A New Scalable and Cost-Effective Congestion Management Strategy for Lossless Multistage Interconnection Networks*

J. Duato, I. Johnson, J. Flich, F. Naven, P. García, and T. Nachiondo

Tech. Univ. of Valencia, Spain
{jduato,jflich,tnachion}@disca.upv.es

Xyratex, United Kingdom
{Ian_Johnson,Finbar_Naven}@xyratex.com

Univ. of Castilla-La Mancha, Spain
pgarcia@info-ab.uclm.es

Abstract

In this paper, we propose a new congestion management strategy for lossless multistage interconnection networks that scales as network size and/or link bandwidth increase. Instead of eliminating congestion, our strategy avoids performance degradation beyond the saturation point by eliminating the HOL blocking produced by congestion trees. This is achieved in a scalable manner by using separate queues for congested flows. These are dynamically allocated only when congestion arises, and deallocated when congestion subsides. Performance evaluation results show that our strategy responds to congestion immediately and completely eliminates the performance degradation produced by HOL blocking while using only a small number of additional queues.

1. Introduction

Congestion management is one of the most critical and challenging problems interconnect designers face today. Without suitable congestion management, network throughput may degrade dramatically when the network or part of it reaches saturation. Also, packet latency may increase by several orders of magnitude. The problems produced by congestion are much simpler to solve in lossy networks such as the Internet because packets are dropped when congestion occurs, thus preventing the formation of congestion trees [4, 14]. However, dropping packets and the associated retransmissions may lead to some packets experiencing a very high latency, which may have a direct impact on application execution time in a parallel computer. As a consequence, most parallel computers and clusters are based on lossless interconnects (e.g., Cray T3E [26], IBM BlueGene/L [12], Myrinet 2000 [20], Quadrics [24], InfiniBand [13], etc.).

For lossless networks, congestion has much more dramatic consequences because congestion trees may grow very quickly (depending on congestion severity), exhausting network buffer resources and preventing other packets from making progress. A single hotspot may lead a large fraction of the network to total collapse unless some action is taken. The de facto solution to this problem has traditionally been to overdimension the interconnection network. Effectively, reported network utilization in parallel machines and clusters has been quite low for almost two decades.

This situation has recently changed. On the one hand, interconnect technologies are expensive compared to processors in current clusters. Fortunately, most current interconnects (Myrinet 2000, Quadrics, InfiniBand, PCI Express Advanced Switching (AS) [21, 22]) allow the customer to modify the cost of the network by using either a different number of processors per node or a different fraction of ports in the switches to attach processing nodes to them. Similar considerations are also valid for storage and system area networks (SAN). This allows the customer to reduce cost by reducing the number of communication components (i.e. switches and links) at the expense of increasing link utilization, thus driving the network as close to saturation as possible. On the other hand, power consumption is becoming increasingly important. As link speed increases, interconnects are consuming an increasing fraction of the total system power [27]. Moreover, power consumption in current high-speed links is almost independent of link utilization. Thus, the only ways to reduce network power consumption are: a) reducing the number of links in the network (and using the remaining links more efficiently), and b) using some frequency/voltage scaling technique [27]. Unfortunately, dynamic voltage scaling (DVS) techniques are quite inefficient due to their slow response in the presence of traffic variations and the suboptimal frequency/voltage settings during transitions [32].

Thus, cost and power consumption constraints lead to a higher utilization of communication components (i.e. switches and links), thus reducing component count. Unfortunately, as traffic is usually bursty, increasing link utilization will inevitably lead to network saturation during certain time intervals. Saturation usually produces dramatic performance degradation due to interference between different traffic flows. In particular, congestion leads to blocked packets that prevent the advance of other packets stored in the same queue. This happens even if those packets are not going to cross the congested area. This situation is referred to as head-of-line (HOL) blocking. Moreover, this situation quickly spreads through the network, forming a congestion tree. In this context, congestion management becomes crucial in order to achieve high network utilization without degrading performance when the network reaches saturation

* This work was supported by CICYT under Grant TIC2003-08154-C06 and by Junta de Com. de Castilla-La Mancha under Grant PBC-02-008.

Proceedings of the 11th Int'l Symposium on High-Performance Computer Architecture (HPCA-11 2005)
1530-0897/05 $20.00 © 2005 **IEEE**

during traffic peaks. We claim that if an efficient congestion management technique were available, it would be possible to dramatically reduce network cost by using interconnects with a smaller number of switches and links, and by running the network closer to its saturation point. Also, efficient congestion management would allow power saving techniques like the ones proposed in [32] to work more efficiently.

Many congestion management strategies as well as techniques to eliminate HOL blocking have been proposed, as will be described in Section 2. Unfortunately, these either become unstable or excessively slow when link bandwidth and/or the number of stages increase, or require a number of resources per link that grows linearly with network size. Thus, to the best of our knowledge, no scalable and cost-effective solution to the congestion management problem for lossless networks has been proposed so far.

In this paper, we take on this challenge. We propose a new strategy to eliminate the negative effects of congestion trees. This strategy is based on avoiding HOL blocking. We achieve this in a scalable manner by using separate queues for congested flows (i.e. packet flows contributing to congestion) that are dynamically allocated only when congestion appears, and deallocated when congestion subsides. This strategy is scalable because only congested flows need the allocation of additional queues (i.e. one additional queue per congestion tree that concurrently overlaps with other congestion trees at the corresponding switch port). We claim that uncongested flows (i.e. those packets not contributing to congestion) can be mixed in the same queue without any noticeable performance degradation due to HOL blocking. Performance evaluation results show that our strategy responds to congestion immediately and completely eliminates the performance degradation produced by HOL blocking while using only a very small number of queues per link. The rest of the paper is organized as follows. Section 2 describes related work. The new congestion management strategy is described in Section 3. Evaluation results are presented in Section 4, and finally, in Section 5, some conclusions are drawn.

2. Related Work

The congestion management problem in lossless networks has been studied for almost 20 years [23]. There exist hundreds of papers on this topic. Proposed solutions belong to one of three classes: proactive congestion management, reactive congestion management, and HOL blocking avoidance/reduction. Proactive congestion management is based on a priori knowledge of resource requirements, reserving them before starting data transmission. However, this knowledge is not always available. Besides, resource reservation incurs significant overhead. Thus, this kind of technique is not suitable for general purpose congestion management. Proactive congestion management techniques have been traditionally used to provide quality of service (QoS) support.

Reactive congestion management is based on measuring congestion, notifying the sources injecting traffic into the network, and reducing/stopping injection at those sources. There is a huge research body in this area comprising several subclasses. Some solutions detect congestion and broadcast notifications to all the sources [36, 34]. These solutions consume a large fraction of the bandwidth for notification, thus being inefficient. Some other solutions send notifications to only the sources contributing to congestion [16], thus being more efficient. However, none of these solutions scales well because as network size and/or link bandwidth increase, the effective delay between congestion detection and reaction (and the number of injected packets during that time) increases linearly. This leads to slow response (i.e. by the time congestion notification reaches the offending sources, a congestion tree has already formed) and/or the typical oscillations that arise in closed-loop control systems with delays in the feedback loop. Both of these translate into significant performance degradation. Moreover, these limitations are due to physical constraints in that notifications cannot propagate faster than the speed of light, hence being unavoidable. Thus, reactive congestion management alone may not be appropriate for future systems with long round trip times (RTT).

