Adaptive Routing on the New Switch Chip for IBM SP Systems

Bulent Abali[1], Craig B. Stunkel[1], Jay Herring[2], Mohammad Banikazemi[3],
Dhabaleswar K. Panda[3], Cevdet Aykanat[4], and Yucel Aydogan

[1] *IBM T.J. Watson Research Center, P.O.Box 218, Yorktown Heights, NY 10598*

[2] *Server Group, IBM Corporation, Poughkeepsie, NY 12601*

[3] *Dept. of Computer and Information Science, Ohio State University, Columbus, OH 43210*

[4] *Dept. of Computer Engineering, Bilkent University, 06533 Ankara, Turkey*

The IBM RS/6000 SP is one of the most successful commercially available multicomputers. SP owes its success partially to the scalable, high bandwidth, low latency network. This paper describes the architecture of Switch2 switch chip, the recently developed third generation switching element which future IBM RS/6000 SP systems may be based on. Switch2 offers significant enhancements over the existing SP switch chips by incorporating advances in both VLSI technology and interconnection network research. One of the major new features of Switch2 is the incorporation of adaptive routing support into it. We describe the adaptive source routing architecture of the Switch2 chip which is a unique feature of this chip. The performance of the adaptive source routing and oblivious routing for a wide range of system characteristics and traffic patterns is evaluated. It is shown that adaptive source routing outperforms or performs comparably with oblivious routing. We propose two novel algorithms for generating adaptive routes specifications required for enabling the usage of adaptive source routing. The comparison between the cost of these two algorithms and the performance improvement obtained from using these algorithms are discussed. We also propose different output selection functions to be used in switching elements for implementing the adaptive routing. We evaluate and compare the performance of these selection functions and discover that the best selection functions for BMINs are not dependent on the traffic pattern, message size, or system size.

Key Words: adaptive routing, bidirectional multistage interconnection networks, source routing, output selection functions.

1. INTRODUCTION

A high performance interconnection network [7] is a crucial component in multicomputer and clustered systems. The RS/6000 SP is one of the most successful multicomputer systems available today. The SP system's success has been partially due to the scalable and high bandwidth SP interconnect. This paper describes a recently developed third generation switching element which future IBM RS/6000 SP systems may be based on. In this paper this switching element will be referred as the Switch2[1] chip. Switch2 chips are completely functional and have been running in the laboratory for some time. Switch2 offers significant enhancements over existing SP network switch chips. In particular, it introduces a new form of adaptive routing with the potential to significantly improve network bandwidth, and it adds a powerful hardware multicast replication capability.

The most important measure of a computer network is perhaps its aggregate throughput. This throughput measure depends on topology, link speed, and achievable utilization through the individual switching elements. The switch utilization is heavily dependent upon the internal queuing mechanism and the buffer organization within the switches. The Switch2 chip employs the same central buffer strategies successfully employed in current SP network switches to implement output queuing. However, buffer sizes are increased throughout the chip. As a consequence of this Switch2 supports up to 200 meter long links, longer than current SP switches. This in turn allows the construction of richly-connected topologies with a high bisection bandwidth. Switch2 also increases the bandwidth per link to 500 MB/s per direction.

An important design consideration for any interconnection network is the routing mechanism. Routing mechanisms can be subdivided into *route specification* and *routing decisions*, where route specification occurs once at the source, and routing decisions occur at each switch on the path between a source and destination. For route specification, modern networks are almost evenly divided between source routing (in which, for each device on the path, the source node embeds a routing directive within the packet) and destination or logical-address routing (in which only an address or an address offset is embedded). Routing decisions can be classified as adaptive (a routing choice can be made among several options), and non-adaptive or oblivious (there is only one valid choice) [12]. For many networks, it has been shown that adaptivity can increase performance. However, all adaptive routing schemes to date have relied on destination routing.

A unique feature of Switch2 architecture is combining source routing and adaptive routing mechanisms, referred to as *adaptive source routing.* This feature is used to allow control over the degree of adaptivity on a per packet and per switch basis, an important consideration for fault-tolerance, to provide backward compatibility, and to provide support for some protocols that require in-order packet arrival. It is particularly well-suited for the bidirectional multistage interconnections networks (BMINs) used in today's SP networks. BMINs have aggregate bandwidth scaling linearly as the number of nodes increases, as do unidirectional multistage networks.

[1] Note: Switch2 is not an IBM product and no assumptions should be made regarding its availability in the future.

They have smaller diameter than the mesh and torus networks. Fat trees [18] and least common ancestor networks [21] are examples of BMINs.

In the source routing mechanisms the network interface must perform a route table lookup before inserting messages into the network. Route tables are built by various algorithms that use the network topology as an input. In the adaptive source routing scheme, these algorithms must generate routing directives that maximize the degree of adaptivity in the network. In this paper we present two novel algorithms, fully adaptive and partially adaptive algorithms which make a tradeoff between execution time and network performance.

Another important design consideration in adaptive networks is how routing decisions are made within the switching elements. A switching element may determine that more than one output port are available for forwarding a message. The switching element uses an *output selection function* to decide which output port will be used [9, 12]. It has been shown that using a proper output selection function is a key to the good adaptive routing performance. In this paper, we examine output selection functions for SP-like, bidirectional multistage interconnection networks (BMIN).

Previous studies on output selection functions have focused on the mesh and torus networks [22, 13, 14, 29, 30, 11, 4]. To the best of our knowledge, output selection functions for BMINs or MINs have not been studied to this date. We introduce six output selection functions. The performance of adaptive routing under the proposed output selection functions is studied with extensive simulations.

There are a number of other enhancements offered in the Switch2 switch that will not be described in detail in this paper. For example, Switch2 switches provide a powerful hardware multicast replication capability. In addition to this enhancement, Switch2 provides support for high-priority traffic, and locking between individual elements of the switching network becomes asynchronous to help distribute faster oscillator signals.

The major contributions of this paper are as follows:

- We describe the architecture of Switch2, the newly developed third generation switching chip for IBM RS/6000 SP systems, and present the major Switch2 enhancements in comparison with the previous generations of SP switches.
- We compare the performance of the adaptive source routing and oblivious routings for a wide range of system characteristics and traffic patterns.
- We propose two novel algorithms for generating adaptive route specifications. We provide a comparison between the cost and performance of these two algorithms.
- We propose several output selection functions to be used in switch chips for implementing the adaptive routing. We compare the performance of these selection functions and discover that the choice of a selection function for BMINs is not dependent on the traffic pattern, message size, or system size.

In the rest of this paper, we begin with an overview of current SP Switch architecture and concepts in Section 2. Section 3 gives an overview of the major Switch2 enhancements. In Section 4, we discuss the new adaptive routing technique used in Switch2. In Section 5, we describe the adaptive route generation algorithms, and in Section 6, we describe the output selection functions.

2. BACKGROUND

In this section we examine several properties of the current generation of SP switching networks before discussing the major Switch2 enhancements.

The current generation of SP networks is called the *SP Switch*. The basic organization of an SP switch chip is shown in Fig. 1. This is an 8-ported switch chip that achieves link bandwidths of 150 MB/s per direction, and it is architecturally similar to the 40 MB/s *Vulcan* switch chip [28] used in the original IBM SP1 [1, 27] and SP2 [25] High Performance Switch networks.

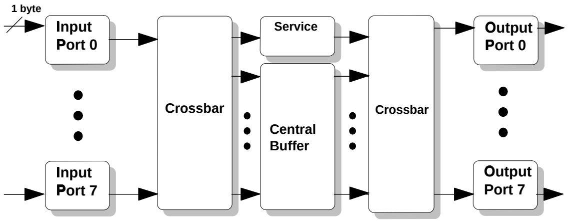

FIG. 1. The SP switch chip organization.

2.1. Flow-Control and Switching

SP switching networks use *wormhole routing* [10], which consists of both flit-based flow-control and cut-through switching. In flit-based flow-control, a packet is broken into small units called flits, or flow-control digits. A flit is the smallest unit of a packet that can be accepted or rejected by the flow-control mechanism. In SP systems, each output port of each network device maintains a credit count, and can send one flit for each credit that it holds. Each time a downstream input port releases buffer space for one flit, it sends a new credit back to the upstream output port. In this manner multiple flits can proceed at full bandwidth over a long link, given sufficient buffer space for flow-control at the input port.

In cut-through switching, once the packet header (which contains the route) is received, the route is decoded, and an immediate attempt is made to forward the packet to the desired output port. In SP systems, this forwarding occurs via a large, dynamically-allocated central buffer.

2.2. Central Buffer (Buffered Wormhole Routing)

In most wormhole implementations, when a packet cannot be routed to the desired output port, it is blocked in place which prevents any subsequent packet at that input port from being forwarded. SP switches contain a large central buffer in which packet flits can be stored when an output port is busy. In most cases, the entire packet can be stored in this central buffer. This frees the input port to route the next packet in the input buffer. This technique is termed *buffered wormhole routing* [25].

The central buffer implements multiple FIFO queues of packets, one queue for each output port. This is a space-efficient form of output queuing, which is known to be superior to input queuing methods. In addition, central buffer space is dynamically shared according to the demand from the input and output ports, which further increases maximum utilization in the switching element.

The central buffer must be able to store and fetch packet flits at the same aggregate rate as the input and output ports. Under worst-case conditions, all 8 input ports and 8 output ports may be receiving and sending one flit per switch cycle. Thus the central buffer must be able to store and/or fetch 8 flits every cycle. Rather than building a very expensive and slow 16-port SRAM to handle this load, it is far more efficient to provide an 8-flit-wide 2-port SRAM. This 8-flit central buffer width is called a *chunk*. Before storing into the central buffer, an input port therefore collects one chunk of packet data.

Now that we gave an overview of the central buffer and its queuing mechanisms, we are ready to examine the high-level data flow of SP switch chips.

2.3. Basic Data Flow

When an input port receives a flit, it stores it in the input buffer. Concurrently, the input port may also be fetching flits out of this buffer. When a packet header is fetched from the buffer, the packet route is decoded. Once an entire 8-flit chunk has been collected in the input buffer or the last flit of the packet has been received, a store request is made to the central buffer to forward the packet chunk. The central buffer must eventually grant every received store request, and it does so on an LRU basis. When a central buffer store request is granted, the chunk is sent to the central buffer. To ensure that no queue in the buffer can be starved, each queue maintains an exclusively reserved *emergency* chunk. This emergency chunk can only be used by a *critical chunk* data that is immediately needed by the output port. For instance, if only 2 chunks of a 3-chunk packet have been stored in the central buffer, and both of these chunks have been subsequently fetched by the output port, then the third chunk is critical, and is eligible to be stored in the emergency chunk. Critical chunk requests are considered before non-critical requests.

