PDF file

ANALYSIS OF THE SCI RING

by

Steven L. Scott
James R. Goodman
and
Mary K. Vernon

Computer Sciences Technical Report #1055

November, 1991

Analysis of the SCI Ring

Steven L. Scott, James R. Goodman and Mary K. Vernon

Department of Computer Sciences
University of Wisconsin - Madison
1210 West Dayton Street
Madison, WI 53706

sls@cs.wisc.edu

Abstract

The Scalable Coherent Interface (SCI) is an emerging IEEE standard that provides computer-bus-like services via a set of fast, unidirectional links. SCI includes physical and logical levels, which provide a mechanism for connecting nodes and transmitting information, and a coherence level, which provides a variety of coherence, synchronization and message passing capabilities to multiprocessor systems. While many person-years of effort (including both industry and academic participants) have gone into designing a high performance, highly functional protocol, surprisingly little formal effort has been devoted to characterizing the performance of the design. This paper presents the first detailed performance study of the SCI ring, focusing at the logical level.

The paper presents an analytical performance model for the base protocol without flow control. The model is based on an M/G/1 queue, augmented to include the effect of packet trains on the mean and variance of the source transmission time. The model is validated against simulation results, and shown to be quantitatively accurate for uniform workloads, and at least qualitatively accurate for non-uniform workloads.

The paper also presents extensive simulation results for SCI rings, considering both uniform and non-uniform communication patterns. The flow control mechanism in the protocol is shown to effectively prevent node starvation and reduce the ability of nodes to unfairly consume ring bandwidth, but at the cost of decreased overall ring utilization. The SCI ring is also compared to a standard bus, modeled with a simple M/G/1 queue, and shown to provide substantially higher throughputs and lower latency than a bus with realistic clock speeds.

Steven Scott was supported by a fellowship from the Fannie and John Hertz Foundation. This work was also supported by National Science Foundation grant #CCR-892766.

1. INTRODUCTION

The Scalable Coherent Interface is a proposed IEEE standard (P1596) that provides very-high-performance, bus-like functionality to a large number of processor nodes [IEEE91, Jame90]. Using a packet-based communication protocol based on unidirectional links connected in a ring, it provides a shared-memory interface, including cache coherence, to the nodes. The protocol has been developed over a period of approximately four years, has included participation by representatives of dozens of companies and many universities and has assembled appropriate expertise in many different disciplines to solve the plethora of problems associated with a novel design.

The SCI includes protocols at three different levels: the physical level, the logical level, and the cache-coherence level. The logical level provides the protocol for reliably transmitting packets between nodes. The node interface consists of two unidirectional links, an input and an output, which are used to connect nodes together in the basic topology of a ring. The ring can in theory be arbitrarily large, but performance considerations lead to the expectation that a ring will be limited to a modest number of processors, numbering at most a few dozen and perhaps as few as two. Larger systems can be built by connecting together multiple rings by means of switches, that is, nodes containing more than a single interface.

The ring is unusual in that each node provides a bypass buffer capable of storing temporarily a packet arriving from its upstream neighbor while it is transmitting a packet. This buffer allows nodes to transmit concurrently rather than having to wait for a token, but results in long latency if all nodes happen to initiate transmission simultaneously on an idle ring. Because of the novel construction of the ring and the attendant clock rates achievable in the design, very high performance is expected, and a peak bandwidth of one gigabyte per second is easy to demonstrate. The nature of the protocol, however, makes both the achievable bandwidth and the observed latency much harder to predict. Reported here is the most detailed study to date attempting to analyze the performance of a single ring. It includes analyzing the performance under a variety of conditions, including the number of nodes in the ring, the service rate of queues, the size of packets, and the distribution of sources and destinations for the packets. In addition, a mechanism to assure fairness in the ring is investigated to assess its impact on the performance. The SCI protocol does provide priority scheduling, but this aspect of the protocol is not investigated in this study. The ring is studied both by the use of an analytical model and through simulation.

The remainder of the paper is organized as follows. Section 2 of the paper describes the protocol of the SCI logical layer. Section 3 describes the analytical model. Section 4 presents and analyzes the results of the study, and Section 5 summarizes the conclusions.

2. THE SCI LOGICAL-LEVEL PROTOCOL

The key idea behind the SCI logical level protocol is the use of unidirectional, point-to-point links that can be clocked at a rate independent of the signal latency between nodes. The basic building block of an SCI system is a *ring* (sometimes called a *ringlet*) of two or more nodes connected by these links. The protocol is designed such that (short of an actual hardware failure) messages are guaranteed to be accepted by all nodes that they pass through as they traverse the ring. Link-level acknowledgements, therefore, are not used and a node outputs a *symbol* of information on every clock cycle (there is no direct feedback from a node to its upstream neighbor). A packet might not be accepted into the sink queue at its *destination*, however, due to queue congestion. The protocol uses ring-level acknowledgements to deal with this issue.

2.1. Basic Protocol

This section presents a summary of the basic protocol. Details such as ring initialization and error detection/recovery are not covered, nor is all the functionality of the standard presented. Buffer management is also somewhat simplified. In order to avoid deadlock at a higher level, an actual system must classify end-to-end messages as either *requests* or *responses*, and must provide separate queue space for both. Since we are dealing only with the logical level protocol on a single ring, deadlock is not an issue, and we deal with unified queues. The coherence level of the SCI standard is not dealt with at all. More detail can be found in the standard [IEEE91].

A message traversing an SCI ring is sent from a *source* node to a *target* node in the form of a *send packet*. The target node then *strips* the send packet, and returns an *echo packet* around the remainder of the ring. This echo packet tells the source node whether the send packet was accepted by the target or whether it was rejected due to queue overflow or other error. If the packet was not accepted, then the source must retransmit it.

A send packet consists of a 16 byte header and an optional data component of up to 256 bytes. The header contains command and control information, a 16-bit CRC and a 64-bit memory address (16-bit node id and 48-bit intra-node address). Throughout most this paper we assume a data component size of 64 bytes, which corresponds to the SCI cache block size. Echo packets are 8 bytes long. The link width is 16 bits (the standard also defines a serial, fiber optic implementation, which we do not consider).

Figure 1 shows a block diagram of an SCI node and ring interface. A node transmits a symbol onto its output link on every network cycle. When a node has no packet to transmit, it sends an *idle symbol*, which contains housekeeping information related to flow control, priorities, *etc*. When a source desires to send a packet over the ring, it places the packet in its source queue. If the ring buffer is empty and the node is not currently transmitting a packet from the stripper, the send packet may be immediately output onto the ring. When a source packet is

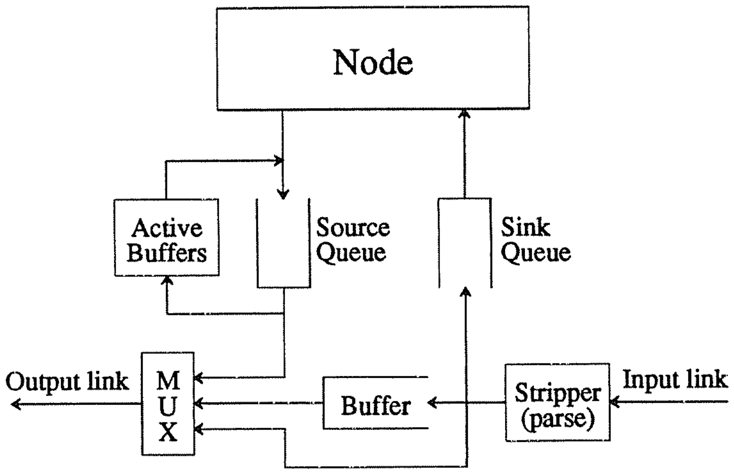


Figure 1: An SCI Node

transmitted, a copy must either must be saved at the head of the queue (thus blocking further transmissions) or placed into an optional *active buffer*. The copy is either discarded or used for retransmission when the echo packet for the packet is received.

Upon arrival at the downstream node, a send packet is parsed and either stripped or *passed* along the ring. In the absence of contention, a passing packet may be routed directly from the *stripper* to the output link. When a packet is to be passed but the source queue at the node is currently transmitting a packet, the passing packet is routed into the ring buffer instead. If a passing packet and a source packet are ready to transmit on the same cycle, the source queue is given priority and the passing packet is routed to the ring buffer.

When the source queue is done transmitting, if the ring buffer has accumulated any symbols, output resumes from the ring buffer (which may still be receiving symbols from the stripper). This is known as the *recovery stage*, and lasts until the ring buffer is completely emptied. The node is not allowed to transmit another source packet during the recovery stage. To empty the ring buffer, the node either must see gaps in the stream of incoming packets, or create gaps in the packet stream by stripping packets for which it is the target. During these gaps, the buffer can be drained while not being simultaneously filled. If the ring buffer is empty after a source transmission completes, then there is no recovery stage.

When a send packet reaches its target, a check is made for adequate sink queue space. The packet is stripped and either placed into the sink queue (space permitting) or discarded. The node uses the bandwidth created by stripping the packet either to insert idle symbols or to transmit symbols from the source queue or ring buffer. The last four symbols of the send packet are replaced with an echo packet that continues its way around the ring to the packet's source. At the source, the echo packet is matched with a saved send packet in an active buffer or at the head of the source queue, and the appropriate action is taken (discarding or retransmitting the send packet).

One last feature of the protocol that needs mentioning is that packets are always separated by at least one idle symbol. This allows the stripper to periodically delete an idle symbol, if necessary to adjust for a slowly varying clock period between neighbors (this is known as *elasticity*). It also assures timely distribution of priority and other information carried in the idle symbols. We do not consider elasticity or priorities here, but *do* require the intervening idle symbols. For the purposes of the basic model, this is equivalent to increasing the length of all packets by one symbol.

2.2. Flow Control

The basic protocol described above works fine for uniform traffic rates and routing distributions. However, it allows for nodes to be unfairly starved in the presence of certain non-uniform traffic patterns. Consider a node that partially fills its ring buffer during a source queue transmission. If the node then receives a continuous stream of passing packets, then its recovery stage can take arbitrarily long, denying it the chance to transmit another packet. For this reason, the SCI protocol includes a flow control mechanism that uses *go bits* in the idle symbols to enforce an approximate round robin ordering under heavy loads. The flow control mechanism is considerably complicated by a priority mechanism that partitions the ring's bandwidth between high and low priority nodes. We assume here that all nodes have equal priority, and present the simpler flow control mechanism that results.

Each idle symbol contains a go bit which is either set (making it a *go-idle*) or cleared (making it a *stop-idle*). The stripper passes all idles and passing packets (as well as echos for packets that it strips) to the transmitter stage of the node interface. When it strips a packet, it fills the empty slots with whatever kind of idle directly preceded the stripped packet. Whenever the stripper emits a go-idle, it continues to emit go-idles until the next packet boundary (this is called go bit *extension*).

