Tailoring Routing and Switching Schemes to Application Workloads in Multicomputer Networks

Wu-chang Feng, Jennifer Rexford, Stuart Daniel,
Ashish Mehra, and Kang Shin

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122

Abstract

Achieving good overall performance in multicomputers requires matching application communication characteristics with a suitable network design. In order to study the complex interactions between router policies and communication workloads, we are building SPIDER (Scalable Point-to-point Interface DrivER), an experimental router that implements various routing-switching combinations through microprogrammable routing engines. By simulating a network of SPIDERs at the cycle level, we evaluate the performance of several routing and switching schemes under a variety of traffic loads. These results show that tuning network policies to application communication characteristics can significantly improve multicomputer performance.

Keywords: Multicomputers, interconnection networks, routers, switching, routing

1 Introduction

Message-passing multicomputers have emerged as a cost-effective platform for exploiting concurrency in a variety of applications. In these systems, fast message exchange enables efficient, fine-grained cooperation between processing elements. Achieving good overall performance requires matching application communication characteristics with a suitable network design. However, parallel applications impose a wide range of communication patterns on the underlying interconnection network. Scientific computations [1, 2], parallel databases, and real-time applications [3, 4] generate distinct distributions for message lengths, interarrival times, and target destination nodes.

Message lengths can vary depending on packetization policies as well as the type of communication (e.g., data requests, data transfers, and acknowledgement messages), while message interarrival times depend on task granularity and scheduling within the network. Finally, message destination distributions depend on the mapping of communicating tasks across the nodes in the network. This paper shows that tuning network policies to these diverse application

The work reported in this paper was supported in part by the National Science Foundation under Grant MIP-9203895. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the view of the NSF.

characteristics can significantly improve multicomputer performance. These network policies are implemented in the router hardware that connects an individual processing node to the interconnection fabric and manages traffic flowing through the node en route to other destinations.

It is difficult to design a router which performs well under all communication workloads. In particular, these diverse communication patterns significantly impact the suitability of network switching and routing schemes. The switching scheme affects communication performance by determining what link and buffer resources a packet consumes at a given node in its route. Traditional *packet switching* requires incoming packets to buffer completely before transmission to a subsequent node can begin. In contrast, cut-through switching schemes, such as *virtual cut-through* [5] and *wormhole* [6], try to forward an incoming packet directly to an idle output link. If the outgoing link is busy, virtual cut-through switching buffers the packet, while a blocked wormhole packet stalls pending access to the link.

The routing algorithm determines which links a packet traverses to reach its destination. *Oblivious* routing generates a single, deterministic outgoing link for an incoming packet, whereas *adaptive* schemes base the routing decision on prevailing network conditions. By considering multiple outgoing links, adaptive algorithms can increase the likelihood of cut-through at intermediate nodes and balance the traffic load in the network. These algorithms, however, may also increase protocol processing overhead due to the possibility of out-of-order arrivals [7]. While most oblivious routing algorithms generate only minimum-hop routes between the source and destination nodes, some adaptive schemes consider *nonminimal* routes in the hope of circumventing network congestion or faulty links.

Given the unique characteristics of each routing and switching scheme, choosing different schemes for each workload allows the network to accommodate diverse application characteristics. Most existing routers, however, only support one routing-switching combination, limiting the network's ability to adapt to applications. This paper shows that flexible hardware support for routing and switching enables multicomputer networks to effectively handle a variety of applications. In order to study the complex interactions between router polices and communication workloads, we are building SPIDER (Scalable Point-to-point Interface DrivER), an experimental router that implements various routing-switching combinations through microprogrammable routing engines [8]. Microprograms compute the routing-switching decision for each arriving packet, depending on the packet's header and prevailing network conditions.

The next section of the paper highlights application communication patterns and motivates how flexible routing and switching can improve network performance. Section 3 describes SPIDER and discusses how it can tailor routing and switching schemes for a diverse set of parallel applications. By simulating a network of SPIDERs at the cycle level, Section 4 evaluates the performance of SPIDER and the different routing-switching schemes under various application workloads. Varying these low-level parameters required a flexible simulation environment for evaluating multicomputer router designs [9]. Section 5 concludes the paper with a discussion of future work on tailoring network policies to application characteristics.

2 Motivation

Parallel applications generate a wide range of communication workloads depending on the application's granularity and mapping across multiple nodes. Multi-user systems exacerbate these effects since different applications may run *simultaneously*; these applications may execute on different parts of the network or even time-share the same processing elements. Consequently, communication characteristics such as message interarrival times, lengths, and target destinations vary substantially on modern parallel machines, as discussed below.

Message/packet arrival: Earlier studies of multicomputer networks have typically modeled message arrivals as a Poisson process, with exponentially-distributed interarrival times. This assumption was made, in part, due to the analytical tractability of such models and the lack of more realistic data. However, detailed measurements of multicomputer applications have led to more sophisticated message generation models. In particular, these studies show that applications typically generate bursty network traffic [1, 2], due to multi-packet messages and fine-grain handshaking between cooperating nodes. Similarly, multicast communication, for barrier synchronization or global reduction operations, can spawn several copies of a single message. These traffic models have significant impact on network evaluation, since Poissonian arrival processes typically yield overly optimistic performance results.