The only reactive congestion management strategies that are scalable are those that react locally by reducing/stopping injection at the endpoints attached to the switch where congestion is detected [7, 19, 2, 3]. Although those techniques have been reported to achieve good results with synthetic traffic where all the nodes have the same injection rate, they may fail under real traffic because sources contributing to congestion are usually different from those that are throttled.

A closely related problem that has traditionally been addressed separately is HOL blocking. It occurs when a packet in a queue could be forwarded but is blocked because the packet ahead of it is also blocked. This effect is so closely related to congestion trees, that if HOL blocking was completely eliminated, congestion trees would be of no harm. This happens because congestion trees create HOL blocking between packets belonging to congested flows and packets belonging to uncongested flows stored in the same queue. Thus, if HOL blocking could be eliminated then packets from uncongested flows would be able to advance.

HOL blocking has also been studied extensively, and very efficient techniques exist for avoiding HOL blocking within a single switch (e.g. virtual output queues (VOQs) [1], dynamically allocated multiqueues (DAMQs) [33], congestion buffers [29], etc.). These techniques work by allocating separate buffers for packets destined to different output ports or by providing a way for packets to pass blocked packets. Virtual channels [6] also reduce HOL blocking at the switch level but do not eliminate it. Again, these solutions either do not work efficiently for multihop networks (e.g. DAMQs, virtual channels) or are not scalable because the number of buffers required at every switch port to completely eliminate HOL blocking increases linearly with the number of endpoints attached to the network. Thus, overall buffer capacity

increases at least quadratically with the number of network endpoints. Although some implementations of network-level VOQs exist [5], they are very expensive and may become infeasible beyond certain network size. Another way of eliminating HOL blocking is by using a non-blocking topology [9]. Once again, this solution is expensive and not scalable.

To the best of our knowledge, the only strategy proposed so far that has relatively good scalability and eliminates HOL blocking for a limited number of congested endpoints is the one proposed in [15] for the ATLAS I. This strategy allocates separate buffers for each congested endpoint, thus eliminating the interference between packets going to those congested endpoints and the remaining packets. However, this strategy may consume a large number of buffers when hotspots arise within the network because packets crossing through a given hotspot may have many different destinations. As a consequence, scalability is significantly affected. The strategy proposed in this paper solves this problem by allocating a single buffer at each link for packets crossing a given hotspot, regardless of their final destination.

The strategy proposed in [18] for handling transient congestion also allocates separate buffers for each congested link within the network. However, this strategy is not scalable because it would require a separate buffer for every potential congested link downstream in the network. Indeed, this strategy has been designed to handle congestion at the next switch only. Yet another strategy that also allocates a separate buffer for congested packets has been proposed in [30]. However, this strategy has been proposed for a different goal—eliminating congestion cycles produced by adaptive routing—and cannot handle congestion trees.

Related solutions that contribute to alleviate or delay the problem are the use of fully adaptive routing [8, 17, 11, 35, 28] and load balancing techniques [10, 28]. These techniques delay saturation but do not eliminate performance degradation once saturation is reached. Finally, there exist many papers on congestion control in packet switching networks. See [37] for a taxonomy and pointers to other work.

3. A New Strategy for Congestion Management

This section describes the behavior of our congestion management strategy. It will be referred to as regional explicit congestion notification (RECN) because it detects and explicitly notifies congestion throughout the region that will be affected by a given congestion tree. It also eliminates HOL blocking in that region. RECN differs from local explicit congestion notification (LECN) strategies [18] recently proposed for PCI Express Advanced Switching in that it is able to eliminate HOL blocking across multiple stages.

RECN is valid for any network topology, including both direct networks (e.g. meshes and tori) and multistage interconnection networks (MINs). However, taking into account that all the commercially available interconnect technologies support some kind of MIN, we will restrict some aspects of our description to these topologies. The reason is that, when using these topologies, it is possible to implement some optimizations based on their self-routing properties. In particular, the path from a given switch to a hotspot within the network can be encoded as a sequence of bits that matches a subset of the destination address field in the header of packets crossing that path. This restriction is not necessary when using source routing (as is the case for IBM SP-2, Myrinet, and PCI Express Advanced Switching) because the destination address field in the packet header is always encoded as a sequence of turns in the network. Thus, it is obvious that the encoding for any subpath of a path will match a subset of the bits required to encode the entire path. Similarly, we will restrict our description to deterministic routing since this is the only kind of routing available in all the standard interconnect technologies currently available for clusters and SANs.

3.1. The Basic Idea

The existence of congestion at the delivery ports located at network destination endpoints is not a problem by itself. Simply, the network is working to deliver the maximum throughput accepted by the delivery ports. If the links leading to those ports are working at 100% of their bandwidth, nothing can be done to improve throughput unless link bandwidth is increased. Moreover, the aggregation of link-level flow control across the links in the congestion tree will collectively produce some backpressure that will automatically limit packet injection at the sources sending packets to the congested destinations. The real problem is that congestion trees fill the queues and produce HOL blocking, usually preventing other flows not destined to any of the congested ports from running at the maximum speed allowed by the network. This is true even for non-blocking topologies because congested and uncongested flows may have the same source. If HOL blocking was completely eliminated then congestion trees would be harmless. We followed this approach.

We observed through simulation that packets from non-congested flows do not produce any significant interference between each other if they are mixed in the same queue, provided that those packets are not blocked by any packet belonging to a congested flow. This observation has two important consequences. First, we can significantly reduce the number of queues required to effectively eliminate HOL blocking when compared to a VOQ mechanism by allowing all the packets from non-congested flows arriving at a switch input port to share the same queue.[1]It should be noted that the majority of queues in a VOQ system are empty most of the time due to spatial and temporal locality in packet destinations. Thus, storing all the packets from non-congested

1 This is not a strict requirement. Several queues can be used for non-congested flows, thus providing support for multiple traffic classes. However, for the sake of simplicity and in order to emphasize that one queue is enough, we will assume in this paper that only one queue is used for non-congested flows.

flows in the same queue significantly enhances queue utilization. Secondly, we simply need to implement some techniques to make sure that all the packets from congested flows are stored in separate queues, and therefore they are not going to interfere with uncongested flows. Moreover, they do not interfere even among themselves. This is important since different congested flows usually experience significantly different degrees of congestion. Thus, our proposal focuses on techniques to make sure that congested flows are quickly detected and immediately mapped to separate queues. The deallocation of those queues when they are no longer required is also important in order to minimize the number of resources needed to eliminate HOL blocking.

3.2. Switch Architecture

The strategy proposed in this paper is valid for any switch architecture. However, some mechanisms and implementation details may differ from one architecture to another. Taking into account current trends for high-speed interconnects, we selected a switch architecture based on an internal multiplexed crossbar with input and output buffers. Also, all the buffers for a given port are implemented using a high-speed data RAM and a separate control RAM to store the pointers. This implementation allows the use of dynamically allocated queues of variable size. This architecture matches current commercial switch design methods.

3.3. Congestion Detection

The mechanism for congestion detection should be simple and fast, and should accurately identify the sources that are contributing to congestion. A simple yet efficient congestion detection mechanism consists of detecting when the occupancy of a given output queue reaches a threshold. This technique works for both congestion at the endnodes of the network as well as internal network congestion. Now, the problem is determining the sources that are contributing to congestion. Reading the header of the packets stored in the queue to know their source is infeasible due to data RAM bandwidth consumption and the corresponding interference with packet transmission. Thus, we simply propagate notifications to the corresponding input ports the first time they send a packet to the congested output port. This way, only the input ports that are really contributing to congestion are notified. Each output port has a flag associated with each input port, which records the transmission of the corresponding notification, thus avoiding repeated congestion notifications.