When a node is not transmitting a packet from its source queue and is not in the recovery stage, it simply passes all symbols -- send, echo and idle -- from the stripper to its output link. A node may *only* transmit a source packet immediately following a go-idle. During transmission of a packet, a node maintains the inclusive-OR of all go bits it receives from the stripper. If the ring buffer does not fill up at all during transmission, then the node postpends an idle symbol to its packet using the saved go bit it maintained during the transmission, and then continues to either transmit another source packet or output symbols from the stripper.

If the ring buffer *does* fill up at all during source queue transmission, then the node enters the recovery stage. All idles sent during the recovery stage, including the idle postpended to the original source transmission, are stop-idles. The node continues, however, to maintain the inclusive-OR of go bits it receives throughout the recovery stage. When the recovery stage ends (the last symbol is drained from the ring buffer), the saved go bit is released in the postpending idle just as it was for the postpending idle of a source transmission when the recovery state was not entered.

The reason that the flow control works is that when a node is trying to recover so that it may send another packet, it sends only stop idles. This inhibits the downstream neighbors from sending new packets and eventually provides enough slack for the node to drain its ring buffer and send a packet. In the absence of contention, all idles on the ring will be go-idles, and a newly arriving send packet can always be sent immediately.

3. ANALYTICAL MODEL

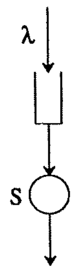
This section presents an analytical performance model of the SCI ring. The model is useful for a variety of reasons: it allows the quick exploration of a large state space, it's precisely defined and can be implemented by other researchers and engineers and it helps us gain insight into the processes being modeled. Results from the model, as well as from detailed simulations, will be presented in the section 4.

The model does not consider flow control, limited active buffers or target queue overflow. The effect of these factors can be gleaned from the simulation results in section 4, however. The model does consider, and effectively deals with, ring buffer fill-up during source queue transmissions, the transmission recovery process, the formation and effect of packet trains, non-homogeneous message arrival rates and non-uniform message destination probabilities, delays due to queueing in ring buffers and variance of source queue service times. We present only a summary of the model here, highlighting the important aspects of the approach. Detailed equations are given in Appendix A.

3.1. Model Overview

The message transmission latency in an SCI ring consists of two distinct parts: waiting for permission to transmit, and traversing the ring from source to target.

To solve for the first component, we view the source queue as an M/G/1 queue [Klei75], with service time equal to the sum of the packet transmission time (i.e., the packet length) plus the length of the recovery stage following the transmission. If we know the mean *and* variance of this service time, then we can calculate the time spent waiting for permission to send by using the standard M/G/1 wait time formula, as shown in Figure 2.



$$c = \frac{\sqrt{\mathrm{Var}(S)}}{S}$$

$$\rho = \lambda S$$

$$Q = \rho + \frac{\rho^2(1+c^2)}{2(1-\rho)}$$

$$L = \frac{V+S^2}{2S}$$

$$W = (Q-\rho)S + \rho L$$

Figure 2: The M/G/1 queue

To solve for the second component, we compute the mean backlog that a packet finds in the ring buffer of each node that it passes through. The latency to traverse the ring is then easily calculated from the routing probabilities, the fixed overheads (wire and parsing delay) and the mean delays due to buffer backlog.

3.2. Model Inputs

Inputs to the model are the ring size (N), message arrival rates (λ_i), routing probabilities (z_{ij}), packet lengths (l_{addr}, l_{data}, l_{echo}), packet type ratio (f_{data}, f_{addr}), transmission delay (T_{wire}) and parsing delay (T_{parse}). Note that each node has a distinct arrival rate and a distinct, possibly non-uniform message destination probability distribution. Several quantities (such as ring utilization, mean and variance of packet lengths passing through each node and various throughputs and ratios) are derived directly from the inputs.

3.3. Modeling Packet Trains and Computing Service Times

To calculate the mean and variance of the source queue service time and the mean delay due to buffer backlog seen by a packet passing through a node, we must compute various statistics about the packet trains that form in the system. To model these trains, we first make the assumption that both the length of a packet train (measured in packets) and the inter-packet-train spacing (measured in cycles), are geometrically distributed. [1] We then compute, for each link and node in the ring, *coupling probabilities* denoted by $C_{link,i}$ and $C_{pass,i}$, respectively. The coupling probability for a link (node) is the probability that a packet traversing the link (passing through the node) is separated from the packet in front of it only by the required single idle symbol. Due to the memoryless property of the geometric distribution, $1 - C_{pass}$ and $1 - C_{link}$ are the parameters of the corresponding geometric

[1] We make these assumptions in order to make the model tractable, and comment on the accuracy of these and other model assumptions in section 4.9.

distributions. The mean and variance of packet train lengths can be calculated from the packet lengths, the fraction of address and data packets, and the geometric distributions.

The following reasoning aids in computing node recovery times. First, we can *pretend* that when a packet of length L begins transmission from a source queue it is magically popped into the local ring buffer (recall that the ring buffer must be empty before transmission), and then consider the time taken to drain the ring buffer. Second, progress on draining the buffer is only made during cycles in which no packet symbols are arriving from the previous link. Third, the recovery period ends after L cycles in which progress is made.

To compute the recovery time, consider the first cycle of a transmission. We can calculate the probability that a passing packet arrives during this cycle and is thus blocked behind the send packet (we refer to this as *cut-in*). The probability that this occurs is a function of the node's coupling probability and ring utilization due to passing packets, and is higher than the probability of a packet arriving during the other cycles of transmission and recovery that we consider. If cut-in does occur, symbols are transmitted until the incoming packet train finishes, and then the first symbol is drained.

Now consider the $L-1$ remaining symbols to be drained. For each of these symbols, either it is drained right away, or an incoming packet train arrives, in which case symbols are again transmitted until the incoming packet train finishes, and then a symbol is drained. Since inter-packet-train spaces are geometrically distributed, and since we know the mean inter-packet-train space from the mean packet train length and the ring utilization due to passing packets, we can calculate the probability that a packet train arrives on the next cycle. The number of packet trains that arrive during the transmission/recovery period is thus given by a binomial distribution parameterized by the original send packet length and the probability of a packet train arrival. The mean service time (transmission plus recovery time) and mean number of packets that become coupled due to a packet injection can be calculated directly from this distribution, plus the impact of cut-in and the mix of send packet types.

The relation between service time and coupling probabilities is cyclic. After the mean service time is computed, new coupling probabilities are calculated by considering packets removed (send packets are converted to echo's, echo packets are fully removed) by the stripper and packets injected by the source queue. The model then re-computes the mean service times, mean number of packets coupling by an injection, and then new coupling probabilities, until the coupling probabilities converge.

After the above convergence, several metrics can be computed. The service time variance and mean buffer backlog seen by a passing packet are computed using the binomial distribution representing the number of packet trains arriving during the transmission/recovery period. The variance calculation involves some approximations, as will be discussed in section 4.9. The mean wait times are computed using the M/G/1 queueing formula, and the transmission times are calculated using routing probabilities and the mean buffer backlogs.

Experience implementing this model has shown that convergence is faster for smaller ring sizes. We required average change in coupling probabilities to be less than 10^{-5} for convergence. Approximately 10 iterations were needed for N=4, 30 for N=16 and 110 for N=64. Total time to solve the model for N=64 on a DECstation 3100 is about 1 second.

4. RESULTS

This section presents results derived from both the analytical model and a detailed, parameter-driven simulator of the SCI ring. The inputs to the model and to the simulator are identical. The ring is modeled as an open system (Poisson arrivals), with the arrival rates, message lengths, mix of message types, routing probabilities, ring size, wire transmission delay and message parsing delay specified as inputs. The simulator has the additional ability to consider flow control and limited buffer space (active buffers and sink queues). Since the ring is modeled as an open system, latency becomes infinite as saturation is reached. An actual system, of course, would have a limit to the number of queued or outstanding requests, and nodes would be stalled at some point rather than continuing to add requests.

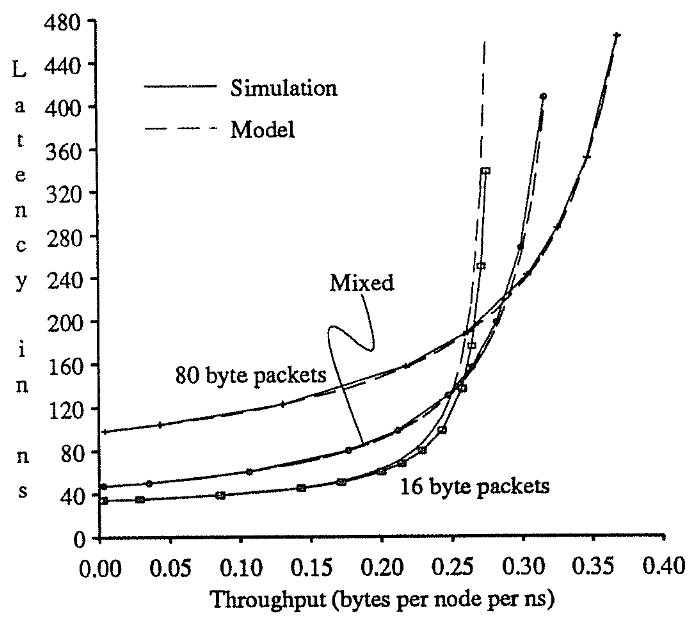
The unit of length in the model and simulator is one link width, and the unit of time time is one clock cycle. We assume (as per the standard) a 16-bit link with a 2 ns cycle time. Using these assumptions, we present output latencies in *ns* and throughputs in *bytes per ns*. Results can be generalized beyond these assumptions, therefore, only while keeping the input parameters fixed. The graphs can be applied to a ring with a 1 ns clock, for example, if the numbers on the latency axis are halved, and the numbers on the throughput axis are doubled. Throughputs are calculated using the entire message, including address, command and control information. Section 4.7 considers sustained data throughput using a read request/read response model.

There are many other parameters that we have fixed or limited in order to make the problem space tractable. Since the number of nodes in a ring is expected to be small, we analyzed ring sizes of primarily 4 and 16 nodes. Except where noted, we assumed that 80% of send packets were address/command only (16 bytes), and 20% included data blocks (80 bytes) (we refer to these as address packets and data packets for the remainder of the paper). We assume a fixed minimum delay of 4 cycles per node traversed by a message: one cycle to gate a symbol onto an output link, one cycle for the symbol to reach its downstream neighbor and two cycles to parse a symbol before routing it to the local node or to the next output link. Message latencies also include one cycle to originally queue the messages, and a delay equal to the message length to consume the message as it arrives at the target node. Any additional time is spent queued behind other messages in the source queue, waiting for permission to transmit or blocked in ring buffers in intermediate nodes. Unless otherwise noted, we assume unlimited active buffers at each node (we shall see that very few buffers are needed to approximates this).

