

Comparative Performance Analysis of Directed Flow Control for Real-Time SCI

Robert W. Todd, Matthew C. Chidester, and Alan D. George

High-performance Computing and Simulation (HCS) Research Laboratory
Department of Electrical and Computer Engineering, University of Florida
P.O. Box 116200, Gainesville, FL 32611-6200

Abstract

The distributed nature of routing and flow control in a register-insertion ring topology complicates priority enforcement for real-time systems. Two divergent approaches for priority enforcement for ring-based networks are reviewed: a node-oriented scheme called Preemptive Priority Queue and a ring-wide arbitration approach dubbed TRAIN. This paper introduces a hybrid protocol named Directed Flow Control that combines node- and ring-oriented flow control to yield greater performance. A functional comparison of the three protocols as implemented on the Scalable Coherent Interface (SCI) is presented, followed by performance results obtained through high-fidelity modeling and simulation.

Keywords: Discrete-event simulation, high-performance networks, real-time protocols, Scalable Coherent Interface

1. INTRODUCTION

The information revolution has created a world in which data is a valuable commodity. In the past, the worth of a network was judged by its ability to deliver data reliably and rapidly. With the emergence of a commercial market for video- and audio-on-demand and an increasing reliance on computer control of mission-critical applications in the military and industrial sectors, networks must also deliver data within a guaranteed maximum response time. As the number of users in the global network grows, the capacity and performance requirements for such real-time networks are rapidly increasing. The combination of a high-bandwidth, low-latency network protocol with real-time capabilities is necessary to meet these needs.

The protocols in this study implement real-time capabilities on a ring-based network. In addition to the cost-effectiveness of ring-based topologies for large-scale networks, chip-multiprocessor architectures frequently include rings as the on-chip interconnect [5]. Though the protocols in this paper are generically applicable to any ring-based network, the Scalable Coherent Interface (SCI) is used as the underlying transport [8]. As IEEE Standard 1596-1992, SCI provides a bus-like interface over unidirectional, point-to-point links to implement distributed shared memory. The protocol calls for link speeds of 1 GB/s and sub-microsecond latencies over distances of tens of meters. However, the base SCI protocol does not attempt to address real-time issues.

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE 2001		2. REPORT TYPE		3. DATES COVERED 00-00-2001 to 00-00-2001	
4. TITLE AND SUBTITLE Comparative Performance Analysis of Directed Flow Control for Real-Time SCI				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S)				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Florida, Department of Electrical and Computer Engineering, High-performance Computing and Simulation (HCS) Research Laboratory, Gainesville, FL, 32611				8. PERFORMING ORGANIZATION REPORT NUMBER	
				10. SPONSOR/MONITOR'S ACRONYM(S)	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
				12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited	
13. SUPPLEMENTARY NOTES					
14. ABSTRACT					
15. SUBJECT TERMS					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES 22	19a. NAME OF RESPONSIBLE PERSON
a. REPORT unclassified	b. ABSTRACT unclassified	c. THIS PAGE unclassified			

Providing real-time capability in a network requires enforcement of packet priorities. Such a priority scheme is required for implementation of algorithms such as Rate-Monotonic Scheduling (RMS) [10]. To support RMS, all packets are assigned a priority and the network must enforce transmission of higher-priority packets ahead of those with lower priorities.

Two protocols were originally proposed to provide real-time capabilities for SCI: Preemptive Priority Queue (PPQ) and TRAIN. Simulation shows neither solution to be optimal in sufficient numbers of test cases to be acceptable for a generalized standard. This paper presents a third solution to the SCI/RT problem called Directed Flow Control (DFC). DFC is a hybrid that combines selected aspects of the PPQ and TRAIN protocols to provide performance greater than either protocol alone.

Traditionally, real-time protocols for ring-based networks have taken a token-passing approach [2,16]. However, such a scheme is unable to capitalize on the concurrent bandwidth that inherently exists in a register-insertion ring network. Other flow-control schemes allow the concurrent bandwidth to be used efficiently but do not adequately address real-time performance needs [11,12]. The real-time protocols presented in this paper provide for both concurrent communication to make effective use of available ring bandwidth and priority-level enforcement for real-time communications.

This paper begins with an overview of the requirements for a real-time extension to SCI. Next, the three protocols are presented from a functional standpoint, illustrating the novel ways in which each protocol addresses priority inversions in a ring-oriented network. The performance of the protocols is then compared using simulation results from several common test cases.

2. REAL-TIME PROTOCOL EXTENSIONS TO SCI

While the three protocols presented in this paper were targeted specifically for SCI, each could be easily adapted to provide real-time capabilities to any ring-oriented network. For the purposes of direct comparison, all three protocols were simulated with SCI as the underlying network medium. Therefore, it is important to understand the requirements of a real-time protocol extension to SCI.

2.1 Real-Time Issues in SCI

In any physical network, it is often impractical to guarantee that high-priority packets will never be preempted by low-priority packets. When a high-priority packet is delayed while a low-priority packet is freely transmitted, a priority inversion is said to occur. Real-time functionality is achieved by avoiding priority inversions wherever contention for a shared resource may occur.

Figure 1 shows the basic structure of an SCI network interface. Like any register-insertion ring protocol, SCI requires input, bypass, and output queues. The address decode logic determines whether an incoming packet is destined for the local node or for some other node. The packet is then routed to an

input queue or the bypass queue, respectively. In order to output a packet, there must be sufficient free space in the bypass queue to hold all of the incoming data while the outgoing data is transmitted. If the bypass queue does not have sufficient free space, the node must hold its output data until enough incoming idle symbols or packets destined for the node are received to empty the bypass queue.

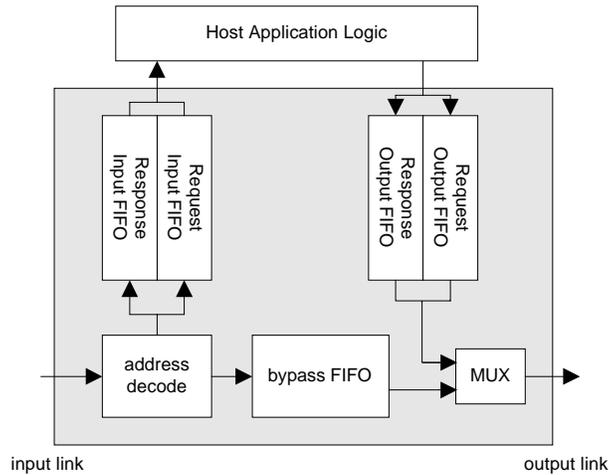


Figure 1. High-level structure of an SCI node.

In such a register-insertion ring system, priority inversions can occur in two places:

- 1) If there is insufficient space in the bypass queue to hold all incoming traffic while a node outputs its own data, a packet in the output queue must stall. Under certain traffic loads, high-priority packets waiting in a node's output queue may be forced to stall while lower-priority packets sent from an upstream node are passed along. This phenomenon will be called *output inversion*.
- 2) When a node's input queue becomes full, high-priority packets may be rejected even though lower-priority packets are occupying the input queue space. This problem will be called *input inversion*.

To provide real-time capabilities in SCI that satisfy the requirements of RMS theory, output and input inversion must be minimized. The mechanisms by which both types of inversion are handled in the three real-time protocols will be discussed below.

All three real-time protocols assume the use of prioritized FIFO queues for the input and output queues in the SCI node interface. Packets will be removed from the queue first by highest priority, then by oldest packet within that priority. All solutions also support at least 128 priority levels, allowing the protocols to serve the real-time needs for a large variety of applications.