3.4. Congestion Notification and Queue Allocation

In order to keep track of the switch ports belonging to a congestion tree, the switch output port that has detected congestion records the fact that it is the root of the congestion tree. Also, congestion notifications carry a token, identifying

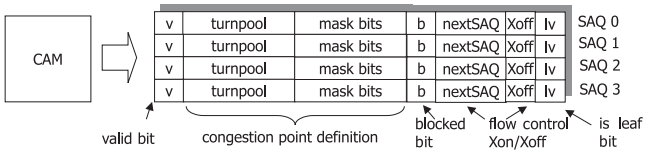


Figure 1. CAM structure.

the location of the leaves of the congestion tree. This will be useful when deallocating queues once congestion disappears. When congestion notifications are received at some input port, a set aside queue (SAQ) is locally allocated for the packets leading to the congested output port. All the incoming packets that request the congested output port will be stored in that SAQ, thus preventing them from blocking other packets stored in the main queue for that input port.

Two problems need to be solved in order to guarantee that packets will be correctly processed. The first one is ensuring that only the packets leading to a congested port will be stored in the corresponding SAQ. In order to achieve this, each SAQ has an associated routing information that indicates the path from the port where the SAQ is located to the port where congestion was detected. As we have assumed the use of deterministic routing, that path is unique. The encoding of such path depends on the topology and routing algorithm. Such encoding is not relevant for the behavior of RECN. However, as mentioned above, such encoding is straightforward in MINs and in networks using source routing. The path information is stored in a CAM. Each CAM line contains the path information associated with the corresponding SAQ. The reason for using a CAM is that the switch has to check the destination address field in the header of every incoming data packet and compare it against all the path informations stored in the CAM to determine whether this packet is going to cross the root of some congestion tree. In such a case, it will be stored in the corresponding SAQ. It should be noted that packets from uncongested flows that will use the same output port at the current switch as other packets from congested flows will not match any CAM line, and therefore, will not be stored in any SAQ, thus avoiding HOL blocking. Finally, it should be noted that two congestion trees may overlap. Moreover, one of them could be a subtree of the other tree. All of these cases are automatically solved when using a CAM to store path information and incoming notification packets are matched against it.

Figure 1 shows the CAM structure required to store path information in a switch for PCI Express AS. PCI Express AS implements source routing and uses a header field to encode the destination address. This field is referred to as *turnpool* and contains a sequence of turns at consecutive switches, where each turn is the relative address of the required output port with respect to the input port at the corresponding switch. In this case, encoding a subpath within the network is as easy as selecting the corresponding subset of consecu-

tive turns from the turnpool. This corresponds to selecting a subset of consecutive bits from the turnpool. The set of selected bits is stored in a bit mask associated with the path information in every CAM line.

The second problem is guaranteeing that all the packets will be delivered in order. This can be easily achieved if, after allocating a new SAQ, all the packets that should be stored in that SAQ and are currently stored in the main queue for that port are moved to the SAQ. Unfortunately, this is infeasible due to the bandwidth required to access the data RAM. Thus, we solved this problem in a more efficient manner by storing a marker in the main queue that points to the newly allocated SAQ and disabling that SAQ using the blocked bit in Figure 1 until the marker reaches the head of the queue. The details of this mechanism are described in Subsection 3.8.

Once a SAQ has been allocated at a given input port, it will store all the incoming packets leading to the congested port. If congestion persists, that SAQ will begin to fill. When SAQ occupancy reaches a threshold, a congestion notification will be propagated to the switch upstream. Again, a flag records this transmission to avoid repeated notifications that would consume link bandwidth unnecessarily. Also, a token is forwarded together with the notification. The receiving switch will store this token to record the fact that it is now a leaf of the congestion tree. Subsequently a new CAM line and the corresponding SAQ will be allocated at this output port. Again, when the SAQ reaches a threshold, it notifies congestion to all the input ports at the same switch that forward packets to that output port, recording such an action with a flag. Such a notification will indicate the path up to the congested port in the network. Thus, it is necessary to extend the path information received in the previous congestion notification with the turn corresponding to the current switch. From this point, if congestion persists, the process is repeated, propagating congestion notification and SAQ allocation to the upstream switches affected by the congestion tree. Congestion notifications may eventually reach the sources that inject the packet flows that produce the congestion. However, this may not always occur and it is not necessary for RECN to work. It should be noted that congestion notification and SAQ allocation are always done before SAQs are full. Hence, resources to prevent HOL blocking are always allocated before the congestion tree reaches the corresponding switch. Also, congestion notifications propagate faster when the thresholds are set to zero, but we want to avoid consuming resources unnecessarily. Thus, when using higher threshold values, transient congestion situations will not be notified beyond the leaves of the real congestion trees.

3.5. Queue Deallocation

SAQ and CAM line deallocation is simplified thanks to the use of tokens to tag the leaves of the congestion trees. The idea is to deallocate those resources when they are no longer used. Thus, when a given switch output port or endnode owns a token and the corresponding SAQ becomes empty, this SAQ and the corresponding CAM line are deallocated, notifying this situation to the downstream switch and sending the token to it. The input port of the downstream switch will proceed in a similar manner, notifying the corresponding output port, which is identified thanks to the path information available in the CAM line. The output port will first wait for the reception of tokens from all the tree branches. Then it will proceed in a similar manner. This process is repeated until the root of the congestion tree is reached.

3.6. Packet Processing

The only additional operation with respect to normal packet processing consists of identifying packets belonging to congested flows and storing them in the corresponding SAQ. This can be easily done thanks to the CAM associated with each group of SAQs. When a packet is received at an input port, its destination address field is compared with the path information in the CAM, searching it for the longest match. If there is a match, the packet will be stored in the corresponding SAQ. Otherwise, it will be stored in the queue for uncongested packets. In case of multiple matches, the longest match is used, thus storing the incoming packet in the SAQ for the largest congestion tree this packet is contributing to.

3.7. Flow Control

Flow control must be implemented for both standard queues and SAQs. This is necessary to avoid queue overflow and packet losses. Any efficient flow control mechanism can be used in combination with RECN. However, flow control operations may be constrained by communication standards and by the need to make SAQs transparent to the programmers. Most communication standards today implement credit-based flow control. This is also the case for our switch. It is important to note that credits are global, that is, they refer to all the space available for buffers in the data RAM for each port, including both standard queues and SAQs. Therefore, no packet is transmitted unless there is buffer space for it. Additionally, in order to prevent SAQs from dynamically growing excessively, we implemented Xon/Xoff flow control (also referred to as Stop & Go) for each individual SAQ. This mechanism was preferred over credits because congested flows tend to fill the SAQs and keep them over the threshold for Xoff for long periods. Therefore, Xon/Xoff leads to lower control traffic overhead.

3.8. Implementation Details

In addition to adhering to some standards, every electronic system design implies many tradeoffs. In particular, the main constraints when incorporating the congestion management strategy proposed in this paper to our switch design have been the finite amount of data RAM available for packet buffers and the limited bandwidth of this memory.

Many high-performance switches implement VOQ at the switch level. The technique proposed in this paper provides a viable alternative to VOQs. As a consequence, the data RAM used for implementing switch-level VOQ can now be reused to buffer congested flows. This saves space, allowing the use of a smaller data RAM and making room for the CAMs required for packet and notification processing. In our design, the replacement of a VOQ strategy allows us to implement 64 SAQs per port including the associated CAMs. The area for a 64-entry CAM and associated logic is approximately 2 mm^2 in 0.18μ technology.

