Dynamic Evolution of Congestion Trees: Analysis and Impact on Switch Architecture*

P.J. García[1], J. Flich[2], J. Duato[2], I. Johnson[3], F.J. Quiles[1], and F. Naven[3]

[1] Dept. de Informática. Univ. Castilla-La Mancha 02071-Albacete, Spain
{pgarcia, paco}@info-ab.uclm.es
[2] Dept. of Computer Science, Univ. Politécnica de Valencia 46071-Valencia, Spain
{jflich, jduato}@disca.upv.es
[3] Xyratex, Haven, United Kingdom
{Ian_Johnson, Finbar_Naven}@xyratex.com

Abstract. Designers of large parallel computers and clusters are becoming increasingly concerned with the cost and power consumption of the interconnection network. A simple way to reduce them consists of reducing the number of network components and increasing their utilization. However, doing so without a suitable congestion management mechanism may lead to dramatic throughput degradation when the network enters saturation. Congestion management strategies for lossy networks (computer networks) are well known, but relatively little effort has been devoted to congestion management in lossless networks (parallel computers, clusters, and on-chip networks). Additionally, congestion is much more difficult to solve in this context due to the formation of congestion trees.

In this paper we study the dynamic evolution of congestion trees. We show that, contrary to the common belief, trees do not only grow from the root toward the leaves. There exist cases where trees grow from the leaves to the root, cases where several congestion trees grow independently and later merge, and even cases where some congestion trees completely overlap while being independent. This complex evolution and its implications on switch architecture are analyzed, proposing enhancements to a recently proposed congestion management mechanism and showing the impact on performance of different design decisions.

1 Introduction

Designers of large parallel computers, clusters, and on-chip networks are becoming increasingly concerned with the cost and power consumption of the interconnection network. Effectively, current interconnect technologies (Myrinet 2000 [24], Quadrics [28], InfiniBand [16], etc.) are expensive compared to processors. Also, power consumption is becoming increasingly important. As link speed increases, interconnects consume a greater fraction of the total system power [30]. Moreover, power consumption in current high-speed links is almost independent of link utilization. In order to reduce system power consumption, researchers have proposed using frequency/voltage scaling techniques [30]. Unfortunately, these techniques are quite inefficient due to their

* This work was supported by 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.

slow response in the presence of traffic variations and the suboptimal frequency/voltage settings during transitions [34].

As the interconnection network has traditionally been overdimensioned, a simple way to reduce cost and power consumption is reducing the number of network components (i.e. switches and links) and increasing their utilization. However, this solution will increase network contention that, as traffic is usually bursty, may lead to congestion situations. In general, congestion will quickly spread through the network due to flow control, forming congestion trees. The main negative effect of these situations happens when packets blocked due to congestion prevent the advance of other packets stored in the same queue, even if they are not going to cross the congested area. This effect is referred to as head-of-line (HOL) blocking, and may produce a dramatic network throughput degradation.

The behavior of network congestion depends on switch architecture. HOL blocking is one of the main problems arising in networks based on switches with queues at their input ports (Input Queuing, IQ switches), because blocked packets destined to congested output switch ports prevent the advance of packets destined to other non-congested output ports. This problem can limit switch throughput to about 58% of its peak value [17]. So, from the point of view of switch architecture, HOL blocking will affect greatly to IQ switches and also to switches with queues at their input and output ports (Combined Input and Output Queuing, CIOQ switches). These architectures are different from that of traditional switches in communication networks, that uses queues only at their output ports (Output Queuing, OQ switches). The OQ scheme has become infeasible because requires the switch to operate at a much faster speed than the links in order to handle all the possible packets concurrently arriving at the input ports[1], and link speed in current high-speed interconnects is on the order of Gbps.

So, as the switch architecture used in most recent designs follows the IQ or CIOQ model, HOL blocking could be a serious problem in modern interconnects. In CIOQ switches, HOL blocking can be reduced by increasing switch speedup. A switch with a speedup of S can remove S packets from each input and deliver up to S packets to each output within a time slot, where a time slot is the time between packet arrivals at input ports. Note that OQ switches have a speedup of N while IQ switches have a speedup of 1. CIOQ switches have values of S between 1 and N. However, for larger values of S, switches become too expensive, and practical values are between 1.5 and 2. Therefore, the switch speedup increase is limited, and HOL blocking should be controlled by a suitable congestion management mechanism.