2.2 Preemptive Priority Queue Protocol

The *Preemptive Priority Queue* (PPQ) protocol is an example of a queuing theory solution to the SCI real-time problem [1]. PPQ can be described as a node-oriented protocol; each individual node makes prioritization decisions based solely upon its own current state. Low-priority packets are preempted in favor of high-priority packets whenever a full-queue condition is encountered within the node.

The system-level structure of a PPQ node is shown in Figure 2. A PPQ node is similar in structure to base SCI. The *Host Input Queue* (HIQ), *Link Input Queue* (LIQ), and *Bypass Queue* (BQ) are preemptive priority queues, while the *Echo Waiting Queue* (EWQ), and *Response Waiting Queue* (RWQ) are content-addressable queues.

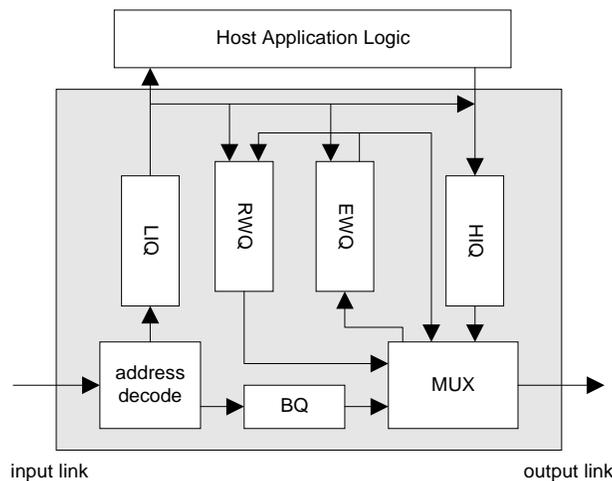


Figure 2. High-level structure of a PPQ node.

PPQ solves the input inversion problem by allowing low-priority packets in the LIQ to be preempted by high-priority packets when there would otherwise be insufficient space to hold the high-priority packet. A preempted low-priority packet is busy-retried using an echo packet in exactly the same way that an incoming packet in base SCI is retried when the input queue is full.

In PPQ, a packet can be preempted from the destination's input queue at any time. Therefore, an acknowledgment echo for a packet is not sent until the packet is actually serviced from the destination node's LIQ. However, in base SCI, a packet that is accepted into the destination's input queue generates an acknowledgement echo as soon as it enters the queue, allowing the packet to be removed from the sender's output queue at the earliest possible opportunity. PPQ requires the sending node to keep the packet in its EWQ in the event that it is preempted from the destination node's LIQ and must be retried. The RWQ is a higher-level structure that handles retransmissions in the event of a timeout condition while the destination node is servicing a request.

Output inversion is prevented in PPQ by allowing the BQ to be preemptable. If there is insufficient free space in the BQ to output a packet, a node can attempt to make room by preempting any packet in the BQ with a lower priority than the packet waiting to be transmitted. The preempted packets are converted into busy-echos that will cause the source node to retry them.

A significant feature of PPQ is that the BQ is also prioritized. If a high-priority packet enters a node's BQ behind a low-priority packet, the packets may be reordered in the queue, allowing the high-priority packet to be transmitted out of the node in front of the low-priority packet. This technique introduces a potential problem whereby a low-priority packet may become stuck in an intermediate node's BQ for an indefinite period of time. If high-priority packets continually enter a BQ behind a waiting low-priority packet yet there is always sufficient free space that the low-priority packet is never retried, the low-priority packet may wait indefinitely. The corresponding slot in the EWQ of the sending node for this packet will also be tied up indefinitely.

To solve this problem, two solutions are feasible. The first solution is to temporarily increase the priority of all packets waiting in a BQ such that the oldest packets will eventually have sufficiently high priority to be sent. This method introduces priority inversions at the destination node where an aged packet may have an effective priority higher than a recently sent packet when their original priorities were exactly the opposite.

The second solution uses priority inheritance at the source node. In this case, the low-priority packet can remain stuck in a BQ until the slot occupied in the sender's EWQ is needed for a higher-priority packet. The sending node transmits an event message to all nodes requesting that the original packet inherit the higher priority. All nodes check to see if the packet is in their BQ or LIQ and alter the priority accordingly.

The most desirable feature of PPQ is that it maintains the low latency of base SCI. In a lightly loaded system or for the high-priority packets in a more heavily loaded system, the latency performance is similar to SCI. Also, PPQ easily handles switched-ring configurations. However, the preemptive nature of the protocol results in numerous retries of low-priority packets that unnecessarily consume bandwidth on the ring. The bandwidth consumption significantly reduces the overall performance of PPQ at high loadings. The magnitude of this effect will be examined in Section 3.

2.3 *TRAIN Protocol*

The TRAIN protocol is a starkly different solution to real-time demands over SCI than PPQ [14]. This protocol solves input and output inversion by arbitrating for bandwidth on the ring collectively rather than within a node's queues. This technique can be described as a ring-oriented protocol. Rather than allowing each node to make individual prioritization decisions, all nodes collectively arbitrate for access

to the network through a data structure broadcast on the ring. Only the nodes with the highest-priority traffic are granted permission to send their data packets.

The TRAIN protocol defines two basic modes of operation: *immediate send mode* and *arbitrated send mode*. In both modes, a data structure called a *train* is broadcast around the ring. The train is constructed of two special types of packets: a *LocalMotive*, which is created and sent out by the *TicketMaster*; and attached tickets. In immediate send mode, only the LocalMotive is used. It keeps a running tally of how much bandwidth is currently available on the ring. Nodes wishing to transmit packets simply decrement this field by the amount they wish to send and then consequently send the packet after the LocalMotive has passed. If the TicketMaster detects that the ring bandwidth is approaching some maximum capacity, the ring is switched into arbitrated send mode.

In arbitrated send mode, the transmission requests of all nodes are sampled on the interconnect by using an *arbitration train*. As the arbitration train circles the ring, nodes wishing to transmit packets attach a ticket to the train. Because the LocalMotive stores the total amount of bandwidth being arbitrated in a *size* field, each node, while inserting its own tickets into the train, sums the total amount of bandwidth that the requests would consume. If this total begins to exceed the amount of available bandwidth, the train is truncated and the remaining tickets are discarded. Figure 3 shows a ticket insertion example where the ticket being inserted causes an overflow and all remaining tickets are discarded.

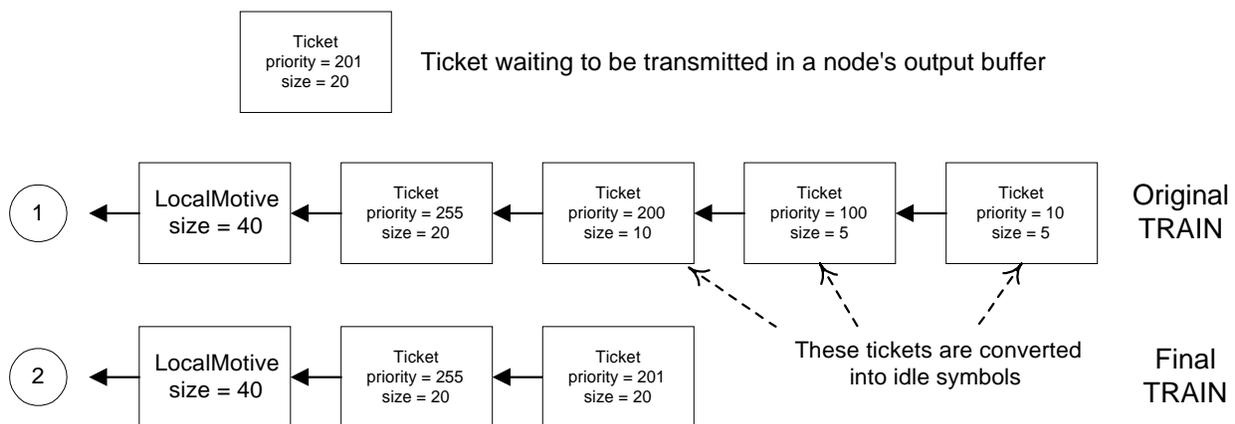


Figure 3. Example of ticket insertion in the TRAIN protocol.

