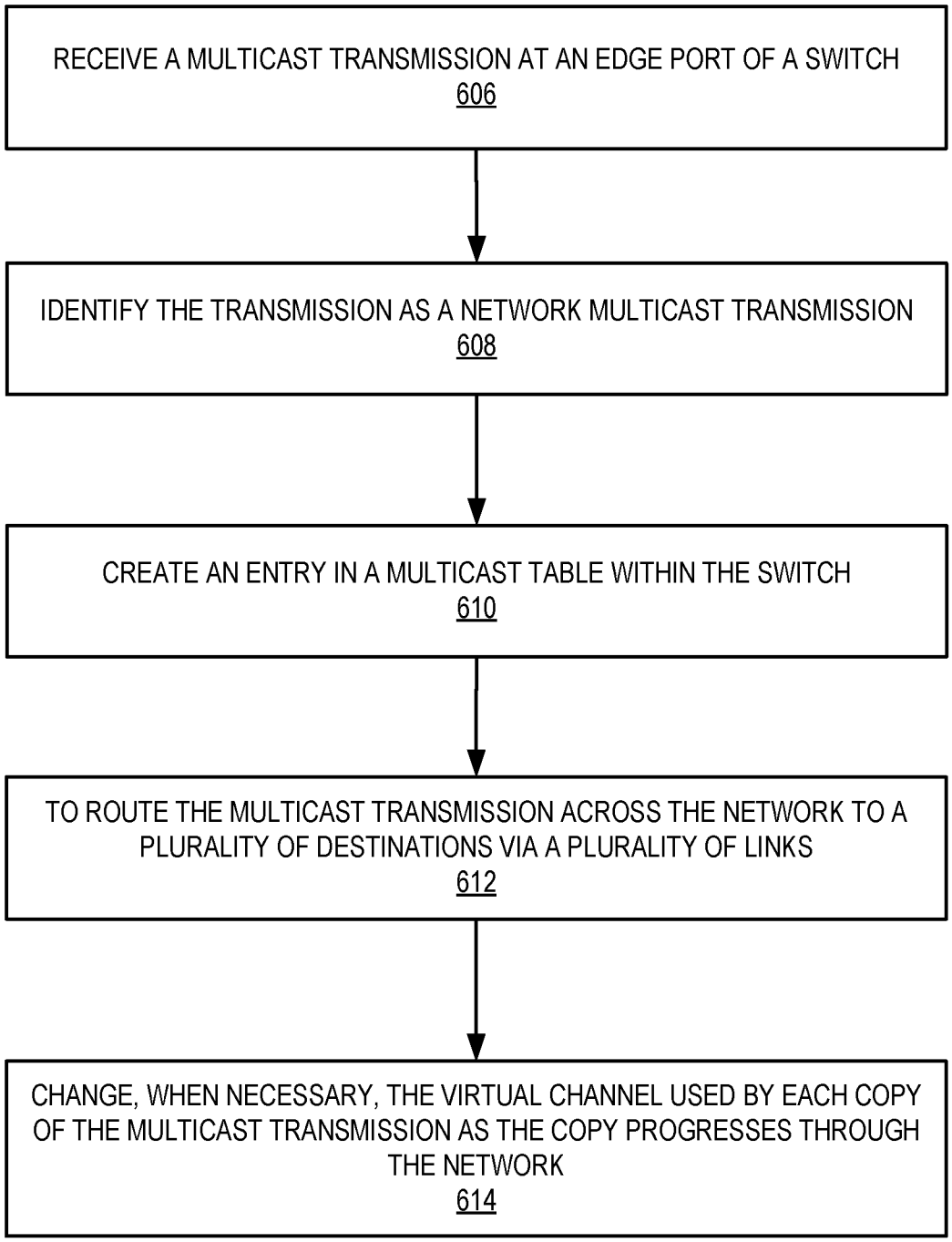
RECEIVE A MULTICAST TRANSMISSION AT AN EDGE PORT OF A SWITCH
606
IDENTIFY THE TRANSMISSION AS A NETWORK MULTICAST TRANSMISSION
608
CREATE AN ENTRY IN A MULTICAST TABLE WITHIN THE SWITCH
610
TO ROUTE THE MULTICAST TRANSMISSION ACROSS THE NETWORK TO A PLURALITY OF DESTINATIONS VIA A PLURALITY OF LINKS
612
CHANGE, WHEN NECESSARY, THE VIRTUAL CHANNEL USED BY EACH COPY OF THE MULTICAST TRANSMISSION AS THE COPY PROGRESSES THROUGH THE NETWORK
614

