Analysis and Implementation of Hybrid Switching

Kang G. Shin and Stuart W. Daniel

Real–Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109–2122
E-Mail Address: {kgshin,stuartd}@eecs.umich.edu

Abstract

The *switching* scheme of a point-to-point network determines how packets flow through each node, and is a primary element in determining the network's performance. In this paper, we present and evaluate a new switching scheme called *hybrid* switching. Hybrid switching dynamically combines both virtual cut-through and wormhole switching to provide higher achievable throughput than wormhole alone, while significantly reducing the buffer space required at intermediate nodes when compared to virtual cut-through. This scheme is motivated by a comparison of virtual cut-through and wormhole switching through cycle-level simulations, and then evaluated using the same methods. To show the feasibility of hybrid switching, as well as to provide a common base for simulating and implementing a variety of switching schemes, we have designed SPIDER, a communication adapter built around a custom ASIC, the *Programmable Routing Controller* (PRC).

1 Introduction

The effectiveness of a parallel or distributed system is often determined by its communication network. Many distributed and parallel applications require the network to provide low latency communications in order to operate efficiently, while others may require the network to handle a large amount of traffic. In addition, the burden placed on the host to handle communication-related activities should be minimized.

One of the key factors that determines how well a point–to–point network meets applications' requirements in these areas is its switching scheme(s). Wormhole [1] and virtual cut-through [2] switching are two common schemes for forwarding packets through a point–to–point interconnection network. Both are "cut-through" switching schemes that decrease packet latencies by immediately forwarding incoming packets to idle output links. In this paper, we compare the impact of each scheme upon packet latency, the maximum network throughput, and the resources required for buffering packets at intermediate nodes. Based on this evaluation, we then propose and evaluate a "hybrid" switching scheme that combines the salient features of both schemes.

1.1 Background

Virtual cut-through and wormhole switching differ in how they handle packets that cannot immediately proceed to the next node because the appropriate output links are busy with other traffic. Virtual cut-through switching buffers blocked packets at the local node, releasing the incoming link(s), but wormhole switching stalls the packet in the network, while holding any links the packet has acquired. Since packets never buffer at intermediate nodes, nodes only handle packets destined for them. Stalling the packet in the network, however, consumes network resources to "store" the packet, effectively dilating the packet's length. Virtual cut-through, on the other hand, minimizes the network bandwidth consumed by packets, but uses memory and control resources at intermediate nodes to store blocked packets.

In this paper, virtual cut-through and wormhole switching are shown to have their strengths and weaknesses. Virtual cut-through switching provides better throughput and lower latencies at heavy loads at the cost of buffering in-transit packets, while wormhole switching only requires few small buffers and completely isolates nodes from in-transit packets. One alternative to improving wormhole switching's performance at higher loads would be to *selectively* buffer blocked packets; this would free some network resources sooner while still isolating nodes from much of the in-transit traffic.

Virtual cut-through and wormhole switching are both cut-through switching schemes, but their performance may differ drastically under different traffic loads. For low traffic loads, the latencies of both schemes are almost identical. This is because in a lightly-loaded network the probability of blocking is very small and the latency is then determined primarily by the length of the packet and the link transmission time. As the traffic load increases, however, the probability of blocking increases, as does the likelihood of blocking other packets. Consequently, networks that use wormhole switching generally saturate from contention well before they exhaust their bandwidth [3, 4]. The effects of this contention can be reduced by increasing the number of virtual channels per physical link [4]. Since either wormhole or virtual cut-through switching may yield shorter packet latencies, depending on the network traffic and the number of hops the packet must travel, it is advantageous to support both switching schemes in order to adapt to a wider range of circumstances. Furthermore, a network which can dynamically switch from one scheme to the other can respond to the offered traffic load and the needs of the system's applications.

To address these tradeoffs, Section 4 introduces and evaluates a hybrid switching scheme which balances the use of network resources against the use of memory resources for storing blocked packets. This hybrid scheme decides whether to buffer or stall blocked packets based on a field within the routing header; this field identifies the number of links the packet can hold while stalling in the network. If this threshold is exceeded, the blocked packet buffers.