Keeping in-order delivery is required by most current interconnect standards. When a new SAQ is allocated for congested flows, packets from that flow that are currently stored in the queue for uncongested flows should be moved to the newly allocated SAQ. However, this operation is prohibitive due to limited data RAM bandwidth even when queues are dynamically allocated and only the pointers to the packets in the control RAM need to be updated. The reason is that the data RAM has to be accessed to read the packet destination. Therefore, we follow a different approach. Instead of sweeping a queue for uncongested flows searching for packets for a certain destination, we disable the newly allocated SAQ and store a marker in the queue for uncongested flows. This marker contains the ID of the newly allocated SAQ. When that marker reaches the head of the queue for uncongested flows, the corresponding SAQ is enabled. By using this simple strategy, in-order delivery is guaranteed.

We observed from trace-driven simulations that SAQs did not always become empty once congestion vanishes. The reason is that when congestion disappears, the same sources that previously produced congestion continue injecting packets for the same destinations. Thus, although aggregated bandwidth demand is no longer enough to keep the congestion tree, the conditions for SAQ deallocation may not be met and SAQs continue to be used according to the above mentioned rules. In this case, we observed that SAQ occupancy is usually very low (one or two packets). Thus, in order to deallocate SAQs as soon as possible, we modify the priorities in the switch arbiter so that a SAQ storing only a few packets is assigned the highest priority if it owns a token.

As the number of implemented SAQs and CAM lines is obviously limited to some finite number, it may happen that there are no more SAQs available when a congestion notification is received. This can only happen at an input port. In this case, no SAQ is allocated and the token is returned to the notification sender. As a consequence, some amount of HOL blocking may be introduced. On the other hand, when a congestion notification is accepted at an output port and a new SAQ is allocated, an acknowledgment is returned to the downstream switch, also indicating the ID of the allocated CAM line. This ID is later used to reduce control traffic overhead and processing time for flow control packets by sending the ID instead of the corresponding subpath information.

4. Performance Evaluation

In this section we evaluate the RECN strategy. For this purpose we have developed a detailed event-driven simulator that allows us to model the network at the register transfer level. Firstly, we describe the main simulation parameters and the modeling considerations we have used in all the evaluations. Secondly, we present the evaluation results and analyze them.

4.1. Simulation Model

The simulator models a multistage interconnection network with switches, hosts, and links. To evaluate the proposed strategy, we have used three network configurations:

- 64 hosts connected using a 64×64 multistage network. This network consists of 48 switches with 8 bidirectional ports each, distributed in 3 stages.
- 256 hosts connected using a 256×256 multistage network with 256 8-port switches distributed in 4 stages.
- 512 hosts connected using a 512×512 multistage network with 640 8-port switches distributed in 5 stages.

In all the cases the perfect shuffle pattern has been used to interconnect the switches. Also, deterministic routing has been used. Memories have been modeled for both input and output ports of every switch. In this evaluation, we have used the same memory size for input and output ports: 128 KB (192 KB for the 640-switch network in order to support VOQ at the network level). Our simulator models several storage policies that require different memory configurations. When our congestion control methodology is used, the memory associated with a port is shared by all the queues (normal queues and SAQs) in such a way that memory cells are dynamically allocated to (or deallocated from) any queue when it is required. When VOQ policies are used, the total memory size per port is equally divided among all the established queues, resulting in a fixed maximum size for each queue.

At every switch, packets are forwarded from input queues to output queues through a multiplexed crossbar. We have considered a crossbar bandwidth of 12 Gbps. The access to the crossbar is controlled by an arbiter that receives requests from packets at the head of any input queue. A requesting packet is forwarded only when the corresponding crossbar input and crossbar output are free. Requests from packets in normal queues have preference over requests from packets in SAQs. At the output ports, another arbiter selects the output queue for injecting packets into the output link. This selection is made using a weighted round-robin scheme in such a way that normal queues have preference over SAQs.

To model the links, we have assumed serial full-duplex pipelined links. We have modeled links with 8 Gbps bandwidth. We have modeled different types of flow control. The RECN mechanism uses credit-based flow control at the port

level. So, whenever a new packet is transmitted from an output port to the corresponding input port of the next switch (or destination host), a credit is consumed. When a packet leaves an input port, a new credit is granted to the previous output port at the upstream switch (or host). Output port credits can be consumed for transmitting packets from any normal queue or SAQ at this port. A similar flow control scheme has been implemented for the internal switch (input-output) packet forwarding. So, the maximum number of credits per output (or input) port depends on the total memory size at the next input (or output) port. In addition, Xon/Xoff flow control has been modeled for limiting the transmission of packets between SAQs. When the occupancy of a SAQ grows up to a given threshold, an Xoff packet is sent to the corresponding upstream SAQ. Any SAQ that receives an Xoff packet stops the transmission of packets until the reception of an Xon packet. Any SAQ that previously sent an Xoff packet sends an Xon packet when its occupancy goes below another given threshold. Also, a credit-based flow control at the queue level has been implemented for the VOQ mechanisms. In these cases, the maximum number of credits per queue depends on the total memory size at the next input (or output) port, and the total number of queues at this port. Flow control packets have been modeled and they share the link bandwidth with data packets.

Hosts are connected to switches using network interfaces (NIs). Every NI is modeled as a fixed number of N message admittance queues following a VOQ scheme, and a variable number of injection queues, following the same scheme used at switch output ports. So, SAQs can be dynamically allocated at the output side of network interfaces when RECN is used. When a message is generated, it is stored completely in the admittance queue associated with its destination, and is packetized before being transferred to an injection queue. We have used 64 and 512-byte packets. The transfer from admittance queues to injection queues is controlled by an arbiter that follows a round-robin scheme. The injection of packets from injection queues to the network is also controlled by an arbiter that selects the next packet to be transmitted, using a round robin scheme among all the queues.

We have modeled in detail the RECN mechanism described in the previous section. Thus, all the control packets needed for congestion and end-of-congestion notifications (RECN messages, tokens, etc.) have been modeled and use the available link bandwidth in the same way as data and flow control packets. The simulator accepts different thresholds for detecting and propagating congestion as well as the number of maximum SAQs allowed at each port.

4.2. Traffic Load

In order to evaluate RECN, two different scenarios are analyzed for the 64×64 multistage network. First, well-defined synthetic traffic patterns are used to check whether RECN reacts appropriately to different congestion patterns. Each traffic pattern represents a different corner case. Table 1 shows the traffic parameters for the corner cases. For each corner case, there are 48 sources injecting traffic to random destinations during the entire simulation. These nodes inject traffic at the full link rate for corner case 2, and 50% of the full rate for corner case 1. In both cases, a congestion tree is formed from time 800 μs to time 970 μs.