It should be noted that although congestion management strategies for computer networks are well known, congestion dynamics in networks (whether on-chip or off-chip) for parallel computers is completely different from the one in computer networks because packets are not dropped when congestion arises. Moreover, congestion is more difficult to be solved due to the quick growth of congestion trees. Additionally, as congestion has not been a problem until recently (networks were overdimensioned), there are relatively few studies on congestion trees and their dynamics [27]. In fact, despite the different queue architectures, it is common belief that congestion trees always grow from the root to the leaves. This is true in networks with OQ switches, but this is not

[1] Theoretically, a $N \times N$ OQ switch must work N times faster than the link speed.

the case for networks with CIOQ switches. In this case, as HOL blocking may occur throughout the network, congestion trees may evolve in several ways.

Thus, we believe that an in-depth study of congestion trees and their dynamic evolution will help in finding better solutions for congestion management in interconnection networks with CIOQ switches. In this paper we take on such challenge. We show that congestion trees do not only grow from the root toward the leaves. There exist relatively frequent conditions under which trees grow from the leaves to the root, cases where several congestion trees grow independently and later merge, and even cases where a congestion tree grows in such a way that completely overlaps with a subset of another congestion tree while being independent from it. This complex evolution has some implications on the way congestion management strategies should be designed. Moreover, some design decisions (e.g., where congestion detection should be performed) also depend on switch architecture, as a consequence of the way congestion trees evolve.

So, in addition to study the evolution of congestion trees, we present some enhancements to a recently proposed congestion management mechanism [12]. These enhancements will provide support for efficiently handling the situations described in the analysis of congestion dynamics.

To sum up, the main contributions of this paper are: 1) a detailed analysis of the dynamics of congestion trees when using CIOQ switches, showing their complex evolution, 2) the proposal of several enhancements to a previously proposed congestion management mechanism so that it will efficiently handle the complex situations (we also show the relationship between those enhancements and switch architecture), and 3) an evaluation of the impact on performance of the enhancements.

The rest of the paper is organized as follows. In Section 2, related work is presented. In Section 3, the dynamic evolution of congestion trees is analyzed in depth. Then, in Section 4, the proposed enhancements, based on the analysis of the dynamic evolution of congestion trees, are presented. Evaluation results are presented in Section 5. Finally, in Section 6, some conclusions are drawn.

2 Related Work

The formation of congestion (or saturation) trees on multistage interconnection networks (MINs) under certain traffic conditions has deserved the attention of researchers for many years [27]. 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 saturation trees appear due to concurrent requests to the same shared memory module. In [9], a taxonomy of hot-spot management strategies is proposed, dividing them into three categories: avoidance-based, prevention-based and detection-based strategies. Although different taxonomies are possible for other environments [40], the former classification is roughly valid also for non-multiprocessor-oriented congestion management techniques.

Avoidance-based strategies require a previous planning in order to guarantee that congestion trees will not appear. Some of these techniques are software-based [41,5], while others are hardware-oriented [39]. In general, these strategies are related to quality of service requirements.

Prevention-based strategies 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. These techniques can be software-based [15] or hardware-oriented [1,29].

When detection-based strategies are used, congestion trees may form, but they can be detected in order to activate some control mechanism that should solve the problem. Usually, this kind of mechanism requires some feedback information. For instance, it is possible to measure the switch buffer occupancy [38,22] or the amount of memory access requests [29] in order to detect congestion. Later, 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 could be sent to all the sources [36] or just to those that cause the congestion [19]. Other mechanisms [8,23,3,4] notify congestion just to the endpoints attached to the switch where congestion is detected.

Recently, a new mechanism have been proposed for networks with CIOQ switches [12]. It is based on dynamically separating the traffic belonging to different congestion trees, eliminating so the HOL blocking introduced by congestion. Additionally, many proposals minimize or eliminate HOL blocking regardless of its cause. Some of them focus on HOL blocking formed at the switch level [2,35,32,7,21] while others at the entire network [6,18]. The use of non-blocking topologies [11] also eliminates HOL blocking. Finally, other strategies like fully adaptive routing [10,20,14,37,31] or load balancing techniques [13,31] may help to delay the appearance of congestion.

It is not in the scope of the present paper to discuss the advantages and drawbacks of all these strategies. However, we must remark that, as far as we know, none of them take into account the formation process of congestion trees. This paper is focused on analyzing this important subject.

3 Dynamic Evolution of Congestion Trees

In this section we will analyze the different scenarios that may arise in the formation of congestion trees for networks with CIOQ switches. In particular, we will focus on two key aspects that greatly influence how congestion trees are formed. The first one is the architecture of the switch whereas the second one is the traffic pattern. As we will show later, the scenarios analyzed in this section must be properly handled by a congestion control mechanism in order to be effective.