To demonstrate the feasibility of supporting multiple schemes on a single platform, Section 2 describes SPIDER, a front-end communication interface that supports a wide range of routing and switching schemes. In Section 3, we compare the performance of virtual cut-through and wormhole switching operating on SPIDER. This comparison focuses on three metrics: the mean communication latency, the memory resources required by each scheme, and the maximum achievable throughput of the network. In Section 4, we introduces hybrid switching and evaluate it relative to both virtual cut-through and wormhole switching. The paper concludes with Section 5, which summarizes our main contributions and future directions.

2 A Flexible Router Architecture

In order to isolate and take advantage of the differences in performance between cut-through switching schemes, we have developed SPIDER (*Scalable Point-to-Point Interface DrivER*) [5, 6], a communication adapter that implements multiple switching schemes. SPIDER is microprogrammable with a wide range of routing and switching schemes, providing an ideal platform for experimenting with and comparing routing and switching schemes.

2.1 Existing Router Architectures

Several routers that use wormhole switching have been developed [1, 7–9]. In general, the design of these routers has emphasized speed and simplicity, with the routing algorithm hardwired into the system. Each router only supports a small number of links, allowing a crossbar to be used to transfer data without internal blocking. Furthermore, the short internode distances allow flow control and parallel internode links to be efficiently implemented. The Vulcan Switch chip [10] uses an interesting variation, by adding a central, dynamically allocated queue to the switching element. This queue improves throughput by buffering "chunks" of packets in the blocking switch, rather than buffering the flits in several different switches and blocking those channels.

Virtual cut-through routers typically provide better throughput under heavy loads at the cost of increased buffer requirements. The Mayfly Post Office [11], uses several (hardwired) routing algorithms and provides an internal buffer for packets that cannot cut through, but only supports virtual cut-through switching. It uses a shared internal bus to transfer packets between ports and also to and from the buffer pool. The Chaos router [12] also provides an internal buffer for packets, but this buffer is much smaller — the router deroutes packets to avoid blocking or dropping them.

2.2 SPIDER

SPIDER is designed to support multiple switching schemes, including store-and-forward, virtual cut-through, and wormhole switching. Supporting the first two schemes requires that the node be able to buffer several packets simultaneously so that packets can be received without blocking. SPIDER provides this using a demand-driven, time-multiplexed memory interface that shares memory bandwidth between all active injection and reception ports. Similarly, cut-through switching schemes require a high-bandwidth switch for transferring data between incoming and outgoing channels. In SPIDER, this is provided by a demand-slotted, *time division-multiplexed* (TDM) bus with bandwidth equal to the physical links. Access to the bus is regulated by a binary priority-tree arbiter [13, 14].

2.3 SPIDER Components

As shown in Figure 1, SPIDER manages bidirectional communication with up to four neighboring nodes, with three virtual channels [4] on each unidirectional link. The programmable routing controller (PRC), a 231-pin, 1.3×1.5 cm custom integrated circuit, is the cornerstone of SPIDER [5, 6, 13]. The 12 *Transmitter Fetch Units* (TFUs) 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 Transmitters* (NI TXs) and *Network Interface Receivers* (NI RXs) perform the necessary interleaving of virtual channels to and from the physical links, on a word-by-word basis[1]. The network interface (NI) performs the media access and flow control on four pairs of AMD TAXI chips [15]; these TAXI transmitters and receivers control the physical links, providing a low-cost fiber-optic communication fabric.

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 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 TFU. The host transmits a packet by feeding this TFU 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.

[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.

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

2.4 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 TFU to initiate packet transmission. After reserving the NI TX, the TFU 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 TFU 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 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-

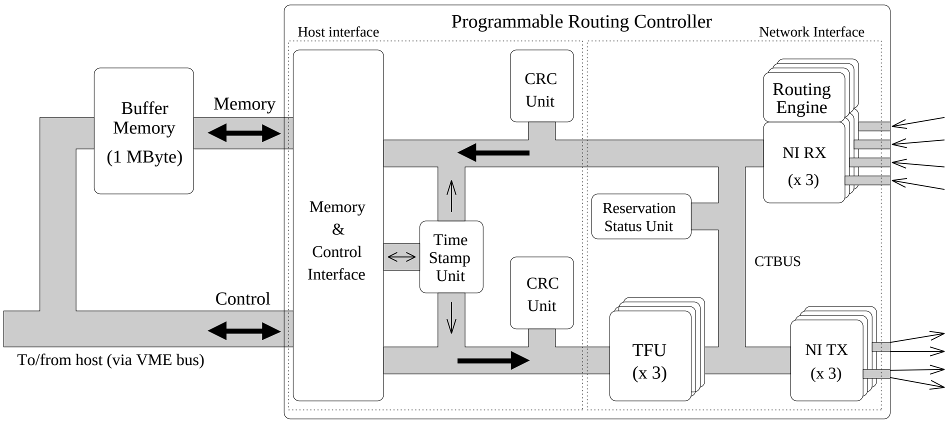


Figure 1: SPIDER

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 Comparing Wormhole and Virtual Cut-through Switching