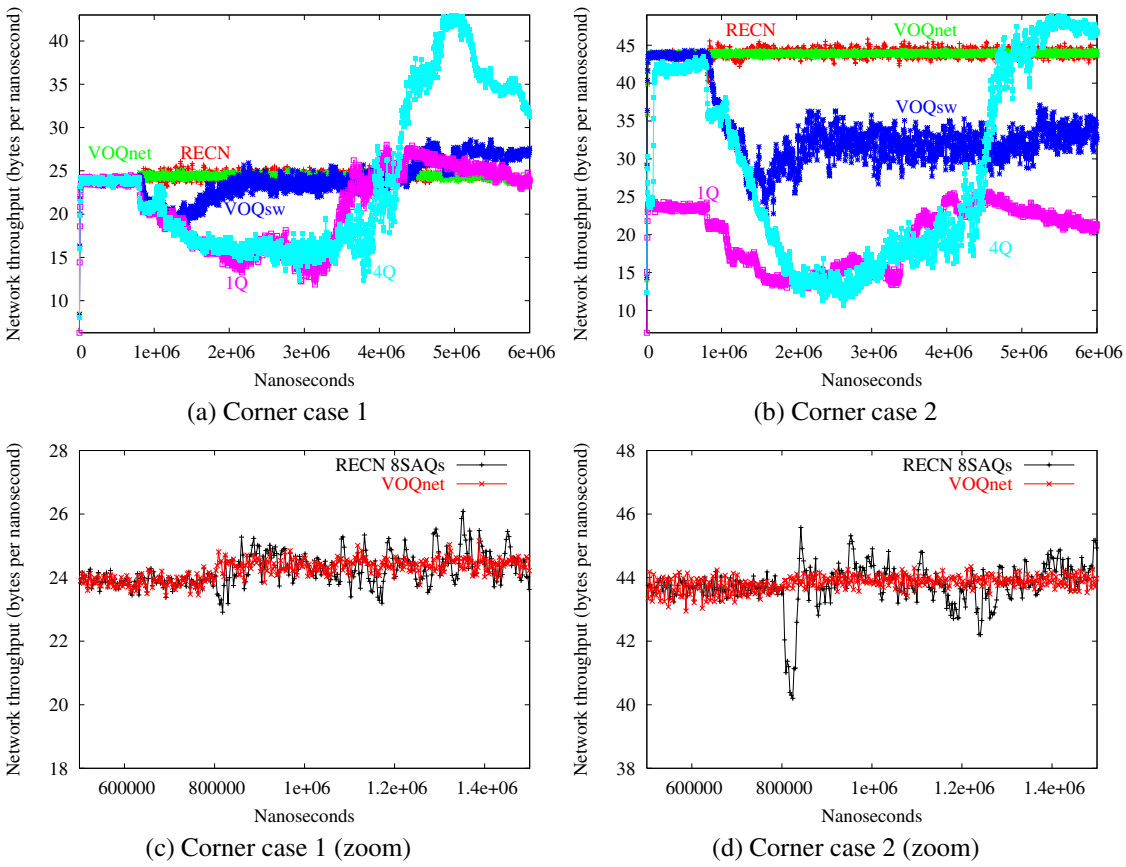
# Srcs.	Dst.	Injection Rate (%)	Traffic Start Time	Traffic End Time
Corner case 1				
48	Random	50%	0	Sim. end
16	32	100%	800 μs	970 μs
Corner case 2				
48	Random	100%	0	Sim. end
16	32	100%	800 μs	970 μs

Table 1. Traffic parameters for corner cases.

As a second scenario we use traces. In particular, the I/O traces used in our evaluations were provided by Hewlett-Packard Labs [31]. They include all the I/O activity generated from 1/14/1999 to 2/28/1999 at the disk interface of the *cello* system. They provide information both for the requests generated by the hosts and the replies generated by the disks. The *cello* system is a timesharing system with a storage subsystem consisting of twenty-three disks used by a group of researchers at HP Labs. A detailed description of similar traces, collected in the same system in 1992, can be found in [25]. However, the used traces are five years old. It seems reasonable to assume that nowadays I/O traffic has changed. In particular, the technology is quickly improving every year, allowing faster devices (hosts and storage devices) to be used, and thus generating higher injection rates. Thus, we have applied different time compression factors to the traces.

4.3. Performance Comparison

In this section we analyze how the RECN mechanism performs. In order to do that we have modeled five different mechanisms. The first one consists of using the VOQ mechanism at the network level. This mechanism uses as many queues as final destinations at every input and output port of every switch. It will be referred to as *VOQnet* and will be used to show the maximum achievable performance for each test. The second one consists of using just one queue on every input and output port. This mechanism will be referred to as *1Q* and will be used to show the worst achievable performance on each test. The third one consists of using the VOQ mechanism at the switch level. Thus, at every input port it uses as many queues as ports in the switch, and packets are mapped to queues according to the output port they will use at the current switch. This mechanism will be referred to as *VOQsw*. The fourth one will show the perfor-



(a) Corner case 1

(b) Corner case 2

(c) Corner case 1 (zoom)

(d) Corner case 2 (zoom)

Figure 2. Network throughput for corner cases, 64-byte packets.

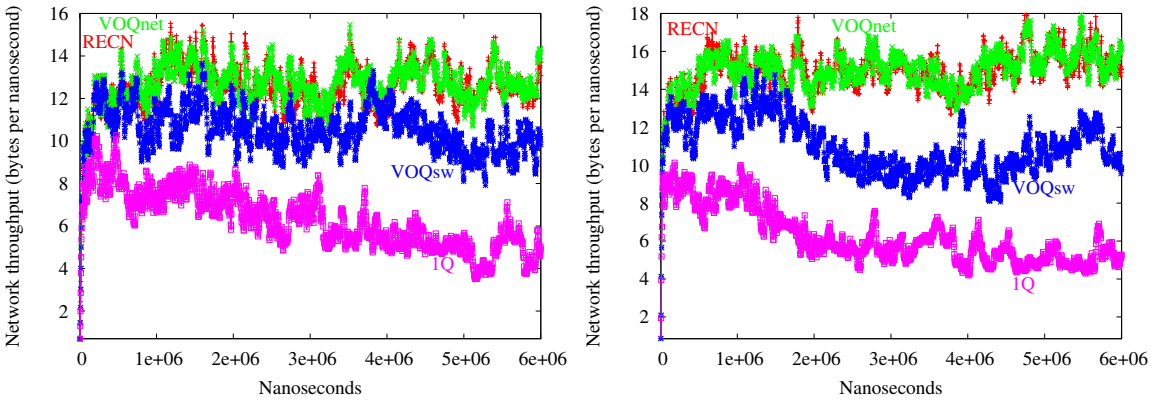
mance when implementing virtual channels and consists of using four queues at every input and output port. When using this mechanism (referred to as *4Q*), packets are stored in the queue with lower occupancy. Finally, we analyze the proposed RECN mechanism with 8 SAQs at every input and output port. It will be referred to as *RECN*.

Figures 2.a and 2.b show the performance of the different mechanisms for the corner cases when 64-byte packets are used. In this scenario, a congestion tree that lasts for 170 μs is formed. For the *VOQnet* mechanism, we can observe that there is no throughput degradation in the presence of the congestion tree. The network throughput is around 25 bytes/ns during all the simulation (45 bytes/ns for corner case 2). As the traffic flows to different destinations are separated, HOL blocking is avoided and maximum performance is achieved. It should be noted that, although traffic through the congested links is limited by the hotspot (i.e., the root of the congestion tree), sources continue generating traffic to other endnodes even though they cannot inject traffic to the hotspot.

On the other hand, with the *1Q* mechanism, the congestion tree introduces HOL blocking with non-congested packets, and therefore, performance decreases. In the first corner case, network throughput decreases from 25 bytes/ns to 12 bytes/ns. For the second corner case (Figure 2.b) the *1Q* mechanism is not able to efficiently handle all the random traffic even in the absence of the congestion tree, since it does not achieve the same network throughput achieved by *VOQnet*. Also, it suffers more HOL blocking, and thus, recovery time (once the congestion tree has disappeared) is longer.