Message/packet length: Message and packet lengths depend on several factors including packet-size restrictions and the mixture of data and control messages. Although fixed-length packets or exponentially-distributed lengths simplify analytic models, recent work shows that real multicomputer applications typically generate *bimodal* packet-length distributions [1, 2]. This occurs because inter-node communication often consists of large data transfers, coupled with small request and acknowledgements packets. A router design can accommodate different packet lengths by separating short and long packets onto different virtual channels [10, 11]. As a further optimization, the Segment Router Architecture [12] employs different switching schemes based on packet length; long packets use wormhole switching to limit buffer-space requirements, while short packets use virtual cut-through switching to reduce channel contention.

Message destination: Message destination distributions vary a great deal depending on the network topology and the application's mapping onto different processing elements. While many analytical and simulation studies evaluate a uniform random distribution of destination nodes, this pattern does not capture the communication locality or traffic non-uniformities that arise in many applications. Hop-uniform traffic distributions can represent spheres of spatial locality, but these still do not capture the communication structure of specific parallel algorithms or applications. In particular, many scientific programs generate permutation patterns such as matrix-transpose (dimension-reversal), bit-complement, and bit-reversal [13–16]. Other application constructs, such as synchronization or multicast operations, may induce "hot-spots" of heavily-utilized nodes and links [16–18]. Finally, dynamic models [1] can produce variation in target destinations during the course of application execution.

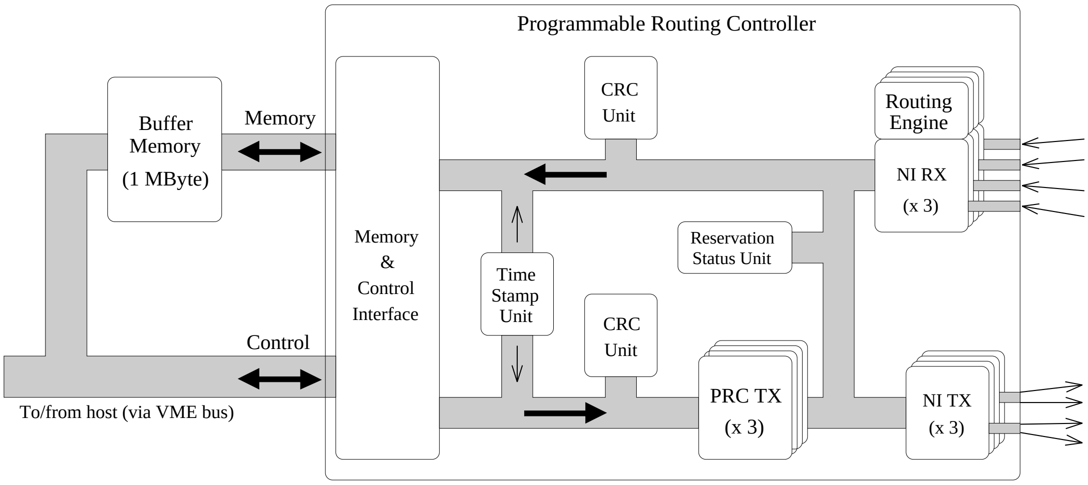


Figure 1: SPIDER

3 SPIDER Architecture

With such diverse application characteristics, no single set of router policies performs best under all conditions. SPIDER supports a broad spectrum of multicomputer applications through flexible support for routing and switching. While this flexibility could be achieved through a pure software router implementation, servicing multiple incoming and outgoing links entirely in software would overwhelm a conventional processor. Instead, SPIDER implements software control as close to the network as possible by dedicating a small custom routing engine to each incoming link.

3.1 SPIDER Components

As shown in Figure 1, SPIDER manages bidirectional communication with up to four neighboring nodes, with three virtual channels [19] on each unidirectional link. The programmable routing controller (PRC), a 236-pin custom integrated circuit measuring $1.4\,cm \times 1.4\,cm$, is the cornerstone of SPIDER [8, 20]. The 12 PRC TXs control packet transmission, while the 4 microprogrammable routing engines coordinate packet reception. Each routing engine performs low-level routing and switching operations for a single incoming link, with the three virtual channels sharing the custom processor. The network interface (NI) performs the media access and flow control on four pairs of AMD TAXI chips [21]; these TAXI transmitters and receivers control the physical links, providing a low-cost fiber-optic communication fabric. The NI TXs and NI RXs perform the necessary interleaving of virtual channels to and from the physical links, on a word-by-word basis[1].

SPIDER treats outbound virtual channels (NI TXs) as individually reservable resources, allowing the device to support a variety of routing and switching schemes through flexible control

[1]To reduce the package size of the PRC, a pair of outgoing links shares a single set of pins; internally, the PRC operates at 30 MHz, twice the link speed, to serve each outgoing link at its full rate.

over channel allocation policies. The reservation status unit handles requests from arriving packets to reserve or relinquish NI TXs, providing low-level support for both connection-oriented and connectionless transfer on each virtual channel. An arriving packet can invoke a variety of policies for selecting and reserving outbound channels. Upon receiving the header bytes from the incoming channel, the routing engine decides whether to buffer, stall, forward, or drop the packet, based on its microcode[2] and the packet's routing header. A routing engine can respond to network congestion by basing its routing decision on the reservation status of the outgoing virtual channels. By reserving multiple NI TXs, the PRC can forward an incoming packet to several output links simultaneously, allowing SPIDER to support efficient broadcast and multicast algorithms.