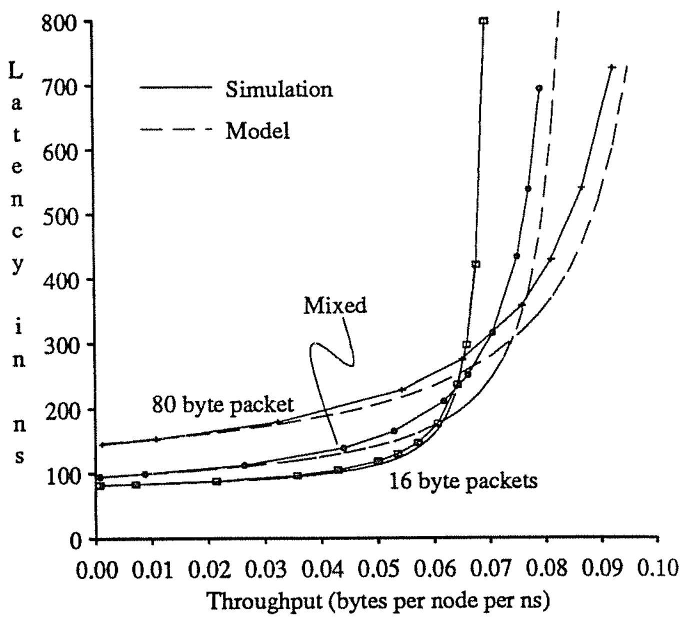
The simulator implements the protocol described in section 2 on a cycle by cycle basis, explicitly tracking each symbol on the ring. Simulations were run for 9.3 million cycles each, and 90% confidence intervals were computed using the method of batched means. Confidence intervals were generally under or about 1%, except near saturation, where they sometimes increased to a few percent.

4.1. Uniform traffic

Figure 3 shows the performance of 4- and 16-node SCI rings with uniform arrival rates and routing probabilities and no flow control. Each graph includes three sets of data, one with all address packets, one with all data packets and one with 20% data packets. Both simulation and model results are shown. The model is very accurate for the 4-node ring. For the 16-node ring, the model is accurate for the all-address-packet workload, but underestimates latency under moderate to heavy loading for the other workloads. Even for the worst case, however, the model provides a good estimate for the behavior of the ring. The reason for the error is identified and discussed in section 4.9.



(a) R = 4



(b) R = 16

Figure 3: Uniform traffic without flow control

Throughput is higher for the workload with larger packet sizes. There are two reasons for this. First, a smaller proportion of the ring bandwidth is used for the idle symbols that must separate each packet (we include only bytes within packets in the throughput measure). Second, the bandwidth consumed by echo packets becomes smaller relative to the bandwidth used by send packets. Throughput could also be increased by use of message locality. Unlike a shared bus, a ring requires less bandwidth if the messages are sent a shorter distance (message latency is similarly reduced). For the purposes of this paper, we assume equally distributed destinations.

Figure 4 illustrates the effect of flow control on uniform traffic for ring sizes of 4 and 16. Each graph includes two sets of data, one with all address packets, and one with all data packets. Results for the mixed address/data workload fall in between these. We can see that even with uniform traffic loading, flow control significantly reduces the maximum throughput. The reason for this is that there are times when a node cannot transmit a source packet, even though there are available slots in which to do so, because another node has stopped sending go bits in order to clear its ring buffer.

In Figure 5, the degradation due to flow control is shown for a variety of message and system sizes. Each curve plots the ratio of maximum bandwidth with flow control to maximum bandwidth without flow control as

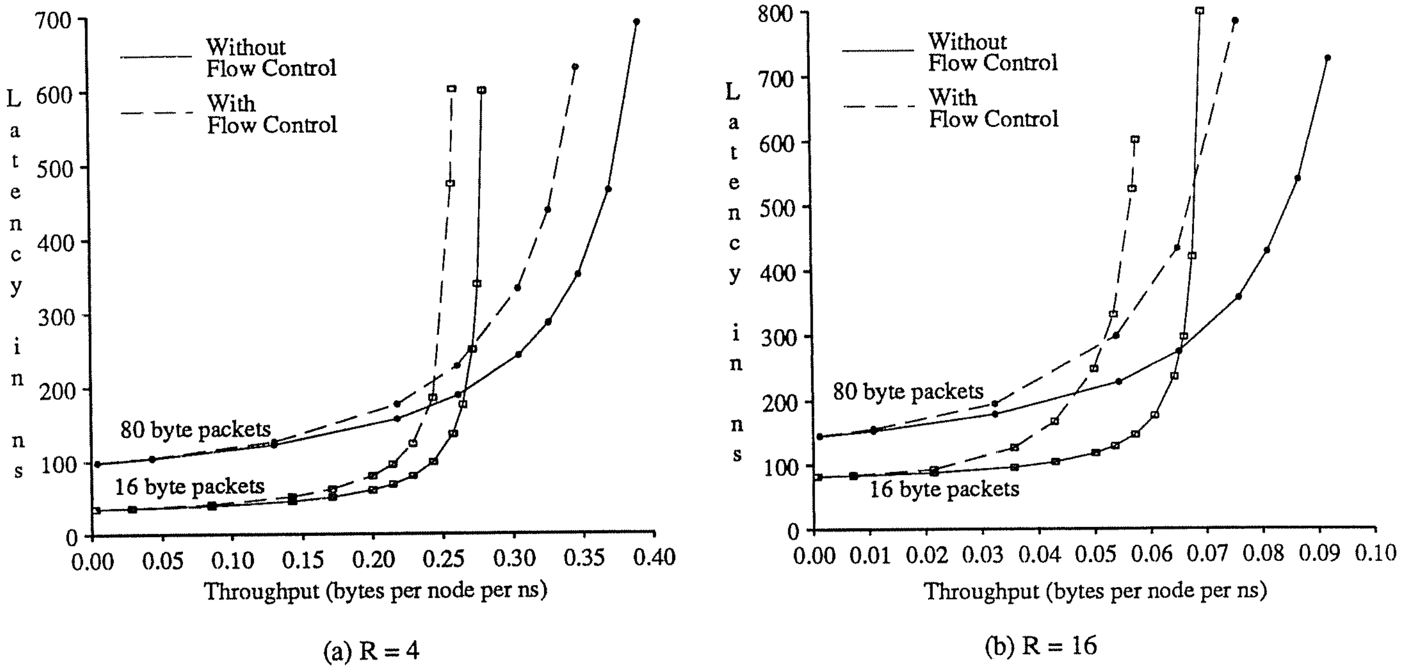


Figure 4: Effect of flow control on uniform traffic

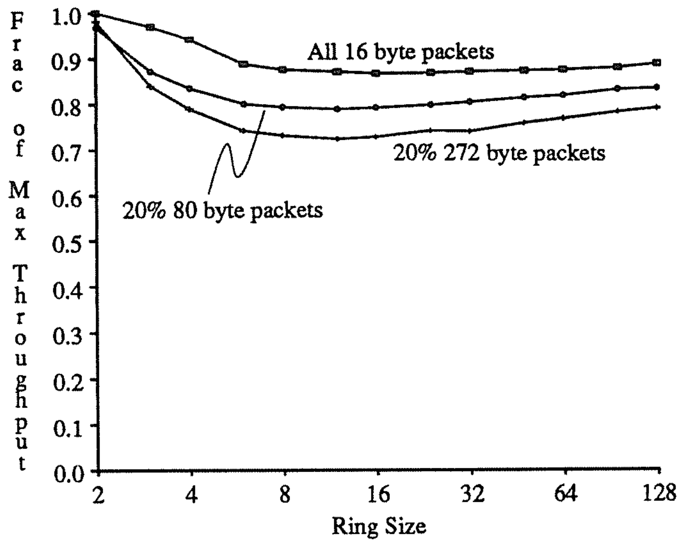


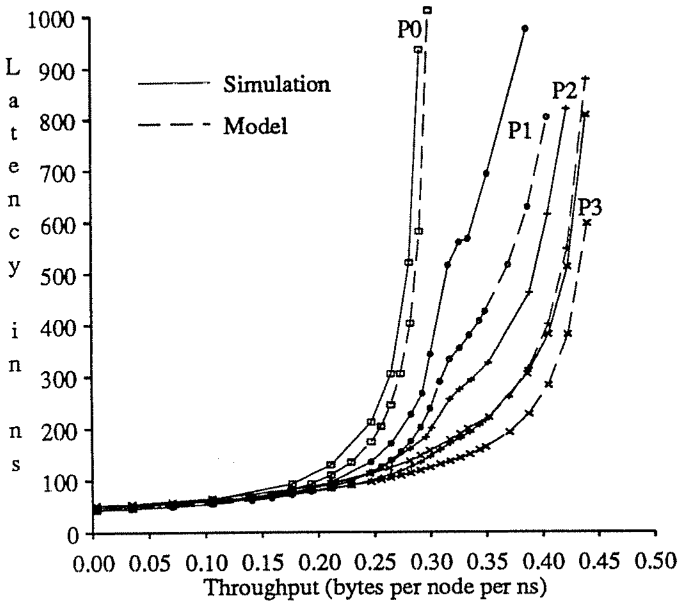
Figure 5: Throughput degradation from flow control

the ring size is varied. The top curve represents a workload with only address packets. The middle curve represents a workload where 20% of the packets are data packets with 64 bytes of data. The bottom curve represents a workload where 20% of the packets are data packets with 256 bytes of data. We can see that the degradation due to flow control increases with ring size up to about 16 nodes, and then slightly tapers off for larger rings. Flow control has very little effect for a ring size of 2. The degradation is worse when larger packets are being transmitted. The most common data block size is likely to be 64 bytes, as this is the size of the SCI cache block.

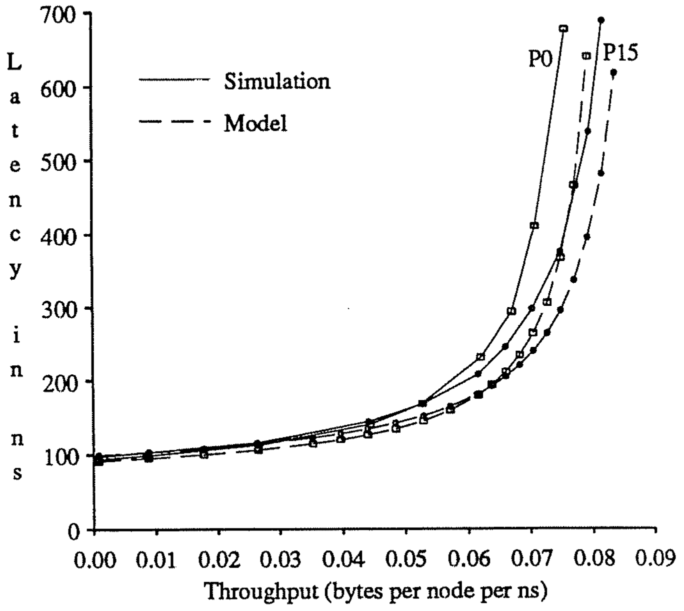
4.2. Node Starvation

This section examines the situation in which a node is inhibited from transmitting by reducing the number of breaks it sees in its pass-though traffic. Figure 6 presents the performance for 4- and 16-node rings where all nodes are routing uniformly, except that no messages are routed to node 0 (the starved node). Message latencies are plotted for individual nodes (labeled P0, P1, *etc.*). In Figure 6(a), we see that P0 saturates before the other nodes. As the throughput per node reaches about 3.2 bytes/ns, P0's arrivals can no longer be satisfied and its message latency goes to infinity (recall that this is simulated as an open system). As P1, P2 and P3 increase their throughput beyond this point, the realized throughput of P0 is actually driven back down to 0. This causes the unusual shape in the curves for P1 and P2. P1, P2 and P3 all reach the same saturation bandwidth.