In this example, a node has a packet with a size of 20 and a priority of 201 waiting to be sent. State 1 shows the contents of the arbitration train immediately before it passes through the node. The LocalMotive indicates that the maximum total size for all requests that can be granted is 40. As the LocalMotive passes through the node with the waiting packet, the node makes a note of this maximum size. When the first ticket passes through the node, it notes that the priority is greater than its own packet

and therefore passes the ticket through unchanged. The next ticket, however, has a priority of only 200. Therefore, the node will insert its ticket instead. Since the sum of the previous ticket and the new ticket now equals the maximum size specified in the LocalMotive, the node will strip the remaining tickets from the arbitration train, leaving the final train shown in state 2.

After an arbitration train circles the ring completely, the TicketMaster node sets a field in the LocalMotive that indicates that the train is now becoming a grant train. The grant train then circles the ring, granting any node whose ticket is still present in the train the permission to send its packet. As a result of the arbitration phase, the grant train will only contain the tickets for the highest-priority packets that will fit into the available ring bandwidth. The cycle is then repeated with another request train. This mechanism prevents output inversion from ever occurring.

Input inversion is handled by allowing a destination node to delete tickets for packets that would not fit into its input queue. Since a ticket will progress all the way around the ring before returning to the source and granting permission to send the associated packet, it is guaranteed to pass through the destination node. This approach gives all destination nodes an opportunity to delete tickets for packets that will not fit into their input queues. If the destination node lies between the TicketMaster and the source node, then the tickets will be deleted from the grant train, resulting in an underutilization of ring bandwidth. If the highest-priority item in the destination node's input queue is of lower priority than the deleted ticket, then a priority inheritance scheme can be used to minimize the duration of the input inversion.

Because of the arbitrated nature of the TRAIN protocol, the latency of every transaction is increased. For enhanced performance, a system can switch from arbitration mode back into immediate send mode when the TicketMaster detects an under-filled request train. Even when using immediate send mode, a node must minimally wait for the LocalMotive to make a complete revolution between every packet transmission. However, the arbitration guarantees that no packet will be rejected and retried, making efficient use of the bandwidth. These performance features will be demonstrated via the simulation results presented in Section 4.

3. THE DIRECTED FLOW-CONTROL PROTOCOL

The DFC protocol was developed to blend the best features from the other real-time SCI protocol extensions into a single protocol that preserves the low latencies and high bandwidths of base SCI. The original idea behind the DFC protocol was presented in [9]. The concepts were refined and the protocol developed in [3,4]. The original goals of DFC are accomplished by following several guidelines:

- 1) Flow control is generated on an "as needed" basis; only those nodes that are congested and are in need of flow control will generate flow-control symbols.

- 2) Each node maintains a minimal amount of state information about the other nodes on the ring to allow the node to make its own flow-control decisions.
- 3) Flow-control symbols are used only when a node must change its state; each symbol broadcasts the state change to all other nodes on the ring.
- 4) The flow-control symbols themselves are kept small.

DFC provides a mixture of ring-based and node-based bandwidth arbitration. Ring-based arbitration is used to broadcast flow-control state information about each of the nodes on the ring in order to reduce priority inversions, relieve congestion, and minimize busy-retries. Node-based arbitration allows the flow-control overhead to be kept small but requires that each node dedicate some amount of register space to hold state information about all of the other nodes on the ring.

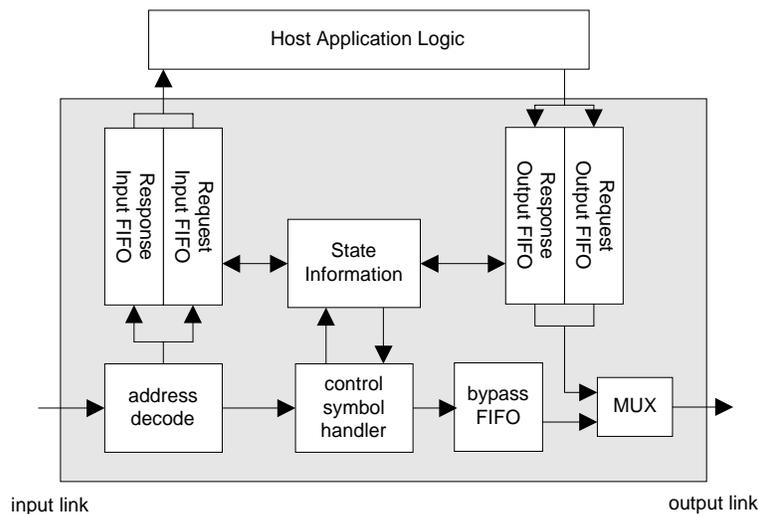


Figure 4. High-level structure of a DFC node.

The structure of a node using the DFC protocol is similar to that of base SCI. Figure 4 shows a block diagram of a DFC node. The only addition to the node structure of a basic SCI node is control symbol logic. The logic to implement DFC requires additional internal register space for storing state information.

The flow-control information in DFC is carried in 32-bit *control symbols*. Congested nodes broadcast control symbols to the other nodes on the ring in order to relieve their congestion. Control symbols always circulate the ring exactly once and are removed by the node that originally sent them. Each node is responsible for monitoring the status of the other nodes on the ring by monitoring control symbols.

Two types of control symbols are used to enforce the real-time priority restrictions: *STOP-THRU* and *STOP-TO* control symbols.

3.1 *STOP-THRU Behavior*

To solve the output inversion problem, DFC uses *STOP-THRU* control symbols to restrict the priority of traffic that is permitted to pass through a congested node, thus allowing the node to empty its bypass queue and send its own traffic. When a congested node has a packet to send but its bypass queue has insufficient free space to allow the packet to be sent, the node circulates a *STOP-THRU* control symbol. The control symbol carries the priority of the packet that the congested node has waiting in its output queue. Each of the other nodes on the ring, upon receiving the *STOP-THRU* control symbol from the congested node, will not transmit any packets waiting in their own output queues that would pass *through* the congested node unless the packets have a priority greater than or equal to the priority of the packet the congested node is trying to send. It is assumed each node obtains knowledge of the network map during the SCI initialization sequence. In a ring with sequentially numbered nodes, determining whether a message will pass through a particular node is trivial.

If all nodes follow the restrictions imposed by the *STOP-THRU* control symbol and there is bandwidth available for the priority of the traffic in the congested node's output queue, the congested node will eventually be able to empty its bypass queue and send its packet. After the congested node is able to send its packet, it must send a second *STOP-THRU* control symbol to either lift the restriction entirely or change the priority to match that of the next packet in its output queue.

This scheme allows efficient use of the remaining network bandwidth by allowing nodes to send any packet that will not interfere with the congested node. Nodes with empty bypass queues will not restrict traffic passing through them since they will be able to immediately send any packet that is placed in their output queue.

