RECN-DD: A Memory-Efficient Congestion Management Technique for Advanced Switching *

P.J. García, F.J. Quiles
Dept. de Sistemas Informáticos
Univ. de Castilla-La Mancha
Albacete, Spain 02071
{pgarcia,paco}@info-ab.uclm.es

J. Flich, J. Duato
DISCA
Univ. Politécnica de Valencia
Valencia, Spain 46071
{jflich,jduato}@disca.upv.es

I. Johnson, F. Naven
Xyratex Ltd.
Havant, United Kingdom
Ian_Johnson@xyratex.com

Abstract

As VLSI technology advances, the interconnection network represents a larger percentage of the total system cost and power consumption. In fact, a current trend in network design is to reduce the number of components. However, this leads to systems working closer to saturation point, and therefore an efficient congestion management technique is required. In that sense, RECN has been recently proposed for Advanced Switching (AS). RECN detects the formation of congestion trees and dynamically allocates queues for storing congested packets, thus, eliminating the HOL blocking introduced by congestion trees. These queues are deallocated when congestion vanishes.

We have identified two shortcomings that may affect RECN scalability and implementation. Firstly, although RECN allocates queues in an efficient way, resource deallocation is performed in-order, thus losing efficiency and wasting resources. This leads to an excessive requirement of memory at switch ports. Secondly, both allocation and deallocation mechanisms involve the use of specific control packets not supported by the AS standard, thus preventing RECN implementation. In this sense we provide a detailed description of the current RECN deallocation mechanism.

In this paper we present an enhanced RECN version (RECN-DD) where these problems have been eliminated. Specifically, we propose a new distributed queue deallocation mechanism that reduces the number of required resources and does not require the use of control packets. Moreover, we propose a new congestion notification mechanism that does not require non-standard AS packets. Instead, flow control packets are used to notify congestion, thus simplifying the implementation of RECN-DD in AS.

*This work was supported by Spanish CICYT under Grant TIC2003-08154-C06, by UPV under Grant 20040937 and by Junta de Comunidades de Castilla-La Mancha under Grant PBC-05-005.

1. Introduction

As supercomputers and PC clusters increase in size, the interconnection network is becoming one of the critical system components. Most modern interconnection networks (Myrinet 2000 [18], Quadrics [20], InfiniBand [13], Advanced Switching (AS)[1] [1]) are quite expensive compared to endnodes in current clusters. Also, network power consumption is becoming increasingly important. As VLSI technology advances and link speed increases, interconnects consume an increasing fraction of the total system power [22]. One solution to reduce network cost and power consumption is by using fewer network components (i.e. switches and links), but this implies a higher utilization of such components. Unfortunately, as traffic is usually bursty, increasing link utilization will lead to network saturation and congestion during certain time intervals. In these cases, network performance degrades dramatically (congestion propagates quickly through the network due to flow control, forming congestion trees[2]), thus requiring an effective congestion management technique. In fact, congestion management is one of the most challenging problems interconnect designers face today.

In [8], a new congestion management mechanism, referred to as RECN (Regional Explicit Congestion Notification) was proposed. It differs from traditional solutions in that the congestion itself is not eliminated. Instead, RECN focuses on eliminating the head-of-line (HOL) blocking caused by congestion trees. This phenomenon occurs if a packet at the head of a FIFO queue is blocked (as it requests a busy output port), preventing other packets in that queue from advancing, even if they request free output ports.

Basically, RECN detects the formation of congestion trees and consequently allocates specific queues (referred

[1] AS is an open standard for fabric-interconnection technologies developed by the ASI SIG. It is based on PCI Express, extending it to include other features. AS is supported by many leading enterprises.

[2] We consider lossless networks, where packet dropping is not allowed.

Proceedings of the 2006 International Conference on Parallel Processing (ICPP'06)
0-7695-2636-5/06 $20.00 © 2006 IEEE

to as Set Aside Queues, SAQs) for storing packets belonging to those trees. Since non-congested packets are stored in other queues, the HOL blocking that congested packets could cause is avoided. SAQs are dynamically managed, and they can be deallocated when RECN detects congestion vanishment. RECN is implemented at every switch in the network, so it is able to react effectively and immediately during the formation of congestion trees. In [11] it was shown that the number of SAQs required for handling congestion is low and does not depend on network size. Therefore, RECN is scalable and cost-effective.

In this paper, we focus on how vanishment of congestion trees is managed in RECN. Specifically, we have detected that the SAQ deallocation procedure is not efficient. RECN assumes that congestion trees always collapse from leaves to root, and thus uses a conservative approach in order to deallocate SAQs. Specifically, tokens are exchanged between switches in order to identify the leaves of any detected congestion tree, and only SAQs at leaf points can be deallocated. Upon deallocation of a leaf SAQ, a token is sent to the downstream SAQ on the same tree branch. In [11], it was found that the way congestion trees evolve depends on switch architecture and traffic patterns, thus producing different congestion dynamics. In that paper we identified cases where congestion trees did not vanish from leaves to root. Thus, the deallocation mechanism will not follow, in many cases, the tree collapsing. This means that RECN may keep allocated many SAQs while the corresponding congestion tree has vanished, thus consuming an excessive number of queues.

Moreover, the exchanging of tokens implies the use of specific control packets. This is a handicap for the implementation of the whole RECN mechanism. Although RECN was proposed as a congestion management technique that could be used in most networks, it was applied to Advanced Switching (AS). However, as the current AS specification does not consider the explicit control packets required by RECN, the original RECN may not be suitable for implementation in AS. In this paper, as a first contribution, we propose a new fully distributed SAQ deallocation mechanism. It will deallocate SAQs independently of the location of the resource in the tree (thus closely following the real way congestion trees collapse). The new mechanism will not exchange tokens between switches, thus facilitating the implementation of the mechanism in AS.