Regarding the *4Q* mechanism, it also suffers from HOL blocking degradation. For both corner cases, network throughput decreases to the level achieved by the *1Q* mechanism when the congestion tree is formed, although for the second corner case the mechanism is able to handle the random traffic more efficiently than the *1Q* mechanism.

Focusing on the *VOQsw* mechanism we can observe that there is some HOL blocking that prevents the mechanism from achieving the maximum throughput. Although *VOQsw* eliminates the HOL blocking at the switch level, there exists some second-order HOL blocking that is not eliminated, and thus, throughput is affected. For corner case 1, network throughput slightly decreases from 25 bytes/ns to 20 bytes/ns. Although the system recovers faster than with *1Q*, it does not achieve the maximum performance until it is close to the end of the simulation. Moreover, for the second corner case, where all the sources are injecting at the full link



(a) Compression factor set to 20

(a) Compression factor set to 40

Figure 3. Network throughput for SAN traffic.

rate, we can observe that *VOQsw* experiences a high degree of second-order HOL blocking. Network throughput is decreased from 45 bytes/ns to 23 bytes/ns. Even more, the system does not recover before the simulation finishes.

Finally, and most important, we can observe from the figure that *RECN* works smoothly and is practically unaffected by the congestion tree. *RECN* manages to keep network throughput constant and almost identical to the throughput achieved by *VOQnet*. Figures 2.c and 2.d zoom in on the behavior of the *RECN* and the *VOQnet* mechanisms at the time when the congestion tree appears (in both scenarios). As can be observed, there is a slight oscillation for the *RECN* mechanism in corner case 1 around the throughput achieved by *VOQnet*. Network throughput is slightly decreased once the tree appears by less than 1 byte/ns. Also, this lower performance is quickly solved and it only lasts for less than 50 μs. In the case of the second corner case, we can see that the decrease in network throughput once the tree appears is higher. However, the decrease is lower than 4 bytes/ns and is quickly solved (again in less than 50 μs). This decrease in network throughput is due to the fact that the mechanism reacts once it detects some degree of congestion, so there is a small penalty in the HOL blocking introduced. However, by mapping SAQs for the detected congestion points, HOL blocking is removed in a fast and effective manner. Results for 512-byte packets (not shown) are quite similar to the 64-byte results. The behavior of the *RECN* mechanism is close to *VOQnet* mechanism, and *VOQsw*, *4Q* and *1Q* mechanisms suffer different degrees of HOL blocking degradation.

Figure 3 shows performance results for SAN traces for different compression factors. For both tests, *RECN* behaves quite similar to *VOQnet*, thus completely eliminating HOL blocking. For *VOQsw* we can observe that it again experiences some degree of second-order HOL blocking, and thus, it does not achieve maximum performance. Even more, for a compression factor of 40 the network throughput decreases from 16 bytes/ns to 10 bytes/ns.

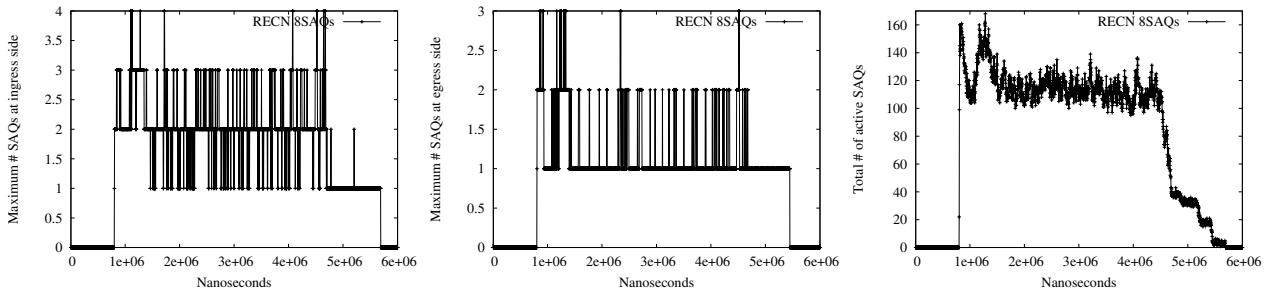
4.4. Scalability Analysis

In this section we analyze scalability issues of the RECN mechanism. Firstly we analyze the number of additional resources (SAQs) required by RECN. To do this, we show the number of SAQs required for the cases discussed in the previous section. Then, we analyze the number of SAQs required for larger networks.

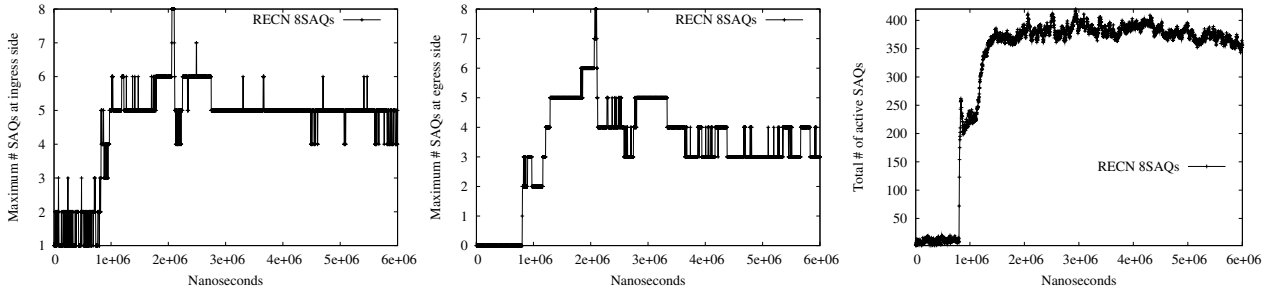
Figure 4 shows the required SAQs for the corner cases analyzed in the previous section, for 64-byte packets. The first figure shows the maximum number of SAQs required at the input (ingress) port with the highest demand of SAQs. The second one shows the same for the output (egress) ports of the switches. The third one shows the total number of SAQs required in the entire network.

As can be observed, for the first corner case the maximum number of SAQs is always lower than 5 and 4 at ingress and egress ports, respectively. Thus, with only eight SAQs, all the HOL blocking is removed. Indeed, the total number of SAQs is always lower than 170. Thus, on average, there are 0.22 SAQs mapped on every port in the network (there are 48 switches with 8 bidirectional ports each). For the second corner case, results are similar except for the ingress ports, where in some cases the 8 SAQs are required at a particular input port of a switch. Anyway, on average, the maximum SAQ utilization at ingress and egress ports is 5 and 4, respectively. On average there are 0.49 SAQs per port. Once again, results for 512-byte packets (not shown) are similar to the ones for 64-byte packets.

For the SAN traces, more SAQs are required. In particular, Figure 5 shows that in some cases up to 8 and 6 SAQs are required at ingress and egress ports, respectively, for a compression factor of 20 (8 and 8 SAQs for a compression factor of 40). However, total SAQ utilization is around 800 in the entire network for a compression factor of 20 (on average, one SAQ per port) and around 1400 for a compression factor of 40 (less than two SAQs per port on average).



(a) Corner case 1



(a) Corner case 2

Figure 4. SAQ utilization for corner cases, 64-byte packets.

