Architectural Support for Managing Communication in Point-to-Point Distributed Systems

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

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

CSE-TR-197-94

Abstract

Point-to-point networks provide a natural platform for distributed systems due to their high bandwidth, scalability, and the potential for exploiting communication locality. Managing these networks, however, is complicated by the disparate quality-of-service requirements of distributed applications. Since the various routing and switching schemes are best-suited for different classes of traffic, the network adapter should be able to dynamically tailor them according to the current need. In this paper, we present and evaluate SPIDER (Scalable Point-to-point Interface DrivER), a network adapter that supports a variety of routing and switching schemes, and includes means for dynamically selecting between them. By exercising low-level control over each network link, SPIDER exploits concurrency amongst the links to reduce both packet latency and intrusion on the protocol software. We evaluate a mesh of SPIDERs through cycle-level simulations over a range of routing and switching schemes. We also demonstrate how this architecture can dynamically partition network bandwidth to meet the performance requirements of disparate traffic classes.

1 Introduction

Distributed systems have emerged as scalable, cost-effective platforms for applications with widely-varying characteristics and resource requirements. The advent of faster networks has enabled distributed systems to employ mechanisms previously applied only to tightly-coupled parallel machines, including system-wide shared memory [1] and a finer grain of computation [2]. Scalable application performance in such environments depends largely on the provision of concurrent low-latency and high-bandwidth communication. Network designs employing shared buses or token rings, for example, can effectively interconnect a small number

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.

of computing nodes, but the serialization of medium access limits performance in larger configurations. Alternatively, a network of interconnected switches [3–5] can enable concurrent communication between nodes, delivering scalable performance at increased cost.

Partially-connected point-to-point topologies such as meshes [6–8] offer an effective way to interconnect the nodes of a distributed system. Besides providing a higher degree of concurrent communication, they permit incremental scaling of the network, and hence the aggregate communication bandwidth, while maintaining the connectivity of existing nodes. In point-to-point networks each node must manage several input and output links, providing routing and switching services for every incoming packet. Exploiting the available network bandwidth requires effective, fine-grain management of this concurrency at the physical links. Managing this concurrency entirely in software would consume substantial processing and memory resources while incurring large packet delays.

With network bandwidth increasing faster than processor and memory bandwidth, handling low-level communication protocols in software is increasingly burdening the communication subsystem. By migrating frequent and time-consuming communication functions into hardware, both parallel and distributed systems can effectively utilize the available network bandwidth. Since many tightly-coupled parallel machines support a fixed set of communication policies, they implement internode communication in dedicated hardware routers [9–13]. In contrast, distributed systems typically support a wide range of communication protocols. Since the network adapter defines the communication primitives available to the protocol software, flexible support for network I/O necessitates a careful division of functionality between the adapter and the controlling processor (network host) [14]. The overhead of host-adapter interaction can limit the amount of useful concurrency between the software and hardware portions of the protocol stack. Single-port network adapters either facilitate flexibility through simple designs [15–18], or restrict flexibility, and hence improve performance, by supporting specific communication protocols and resource management strategies [19, 20].

This paper proposes architectural support for managing communication in point-to-point distributed systems. In particular, we present flexible, integrated hardware for routing and switching which supports, but does not dictate, higher-level host policies. Section 2 discusses the influence of routing and switching techniques on both host resource consumption and packet latency. Section 3 describes the architecture and basic operation of SPIDER, the *Scalable Point-to-point Interface DrivER*, that provides flexible hardware support for routing and switching in point-to-point distributed systems [21]. Section 4 addresses how SPIDER insulates the software from low-level protocol processing without precluding host control over network policies. By simulating a mesh of SPIDERs at the cycle level, Section 5 evaluates how different routing and switching schemes affect packet latency and consumption of host resources. Section 6 concludes the paper.

2 Motivation

Distributed systems employ a variety of communication paradigms, such as remote procedure call and connection-oriented data transfer, that affect the quantity and frequency of communication between nodes. In addition, emerging distributed applications require communication services with strict quality-of-service requirements, such as latency or bandwidth guarantees [22, 23]; this necessitates the coexistence of *guaranteed* and *best-effort* traffic. For point-to-point networks, these traffic patterns and classes affect the suitability of particular routing and switching schemes. Effective mixing of different traffic patterns mandates flexible selection of routing and switching schemes as well as mechanisms for controlling the interaction between the traffic classes.

The switching scheme dictates how packets flow through intermediate nodes. Traditional *packet switching* requires an arriving packet to buffer completely before transmission to a subsequent node can begin. This consumes processing and memory resources at each intermediate node. In contrast, cut-through switching schemes, such as *virtual cut-through* [24] and *wormhole* [9], attempt to directly forward the incoming packet to an idle output link. If the packet encounters a busy outgoing link, virtual cut-through switching stores the packet, consuming memory and processing resources at the intermediate node. On the other hand, a blocked wormhole packet stalls in the network, effectively dilating its length until the outgoing link becomes free. Although wormhole switching completely insulates the host from in-transit packets, it increases contention for network bandwidth.

While the switching scheme determines what resources are used at intermediate nodes, the routing algorithm determines *which* nodes a packet visits en route to its destination. *Static* routing generates a single, deterministic path between each source and destination node, whereas *adaptive* routing schemes can incorporate network conditions into the routing decision. By considering multiple outgoing links, adaptive algorithms can increase the likelihood of cut-through at intermediate nodes. Most static routing algorithms generate only minimum-hop routes between the source and destination nodes; in contrast, some adaptive schemes consider *nonminimal* routes in the hope of circumventing network congestion or faulty links. While adaptive algorithms can reduce the end-to-end latency for individual packets, the destination host may incur increased overhead due to out-of-order packet arrival. This may also complicate the flow control and error control for software protocols providing reliable and/or in-order message delivery.