On the other hand, RECN uses explicit notification packets between switches in order to keep track of congestion trees. As the control packets used for exchanging tokens, they are not supported currently in AS. In this paper, as a second contribution, we redefine the mechanism of notifying congestion between switches in order to conform to the AS standard. Specifically, the new notification mechanism will use flow control packets for notifying congestion.

The resulting version of RECN will be referred to as RECN Distributed Deallocation (RECN-DD). To sum up, the main benefits of this new version will be the following:

- It will minimize the number of resources required in order to deal with congestion trees. This will allow us to handle a higher number of concurrent trees with the same set of resources. Alternatively, the number of SAQs can be reduced while maintaining performance.
- Compatibility with AS will be facilitated, since explicit congestion notifications and token-exchanging packets will be eliminated.
- The SAQ deallocation policy will be simplified and thus it will be simpler to implement.

The rest of the paper is organized as follows. Section 2 shows an overview of the existing related work. Next, in Section 3, the basic RECN mechanism is described, focusing on the deallocation stage. Then, in Section 4, the new version of RECN (RECN-DD) is presented. In Section 5, RECN and RECN-DD are compared in terms of performance and resource needs. Section 6 shows an estimation of the memory area required at each port for implementing both mechanisms. Finally, in Section 7, some conclusions are drawn.

2. Related work

The problem of congestion on interconnection networks has attracted the attention of researchers for many years [19]. A large number of strategies have been proposed for controlling the formation of congestion trees and for eliminating or reducing their negative effects. Many of them consider congestion in multiprocessor systems, where congestion trees appear due to concurrent requests to the same memory module. In [6], a taxonomy of the different hotspot management strategies is proposed, dividing them into three categories: avoidance, prevention and detection strategies. Although different taxonomies are possible for other environments [29], the former classification is roughly valid also for non-multiprocessor-oriented congestion management techniques.

Avoidance strategies [30, 28] require previous planning in order to guarantee that congestion trees will not appear. In general, these strategies are related to quality of service requirements and are based on reserving network resources, thereby introducing a significant overhead.

Prevention strategies [12, 21] control the traffic in such a way that congestion trees should not happen. In general, decisions are made “on the fly”, based on limiting or modifying routes or memory accesses. However, it is required a knowledge of network status that is not always available.

Detection strategies are based on detecting congestion trees, and in consequence activating a control mechanism that solves the problem. For instance, congestion is detected by measuring switch buffer occupancy [27, 17] or the number of memory access requests [21]. Then, a notification is sent to the sources injecting traffic or to the processors requesting memory accesses, in order to cease or reduce their activity. Notifications can be sent to all the sources [26], just to those causing congestion [15] or just to the endpoints attached to the switch where congestion is detected [4, 2]. The main drawback of these strategies is the delay between congestion detection and reaction, which leads to slow response and so to performance degradation.

Other strategies like fully adaptive routing [7, 23] or load balancing techniques [10, 23] may delay congestion appearance, but they become useless once saturation is reached.

On the other hand, many techniques minimize or eliminate the main negative effect of congestion: the HOL blocking. One of them is the use of non-blocking topologies [9]. Unfortunately, it is not always possible to build such configurations, so other topology-independent HOL blocking elimination techniques exist. Some of them focus on HOL blocking formed at the switch level [3, 25, 16] while others at the entire network [5, 14]. In general, switch-level techniques, like Virtual Output Queues at switch level (VOQsw), are scalable but not fully effective, as HOL blocking may be produced outside the switch, while network-level techniques, like Virtual Output Queues at network level (VOQnet), are usually effective but not scalable, as they require too many queues per port.

Recently, RECN was proposed as a scalable and fully effective solution for HOL blocking elimination [8]. In [11], some enhancements to RECN were proposed, in order to correctly detect and isolate congestion trees regardless of the way they form. These changes were necessary due to the varied evolution of the trees under different conditions of traffic and switch architecture, also analyzed in [11]. Subsequently, that analysis also allowed us to find that the RECN deallocation mechanism may lead to an inefficient use of network resources. The first goal of the new RECN version presented in this paper is to solve this problem (the second one is to facilitate RECN compatibility with AS).

3. RECN

Although RECN could work under different network technologies, it benefits from the routing mechanisms found in AS. Specifically, AS uses source deterministic routing. The AS packet header includes a turnpool made up of 31 bits that contains all the turns (offset from the incoming port to the outgoing port) for every switch along the path. This allows a particular network point to be addressed from any other point in the network. Thus, a switch, by inspecting the appropriate turnpool bits of a packet, can know in advance if the packet will pass through a particular network point.

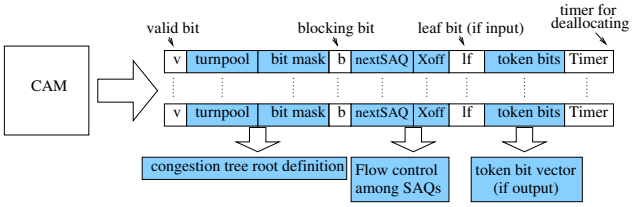


Figure 1. CAM structure for RECN.

3.1. Resources

RECN adds a set of SAQs to the standard (normal) queue at every input and output port of a switch. While all the non-congested packets are stored in the normal queue (RECN assumes that this does not introduce significant HOL blocking), SAQs are dynamically allocated to store packets passing through a specific congested point (root of a congestion tree). Each set of SAQs is managed by means of a CAM (Content Addressable Memory). Figure 1 shows the CAM structure. Each CAM line is associated to a specific SAQ, and includes the definition of the root for the corresponding congestion tree and other control information. The root position is defined by a network subpath encoded by means of a complete turnpool and an associated bit mask that selects the appropriate subset of consecutive bits from the turnpool. The rest of the control information will be described in the following sections.

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.