2.4. Routing

In the first two generations of SP switch chips, the 8-port switches have had no topological assumptions built into them. The switches are source-routed, which means that the routing decision for each switch chip is embedded within the packet. Each switch chip decodes–and then discards–the first route field in the packet. Each route field contains a 3-bit encoded output port ID. Source routing allows different packets going to the same destination to traverse different paths based upon the embedded route. It is straightforward to bypass faulty links or switches using source routing, and, in principle, no switch chip setup is required before sending a source-routed packet through that switch chip. To better understand the typical routing strategy, let us describe the current SP topology.

2.5. Topology

Even though the source routing technique assumes no particular topology, certain topologies are more scalable in aggregate bandwidth than others. Because SP systems are positioned to be highly scalable, the choice of topology is critical. From the earliest SP1 machines, only bidirectional multistage interconnection networks (BMINs) have been built. An example of a 16-node SP network is shown in Fig. 2, where each block represents a switch chip in the network. As do uni-directional multistage networks, these networks scale aggregate bandwidth linearly as the number of nodes increases. BMINs reward communication locality, and require no *virtual channels* to avoid deadlock among packets in the network. Fat-trees [18] and least common ancestor networks [21] are examples of BMINs.

Figure 2 shows the path of two packets sent from node P13. The first message is destined for P8, and need only be routed through one switch chip. The second message is destined for P6, and traverses 3 switch chips. Note that any of the right-hand-side switch chips could have served as the second switch chip in the path of this packet. In all cases a *minimal* (shortest-path) route is selected. Note that the node numbering in Fig. 2 is the same as the 16 node SP system's node numbering and it is an artifact of the physical wire layout.

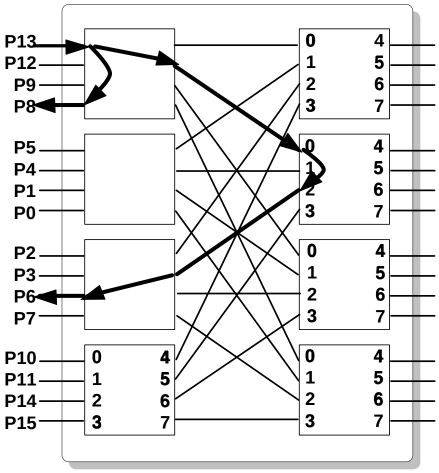

FIG. 2. A 16-way SP switching network.

3. OVERVIEW OF SWITCH2 ENHANCEMENTS

In this section we discuss differences between the 8-port Switch2 switch chip and current SP switches. Just as its predecessors, external links carry one byte of data (plus clock and control) in each direction, but at a signaling rate of 500 Mb/s per signal. The clock is sent at half-speed, and is used to capture the data on each edge (every 2 ns) upon arrival at the downstream input port. Internally, Switch2 executes on 8 ns cycles, and therefore has 4-byte-wide data paths. Table 1 compares and summarizes many of the attributes of the first two generations of SP switch chips and the Switch2 chip.

Flits are 4 bytes, and Switch2 switch chips provide 512 bytes (128 flits) per input buffer. At 500 MB/s, and taking into account the round-trip delay to send a flit and receive the returned credit, a Switch2 switch chip can support link lengths up to 200 meters. This length is beyond the range of a typical copper link operating at 500 Mb/s per signal, but optics could potentially be used to extend the usable range if desired. We have 25 meter copper link cables working in the laboratory.

TABLE 1
Properties of SP switch chips and the Switch2 switch chip

Property	SP2 HPS (Vulcan)	SP Switch (current)	Switch2
(Uni) link BW	40 MB/s	150 MB/s	500 MB/s
(Uni) link width	11 signals	11 signals	10 signals
Internal cycle time	25 ns	13.3 ns	8 ns
Internal data path	1 byte	2 bytes	4 bytes
Input buffer size	32 bytes	128 bytes	1024 bytes
Central buffer size	1 KBytes	4 KBytes	8 KBytes
Max link length	25 meters	25 meters	200 meters
Hardware Multicast	No	No	Yes
Adaptive Routing	No	No	Yes
Clocking	Globally synchronous	Globally synchronous	Asynchronous (plesiochronous)

3.1. Central Buffer Enhancements

The Switch2 chip enhances the central buffer by maintaining two queues per output port: a high-priority queue and a normal-priority queue. This often allows high-priority packets to bypass queued normal-priority packets. Switch2 also increases the size of the central buffer to 8 KBytes. Finally, Switch2 pipelines access to the central buffer to reduce data path width and latency to and from the buffer [16]. Logically, the central buffer remains one chunk wide, but it is divided into 4 *banks*. Bank 0 receives the first 2 flits of a stored chunk. Two cycles later, bank 1 receives the next two flits, and so on. This bank arrangement reduces cost without affecting bandwidth through the central buffer.

3.2. Other Enhancements

The Switch2 chip adds two new forms of routing (adaptive and multicast). Unlike the switches in current SP machines, these new forms of routing do depend upon topology. We should emphasize that although these functions exist in the Switch2 switching hardware, the use of both functions is dependent upon the software that controls the route format at the network interface. Any product-level use of these new capabilities might be phased in over time.

The last row in Table 1 mentions clocking. All SP networks to date have been driven by a common oscillator. (There are multiple oscillators in the system but only one master oscillator is used at any point in time.) As clock speeds increase, the reliable and redundant distribution of a global clock becomes more difficult. Therefore, Switch2 uses an asynchronous clocking scheme. Data travels with a clock signal over the link, and the clock is used to latch the data. Clocking is *plesiochronous*, meaning that all clocks in the system are approximately the same frequency. This fact can be used to minimize the number of idle cycles needed to avoid input buffer overflow [26].

4. ADAPTIVE SOURCE ROUTING

One of the most important architectural enhancement in the Switch2 chip is the adaptive source routing. In non-adaptive (or oblivious) routing, a fixed routing decision is made for traveling between a source and a destination node. Each switch must forward the message through a fixed output port regardless of the traffic. If the pre-determined output port is busy, the packet must wait for the busy port to clear although there may be other available ports leading to the destination. In contrast, adaptive routing methods allow for more than one choice of output ports. A switch may forward the packet to one of many output ports making the routing decision on the fly as the packet header is decoded. The decision for selecting an output port is made by an *output selection function.* This will be described in detail in Section 6.

While the Switch2 chip supports adaptive routing, for many reasons including backward compatibility Switch2 must also continue to support oblivious routing and even a mix of oblivious and adaptive routing. Existing adaptive routing schemes rely on distributed routing techniques (destination routing) as opposed to source routing. For example, in a regular 2-D mesh, an intermediate switch can, based upon the destination address or a distance vector from the destination, choose from more than one output port that will lead to that destination. Fault-tolerance becomes an issue for destination routing networks, as faulty links make it possible to select a port that eventually leads to a dead-end. This problem is sometimes addressed by adding virtual channels for adaptive routing and/or by allowing non-minimal routing. A more powerful fault-tolerance capability can be achieved through a form of destination routing known as table-lookup routing, in which each packet carries a logical address that is used to index a route lookup table inside each switch. The tables throughout the network can be configured to avoid dead-ends. However, in all of these schemes, the source processor has no per-packet control over the route or the amount of adaptivity. Instead, the routing decisions are made by the switches distributed throughout the network. In SP systems, we desire to maintain control over adaptivity at the source node, and to use minimal routing without any need for virtual channels. For instance, in-order packet arrival may be required in some cases but not others. All of these objectives are met by adaptive source routing [3].

4.1. Route Specification

In adaptive source routing, just as in the current SP source routing technique, the source node embeds a single route field for each switch chip to be traversed in the path. However, for adaptive route fields, a 4-bit field is used to specify the *permissible set* of output ports instead of a single output port. To see why 4 bits are sufficient (instead of 8 bits each corresponding to an output port,) recall the SP BMIN topology shown in Fig. 2. Each packet in an SP network traverses a minimal path, traveling away from the source node to some least common ancestor (LCA) of the source and destination nodes. In Fig. 2 for the packet traveling to P6, any of the right-hand-side switches is an LCA of the P13-P6 pair. The packet then travels downward from LCA to the destination. To maintain a minimal route, there is only one path going downward from an LCA. Thus all adaptivity must occur on the upward path to the LCA. Each routing decision along the upward path involves at most the 4 output ports on the upper side of the switch chip. As shown in Fig. 3, the entrance port guides the interpretation of the route field. On the upward path to LCA, a packet entering the switch chip from one side must always exit from the other side. Therefore, the adaptive routing field needs to specify only 4 bits each corresponding to an output port on the opposite side of the input port.

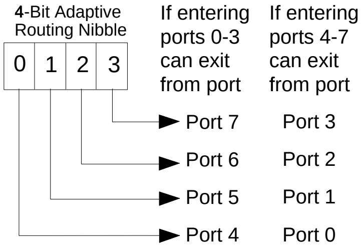
4-Bit Adaptive Routing Nibble
0
1
2
3
If entering ports 0-3 can exit from port
If entering ports 4-7 can exit from port
Port 7
Port 3
Port 6
Port 2
Port 5
Port 1
Port 4
Port 0

FIG. 3. Adaptive routing nibble bits set to '1' indicate which ports a message can exit from.

Because there is only one route field per switch stage in the packet header, all LCA switches reachable via the adaptive choices must have the same downward view of the destinations. Fortunately, this is an inherent property of fat-tree networks, and in most SP networks this is also the case. For instance, to send a packet from P13 to P6, the same port (port 2) of each LCA switch leads to the destination (Fig. 2). There are few SP network topologies in which only a subset of LCAs satisfy this property. For these networks, the adaptivity on the upward path can be restricted with adaptive source routing so that only this subset of LCA switches will be reached. Note that a 4-bit adaptive route field is not sufficient to determine the LCA switch (the point at which the packet may turn and continue non-adaptively). Therefore Switch2 packets also carry an initial field in the first flit which maintains the number of adaptive routes in the packet. All remaining route fields in the packets are then non-adaptive, and carry the same format as previous generations of SP switch chips. This has the added advantage that the routes generated for older SP systems can be trivially embedded in Switch2 packets.