3.1 Traditional View

Traditionally, it has been thought that congestion trees form as follows: the root of the congestion tree (e.g., an output port at some switch) becomes congested and, due to the use of flow control, congestion spreads from the root to the leaves. However, this situation happens only in a particular scenario that rarely occurs: when the different traffic flows that form the tree join only at the root. Figure 1.a shows an example: five flows form a congestion tree by meeting at the root switch. The sum of the injection rate of all the flows is higher than the link bandwidth, thus a congestion tree is formed. Before congestion occurs, all the queues used along the path followed by the flows are

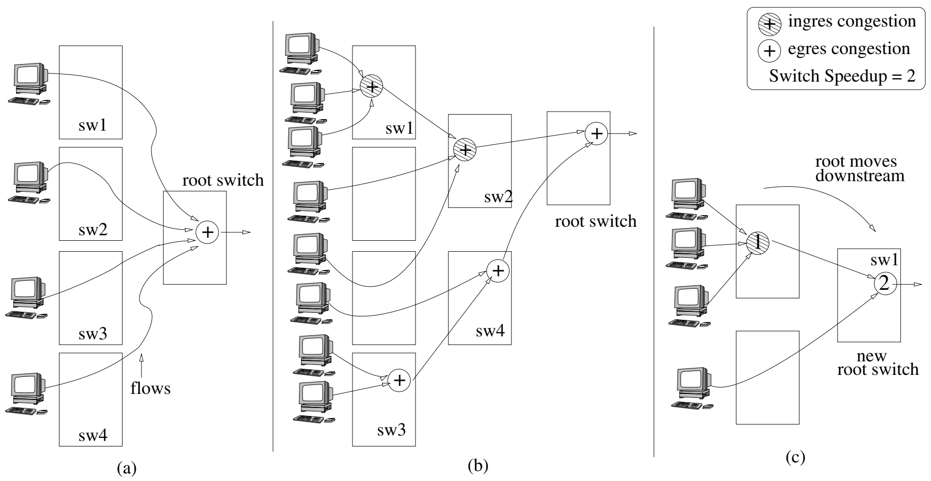


Fig. 1. Different dynamic formations of trees. Speedup is two. (a) Traditional view about the formation of a congestion tree, (b) Different locations (ingress and egress) where congestion is detected for one tree, and (c) the root switch moves downstream.

nearly empty because there is enough bandwidth. However, when flows meet at the root switch, the rate at which packets arrive is higher than the rate at which packets can be transmitted through the output port. Thus, queues will start to fill at the root switch (assuming an appropriate crossbar speedup, egress queues fill first, and later ingress queues), and thus, congestion will begin. This situation will last until the queues at the root switch fill up and, at this time, queues at the previous switches of the root switch will start to queue packets (again, first at the egress queues and then at the ingress queues). This is because the available bandwidth at the output port of the root switch will be divided among the different flows and will be lower than the bandwidth required by each flow. At the end, the congestion tree will grow from the root to the leaves.

Traditionally (and in the former example), it has also been assumed that in a CIOQ switch with speedup, packets will start queuing at the egress ports when congestion forms. However, as will be seen in the next section, this may not always be true.

3.2 Effect of Switch Architecture on the Dynamics

As mentioned above, the switch architecture, specifically the speedup, can alleviate the possible HOL blocking at the input ports. However, it is not completely eliminated. In particular, depending on the number of flows and the rate at which they arrive at a switch, the formation of a congestion tree will be different. Figure 2.a shows an example where two flows, injected at the full rate and headed to the same destination join at a switch with no speedup. As the total reception is higher than the rate at which packets are forwarded to the egress side, packets are queued at the ingress side. Thus, congestion will occur at the ingress side of the switch. However, if speedup is used (Figure 2.b, speedup of 2) congestion occurs at the egress side. As the number of flows arriving at