The equations in the model assume that the system is not in saturation, so in order to model the behavior after P0 has saturated, the model detects saturated queues, and automatically throttles back the corresponding



(a) R = 4



(b) R = 16

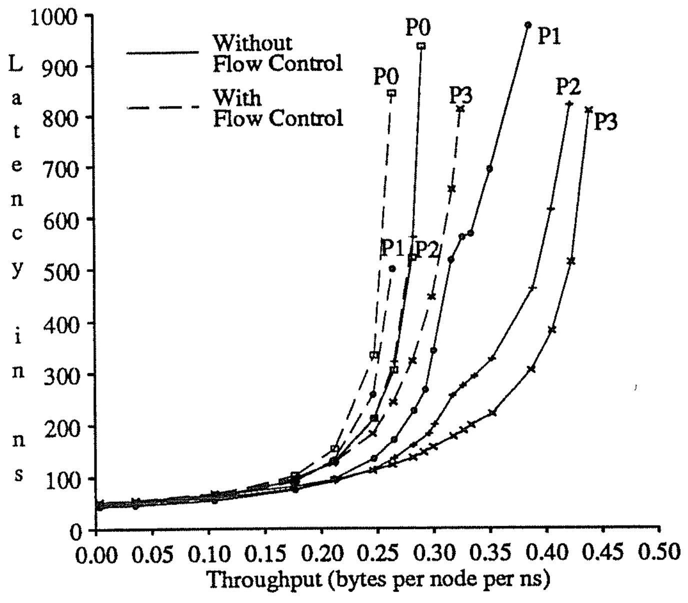
Figure 6: Node starvation without flow control

arrival rates to keep the source queue utilization at exactly one. The model qualitatively predicts correct behavior, including the throttling of P0's throughput and the corresponding inflection points in the P1 curve. However, the model underestimates the impact of the non-uniform traffic, and quantitative error is fairly large when the starved node is in saturation.

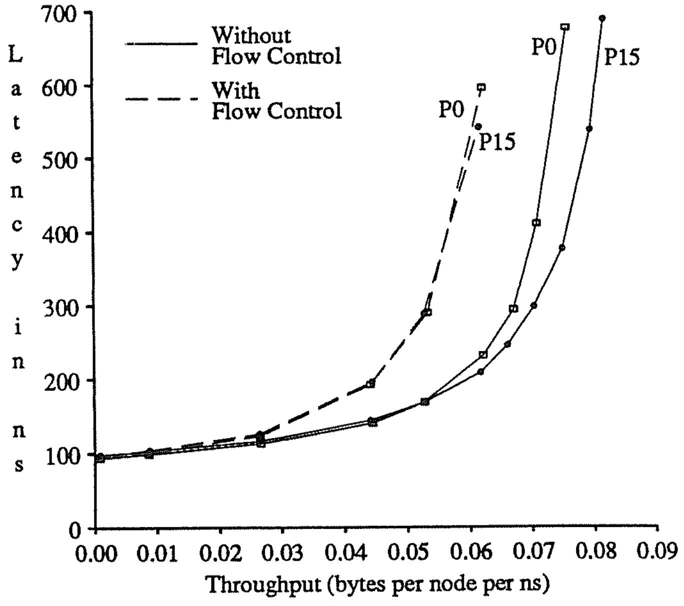
For a ring size of 16 (Figure 6(b)), the disparity between nodes is not as pronounced. The starved node reaches almost as high a bandwidth as the other nodes before it saturates. This is because the non-uniform routing causes smaller differences in link utilizations for the larger ring. The model correctly predicts the spread in performance between the starved node (P0) and the least affected node (P15). The absolute error, however, is about the same as for the uniform traffic case (the "mixed" traffic curves in Figure 3(b)).

Figure 7 demonstrates the effect of flow control on node starvation. Parts (a) and (b) show the message latency for each node as the traffic is varied. In parts (c) and (d), the ring is in saturation (all nodes are trying to send as often as possible), and the realized throughput for each node is shown.

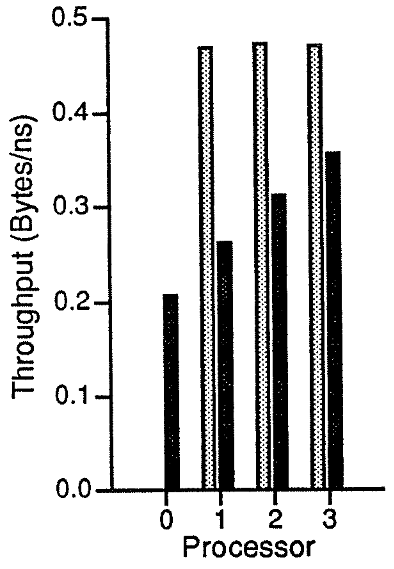
In Figure 7(a), we see that the addition of flow control reduces the disparity between the performance of the four nodes, but at an overall reduction in throughput. The throughput of P0 is not driven back down by the other nodes, as it is without flow control. Note, however, that the performance is not fully equalized; P0 achieves a smaller maximum throughput than P1, P1 achieves a smaller maximum throughput than P2, *etc.*



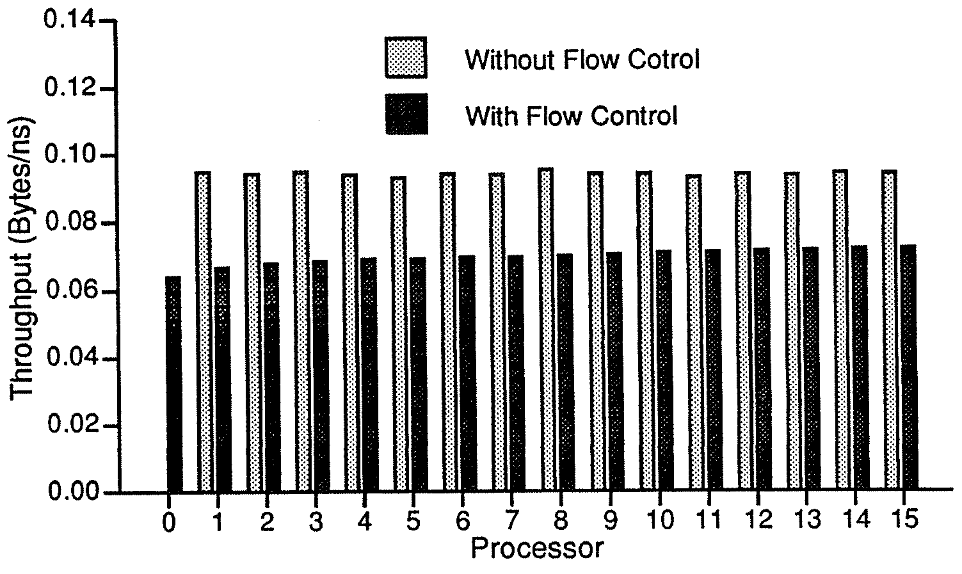
(a) R = 4



(b) R = 16



(c) R = 4



(d) R = 16

Figure 7: Effect of flow control on node starvation

Figure 7(c) shows the saturation bandwidths for the 4-node ring with P0 still being starved. Without flow control, P1, P2 and P3 all achieve roughly the same throughput, but P0 is *completely* starved. Because the ring is fully utilized and it is not receiving any messages, it has no opportunities in which to transmit a message (*i.e*: it enters an infinite recovery stage). The flow control mechanism successfully deals with this problem. With flow control, the total ring throughput is reduced slightly, and the throughput of the non-starved nodes is reduced significantly, but the starved node is no longer kept from transmitting. The flow control mechanism does not achieve full equal partitioning of bandwidth, however. The throughput of a node is limited by how quickly it can empty its ring buffer after a transmission, and the flow control protocol guarantees that even a starved node will make forward progress in the recovery stage by throttling downstream transmissions and thus creating gaps in its incoming packet stream. However, a node whose input link is less utilized overall (due to non-uniform traffic) will still be able to recovery more quickly on average.

Figure 7(b) shows the effect of flow control for a ring size of 16 with P0 being starved. The addition of flow control almost completely equalizes the performance of the various nodes (again at an overall reduction in ring capacity). Figure 7(d) shows the saturation bandwidths for the 16-node ring. Although the impact on P0 was small under light to medium traffic, P0 is completely starved when the ring is fully loaded. Flow control reduces the realized throughput of the non-starved nodes and allows the starved node to transmit. The bandwidth is much more equally divided than it was for the 4-node ring. This is because the differences in link utilizations caused by the non-uniform traffic are smaller for the larger ring.

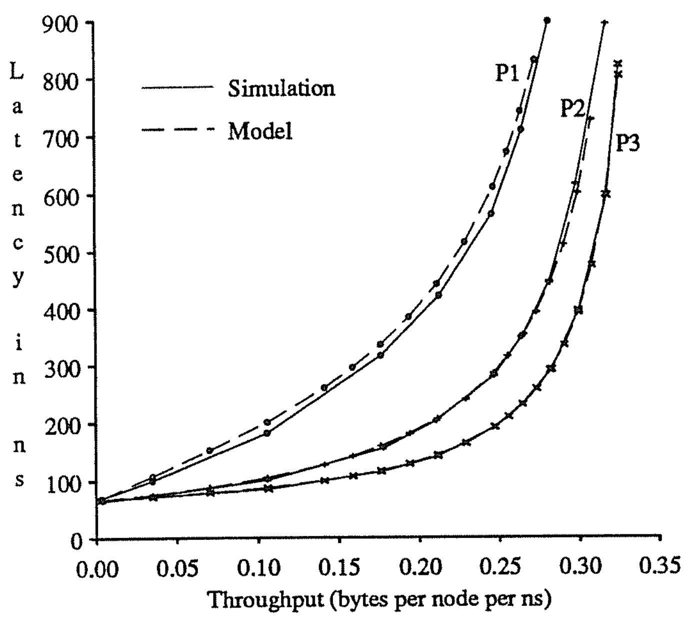
4.3. Hot Sender

In this section, we examine the ring's behavior in the presence of a "hot sender" (a node that attempts to use as much ring bandwidth as possible). Figure 8 presents the performance for 4- and 16-node rings where message destinations are uniformly distributed, but node 0 always tries to transmit a packet. P1, the first downstream node from the hot sender, is severely affected by the extra traffic. The hot node degrades the performance of all other nodes on the ring, affecting the closest nodes more heavily.