For a complete example, consider the 128 node SP network in Fig. 4. From source node S to destination node D there are 64 different routes one of which a message may follow to avoid network congestion: In the first three stages of switches (in the upward path to the LCAs) a message has a choice of four output ports per stage thereby having a total choice of $4 \times 4 \times 4 = 64$ different routes. Once the message moves on to the fourth stage (in the downward path from LCA), it has no choices left but must exit from a fixed port in each stage. Therefore, in overall there are a total of 64 possible routes from node S to D. This example in Fig. 4 illustrates the power of adaptive routing in BMIN where a single oblivious route is replaced by 64 possible routes, one of which will be selected dynamically depending on the traffic.

In Fig. 4 since there are six stages of switch chips from S to D, the 4-bit routing fields in the message packet will be in the form of $A, R1, R2, R3, R4, R5, R6$. $A = 3$

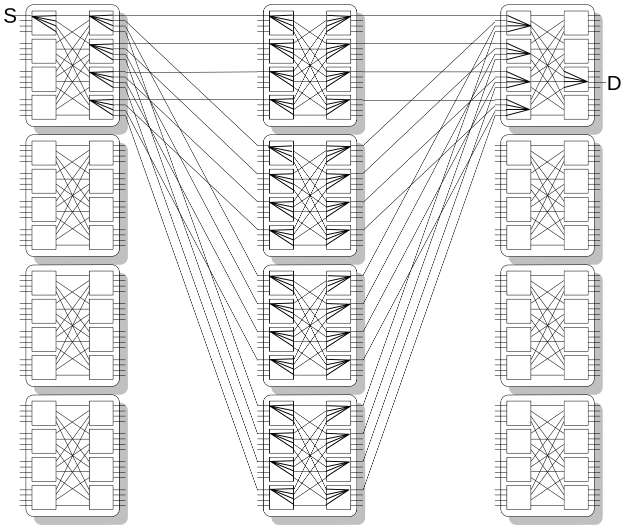

FIG. 4. A 128-node SP switching network. A subset of the entire set of links is shown.

indicates that there are three adaptive routing fields followed by the non-adaptive routing fields. Each switch chip decrements A while consuming an adaptive routing field. The source node may set a field as $R = 1111$ instructing the corresponding switch that any one of the four output ports may be used. For *maximum* adaptivity the first three route fields can be set to $R1 = 1111$, $R2 = 1111$, and $R3 = 1111$. For *partial* adaptivity the source node S may send the message with some bits of $R1, R2$, or $R3$ of the packet header turned off. Thus, the number of distinct routes a packet may follow is:

$$N_{\text{routes}} = |R_1| \times |R_2| \times \cdots\cdots \times |R_{n-1}| \times |R_n|$$

where $|R_i|$ is defined as the number of bits set in the routing field R_i.

Being able to specify the degree of adaptivity on a per packet basis and at the source node is one of the unique advantages of the Switch2 chip. This architecture is not only backward compatible with previous generations of SP networks but it also allows a mix of adaptive and oblivious traffic to coexist in the same network. For in-order-delivery, consecutive packets may be sent obliviously through a single route ($N_{\text{routes}} = 1$), while other packets not sensitive to in-order delivery may be sent with maximum adaptivity ($N_{\text{routes}} = \text{max}$). For network security or partitioning, packets may be sent partially adaptive to avoid certain regions of the network ($1 < N_{\text{routes}} < \text{max}$).

5. ROUTE GENERATION ALGORITHMS

The use of adaptive routing in Switch2 depends on the software that creates the route tables. Given the network topology the route generation algorithm must compute adaptive routing headers for each destination. In this section, we discuss two different algorithms for generating the adaptive routing headers. The first algorithm generates fully adaptive headers (as described in Section 4.1). The second algorithm generates partially adaptive headers by restricting adaptivity to the first stage of the network. The partially adaptive algorithm has the advantage of being a simple extension of the current generation SP algorithm. It's execution is also faster than the fully adaptive algorithm. Network simulations show that the three algorithms: oblivious, partially adaptive, and fully adaptive have increasing performance in the given order. Thus, a tradeoff may be made between the execution time and performance depending on the network size and system requirements. Another possibility is to create hybrid route tables that may contain fully adaptive, partially adaptive, oblivious routing headers, since the Switch2 chip allows a mix of adaptive and oblivious traffic to coexist in the same network.

5.1. Fully Adaptive Routes

The fully adaptive route generation algorithm maximizes the amount of adaptivity (N_{routes}) of packet headers. While it may appear straightforward in Figs. 2–4 to generate fully adaptive route headers, the problem of maximizing the adaptivity is complicated by irregularities in the network topology. The presence of faulty links and switches makes this problem even more challenging.

In the proposed algorithm (Fig. 6), we adopt a graph-theoretical approach. For each source-destination processor pair (s, d), we construct a multistage *routability graph* G_R which enumerates all possible minimal paths from s to d. Then, using G_R we construct a multistage *solution graph* G_S which enumerates every feasible adaptive route solution (route-word encoding) from s to d. Finally, we find a maximally adaptive route through a dynamic-programming formulation on G_S.

We represent the topology of the network by a directed graph $G_T = (V_T, E_T)$, which is referred here as the *topology graph*. Vertex set V_T contains two types of nodes, processor nodes and switching nodes. Edge set E_T represents the interconnections between the switching nodes and between the processor and switching nodes. Each edge $e =< u, v >$ has an m-bit binary label $\ell_T[e]$ whose 1-bit position denotes the output port number of the switching vertex u it is sourced from. Multiple edges between a pair of vertices in the same direction are coalesced into a single edge by bitwise OR'ing the labels of the individual edges.

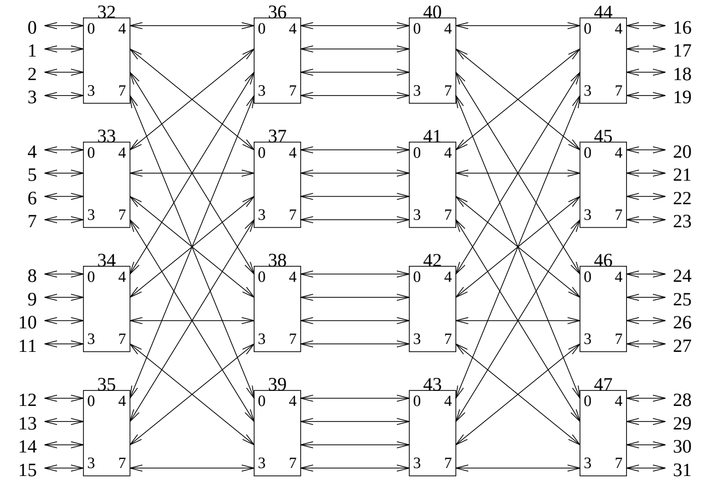

FIG. 5. A 32-processor node bidirectional multistage network (BMIN)

```
GENERATE_ROUTES($G_T, s$)
$G_\pi(s) \leftarrow$ BFS1($G_T, s$);
for each destination processor $d \neq s$ do
    $G_R(s,d) \leftarrow$ BFS2($G_\pi, d$);
    $G_S(s,d) \leftarrow$ ALL_FEASIBLE_ROUTES($G_R$);
    $RT(s,d) \leftarrow$ MAX_ADAPTIVE_ROUTE($G_S$);
return the routing table $RT$
```

FIG. 6. Generating routes from a source processor s to other processors.

While describing the algorithm we will use an example of the 32-node SP network shown in Fig. 5. The respective topology graph $G_T = (V_T, E_T)$ contains 48 vertices. Processors are indexed from 0 to 31 and switches are indexed from 32 to 47.

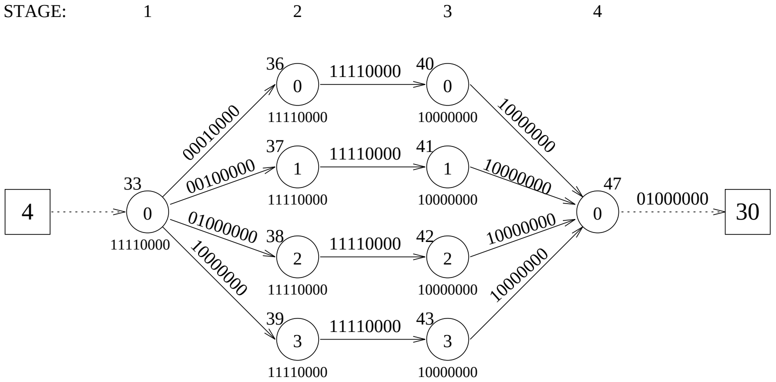

FIG. 7. Routability graph $G_R(4,30) = (V_R, E_R)$ for source-destination processor pair $(4,30)$.

5.1.1. *Routability Graph*

Routability graph $G_R(s,d) = (V_R, E_R)$ for a given source–destination processor pair (s,d) is a directed n-stage graph [15], where n denotes the shortest path distance from s to d. Here, distance refers to the number of switching elements in a route. $G_R(s,d)$ contains only switching nodes and it is a subgraph of G_T with all switching nodes and edges that are not in the minimal paths from s to d eliminated. Each vertex $v \in V_R$ has an m-bit ($m = 8$ for SP switch) binary attribute $ports_R[v]$ whose 1-bit positions denote the output ports allowed during routing to reach the destination processor. Here, we will assume that the route specifications are 8-bits wide, rather than the 4-bit specification used in the actual hardware (Section 4.1). Vertices V_R^i in each network stage i are indexed from 0 to $|V_R^i| - 1$, for $i = 1, 2, \ldots, n$. Both first and last stages contain a single vertex v_0^1 and v_0^n which correspond to the source and destination switches, respectively. Routing word R_n for reaching the destination processor d from the destination switch is known in advance. Edges exist only between the vertices of the successive stages. That is, $< u, v > \in E_R$ only if $u \in V_R^i$ and $v \in V_R^{i+1}$ for some $i = 1, 2, \ldots, n-1$. Each edge $e =< u, v >$ is labeled with $\ell_R[e]$ similar to G_T. Fig. 7 shows the routability graph for the (s,d) pair (4,30) of the network given in Fig. 5.

```
BFS1($G_T, s$)
for each vertex $v \in V_T - \{s\}$ do
    $color[v] \leftarrow$ WHITE;
$color[s] \leftarrow$ GRAY; $depth[s] \leftarrow 0$; $Q \leftarrow \{s\}$;
while $Q \neq \emptyset$ do
    $u \leftarrow$ DEQUEUE($Q$);
    for each $v \in Adj_T[u]$ do
        if $color[v] =$ WHITE then
            $color[v] \leftarrow$ GRAY; $depth[v] \leftarrow depth[u] + 1$;
            $\pi[v] \leftarrow \{u\}$; $\ell_\pi[< v, u >] \leftarrow \ell_T[< u, v >]$;
            ENQUEUE($Q, v$);
        elseif $color[v] =$ GRAY and $depth[v] = depth[u] + 1$ then
            $\pi[v] \leftarrow \pi[v] \cup \{u\}$; $\ell_\pi[< v, u >] \leftarrow \ell_T[< u, v >]$
    $color[u] \leftarrow$ BLACK
return $G_\pi(s) = (V_\pi, E_\pi)$, where
        $V_\pi = \{v \in V_T : color[v] =$ BLACK $\} - \{s\}$
        $E_\pi = \{< u, v >: u, v \in V_\pi$ and $v \in \pi[u]\}$
```

FIG. 8. BFS–like algorithm proposed to construct predecessors subgraph $G_\pi(s) = (V_\pi, E_\pi)$ for source processor s.