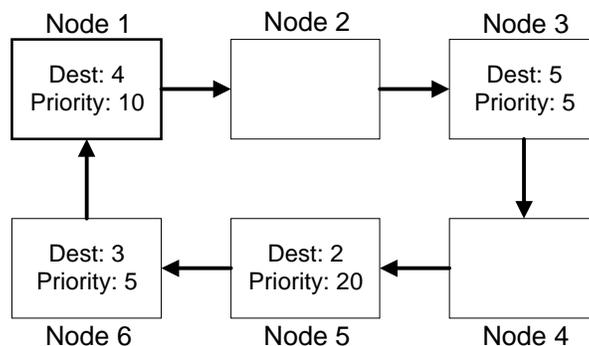


Figure 5. Example of STOP-THRU arbitration in DFC.

For an example of this behavior, consider the situation in Figure 5. In this figure, nodes 1, 3, 5, and 6 have packets to send to the destination nodes shown with the priorities listed. Assume that node 1 has a full bypass queue while the other nodes have empty bypass queues. Node 1 has a packet of priority 10 to transmit but cannot do so due to its full bypass queue. Therefore, it will circulate a *STOP-THRU* control symbol with a priority of 10. The state shown in Figure 5 is the state of all nodes after the last node (i.e. node 6) receives the *STOP-THRU* symbol from node 1.

Provided that there are no other flow-control restrictions being imposed (i.e. restrictions on traffic through or to other nodes), the nodes may proceed as follows. Node 3 can send its packet because it will not pass through node 1. Node 5 can send its packet because, even though it will pass through node 1, it has a priority greater than the priority of the packet waiting to be sent at node 1. Node 6 cannot send its packet because it has a lower priority than node 1's packet.

3.2 *STOP-TO Behavior*

To address input inversion, *STOP-TO* arbitration is triggered when a node's input queue becomes full or reaches a high-water mark. This congested node sends out an initial *STOP-TO* symbol that notifies all of the other nodes on the ring that the congested node is now in arbitration mode. When a node is in arbitration mode, it is referred to as a blocked node. The other nodes on the ring cannot send any packets to a blocked node unless they are explicitly granted permission in a subsequent *STOP-TO* control symbol from the blocked node.

Whenever an element is removed from the blocked node's input queue, another *STOP-TO* control symbol is sent around the ring. This *STOP-TO* control symbol simultaneously grants a node permission to send a packet to the blocked node while collecting priority information from the other nodes with waiting traffic for the blocked node. The priority information is used in the next *STOP-TO* control symbol to grant the node with the highest-priority packet permission to send. When the blocked node's input queue reaches a low-water mark, a *STOP-TO* control symbol is used to notify the other nodes on the ring that the blocked node is no longer in arbitration mode and all packets may be sent freely to the node.

Figure 6 shows the typical life-cycle of a node that enters arbitration mode, uses *STOP-TO* flow control until its input queue empties, then leaves arbitration mode. In this figure, a four-node ring is depicted in four different states proceeding chronologically from the first state to the fourth state. In each of the four states, node 1 is sending a *STOP-TO* control symbol around the ring. The figure has been simplified to show only the *STOP-TO* symbols sent by node 1; other control symbol traffic and data transfers may also be taking place between and during the four states shown. The numbers denoted inside the node boxes each represent the priority of a packet waiting to be sent to node 1. If no number is shown, then no packets are waiting at that node for node 1.

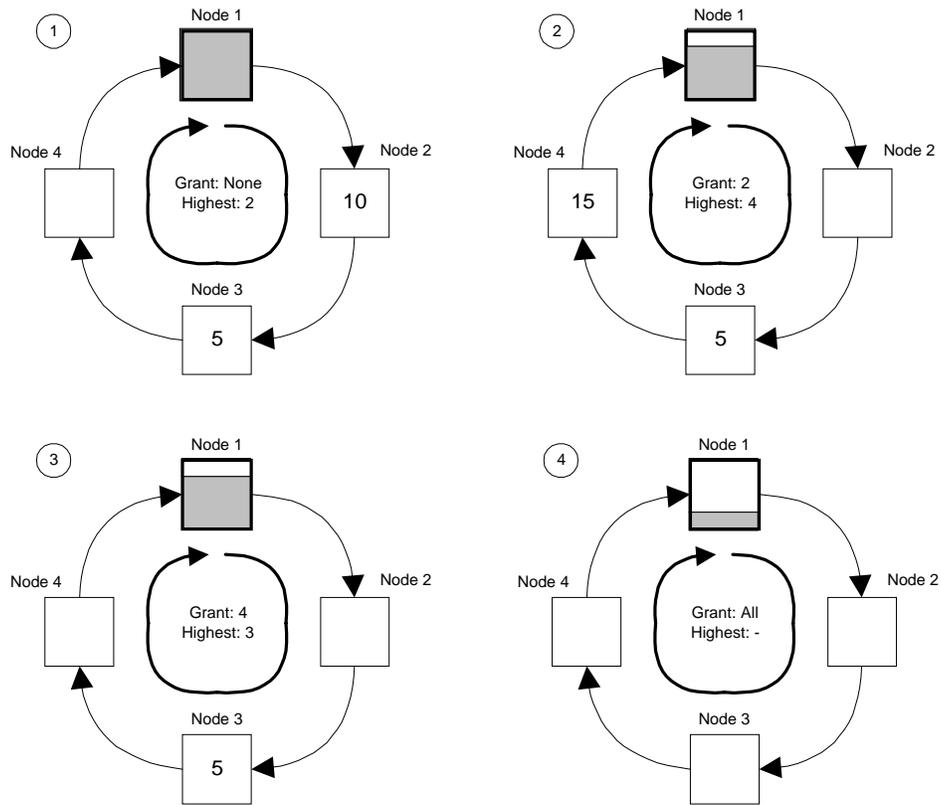


Figure 6. Example of STOP-TO arbitration in DFC.

In the first state, the input queue in node 1 has filled, and so it sends an initial *STOP-TO* around the ring to notify the other nodes of its status. This *STOP-TO* symbol does not grant access to any of the nodes to send to node 1 since node 1 does not have any free input queue space. At the time this initial control symbol circulates the ring, node 2 has a packet of priority 10 to send to node 1 while node 3 has a packet of priority 5. When the control symbol returns to node 1, it indicates that the highest-priority packet waiting for node 1 is held by node 2.

The second state occurs when node 1 empties an item from its input queue. To fill the empty space, a second *STOP-TO* control symbol is sent to grant access to node 2 since it had the highest-priority packet waiting when the previous control symbol was sent. When this second *STOP-TO* returns to node 1, it will indicate that node 4 now has the highest-priority packet waiting since its priority of 15 is higher than the priority 5 packet still waiting at node 3. As a result, when node 1 frees another input queue slot and sends the next control symbol as in state 3, it will grant node 4 permission to send its traffic to node 1.

Finally, when node 1 empties its input queue sufficiently far that it reaches a low-water mark, a final *STOP-TO* control symbol is used to notify all nodes that they may now send freely to node 1. State 4

shows this event. The final *STOP-TO* symbol acts as the equivalent of a grant to all nodes. Since node 1 is leaving arbitration mode, the *STOP-TO* symbol does not need to collect any information about the waiting packets.

DFC requires greater hardware complexity in terms of state storage space than the other approaches. However, since the flow-control symbols are kept minimally small and are only used on an as-needed basis, the impact on the latency and throughput of base SCI is minimized in DFC. In fact, the flow control in DFC is actually less restrictive than that of base SCI, yielding higher performance in many cases. DFC restricts the output of traffic only when it will interfere with another node's ability to send. Traffic that will not affect a congested node (not counting the required echo packets) can be sent freely. In contrast, the base SCI flow-control protocol has been shown to be non-optimal in its restriction of the entire ring when a single node is trying to send [12].