FIG. 6

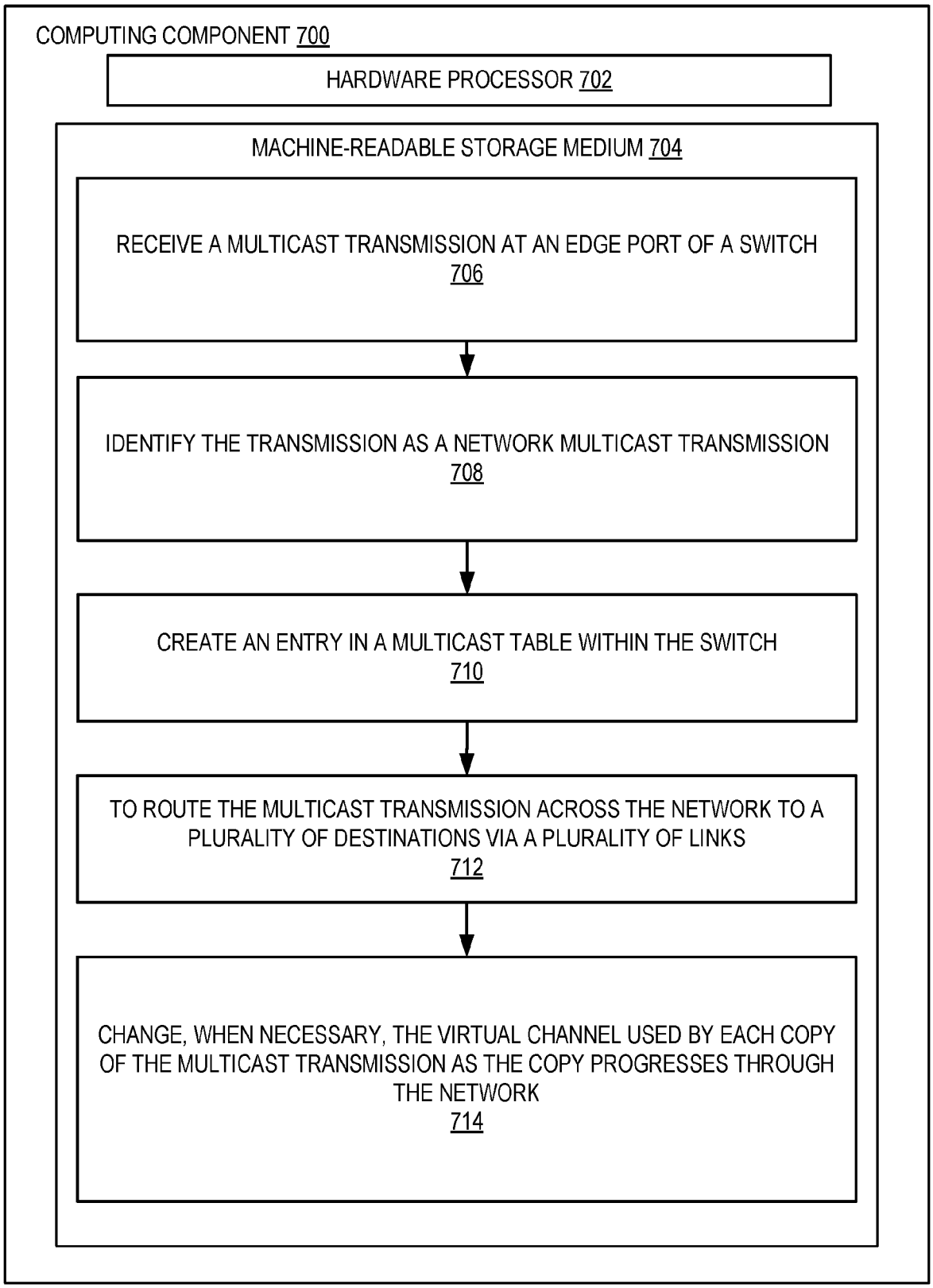
COMPUTING COMPONENT 700
HARDWARE PROCESSOR 702
MACHINE-READABLE STORAGE MEDIUM 704
RECEIVE A MULTICAST TRANSMISSION AT AN EDGE PORT OF A SWITCH 706
IDENTIFY THE TRANSMISSION AS A NETWORK MULTICAST TRANSMISSION 708
CREATE AN ENTRY IN A MULTICAST TABLE WITHIN THE SWITCH 710
TO ROUTE THE MULTICAST TRANSMISSION ACROSS THE NETWORK TO A PLURALITY OF DESTINATIONS VIA A PLURALITY OF LINKS 712
CHANGE, WHEN NECESSARY, THE VIRTUAL CHANNEL USED BY EACH COPY OF THE MULTICAST TRANSMISSION AS THE COPY PROGRESSES THROUGH THE NETWORK 714