$G_R(s, d)$ is constructed in two steps. In the first step, we use a modified Breadth-First-Search (BFS) like algorithm—BFS1(G_T, s) in Fig. 8—on G_T starting from vertex s. BFS1(G_T, s) constructs *predecessors subgraph* $G_\pi(s) = (V_\pi, E_\pi)$ which is different from the breadth-first tree generated during conventional BFS [6]. In $G_\pi(s)$, V_π contains all processor nodes of G_T, and those switching nodes of G_T which are in the minimal route from source processor s to multiple destination processors other than s. Similarly, E_π contains those edges (links) of G_T in reverse direction which are in the minimal route from s to at least one destination processor other than s. As seen in Fig. 8, each node $v \in V_\pi$ contains multiple parents stored in its $\pi[v]$ field which also denotes the adjacency list of vertex v in G_π.

In the second step, we run another BFS–like algorithm—BFS2(G_π, d) in Fig. 9—on $G_\pi(s)$ starting from d. In Fig. 9, each non–black (white and gray) vertex $v \in V_\pi$ encountered while scanning the adjacency list of a vertex u of depth j from the destination switch constitutes an edge from vertex v to u at stages i and $i + 1$ of $G_R(s, d)$, respectively, where $i = n - j - 1$.

5.1.2. Solution Graph

Solution graph $G_S(s, d) = (V_S, E_S)$ is a multistage graph with the same number of stages as in routability graph G_R. Vertex set V_S^i of G_S at stage i is a subset of the power set of V_R^i of G_R excluding the empty set, i.e., $V_S^i \subseteq 2^{V_R^i} - \emptyset$. Both first and last stages (stages 1 and n) of G_S contain a single vertex v_1^1 and v_1^n which correspond to the source and destination switches, respectively.

In a straightforward implementation, we allocate $2^{|V_R^i|} - 1$ vertices for constructing the stage i vertices of G_S. Allocated vertices of each stage are indexed from 1 to $2^{|V_R^i|} - 1$. The positions of the 1-bits in the binary representation of each vertex $v_k^i \in V_S^i$ determine the subset $\mathcal{S}_k^i$ of vertices (switches) at stage i of G_R that it represents. For example, at stage 2 of the routability graph shown in Fig. 7, there are 4 vertices 0, 1, 2, 3 corresponding to switches 36, 37, 38, 39, respectively.

```
BFS2($G_\pi, d$)
$dsw \leftarrow \pi[d]$;   /* $dsw$ is the destination switch */
for each vertex $v \in V_\pi - \{dsw\}$ do
  $color[v] \leftarrow$ WHITE;
$color[dsw] \leftarrow$ GRAY;  $Q \leftarrow \{dsw\}$;
while $Q \neq \emptyset$ do
  $u \leftarrow$ DEQUEUE($Q$);
  for each $v \in \pi[u]$ do
    if $color[v] =$ WHITE then
      $color[v] \leftarrow$ GRAY;   $Adj_R[v] \leftarrow \{u\}$;
      $ports_R[v] \leftarrow \ell_\pi[< u, v >]$;   $stage_R[v] \leftarrow depth_\pi[v]$;
      ENQUEUE($Q, v$);
    else  /* $color[v]$ should be GRAY */
      $Adj_R[v] \leftarrow Adj_R[v] \cup \{u\}$;
      $ports_R[v] \leftarrow ports_R[v] \bigvee \ell_\pi[< u, v >]$;  /* "$\bigvee$": bitwise OR */
    $\ell_R[< v, u >] \leftarrow \ell_\pi[< u, v >]$;
  $color[u] \leftarrow$ BLACK
return $G_R(s, d) = (V_R, E_R)$, where
        $V_R = \{v \in V_\pi : color[v] =$ BLACK $\}$
        $E_R = \{< u, v >: u, v \in V_R$ and $v \in Adj_R[u]\}$
```

FIG. 9. BFS–like algorithm proposed to construct routability graph $G_R(s, d) = (V_R, E_R)$ for source-destination processor pair (s, d).

Therefore, at stage 2 of the corresponding solution graph shown in Fig. 11, there are $2^4 - 1 = 15$ vertices labeled in binary 0001 through 1111, representing all possible subsets of the set of 4 vertices in the routability graph. For example, $v_{13}^2 \in V_S^2$ of G_S represents the vertex subset $\mathcal{S}_{13}^2 = \{s_0^2, s_2^2, s_3^2\} = \{36, 38, 39\}$ of V_R^2 since $13 =$ "1101" in binary.

A vertex $v_k^i \in V_S^i$ only if there exists at least one feasible adaptive route $R_1 R_2 \ldots R_n$ which can forward the message to exactly one of the switches in $\mathcal{S}_k^i$ through $R_1 R_2 \ldots R_{i-1}$. Here, feasibility refers to the fact that the remaining $n - i$ route words $R_i R_{i+1} \ldots R_n$ can forward the message packets at all switches in $\mathcal{S}_k^i$ to the destination processor. An edge $e =< v_k^i, v_\ell^{i+1} > \in E_S$ only if there exists at least one feasible route whose stage-i routing word R_i forwards the message packets at all switches in $\mathcal{S}_k^i$ to exactly one of the switches in $\mathcal{S}_\ell^{i+1}$. Each edge e is associated with a label $\ell_S[e]$ corresponding to the maximal routing word which achieves the above mentioned message forwarding. Here, maximality refers to the routing word with maximum number of 1's. Hence, the sequence of labels (routing words) on the edges of each distinct path from v_1^1 to v_1^n constitutes a feasible route from the source switch to the destination switch. Each one of these feasible routes appended with the last stage routing word R_n from the destination switch to the destination processor constitutes a feasible adaptive route from the source processor to the destination processor.

For example, in Fig. 11, vertex $v_5^2 = 0101$ at stage 2 has an edge with label 11110000 to $v_5^3 = 0101$. This edge indicates that from the set of switches $\mathcal{S}_5^2 = \{s_0^2, s_2^2\} = \{36, 38\}$ at stage 2 of G_R, we can reach the set of switches $\mathcal{S}_5^3 = \{s_0^3, s_2^3\} = \{40, 42\}$ at stage 3 by routing word 11110000 which can be verified from Fig. 7.

```
ALL_FEASIBLE_ROUTES($G_R$)
1  for $i \leftarrow 1$ to $n$ do
2    allocate $|V_S^i| = 2^{|V_R^i|} - 1$ nodes $\{v_j^i\}_{j=1}^{|V_S^i|}$ for $V_S^i$
3    for $j \leftarrow 1$ to $|V_S^i|$ do
4      $mark[v_j^i] \leftarrow$ INACTIVE;
5      $InAdj[v_j^i] \leftarrow \emptyset$;  $OutAdj[v_j^i] \leftarrow \emptyset$;
6  $mark[v_1^1] \leftarrow$ ACTIVE;
7  for $i \leftarrow 1$ to $n-1$ do
8    for each ACTIVE vertex $a \in V_S^i$ do
9      $P_a \leftarrow \bigwedge_{u \in \mathcal{S}_a} ports_R[u]$; /* "$\bigwedge$": bitwise AND */
10     for each possible $R_i \in$ {1-bit position combinations of $P_a$} do
11       $S_R^{i+1} \leftarrow \emptyset$;
12       for each stage $i$ vertex $u \in \mathcal{S}_a$ of $V_R^i$ do
13         for each $w \in Adj_R[u]$ such that $\ell_R[<u,w>] \bigwedge R_i \neq 0$ do
14           $S_R^{i+1} \leftarrow S_R^{i+1} \cup \{w\}$;
15       find the vertex $v \in V_S^{i+1}$ where $\mathcal{S}_v = S_R^{i+1}$;
16       if $v \notin OutAdj[a]$ then
17         $mark[v] \leftarrow$ ACTIVE;
18         $OutAdj[a] \leftarrow OutAdj[a] \cup \{v\}$;  $InAdj[v] \leftarrow InAdj[v] \cup \{a\}$;
19         $\ell_S[<a,v>] \leftarrow R_i$;
20       else /* edge $<a,v>$ already exists */
21         $\ell_S[<a,v>] \leftarrow \ell_S[<a,v>] \bigvee R_i$; /* "$\bigvee$": bitwise OR */
22 for $i \leftarrow n-1$ downto 2 do
23   for each ACTIVE vertex $a \in V_S^i$ do
24     if $OutAdj[a] = \emptyset$ then
25       for each $u \in InAdj[a]$ do
26         remove vertex $a$ from $OutAdj[u]$; /* remove edge $<u,a>$ */
27       $InAdj[a] \leftarrow \emptyset$;  $mark[a] \leftarrow$ INACTIVE;
28 return $G_S(s,d) = (V_S, E_S)$, where
         $V_S = \{v : mark[v] =$ ACTIVE $\}$
         $E_S = \{<u,v>: u,v \in V_S \text{ and } v \in OutAdj[u]\}$
```

FIG. 10. Algorithm for generating solution graph $G_S(s,d) = (V_S, E_S)$ for source-destination processor pair (s,d).

Figure 10 illustrates the pseudo-code of the algorithm for creating the solution graph. Here, *InAdj* and *OutAdj* denote the adjacency lists of the vertices for their incoming and outgoing edges in G_S, respectively.