To evaluate the performance of SPIDER and also to more accurately compare the performance of the various routing and switching schemes, we have developed a cycle-level discrete-event simulator [13, 16]. Written in C++, this simulator accurately models the flow of the individual bytes of packets through SPIDER. This captures features such as the low-level flow control, bus arbitration delays, and microcode execution time. While the simulator does not model the actual protocol software executing on the host, it does capture the effects of these protocols on packets that buffer at intermediate nodes.

This section presents the results of a set of experiments that vary the packet generation rate while holding other parameters constant. At each node, the interarrival time of packets for transmission conformed to a negative exponential distribution. Packet destinations were uniformly distributed across all of the nodes (except where otherwise specified). The simulations also used a fixed packet size of 64 bytes (except where specified).

To focus the experiments on the switching scheme, all packets use a static, dimension-ordered routing scheme [17]. Furthermore, most of the simulations use an unwrapped square mesh topology where only one virtual channel per link is required to prevent deadlock under wormhole switching. This allows the switching schemes to be compared with the same number of virtual channels.

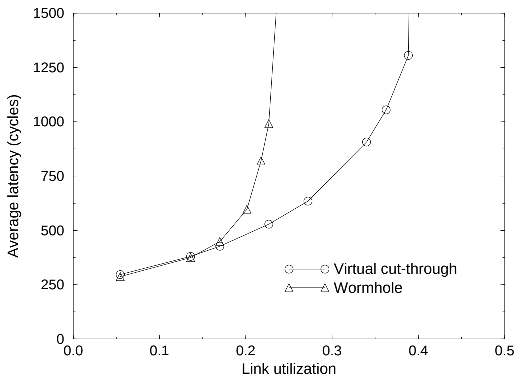


Figure 2: Packet delivery latencies for virtual cut-through and wormhole switching.

To collect the data, the network was first placed into a steady state and data collected for 2000 packets at each node. For latency, the standard error of the mean is less than 5 cycles for the 95% confidence interval on all traffic loads. When the network is saturated, however, this steady state cannot be achieved.

3.1 Latency

In Figure 2, the mean packet latency is shown as a function of the link utilization, which is given as a percentage of the maximum capacity of the network's physical links. When the offered load is low, the average packet latency is the same under both switching schemes. Wormhole, however, reaches saturation under lighter loads than virtual cut-through due to contention for channels, resulting in a dramatic increase in the mean packet latency. Saturation occurs at a link utilization of 0.2 in this experiment. Other experiments have shown that these trends are not significantly affected by packet length or the topology of the network.

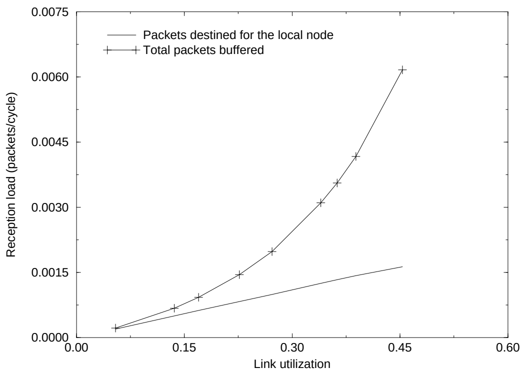


Figure 3: Rate of in-transit packet arrival

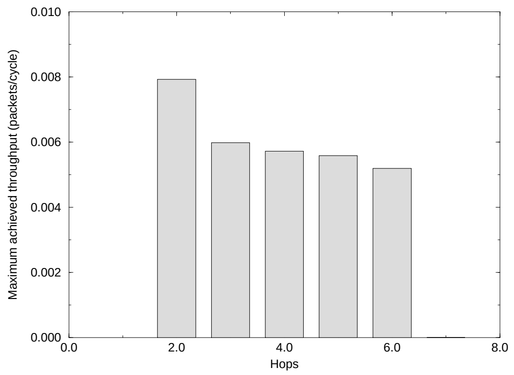


Figure 4: Maximum throughput for wormhole switching under a hop-uniform traffic load.