3.2. Congestion detection and notification

RECN detects congestion at both sides of a switch (input and output). At output ports, when a normal queue receives a packet and spills over a given threshold (detection threshold), a notification is sent to the sender input port indicating that the output port is congested. This notification includes the routing information (a turnpool and the corresponding bit mask) to reach the congested output port from the notified input port (only a turn). Upon reception of a notification, the input port allocates a new SAQ (setting the valid bit) and fills the corresponding CAM line with the received turnpool and bit mask. From that moment, every incoming packet that will pass through the congested point (easily detected from the packet turnpool) will be stored in the newly allocated SAQ, thus eliminating the HOL blocking it could cause.

At input ports a different detection mechanism is used. When a normal input queue spills over a threshold, it is because packets requesting a certain output port are being blocked, but, as these packets can head toward different output ports, it is not trivial to determine which one is the congested output port. In response to this requirement, RECN replaces the normal queue at each input port by a set of small buffers, referred to as detection queues. The detection queues are structured at the switch level: there are as many detection queues as output ports in the switch, and packets heading toward a particular output port are stored in the associated detection queue. By doing this, when a detection queue spills over a given threshold, congestion is detected, and the output port causing the congestion is easily computed as the port associated with that detection queue. Once congestion is detected at an input port, a new SAQ is allocated at this port, and the turnpool identifying the output port causing congestion is computed and stored in the CAM line. The congested detection queue and the allocated SAQ are swapped and a notification is sent upstream.

Congestion information propagates to other switches whenever an input SAQ becomes congested. Then, a new notification is sent upstream to some output port that will react allocating an output SAQ. In the same way, if an output SAQ reaches a given threshold, an internal notification is sent to the corresponding input port, which will react allocating an input SAQ, and so on. As notifications go upstream, the information encoding the route to the congested point is updated accordingly. So, congestion detection is propagated through all the branches of the tree and SAQs are allocated along the branches.

In order to guarantee that packets are delivered in order, when a SAQ is allocated, a marker (link pointer) is stored in the normal queue, pointing to the newly allocated SAQ. This SAQ will not forward packets until the marker reaches the head of the normal queue. The SAQ blocking is implemented by means of the CAM blocking bit (Figure 1).

3.3. Flow control

RECN implements a special Xon/Xoff flow control for SAQs. This mechanism is different from the credit-based flow control used for normal queues, which considers all the unused space of the port data memory available for each individual queue. If this "global-credits" scheme were used for SAQs, a congested flow could fill the whole port memory. Instead, the Xon/Xoff scheme guarantees that the number of packets in a SAQ will be always below a certain threshold. The CAM line associated to each SAQ contains the identifier (field *nextSAQ*) of the immediate downstream SAQ storing packets for the same congestion tree. Whenever the occupancy of a SAQ reaches the Xoff threshold, a Xoff flow control packet containing the SAQ identifier will be sent upstream. Upon reception of a Xoff packet, the SAQ whose *nextSAQ* field matches the received SAQ identifier must be set at Xoff state, and consequently the Xoff bit of the associated CAM line will be activated. Of course, a SAQ at Xoff state will not forward packets. The Xoff bit will be deactivated if an analogous Xon control packet is received.

It should be noted that detection thresholds and flow control thresholds (Xon and Xoff) may differ in RECN, and most importantly, they generate different control packets between switches. As we will see later, in RECN-DD, the detection thresholds and the Xoff threshold will be the same, and Xoff flow control packets will be used also to notify the detection of congestion.

3.4. SAQ deallocation mechanism

The RECN SAQ deallocation mechanism is based on controlling which SAQs are placed at the leaves of a congestion tree. For an input SAQ, this is indicated by a leaf bit on the associated CAM line. When an input SAQ is allocated, the associated leaf bit is activated (so, the SAQ "owns a leaf token"), and it is deactivated when the SAQ sends a congestion notification (so, the SAQ "sends its leaf token"). On the other hand, for an output SAQ, there is a list of token bits in the associated CAM line, one for each input port of the switch. When an output SAQ is allocated, all the token bits in the associated CAM line are activated. Whenever an output SAQ sends an internal congestion notification to any input port, the corresponding token bit is deactivated. Thus, an output SAQ is considered a leaf only if all its token bits are activated (so, the output SAQ "owns all its tokens").

In order to filter transient states, each SAQ has a timer (started when the SAQ is allocated). A SAQ is deallocated only when the following three conditions are met at the same time: the SAQ is empty, it is at a leaf, and its associated timer has expired. Upon deallocation of a SAQ, a notification containing a token and the *nextSAQ* value (from the associated CAM line) is sent downstream. If the deallocated SAQ is at an output port, then the notification is sent downstream to the next switch. In this case, the token will be accepted by an input SAQ (the one whose identifier matches the received *nextSAQ* value), and its corresponding leaf bit will be activated. In the case of deallocating an input SAQ, an internal notification is sent to the associated output port. In this case, an output SAQ will accept the token, and the corresponding token bit in the CAM line will be activated. The deallocation process is repeated in this way until the root of the congestion tree is reached, so it is always performed in order, from leaves to root.

Figure 2 shows an example of the RECN deallocation mechanism. Figure 2.a depicts a tree that is formed by three sources (nodes 0, 1 and 2) injecting packets at 50%, 25%,

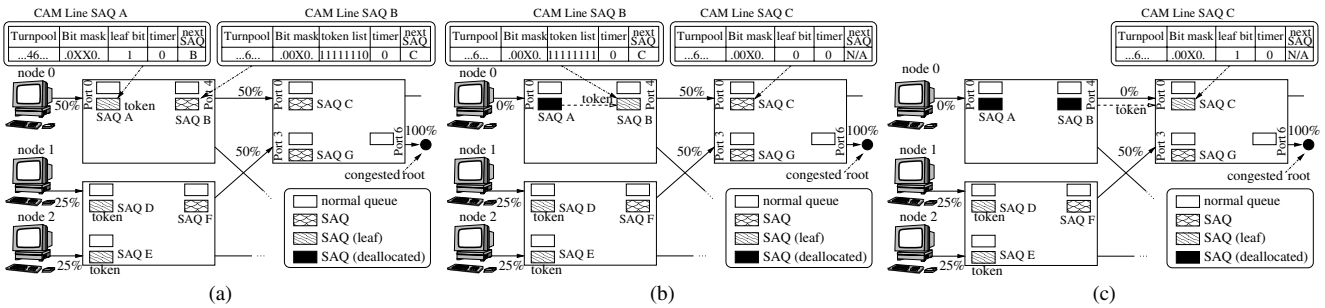


Figure 2. SAQ deallocation in RECN (vanishment of a tree branch). N/A means Not Applicable.