The second outer *for-loop* (lines 7–21) performs a forward pass over the vertex stages starting from the only active vertex v_1^1 at stage 1 which corresponds to the source switch. In this *for-loop*, only active vertices are processed at each stage i to determine the active vertices at the following stage $i+1$ and create the edges between the active vertices at stages i and $i+1$. At line 9, P_a is an m-bit binary number whose 1-bit positions correspond to the common ports of the switches in $\mathcal{S}_a$ which can be used to reach the destination. In the *for-loop* at lines 10–21, all possible route words corresponding to the 1-bit positions of P_a are enumerated and processed. For a P_a with $1 \leq k \leq m$ 1-bits, $2^k - 1$ route words are generated by fixing the bit positions corresponding to the 0-bit positions of P_a to all 0's and enumerating $2^k - 1$ distinct non-zero binary numbers from 1 to $2^k - 1$ on the bit positions corresponding to the 1-bit positions of P_a. The set $S_R^{i+1} \subseteq V_R^{i+1}$ of switches reached from the switch set $\mathcal{S}_a \subseteq V_R^i$ by routing word R_i is constructed

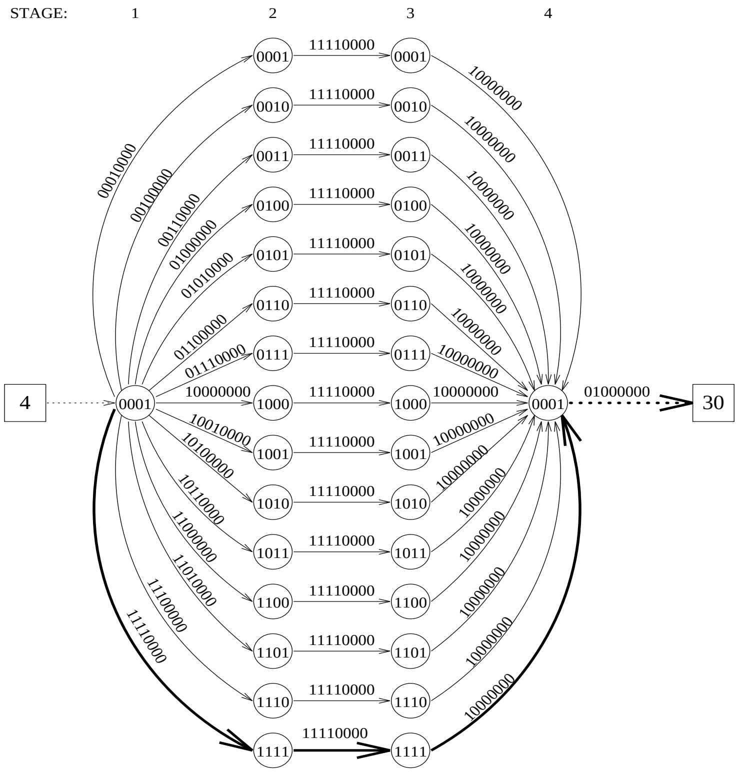

FIG. 11. Solution graph $G_S(4, 30) = (V_S, E_S)$ for source-destination processor pair $(4, 30)$.

in the *for-loop* at lines 12–14. The search operation at line 15 can be performed in constant time by exploiting the proposed vertex encoding in G_R and G_S. The *if-clause* at lines 16–19, adds edge $e =< a, v >$ to E_S, activates vertex v at stage $i + 1$ of G_S, and initializes route-word label $\ell_S[e]$ of edge e. The *else* clause at lines 20–21 ensures the maximality of the route-word label $\ell_S[e]$.

G_S generated at the end of the second outer *for-loop* (lines 7–21) may contain vertices and edges which are not involved in any feasible solution path from the source to the destination because of the vertices at later stages which do not have any outgoing edges. These infeasible vertices and edges are removed in the last outer *for-loop* (lines 22–27) in order to reduce the computational complexity of the dynamic programming algorithm to be executed in the next phase. The backward processing order over the vertex stages of G_S ensures the feasibility of all remaining vertices and edges.

The number of vertices in each stage of G_S is a power of 2, which may be extremely large in some networks. However, only a small fraction of those vertices are utilized in large networks and remaining vertices do not result in any routing solutions. Therefore, for such stages of G_R, we used the ***digital search tree*** data structure in G_S to reduce the space complexity, rather than allocating storage for all possible vertices. The binary representation of the vertex numbers of G_S are used as search keys in the digital search tree. Digital search tree adds only a constant complexity to the algorithm, but reduces space complexity considerably.

```
MAX_ADAPTIVE_ROUTE($G_S$)
$\text{ADP}[v_1^n] \leftarrow 1$;
for $i \leftarrow n-1$ downto 1 do
  for each vertex $u \in V_S^i$ do
    $\text{ADP}[u] \leftarrow 0$;
    for each $v \in OutAdj[u]$ do
      $adp \leftarrow |\ell_S[< u, v >]| \times$ ADP $[v]$;
      if $adp >\text{ADP}[u]$ then
        $\text{ADP}[u] \leftarrow adp$;
        $next[u] \leftarrow v$;
$u \leftarrow v_1^1$;
for $i \leftarrow 1$ to $n-1$ do
  $v \leftarrow next[u]$;
  $R_i \leftarrow \ell_S[< u, v >]$;
  $u \leftarrow v$;
$R_n \leftarrow \ell_S[< v_1^n, d >]$;   /* $d$ : destination processor */
return $R = R_1 R_2 \ldots R_{n-1} R_n$ with $N_{\text{path}} = \text{ADP}[v_1^1]$
```

FIG. 12. Algorithm for determining maximum adaptive route in an n-stage solution graph $G_S(s, d) = (V_S, E_S)$.

5.1.3. *Maximizing Adaptivity*

Once the solution graph is created, a maximally adaptive route may be found by finding a path from source to destination node in the solution graph that maximizes the *product* of the adaptivity values of edges. The adaptivity of an edge $e \in E_S$ is defined as the number of 1-bits (i.e., $|\ell_S[e]|$) in its edge label $\ell_S[e]$, representing the number of *common* output port choices of the switches in $\mathcal{S}_a$ that can be used to lead the messages at those switches to the destination. Hence, the problem reduces to finding a path from v_1^1 to v_1^n in G_S with maximum adaptivity. For example, in Fig. 11, the top most path has a product cost (adaptivity) of $1 \times 4 \times 1 = 4$ (i.e., $|00010000| \times |11110000| \times |10000000|$), which indicates that the given sequence of routing words results in 4 different routes. The bottom most path, which has a product cost of $4 \times 4 \times 1 = 16$ (i.e., $|11110000| \times |11110000| \times |10000000|$), is the solution with maximum adaptivity since there are no more than 16 distinct shortest paths from processor 4 to 30, as can be verified from Figs. 5 and 7. Therefore, the route header encoding with the maximum adaptivity is $R_1 = 11110000, R_2 = 11110000, R_3 = 10000000$, and $R_4 = 01000000$ in this example.

A *dynamic-programming* [6, 15] formulation for an n-stage solution graph G_S is obtained by first noticing that every source to destination path is a result of a sequence of $n-2$ decisions. The i-th decision involves determining which vertex in V_S^i $(1 < i < n)$ is to be on an optimal path. Let $\text{ADP}[v_k^i]$ denote the adaptivity of an optimal path $p(v_k^i, v_1^n)$ from stage-i vertex $v_k^i \in V_S^i$ to destination switch v_1^n. Then, the optimal substructure property gives the recursive formulation

$$\text{ADP}[v_k^i] = \max_{<v_k^i, v_\ell^{i+1}> \in E_S} \{|\ell_S[< v_k^i, v_\ell^{i+1} >]| \times \text{ADP}[v_\ell^{i+1}]\} \tag{1}$$

The adaptivity of optimal routes from all vertices of G_S can easily be computed by performing a backward pass over the vertex stages of G_S as shown in Fig. 12. $\text{ADP}[v_1^1]$ contains the adaptivity value of the optimal routing solution(s) when the first *for-loop* (lines 2–9) terminates. In this *for-loop*, the *next* attribute for each vertex is computed to enable the construction of an optimal routing in the second outer *for-loop* (lines 11–14) by simply following the *next* fields of the vertices in forward direction starting from the source switch at stage 1.

5.2. Performance Evaluation

In this section we present performance evaluation of the adaptive routing on SP-like BMIN topologies. We compare the performance of the adaptive routing to the oblivious routing schemes used in SP systems.

We conducted network simulations based upon a C++ model of SP-like switches. These switches implement *buffered wormhole routing* [25] for flow-control and contain the dynamically-shared central buffer. Under light to medium loading, a switch is typically able to buffer an entire arriving packet when that packet becomes blocked due to output port contention. Thus, in effect, the switch often operates in virtual cut-through [17] fashion, completely removing blocked packets from network links. However, under heavy loading the central buffer may become full, and packets may then be blocked across several switches, just as in wormhole routing [8]. In the Switch2 chip, when more than one output port is idle and permitted for adaptive routing, an output port selected is based on least-recently-used basis. Other output selection functions are examined in detail in Section 6.

All simulations assume an open network model containing idealized processor nodes: the nodes contain an infinite transmit queue buffer, and packet flits are immediately pulled from the network as they arrive. We assume an exponential distribution for message injection time (message arrival time). We apply a range of loading to the network, where a load of 1.0 indicates that each node is injecting packets in the network at the maximum link data rate. Latency curves include input queuing time and are not shown after saturation (steady-state latency is infinite after saturation, assuming infinite input queues). The maximum packet size is 255, and messages longer than 255 bytes are broken into multiple packets before transmission unless otherwise stated.

The open network model makes it possible to stress the network to a far greater degree and cause more contention than might be possible in a real environment. For instance, in the SP systems, the processor software and the network interface hardware control the injection of message packets via strategies such as end-to-end flow control and message interleaving that significantly reduce the possibility of network saturation [24]. Therefore the heavily-loaded simulation results shown here are extremely unlikely to be reproducible in an actual machine. However, the open network model simplifies analysis by removing the complex software and network interface factors, and makes it possible to examine a single issue: the effect of adaptive routing.

In each experiment, we compare adaptive routing with oblivious routing. In current SP systems, each node maintains a route table containing 4 oblivious routes to each destination node. If there are less than 4 unique minimal routes, as when the source and destination nodes are connected to the same switch, then these 4 routes are identical. The SP network interface adapter uses these 4 oblivious routes in a round-robin fashion for consecutive message packets to reduce contention

5.2.1. *Permutation Traffic Simulation*

In this section we investigate the relative performance of adaptive routing when the communication pattern is a static permutation. We test 2 permutations: bit-reversal and transpose. Some permutations such as bit-reversal and transpose often reveal weaknesses of networks better than random traffic. In bit-reversal, a source processor represented in binary by $s_{n-1}s_{n-2}\dots s_1s_0$ sends messages to destination $s_0s_1\dots s_{n-2}s_{n-1}$. In transposes for even n, the destination is $s_{\frac{n}{2}-1}s_{\frac{n}{2}-2}\dots s_1s_0s_{n-1}s_{n-2}\dots s_{\frac{n}{2}+1}s_{\frac{n}{2}}$. We simulate 16-way and 64-way BMIN's. For our simulations the 64-way BMIN is constructed from 4 of the 16-way BMIN's shown in Fig. 2. For each 16-way BMIN, the 16 unused right-side bidirectional links are connected to a separate switch in a 3rd stage of 16 switches. Thus each 3rd stage switch is connected to each 16-way BMIN by one link.