Each routing and switching scheme is best suited for certain classes of traffic with particular characteristics and quality-of-service requirements. Although cut-through switching techniques can improve average packet latency, they limit the ability of intermediate nodes to influence the allocation of buffer and network resources. By buffering packets at each intermediate node, packet switching facilitates resource scheduling to provide delay or bandwidth guarantees [25]. Likewise, static routing ensures that a packet utilizes buffer and link resources at predetermined nodes and links along its route. Adaptive routing algorithms, on the other hand, select intermediate nodes and links dynamically, complicating reservation of buffer and link bandwidth. Guaranteed traffic can, therefore, employ packet switching and static routing for high predictability, while best-effort traffic exploits cut-through switching and adaptive routing for low-latency communication.

While it is possible to implement routing and switching entirely in software, the sheer complexity of managing multiple links can easily overwhelm a conventional processor. Implementing routing and switching entirely in hardware, as in many parallel machines, exploits network concurrency but limits host influence over network policies. Flexible hardware support, coupled with rich interface semantics, can provide efficient, fine-grain control over low-level routing and switching. The next section presents a hybrid approach that pushes software control as close to the links as possible, allowing the host to tailor the routing and switching schemes to current network conditions and control the interaction between different classes of traffic.

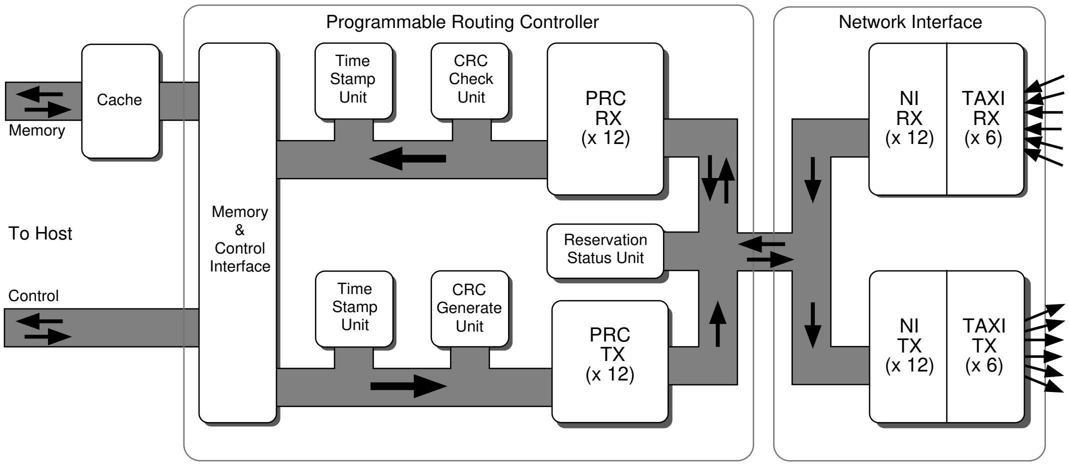


Figure 1: SPIDER

3 SPIDER

SPIDER, a network adapter for point-to-point distributed systems, exploits concurrency between the physical links, while enabling host control over communication policies. The device offloads low-level routing and switching from the software protocols running on the controlling host processor. User applications may run on this host or on another processor that uses the host, coupled with SPIDER, as a dedicated communication engine.

The adapter, shown in Figure 1, implements programmable routing and switching schemes for best-effort traffic, while facilitating host control over scheduling and resource allocation for guaranteed communication. Designed to reside on the host processor's private memory bus, SPIDER has direct access to the host memory and provides the host with a memory-mapped control interface. The adapter coordinates bidirectional communication with up to six neighboring nodes, with two virtual channels [26] on each unidirectional link. Using SPIDER, one can construct point-to-point distributed systems with a variety of topologies, including rings, meshes, and irregular configurations.

3.1 Architecture

Virtual channels are the unit of resource allocation and scheduling throughout SPIDER. The channel abstraction transcends the multiple protocol layers in the device, allowing host operations at its control interface to influence low-level routing and switching operations at the links. Many parallel machines utilize virtual channels to provide routing flexibility with wormhole switching [26–28]. By exporting this abstraction to the host, virtual channels can also facilitate separate management of traffic classes on distinct virtual networks. The programmable routing controller (PRC), a custom integrated circuit,[1] exploits concurrency amongst the virtual channels and provides fair, fine-grain arbitration at the memory and network interfaces [29]. The twelve PRC TXs provide low-level control of packet transmission while the twelve microprogrammable PRC RXs coordinate packet reception, as well as low-

[1] The PRC is a 187-pin ASIC designed in a 0.8μm process using Cascade's Epoch silicon compiler. The chip has a die size of 511 by 529 mils, and operates at a clock speed of 27.8 MHz on the network interface. The control interface operates off the host's bus clock, while the memory interface is asynchronous.

level routing and switching for in-transit packets.

The PRC TXs and PRC RXs implement the low-level drivers controlling the actual transmitter and receiver devices. The network interface (NI) performs the media access and flow control on six pairs of AMD TAXI chips [30], insulating the PRC from dependence on a particular communication medium. The NI TX and NI RX control units perform the necessary interleaving of virtual channels to and from the physical links, on a byte-by-byte basis. The TAXI chips control the physical links, providing a low-cost communication fabric of either fiber-optic or twisted-pair interconnect. Each TAXI TX–RX pair forms a bidirectional link to an adjacent node. A TAXI TX accepts injected data from two NI TXs and each TAXI RX delivers data to two NI RXs providing the pair of virtual channels for each physical, unidirectional link.