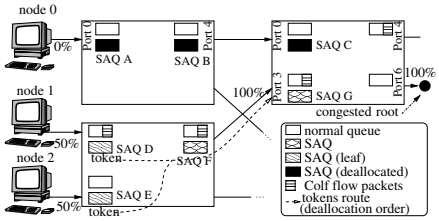


Figure 3. Collapsing of the rest of the tree.

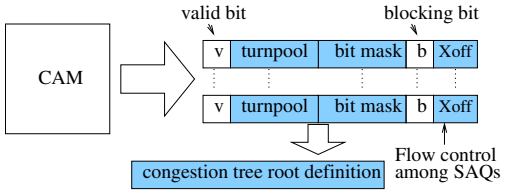


Figure 4. CAM structure for RECN-DD.

and 25% of the link injection rate to the same network point (the root of the tree). As can be seen, SAQs have been allocated along the tree branches for storing packets belonging to the congestion tree. Note that leaf tokens are owned by SAQs allocated at the input ports connected to the sources. Suppose now that the upper branch collapses, as node 0 stops injecting packets toward the congested point. Sooner or later, SAQ A will become empty, and (if its timer has expired), it will be deallocated as it owns the token. Once this happens (Figure 2.b), the token is sent to SAQ B. The receiving SAQ can be deallocated (if it is empty and its timer has expired) as the token list on its CAM line indicates now that it owns all the tokens. Once SAQ B is deallocated (Figure 2.c), the token is sent to the corresponding input SAQ at the downstream switch (SAQ C). Upon token reception, the CAM line of SAQ C will activate the leaf token, allowing the deallocation of that SAQ.

However, the token-based mechanism explained above presents important drawbacks. In the former example, the SAQs of the upper branch become empty from the leaf toward the root, and they are deallocated in the same order. But note that, once the upper branch disappears, the rest of the tree may collapse in a different way (even if sources keep the injection rate). For instance, as is depicted in Figure 3, if congestion tree packets share the link between switches with a cold flow, they will be consumed at the root of the tree faster than they are injected to SAQ G (assuming some crossbar speedup). Therefore, SAQ G will become empty before SAQ F, so the tree colapses from the root. As tokens are always passed downstream (in Figure 3, from SAQs D and E to SAQ F, later to SAQ G), empty SAQs near the root will remain allocated until the leaf SAQs become empty and start sending tokens. Note that those empty SAQs could have been used meanwhile to store packets from another congestion tree. Therefore, during congestion trees collapse, RECN may use more SAQs than required.

Another drawback of this deallocation mechanism is the use of specific control packets for sending tokens. Note that other RECN features (for instance, congestion notification) also require specific packets. This makes the implementation of RECN difficult because, in general, interconnect standards restrict the number and format of special packets.

With the aim of solving the drawbacks explained above, we propose a new RECN version in the following section.

4. RECN-DD

RECN-DD improves the previous RECN version in several ways. Firstly, SAQs can be deallocated independently of other SAQs allocated for the same congestion tree. Secondly, explicit congestion notification packets are no longer needed as congestion notifications are now attached to flow control packets. Both changes lead to lower requirements of queues and control information. Figure 4 shows the CAM structure required by RECN-DD. As can be seen, the timer, leaf bit, token list, and *nextSAQ* fields are no longer needed, thus producing a great saving of control memory.

4.1. Congestion detection and notification

With RECN-DD, congestion detection is made in the same way as in the original RECN. However, the propa-

gation of congestion detection differs significantly, as congestion notifications are now associated with the Xon/Xoff flow control. Moreover, SAQs for the same congestion tree are not linked (there is no *nextSAQ* field in the CAM).

For the case of Xon/Xoff flow control between two switches, whenever the occupancy of the input SAQ reaches the Xoff threshold, an Xoff control packet will be sent upstream. This packet includes the turnpool and the bit mask associated to the SAQ. The upstream output port will compare the received turnpool and bit mask to those associated to all the allocated SAQs at the output port. When matching, the corresponding SAQ will be set at Xoff state (an Xoff control packet has been received), activating the corresponding Xoff bit. However, when failing to match, the switch will consider the Xoff control packet as a congestion notification. Thus, a new SAQ will be allocated for the received turnpool and bit mask. Furthermore, the new allocated SAQ will be set at Xoff state.

For the case of Xon/Xoff flow control within a switch, whenever the occupancy of a SAQ at the output side reaches or is beyond the Xoff threshold, an internal Xoff notification is sent to all the input ports of the switch. This notification contains the turnpool and the bit mask from the associated CAM line. The Xoff notification also includes the ID of the input port that sent the last packet to the SAQ. On every input port, upon reception of the internal Xoff notification, the turnpool and bit mask are compared to those associated to all the allocated SAQs. When matching, the corresponding SAQ is set at Xoff state (an internal Xoff notification has been received). When failing to match, the input port checks if its ID port corresponds with the ID included in the internal notification. If so, the input port must allocate a new SAQ for the congestion tree. Additionally, the newly allocated SAQ will be set at Xoff state.

Note that Xon control packets (and Xon internal notifications) also contain a turnpool and a bit mask. This is needed in order to identify which SAQ must be set at Xon state.