Figure 13 displays simulation results for the bit-reversal permutation on a 16-way BMIN topology. Adaptive routing attained both the lowest latency and the highest saturation bandwidth for this difficult permutation. For our 1-route oblivious routing, each packet traverses a "straight" path to a least common ancestor switch, and then the packet proceeds on the unique path to the destination [1]. This topology has a maximum of 4 distinct paths between pairs of nodes, and therefore the 4-route oblivious routing exercises all the paths between a source and destination as can be seen in Fig. 2. In general, 1-route oblivious routing either performs very well or very poorly depending on the permutation. Its dismal worst-case performance and high variability make it a poor choice for a general routing strategy and it is not being used in current SP systems. Therefore we will not consider it further in this paper.

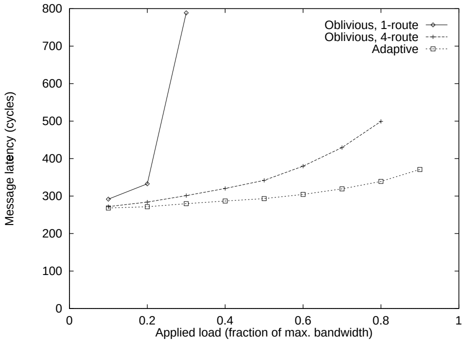

FIG. 13. Bit-reversal permutation traffic on a 16-node BMIN topology

In 16-way topology, adaptive routing and 4-route oblivious routing have exactly the same paths available. However, with adaptive routing any packet traveling a 3-hop path is guaranteed *not* to contend with any other packets while traversing the first switch stage. Because, for this first hop, only four input ports (the "left" input ports in Fig. 2) are contending for the 4 "right" output ports of the switching element (packets cannot enter and then exit the "right" side of the switching element, because the resulting path would not be minimal). Therefore if a packet is entering the "left" side, there are less than or equal to three other input ports currently sending packets to the "right" side, leaving at least one "right" output port open. The 4-route oblivious packets may often contend in the first stage, and this is the major cause of higher latency for this experiment.

Figure 14 illustrates the adaptive routing performance for the transpose permutation. Again adaptive routing performs better, and in fact encounters *no* contention. Because, for transpose permutation, some communication is made directly between pairs of nodes attached to the same switch chip. This results in more output ports available for adaptive routing in the first hop switches. The 4-route oblivious method loses bandwidth principally because of contention in the first hop.

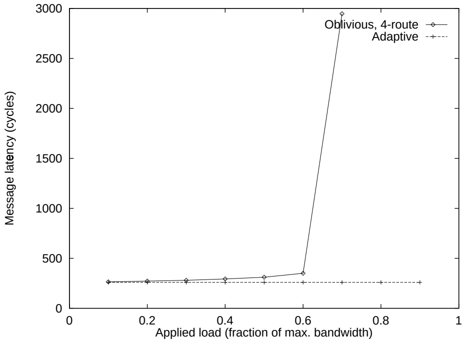

FIG. 14. Transpose permutation traffic on a 16-node BMIN topology

We have established that adaptive routing performs well for several types of permutation traffic on small systems. We now briefly examine the performance of one permutation on a larger system to illustrate that the benefits of adaptive routing extend over a range of system sizes. Figure 15 displays the latency curves for the transpose permutation on a 64-way BMIN topology. Adaptive routing still obtains lower latency and higher saturation throughput, although it no longer achieves the "no contention" curve of the 16-way system. For the 64-way system, packets with source and destination in different 16-way groups will traverse 5 switches and have 16 possible least common ancestors. Thus, 4-route oblivious routing no longer exercises all the possible paths, and we include the 16-route oblivious case to demonstrate that adaptive routing still maintains performance advantages over oblivious routing as the system size grows. Other permutations and system sizes support similar conclusions, but we will not exhaustively detail results to further support these claims here.

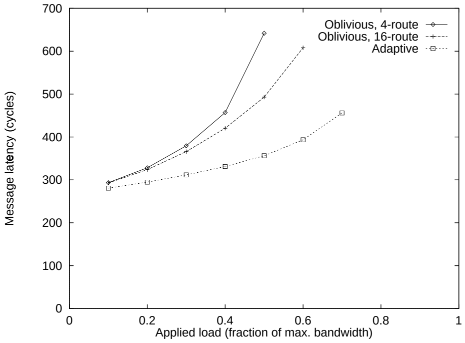

FIG. 15. Transpose permutation traffic on a 64-node BMIN topology

5.2.2. Random traffic simulation

In these experiments, we injected traffic with a uniform destination distribution: for each message, the source randomly chooses any node except itself as the destination. Figure 16 plots message latency for adaptive routing and 4-route oblivious routing for short (100-byte and 500-byte) messages. Latency before saturation is lower and saturation load is higher for adaptive routing, although neither criteria is significantly better than that of oblivious routing.

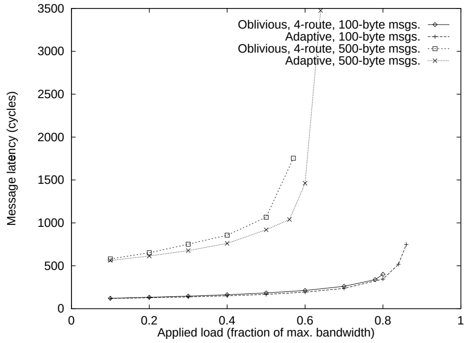

FIG. 16. Short message random traffic on a 16-node BMIN topology

To see how the effect of adaptive routing for random traffic changes with system size, Fig. 17 shows the results of the same short message experiment conducted on the 128-way SP network in Fig. 4. For this larger topology, the positive effects of adaptive routing on random routing are more pronounced. There are more stages in which adaptive routing avoids contention compared with oblivious routing.

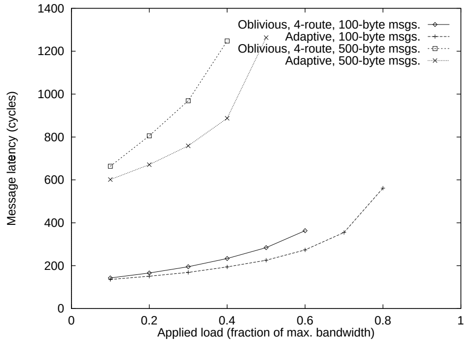

FIG. 17. Short message random traffic on a 128-node BMIN topology

Figure 18 shows the results of the same 128-way experiment conducted with longer (2000-byte and 8000-byte) messages. For longer messages, adaptive routing saturates the network at a 25% higher input load than 4-route oblivious routing. As messages become longer, the effect of hot-spots becomes greater, and adaptive routing tends to shift traffic away from heavily loaded parts of the BMIN network.

We intended to create a hotspot in the network and observe how adaptive routing and oblivious routing performance differ. Each node injects messages to the network with uniform destination distribution except that it also sends a a fixed amount of messages to node 0. This creates a congestion at the node 0 output of

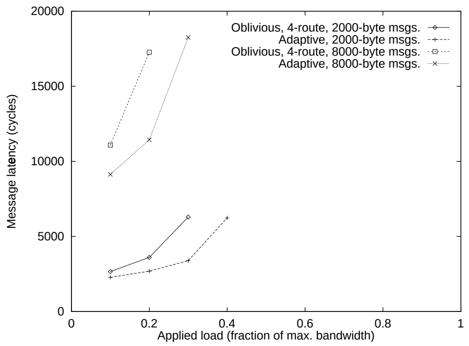

FIG. 18. Long message random traffic on a 128-node BMIN topology

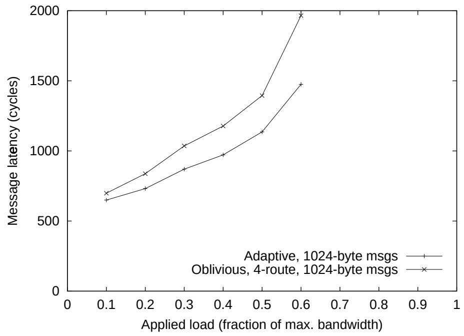

FIG. 19. Random traffic in the presence of a hotspot on a 16-node BMIN topology

the network and at different times depending on the queue lengths this congestion extends back to all the network links and switch queues therefore interfering with messages intended for other destinations. Simulations were performed on the 16-node topology using an SP switch model with 2 flits/cycle links, 4KB central queue size and 1 KB message size. To create a hotspot, each of the nodes 1–15 directs 0.05 units of load to destination node 0. This results in 0.75 units of total load intended for node 0. When network was lightly loaded average latencies for messages intended for node 0 were determined as 1767 and 1865 cycles, and maximum latencies were determined as 9936 and 8876 cycles, for the adaptive and oblivious routing schemes, respectively. This means that on the average there were 3.45 and 3.64 messages queued for node 0 in the network and at maximum there were 19.4 and 17.3 messages queued for node 0 in the network, for the adaptive and oblivious routing schemes, respectively. While the hotspot load to node 0 was fixed, regular traffic to destinations 1–15 was varied from 0.1 to 1.0 units of load and average latencies for adaptive and oblivious routing schemes were measured as a function of the load. Figure 19 shows that adaptive routing still performs better than oblivious routing under hotspot traffic.

To summarize: for BMIN's, adaptive routing is generally superior to oblivious routing for both permutation and random routing. The advantages accrue for two reasons: (1) Adaptive routing does not contribute to contention in the upward path from the processors to the LCA switches, because for this portion of the path each packet always finds an output port link available. (2) Even in the absence of contention, adaptive routing randomizes traffic by choosing among several available output ports.

5.3. Partially Adaptive Routes

We also considered using partially adaptive route headers by restricting adaptivity to the first stage of the network. This algorithm has the advantage of being a simple extension of the current 4-route oblivious routing algorithm used in SP, and it requires only small changes in the SP system software. In this section we describe the partially adaptive routing algorithm and its performance.