4. SIMULATION EXPERIMENTS AND RESULTS

Simulation was used as a tool to compare the relative performance of the three real-time protocols to one another and to base SCI. The experiments also show the ability of the protocols to enforce priority restrictions on SCI packets. To model and simulate the protocols, the *Block-Oriented Network Simulator* (BONeS) *Designer* tool from Cadence Design Systems was used.

BONeS is an integrated software package for event-driven simulation of data transfer systems. It allows for a hierarchical, dataflow representation of networks with the ability to import finite-state machine diagrams and user-developed C/C++ code as functional primitives. BONeS was developed to model and simulate the flow of information represented by bits, packets, messages, or any combination of these. As a result, the models constructed to evaluate the three real-time protocols and base SCI are of a high fidelity, with accuracy down to the clock cycle. An overview of BONeS can be found in [13] while the validation of our base SCI models against testbed hardware can be found in [6].

A detailed description of the implementation of the BONeS models for the three protocols is beyond the scope of this paper. However, the construction of the base SCI model is detailed in [7], the PPQ and TRAIN models are discussed in [15], and the structure of the DFC model is outlined in [3].

Although each real-time application is different in its requirements of an interconnect, a series of case studies have been developed to allow direct comparison of each real-time protocol. The models also allow evaluation of specific real-time applications. Table 1 outlines the simulation parameters that remain constant through all case studies. Ideal links with zero propagation delay were used since typical SCI applications involve short links where the per-node delay dominates the overall latency.

Table 1. Simulation parameters.

Parameter	Value
Link Speed	1.0 GB/s
Link Propagation Delay	Ideal (0 clock cycles)
Per-node Delay	Ideal (1 clock cycle)
Traffic Distribution Type	Poisson
Requestor Input Queue Size	5 Packets
Responder Input Queue Size	5 Packets
Transaction Type	64-byte remote reads (read64)
Network Topology	8-node ring

Each case study measures the cumulative effective throughput in the system and the average one-way latency for all packets. Figure 7 shows the locations at which each statistic is measured. The effective throughput is the final data throughput of the system only (i.e. the overhead bits and busy-retried packets are not counted as part of the throughput). The cumulative throughput is the sum of all data received by all nodes divided by the amount of simulated time. The one-way latency is the average time a request or response packet takes to successfully reach its destination including time spent in queues and time spent retrying busied commands. All simulations are run for a sufficiently long time that a steady-state is reached and the throughput and latency are averaged over hundreds of thousands of messages.

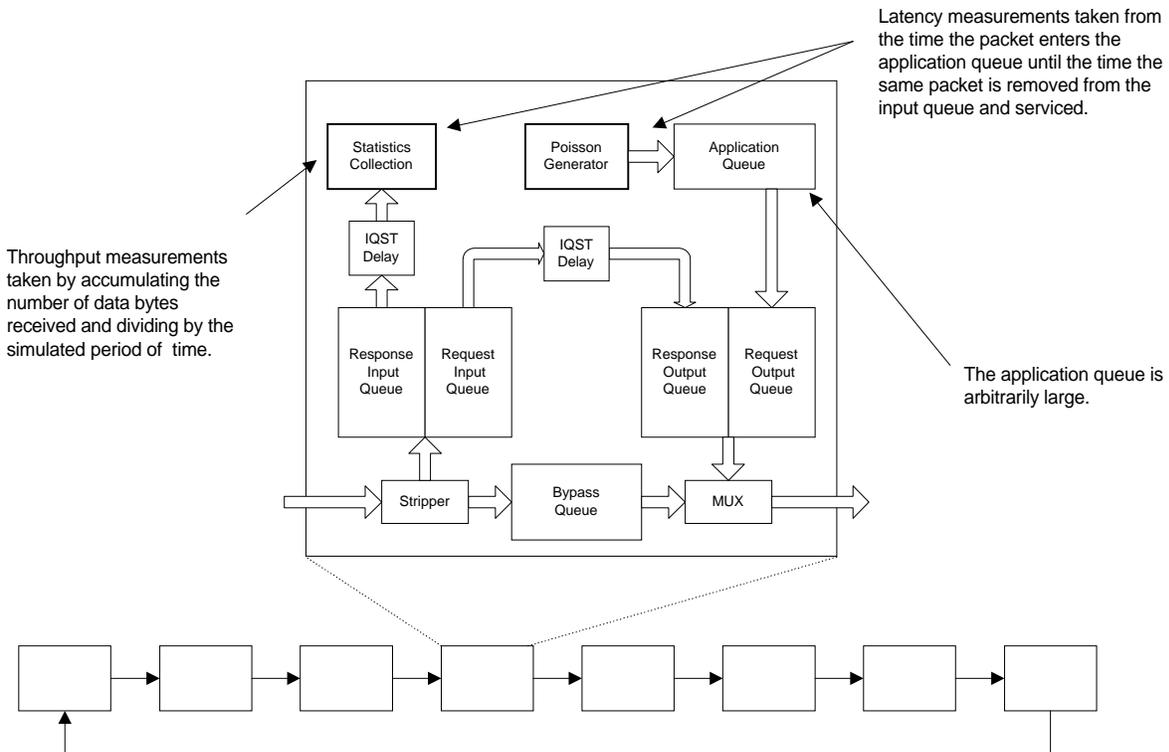


Figure 7. Points of measurement for quantitative statistics.

For each simulation, a number of parameters are varied. Among these parameters are the destination node, the number of priorities in use, the offered load per node per priority, and the input queue service time (IQST). As IQST is increased, the input queues of all nodes remain full for longer periods of time, increasing the occurrence of input inversion.

To test the efficiency of each real-time protocol to handle output and input inversion, some basic metrics need to be determined to find an interesting range for offered load and to find values of IQST that will yield traffic patterns that do and do not result in full input queue conditions. To find these values, the maximum throughput of the 8-node SCI ring must first be found.

Theoretically, an N -node SCI ring using 1 GB/s links is able to attain a total of N GB/s worth of concurrent transfers. However, this total is unreachable for several reasons. First, there is significant overhead in every SCI transaction. For the *read64* transactions used in these experiments, each transaction generates a 16-byte request packet followed by an 8-byte echo, then the 64 bytes of data plus 16 bytes of overhead is sent in the reply packet followed by its 8-byte echo. Therefore, only 57% of the raw transmitted bits represent actual data.

The second barrier to reaching an ideal N GB/s throughput is the distribution of the data. Since all nodes uniformly distribute their requests to all other nodes on the ring, the data passes through half of the links in the system, on average. With uniform traffic distribution, the maximum total bandwidth of any ring-based system is only twice the link speed, in this case 2 GB/s. Taking the aforementioned overhead into consideration, this limit leaves a peak effective throughput of 1.14 GB/s for any SCI ring. Assuming an even load distribution, each individual node can sustain a maximum outgoing throughput of $1.14/N$ GB/s.

For the 8-node ring used in these experiments, the theoretical maximum sustainable throughput per node is about 146 MB/s. Therefore, the case studies that are intended to measure peak throughput performance will use offered load values in the range of 1 MB/s/node to 150 MB/s/node. Due to other overhead such as busy-retries and idle symbols that are not accounted for in this simple mathematical estimate, the maximum throughput of all protocols in all test cases will be less than this value.

To test the protocols' performance in the presence of output inversion, the IQST value is chosen to be low enough that all responder input queues can service the incoming data faster than it is received. The responder input queue of a node is able to service a maximum of $(64 \text{ bytes/response}) / (\text{IQST sec/response})$ bytes of data per second. Using an IQST value of $0.1 \mu\text{s}$ allows 610 MB/s of incoming data to be serviced. This rate is significantly higher than the maximum of 146 MB/s data in the 8-node test system, therefore resulting in very few input inversion situations.