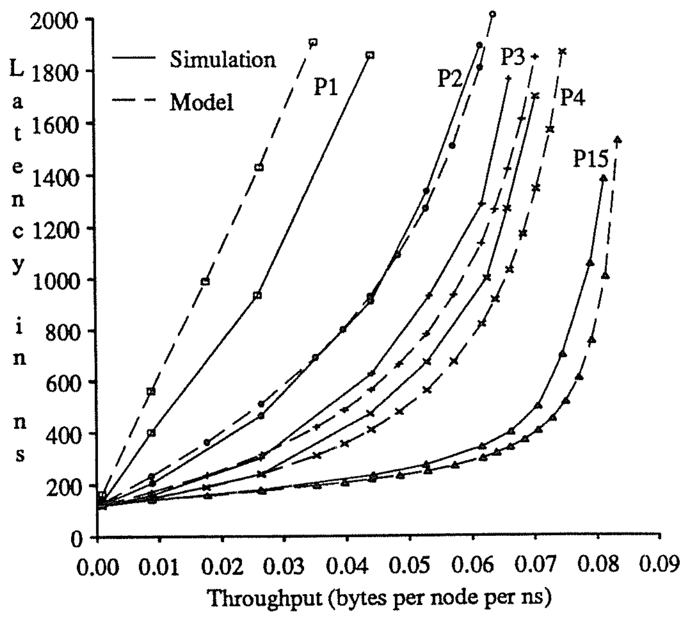
The model is very accurate for a ring size of four (Figure 8(a)). For a ring size of 16 (Figure 8(b)), the model is qualitatively accurate, but slightly underestimates latency for most of the nodes, and significantly overestimates the latency for the immediate downstream neighbor of the hot node (the reason that the model is pessimistic for P1 is actually that it is overly optimistic for P0).

Figure 9 demonstrates the effect of flow control on a hot sender. Parts (a) and (b) show the message latencies for each node as a function of throughput. The addition of flow control equalizes the effect of the hot node on the message latency for the other nodes on the ring. Performance is improved for some nodes and degraded for others, but the hot node's downstream neighbor is no longer severely penalized.

This phenomenon can be clearly seen in parts (c) and (d), which show vertical slices of the throughput-latency curves under moderate throughput from the "cold" nodes (0.177 bytes/ns in Figure 9(c), 0.044 bytes/ns in



(a) R = 4



(b) R = 16

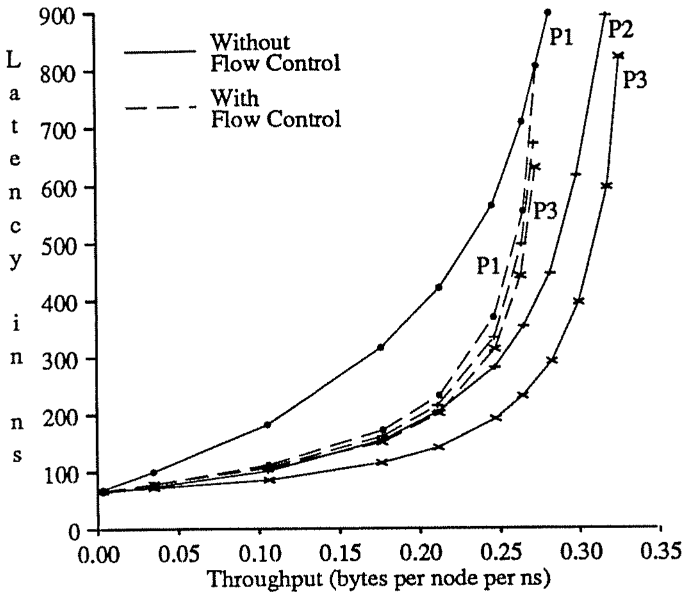
Figure 8: Hot sender without flow control

Figure 9(d)). Without flow control, the mean message latencies experienced by the cold nodes vary significantly, with the closest downstream nodes being affected the most. With flow control, the hot node affects all other nodes approximately equally. The nearest downstream node, in particular, is no longer subjected to extremely large latencies. The improved ring fairness is achieved at the expense of the hot sender's throughput. Without flow control, it realizes a rate of 0.643 bytes/ns on the 4-node ring. With flow control, it realizes only 0.517 bytes/ns. For the 16-node ring, the hot sender's throughput is reduced from 0.511 bytes/ns to 0.264 bytes/ns. For certain applications, most notably real-time systems, it may be desirable to allow one node or a set of nodes to consume more than their share of ring bandwidth. SCI provides a priority mechanism to satisfy this requirement.

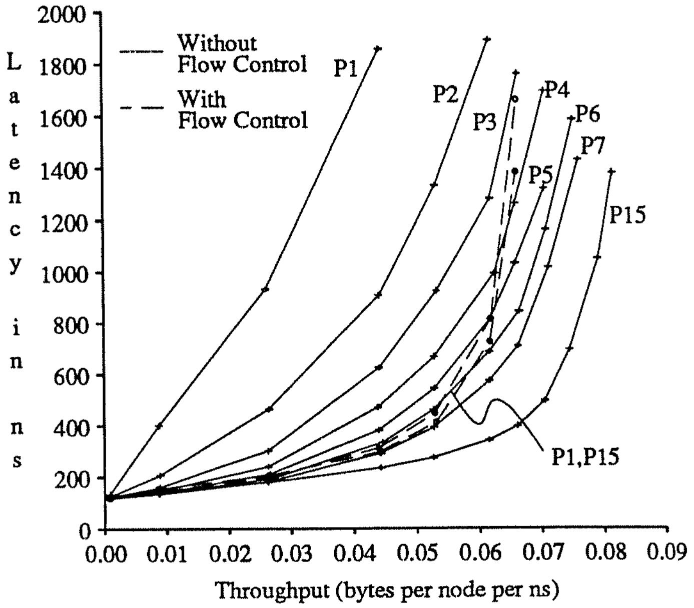
In addition to hot senders and node starvation, we have examined producer-consumer and other non-uniform workloads. Though not presented here, the results are similar. The flow control mechanism reduces the effects of greedy nodes on the rest of the ring, and provides all nodes with a reasonable approximation to their share of the bandwidth, regardless of the non-uniformities present in the communication pattern.

4.4. Varying the Number of Active Buffers

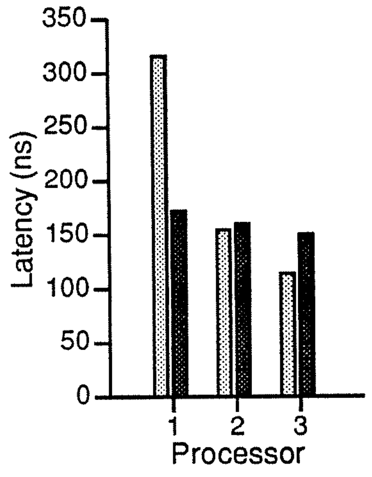
Figure 10 illustrates the effect of varying the number of active buffers in the ring interfaces (see Figure 1). With no active buffers, only a single send packet can be outstanding from a node at any time. Each additional