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 over channel allocation policies. The reservation status unit handles requests from the PRC RXs and PRC TXs to reserve or relinquish the 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 PRC RX decides whether to buffer, stall, forward, or drop the packet, based on its microcode[2] and the packet's routing header. A PRC RX can respond to network congestion by basing its routing decision on the reservation status of the outgoing virtual channels. By reserving multiple NI TXs, a PRC RX can also direct an incoming packet to several output links simultaneously, allowing SPIDER to support efficient broadcast algorithms [31].

The host can influence operation in the PRC RX through notification FIFOs, addressable as part of SPIDER's control interface. These FIFOs provide bidirectional information exchange between a PRC RX and the host. The host controls channel reservations for any packet stored in the host 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 provides each PRC RX with pointers to free pages in the memory, for use by arriving packets. The control interface also allows the host read access to an event queue that logs page-level activities on each channel.

3.2 Basic Operation

Figure 2 shows how a message from a user application moves through the host and SPIDER on various nodes, in terms of the OSI reference model. While the host is responsible for the presentation, session, and transport layers, SPIDER handles the data link and physical layers, as well as part of the network layer. To illustrate the interaction between the host, SPIDER, and the network, consider how a message travels through the OSI layers during transmission from the source node, cut-through at an intermediate node, and reception 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 then instructs the appropriate PRC TX to transmit these pages. After reserving the NI TX, the PRC TX

[2] Each PRC RX has a 128-instruction control store. Microprograms for typical routing and switching schemes require about 60 to 70 instructions to implement.

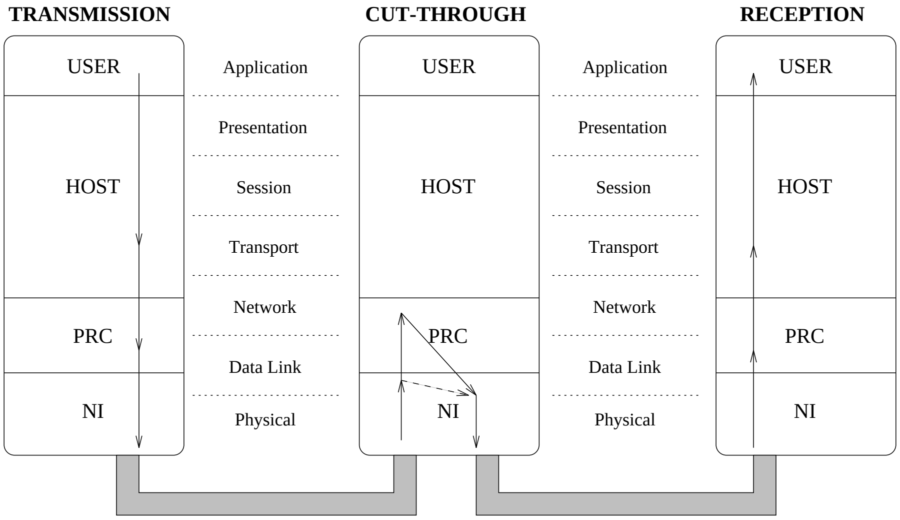


Figure 2: OSI reference model

fetches the data from each page. Data is initially retrieved from the host memory in *cache lines*, and converted to *words* as the packet travels from the memory interface to the PRC TX. During this transfer to the PRC TX, 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 PRC TX transmits each of these words to the NI a *byte* at a time. Under NI TX control, the TAXI TX on the NI 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 RX in the network interface. The NI RX initially forwards data to its associated PRC RX, until the PRC RX has received enough header bytes to make a routing decision for the packet. If the packet is destined for a subsequent node, the PRC RX can try to send the packet directly to the next node by reserving an NI TX. If the PRC RX is able to establish a cut-through, the PRC RX then sends the data it has accumulated to that transmitter and reconfigures the NI RX to forward data directly to the reserved NI TX, bypassing the PRC entirely (as indicated by the dashed line in Figure 2). When the packet has cleared the node, the NI RX automatically reconfigures itself to forward the next packet to its associated PRC RX.

Reception/Buffering: When a packet buffers at the local node, however, the PRC RX simply collects the bytes from the NI, reaccumulating the CRC while transferring each word of the packet to the memory interface. These words are assembled into cache lines and written into pages in the host memory. The PRC event queue logs the arrival of each page, noting the address and size. At the end of the final page of the packet, the PRC RX 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.

4 Managing High Connectivity

Besides managing traffic originating and terminating at the node, the host may have to manage in-transit traffic that cannot cut through. Because of the concurrency at the physical links, the host must process a large number of transmission and reception events, and service each channel in a fair and efficient manner. SPIDER's host interface facilitates fine-grain bandwidth allocation between channels while increasing throughput and minimizing the intrusion on the host. Using the primitives exported by SPIDER, the host can exercise dynamic, fine-grain control over the individual transmission and reception channels.

4.1 Host Interface

To facilitate the coexistence of different classes of traffic, the host may support various priority-based scheduling policies. Implementing priority-based queueing entirely in hardware incurs higher cost and limits flexibility; implementing it entirely in software increases overhead and limits throughput. The scheduling queues are therefore implemented partially on SPIDER, as short FIFO page tag queues, allowing the host to employ different algorithms and arbitrary queue structures in software to schedule messages and packets. The page tag queues for outgoing channels form the *logical head* of these scheduling queues; they are fed from the host-managed scheduling queues through the memory-mapped control interface. The event queue, which logs SPIDER's use of pages for both transmission and reception, forms the *logical tail* of the internal event queues maintained by the host. The host can parse the logged events into several internal event queues, possibly corresponding to different channel priorities, and influence packet scheduling and free-list maintenance based on this feedback. Using this information the host can reconstruct received packets and update internal state information for each active channel.