Testing the other extreme — performance in the presence of much input inversion — requires the service rate of the responder input queue to be slower than the incoming data rate. An IQST of $1.0 \mu\text{s}$

satisfies this requirement, yielding a service rate of 61 MB/s that is less than half the peak incoming data throughput.

4.1 Case Study #1: Single-priority performance

The first case study simulates a simple single-priority system to test the baseline performance of each protocol and to allow direct comparison with the base SCI model. The throughput levels achieved with base SCI, PPQ, TRAIN, and DFC are shown in Figure 8.

For an IQST of 0.1 μ s, Figure 8a shows that the DFC protocol yields the highest sustained throughput at over 845 MB/s. This throughput is actually higher than that of base SCI because the DFC flow-control mechanism is less restrictive. Since PPQ uses the same flow-control mechanism as SCI within priority levels, the 674 MB/s throughput of PPQ is close to the 698 MB/s performance of base SCI. Because of the need to participate in the ring-wide arbitration scheme, TRAIN throughput levels out at only 611 MB/s.

The arbitration mechanism in TRAIN increases overall latency as well. While the one-way latencies for packets at low loading levels in base SCI, PPQ, and DFC were all measured to be about 0.17 μ s, the TRAIN protocol yield latencies over twice as high at 0.36 μ s.

The overhead of busy-retries in the PPQ protocol is significantly magnified for longer IQST. As several input queues around the ring begin to fill and reject new incoming packets, the source and destination nodes begin a very fast *ping-pong* between request/response send packets and their subsequent busy echos. This flow can prevent other nodes on the ring from being able to transmit their own packets.

With an IQST of 1.0 μ s, Figure 8b shows that base SCI, TRAIN, and DFC each achieve similar throughputs of about 290 MB/s. Since the bottleneck is now at the input queues, the differences in output flow control between the three protocols do not significantly affect the results. However, the busy-retry effect of PPQ rapidly dampens the system's throughput as the loading increases.

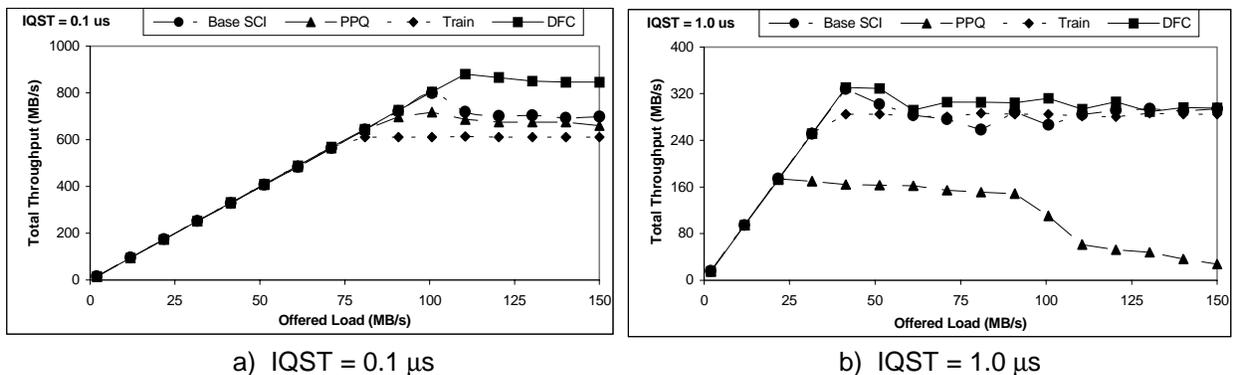


Figure 8. Effective throughputs for base SCI, PPQ, TRAIN, and DFC in single-priority experiments.

These results show that the low overhead and efficient blocking mechanism of DFC is significantly better than the other protocols at handling output-bounded traffic loadings as in the $0.1 \mu\text{s}$ IQST case. For input-bounded loadings such as the $1.0 \mu\text{s}$ IQST case, neither DFC, SCI, nor TRAIN have any relative throughput advantages while PPQ suffers from a busy-retry problem. In all cases, the latency of TRAIN is significantly higher than any of the other approaches.

4.2 Case Study #2: Multi-priority performance

The second case study involves the reaction of each real-time protocol to saturation in a multi-priority environment. For simplicity, only three different priority levels are used on the interconnect. As the offered load for each priority level ramps up, the throughput and latency is measured.

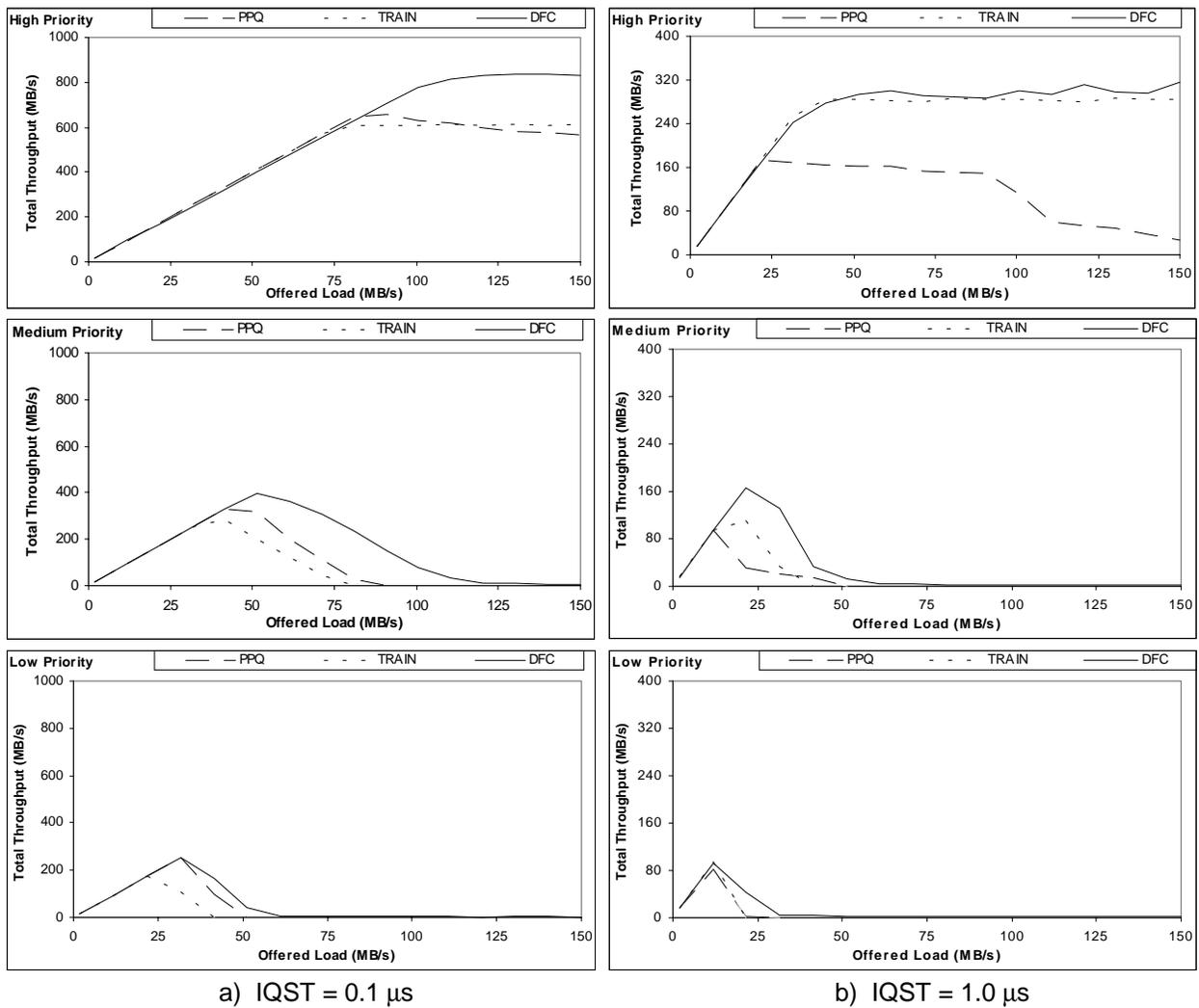


Figure 9. Effective throughputs for PPQ, TRAIN, and DFC in multi-priority experiments.