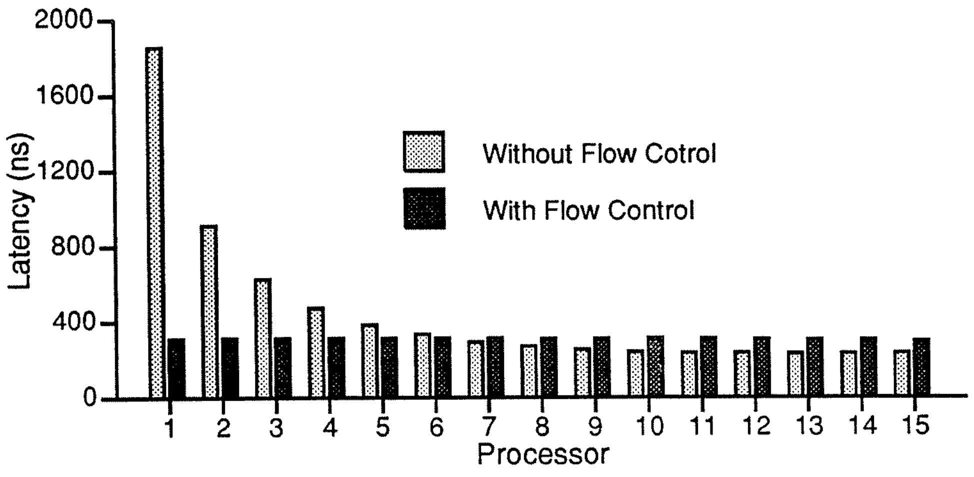
(a) R = 4



(b) R = 16

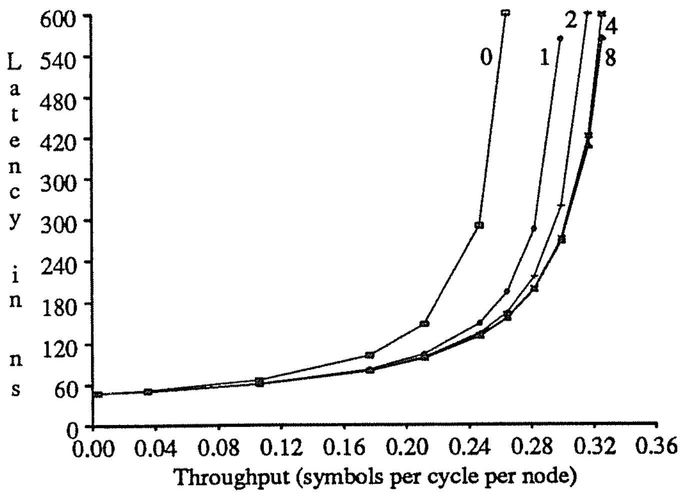


(c) R = 4

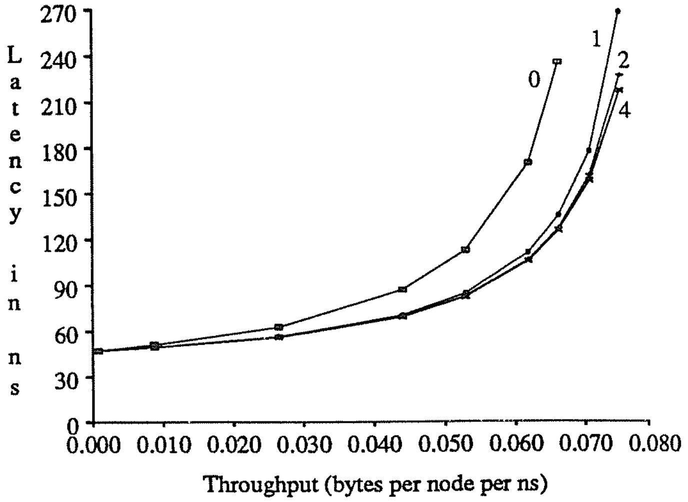


(d) R = 16

Figure 9: Effect of flow control on hot sender



(a) R = 4



(b) R = 16

Figure 10: Varying the number of active buffers

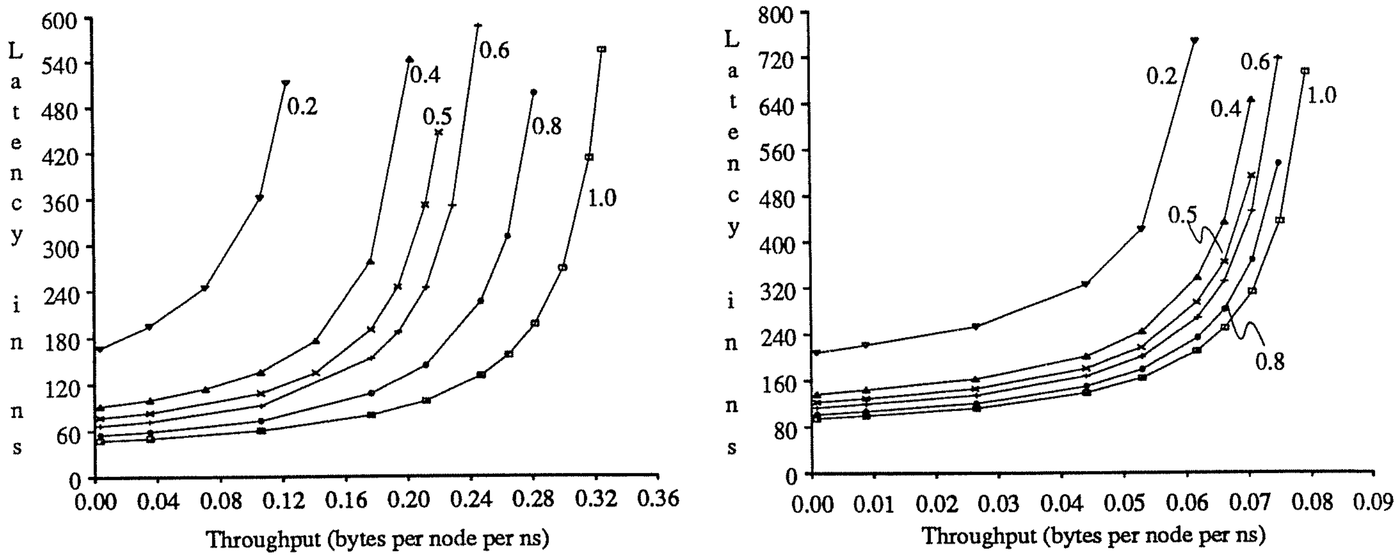
active buffer allows another packet to be transmitted before the first packet is acknowledged. Both the 4- and 16-node rings benefit from a single active buffer, and the 4-node ring benefits from a second active buffer. There is very little incremental benefit from additional buffers.

It is somewhat counter-intuitive that the smaller ring benefits more from additional active buffers, because the time between transmitting a packet and receiving the acknowledgement is greater for the larger ring. The smaller ring, however, has a greater per-node message arrival rate, given the same ring utilization. This increases the need for multiple outstanding requests at one node. Another way to view this is that the overall message arrival rates for the rings are the same, and thus the smaller ring needs to be able to buffer more active messages per node. Certain types of systems may require a greater number of active buffers, perhaps only at specific nodes on the ring.

4.5. Sink Queue Overflow

In this section, the effect of sink queue overflow is briefly examined. We assume that a node has a sink queue large enough to hold the largest incoming data packet, and that this queue can accept a new symbol on each network cycle. If the queue can be drained by the target node at one symbol per clock, then this queue will never overflow. For various reasons (either congestion or a difference in clock speeds), this queue might not be drained as fast as it can be filled, leading to the possibility of queue overflow.

Figure 11 illustrates the effect that this has on overall ring performance. The rate at which sink queues are drained, relative to the network speed, is varied from 0.2 to 1.0. A value of 0.8, for instance, means that on each



(a) R = 4 (b) R = 16

Figure 11: Varying the symbol consumption rate

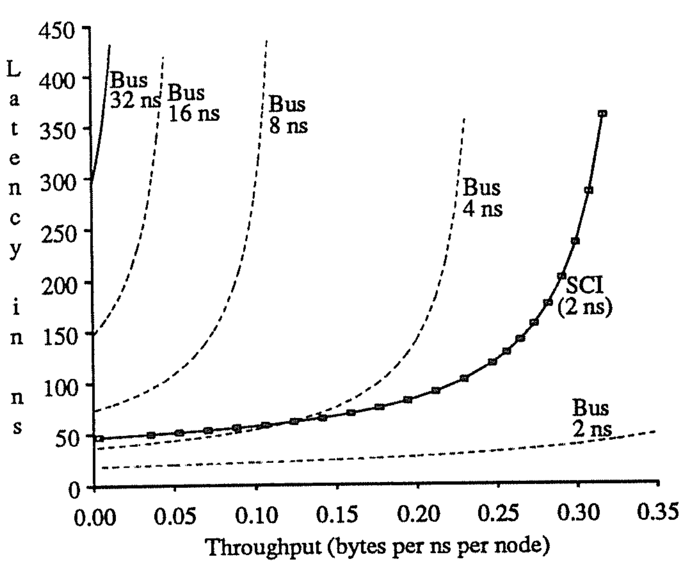
network cycle, a symbol is consumed from a non-empty sink queue with probability 0.8.

Reducing the sink queue consumption rate causes queue overflow to periodically occur, thus reducing the maximum ring throughput and increasing the mean message latency. Packets that are not accepted at their target sink queues are *nacked*, and must be retransmitted by their sources. The effect of a reduced sink queue servicing rate is greater for smaller rings. This is simply because the packet arrival rate per node is greater when there are fewer nodes on the ring. Increasing the size of the sink queues would alleviate the overflow problem, given that the queue service rate was still larger than the average rate at which messages arrived at a node.

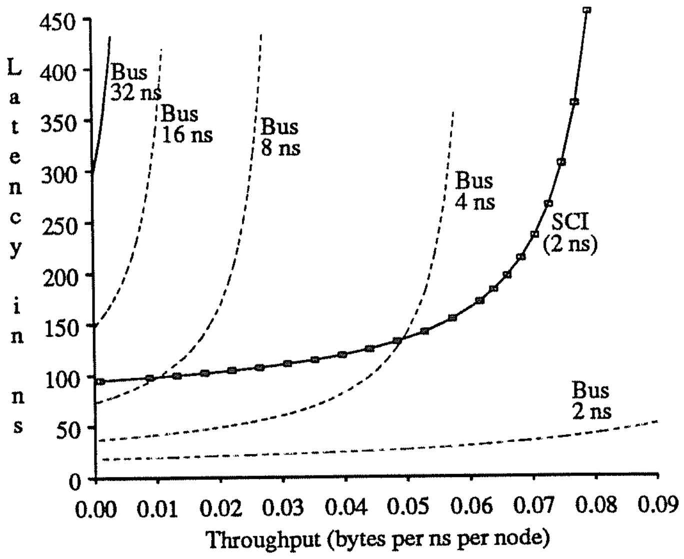
4.6. Comparison to a Conventional Bus

This section compares the SCI ring to a conventional, synchronous bus. The prime advantage of SCI, at the logical-layer level, is its use of fast, point-to-point links. The unidirectional nature of the communication allows the cycle time to be limited only by the speed of the technology. Standard ECL circuitry available in 1991 allows a 2 ns clock. To compare this against a conventional bus, we developed a simple M/G/1 bus model. The model assumes no overhead for arbitration, and single-cycle synchronous transmission in 32-bit chunks. The pin-out for an SCI interface is also 32 bits (16-bit input link plus 16-bit output link).

Figure 12 compares the throughput-latency characteristics of an SCI ring to a bus as the bus cycle time is varied. We assume a workload of 80% address packets (16 bytes) and 20% data packets (80 bytes). Ring/bus



(a) R = 4



(b) R = 16

Figure 12: SCI ring vs conventional bus

sizes of 4 and 16 nodes are used. If a synchronous bus had the same cycle time as the SCI ring, it would clearly provide better performance. This is due not only to the bus' greater width, but to the single cycle transmission latency to all nodes on the bus. With a bus cycle time of 4ns, latency is still lower when lightly loaded, but the maximum throughput is also lower. This is due to the bus' lack of concurrency.

As the bus cycle time is increased, the latency goes up significantly, and the maximum throughput drops off significantly. Realistic bus cycle times range from 20 to 100 ns. A typical high performance shared bus has a 30 ns cycle time. The Stardent Titan graphics supercomputer uses a 31.25 ns bus [Siew91], for example, and the Silicon Graphics Power Series computers use a 30 ns bus [Grap89]. The ELXSI System 6400 used expensive twisted-pair ECL with differential signaling for their shared backplane, achieving a cycle time of 25 ns [Olso83]. The SCI ring provides far greater bandwidth and lower latency than a bus of comparable width running at 20 ns or slower.

As the number of nodes on a ring increases, the average message latency will increase. As the number of nodes on a bus increases, the average message latency will also increase, due to greater contention for the bus and because the cycle time of the bus will have to be increased to accommodate the greater capacitive loading and longer physical distances. Because of the increased cycle time, the total bandwidth of the bus will decrease as well. Point-to-point links avoid this problem by divorcing throughput from transmission delay.

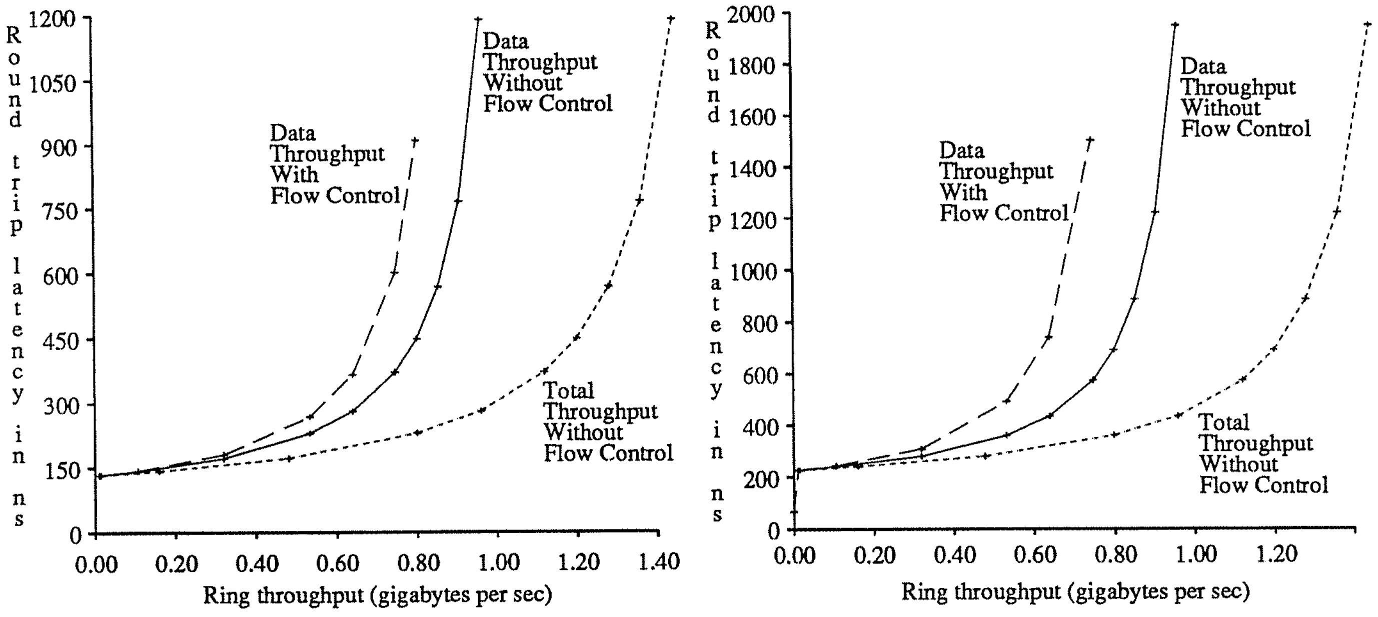
4.7. Sustained Data Throughput using a Request/Response Model

This section considers total sustained data transfer rates on a ring. We assume that the ring traffic consists solely of read request packets and their associated read response packets. Latencies represent an address packet transmission from a processor to a memory, followed by a data packet transmission from the memory to the processor (memory lookup time is not included). We use a data block size of 64 bytes, and the throughput includes only the data bytes.