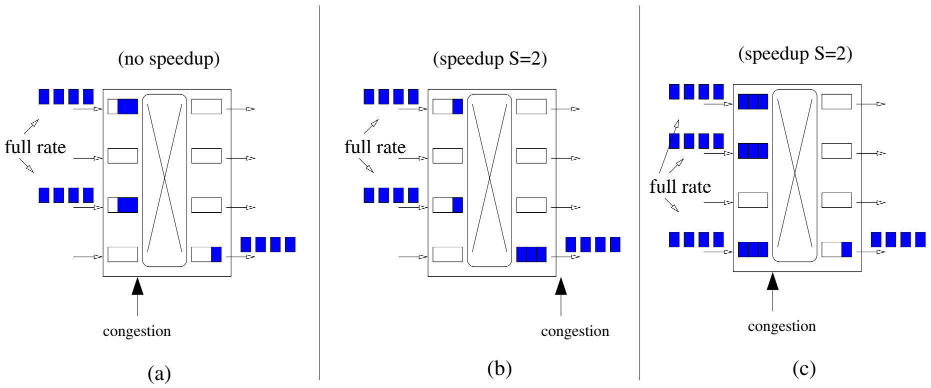


Fig. 2. HOL blocking within a switch with different speedups and different flows

the root switch is equal to or less than the speedup, the switch can forward the flows to the egress side at the same rate they are arriving. But, as the output port bandwidth is half the internal switch bandwidth, packets start to queue at the egress side.

It should be noted that this situation depends on the number of incoming flows, the rate at which they arrive, and the switch speedup. Although the speedup can be increased, it drastically increases the switch cost. So, limited speedups are often used (i.e. not higher than 2). As an example, Figure 2.c shows a case where three flows headed to the same destination arrive at a switch with speedup of 2. The rate at which packets arrive is higher than the rate at which they are forwarded to the egress side. Thus, congestion arises at the ingress side, contrary to the common belief.

This effect may also occur at several points along a congestion tree. In Figure 1.b we can observe a congestion tree formed by eight flows heading to the same destination. Switch speedup is 2. Three flows merge at $sw1$, and other flows merge at different points, finally meeting all together at the root switch. In this situation, at $sw1$ and $sw2$, congestion first occurs at the ingress side. Once all the flows arrive at the root switch they merge and congestion occurs again. However, in this case, congestion first occurs at the egress side, and so happens at $sw3$ and $sw4$.

Therefore, the speedup affects the way congestion trees form. In particular, congestion may first occur at ingress or egress side and, at the same time, different local congestion spots may arise during the formation of the congestion tree. Although the different local congestion spots can be initially viewed as independent congestion trees, they end up being part of a single and larger congestion tree. Whether only the complete congestion tree or each local congestion spot should be treated by a congestion control mechanism will be discussed in section 4. This decision will significantly affect the effectiveness of the congestion control mechanism.

3.3 Impact of Traffic Patterns on the Dynamics

In the previous scenario it was assumed that all the flows were injecting packets at the same rate and started at the same time. This could be the case when executing a barrier synchronization among different processes in a multiprocessor system. However, other scenarios may arise where different flows contributing to the same congestion tree start at different times and inject at different rates. This will lead to more sophisticated dynamics in the formation of congestion trees. As an example, consider Figure 1.c where a first congestion tree made up of three flows is created. These flows, headed to the same destination, join together at the first stage of the network. Thus, the root switch is at the first stage (as speedup of 2 is assumed, congestion first occurs at the ingress side). Later, an additional flow headed to the same destination is injected. However, contrary to the other flows, it merges with them at the second stage. Assuming that all the flows are injecting at the full injection rate, the bandwidth at the egress port of $sw1$ must be shared among the four flows. Thus, a new congestion is formed at the egress side of $sw1$. In fact, two congestion trees have been created at different instants, but the first one later becomes a subtree of a larger tree. This situation can be viewed as if the root of the congestion tree had moved downstream in the network.

Another scenario occurs when two congestion trees overlap. This may happen quite frequently in server systems with different disks placed close to each other. As each accessed disk may produce a congestion tree, several trees may overlap.

When two congestion trees overlap, new dynamic behaviors appear. Figure 3.a shows two congestion trees. The one plotted in solid lines ($ct1$) is formed first, whereas the one plotted in dashed lines ($ct2$) appears later. In this situation, $ct1$ has its root switch at $sw1$ and $ct2$ at $sw2$. However, when $ct2$ appears, a new congestion point is located at $sw3$. This new congestion point can be viewed as a new root for both trees, that will finally merge into one. However, as the congestion point has been formed by