The host controls channel reservations for any packet stored in the buffer memory by assigning the packet to a particular PRC TX. The host transmits a packet by feeding this PRC TX with page tags that each include the address of an outgoing page and the number of words on the page. Likewise, the host equips each NI RX with pointers to free pages in the memory, for storing arriving packets. The control interface also provides read access to an event queue that logs page-level activities on each channel. The host can influence a routing engine's operation at run time through notification FIFOs, addressable as part of SPIDER's control interface. These FIFOs provide bidirectional communication between a routing engine and the host. For example, the host may wish to inform SPIDER about faulty links or memory usage at the local node, so the routing engines can adapt their routing-switching decisions accordingly.

3.2 Basic Operation

To illustrate the interaction between the host, SPIDER, and the network, consider how a message travels from the source node, cuts through an intermediate node, and arrives at the destination node.

Transmission: When an application requests the host to transmit a *message* to another node, the host disassembles the message into multiple *packets*, where a packet consists of one or more (possibly non-contiguous) *pages*. Using the control interface, the host feeds page tags to the appropriate PRC TX to initiate packet transmission. After reserving the NI TX, the PRC TX fetches the 32-bit data words from each page. During this memory transfer, the PRC transparently accumulates a 32-bit cyclic redundancy code (CRC) for error detection. After sending the last data word of the packet, the PRC TX transmits a 32-bit timestamp, read from a counter on the PRC, followed by the CRC; the timestamp values facilitate clock synchronization and computation of end-to-end packet latencies. The NI TX transmits each of these words to the TAXI transmitter a *byte* at a time; the TAXI device converts each byte into a string of *bits* for transmission on the serial link.

Cut-through: Packet reception begins when data arrives at a TAXI receiver. The receiving NI RX initially forwards data to its routing engine until it has accumulated enough header words to make a routing decision for the packet. If the packet is destined for a subsequent node, the routing engine can try to forward the packet directly to the next node by reserving

[2] Each routing engine has a 256-instruction control store. Microprograms for typical routing-switching schemes require about 60 to 70 instructions to implement.

an NI TX. If the routing engine is able to establish a cut-through, the engine then sends the data it has accumulated to that transmitter and configures the NI RX to forward subsequent data words directly to the reserved NI TX, bypassing the routing engine entirely. When the packet has cleared the node, the NI RX automatically reconfigures itself to forward the next packet header to the routing engine.

Reception/Buffering: When SPIDER stores the packet at the local node, however, the routing engine configures the NI RX to directly buffer the packet, reaccumulating the CRC as the data words travel to the memory interface. SPIDER writes these words into pages in the buffer memory and logs the arrival (and size) of each page in the PRC event queue. At the end of the final page of the packet, SPIDER appends the packet with a receive timestamp and logs a packet-arrival event indicating the outcome of the CRC check. If the packet has reached its destination, the host reassembles the pages into a packet and the packets into a message. Otherwise, the host schedules the packet for transmission to the subsequent node in its route.

3.3 Flexibility

SPIDER enables a multicomputer network to tailor its routing-switching schemes to the characteristics and requirements of parallel applications. Each routing engine includes an 8-bit arithmetic logic unit (ALU) for manipulating routing headers. Packets can invoke different routing and switching algorithms through conditional branches off packet header fields; this provides more flexibility than a table-lookup scheme, since microprograms can parse a variety of header formats. The routing engine can also base its routing-switching decisions on the incoming virtual channel (NI RX) identifier, allowing SPIDER to implement different microprograms for each virtual channel. This is useful for implementing deadlock-free wormhole routing algorithms that differentiate between packets on different virtual channels [22].

In addition, SPIDER may partition traffic across different NI RXs with distinct network policies. For example, time-constrained messages can use packet switching and static routing for predictable performance, while best-effort packets improve their average latency through cut-through switching and adaptive routing. Carrying these two types of traffic on different virtual channels allows real-time communication to coexist with best-effort packets without sacrificing the performance of either class [4, 23]. Similarly, the router may separate short control messages and long data packets onto different virtual channels, perhaps with different switching policies [10–12].

SPIDER can also assign routing algorithms to improve end-to-end performance. Although adaptive routing can reduce network latency, out-of-order packet arrivals can increase software processing delays at the receiving node. Opportunities for adaptive routing depend on the network topology and the distance a packet must travel to reach its destination. SPIDER can balance the trade-off between network latency and depacketization overheads by implementing adaptive routing only for single-packet messages or packets that must visit a large number of intermediate nodes. For these messages, additional routing adaptivity may significantly reduce network latency, outweighing the cost of packet reordering. For messages traveling short distances, SPIDER can impose static routing to eliminate out-of-order packet arrivals.

SPIDER's flexibility allows multiple parallel applications to execute simultaneously, without forcing all communication to adopt the same routing-switching policies. At run time, the system may assign a new application to a set of processors and download the appropriate microcode to the routing engines in each node. For example, one application may employ static wormhole routing on a 4×4 submesh, while another application allocates a separate 6×8 submesh using virtual cut-through switching with adaptive routing. Each application may determine how to allocate the virtual channels on each of its links, using the virtual channels to partition different traffic classes or to add flexibility to deadlock-free routing algorithms. These options enhance user control over the underlying interconnection network to improve application performance.