3.2 In-transit Load

While virtual cut-through can support a greater traffic load than wormhole, it also buffers packets at intermediate nodes. Each packet that buffers at a node consumes memory resources for its storage and control resources to process the header. Under wormhole switching, on the other hand, nodes only handle those packets destined for them.

The relative costs of the two schemes are illustrated for a node-uniform traffic load on an unwrapped 8×8 square mesh in Figure 3. This figure shows the average rate (in packets per cycle, per node) of packets buffering at a node using virtual cut-through switching. This rate is composed of two components: the "in-transit" rate and the "destination" rate. The former is the average rate of packets that are destined for other nodes buffering at a node, while the latter is the average rate of packets buffering at a node that are destined for that node. The in-transit rate is the region between the destination rate (the lower curve) and the total rate of packets buffering (the higher curve). At low loads, almost all packets successfully cut through and the in-transit arrival rate is very low. As the load increases, the probability of cut-through also drops, resulting in an increased in-transit packet arrival rate. When the network is in or near saturation, the arrival rate of in-transit packets surpasses the rate of packet generation. In this case, the load on the host for buffering and rescheduling these packets is severe.

3.3 Maximum Achievable Throughput

Wormhole and virtual cut-through switching are affected differently by packet distance. This can be directly shown by varying the average number of hops that packets travel. This was accomplished through a hop-uniform destination mapping, where every packet travels the same number of hops. In order to spread traffic uniformly through the network, a wrapped 8×8 square mesh (torus) is used with two virtual channels per link (the minimum to prevent deadlock under dimension-ordered routing).

Figure 4 shows the maximum throughput (in packets per cycle) of wormhole switching as a function of the hop count of packets. Using wormhole switching, the network saturates under a lighter link load as the packet distance increases. This is due to increased contention: packets are traveling more hops, and thus stalling more links when blocked. This has a snowball effect: blocked packets stall more links, and block other packets that may then block still other links. The overall effect, therefore, is to degrade the maximum achievable throughput.

Virtual cut-through switching does not exhibit this behavior, as it uses memory resources and not network resources to stall blocked packets. Its peak throughput is dependent upon the link load and not upon packet distance.

The maximum throughput of a network using wormhole switching can be increased by adding virtual channels [4], or by significantly enlarging the number of flits buffered at each node. Adding virtual channels on each link, on the other hand, improves throughput by allowing packets to "bypass" stalled packets. The primary cost is in the increased complexity of the crossbar connecting the reception channels to the transmission channels — either the size of the crossbar must be increased, or the arbitration becomes more complex [18]. Giving each virtual channel a flit buffer large enough to hold one packet should significantly improve throughput — each blocked packet only stalls a single link. Similarly, buffers capable of holding half of a packet's flits will prevent blocked packets from stalling more than two links.

There are significant differences in the performance of wormhole and virtual cut-through switching under different traffic loads. Wormhole switching requires fewer buffers than virtual cut-through, but its maximum throughput is relatively limited and dependent on packet distance. At heavy loads, virtual cut-through (as predicted) outperforms wormhole, but the cost of buffering in-transit packets can cancel out the performance gains. The following section presents a hybrid switching scheme that addresses the shortcomings of both schemes.

4 Evaluating Hybrid Switching

This section examines how hybrid switching provides a level of performance that bridges the gap between virtual cut-through and wormhole switching. We evaluate hybrid switching's performance relative to these schemes using the same metrics as the previous section.

4.1 Hybrid Switching

A "hybrid" switching scheme dynamically combines wormhole and virtual cut-through switching, using both network and memory resources to store blocked packets. There are a number of potential hybrid switching schemes that meet this requirement. To implement these schemes efficiently, however, the switching decisions should be based on information available in the packet header or at the local node.

In Section 3.3, we saw that increasing the number of links held by packets degraded the throughput achievable with wormhole switching. One method for improving wormhole's performance under heavier loads would be to relieve contention by buffering packets that cannot advance yet are stalling several links behind them. This scheme would avoid the long "tails" of stalled links held by blocked packets, reducing contention. Such a switching scheme would dynamically combine virtual cut-through and wormhole switching to provide improved packet latencies and a higher achievable throughput than wormhole alone, without buffering packets as often as virtual cut-through.