In current SP systems, each node maintains a route table containing 4 oblivious routes to each destination node. The SP network interface adapter uses these 4 oblivious routes in a round-robin fashion for consecutive message packets to reduce contention and to reduce the probability of creating "hot-spots" in the network. These 4 oblivious routes to each destination are found by building balanced breadth-first spanning trees [1]. This algorithm is quite simple and we will not describe it here.

Partially adaptive routing algorithm is based on a property of the 4-route oblivious algorithm: analysis of the four routes between source and destination node pairs shows that in many cases the switch output port numbers used in four routes differ only in the first stage. For example, in Fig. 2 consider the source and destination nodes P10 to P6. The four routes respectively use the following packets headers: {4,2,2}, {5,2,2}, {6,2,2} and {7,2,2} where numbers indicate output port numbers. Since, the 4 route headers differ only in the first stage nibble, these may be merged into one adaptive route header. Thus, the partially adaptive header for the example above is {{4,5,6,7},2,2}. Using the notation $A, R1, R2, R3$ of Section 4.1 the partially adaptive route header becomes $A = 1, R1 = 1111, R2 = 0010, R3 = 0010$. The nibble $R1 = 1111$ indicates that the first stage switch chip can adaptively select one of the ports 4,5,6,7.

Table 2 shows the number of four routes that can be converted to one partially adaptive route for different SP networks. Note that the 64-node network has a relatively irregular topology which results in the 4 routes per destination being dissimilar most of the time and therefore not as many 4 routes can be combined. Likewise, the turn restrictions used for deadlock avoidance [23, 2, 14, 20, 19] in 256-node and 512-node networks result in fewer percentage of routes that can be combined into adaptive in comparison to other networks.

Table 3 compares the execution time of of fully adaptive (Sec. 5.1) and partially adaptive algorithms for different system sizes. Figure 20 compares saturation bandwidth of 16-node and 128-node networks for random traffic and for 4-route oblivious routing, partially adaptive routing, and fully adaptive routing methods. For 16-node networks partially adaptive routing is equivalent to fully adaptive routing because adaptivity occurs only in the first stage switch for this network. Results show that the three algorithms: oblivious, partially adaptive, and fully adaptive have increasing performance in the given order. However, partially adaptive algorithm executes much faster for large networks. A tradeoff may be made between execution time and performance depending on the network size and system requirements.

TABLE 2

Number of four-routes that can be combined into partially adaptive routes

System Size	Total Number of Routes	Number of 4 Routes Combined
16	64	48
32	128	112
48	192	176
64	256	48
128	512	496
256	1024	448 (avg.)
512	2048	1456 (avg.)

TABLE 3

Average route table generation times (in seconds) for one processor

Network Size	Partially Adaptive	Fully Adaptive
16	0.08	0.06
64	0.41	0.39
128	1.84	24.98
512	36.61	2394.87

Another important issue which affects the performance of the adaptive routing is the choice of the *output selection function* [9, 12]. In the next section, we present several output selection functions and evaluate their impact on the performance of the system.

6. OUTPUT SELECTION FUNCTIONS

In this section, we focus particularly on the output selection functions for SP-like BMINs [5]. Previous studies on output selection functions have focused on the mesh and torus networks [22, 13, 14, 29, 30, 11, 4]. To the best of our knowledge, output selection functions for BMINs or MINs have not been studied to this date. Based on the architectural characteristics of the Switch2 chip and SP-like BMINs, we introduce six output selection functions with different hardware complexities. These functions are: LRU, MRU, RND, RR, LRUC, and LRUD. The performance of adaptive routing on SP-like BMINs with the proposed output selection functions is studied with simulations.

In the new Switch2 chip when an input port decodes an adaptive route field, if there are more than one output port candidates, the input port requests service from all of these output ports. All free output ports will grant the service request.

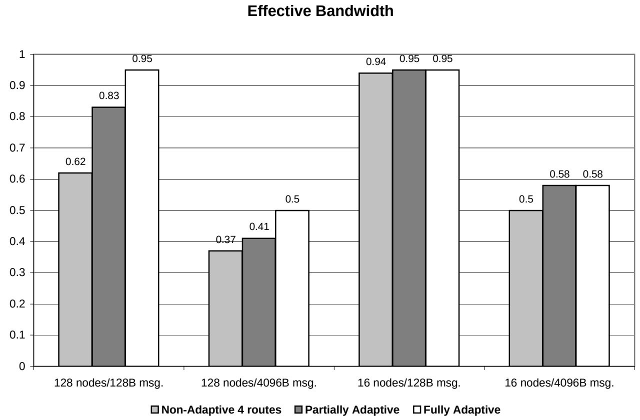

FIG. 20. Comparison between non-adaptive four routes, partially adaptive and fully adaptive routing methods.

If no grants are received, the packet is routed to the central buffer, just as for non-adaptive packets. If one or more grants are received from the output ports, the input port selects a winning output port among granted ports (based on a selection function) and immediately begins forwarding data to that output port through the central buffer. The losing output ports are notified that they are free to grant requests from other input ports. In the rest of this section, we discuss several selection functions.

6.1. Alternatives for the Output Selection Function

Random (RND) Selection Function: One output port is randomly selected from output ports which have granted input port's service request.

Least Recently Used (LRU) Selection Function: In the switch chip every input port keeps an LRU ordered list of output port numbers. Each list is kept ordered based on the time an output port is used by a message entering from this input port. After the input port makes a service request, among all the granted output ports, the LRU output port wins the selection. The motivation for the LRU selection function is to distribute message traffic evenly among output ports.

Most Recently Used (MRU) Selection Function: This selection function is similar to the LRU selection function except that the list is ordered in reverse. After the input port makes a service request, among all the granted output ports, the MRU port wins the selection. The motivation for MRU selection function is to see if multi–packet messages will perform better if they are concentrated into a single output port to avoid mixing with other traffic in the network.

Round Robin (RR) Selection Function: In the 8-port switch chip, each input port has a 3-bit register called RRID, which holds the ID (ID=0...7) of the most recently used output port used by a message entering from the input port. After the input port makes a service request, then one or more outputs will grant the request. The RR selection function is such that the RRID register is incremented (modulo 8) until a granted output port is found whose ID matches the RRID register. Thus, granted outputs are considered in round robin fashion starting with the last used output port number + 1 (mod 8). The RR function is not same as the LRU function. LRU keeps a history of all output ports and favors un-granted output ports by increasing their priority. RR doesn't keep any history except for the most recently used output port ID.

Centralized Least Recently Used (LRUC) Selection Function: LRUC is similar to the LRU selection function. The difference is that in the LRUC selection function there is only one LRU list *per switch chip*, while the LRU selection function uses one LRU list *per input port*. The motivation behind the LRUC function is to share the information among input ports thereby making switch-chip-wide decisions when balancing the output traffic. In contrast, in the LRU selection function, the input ports do not share their LRU information. For example, an output port may appear as the least recently used in one input's LRU list and it may appear as the most recently used in another input's LRU list.

Destination based Least Recently Used (LRUD) Selection Function: Similar to the LRU selection functions, LRUD uses multiple LRU lists. However, in every switch chip there is one LRU list per *node switch* in the network. A node switch refers to a switch chip which is connected to one or more (usually four) nodes. For example, in Fig. 2 there are 4 node switches connecting to the nodes P0–P15 on the left-hand-side of the network and in Fig. 4 there are 32 node switches connecting to 128 nodes on both sides of the network.

The motivation behind the LRUD selection function is to make a network-wide decision when selecting outputs in a switch chip. The LRUD function attempts to keep separate LRU history of messages going to the same destination in the network. This permits consecutive incoming messages going to different destinations to use the same output port. Since consecutive messages will use separate downward paths from LCAs, even if the first message is blocked downstream, the next message(s) may continue because they will be using different output ports in the LCA switches.

6.2. Performance Evaluation

In this section, we focus on the effect of output selection function on the performance of adaptive routing. We also present the results for the four-route oblivious routing along adaptive routing algorithms. In addition to the random traffic, we used three other permutation traffic patterns: bit-complement, matrix-transpose, and bit-reversal permutations. For bit-complement, source nodes, represented in binary by $s_{n-1}s_{n-2}\ldots s_1s_0$, send messages to $\overline{s_{n-1}s_{n-2}}\ldots\overline{s_1s_0}$. We used two topologies, for 16-node and 128-node systems illustrated in Figs. 2 and 4 and six different message sizes, 128, 256, 512, 1K, 2K, and 4K bytes.

Figure 21(a) shows the effective throughput of the network (as a percentage of maximum 1.0) using the four-route oblivious routing and the adaptive routing algorithms with different selection functions and for 128-byte messages on a 16 node system under random traffic. All algorithms perform more or less the same for small message traffic. Figure 21(b) is for the same conditions, except message size is 4K bytes, which shows that all adaptive routing algorithms perform the same regardless of the choice of output selection function. For long messages LRUD outperforms the others negligibly. However, all adaptive algorithms perform better than the oblivious algorithm.

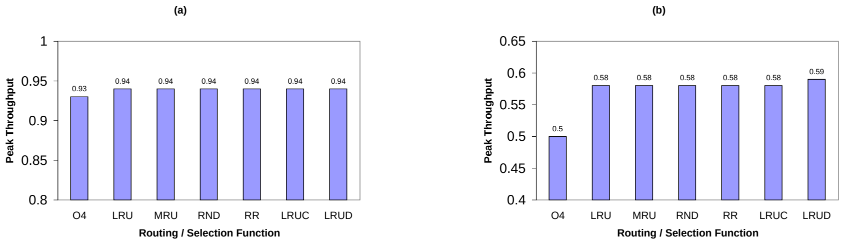

FIG. 21. Peak throughput for the oblivious routing and adaptive routing with different selection functions on a 16-node system with random traffic pattern for (a) 128-byte messages and (b) 4096-byte messages.

The results for the bit-reverse permutation displayed in Fig. 22(b) show that the MRU selection function performs poorly for long messages. The other selection function which performs slightly worse is the RND selection function.

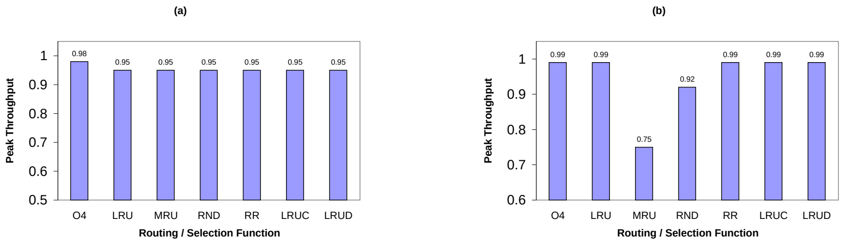

FIG. 22. Peak throughput for the oblivious routing and adaptive routing with different selection functions on a 16-node system with bit-reversal traffic pattern for (a) 128-byte messages and (b) 4096-byte messages.