Even though the control port introduces a physical serialization point, the primitives exported to the host preserve the channel abstraction maintained inside SPIDER. Providing transmission and reception page tag queues for each virtual channel enhances SPIDER's throughput by preserving inter-channel concurrency. A single, shared event queue simplifies event logging, and effectively decouples the host from the events generated by SPIDER during packet transmission and reception. The host can handle groups of events by draining the event queue through the control port. Together with selective interrupt masking, this effectively reduces the number and frequency of interrupts delivered to the host, and amortizes the cost of fielding an interrupt over multiple events. The host can further avoid the relatively high cost of fielding interrupts by polling SPIDER instead, further bounding the potential intrusion by in-transit traffic.

Because SPIDER views a packet as a collection of one or more pages, it places few restrictions on packet format or size. These pages could be partially-filled and non-contiguous in memory (similar to MBUFs and MBUF clusters [32]). This allows a sending host to generate the packet header on a separate page while leaving the data pages untouched, minimizing data-copying overhead and host memory fragmentation. SPIDER coordinates memory transfer for packet transmission and reception, insulating the host from direct involvement with data movement. To avoid replicating bus arbitration logic for each channel and to minimize host bus loading, the PRC multiplexes the active channels at the granularity of 32-bit words. Channel multiplexing significantly reduces the pin-out of the PRC, allowing an integrated, single-chip solution. The internal design bounds worst-case access latency, guaranteeing forward progress for each active channel. The external cache interacts with the host memory using block transfers of cache lines, amortizing the overhead of bus arbitration over multiple words.

4.2 Host Policies

The fine-grain coordination between the host and SPIDER enables the software protocols to efficiently manage different traffic characteristics and quality-of-service requirements. To support a mix of traffic classes, the host can assign individual traffic classes to distinct virtual networks, precluding channel contention in one class from directly influencing the other classes. The host can control traffic mixing on these virtual channels/networks by maintaining distinct scheduling queues and appropriate multiplexing policies. Within SPIDER, packets on different virtual networks interact only to compete for access to the physical link. Careful selection of switching and routing schemes, coupled with fine-grain arbitration between the virtual networks, allows the multiple traffic classes to share communication bandwidth without affecting the performance of each class.

Since the channel abstraction is preserved across SPIDER's control interface, the host can efficiently retrieve status information for all virtual channels through a single control-interface primitive. The host uses this feedback to determine the rate at which data is fed into the active channels. The host can exercise fine-grain control over guaranteed traffic, while pipelining packet transmissions to support high-throughput best-effort packets. While SPIDER supports low-level cut-through switching schemes, the host at intermediate nodes can also implement *partial cut-through* [24, 33] for large packets which may be buffered because of busy outgoing links. This technique improves performance by overlapping the forwarding of buffered pages with the arrival of subsequent pages of a packet.

SPIDER facilitates the detection of anomalous network conditions, such as congestion or faulty links, by software protocols. Since SPIDER tags each incoming and outgoing packet with a timestamp, the host can determine the latency experienced by incoming packets. Similarly, since SPIDER logs the outcome of the CRC check for each arriving packet, the host can diagnose intermittent link faults. The notification FIFOs provide a rendezvous point between each PRC RX and the host, and are used to diagnose and respond to dynamic network conditions. For example, each PRC RX can relay the cut-through statistics for recently arrived packets or report dropped packets. Using the notification FIFOs, the host can direct a PRC RX's response to the prevailing network conditions.

5 Evaluation

To capture the interaction between routing, switching, and host policies, we have developed pp-mess-sim (Point-to-Point MESS SIMulator) [29]. Implemented in C++, pp-mess-sim is an object-oriented discrete-event simulation environment for evaluating point-to-point network adapters. Using a high-level specification language, the user can select the interconnection topology and the communication patterns generated by each node in the network. These patterns stem from a collection of independent traffic classes, each with its own performance metrics and packet characteristics. The simulator allows the derivation of packet length, interarrival times, and target nodes from a variety of stochastic processes. To evaluate traffic mixing, the simulator associates each traffic class with a routing and switching scheme on a set of virtual networks.

Besides providing a general framework for evaluating adapter architectures, pp-mess-sim includes a cycle-level model of SPIDER that captures the details of flow control, resource arbitration, and microcode execution. The simulations capture the activity at SPIDER's memory and control interfaces, including memory access delay and packet scheduling. Although SPIDER supports a variety of network topologies and packet sizes, the simulation studies focus on fixed-length packets traveling in a 6×6 wrapped square mesh (torus). Most of the experiments employ dimension-ordered routing, which requires a packet to travel along a minimal path completely in the x-direction before proceeding in the y-direction to reach the destination node [9, 28]. For the simulation results below, each node independently generates packets with exponentially-distributed inter-arrival times and uniform random selection of destination nodes, unless otherwise stated.