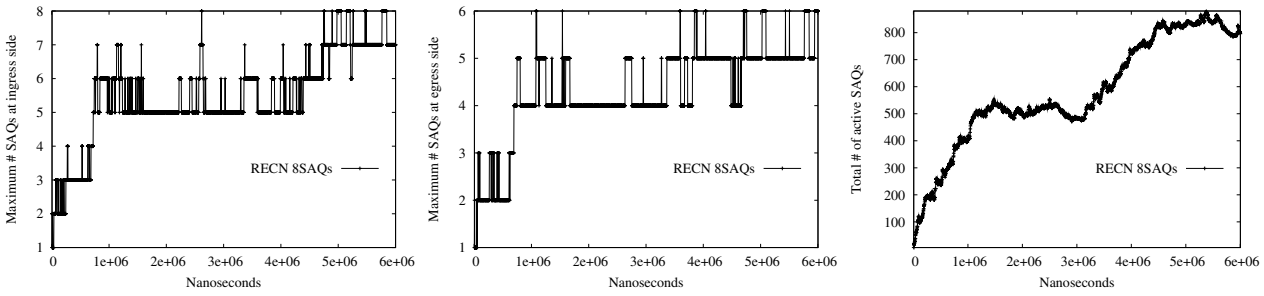
Now, let us turn our attention to larger networks. Figure 6.a shows the performance achieved in a 256×256 multistage network made of 256 8-port switches. The total number of sources is 256. This figure represents the throughput achieved when there are 192 sources sending random traffic at the full link rate and suddenly the remaining 64 sources start injecting to the same destination (hotspot destination) at the full link rate during 170 μs (corner case 2). As can be seen, the *RECN* mechanism keeps working efficiently and reaches the *VOQnet* level most of the time. There is a short transient period (less than 100 μs) where *RECN* suffers the HOL blocking introduced by the packets heading to the congested destination. Moreover, *RECN* achieves a good performance with a reduced number of SAQs. Thus, it scales very well. Figure 6 shows that *RECN* requires at most 8 SAQs. On the other hand, *VOQsw* does not scale at all, suffering substantial degradation throughout the entire congestion simulation time. In fact, the *VOQsw* performance does not significantly recover after the formation of the congestion tree.

Figure 6.b shows the performance achieved in a 512×512 multistage network made of 640 8-port switches. The total number of sources is 512. This figure represents the throughput achieved when there are 384 sources sending random traffic at the full link rate and suddenly the remaining 128 sources start injecting to the same destination (hotspot) at the full link rate during 170 μs (corner case 2). Again, the *RECN* mechanism keeps working very close to the *VOQnet* one. And again, it is achieved with a maximum of 8 SAQs. In general, the maximum number of SAQs only depends on the number of concurrent overlapping congestion trees, and not on the size of the network. Thus, an a priori analysis of traffic traces, if available, could provide a suitable upper bound on the number of required SAQs per port.

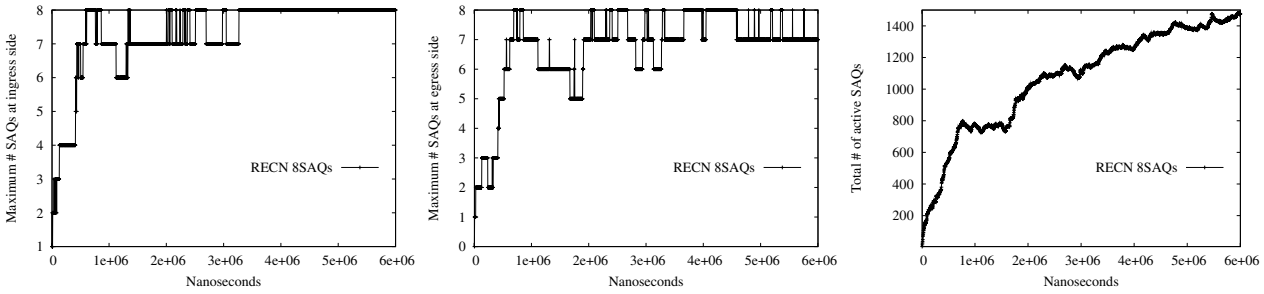
5. Conclusions

In this paper we have proposed RECN, a new strategy for congestion management in lossless interconnection networks, also describing the most relevant implementation details for multistage interconnection networks. To the best of our knowledge, this is the only congestion management strategy for lossless interconnects that scales with network size and link bandwidth without leading to prohibitive costs. RECN provides fast response because it acts locally instead of requiring traffic sources to react to congestion. Also, it achieves scalability because it only requires additional buffer resources for congested flows instead of requiring as many buffers as network destinations. Performance evaluation results with both synthetic traffic and traces clearly show that RECN significantly outperforms systems without support to avoid head-of-line (HOL) blocking as well as systems implementing either virtual channels or virtual output queuing (VOQ) at the switch level. Moreover, it achieves performance identical to that of VOQ at the fabric level, except for very short transient periods. This is a remarkable result since RECN completely eliminates HOL blocking at a small fraction of the cost of VOQ at the fabric level. We also presented an scalability analysis, showing that RECN scales very well with network size.

Proceedings of the 11th Int'l Symposium on High-Performance Computer Architecture (HPCA-11 2005)
1530-0897/05 $20.00 © 2005 **IEEE**

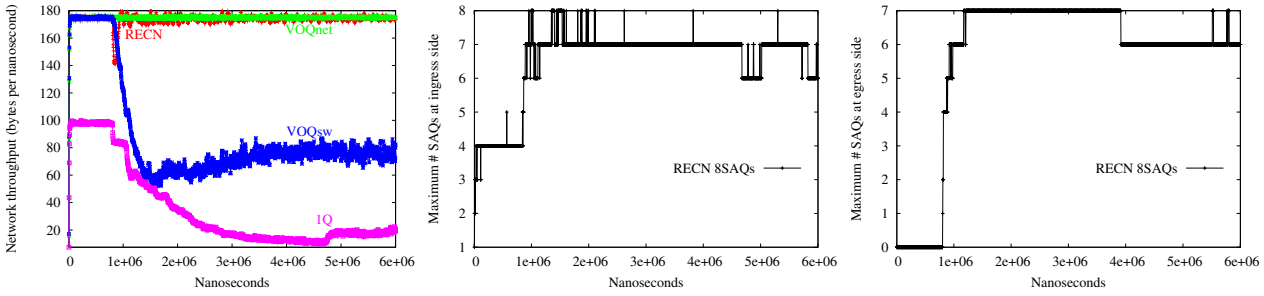


(a) Compression factor set to 20

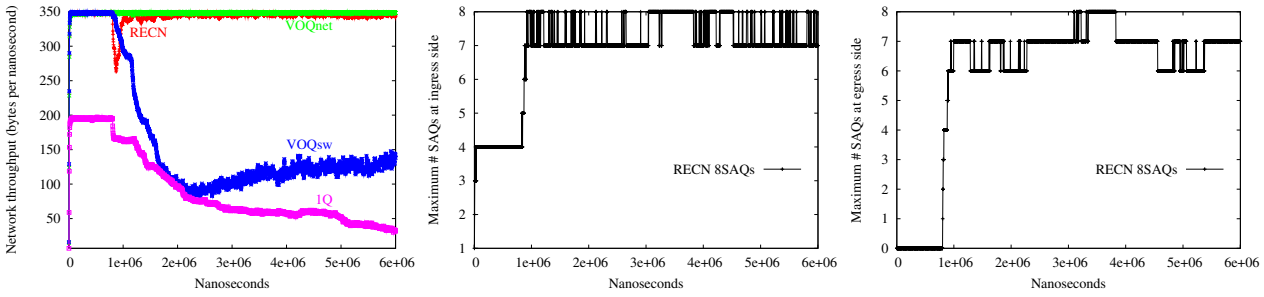


(b) Compression factor set to 40

Figure 5. SAQ utilization for SAN traffic.