4 Performance Evaluation

SPIDER's flexibility facilitates experimentation with different routing-switching schemes and communication workloads. This enables comparisons between candidate router policies on a common platform, in order to tune network parameters to application characteristics and performance requirements.

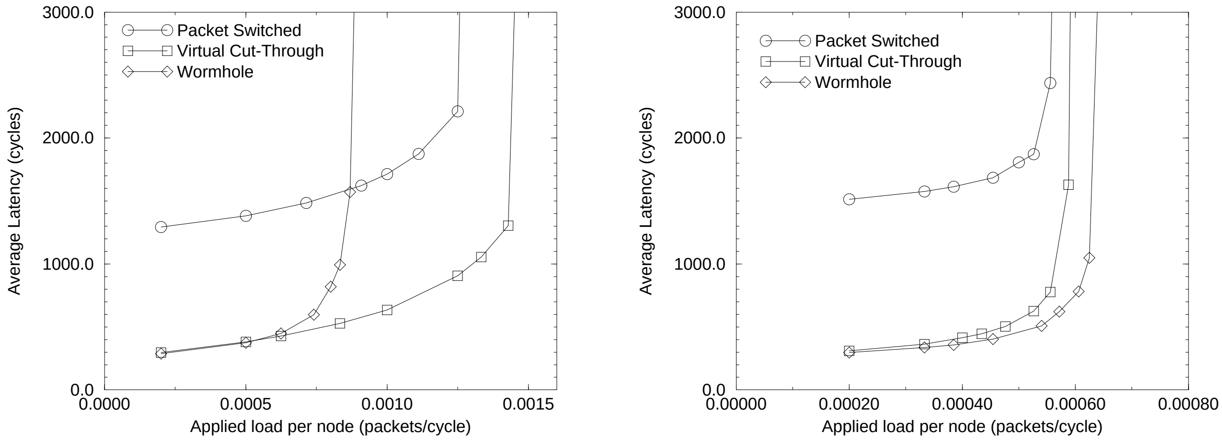
4.1 PP-MESS-SIM

Evaluating multicomputer network policies requires a flexible simulation environment. Implemented in C++, `pp-mess-sim` (point-to-point message simulator) is an object-oriented discrete-event simulation tool for evaluating multicomputer router architectures [9, 20]. Besides providing a general framework for evaluating router architectures, `pp-mess-sim` includes a cycle-level model of SPIDER that captures the details of flow control, resource arbitration, and microcode execution. Using a high-level specification language, the user can select the network topology, internal router policies, and the traffic patterns generated by each node.

Communication patterns stem from a collection of independent "tasks," each with its own performance metrics and packet characteristics. These tasks are then mapped onto individual nodes in the network to represent the communication behavior of concurrent applications. The simulator derives packet lengths and interarrival times from a variety of stochastic processes; similarly, tasks select packet destinations from various distributions to represent different application constructs. The simulation environment includes all of the packet length, interarrival, and target models discussed in Section 2. To evaluate a collection of routing-switching pairs, `pp-mess-sim` associates each task with a particular routing algorithm and switching scheme on a set of virtual channels. The simulator currently supports wormhole, virtual cut-through, and packet switching, as well as hybrid schemes, each under a variety of routing algorithms.

4.2 Simulation Experiments

Although SPIDER supports a variety of network topologies and router policies, the simulation studies focus on 64-byte packets traveling in an 8×8 square mesh using minimal-path unicast routing algorithms. Because other studies have shown how tailoring network policies to different packet sizes can significantly improve application performance [10–12, 24], our experiments do not specifically address this issue. Instead, the simulations examine network per-



(a) Uniform random traffic

(b) Matrix transpose traffic

Figure 2: Comparison of switching schemes under dimension-ordered routing

formance as routing, switching, network load, packet interarrival distributions, and destination node selection vary.

The destination node patterns include uniform random traffic, as well as the matrix-transpose, bit-complement, and bit-reversal permutations. In an 8×8 mesh, source node (c, d) communicates with destination node (d, c) for the matrix-transpose pattern and node $(7 - c, 7 - d)$ for the bit-complement permutation. For the bit-reversal pattern, source node (c, d) has identifier $c + 8d$ and transmits to node j, where the binary representation of j is the reverse of $c + 8d$. Although these permutations alone do not capture the communication characteristics of all parallel applications, they allow the simulation experiments to evaluate routing and switching schemes under a variety of different workloads.

Varying switching under static routing

In defining how packets flow through the network, the various switching schemes stress different resources at intermediate nodes. Figure 2 shows average packet latency for wormhole, virtual cut-through, and packet switching under two different traffic patterns. Packets employ static dimension-ordered routing using one virtual channel on each physical link. As expected, virtual cut-through switching consistently outperforms packet switching, since virtual cut-through traffic often avoids buffering delay at intermediate nodes. At low loads, wormhole switching performs extremely well for both communication patterns.