Figure 9a illustrates the throughput achieved by the PPQ, TRAIN, and DFC protocols with IQST = 0.1 μ s. In this scenario that is dominated by output inversion, DFC performs the best of the three in terms of effective throughput at high offered loads with TRAIN second and PPQ third. DFC provides sustained throughput of 837 MB/s. The PPQ throughput peaks at 660 MB/s, which is somewhat higher than TRAIN's 611 MB/s. However, as the offered load increases, the effect of busy-retries steadily reduces PPQ's throughput while that of TRAIN remains constant.

The one-way latency results for multiple priorities at an IQST of 0.1 μ s are similar to those of the single-priority test. At low loadings, DFC provides the lowest latencies at 0.17 μ s for all priority levels while PPQ and TRAIN yield 0.20 μ s and 0.36 μ s, respectively. The average latency for a given priority remains relatively constant as the loading increases and immediately approaches infinity for each priority level as it is denied bandwidth on the ring. These points correspond with the roll-off regions in the throughput plots of Figure 9. At these levels of high loading, the latency for each packet is dominated by the waiting time in the arbitrarily-large application queue.

To illustrate the input-inversion dominated case, Figure 9b shows the multi-priority throughput with IQST = 1.0 μ s. As in the single-priority tests, a longer IQST most negatively affects PPQ. While TRAIN and DFC are each able to maintain around 300 MB/s, the PPQ throughput rapidly declines as busy-retries consume more of the bandwidth.

As in the single-priority tests, the throughput of DFC is unmatched in the output-limited cases. As illustrated by the roll-off rates in the medium- and low-priority graphs of Figure 9, this fact also results in real-time systems that are able to handle a larger amount of lower-priority traffic in the presence of higher-priority traffic that does not consume all of the available bandwidth. In the input-limited situations, the throughput of TRAIN is comparable to that of DFC, but the latency of the TRAIN is almost twice that of DFC.

4.3 Case Study #3: 128-priority performance

The third case study is meant to demonstrate the priority enforcement for all three protocols. The system is flooded with packets of 128 different levels of priority. The traffic load is evenly distributed among all priority levels. The effective throughputs and latencies were measured at each priority level to insure correct real-time behavior. A sampling of five priorities was then taken and the results shown in Figure 10. The simulation was run for an IQST value of 0.1 μ s.

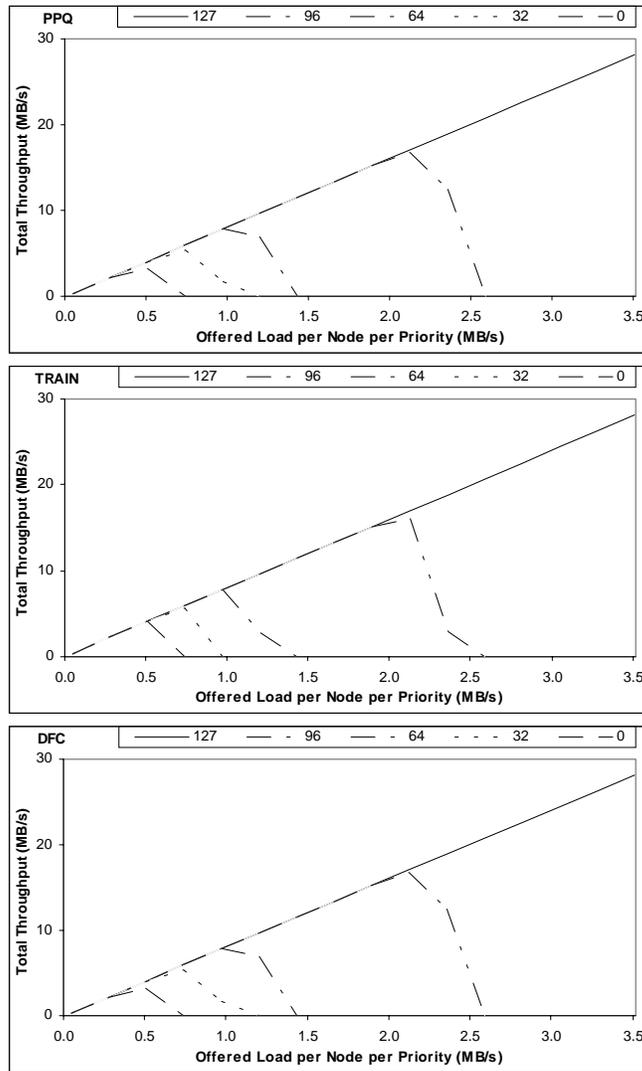


Figure 10. Effective throughputs for PPQ, TRAIN, and DFC in 128-priority experiments.

For correct real-time behavior, the throughputs of the lower-priority packets should roll off as the offered load increases and the higher-priority packets begin to consume all available bandwidth. The results demonstrate this ability for all three protocols to correctly limit lower-priority traffic in the presence of high-priority traffic.

4.4 Case Study #4: “Hot-spot” performance

The final case study shows the effects of a single overloaded node on the performance of the real-time interconnect as a whole. Seven nodes on the ring send a variable percentage of their total fixed-priority traffic, ranging from 50% to 100%, to one “hot-spot” node while the remainder of their

offered load is distributed evenly among the other nodes. The IQST on all nodes is fixed at 0.1 μ s. The total effective throughput for the ring is shown in Figure 11.

The throughput peaks of PPQ and TRAIN are approximately the same — in the range of 330 to 500 MB/s — in this test. Like the other tests, the throughput of PPQ degrades at higher loadings. With a range of 560-750 MB/s, DFC provides significantly higher performance results than the other protocols in this test. In this case, not only is the performance of DFC over 50% higher than the other protocols, but as the “hot-spot” loading is increased from 50% to 100%, the total throughput of DFC degrades only about 25% versus a 34% degradation in the PPQ and TRAIN protocols.

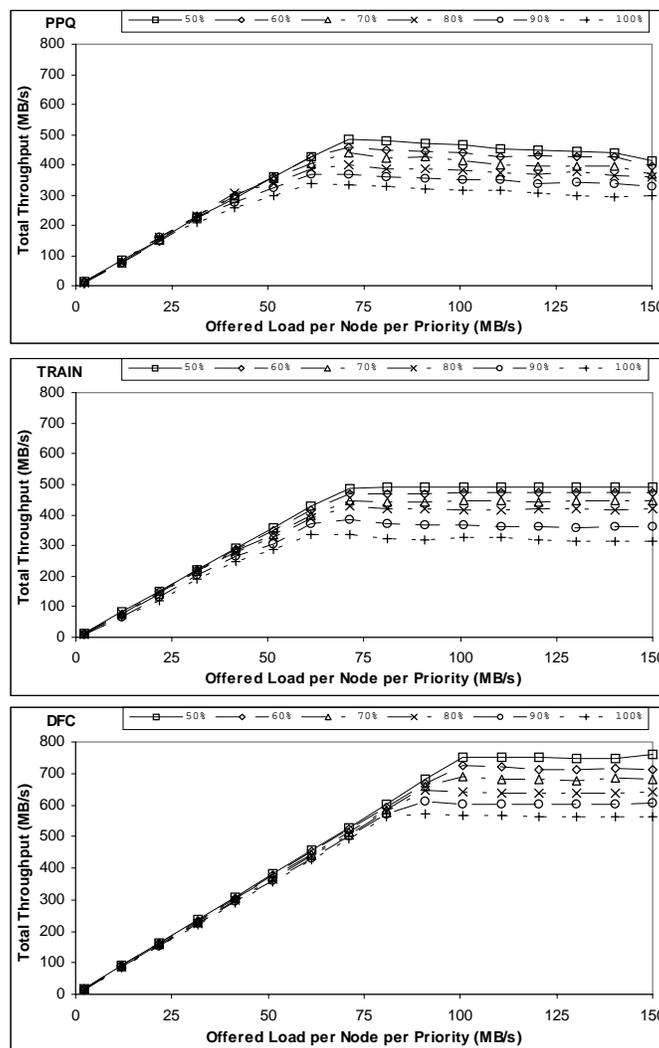


Figure 11. Effective throughputs for PPQ, TRAIN, and DFC in “hot-spot” experiments.