FIG. 7

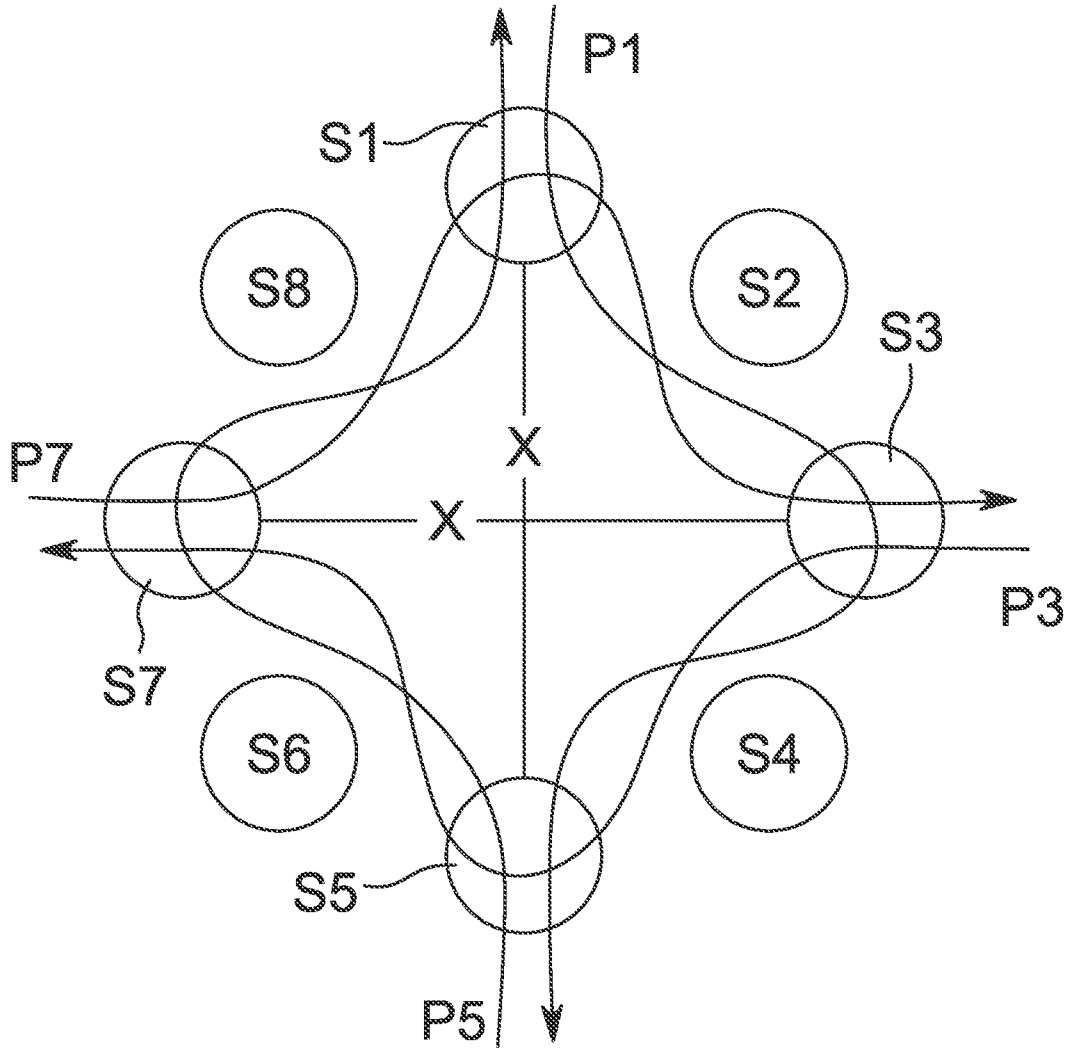

FIG. 8

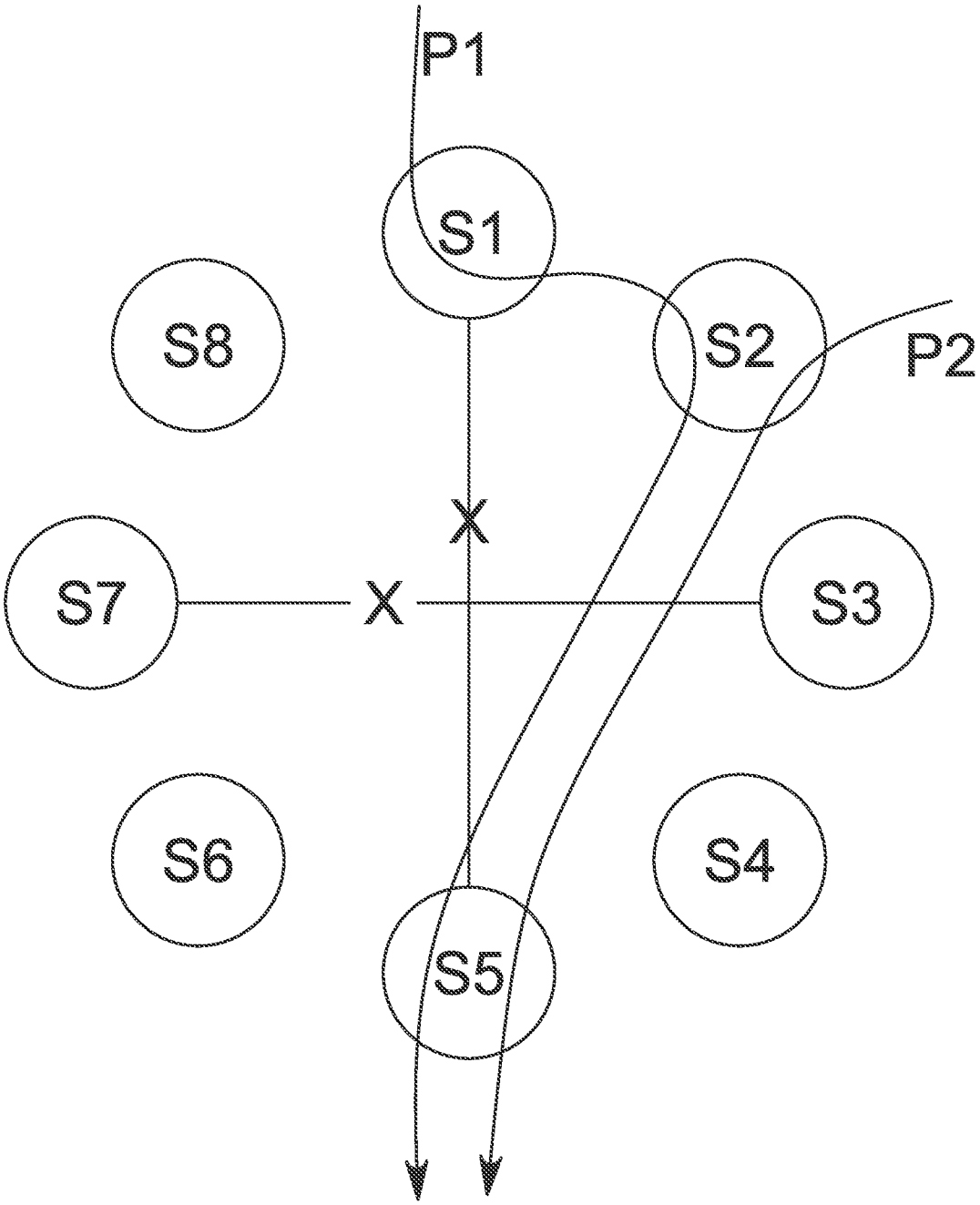

FIG. 9

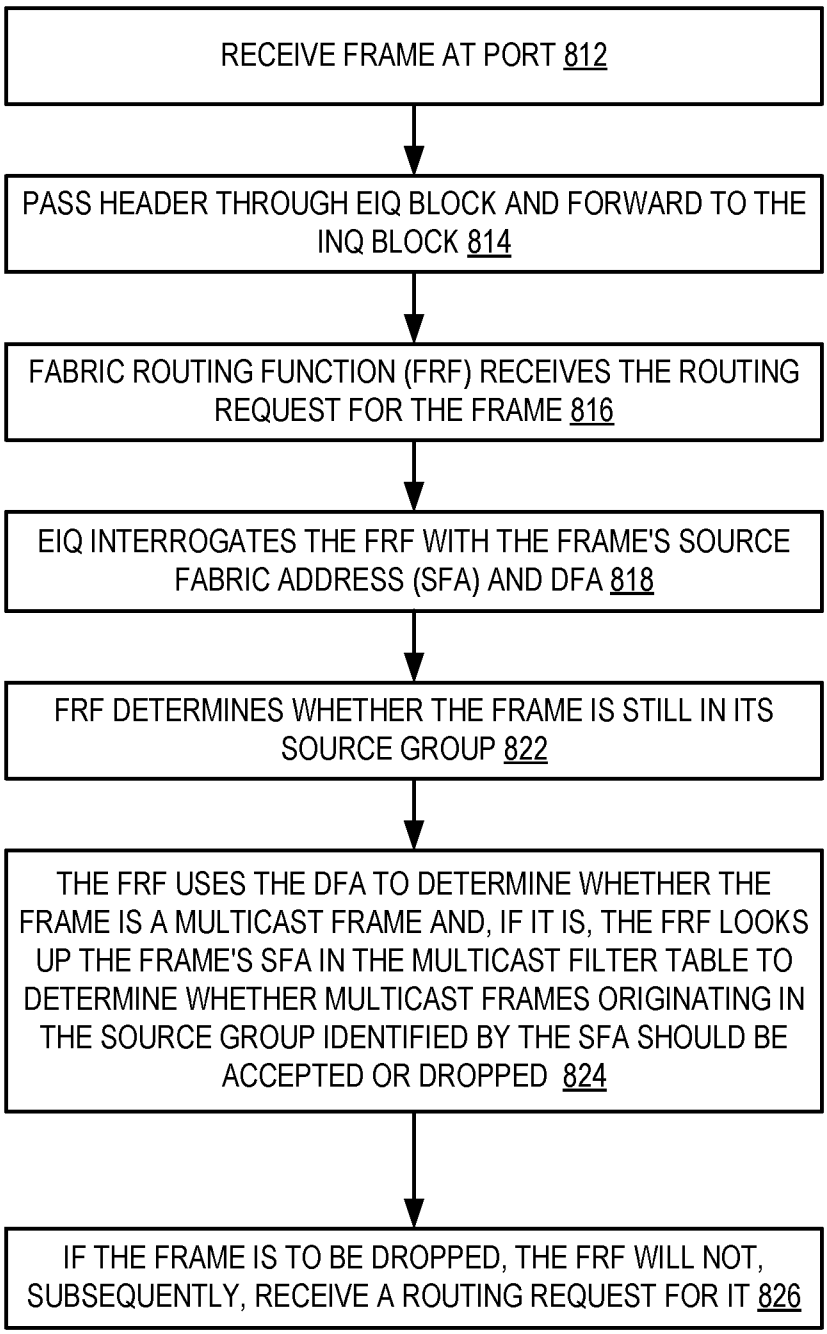
RECEIVE FRAME AT PORT 812
PASS HEADER THROUGH EIQ BLOCK AND FORWARD TO THE INQ BLOCK 814
FABRIC ROUTING FUNCTION (FRF) RECEIVES THE ROUTING REQUEST FOR THE FRAME 816
EIQ INTERROGATES THE FRF WITH THE FRAME'S SOURCE FABRIC ADDRESS (SFA) AND DFA 818
FRF DETERMINES WHETHER THE FRAME IS STILL IN ITS SOURCE GROUP 822
THE FRF USES THE DFA TO DETERMINE WHETHER THE FRAME IS A MULTICAST FRAME AND, IF IT IS, THE FRF LOOKS UP THE FRAME'S SFA IN THE MULTICAST FILTER TABLE TO DETERMINE WHETHER MULTICAST FRAMES ORIGINATING IN THE SOURCE GROUP IDENTIFIED BY THE SFA SHOULD BE ACCEPTED OR DROPPED 824
IF THE FRAME IS TO BE DROPPED, THE FRF WILL NOT, SUBSEQUENTLY, RECEIVE A ROUTING REQUEST FOR IT 826