However, the relative performance of virtual cut-through and wormhole switching varies significantly between Figure 2(a) and 2(b). Under the traditional uniform random traffic pattern, the two switching schemes exhibit comparable performance at low loads, as shown in Figure 2(a). However, network contention limits wormhole throughput at higher loads; similar trends occurred for bit-complement traffic. By removing blocked packets from the network, virtual cut-through and packet switching consume network bandwidth proportional to the offered load. In contrast, a blocked wormhole packet stalls in the network until its outgoing channel becomes available; this stalled packet may then block other traffic destined for different output links. At higher loads, this effect enables packet switching to outperform wormhole switching, even though packet switching introduces buffering delay at each hop in a packet's route.

Despite channel contention, wormhole switching excels for the matrix-transpose permutation, as shown in Figure 2(b); similar trends occurred for bit-reversal traffic. This effect occurs because matrix-transpose traffic, coupled with dimension-ordered routing, limits harmful contention between packets heading to different parts of the network. In a square mesh, the matrix-transpose permutation requires node (c, d) to communicate with node (d, c). With dimension-ordered routing, each packet starting on row d proceeds in the x-direction to node (d, d), before traveling in the y-direction to reach the destination node. As a result, source nodes in row d inject packets that use the same row and column links. Although a blocked wormhole packet may still restrict other traffic from entering a node, this traffic must ultimately traverse the same links as the stalled packet.

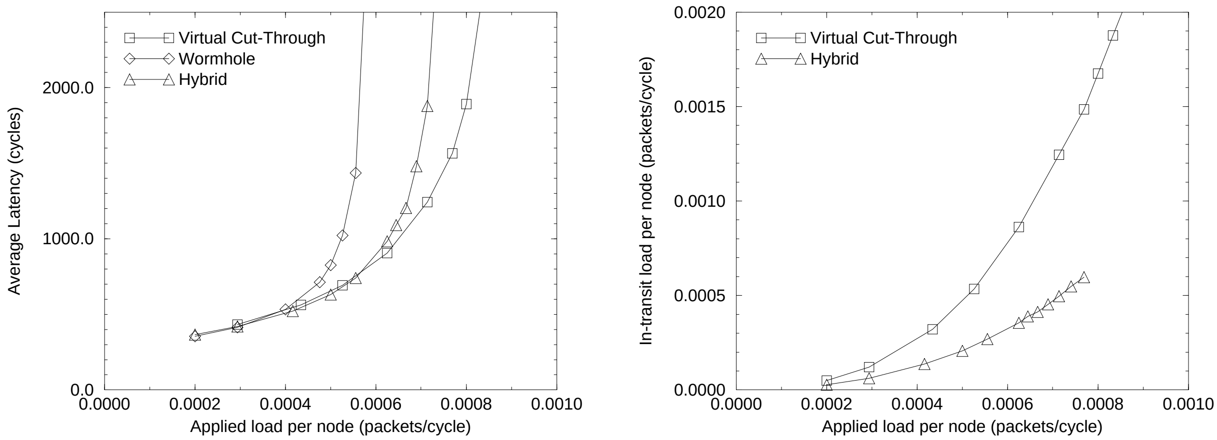
Buffering the blocked packet cannot alleviate this contention; in fact, virtual cut-through switching achieves a slightly lower peak throughput than wormhole switching, due to bandwidth limitations at the PRC memory interface. Neither wormhole nor virtual cut-through switching performs best in all situations. Supporting both schemes enables SPIDER to tune its switching policies to application characteristics and performance requirements.

Hybrid switching under static routing

Hybrid switching schemes attempt to balance the performance trade-offs between virtual cut-through and wormhole switching by combining the salient features of both approaches. Wormhole switching can achieve low end-to-end latency, but stalled packets can block access to network channels and reduce network throughput. Virtual cut-through switching alleviates this contention by buffering blocked packets, at the cost of consuming memory resources at intermediate nodes. A hybrid switching scheme dynamically combines wormhole and virtual cut-through switching, balancing the use of network and memory resources for "storing" blocked packets [25].

Figure 3 shows the performance of a hybrid algorithm that prevents blocked packets from stalling more than two virtual channels. The SPIDER routing engines implement this algorithm by inspecting an additional packet header field, used to record the number of channel reservations the packet holds. The algorithm changes from wormhole to virtual cut-through switching if the packet has traveled two or more hops without buffering. Whenever a routing engine establishes a cut-through for a packet, it increments the header field before forwarding the routing header to the next node in the route; the field is reset to zero whenever the packet buffers at an intermediate node.

As shown in Figure 3(a), selectively buffering blocked packets improves the achievable throughput over wormhole switching for bit-complement traffic; these trends held for uniform random traffic as well. Virtual cut-through switching achieves lower latency than both hybrid