The hybrid algorithm used in the remainder of this paper decides whether to buffer or stall blocked packets based on a field within the routing header; this field identifies the number of links the packet can hold while stalling in the network. If this threshold is exceeded, the blocked packet buffers. The system can dynamically vary this threshold depending on the packet's needs or the current network load by changing the initial value of this header field.

Implementing the scheme is simple: a field in the routing header is set to h when the packet is generated and then decremented after every hop until it reaches 0. While $h > 0$, the packet will stall if blocked. Once $h == 0$, the packet buffers when blocked. Buffering the packet resets h to its initial value. Virtual cut-through and wormhole switching can be viewed as special cases of this algorithm: wormhole switching is equivalent to hybrid switching with $h = \infty$, while hybrid switching with $h = 0$ effectively implements virtual cut-through switching.

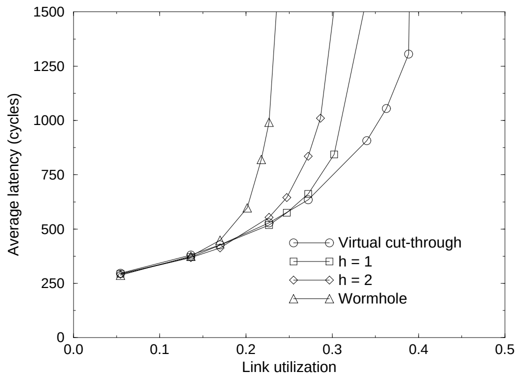


Figure 5: Packet delivery latencies for hybrid and wormhole switching.

The requirements for supporting hybrid switching are not much greater than those for supporting wormhole or virtual cut-through switching alone. When a router receives a packet, it must be able to determine how many hops the packet has traveled. If the link reservation fails, the router can then choose to buffer the packet. Due to the reduced in-transit load, the buffer requirements for hybrid switching are significantly reduced compared to virtual cut-through switching.

In the following simulations, all packets use the same dimension-order routing as in Section 3. As before, the simulations use a fixed packet size of 64 bytes, except where indicated otherwise.

4.2 Latency

In Figure 2, we saw that wormhole switching saturates from contention well before virtual cut-through, resulting in dramatically increased latencies. By preventing blocked packets from holding more than h links, hybrid switching decreases contention. The effects are shown in Figure 5, which compares the average packet latencies for wormhole switching, hybrid switching with $h = 1$, hybrid switching with $h = 2$, and virtual cut-through switching.

At very low loads, with a low probability of blocking, the mean latencies of the schemes are similar. Once this probability rises, however, hybrid switching provides lower packet latencies than wormhole switching. As h decreases, the network can handle a higher offered load without saturating. Higher values of h will resem-

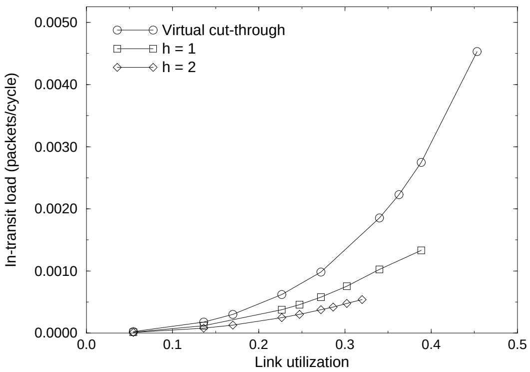


Figure 6: In-transit packet load for virtual cut-through and hybrid switching.

ble pure wormhole switching more closely — saturating at lower offered loads. These trends also hold over a range of packet sizes and network topologies.

4.3 In-transit Load

One of the primary advantages of wormhole switching is that it completely insulates nodes from in-transit traffic; the cost, however, is the consumption of network bandwidth by blocked packets. Virtual cut-through switching utilizes the network's bandwidth more efficiently, but can require nodes to handle large amounts of in-transit traffic (as shown in Section 3). By only buffering *some* blocked packets, hybrid switching significantly reduces this load.

A comparison of the in-transit load for hybrid switching and virtual cut-through switching is shown in Figure 6. This graph shows the arrival rate of in-transit packets for a range of offered loads. Even at low loads, with a very high probability of cut-through, hybrid switching significantly reduces the rate of in-transit traffic when compared to virtual cut-through. As the offered load increases, the probability of cut-through decreases and the in-transit load increases. At high loads, virtual cut-through switching uses at least $h + 1$ times more memory resources than the hybrid scheme, since the hybrid algorithm allows packets to buffer at most once every $h + 1$ hops. The actual reduction in buffering is often larger. For example, a packet traveling five hops using virtual cut-through may buffer up to four times, while hybrid with $h = 2$ will only buffer it at most once.