FIG. 10

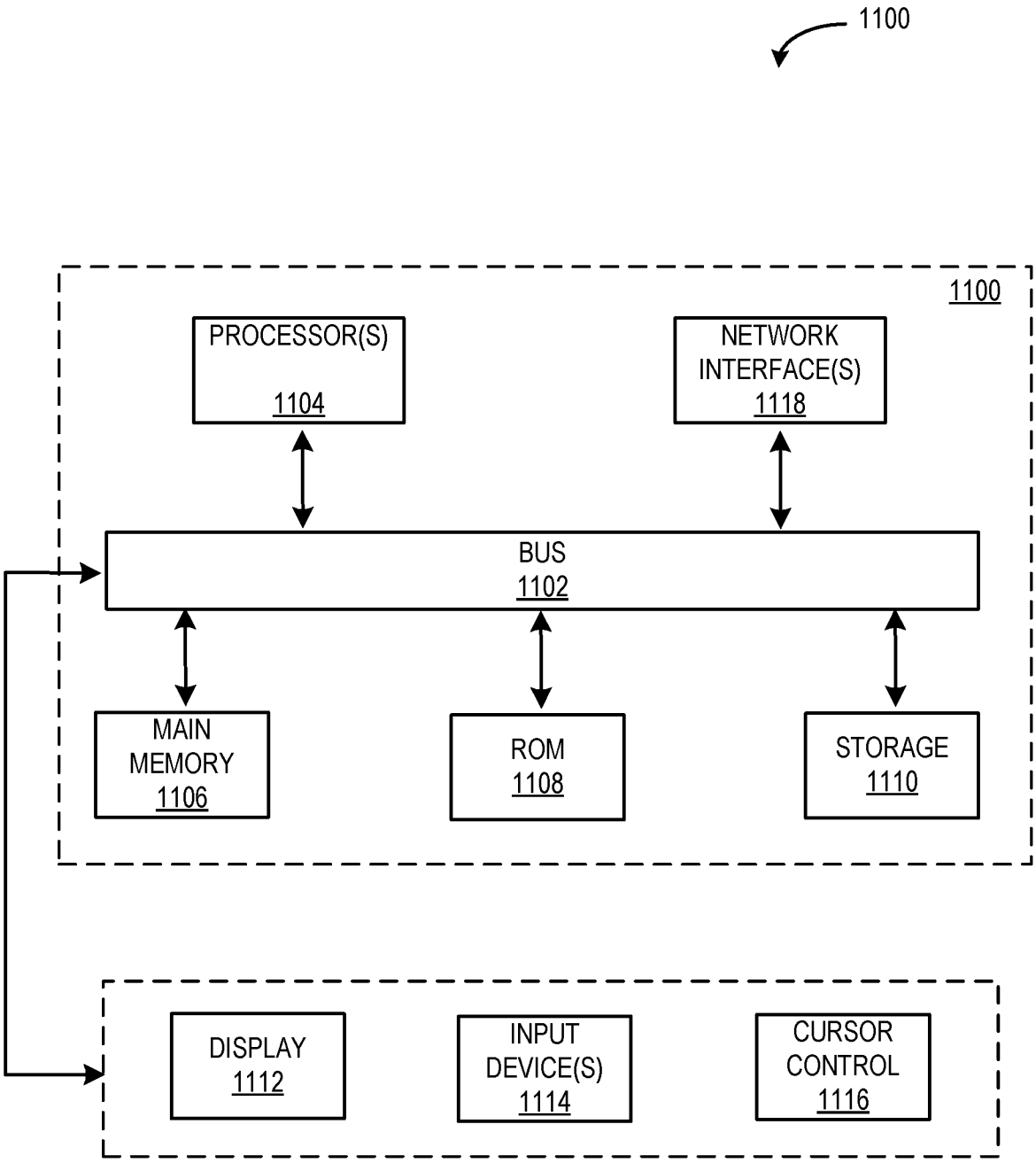
1100
1100
PROCESSOR(S)
1104
NETWORK
INTERFACE(S)
1118
BUS
1102
MAIN
MEMORY
1106
ROM
1108
STORAGE
1110
DISPLAY
1112
INPUT
DEVICE(S)
1114
CURSOR
CONTROL
1116