At the 100% “hot-spot” loading level, all 7 nodes are sending all of their offered load to the single “hot-spot” node. This situation means that the total throughput of the system is limited by the input queue service rate of the “hot-spot” node. For an IQST of 0.1 μ s and 64-byte data packets, a maximum of 610 MB/s can be serviced. The DFC protocol obtains 92% of this maximum possible throughput while the TRAIN and PPQ protocols reach only 51% and 49% of the ideal values, respectively.

The superior performance is attributable to the fact that DFC’s method of arbitration allows the “hot-spot” node to enable packets to be transmitted to itself on a space-available basis. In TRAIN, nodes place tickets in the arbitration train that are subsequently removed by the “hot-spot” node due to a full input queue condition. As a result, nodes waste ticket requests trying to send to the busy node when they could easily be sending other traffic. PPQ has a similar effect except that the packets are actually sent to the busy node and are subsequently busy-retried.

5. CONCLUSIONS

In this paper, we have presented three protocols that provide priority-based, real-time networking capabilities to SCI. The PPQ protocol takes a node-oriented approach to flow control while the TRAIN protocol employs ring-wide arbitration. The DFC protocol joins the two methods by providing a lightweight form of ring-oriented arbitration with state-based, node-centric decisions for flow control. High-fidelity simulation has shown the DFC protocol to have low latencies comparable to those of base SCI and the PPQ protocol, coupled with the highest throughputs of any of the approaches in a variety of case studies.

To improve the performance of the PPQ and TRAIN protocols, several modifications have recently been proposed. These additions include mechanisms to improve the throughput of PPQ under high loadings or long IQST delays by limiting the busy-retries, and a “gearshift” mechanism for the TRAIN protocol to allow a TRAIN system to operate exactly like base SCI under low loadings and switch into arbitration mode as needed. The design, specification and analysis of these and other potential enhancements are the subject of future study.

While the DFC protocol has superior performance, the hardware complexity and storage space required to implement the protocol may make a modified version of the PPQ protocol a more attractive option. Also, the ways in which DFC can operate in a switched network environment or interoperate with base SCI are as yet not clear. The PPQ protocol offers a more straightforward approach for switching and base SCI interoperability.

Future research should also consider other factors that would affect the real-time performance. Example areas of interest include larger ring sizes, non-ideal link and forwarding delays, and

extensions to support more complex topologies such as direct networks based on multi-dimensional tori and indirect networks involving switches. Integrating real-time nodes with traditional SCI hardware as well as other real-time networks is also an avenue for future investigation.

As often happens in the real world, the optimum solution is probably not to be found in either extreme of a strictly node-oriented protocol such as PPQ or in an arbitrated ring-oriented protocol like TRAIN. Instead, a mixture of the two approaches can be taken with careful attention to minimizing overhead to yield a superior solution. One such promising approach is DFC.

ACKNOWLEDGEMENT

This work was supported in part by the Naval Air Warfare Center (NAWC) and the Office of Naval Research (ONR), in concert with the IEEE P1596.6 Working Group. In addition, the authors acknowledge Mr. Ralph Lachenmaier at NAWC for insightful ideas and feedback in the development of the DFC protocol and the anonymous reviewers for helping improve the presentation of the content in this paper.

REFERENCES

- [1] D. Anderson, Proposal to the P1596.6 (SCI/RT) Working Group for a preemptive priority queue protocol, White Paper, IEEE P1596.6 Working Group, Apr. 1995.
- [2] B. Chen, S. Kamat, W. Zhao, Fault-tolerant, real-time communication in FDDI-based networks, *IEEE Computer*, Vol. 30, No. 4, 1997, pp. 83-90.
- [3] M. Chidester, Specification and simulation of a directed flow control for ring-based, real-time networking, M.S. Thesis, Department of Electrical and Computer Engineering, University of Florida, 1998.
- [4] M. Chidester, A. George, R. Todd, R. Lachenmaier, Specification of a directed flow control for SCI/RT, White Paper, IEEE P1596.6 Working Group, Jan. 1997.
- [5] L. Codrescu, D. Wills, J. Meindl, Architecture of the Atlas Chip-Multiprocessor: Dynamically Parallelizing Irregular Applications, *IEEE Trans. on Computers*, Vol. 50, No. 1, 2001, pp. 67-82.
- [6] A. George, J. Markwell, R. Fogarty, M. Miars, An Integrated Simulation Environment for Parallel and Distributed System Prototyping, *Simulation*, Vol. 72, No. 5, 1999, pp. 283-294.
- [7] A. George, W. Phipps, R. Todd, D. Zirpoli, K. Justice, M. Giacoboni, M. Sarwar, SCI and the Scalable Cluster Architecture Latency-hiding Environment (SCALE) project, *Proc. 6th International SCI Workshop*, Santa Clara, California, Sep. 1996, pp. 9-24.
- [8] IEEE, 1596-1992 IEEE Standard for Scalable Coherent Interface (SCI), Piscataway, NJ: IEEE Service Center, 1993.

- [9] R. Lachenmaier, T. Stretch, A draft proposal for an SCI/RT protocol using directed flow control symbols, White Paper, IEEE P1596.6 Working Group, Nov. 1996.
- [10] C. Liu, J. Layland, Scheduling algorithms for multiprogramming in a hard-real-time environment, *Journal of the ACM*, Vol. 20, No. 1, 1973, pp. 46-61.
- [11] Y. Ofek, K. Sohraby, H. Wu, Integration of synchronous and asynchronous traffic on the MetaRing and its performance study, *IEEE/ACM Trans. on Networking*, Vol. 5, No. 1, 1997, pp. 111-121.
- [12] D. Picker, R. Fellman, An extension to the SCI flow control protocol for increased network efficiency, *IEEE/ACM Trans. on Networking*, Vol. 4, No. 1, 1996, pp. 71-85.
- [13] K. Shanmugan, V. Frost, W. LaRue, A Block-Oriented Network Simulator (BONeS), *Simulation*, Vol. 58, No. 2, 1992, pp. 83-94.
- [14] T. Scott, Full performance TRAIN protocol for SCI/RT, White Paper, IEEE P1596.6 Working Group, Sep. 1995.
- [15] R. Todd, A simulation case study of proposed real-time protocols based on the Scalable Coherent Interface, M.S. Thesis, Department of Electrical Engineering, Florida State University, 1996.
- [16] L. Yao, W. Zhao, C. Lim, Comparative study of three token ring protocols for real-time communications, *International Journal of Mini and Microcomputers*, Vol. 17, No. 2, 1995, pp. 85-97.