(a) Average latency (b) In-transit traffic load

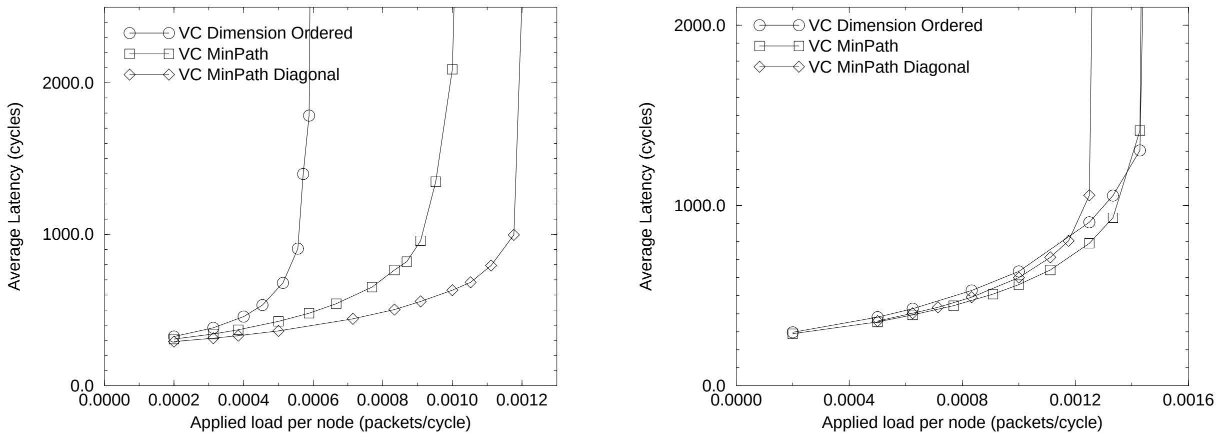
Figure 3: Performance of hybrid switching for bit-complement traffic

and wormhole switching, at the expense of additional memory usage, as shown in Figure 3(b). This graph plots the average packet memory demands for in-transit traffic under virtual cut-through and hybrid switching; wormhole switching does not consume any packet buffers at intermediate nodes. At high loads, virtual cut-through switching uses more than three times more memory resources than the hybrid scheme, since the hybrid algorithm allows packets to buffer at most once every three hops.

To further reduce memory requirements, SPIDER can also implement hybrid algorithms that base switching decisions on buffer utilization. In particular, the routing engines could force blocked packets to stall whenever the buffer memory is full, similar to the buffered wormhole scheme implemented in IBM's Vulcan switch [26]. The routing engines' notification FIFOs, described in Section 3, allow the host to inform SPIDER about memory availability. For additional flexibility, the routing engines can base their switching decisions on both memory *and* channel utilization, reserving a portion of the buffer memory for blocked packets that hold multiple virtual channels.

Varying routing under bit-reversal and uniform random traffic

Application workloads impact routing algorithm performance by determining the distribution of traffic in the network. Tailoring the routing algorithm to specific communication patterns can reduce packet latency and increase the achievable network throughput. Figure 4 shows average packet latency for three different minimal-path routing algorithms using virtual cut-through switching. As shown in Figure 4(a), the bit-reversal permutation generates non-uniform traffic that limits the performance of static routing algorithms; some links remain completely idle, while other links experience heavy traffic load.



(a) Bit-reversal traffic (b) Uniform random traffic

Figure 4: Comparison of virtual cut-through routing algorithms

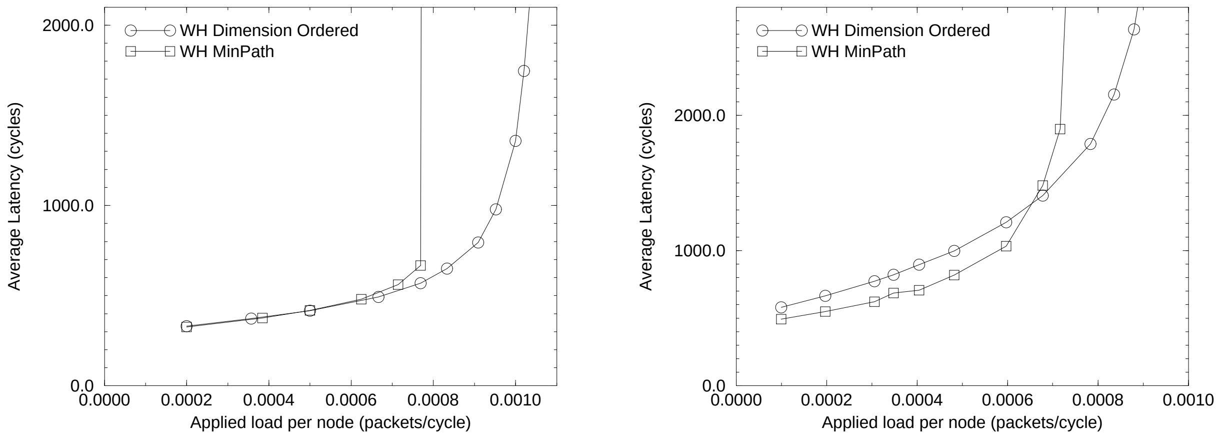
The minimal-path adaptive algorithm tries to circumvent network congestion by considering an alternate output link along a shortest path. This reduces packet latency and extends the achievable throughput by balancing traffic load across the network links. The router can further improve performance by favoring outgoing links that provide more future routing options [27]. The minimal diagonal algorithm in Figure 4(a) favors the x-direction whenever a packet has fewer remaining hops in the y-direction, and vice versa; this aggressive approach increases the number of two-choice nodes in a packet's route, allowing traffic to avoid more congested nodes and links.

Although diagonal routing reduces latency for bit-reversal traffic, Figure 4(b) suggests that increased adaptivity does not always improve communication performance. Uniform random traffic generates a balanced network load, so static routing algorithms typically perform fairly well. In fact, dimension-ordered routing outperforms diagonal routing in Figure 4(b). In dimension-ordered routing, a packet entering a node in one direction generally exits the node traveling in the same direction; this reduces the likelihood that packets from different incoming links contend for the same output port [28].