5.1 Switching

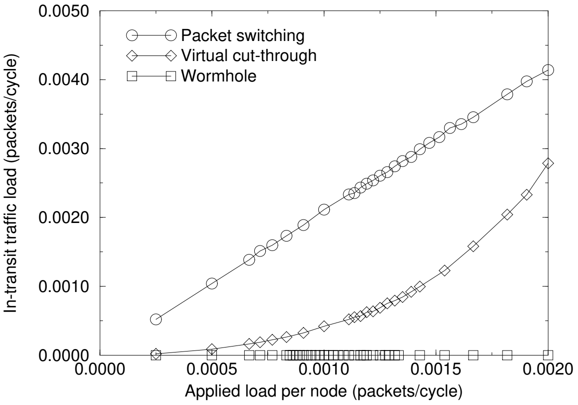
In defining how packets flow through the network, the various switching schemes stress different resources at intermediate nodes. Figure 3(a) shows the average rate of in-transit packet arrivals at a node as a function of the switching scheme and the traffic load. These experiments employ dimension-ordered routing, with wormhole packets utilizing two virtual channels to guarantee deadlock-avoidance [27]. As expected, packet switching imposes the highest in-transit load on the host, regardless of the traffic load. Wormhole switching, on the other hand, offers complete insulation of the host from in-transit packets by using channel resources to store blocked packets. When the traffic load increases, however, channel contention causes wormhole latencies to become worse than those with packet switching, as seen in Figure 3(b). This contention also results in a dramatically lower peak throughput [26, 34], as seen in Figure 3(c).

By removing blocked packets from the network, virtual cut-through offers lower latencies under heavy loads than wormhole switching. Virtual cut-through also provides excellent latencies under light loads, but as seen in Figure 3(a), it does not completely insulate the host from in-transit packets. As an effective compromise, a best-effort traffic class can employ deadlock-free wormhole routing but allow lightly-loaded hosts to buffer blocked packets to alleviate contention. By using the notification FIFOs, individual nodes can alternate between stalling and buffering any blocked in-transit packets, depending on current host conditions. Dynamically selecting the switching scheme allows hosts to insulate themselves from in-transit packets when necessary or to accept additional communication overhead in order to improve network throughput.

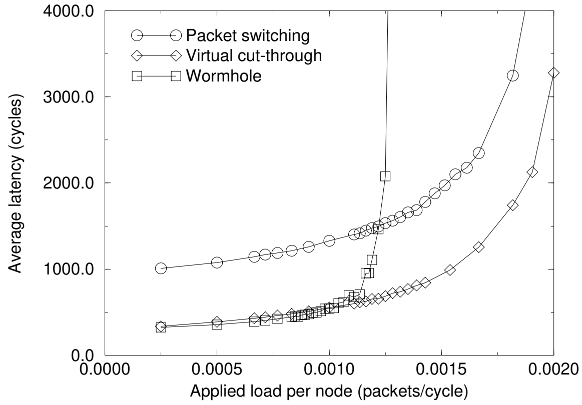
5.2 Routing

The benefits of cut-through switching depend largely on the likelihood of cut-through at each node. Adaptive routing algorithms can significantly increase this probability, especially in the presence of non-uniform traffic patterns. Figure 4 compares the performances of dimension-ordered routing and an adaptive minimal path algorithm, in a network containing both uniform and non-uniform traffic.[3] The adaptive algorithm used allows a packet to consider an alternate

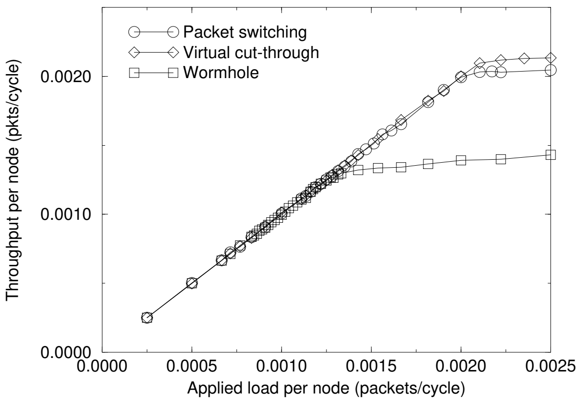
[3] To generate the non-uniformity, a single node in the network was made the "hot" node by having its adjacent neighbors transmit additional packets to it. Each neighbor generates an additional 0.02 packets per cycle to send to the "hot" node, possibly emulating the traffic near a server.



(a) Load on intermediate nodes

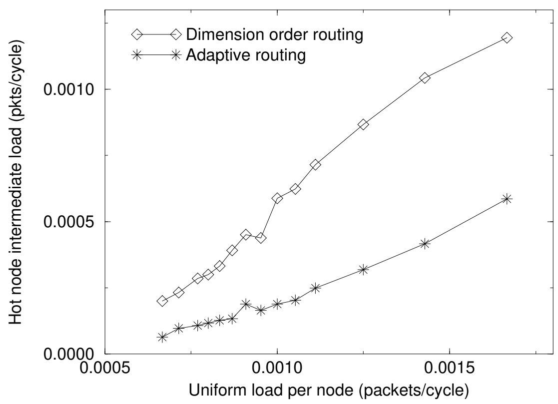


(b) Average packet latency

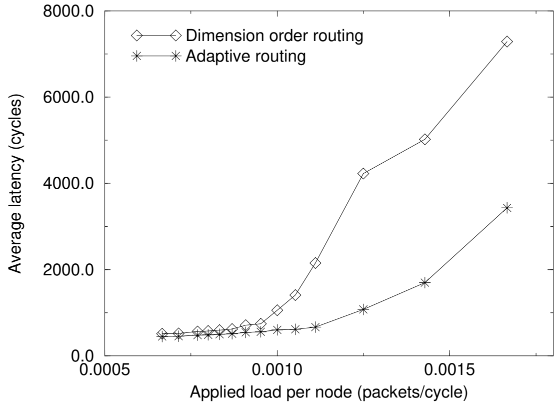


(c) Average throughput

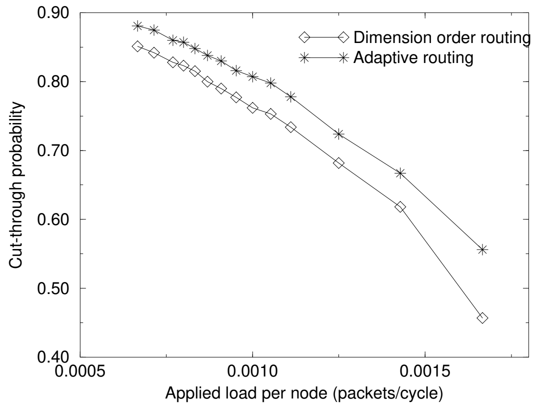
Figure 3: Comparison of switching strategies using 64-byte packets.