Figures 23 and 24 illustrate the performance of the oblivious routing and the adaptive routing with different selection functions in a 128-node system. It can be observed that the adaptive routing algorithms outperform the oblivious routing. The performance of all of the selection functions are the same for 128-byte messages under the random traffic (Fig. 23(a)). The MRU and LRUC perform slightly worse than the other adaptive selection functions for long messages (Fig. 23(b)). The simulation results for the bit-reversal traffic is shown in Fig. 24. Using adaptive routing instead of four-route oblivious routing results in more than 229% improvement in the peak throughput. For short messages LRU and LRUD outperform the rest of the selection functions. The RND, RR, and LRUC selection functions perform marginally worse than LRU and LRUD. The MRU selection function performs poorly. For longer messages (Fig. 24(b)), MRU and LRUC performs worse than the rest of the adaptive selection functions, achieving a throughput of 0.55 and 0.54, respectively. Using LRU instead of LRUC increases the peak throughput by more than 30%.

To summarize the simulation results, for the random traffic, all of the selection functions perform well. LRU, LRUD, and RR outperform MRU, RND, and LRUC only marginally. The performance of the studied selection functions shows a wider range for the bit-reversal traffic. For this traffic pattern, LRU and LRUD outperform the other selection functions (LRUD performing slightly better than LRU).

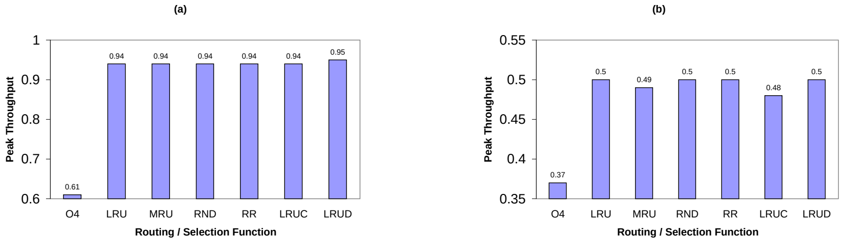

FIG. 23. Peak throughput for the oblivious routing and adaptive routing with different selection functions on a 128-node system with random traffic pattern for (a) 128-byte messages and (b) 4096-byte messages.

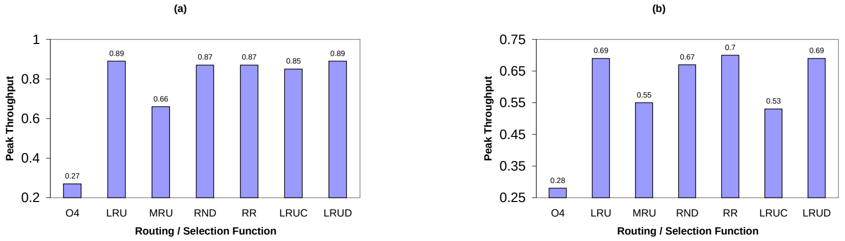

FIG. 24. Peak throughput for the oblivious routing and adaptive routing with different selection functions on a 128-node system with bit-reversal traffic pattern for (a) 128-byte messages and (b) 4096-byte messages.

The RR selection function shows a good performance under the bit-reverse traffic as well. The MRU selection function performs poorly across the board, while LRUC and RND functions perform poorly for long messages. In other words, LRU, LRUD, and RR selection functions perform well under different traffic patterns and system configurations. Because of the ease of implementation of LRU, the Switch2 chip implements the LRU selection function.

Another important factor in choosing a selection function is the cost of the hardware implementation. LRUD function requires very large number of LRU lists hence it is not cost effective considering that it performs equally or marginally better than LRU. Results show that the RR function may be considered as an improvement over LRU function in future generation switch chips because RR requires less chip area but performs as good as LRU. The LRU function implementation on SP switch chips requires $n(n+1)/2$ latches, $n(n-1)$ 2-input NAND gates, and n n-input NAND gates where n is the number of switch ports. A faithful implementation of RR requires a very similar amount of combinational logic, but only requires $\log_2 n$ latches. As latches are typically far more area-intensive than 2-input NAND gates, the area required for latches is a worthwhile consideration.

7. CONCLUSIONS

We have described the Switch2 switch chip, which may be used to interconnect future IBM RS/6000 SP systems. The new features of Switch2 and its differences with the earlier generation switch chips are presented. Switch2 introduces a major routing enhancement: adaptive source routing. We have described how adaptive source routing is supported in Switch2. We have also presented two novel route generation algorithms for adaptive source routing. We have shown that using adaptive source routing results in a significant improvement in the performance of the system under different traffic patterns and for different system sizes. We have also discussed and evaluated different output selection functions and shown that the best output selection functions for BMINs are not dependent on the traffic pattern, message size, or system size. The presented Switch2 switch chip represents a significant advance over its predecessors in performance and usability.

REFERENCES

1. B. Abali and C. Aykanat. Routing Algorithms for IBM SP1. In *Proceedings of the 1st International Workshop PCRCW, In Lecture Notes in Computer Science, vol. 853, Springer-Verlog*, pages 161–165, 1994.
2. B. Abali. A Deadlock Avoidance Method for Computer Networks. In *Proceedings of the 1st International Workshop on Communication and Architectural Support for Network-Based Parallel Computing (CANPC), In Lecture Notes in Computer Science, vol. 1199, Springer-Verlog*, pages 61–72, 1997.
3. Y. Aydogan, C. B. Stunkel, C. Aykanat, and B. Abali. Adaptive source routing in multistage interconnection networks. In *Proceedings of the 10th Int. Parallel Processing Symp. (IPPS'96)*, pages 258–267, April 1996.
4. S. Badr and P. Podar. An Optimal Shortest-Path Routing Policy for Network Computers with Regular Mesh Connected Topologies. *IEEE Transactions on Computers*, 38(10):1362–1371, 1989.
5. M. Banikazemi, C. Stunkel, D. K. Panda, and B. Abali. Adaptive Routing in RS/6000 SP-like Bidirectional Multistage Interconnection Networks. In *Proceedings of the Int. Parallel and Distributed Processing Symp. (IPDPS)*, May 2000.
6. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, NY, 1991.
7. D. E. Culler and J. P. Singh. *Parallel Computer Architecture: A Hardware-Software Approach.* Morgan Kaufmann, March 1998.
8. W. J. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE Trans. on Computers*, 39(6):775–785, June 1990.
9. W. J. Dally. Virtual-channel flow control. *IEEE Trans. on Parallel and Distributed Systems*, 3(2):194–205, March 1992.
10. W. J. Dally and C. L. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transactions on Computers*, pages 547–553, May 1987.
11. J. Duato. Improving the Efficiency of Virtual Channels with Time Dependent Selection Functions. *Parallel Arch. and Lang. Europe 92*, pages 635–650, 1992.
12. J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Networks: An Engineering Approach.* The IEEE Computer Society Press, 1997.
13. W. Feng and K. G. Shin. Impact of Selection Functions on Routing Algorithm Performance in Multicomputer Networks. In *Proceedings of the 11th Int. Conf. Supercomputing*, 1997.
14. C.J. Glass and L. M. Ni. The turn model for adaptive routing. *J. Assoc. Comp. Machinery*, 41(5):874–902, 1994.
15. E. Horowitz and S. Sahni. *Fundamentals of Computer Algorithms.* Computer Science Press, Maryland, 1989.
16. Manolis Katevenis, Panagiota Vatsolaki, and Aristides Efthymiou. Pipelined Memory Shared Buffer for VLSI Switches. In *Proceedings of ACM SIGCOMM*, pages 39–48, August 1995.
17. P. Kermani and L. Kleinrock. Virtual Cut-Through: A New Computer Communications Switching Technique. *Computer Networks*, 3(4):267–286, Sept. 1979.
18. C. E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Trans. on Computing*, C-34(10):892–901, Oct. 1985.
19. T. M. Pinkston and S. Warnakulasuriya. On Deadlocks in Interconnection Networks. *Proceedings of the 24th International Symposium on Computer Architecture*, pages 38–49, June 1997.
20. W. Qiao and L. M. Ni. Adaptive routing in irregular networks using cut–through switches. In *Proceedings of the 25th Int. Conf. Parallel Processing (ICPP)*, August 1996.

21. I. D. Scherson and C.-H. Chien. Least common ancestor networks. In *Proceedings of the 7th Int. Parallel Processing Symp.*, pages 507–513, 1993.

22. L. Schwiebert and R. Bell. The Impact of Output Selection Function Choice on the Performance of Adaptive Wormhole Routing. In *Proceedings of the 10th Int. Conf. Parallel and Distributed Computing Sys.*, pages 539–544, 1997.

23. H. Sethu, R. F. Stucke, and C. B. Stunkel. Technique for accomplishing deadlock free routing through a multi–stage cross–point packet switch. U.S. Patent 5,453,978, issued 9/26/1995.

24. M. Snir, P. Hochschild, D. D. Frye, and K. J. Gildea. The communication software and parallel environment of the IBM SP2. *IBM Systems Journal*, 34(2):205–221, 1995.

25. C. B. Stunkel et al. The SP2 High-Performance Switch. *IBM System Journal*, 34(2):185–204, 1995.

26. C. B. Stunkel, J. Herring, B. Abali, and R. Sivaram. A New Switch Chip for IBM RS/6000 SP Systems. In *Supercomputing 99 (SC99)*, 1999.

27. C. B. Stunkel, D. Shea, D. G. Grice, P. H. Hochschild, and M. Tsao. The SP1 High Performance Switch. In *Scalable High Performance Computing Conference*, pages 150–157, 1994.

28. C. B. Stunkel, D. G. Shea, B. Abali, M. M. Denneau, P. H. Hochschild, D. J. Joseph, B. J. Nathanson, M. Tsao, and P. R. Varker. Architecture and Implementation of Vulcan. In *Proceedings of the International Parallel Processing Symposium*, pages 268–274, 1994.

29. J. Updahyay, V. Varavithya, , and P. Mohapatra. A Traffic Balanced Adaptive Wormhole-Routing Scheme for Two Dimensional Meshes. *IEEE Transactions on Computers*, 46(2):190–197, 1997.

30. J. Wu. An Optimal Routing Policy for Mesh-Connected Topologies. In *Proceedings of the Int. Conf. Parallel Processing*, pages 267–270, 1996.