4.2. SAQ deallocation mechanism

With RECN-DD, SAQs can be deallocated in a distributed way. Thus, the basic deallocation requirement is that the SAQ is empty. However, a new safety condition must be met in order to deallocate SAQs properly: the SAQ is neither blocked due to the blocking bit nor at Xoff state. Note that, since a SAQ is at Xoff state at the moment it is allocated, this condition avoids an empty SAQ being deallocated just after its allocation (premature deallocation). Thus, the timer is no longer needed. Moreover, as the SAQ must be at Xon state to be deallocated, SAQs are not deallocated while congestion is near (a SAQ at Xoff state means that the downstream SAQ is full of packets). Note that all of these conditions can be evaluated completely from local information, and all of them are independent of the state of other SAQs, with the only exception of flow control.

Notice also that the external data required for evaluating the new conditions are just those contained in the Xon/Xoff flow control packets. Thus, RECN-DD does not require specific RECN messages (detection notifications, tokens, etc.). As the Xon/Xoff flow control is considered in the AS specification, RECN-DD does not require any special control message that is not compatible with AS.

Figure 5 shows an example of queue deallocation with RECN-DD. It shows the CAM lines of SAQs assigned along a branch of a congestion tree, before and after a SAQ deallocation. In this case, it can be observed that CAM lines are independent from each other. Note also that, as SAQs near the root (for instance, SAQ E) do not have to wait for a token, they can be deallocated earlier. Thus, RECN-DD has more free SAQs for handling other congestion trees and, as we will show in the next section, this will allow RECN-DD to achieve maximum performance with far fewer SAQs.

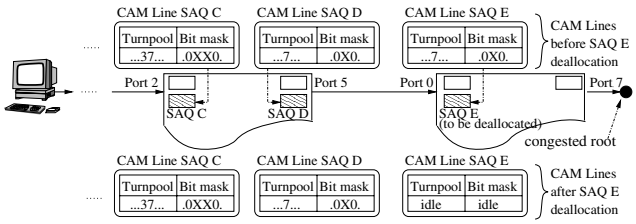


Figure 5. SAQ deallocation in RECN-DD.

5. Performance evaluation

In this Section we will compare RECN and RECN-DD performance. The main goal of this Section is to evaluate if the RECN-DD mechanism is able to keep the same level of performance as RECN while using fewer SAQs. Therefore, we are interested, firstly, in evaluating the performance achieved by both methods with a different number of SAQ queues. Furthermore, we will analyze for each evaluated case the real number of SAQs required by each method.

We will use different scenarios of traffic load and network size. For this purpose we have used a detailed event-driven simulator that models the network at the register transfer level. Firstly, we will describe the modeling assumptions and the main parameters used in the simulations. Secondly, we will analyze the obtained results.

5.1. Simulation model

The simulator models a Bidirectional Multistage Interconnection Network (BMIN) with switches, endnodes, and links. We will evaluate RECN and RECN-DD using BMINs of two sizes, specifically a 64 $\times$ 64 BMIN (48 switches)

		random traffic	congestion tree (dest 32)	
		75% sources	25% sources	period
case	network	injection rate	injection rate	time (μs)
#1	64×64	100%	100%	800 - 1100
#2	64×64	incremental	incremental	0 - end
#3	512×512	100%	100%	800 - 1100

Table 1. Traffic configurations evaluated.

and a 512 $\times$512 BMIN (640 switches). Both BMINs are built following the perfect shuffle interconnection pattern, with 8-port switches. At switches, packets are forwarded from input queues to output queues through a multiplexed crossbar, modeled with a speedup of 1.5 (link bandwidth is 8Gbps, crossbar bandwidth is 12Gbps).

In all the experiments deterministic routing has been used. Memories of 32KB have been modeled for both input and output switch ports. Each port memory is shared by the queues (normal or detection queues and SAQs) defined at this port at a given time, in such a way that memory cells are dynamically allocated (or deallocated) for any queue when necessary. A set of SAQs has been defined at input and output ports. The number of SAQs per set has been varied in the different experiments. At output ports, only one normal queue has been defined, whereas at input ports, the normal queue has been divided into 8 detection queues.

Endnodes are connected to switches using Input Adapters (IAs). Every IA is modeled with a fixed number of N admittance queues (where N is the total number of endnodes), and a variable number of injection queues, which follow a scheme similar to that of the output ports of a switch. When a message is generated, it is stored in the admittance queue assigned to its destination, and is packetized before being transferred to an injection queue. We have used 64-byte packets.

We have modeled in detail the different versions of RECN with their corresponding deallocation strategies: the original "leaves-to-root" deallocation strategy (RECN) and the new, distributed deallocation strategy (RECN-DD). Additionally, in order to compare the results of both versions against those of former proposed techniques, the simulator also allows experiments to be performed using Virtual Output Queues at switch or network level, or Virtual Channels.

5.2. Traffic load

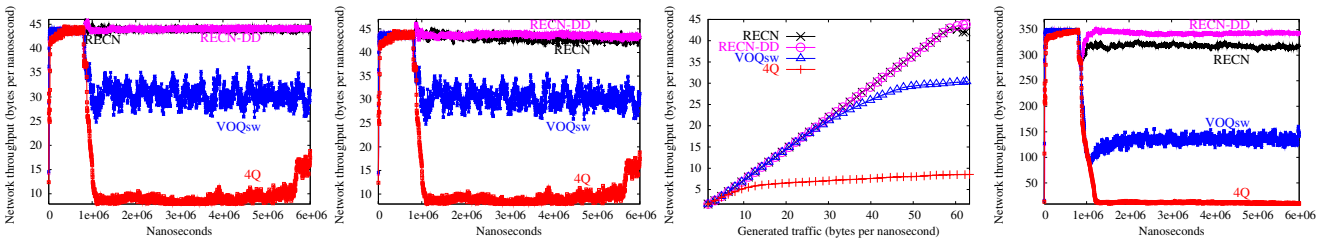
In order to evaluate the different RECN versions, we have used two different traffic scenarios. First, synthetic traffic patterns modeling simple but significant traffic situations have been used. Table 1 shows the traffic parameters of each traffic case. For all the traffic cases, there are 75% of sources injecting traffic to random destinations throughout the entire simulation period. These nodes inject traffic at different link rates depending on the traffic case. This rate is constant (100%) for traffic cases #1 and #3, but for case #2 it has been varied in an incremental way in order to obtain a metric of the network performance under different loads of normal and congested traffic. In all the cases, the remaining sources (25%) inject traffic to a hot-spot destination, thus forming congestion trees. The injection rate of the hot-spot sources is the same as that of the "random" sources for the same traffic case, so it is 100% for traffic cases #1 and #3 and incremental for traffic case #2. If incremental injection rates are used, congestion sources inject packets throughout the entire simulation time. If constant injection rates are used, congestion sources start to inject 800 μs after the simulation begins, finishing 300 μs later.