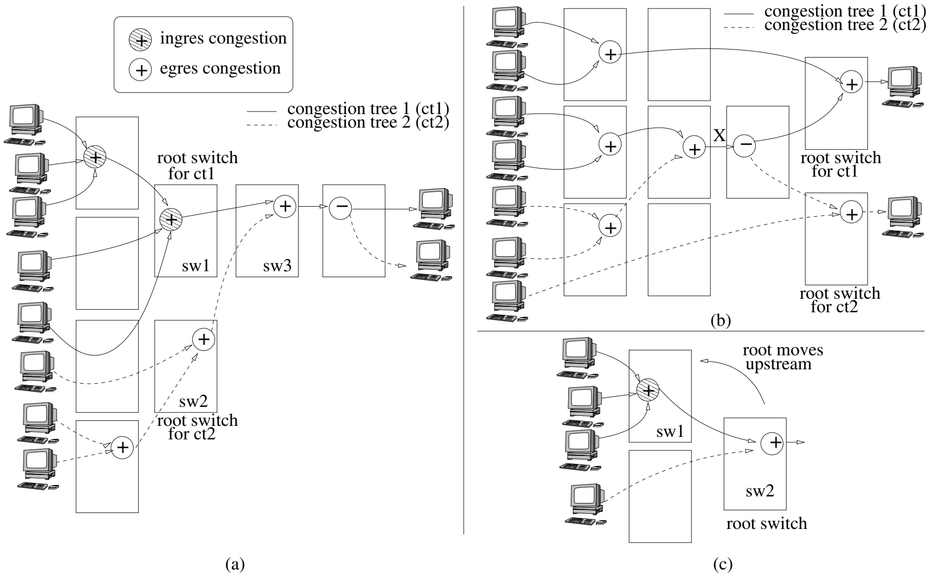


Fig. 3. Different dynamic formations of trees. Speedup is two. (a) two trees overlap and merge, (b) two trees overlap but do not merge, and (c) the root switch moves upstream.

flows headed to different destinations, it could be considered as two different overlapping congestion trees. Therefore, if congestion control mechanisms allocate resources depending on packet destination, separate resources will be needed for each congestion tree. On the other hand, if congestion within the network is going to be treated, this case should be considered as two trees merging into one, thus saving some resources.

Another interesting situation occurs when two congestion trees overlap but do not merge. For instance, once the congestion tree plotted in solid lines ($ct1$) in figure 3.b is formed, a second one ($ct2$, plotted in dots) forms. A $ct2$ branch shares a set of network resources with $ct1$. Thus, point X can be viewed as belonging to both trees. In this situation, it could happen that a congestion control mechanism would consider traffic addressed to $ct2$ passing through X as traffic belonging to $ct1$. As we will show later, a correct differentiation of both trees will improve performance.

Finally, another case occurs when one or several branches of an already formed tree disappear (sources injecting to the congested destination no longer send packets). This case is shown in Figure 3.c. The flow plotted in dashed line disappears and the root switch ($sw2$) no longer experiences congestion. This situation can be viewed as if the root moves upstream the network to $sw1$. However, it should be noticed that while the three flows arriving at $sw1$ keep injecting at the full rate, the queues used at $sw2$ will remain full although it receives packets only from one input port.

To sum up, the root of a congestion tree may move downstream (by the addition of new flows) or upstream (by the collapse of some branches). At the same time, congestion trees may overlap at several network points without merging. And finally, a congestion tree may be formed from local and transient congestion trees that will later merge (due to the limited speedup). Therefore, for all of these cases some kind of architectural support is needed in order to separate each congestion tree and to follow the complex dynamics they may exhibit. As we will see in the evaluation section, keeping track of their dynamics will ensure decisive benefits in terms of network performance.

4 Implications on the Design of Congestion Control Techniques

One of the main objectives of the paper is to develop new mechanisms *able* to keep track of the complex dynamics of congestion trees in networks with CIOQ switches. For this purpose, we present two enhancements to a previously proposed congestion control mechanism referred to as RECN [12]. RECN focuses on eliminating the HOL blocking induced by congestion trees. As we will see in the evaluation, the two new enhancements will be key to achieve maximum performance and will allow RECN to completely eliminate the HOL blocking induced by congestion trees. For the sake of completeness we will first briefly describe RECN and later the two new enhancements.

4.1 RECN (Regional Explicit Congestion Notification)

RECN is based on the assumption that packets from non-congested flows can be mixed in the same queue without significant interference among them. Therefore, RECN focuses on eliminating the HOL blocking introduced by congestion trees. This is accomplished by detecting congestion and dynamically allocating separate buffers for each congestion tree. By completely eliminating HOL blocking, maximum performance is achieved even in the presence of congestion trees.

RECN has been designed for PCI Express Advanced Switching[2] (AS) [25,26]. Although it could work under different technologies, RECN benefits from the routing mechanisms found in AS. In particular, AS uses source deterministic routing. The AS header includes a turn pool 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. An advantage of this routing method is that it allows to address a particular network point from any other point in the network. Thus, a switch, by inspecting the appropriate turnpool bits of a packet, can know in advance if it will pass through a particular network point.