4.4 Maximum Achievable Throughput

Figure 7 shows the maximum achieved throughput (in packet-hops per cycle) as a function of the number of hops traveled by each packet. As in Figure 4, the applied traffic load is hop-uniform — every packet travels the same number of hops. The maximum throughput is only shown for those distances greater than h — when each packet travels h hops or less, hybrid switching is indistinguishable from wormhole switching.

Unlike wormhole switching and virtual cut-through, however, the maximum throughput for hybrid switching *increases* with the number of hops packets travel. This phenomenon can be explained by examining the proportion of packets in each case that have traveled more than h hops without buffering. As the average number of hops traveled by each packet increases, the percentage of packets that are willing to buffer if blocked increases. This alleviates contention in the network, preventing early saturation.

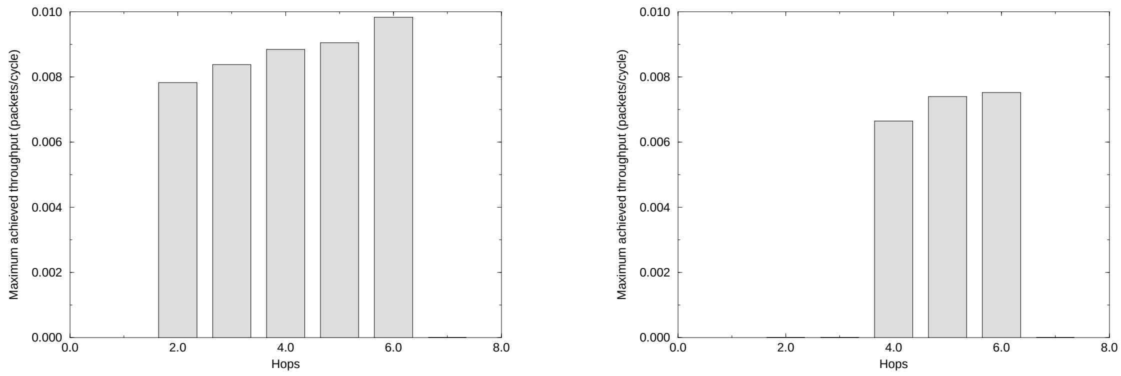
Hybrid switching with $h = 1$ resembles wormhole switching with a large, packet-sized buffer on each virtual channel, since both prevent blocked packets from stalling more than one virtual channel. Unlike wormhole, however, blocked packets do not always stall a link — if they are using more than h channels, they will buffer, freeing the channels for other packets.

In systems with large buffers for packets (such as SPIDER) and wrapped topologies, hybrid switching may use all of the virtual channels in the network. While packets that will stall when blocked must utilize deadlock-free routing schemes, packets where h has reached 0 may take advantage of available channels without regard to preventing deadlock, since they will buffer if blocked. This increases the probability of cut-through for packets by considering channels that could not otherwise be used.

4.5 Discussion

The simulations in this paper did not restrict the number of buffers at each node. When the packet buffers are implemented on the same die as the router, the number and size of the buffers is restricted. By buffering fewer packets than virtual cut-through, hybrid switching reduces the buffer space needed. In addition, hybrid switching schemes can take the available buffer space into account when deciding whether to buffer or stall a blocked packet. By buffering only packets that are currently holding several links and stalling others, hybrid switching can effectively utilize limited buffers.

This section has evaluated only one variant of hybrid switching. Another promising hybrid scheme uses a "credit" scheme to determine when to buffer a blocked packet. Under this scheme, each packet header contains a field indicating the maximum number of times it can be buffered — every time the packet buffers, the field is decremented. Once this value reaches 0, the packet will stall in the network. This scheme allows packets to stall more channels, but buffering other packets should prevent network congestion. The combination of a re-



(a) Hybrid, $h = 1$

(b) Hybrid, $h = 2$

Figure 7: Maximum throughput under a hop-uniform traffic load.

striction on the number of times a packet can buffer with h-hop hybrid switching also holds promise.