As a second scenario we have used traces. Specifically, the I/O traces used in our evaluations were provided by Hewlett-Packard Labs [24]. They include all the I/O activity generated from 1/14/1999 to 2/28/1999 at the disk interface of the *cello* system. As these traces are seven years old, we have applied a time compression factor to the traces.

5.3. Evaluation results

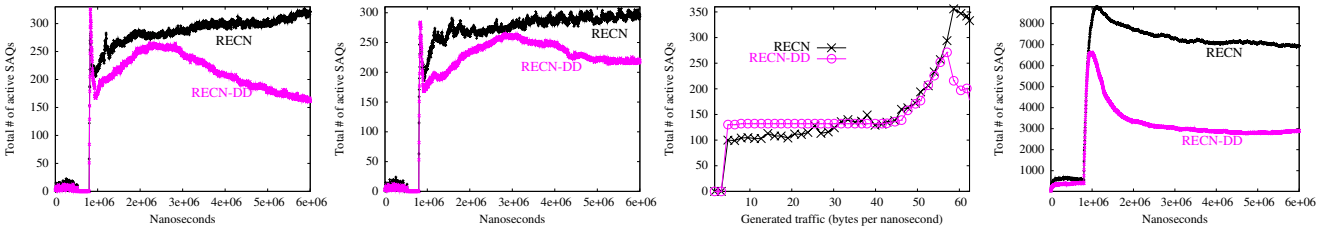
In this section we present results that allow us to compare the behavior of the network when the different RECN versions are used. For each traffic case and RECN version, we show the network throughput achieved, the total number of SAQs used in the network and the maximum number of SAQs used at an input port (SAQ utilization at output ports is always lower than at input side). Also, shown is the maximum bandwidth (as a percentage of total link bandwidth) required for control packets in the network. It has been computed as the bandwidth consumed by control packets in the link with the highest demand for sending such packets. Flow control packets for both RECN and RECN-DD are regarded as control packets as well as explicit congestion notifications for RECN. Congestion notifications on flow control packets have been considered for RECN-DD. This allows us to evaluate the overhead of both methods on link bandwidth. For traffic cases with constant injection rate, results are shown as a function of time. If incremental injection rates are used, results are shown as a function of generated traffic. We also show throughput results for switch-level Virtual Output Queues (VOQsw) and for 4 Virtual Channels (4Q) with a lowest-occupancy storage policy.

Figure 6.a shows throughput results for traffic case #1 (64×64 BMIN) when a maximum of 8 SAQs are allowed at each input or output switch port. As can be seen, both RECN and RECN-DD are able to handle the congestion tree in a very efficient way, as network throughput is kept almost constant, regardless of the presence of the congestion tree. By contrast, VOQsw and 4Q results show that these strategies do not handle congestion trees properly. So, both versions of RECN virtually eliminate the HOL block-



(a) 64×64 BMIN traffic 1, 8 SAQs (b) 64×64 BMIN traffic 1, 2 SAQs (c) 64×64 BMIN traffic 2, 2 SAQs (d) 512×512 BMIN traffic 3, 4 SAQs

Figure 6. Throughput results.



(a) 64×64 BMIN traffic 1, 8 SAQs (b) 64×64 BMIN traffic 1, 2 SAQs (c) 64×64 BMIN traffic 2, 2 SAQs (d) 512×512 BMIN traffic 3, 4 SAQs

Figure 7. SAQ utilization results.

ing in the network. However, Figure 7.a (total SAQ utilization for that configuration) shows that RECN-DD achieves maximum performance by consuming far fewer SAQs than RECN. Although initially both methods seem to use a similar number of SAQs, RECN-DD deallocates many more SAQs than RECN after congestion appearance and, finally, RECN-DD uses fewer SAQs. Therefore, the transient state in the formation of congestion trees is better handled by RECN-DD, as RECN uses more SAQs than required for eliminating HOL blocking.

Moreover, Figure 8.a shows the maximum number of SAQs required in that case at an input port. As can be observed, the SAQs requirements for RECN-DD are much lower than for RECN. Specifically, most of the time RECN-DD requires only 2 SAQs at the most whereas RECN requires up to 5. This result suggests that RECN-DD could keep network throughput in a more restrictive environment (fewer SAQs implemented at each switch port).

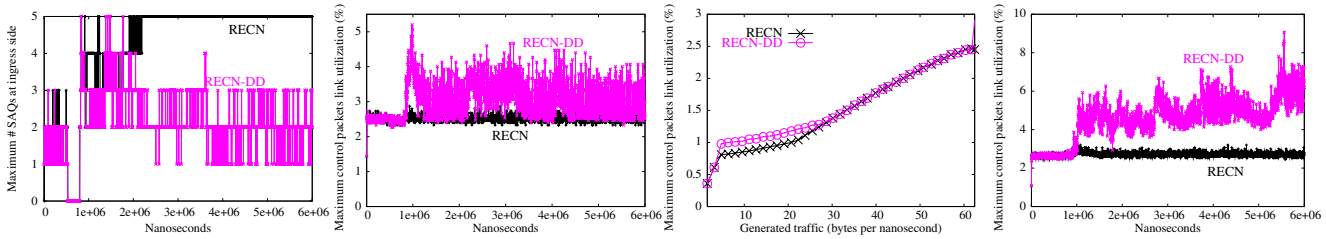
In this sense, Figure 6.b shows throughput results for the same traffic case as Figure 6.a, but this time the maximum number of SAQs allowed per input or output port is 2. It can be seen in Figure 6.b that although both RECN and RECN-DD performances suffer slightly due to the lack of SAQs, throughput degradation is bigger for RECN (throughput drops almost 10%, and it exhibits greater oscillations). However, performance degradation with RECN-DD is smaller, and throughput presents greater stability. Furthermore, Figure 7.b (SAQ utilization for that case) shows that RECN-DD still uses fewer SAQs than RECN (up to a 70% of SAQs, approx.). This result means that many SAQs in RECN remain unnecessarily allocated or allocated for the wrong congested point, while they could be deallocated or re-allocated for a more suitable congested point.