In contrast, adaptive algorithms often allow packets to alternate dimensions, possibly blocking other arriving traffic [16]. Adaptively changing dimensions may also increase congestion in the center of the mesh, as evidenced by the early saturation of the diagonal routing plot in Figure 4(b). Homogeneous networks, such as wrapped meshes or tori, do not exhibit this effect.

Varying routing and arrival process under bit-complement traffic

Adaptive routing algorithms can improve network performance by making profitable routing decisions based on local information. However, these local decisions can increase network



(a) Poisson arrival process

(b) Bursty arrival process

Figure 5: Comparison of wormhole routing algorithms under bit-complement traffic

congestion for some communication patterns, as shown in Figure 5. The graphs show average latency for wormhole switching under both dimension-ordered and adaptive routing; virtual cut-through experiments showed the same qualitative trends. The adaptive routing algorithm is a fully-adaptive minimal routing scheme that requires two virtual channels per link to prevent network deadlocks [29]; in these experiments, both routing algorithms employ a pair of virtual channels to enable fair performance comparisons. The dimension-ordered routing algorithm uses the extra virtual channel to reduce contention between packets traveling on the same link [16, 19].

In Figure 5(a), static routing consistently outperforms adaptive routing, especially at high loads. In an 8×8 square mesh, the bit-complement permutation requires source node (c, d) to communicate with node $(7-c, 7-d)$. As a result, all packets must eventually cross both the middle row and the middle column of the mesh to reach their destinations, irrespective of the routing algorithm. Dimension-ordered routing tends to avoid the center of the network, where the middle row and column meet, by exhausting the x-direction before routing a packet in the y-direction. In contrast, adaptive algorithms may try to avoid the heavily-congested middle column (or row) by routing packets to more lightly-loaded rows (or columns); this ultimately pushes traffic closer to the congested center of the network, increasing end-to-end latency. A local decision at one node causes a packet to travel a lightly-loaded link into a more congested region.

In addition, extra routing flexibility allows source nodes to inject more packets, further increasing contention at the middle of the network. Hence, in some situations, restricted routing flexibility can effectively limit the overuse of network resources [30]. However, this effect varies with the network load and the underlying traffic pattern, as shown in Figure 5(b). This experiment considers bursty traffic, in contrast to the traditional Poissonian packet arrival process in

Figure 5(a). The source nodes generate bursty traffic using a two-stage normal distribution of packet interarrivals [1]. Packet interarrivals stem from two independent normal distributions, with different means; sources randomly select 80% of interarrivals from the distribution with the small mean.

In Figure 5(b), the applied traffic load (x-axis) changes by varying the large mean, keeping the small mean fixed at 100 cycles. This generates relatively small packet interarrival times within a burst to capture the transmission of a multi-packet message or a handful of related messages. Figures 5(a) and (b) exhibit similar trends at high loads, but bursty traffic limits the effectiveness of static routing at low network loads since packets in a burst are queued up while previous packets are being sent. The adaptive algorithm helps dissipate bursts by capitalizing on multiple paths between each source and destination, thus reducing the queueing delay at the sending node. With such variations in performance across different communication workloads, SPIDER can adapt to its operating environment by selecting an appropriate routing-switching combination.

5 Conclusion

These experiments have shown that, for a range of communication workloads, it is useful to support multiple routing and switching policies in the interconnection network. Although these experiments have used synthetic target and interarrival distributions, real communication workloads should also exhibit similar performance effects. Since it is hard to predict how these workloads interact with router architectures, we have implemented SPIDER with a substantial amount of flexibility. Although the costs of such flexibility may sometimes outweigh the performance benefits [31], SPIDER can be used to evaluate a wide spectrum of network policies, from which a subset of the most useful schemes could then be efficiently implemented in subsequent router designs.

For example, an implementation of hybrid switching may prove useful if a router design has limited buffer space or cannot provide enough virtual channels to sufficiently alleviate packet contention. Similarly, a router design could incorporate a small library of routing algorithms and header formats, in lieu of microprogrammable routing engines, to support a handful of the most useful options in hardware. Further study of communication traces and real application workloads should guide the selection of the most beneficial network policies.

We are currently testing the SPIDER design and preparing the PRC chip for fabrication. The completed SPIDER boards will form the basis of a custom interconnection network for a collection of VME-based processing nodes. This platform will enable the benchmarking of routing and switching schemes for a diverse set of multicomputer applications. Using these results, we will also examine software mechanisms for exercising router flexibility to improve application performance.

References

[1] J.-M. Hsu and P. Banerjee, "Performance measurement and trace driven simulation of

parallel CAD and numeric applications on a hypercube multicomputer," *IEEE Trans. Parallel and Distributed Systems*, vol. 3, pp. 451–464, July 1992.

[2] R. Cypher, A. Ho, S. Konstantinidou, and P. Messina, "Architectural requirements of parallel scientific applications with explicit communication," in *Proc. Int'l Symposium on Computer Architecture*, pp. 2–13, May 1993.

[3] J.-P. Li and M. W. Mutka, "Priority based real-time communication for large scale wormhole networks," in *Proc. International Parallel Processing Symposium*, pp. 433–438, April 1994.