The results are shown in Figure 13. Since an address packet is 16 bytes, and a data packet includes a 16 byte header along with the 64 bytes of data, exactly two thirds of the send packet symbols contain data. The actual data throughput is thus two thirds of the total throughput. Given the even split between address and data packets, the mean latency for the round trip is simply twice the mean latency for a single packet transmission. The throughput shown in Figure 13 is the *total* ring throughput, measured in gigabytes per second.

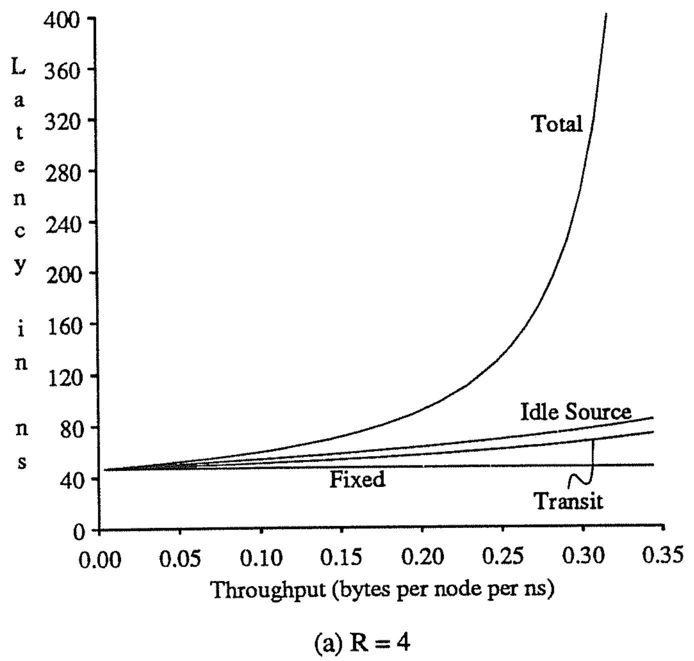
4.8. Breakdown of Message Latency

In this section, we break the mean message latency into several components. Figure 14 plots results from the analytical model for ring sizes of 4 and 16. The message traffic is uniform, with 20% of the packets



(a) R = 4 (b) R = 16

Figure 13: Sustained data throughput



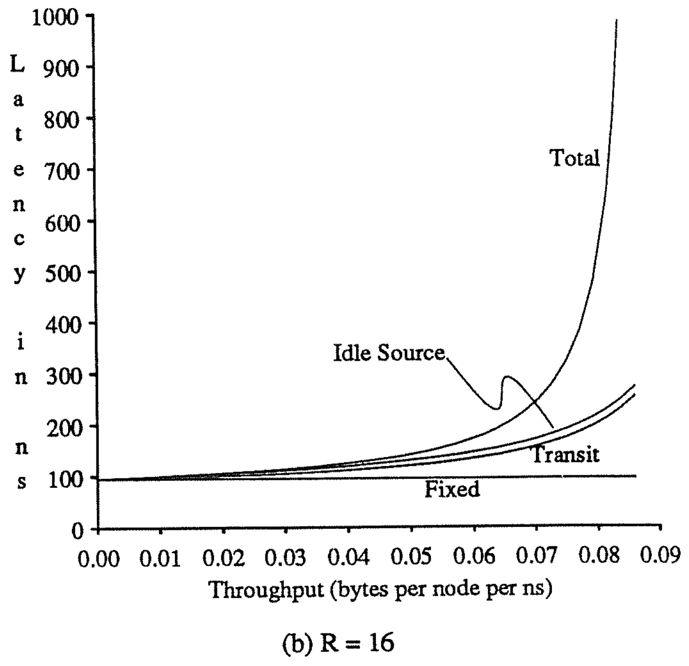


Figure 14: Breakdown of Message Latency

containing 64-byte data blocks. The latency is broken into 4 components. The *Fixed* curve represents latency due to wire transmission delay and fixed switching overheads. The *Transit* curve represents the time from when a source queue begins transmitting until the message is consumed at the destination. The difference between these two curves is due to delays passing through the ring buffers. The *Idle Source* curve represents the latency seen by a message arriving at an idle source queue (there are no packets in front of it and the node is not in the recovery stage). The difference between the *Transit* curve and this curve represents the time a source packet may have to wait while a packet finishes passing through the node. The *Total* curve represents total, end-to-end latency. The gap between the *Transit* curve and this curve represents the total time a packet is queued at a source queue before receiving permission to transmit.

Much of the latency under heavy loads is due to waiting in the source queues. In a closed system (where there is a limit on the number of queued messages), the delay due to source queueing would level off at some point. Delay due to buffer backlog becomes more significant relative to source queueing delay as the ring size is increased from 4 to 16. For very large rings, transmission delay becomes dominant except when the ring is very close to saturation.

4.9. Discussion of Model

Below we identify several possible sources of error in our analytical model, and discuss the likely significance of each.

First, we assume geometrically distributed inter-packet-train spacing, whereas simulations show that certain lengths of spaces are much more common. For example, the space created by replacing an address packet with an echo packet occurs with high frequency. However, simulation estimates of the coefficient of variation of the inter-packet-train spacing, are very close to 1. Thus, we do not anticipate that this assumption causes significant error.

Second, in computing the variance of the recovery period, we do not know the correlation between the cut-in portion, and the remaining portion. We assume a strong correlation, and treat the recovery time due to cut-ins as a scalar multiplier to the remaining recovery time. This may slightly overestimate the service time variance, but we do not believe this causes significant error.

Third, we sum the variance of the wait time for packets arriving at an empty source queue and the source queue service time, yet these random variables are most likely not independent. Again, the effect of this should be small.

Finally, we assume that the source queue utilization and the pass-through ring utilization are independent. That is, the model assumes we see the same rate of passing packets regardless of whether the source queue is in use. *This is the primary source of error in the model.* We have found using the simulator that the pass-through traffic tends to be lower than average when the source queue is idle, and higher than average when the source queue is active (during the transmission and recovery stage). This causes the model to underestimate the length of the recovery stage, thus underestimating the overall message latency. The error increases as the mean length of the recovery period increases, which causes the error to grow for larger rings and packet sizes.

Two worthwhile directions for future research are to reduce the error in the current model and to extend the model to account for flow control.

5. CONCLUSIONS

This paper presented a performance study of the SCI ring, including a description of the logical-level protocol, an efficient, analytical performance model, and extensive simulation results. The performance of the SCI ring was analyzed under uniform and non-uniform workloads and with and without the flow control mechanism. The SCI ring was also compared to a conventional shared bus.

The analytical model developed for the SCI ring did not consider the flow control mechanism, but was found to be accurate for both uniform and non-uniform communication patterns. Where quantitative error was greater, qualitative behavior was still predicted correctly. The primary source of error in the model was identified, and will be the topic of further research, along with extensions to model flow control mechanisms.

The SCI flow control mechanism effectively prevents node starvation, providing all nodes with their approximate fair share of the ring bandwidth. Non-uniform routing still affects the realized node throughputs to some extent, however, with the effect being greater for smaller ring sizes. The flow control mechanism also equalizes the negative impact that a hot node has on the rest of the ring. Without flow control, the downstream neighbors of a hot node see substantially increased message latencies.

The fairness provided by the flow control mechanism comes at the cost of overall ring throughput. Maximum throughput is reduced by up to 30%. The impact is greatest for ring sizes of 8 to 32, and is negligible for a ring size of 2. The degradation is also dependent upon packet sizes. When 20% of the packets contain 64-byte data blocks, maximum throughput is lowered by about 20% by the flow control mechanism. With address packets only, maximum throughput is lowered by about 10%. We are investigating possible modifications to the flow control mechanism that would gracefully increase ring throughput in return for reduced fairness.

Buffering issues were also briefly analyzed. The performance of an SCI ring can be appreciably improved by the addition of a single active buffer at each ring interface (allowing two outstanding messages from each node). Additional active buffers have very limited added benefit, although they help smaller rings more than larger rings. Limited sink queue space was also investigated. This is not an issue when messages are consumed by their targets as fast as they are placed into the target queues. When the target consumption rates drop, however, ring throughput falls significantly. The effect is greater for smaller rings, and could be mitigated by additional sink queue space.

In comparing the SCI ring to a conventional bus, the clock speed was considered along with the number of cycles needed to transmit a message. Although the number of cycles is larger for the ring, the faster clock speed gives a significant advantage. A 32-bit bus would have to have a 4 ns clock to be competitive with a 16-bit wide SCI ring with a 2 ns clock (and even then it would have a lower saturation bandwidth). While a 2 ns clock for SCI is realizable in 1991 with standard ECL circuitry, typical high performance multiprocessor buses have cycle times around 30 ns.

With a 16-bit width and 2 ns cycle time, the SCI ring provides a total peak throughput of over 1 gigabyte per second. Communication locality and larger packet sizes increase this number, while flow control overhead decreases it. Given a read request/read response model and 64-byte data blocks, a total data transfer rate of approximately 600-800 megabytes per second can be sustained over a single ring. The flow control mechanism partitions this bandwidth fairly among all participating nodes. The SCI standard leaves room for future improvements by both increasing the link width and decreasing the cycle time.

Acknowledgements

The authors gratefully acknowledge the assistance of Dave James in clarifying several important concepts in the SCI standard. We would also like to thank Ross Johnson who helped with our early understanding of the protocol and Alain Kagi who pointed out a misunderstanding in the protocol. In addition we have received much help from other SCI colleagues too numerous to mention, but especially from Dave Gustavson, Stein Gjessing, Ernst Kristiansen, and Hans Wiggers.

References

Grap89.
Graphics, Silicon Inc., "Power Series," Silicon Graphics Technical Report (1989).

IEEE91.
IEEE,, "Scalable Coherent Interface, Logical Specification," IEEE Standard Specification P1596: Part II (October 1991).

Jame90.
James, D. V., A. T. Laundrie, G. S. Sohi, and S. Gjessing, "SCI (Scalable Coherent Interface) Cache Coherence," *IEEE Computer*, (July 1990).

Klei75.
Kleinrock, L., *Queueing Systems, Volume I*, John Wiley and Sons, New York (1975).