Hybrid switching also allows the system to dynamically determine (on a per-packet or system-wide basis) whether network or buffer resources are used to store blocked packets. This can be implemented by setting the initial value of h at the source of the packet to reflect whether the packet should consume more network or buffer resources when blocked. For example, large packets that will be traversing a large number of links may initially use larger values of h to reduce the number of times they buffer. On the other hand, systems requiring high bandwidth can use smaller values of h to shift the load to the network's buffers.

Hybrid switching uses both network and memory resources to store blocked packets, addressing the shortcomings of other cut-through switching schemes. Using network resources to store the packets can often have a snowball effect, creating contention throughout the network that limits throughput. Schemes that use memory resources, on the other hand, increase the system's communication overhead. Through hybrid switching, we attempt to balance these concerns. Potentially, the switching decision could be also based on the distance still needs to travel, or the number of buffers available at the local node. In addition, the decision could be time-based: packets could stall for some small amount of time if blocked in the hopes of being able to cut through, and then buffer. Alternately, packets that are blocked just short of their final destination could block in the network, while others that are blocked near their source would buffer. This would keep packets from blocking in the network more than once or twice.

5 Conclusions

The switching scheme used by a point-to-point network is a major factor in determining the latency, throughput, and overhead of communication. The various cut-through switching schemes all improve latency over store-and-forward switching (unless the network is saturated), but each has its strengths and weaknesses.

As we have shown in this paper, virtual cut-through does not limit the achievable network throughput but does impose a significant load on nodes for storing and retransmitting in-transit packets. Wormhole, on the other hand, stalls blocked packets in the network and does not require large buffers for blocked packets, it is cheaper to implement. Its maximum throughput, however, is limited by contention for outgoing links.

In this paper, we have introduced the concept of hybrid switching, which dynamically chooses whether to buffer or stall blocked packets in order to balance resource consumption. Using SPIDER and its simulator model, we plan to explore the potential of a number of hybrid switching schemes. In particular, we plan to examine the effects of different communication patterns on the switching schemes. Other investigations will compare hybrid switching with wormhole switching in the presence of packet-sized input buffers, fixed-size shared buffers, and additional virtual channels.

The hybrid switching scheme presented in this paper combines features of both wormhole and virtual cut-through switching by buffering a small fraction of blocked packets and limiting the number of links that blocked packets can hold. This significantly reduces the buffer requirements for in-transit packets when compared to virtual cut-through, while providing higher maximum throughput than wormhole switching. In this manner, hybrid switching bridges the performance gap between the other cut-through switching schemes.

Acknowledgements

The authors would like to acknowledge James Dolter's invaluable work in developing the simulator, as well as Jennifer Rexford and Wu-Chang Feng for helping improve it and for their discussions and insights.

The work reported in this paper was supported in part by the National Science Foundation under Grant MIP-9203895, and by the Office of Naval Research under Grant N00014-92-J-1080. The opinions, findings and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the view of the funding agencies.

References

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

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

[3] 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.

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

[5] J. Dolter, S. Daniel, A. Mehra, J. Rexford, W. Feng, and K. 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.

[6] S. Daniel, J. Rexford, J. Dolter, and K. Shin, "A programmable routing controller for flexible communications in point-to-point networks." Submitted to *International Conference on Computer Design*, 1995.

[7] S. Borkar, R. Cohn, *et al.*, "Supporting systolic and memory communication in iWarp," in *Proc. Int'l Symposium on Computer Architecture*, pp. 70–81, 1990.

[8] 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.

[9] D. Smitley, F. Hady, and D. Burns, "Hnet: A high-performance network evaluation testbed," Tech. Rep. SRC-TR-91-049, Supercomputing Research Center, Institute for Defense Analyses, December 1991.

[10] 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.

[11] A. L. Davis, "Mayfly: A general-purpose, scalable, parallel processing architecture," *Lisp and Symbolic Computation*, vol. 5, pp. 7–47, May 1992.

[12] K. Bolding, S.-C. Cheung, S.-E. Choi, C. Ebeling, S. Hassoun, T. A. Ngo, and R. Wille, "The Chaos router chip: Design and implementation of an adaptive router," in *Proc. VLSI*, September 1993.

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

[14] A. Kovaleski, S. Ratheal, and F. Lombardi, "An architecture and interconnection scheme for time-sliced buses in real-time processing," *Proc. Real-Time Systems Symposium*, pp. 20–27, 1986.

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

[16] 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.

[17] 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.

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