FIG. 11

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2020/024276

A. CLASSIFICATION OF SUBJECT MATTER

H04L 12/931(2013.01)i, H04L 12/947(2013.01)i, H04L 12/935(2013.01)i, H04L 12/741(2013.01)i, H04L 12/751(2013.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
H04L 12/931; G06F 13/40; G06F 9/455; H04L 12/18; H04L 12/721; H04L 12/733; H04L 1228; H04Q 7/22; H04L 12/947; H04L 12/935; H04L 12/741; H04L 12/751

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
Korean utility models and applications for utility models
Japanese utility models and applications for utility models

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
eKOMPASS(KIPO internal) & keywords: switch, multicast transmission, multicast table, copy, virtual channel

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 6246682 B1 (SUBHASH C. ROY et al.) 12 June 2001 column 3, lines 17-19	1-20
A	WO 2005-094487 A2 (CISCO TECHNOLOGY, INC.) 13 October 2005 claims 10-23	1-20
A	WO 03-019861 A2 (TELEFONAKTIEBOLAGET LM ERICSSON (PUBL)) 06 March 2003 pages 16-18; claim 19; and figure 2	1-20
A	US 2018-0026878 A1 (MELLANOX TECHNOLOGIES TLV LTD. et al.) 25 January 2018 paragraphs [0033]-[0044]; and figure 2	1-20
A	US 2015-0186318 A1 (JOHN KIM et al.) 02 July 2015 paragraphs [0043]-[0045]; and figure 1	1-20

☐ Further documents are listed in the continuation of Box C.

☒ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"D" document cited by the applicant in the international application
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents,such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
13 July 2020 (13.07.2020)	**13 July 2020 (13.07.2020)**

Name and mailing address of the ISA/KR
International Application Division
Korean Intellectual Property Office
189 Cheongsa-ro, Seo-gu, Daejeon, 35208, Republic of Korea
Facsimile No. +82-42-481-8578

Authorized officer
YANG JEONG ROK
Telephone No. +82-42-481-5709

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2020/024276

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 6246682 B1	12/06/2001	CA 2361126 A1	08/09/2000
		CN 1351784 A	29/05/2002
		EP 1163747 A1	19/12/2001
		JP 2002-538718 A	12/11/2002
		WO 00-52858 A1	08/09/2000
WO 2005-094487 A2	13/10/2005	CA 2558338 A1	13/10/2005
		EP 1747690 A2	31/01/2007
		EP 1747690 B1	15/11/2017
		US 2005-0213547 A1	29/09/2005
		US 2007-0280142 A1	06/12/2007
		US 7286853 B2	23/10/2007
		US 8195237 B2	05/06/2012
		WO 2005-094487 A3	27/11/2008
WO 03-019861 A2	06/03/2003	AT 330393 T	15/07/2006
		AU 2002-331230 A1	10/03/2003
		DE 60212404 T2	04/01/2007
		EP 1419614 A2	19/05/2004
		EP 1419614 B1	14/06/2006
		US 2006-0034278 A1	16/02/2006
		US 7606186 B2	20/10/2009
		WO 03-019861 A3	27/11/2003
US 2018-0026878 A1	25/01/2018	None	
US 2015-0186318 A1	02/07/2015	US 10153985 B2	11/12/2018
		US 2010-0049942 A1	25/02/2010
		US 2017-0353401 A1	07/12/2017
		US 9614786 B2	04/04/2017