(a) Intermediate traffic load on hot-spot node

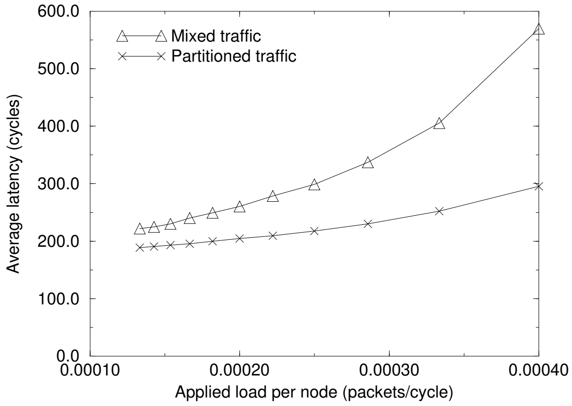


(b) Mean packet latency of uniform traffic

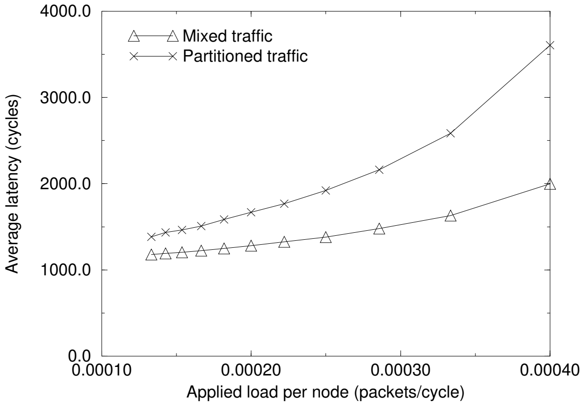


(c) Cut-through probability of uniform traffic

Figure 4: Static and adaptive routing using virtual cut-through with 64-byte packets.



(a) Average latency for short packets



(b) Average latency for long packets

Figure 5: Partitioned and mixed traffic with 32-byte and 256-byte packets.

link along any shortest path when the first-choice link is busy. In this experiment, both algorithms use virtual cut-through switching; the amount of non-uniform traffic in the network is held constant while the level of uniform background traffic is varied.

As shown in Figure 4(a), adaptive routing significantly reduces the number of intermediate packets seen at the hot-spot node compared to dimension-ordered routing. Allowing the in-transit packets to circumvent this congestion relieves the hot-spot node, which is busy with additional traffic destined for it. Avoiding the congestion also increases the probability of cut-through and hence, decreases the average packet latency as shown in Figures 4(b) and 4(c).

While this experiment shows the benefits of adaptive routing in a given environment, using SPIDER, the host can select its routing policies to suit a wide range of operating environments. The host, for instance, may find that that the performance of dimension-ordered routing is comparable to adaptive routing in low-connectivity networks and in networks with uniform traffic patterns. The use of adaptive routing in these cases would be offset by the protocol overhead in reordering packets at the destination nodes, making static routing a better alternative. In contrast, adaptive algorithms could be used for messages traveling longer distances where the benefits of the increased cut-throughs outweigh this overhead. The host, with the variable-length packet features of SPIDER, could also send entire messages in one packet using adaptive routing and avoid reordering costs altogether. In each case, the PRC RX can use the routing header to determine how to route individual packets. Finally, adaptivity facilitates fault-tolerance; by using the notification FIFOs in the PRC, the host can diagnose network failures and change routing policies in response to them. With these mechanisms, SPIDER can tailor its routing algorithms to the needs of its applications and to the state of the network.

5.3 Traffic Partitioning

Since the various routing and switching schemes have diverse performance characteristics, SPIDER allows the host to tailor communication policies to meet the requirements of each traffic class. Figure 5 illustrates the effects of partitioning traffic onto separate virtual networks according to packet length. In the mixed traffic experiment, short and long packets share a pair of virtual networks; the other experiment assigns each traffic class to its own virtual network. As shown in Figure 5(a), mixing the two classes of traffic adversely affects the average latency of the short packets. When the two classes share virtual channels, a short packet may wait behind one or more long packets to reserve an outgoing channel. Avoiding interference with other packets places a tighter bound on end-to-end latency. The host may, for example, dedicate a virtual network to infrequent control messages; this provides a fast path for important messages, while allowing best-effort traffic to utilize any remaining bandwidth.

While the short packets benefit from partitioning the traffic, the long packets incur additional delay as shown in Figure 5(b). When both classes shared the virtual networks, a packet could utilize either virtual channel on a physical link in its route. In effect, additional virtual channels provide greater flexibility and adaptivity in establishing cut-throughs for in-transit packets. Relegating long packets to a single virtual network increases their end-to-end latency by reducing the cut-through probability. Depending on the relative importance of the traffic classes, the host uses virtual channels to limit interference between different traffic classes or to reduce channel contention within each class.