RECN adds, at every input and output port of a switch, a set of additional queues referred to as Set Aside Queues (SAQs). SAQs are dynamically allocated and used to store packets passing through a congested point (root of a congestion tree). To do this, a CAM memory is associated to each set of SAQs. A CAM line contains the control info required to identify a congested point and to manage the corresponding SAQ. Additionally, one queue (referred to as normal queue) is used to store non congested packets.

RECN detects congestion only at switch egress ports. When a normal egress queue receives a packet and fills over a given threshold, a notification is sent to the sender ingress port indicating that the output port is congested. This notification includes the routing information (a turnpool and the corresponding mask bits) to reach the congested output port from the notified ingress port (only a turn). Upon reception of a notification, each ingress port allocates a new SAQ and fills the corresponding CAM line with the received turnpool and mask bits. From that moment, every incoming packet that will pass through the congested point (easily detected from the packet turnpool) will be mapped to the newly allocated SAQ, thus eliminating the HOL blocking it may cause. If an ingress SAQ becomes subsequently congested, a new notification will be sent upstream to some egress port that will react in the same way, allocating an egress SAQ, and so on. As the notifications go upstream, the information indicating the route to the congested point is updated accordingly, in such a way that growing sequences of turns (turnpools) and mask bits are stored in the CAM lines. So, the congestion detection is propagated through all the branches of the tree.

To guarantee in order delivery, whenever a new SAQ is allocated, forwarding packets from that queue is disabled until the last packet of the normal queue (at the moment of the SAQ allocation) is forwarded. This is implemented by a simple pointer associated to the last packet in the normal queue and pointing to the blocked SAQ.

RECN implements for each individual SAQ a special Xon/Xoff flow control, that follows the Stop & Go model. This mechanism is different from the credit-based flow control used for normal queues, that considers all the unused space of the port data memory available for each individual queue. If these "global" credits scheme would be used for SAQs, a congested flow could fill the whole port memory very fast, whereas the Xon/Xoff scheme guarantees that the number of packets in a SAQ will be always below a certain threshold.

[2] AS is an open standard for fabric-interconnection technologies developed by the ASI Special Interest Group. It is based on PCI Express technology, extending it to include other features. ASI is supported by many leader enterprises.

RECN keeps track (with a control bit on each CAM line) of the network points that are leaves of a congestion tree. Whenever a SAQ with the leaf bit set to one empties, the queue is deallocated and a notification is sent downstream, repeating the process until the root of the congestion tree is reached. For a detailed description of the RECN mechanism, please refer to [12].

4.2 Proposed Enhancements

In this section two enhancements to RECN are proposed. They will allow RECN to keep track of the dynamics of congestion trees, and thus, to exhibit significantly better performance.

The first enhancement is allowing RECN to detect congestion at switch ingress ports. RECN defines SAQs at ingress and egress ports, but it only detects congestion at egress ports (it is based on the belief that congestion first occurs at egress side). Thus, ingress ports SAQs are allocated only when receiving notifications. We have previously shown that congestion may first occur at ingress ports (for instance, in switches without speedup). In these cases, RECN never detects congestion at the root. Instead, it detects congestion at the immediate upstream switches, but only when the root ingress queues are full, preventing the packet injection from those switches. So, RECN will not react quickly to eliminate HOL blocking at an important part of the tree.

In order to detect congestion at the ingress side, a different detection mechanism must be used. When detecting congestion at the egress side, the congestion point is the output port by itself. However, when an ingress normal queue fills over a threshold, it is because packets requesting a certain output port are being blocked. As packets in the ingress queue can head to different output ports, it is not trivial to decide which one is the congested output port. In response to this requirement, we propose to replace the normal queue at each ingress port by a set of small buffers (referred to as detection queues). So, at ingress ports, the memory is now shared by detection queues and SAQs. The detection queues are structured at the switch level: there are as many detection queues as output ports in the switch, and packets heading to a particular output port are directed to the associated detection queue[3]. By doing this, when a detection queue fills 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[4]. Once congestion is detected at an ingress port, a new SAQ is allocated at this port, and the turnpool identifying the output port causing congestion is stored in the CAM line. The detection queue where congestion is detected and the SAQ allocated are swapped. As the new SAQ can now be considered to be congested, a notification is sent upstream. Figure 4 shows the proposed mechanism.