On the other hand, Figure 8.b shows the maximum link utilization for this case. It can be seen that control messages on RECN-DD consume more bandwidth than in the RECN case. This is because RECN-DD use more flow control packets and these are larger than those used by RECN. Anyway, this increment in control messages is limited, and it does not seem to affect RECN-DD performance.

Similar conclusions can be drawn from the results presented in Figures 6.c, 7.c and 8.c. In this case, 2 SAQs per set are used, and the same metrics for the 64×64 BMIN are shown now as a function of a variable injection rate instead of time. As can be seen in Figure 6.c, network throughput for RECN drops when the injection rate is around 60 bytes/ns (maximum is 64 bytes/ns), while RECN-DD keeps throughput at the maximum. VOQsw and 4Q results are worse than those obtained with RECN versions. Figure 7.c shows that RECN-DD uses a slightly greater number of SAQs for low injection rates, but this number is much greater (almost two times) in the RECN case for high injection rates. This result reinforces the idea that RECN-DD uses network resources more efficiently.

Figures 6.d, 7.d and 8.d present results for traffic case #3 (512×512 BMIN). In this case, the maximum number of SAQs has been reduced to four[3]. Again, the results show that RECN-DD achieves better performance than RECN while using a smaller number of SAQs. As can be seen, network throughput drops significantly (almost 15%) when

[3]Previous experiments reported that RECN performed correctly for a maximum of 8 SAQs per set.



(a) 64×64 BMIN traffic 1, 8 SAQs

(b) 64×64 BMIN traffic 1, 2 SAQs

(c) 64×64 BMIN traffic 2, 2 SAQs

(d) 512×512 BMIN traffic 3, 4 SAQs

Figure 8. Maximum number of SAQs at input ports and Maximum link utilization by control messages.

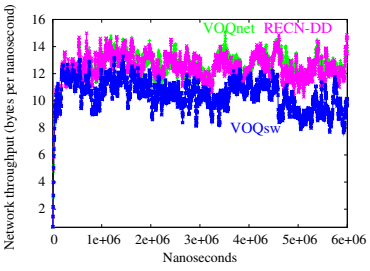


Figure 9. Trace analysis. 64×64 **BMIN, 4SAQs.**

RECN is used (Figure 6.d). It can be observed than VOQsw and 4Q suffer strong degradation. On the other hand, RECN-DD keeps performance almost at a maximum, and again by using fewer SAQs than RECN (up to a 40% of the SAQs required by RECN,Figure 7.d). Therefore, the efficient use of network resources afforded by RECN-DD is the key to handling congestion trees correctly, allowing as it does the required resources to be reduced. Regarding the use of links by control packets, RECN-DD exhibits (Figure 8.d) a more intense use, but it is always lower than 10% and it is necessary for keeping good network performance.

Finally, Figure 9 shows throughput results for VOQsw, VOQnet (VOQs at network level) and RECN-DD with a maximum of 4 SAQs when traces are used as the traffic load in a 64×64 BMIN. It can be observed that, also for real traffic, RECN-DD eliminates HOL blocking with few resources. Notice that it achieves almost the same results as VOQnet by using far fewer queues at each port (in this case, VOQnet requires 64 queues at each port), and again better results than VOQsw.

6. Memory area requirements

From the previous results, it is possible to obtain an estimation of the minimum data memory area required for implementing different congestion control strategies at each switch port. Table 2 shows this estimation for the implementation of VOQnet, RECN and RECN-DD in networks

	Data memory area (per switch input port)	
Queue Scheme	64-endnode network	512-endnode network
VOQnet	$16mm^2$	$128mm^2$
RECN	$4mm^2$	$4mm^2$
RECN-DD	$2.5mm^2$	$3mm^2$

Table 2. Data memory area consumption.

of two sizes (64 and 512 endnodes, respectively). Note that only those strategies that completely eliminate HOL blocking have been considered. Note also that the area values in the table correspond to input ports, and that both RECN and RECN-DD would require less area at output ports.

For each case, the minimum data memory per input port has been calculated as the minimum storage requirements for Virtual Cut-Through switching (one packet for each queue in the input port). In all the cases, we have assumed an Advanced Switching network (allowing a maximum packet size of 2 KB) with 8-port switches. For RECN and RECN-DD, the number of queues per input port has been fixed to the minimum that guarantees maximum performance (taking into account the previous results). Therefore, 8 SAQs + 8 detection queues per input port have been considered for RECN (both networks). For RECN-DD, 2 SAQs + 8 detection queues per input port have been considered for the 64-endnode network case, and 4 SAQs + 8 detection queues per input port for the 512-endnode network. For VOQnet, the number of queues per port is the number of network endnodes. The area required in each case has been estimated from the calculated minimum data memory and from memory datasheets of typical ASIC technologies available to European Universities, assuming the use of two-port SRAM memories (0.18 μm CMOS technology) with a 32-bit organization.

From the values presented, we can conclude that both RECN and RECN-DD not only require much less memory area at each port than VOQnet, but they also exhibit an excellent scalability. Note that VOQnet is not scalable at all. Furthermore, RECN-DD reduces significantly (a third and a quarter, respectively) the memory area required by RECN at each port, thus reducing the overall switch cost.