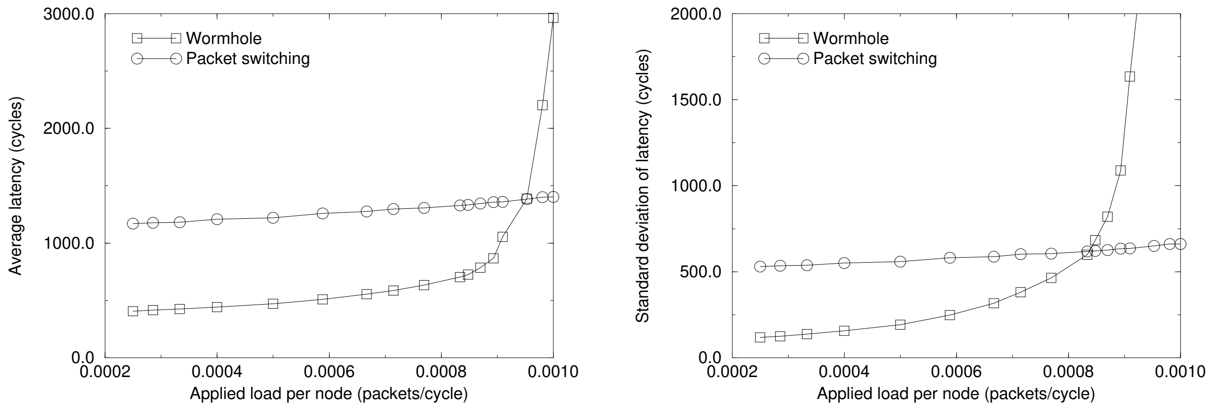
Increasing the number of virtual channels on each link allows the host more flexibility in reducing contention [26] and partitioning traffic. Figure 6 evaluates a configuration where each physical link has three virtual channels, two allocated to best-effort packets for deadlock-free wormhole routing [27] and one dedicated to guaranteed traffic using packet switching. The experiment varies the best-effort traffic load, while injecting guaranteed traffic at a constant rate.[4] Figure 6(a) shows the average end-to-end packet latency for each class of traffic; Figure 6(b) shows the standard deviation of latency for all packets traveling exactly five hops. Both the average latency and predictability of the guaranteed packets are largely unaffected by the best-effort traffic, due to fine-grain arbitration amongst the virtual channels. Channel contention on the best-effort virtual networks does not influence the guaranteed packets, since blocked wormhole packets temporarily stall in their own virtual network instead of consuming any physical link or buffer resources.

6 Conclusion

The location and semantics of the boundary between the network adapter and the software protocols determine not just the performance of data transfer, but also the flexibility afforded the software to intelligently manage this transfer. In point-to-point distributed systems, flexible hardware support for routing and switching improves performance and facilitates the coexistence of traffic with varying characteristics and quality-of-service requirements. Although it is possible to construct a point-to-point distributed system using replicated single-port communication hardware at each node, cost and performance considerations necessitate an efficient, integrated design for the network adapter.

In this paper we have presented and evaluated SPIDER, a network adapter featuring flexible hardware support for point-to-point distributed systems. Protocol support and media access in SPIDER stem from distributed systems, while the low-level packet routing and switching schemes derive from the point-to-point parallel computing domain. While network adapters for distributed systems typically manage a single conduit to the interconnection fabric, SPIDER manages communication on *multiple* network links. Cycle-level simulations justify the flexible support and demonstrate the utility of traffic partitioning using virtual channels. Based on

[4] Each node generates a guaranteed packet once every 1500 cycles.



(a) Average packet latency

(b) Standard deviation of latency ($h = 5$)

Figure 6: Partitioned packet switching and wormhole switching using 64-byte packets.

this evaluation, we plan to increase the number of virtual channels per link by reducing the number of PRC RXs and allowing each PRC RX to service multiple virtual channels.

In the current design, SPIDER interfaces to the Ironics IV-3207 [35], a VMEbus-based 68040 card, through a daughterboard interface to the processor memory bus. An IV-3207 card, coupled with the SPIDER daughterboard, can serve either as a network-based uniprocessor handling both application and communication tasks or as a dedicated network host for the node. Since the daughterboard interface is used for both control and data, the performance of SPIDER is limited by the bandwidth of the host memory bus. To provide additional bandwidth for data transfers, we are modifying the design such that these transfers utilize the VME bus; the control port still connects to the daughterboard interface. To eliminate the intrusion on the host memory, we will provide separate buffers on SPIDER for in-transit packets. This would enable the host to perform buffer management and scheduling for in-transit packets without consuming local memory resources.

SPIDER facilitates fine-grain bandwidth allocation between virtual channels while minimizing the intrusion on the host. It exploits the concurrency between the virtual channels internally, and exports the channel abstraction across the host interface. This allows host software to employ channel management policies that support traffic partitioning and exploit inter-channel concurrency. Using SPIDER-based network hosts, we plan to study point-to-point distributed systems with a variety of regular and irregular topologies. SPIDER's design enables flexible host policies regarding processor, memory, and bandwidth allocation in point-to-point distributed systems.

Bibliography

[1] K. Li and P. Hudak, "Memory coherence in shared virtual memory systems," *ACM Trans. Computer Systems*, vol. 7, no. 4, pp. 321–359, November 1989.

[2] H. Kung, R. Sansom, S. Schlick, P. Steenkiste, M. Arnould, F. J. Bitz, F. Christianson, E. C. Cooper, O. Menzilcioglu, D. Ombres, and B. Zill, "Network-based multicomputers: An emerging parallel architecture," in *Supercomputing 91*, pp. 664–673. IEEE, ACM, New York, NY, USA, November 1991.