[4] J. Rexford and K. G. Shin, "Support for multiple classes of traffic in multicomputer routers," in *Proc. Parallel Computer Routing and Communication Workshop*, pp. 116–130, May 1994.

[5] P. Kermani and L. Kleinrock, "Virtual cut-through: A new computer communication switching technique," *Computer Networks*, vol. 3, pp. 267–286, September 1979.

[6] W. J. Dally and C. L. Seitz, "The torus routing chip," *Journal of Distributed Computing*, vol. 1, no. 3, pp. 187–196, 1986.

[7] V. Karamcheti and A. A. Chien, "Do faster routers imply faster communication?," in *Proc. Parallel Computer Routing and Communication Workshop*, pp. 1–15, June 1994.

[8] J. W. Dolter, S. W. Daniel, A. Mehra, J. L. Rexford, W. Feng, and K. G. Shin, "SPIDER: Flexible and efficient communication support for point-to-point distributed systems," in *Proc. Int'l Conf. on Distributed Computing Systems*, pp. 574–580, June 1994.

[9] J. Rexford, J. Dolter, W. Feng, and K. G. Shin, "PP-MESS-SIM: A simulator for evaluating multicomputer interconnection networks." To appear in *Proc. Annual Simulation Symposium*, April 1995.

[10] J. H. Kim and A. A. Chien, "Evaluation of wormhole routed networks under hybrid traffic loads," in *Proc. Hawaii Int'l Conf. on System Sciences*, pp. 276–285, January 1993.

[11] W. Feng, J. Rexford, A. Mehra, S. Daniel, J. Dolter, and K. Shin, "Architectural support for managing communication in point-to-point distributed systems," Tech. Rep. CSE-TR-197-94, University of Michigan, March 1994.

[12] S. Konstantinidou, "Segment router: A novel router design for parallel computers," in *Symposium on Parallel Algorithms and Architectures*, June 1994.

[13] S. Chittor and R. Enbody, "Performance evaluation of mesh-connected wormhole-routed networks for interprocessor communication in multicomputers," in *Supercomputing*, pp. 647–656, November 1990.

[14] J. H. Kim and A. A. Chien, "An evaluation of planar-adaptive routing (PAR)," in *Proc. International Symposium on Parallel and Distributed Processing*, 1992.

[15] W. Dally and H. Aoki, "Deadlock-free adaptive routing in multicomputer networks using virtual channels," *IEEE Trans. Parallel and Distributed Systems*, vol. 4, pp. 466–475, April 1993.

[16] S. Ramany and D. Eager, "The interaction between virtual channel flow control and adaptive routing in wormhole networks," in *Proc. International Conference on Supercomputing*, pp. 136–145, July 1994.

[17] S. Dandamudi and D. Eager, "Hot-spot contention in binary hypercube networks," *IEEE Trans. Computers*, vol. 41, pp. 239–244, February 1992.

[18] R. Boppana and S. Chalasani, "A comparison of adaptive wormhole routing algorithms," in *Proc. Int'l Symposium on Computer Architecture*, pp. 351–360, 1993.

[19] W. Dally, "Virtual-channel flow control," *IEEE Trans. Parallel and Distributed Systems*, vol. 3, pp. 194–205, March 1992.

[20] J. Dolter, *A Programmable Routing Controller Supporting Multi-mode Routing and Switching in Distributed Real-Time Systems.* PhD thesis, University of Michigan, September 1993.

[21] Advanced Micro Devices, 901 Thompson Place, P.O. Box 3453, Sunnyvale CA 94088-3453, *Am79168/Am79169 TAXI-275 Technical Manual*, ban-0.1m-1/93/0 17490a ed.

[22] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. Computers*, vol. C-36, no. 5, pp. 547–553, May 1987.

[23] J. Rexford, J. Dolter, and K. G. Shin, "Hardware support for controlled interaction of guaranteed and best-effort communication," in *Proc. Workshop on Parallel and Distributed Real-Time Systems*, (Cancun, Mexico), pp. 188–193, IEEE, April 1994.

[24] J. H. Kim and A. A. Chien, "The impact of packetization in wormhole-routed networks," in *Proc. Parallel Architectures and Languages, Europe*, 1993.

[25] K. G. Shin and S. Daniel, "Analysis and implementation of hybrid switching." In review, June 1994.

[26] 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 *Proc. International Parallel Processing Symposium*, pp. 268–274, April 1994.

[27] H. G. Badr and S. Podar, "An optimal shortest-path routing policy for network computers with regular mesh-connected topologies," *IEEE Trans. Computers*, vol. C-38, pp. 1362–1370, October 1989.

[28] A. Agarwal, "Limits on interconnection network performance," *IEEE Trans. Parallel and Distributed Systems*, vol. 2, pp. 398–412, October 1991.

[29] J. Duato, "A new theory of deadlock-free adaptive routing in wormhole networks," *IEEE Trans. Parallel and Distributed Systems*, pp. 1320–1331, December 1993.

[30] F. Hady and D. Smitley, "Adaptive vs. non-adaptive routing: An application driven case study," Tech. Rep. SRC-TR-93-099, Supercomputing Research Center, Bowie, Maryland, March 1993.

[31] A. A. Chien, "A cost and speed model for k-ary n-cube wormhole routers," in *Proc. Hot Interconnects*, August 1993.