Olso83.
Olson, R. A., B. Kumar, and L. E. Shar, "Messages and Multiprocessing in the ELXSI System 6400," *COMPCON83*, pp. 21-24 (February 1983).

Siew91.
Siewiorek, D. P. and P. J. Koopman, Jr., *The Architecture of Supercomputers: Titan, A Case Study*, Academic Press, San Diego, CA (1991).

Appendix A -- Performance Model Equations

List of all variables and their meanings

N	Number of nodes per ring
$z_{i,j}$	Fraction of node *i*'s source packets routed to node *j*
λ_i	Source queue arrival rate at node *i* (packets per cycle)
f_{data}	Fraction of send packets that are data packets
f_{addr}	Fraction of send packets that are address packets
l_{data}	Length in symbols of a data packet (including postpended idle)
l_{addr}	Length in symbols of an address packet (including postpended idle)
l_{echo}	Length in symbols of an echo packet (including postpended idle)
T_{wire}	Number of cycles to traverse wire between nodes
T_{parse}	Number of cycles to parse a packet
λ_{ring}	Total packet arrival rate
$r_{data,i}$	Rate of data packets passing through node *i*
$r_{addr,i}$	Rate of address packets passing through node *i*
$r_{echo,i}$	Rate of echo packets passing through node *i*
$r_{pass,i}$	Total rate of packets passing through node *i*
$r_{rcv,i}$	Rate of packets routed to node *i*
$n_{train,i}$	Mean number of packets in a passing packet train at node *i*
$n_{clump,i}$	Mean number of packets caused to clump by injected source packet at node *i*
$n_{pass,i}$	Mean number of passed packets per injected packet at node *i*
l_{send}	Mean length of a send packet
$l_{pkt,i}$	Mean length of a passing packet at node *i*
$l_{train,i}$	Mean length of passing packet train at node *i*
w_i	Mean wait until break in passing packets at node *i*
$P_{cut\_train,i}$	Probability an injected packet at node *i* "cuts off" a packet train
$P_{cut\_pkt,i}$	Probability an injected packet at node *i* "cuts off" a single packet
$P_{pkt,i}$	Probability during idle stream passing through node *i* that next symbol is a packet
ρ_i	Utilization of source queue at node *i* (including possible wait for break, transmission & recovery)
$U_{pass,i}$	Utilization of node *i*'s bandwidth by passing packets
$S_{cut\_in,i}$	Mean service time component due to "cut-ins" at node *i*
$S_{data,drain,i}$	Mean service time component due to rest of recovery time for data packet injection at node *i*
$S_{addr,drain,i}$	Mean service time component due to rest of recovery time for address packet injection at node *i*
$S_{addr,i}$	Mean service time for an address packet injection at node *i*
$S_{data,i}$	Mean service time for a data packet injection at node *i*
S_i	Mean source queue service time at node *i* (possible wait for break, transmission & recovery)
$C_{pass,i}$	Probability that passing packet at node *i* immediately follows its predecessor
$C_{link,i}$	Probability that packet on node *i*'s output link immediately follows its predecessor
$F_{in,i}$	Mean number of "following" packets entering stripper *i* per stripped packet
$P_{unclump}$	Probability a stripped packet at node *i* causes next packet to become unclumped
F_{out}	Mean number of "following" packets leaving stripper *i* per stripped packet
$V_{pkt,i}$	Variance of passing packet length at node *i*
$V_{train,i}$	Variance of passing packet train length at node *i*
$V_{addr,i}$	Variance of service time for injected address packet at node *i*
$V_{data,i}$	Variance of service time for injected data packet at node *i*
$V_{wait,i}$	Variance of time spent waiting for break to transmit because queue was empty on arrival at node *i*
V_i	Overall variance of service time for an injected packet at node *i*
$B_{addr,i}$	Mean backlog at node *i* due to address packet injection
$B_{data,i}$	Mean backlog at node *i* due to data packet injection
B_i	Overall mean backlog at node *i* seen by a passing packet

c_i	Coefficient of variation of S_i
Q_i	Mean source queue length at node *i*
W_i	Mean wait time in source queue at node *i*
L_i	Mean residual life of source queue service time at node *i*
T_i	Mean transit time for node *i* (once transmission begins)
R_i	Mean response time of a message transmission from node *i*
X_i	Mean throughput at node *i* (symbols per clock)

Preliminary calculations

$$l_{send} = f_{data}\, l_{data} + f_{addr}\, l_{addr}$$

$$X_i = \lambda_i\,(l_{send} - 1)$$

$$\lambda_{ring} = \sum_{i=0}^{N-1} \lambda_i$$

$$r_{echo,i} = \sum_{j \neq i} \lambda_j \sum_{\substack{k=j+1 \\ (mod\,N)}}^{i} z_{jk}$$

$$r_{data,i} = f_{data} \sum_{j \neq i} \lambda_j \sum_{\substack{k=i+1 \\ (mod\,N)}}^{j-1} z_{jk}$$

$$r_{addr,i} = f_{addr} \sum_{j \neq i} \lambda_j \sum_{\substack{k=i+1 \\ (mod\,N)}}^{j-1} z_{jk}$$

$$r_{pass,i} = r_{echo,i} + r_{data,i} + r_{addr,i} = \sum_{j \neq i} \lambda_j$$

$$r_{rcv,i} = \sum_{j \neq i} \lambda_j z_{ji}$$

$$n_{pass,i} = \frac{r_{pass,i}}{\lambda_i}$$

$$U_{pass,i} = r_{data,i} l_{data} + r_{addr,i} l_{addr} + r_{echo,i} l_{echo}$$

$$l_{pkt,i} = \frac{U_{pass,i}}{r_{pass,i}}$$

$$w_i = \left[\frac{r_{data,i} l_{data}^2 + r_{addr,i} l_{addr}^2 + r_{echo,i} l_{echo}^2}{2} \right]$$

$$V_{pkt,i} = \left[\frac{r_{data,i}(l_{data} - l_{pkt,i})^2 + r_{addr,i}(l_{addr} - l_{pkt,i})^2 + r_{echo,i}(l_{echo} - l_{pkt,i})^2}{r_{pass,i}} \right]$$

Calculations inside iteration

$$n_{train,i} = \frac{1}{1 - C_{pass,i}}$$

$$l_{train,i} = l_{pkt,i} n_{train,i}$$

$$P_{pkt,i} = \frac{U_{pass,i}}{(1-U_{pass,i})l_{train,i}}$$

$$P_{cut\_train,i} = \rho_i P_{pkt,i} + (1-\rho_i)U_{pass,i}C_{pass,i}$$

$$P_{cut\_pkt,i} = \frac{(1-\rho_i)U_{pass,i}(1-C_{pass,i})}{l_{pkt,i}}$$

$$S_{cut\_in,i} = P_{cut\_train,i}l_{train,i} + P_{cut\_pkt,i}l_{pkt,i}$$

$$type = addr \text{ or } data$$

$$S_{type,drain,i} = (l_{type}-1)P_{pkt,i}l_{train,i} = \frac{(l_{type}-1)U_{pass,i}}{(1-U_{pass,i})}$$

$$S_{type,i} = S_{cut\_in,i} + S_{type,drain,i} + l_{type}$$

$$S_i = f_{data}S_{data,i} + f_{addr}S_{addr,i} + (1-\rho_i)w_i$$

$$\rho_i = \lambda_i S_i$$

Calculating new clumping probabilities

$$n_{clump,i} = \rho_i P_{pkt,i} + \frac{(1-\rho_i)U_{pass,i}}{l_{train,i}} + P_{pkt,i}l_{send}$$

$$C_{link} = \frac{n_{pass,i}C_{pass,i} + [\rho_i+(1-\rho_i)U_{pass,i}]+n_{clump,i}}{(n_{pass}+1)}$$

$$F_{in,i} = \frac{C_{link,i}\lambda_{ring}}{\lambda_i + r_{rcv,i}}$$

$$P_{unclump,i} = \left[\frac{\lambda_i}{\lambda_i + r_{rcv,i}}\right]\left[\frac{\lambda_{ring} - \lambda_i - r_{rcv,i}}{\lambda_{ring}}\right]$$

$$F_{out,i} = (1-C_{link,i})^2F_{in,i} + C_{link,i}(1-C_{link,i})(F_{in,i} - P_{unclump,i}) + C_{link,i}{}^2(F_{in,i} - 1 - P_{unclump,i}) + C_{link,i}(1-C_{link,i})(F_{in,i} - 1)$$

$$C_{pass,i} = F_{out,i}\left[\frac{\lambda_i + r_{rcv,i}}{\lambda_{ring}-\lambda_i}\right]$$

Calculating final model outputs

$$V_{wait,i} = (1-\rho_i)r_{data,i}l_{data}\left[\frac{(l_{data}+1)(2l_{data}+1)}{6}\right] + (1-\rho_i)r_{addr,i}l_{addr}\left[\frac{(l_{addr}+1)(2l_{addr}+1)}{6}\right] +$$
$$(1-\rho_i)r_{echo,i}l_{echo}\left[\frac{(l_{echo}+1)(2l_{echo}+1)}{6}\right] - [(1-\rho_i)w_i]^2$$

$$V_{train,i} = \frac{V_{pkt,i}}{(1-C_{pass,i})} + \frac{l_{pkt,i}{}^2C_{pass,i}}{(1-C_{pass,i})^2}$$

$$V_{type,i} = \left[\frac{S_{cut\_in,i}}{S_{cut\_in,i}+S_{drain,i}}\right]^2\left[\sum_{j=1}^{l_{type}-1}\binom{l_{type}-1}{j}P_{pkt,i}{}^j(1-P_{pkt,i})^{l_{type}-1-j}(jV_{train,i} + [jl_{train,i}]^2) - [l_{train,i}P_{pkt,i}(l_{type}-1)]^2\right]$$

$$V_i = f_{data}(V_{data,i} + S_{data,i}{}^2) + f_{addr}(V_{addr,i} + S_{addr,i}{}^2) - S_i{}^2 + V_{wait,i}$$

$$c_i = \frac{\sqrt{\mathrm{Var}(S_i)}}{S_i}$$

$$Q_i = \rho_i + \frac{\rho_i{}^2\,(1+c_i{}^2)}{2(1-\rho_i)}$$

$$L_i = \frac{V_i + S_i{}^2}{2S_i}$$

$$W_i = (Q_i - \rho_i)\,S_i + \rho_i L_i$$

$$B_{type,i} = P_{cut\_train,i}\, n_{train,i}\, l_{type} + P_{cut\_pkt,i}\, l_{type} + \sum_{i=1}^{l_{type}-1} P_{pkt,i}\, n_{train,i}(l_{type} - i)$$

$$B_i = f_{data} B_{data,i} + f_{addr} B_{addr,i}$$

$$T_i = 1 + T_{wire} + t_{parse} + l_{send} + \sum_{j \neq i} z_{i\,j} \sum_{\substack{k=i+1 \\ (mod\ N)}}^{j-1} (1 + T_{wire} + t_{parse} + B_k)$$

$$R_i = W_i + (1-\rho_i) w_i + T_i$$