The second enhancement is related to the actions taken upon reception of notifications. RECN does not allocate SAQs for all the notifications received. This is done in order to ensure that no out of order packet delivery is introduced. Figure 5.a shows an example where RECN does not allocate a SAQ when receiving a notification. First, $sw1$

[3] Note that, if there were no SAQs, the memory at ingress ports would follow a Virtual Output Queuing scheme at switch level. Of course, SAQs make a difference.

[4] There are other ways of detecting, at ingress sides, the flows contributing to congestion. Detection queues minimize congestion detection latency, but their use is not mandatory.

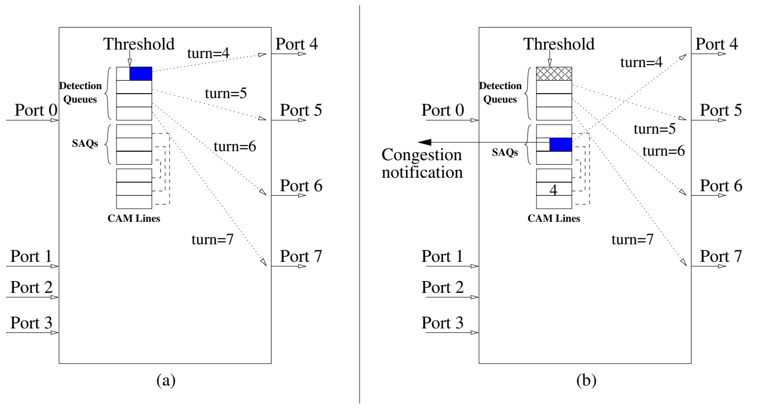


Fig. 4. Mechanism for detecting and handling congestion at the ingress side: (a) Queue status at the detection moment (b) Queue status after detection

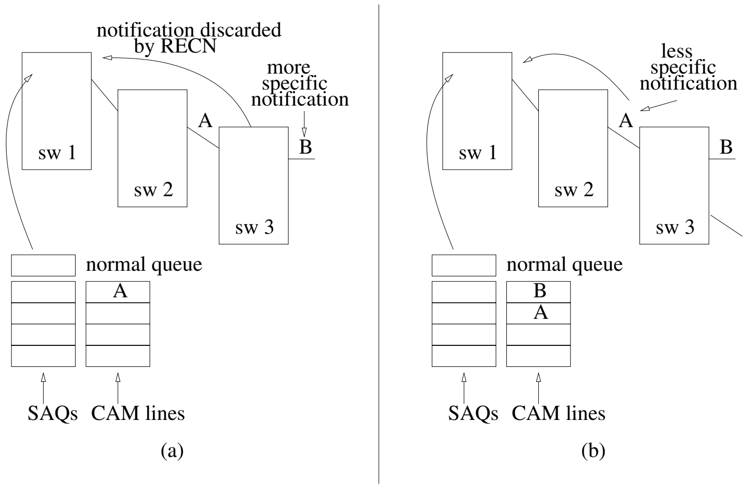

notification discarded by RECN
more specific notification
A
B
sw 1
sw 2
sw 3
normal queue
SAQs
CAM lines
(a)
less specific notification
(b)

Fig. 5. Basic RECN treatment for (a) more specific and (b) less specific notifications

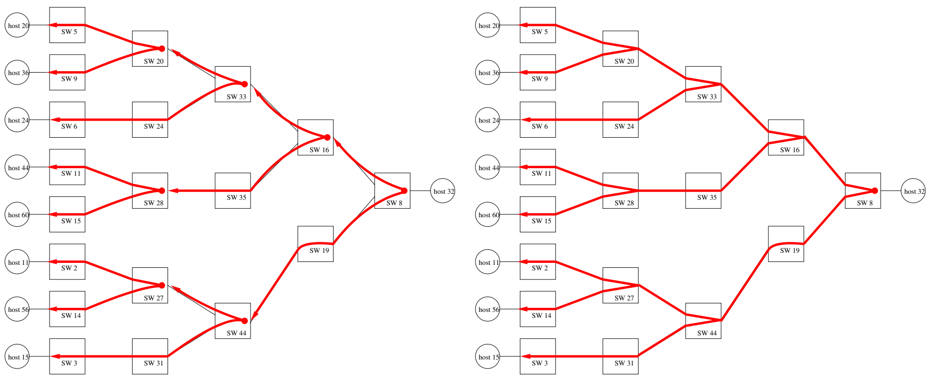
is notified that point A is congested. Then, it allocates a new SAQ for that congested point and packets going through A will be stored from that moment in that SAQ. Later, point B becomes congested at $sw3$ and notifications are sent upstream, reaching $sw1$. This notification is referred to as being a more specific notification (as B is further away than A). Notice that when the notification arrives to $sw1$ it may happen that packets going through B are stored in the SAQ associated to A (as packets have to pass through A before reaching B). If $sw1$ allocates a new SAQ for B, it may happen that packets later stored in that SAQ could leave the switch before the packets stored at the SAQ associated to A (out of order delivery).