7. Conclusions

In current networks for MPPs and clusters, an efficient congestion management technique is needed to keep system performance beyond saturation. Recently, RECN was proposed for handling congestion trees in an efficient way. RECN dynamically separates congested packets from non-congested ones, thus eliminating the HOL blocking. This is achieved by dynamically allocating and deallocating SAQs (Set Aside Queues) for congested packets at switches.

In this paper, we have presented a new RECN version, referred to as RECN-DD. This version introduces a new, completely distributed deallocation SAQ mechanism that does not impose any constraint on the order in which queues are deallocated. By doing this, RECN-DD reduces the RECN queue requirements while exhibiting the same performance. This leads to a significant reduction of the data memory area required at each port.

Additionally, RECN-DD avoids the use of explicit congestion notifications and token-exchanging packets. Instead, only flow control packets are required. This facilitates RECN-DD compatibility with Advanced Switching.

References

[1] ASI-SIG. *Advanced Switching for the PCI Express Architecture*. http://www.intel.com/technology/pciexpress/ devnet/AdvancedSwitching.pdf.

[2] E. Baydal and P. López. A robust mechanism for congestion control. In *Proc. 9th Int. Euro-Par Conference*, August 2003.

[3] W. J. Dally. Virtual-channel flow control. *IEEE Trans. on Parallel and Distributed Systems*, 3(2), March 1992.

[4] W. J. Dally and H. Aoki. Deadlock-free adaptive routing in multicomputer networks using virtual channels. *IEEE Trans. on Parallel and Distributed Systems*, 4(4), April 1993.

[5] W. J. Dally, P. Carvey, and L. Dennison. The avici terabit switch/router. In *Proc. 6th Hot Interconnects*, August 1998.

[6] S. P. Dandamudi. Reducing hot-spot contention in shared-memory multiprocessor systems. *IEEE Concurrency*, 7(1), January 1999.

[7] J. Duato. A new theory of deadlock-free adaptive routing in wormhole networks. *IEEE Trans. on Parallel and Distributed Systems*, 4(12), December 1993.

[8] J. Duato, I. Johnson, J. Flich, F. Naven, P. J. García, and T. Nachiondo. A new scalable and cost-effective congestion management strategy for lossless multistage interconnection networks. In *Proc. 11th Int. Symp. High-Performance Computer Architecture*, February 2005.

[9] J. Duato, S. Yalamanchili, and L. Ni. *Interconnection networks. An engineering approach*. Morgan Kaufmann Publishers, 2002.

[10] D. Franco, I. Garces, and E. Luque. A new method to make communication latency uniform: Distributed routing balancing. In *ACM Int. Conf. on Supercomputing*, May 1999.

[11] P. J. García, J. Flich, J. Duato, I. Johnson, F. J. Quiles, and F. Naven. Dynamic evolution of congestion trees: Analysis and impact on switch architecture. *Lecture Notes in Computer Science (HiPEAC 2005)*, 3793, November 2005.

[12] W. Ho, D. Eager. A novel strategy for controlling hot spot contention. In *Proc. Int. Conf. Parallel Processing*, 1989.

[13] IBA homepage. http://www.infinibandta.org/specs.

[14] M. Katevenis, D. Serpanos, and E. Spyridakis. Credit-flow-controlled atm for mp interconnection: the atlas i single-chip atm switch. In *Proc. of the 4th Int. Symp. on High-Performance Computer Architecture*, February 1998.

[15] J. H. Kim, Z. Liu, A. A. Chien. Compressionless routing: A framework for adaptive and fault-tolerant routing. *IEEE Trans. on Parallel and Distributed Systems*, 8(3), 1997.

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

[17] J. Liu, K. G. Shin, and C. C. Chang. Prevention of congestion in packet-switched multistage interconnection networks. *IEEE Trans. on Parallel and Distributed Systems*, 6(5), May 1995.

[18] Myrinet2000. http://www.cspi.com/multicomputer/products/ 2000_series_networking/2000_networking.htm.

[19] G. Pfister and A. Norton. Hot spot contention and combining in multistage interconnect networks. *IEEE Trans. on Computers*, C-34, October 1985.

[20] Q. QsNet. http://doc.quadrics.com.

[21] S. L. Scott and G. S. Sohi. The use of feedback in multiprocessors and its application to tree saturation control. *IEEE Trans. on Parallel and Distributed Systems*, 1(4), October 1990.

[22] L. Shang, L. S. Peh, and N. K. Jha. Dynamic voltage scaling with links for power optimization of interconnection networks. In *Proc. of 9th. Int. Symp. on High Performance Computer Architecture*, February 2003.

[23] A. Singh, W. J. Dally, B. Towles, and A. K. Gupta. Globally adaptive load-balanced routing on tori. *Computer Architecture Letters*, 3(1), July 2004.

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

[25] Y. Tamir and G. Frazier. Dynamically-allocated multi-queue buffers for vlsi communication switches. *IEEE Trans. on Computers*, 41(6), June 1992.

[26] M. Thottetodi, A. Lebeck, and S. Mukherjee. Self-tuned congestion control for multiprocessor networks. In *Proc. of 7th. Int. Symp. on High Performance Computer Architecture*, February 2001.

[27] W. Vogels and et al. Tree-saturation control in the ac3 velocity cluster interconnect. In *Proc. 8th Conference on Hot Interconnects*, August 2000.

[28] M. Wang, H. J. Siegel, M. A. Nichols, and S. Abraham. Using a multipath network for reducing the effects of hot spots. *IEEE Trans. on Parallel and Distributed Systems*, 6(3), March 1995.

[29] C. Q. Yang and A. V. S. Reddy. A taxonomy for congestion control algorithms in packet switching networks. *IEEE Network*, 9(5), July/August 1995.

[30] P. Yew, N. Tzeng, and D. H. Lawrie. Distributing hot-spot addressing in large-scale multiprocessors. *IEEE Trans. on Computers*, 36(4), April 1987.