[3] E. A. Arnould, F. J. Bitz, E. C. Cooper, H. T. Kung, R. D. Sansom, and P. A. Steenkiste, "The design of Nectar: A network backplane for heterogeneous multicomputers," in *Proc. Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, pp. 205–216. ACM, April 1989.

[4] Y. Oie, T. Suda, M. Murata, D. Kolson, and H. Miyahara, "Survey of switching techniques in high-speed networks and their performance," in *IEEE INFOCOM*, pp. 1242–1251, June 1990.

[5] D. Cohen, G. G. Finn, R. Felderman, and A. DeSchon, "The use of message-based multicomputer components to construct gigabit networks," *Computer Communication Review*, vol. 23, no. 3, pp. 32–44, July 1993.

[6] W. Athas and C. Seitz, "Multicomputers: Message-passing concurrent computers," *IEEE Computer*, pp. 9–24, August 1988.

[7] X. Zhang, "System effects of interprocessor communication latency in multicomputers," *IEEE Micro*, pp. 12–15,52–55, April 1991.

[8] D. Talia, "Message-routing systems for transputer-based multicomputers," *IEEE Micro*, pp. 62–72, June 1993.

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

[10] S. Konstantinidou and L. Snyder, "Chaos router: Architecture and performance," in *Proc. Int'l Symposium on Computer Architecture*, pp. 212–221, May 1991.

[11] W. J. Dally, J. A. S. Fiske, J. S. Keen, R. A. Lethin, M. D. Noakes, P. R. Nuth, R. E. Davison, and G. A. Fyler, "The Message-Driven Processor: A multicomputer processing node with efficient mechanisms," *IEEE Micro*, pp. 23–39, April 1992.

[12] A. L. Davis, "Mayfly: A general-purpose, scalable, parallel processing architecture," *Lisp and Symbolic Computation*, vol. 5, no. 1/2, pp. 7–47, May 1992.

[13] C. L. Seitz and W. Su, "A family of routing and communication chips based on the Mosaic," in *Symp. on Integrated Systems: Proc. of the Washington Conf.*, 1993.

[14] K. Ramakrishnan, "Performance considerations in designing network interfaces," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 2, pp. 203–219, February 1993.

[15] V. Jacobson, "Efficient TCP implementation," in *ACM SIGCOMM'90 Tutorial*, September 1990.

[16] O. Menzilcioglu and S. Schlick, "Nectar CAB: A high-speed network processor," in *Proc. Int. Conf. on Distributed Computer Systems*, pp. 508–515, May 1991.

[17] A. Krishnakumar and K. Sabnani, "VLSI implementations of communication protocols – a survey," *IEEE Journal on Selected Areas in Communications*, vol. 7, no. 7, pp. 1082–1090, September 1989.

[18] C. Dalton, G. Watson, D. Banks, C. Calamvokis, A. Edwards, and J. Lumley, "Afterburner," *IEEE Network Magazine*, pp. 36–43, July 1993.

[19] H. Kanakia and D. R. Cheriton, "The VMP network adapter board (NAB): high-performance network communication for multiprocessors," *Proceedings of the SIGCOMM Symposium*, pp. 175–187, August 1988.

[20] G. Chesson, "XTP/PE overview," in *Conference on Local Computer Networks*, October 1988.

[21] J. Dolter, S. Daniel, A. Mehra, J. Rexford, W. Feng, and K. G. Shin, "SPIDER: Flexible and efficient communication support for point-to-point distributed systems," Technical Report CSE-TR-180-93, University of Michigan, October 1993. To appear in *Proc. Int. Conf. on Distributed Computing Systems*, June 1994.

[22] D. Ferrari, "Client requirements for real-time communication services," *IEEE Communications Magazine*, pp. 65–72, November 1990.

[23] M. Zitterbart, B. Stiller, and A. N. Tantawy, "A model for flexible high-performance communication systems," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 4, pp. 507–518, May 1993.

[24] P. Kermani and L. Kleinrock, "Virtual cut-through: A new computer communication switching technique," *Computer Networks*, vol. 3, no. 4, pp. 267–286, September 1979.

[25] D. D. Kandlur, K. G. Shin, and D. Ferrari, "Real-time communication in multi-hop networks," in *Proc. Int. Conf. on Distributed Computer Systems*, pp. 300–307, May 1991.

[26] W. Dally, "Virtual-channel flow control," *IEEE Trans. Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194–205, March 1992.

[27] 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.

[28] L. Ni and P. McKinley, "A survey of wormhole routing techniques in direct networks," *IEEE Computer*, pp. 62–76, February 1993.

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

[30] *Am79168/Am79169 TAXI^{tm}-275 Technical Manual*, Advanced Micro Devices, ban-0.1m-1/93/0 17490a edition.

[31] D. D. Kandlur and K. G. Shin, "Reliable broadcast algorithms for HARTS," *ACM Trans. Computer Systems*, vol. 9, no. 4, pp. 374–398, November 1991.

[32] S. J. Leffler, M. K. McKusick, M. J. Karels, and J. S. Quarterman, *The Design and Implementation of the 4.3BSD Unix Operating System*, Addison Wesley, May 1989.

[33] S. Abraham and K. Padmanabhan, "Constraint based evaluation of multicomputer networks," in *International Conference on Parallel Processing*, pp. I–521–I–525, 1990.

[34] J. Ngai and C. Seitz, "A framework for adaptive routing in multicomputer networks," in *Symposium on Parallel Algorithms and Architectures*, pp. 1–9, June 1989.

[35] *IV-3207 VMEbus Single Board Computer and Multiprocessing Engine User's Manual*, Ironics Incorporated, 1991 edition.