(a) Network with 256 hosts



(b) Network with 512 hosts

Figure 6. Network throughput and SAQ utilization for larger networks.

Proceedings of the 11th Int'l Symposium on High-Performance Computer Architecture (HPCA-11 2005)
1530-0897/05 $20.00 © 2005 **IEEE**

References

[1] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High-Speed Switch Scheduling for Local-Area Networks", *ACM Trans. on Computer Systems*, vol. 11, no. 4, pp. 319–352, Nov. 1993.

[2] E. Baydal, P. López and J. Duato, "A Congestion Control Mechanism for Wormhole Networks", in *Proc. 9th. Euromicro Workshop Parallel & Distributed Processing*, pp. 19–26, Feb. 2001.

[3] E. Baydal and P. López, "A Robust Mechanism for Congestion Control: INC", in *Proc. 9th International Euro-Par Conference*, pp. 958–968, Aug. 2003.

[4] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End To End Congestion Avoidance on a Global Internet", *IEEE Journal on Selected Areas in Communication*, vol.13, no. 8, pp. 1465–1480, Oct. 1995.

[5] W. J. Dally, P. Carvey, and L. Dennison, "The Avici Terabit Switch/Router", in *Proc. Hot Interconnects 6*, Aug. 1998.

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

[7] W. J. Dally and H. Aoki, "Deadlock-Free Adaptive Routing in Multicomputer Networks Using Virtual Channels", *IEEE Trans. on Parallel and Distributed Systems*, vol. 4, no. 4, pp. 466–475, April 1993.

[8] J. Duato, "A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks", *IEEE Trans. on Parallel and Distributed Systems*, vol. 4, no. 12, pp. 1320–1331, Dec. 1993.

[9] J. Duato, S. Yalamanchili, and L. M. Ni, *Interconnection Networks: An Engineering Approach* (Revised printing), Morgan Kaufmann Publishers, 2003.

[10] D. Franco, I. Garces, and E. Luque, "A New Method to Make Communication Latency Uniform: Distributed Routing Balancing", in *Proc. ACM International Conference on Supercomputing* (ICS99), pp. 210–219, May 1999.

[11] P. T. Gaughan and S. Yalamanchili, "Adaptive Routing Protocols for Hypercube Interconnection Networks," *IEEE Computer*, vol. 26, no. 5, pp. 12–23, May 1993.

[12] IBM BG/L Team, "An Overview of BlueGene/L Supercomputer", in *Proc. ACM Supercomputing Conference*, Nov. 2002.

[13] InfiniBand Trade Association, "InfiniBand Architecture. Specification Volume 1. Release 1.0". Available at http://www.infinibandta.com/.

[14] R. Jain, "A Delay-Based Approach for Congestion Avoidance in Interconnected Heterogeneus Computer Networks", ACM Computer Communication Review, vol. 19, no. 5, pp. 56–71, Oct. 1989.

[15] M. Katevenis, D. Serpanos, E. Spyridakis, "Credit-Flow-Controlled ATM for MP Interconnection: the ATLAS I Single-Chip ATM Switch", in *Proc. 4th Int. Symp. on High-Performance Computer Architecture*, pp. 47–56, Feb. 1998.

[16] J. H. Kim, Z. Liu, and A. A. Chien, "Compressionless Routing: A Framework for Adaptive and Fault-Tolerant Routing", *IEEE Trans. on Parallel and Distributed Systems*, vol. 8, no. 3, 1997.

[17] S. Konstantinidou and L. Snyder, "Chaos Router: Architecture and Performance", in *Proc. 18th International Symposium on Computer Architecture*, pp. 79–88, June 1991.

[18] V. Krishnan and D. Mayhew, "A Localized Congestion Control Mechanism for PCI Express Advanced Switching Fabrics", in *Proc. 12th IEEE Symp. on Hot Interconnects*, Aug. 2004.

[19] P. López and J. Duato, "Deadlock-Free Adaptive Routing Algorithms for the 3D-Torus: Limitations and Solutions", in *Proc. Parallel Architectures and Languages Europe 93*, June 1993.

[20] Myrinet 2000 Series Networking. Available at http://www.cspi.com/multicomputer/products/2000_series_networking/2000_networking.htm.

[21] "Advanced Switching for the PCI Express Architecture". White paper. Available at http://www.intel.com/technology/pciexpress/devnet/AdvancedSwitching.pdf

[22] "Advanced Switching Core Architecture Specification". Available at http://www.asi-sig.org/specifications for ASI SIG.

[23] G. Pfister and A. Norton, "Hot Spot Contention and Combining in Multistage Interconnect Networks", *IEEE Trans. on Computers*, vol. C-34, pp. 943–948, Oct. 1985.

[24] Quadrics QsNet. Available at http://doc.quadrics.com

[25] C. Ruemmler and J. Wilkes, "Unix Disk Access Patterns", in *Proc. Winter Usenix Conference*, Jan 1993.

[26] S. L. Scott and G. Thorson, "The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus", in *Proc. Hot Interconnects IV*, Aug. 1996.

[27] L. Shang, L. S. Peh, and N. K. Jha, "Dynamic Voltage Scaling with Links for Power Optimization of Interconnection Networks", in *Proc. Int. Symp. on High-Performance Computer Architecture*, pp. 91–102, Feb. 2003.

[28] A. Singh, W. J. Dally, B. Towles, A. K. Gupta, "Globally Adaptive Load-Balanced Routing on Tori", *Computer Architecture Letters*, vol. 3, no. 1, pp. 6–9, July 2004.

[29] A. Smai and L. Thorelli, "Global Reactive Congestion Control in Multicomputer Networks", in *Proc. 5th Int. Conf. on High Performance Computing*, 1998.

[30] Y. H. Song and T. M. Pinkston, "A New Mechanism for Congestion and Deadlock Resolution", in *Proc. Int. Conf. on Parallel Processing*, pp. 81–90, Aug. 2002. Extended version in *IEEE Trans. on Parallel and Distributed Systems*. To appear.

[31] *SSP homepage*, http://ginger.hpl.hp.com/research/itc/csl/ssp/

[32] J. M. Stine and N. P. Carter, "Comparing Adaptive Routing and Dynamic Voltage Scaling for Link Power Reduction", *Computer Architecture Letters*, vol. 3, no. 1, pp. 14–17, July 2004.

[33] Y. Tamir and G. L. Frazier, "Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches", *IEEE Trans. on Computers*, vol. 41, no. 6, June 1992.

[34] M. Thottethodi, A. R. Lebeck, S. S. Mukherjee, "Self-Tuned Congestion Control for Multiprocessor Networks", in *Proc. Int. Symp. High-Performance Computer Architecture*, Feb. 2001.

[35] M. Thottethodi, A. R. Lebeck, S. S. Mukherjee, "BLAM: A High-Performance Routing Algorithm for Virtual Cut-Through Networks", in *Proc. Int. Parallel and Distributed Processing Symp.* (IPDPS), April 2003.

[36] W. Vogels et al, "Tree-Saturation Control in the AC3 Velocity Cluster Interconnect", in *Proc. 8th Conference on Hot Interconnects*, Aug. 2000.

[37] C. Q. Yang and A. V. S. Reddy, "A Taxonomy for Congestion Control Algorithms in Packet Switching Networks", *IEEE Network*, pp. 34–45, July/Aug. 1995.

1530-0897/05 $20.00 © 2005 **IEEE**