Notice that this fact can lead to RECN to introduce HOL blocking. Indeed, flows not belonging to the first tree (traffic adressed to point B) passing trough point A will be mapped to the same SAQ (the one associated to A) that flows belonging to the congestion tree. Thus, RECN does not support the downstream tree movement. Also the situation where a congestion tree forms from leaves to root will not be correctly treated.

Figure 5.b shows another situation. In this case $sw1$, having a SAQ allocated for point B, receives a less specific notification (being point A congested). This can be related to the formation of an overlapping congestion tree. In this case, RECN accepts the notification and allocates a new SAQ for point A. In this situation, an arriving packet will be stored in the SAQ whose associated turnpool matches in more turns the packet turnpool. Thus, incoming packets passing trough A and B will be mapped to the SAQ allocated for B, and packets passing through A but not through B will be mapped to the SAQ allocated for A. Notice that in this situation no out of order delivery may be introduced as already stored packets passing through A but not trough B are not mapped to the SAQ asociated to B (they were stored on the normal queue).

Thus, the proposal is to accept all the notifications regardless whether they are more or less specific. In order to deal with out of order issues, when a new SAQ is allocated due to a more specific notification, it must be blocked (must not send packets) until all the packets stored in the SAQ associated to the less specific notification (when the new SAQ is allocated) leave the queue. This can be accomplished by placing in the "old" SAQ a pointer to the new allocated SAQ .

In order to foresee the potential of the proposed enhancements, Figures 6.a and 6.b show how the original (or "basic") RECN and the enhanced RECN mechanisms detect congestion trees. These figures reflect simulation results when a congestion tree is formed by eight sources injecting packets to the same destination (hot-spot) at the full rate of the link. All the sources start sending packets at the same time. Switch speedup has been set to 1.5. In both figures, dots indicate the points (ingress or egress) considered by the mechanism as congestion roots after the tree is formed. The thick arrows indicate the paths followed by congestion notifications (RECN messages) from congestion points, thus indicating where SAQs are allocated for a particular congestion point.



(a) Basic RECN (b) Enhanced RECN

Fig. 6. Congestion tree detection with different RECN versions

As can be observed, with the basic RECN mechanism, one “real” congestion tree is viewed as several subtrees. In particular, seven subtrees are formed, all of them with their roots at egress sides of switches. However, the enhanced RECN mechanism correctly identifies the congestion tree. At the end, only one tree is detected and the root is located at the egress side of switch 8, matching so the real congestion tree.

Although both schemes end up detecting the same congestion (through one tree or several ones), detecting the correct one has powerful benefits. In particular, for the basic RECN mechanism, HOL blocking is not correctly eliminated. As an example, imagine traffic not belonging to the congestion tree that arrives at switch 5 and is passing through some of the detected intermediate congestion points. Those packets will be mapped to the same SAQ used to hold packets addressed to the congested destination, thus introducing massive HOL blocking. As the congestion tree grows in the number of stages, the number of intermediate detected congestion points will increase and, then, more HOL blocking will be introduced. On the other hand, the enhanced mechanism will use a SAQ in all the switches (on every attached port) to store packets exactly destined to the congestion root. Thus the rest of flows will be mapped to the detection queues, and so HOL blocking at the congestion tree will be completely eliminated.

It has to be noted that enabling congestion detection at switch ingress sides also has benefits in the mechanism response. In the basic RECN, intermediate output ports (detected as congested) start to congest only when the ingress queue at the downstream switch totally fills up with congested packets. Thus, at the time congestion is detected some ingress side queues are totally full.

5 Performance Evaluation

In this section we will evaluate the impact of the proposed enhancements over RECN on the overall network performance in different scenarios of traffic load and switch architecture. 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 will describe the main simulation parameters and the modeling considerations we have used in all the evaluations. Secondly, we will analyze the evaluation results.

5.1 Simulation Model

The simulator models a MIN with switches, end nodes, and links. To evaluate the different congestion manegement techniques, we have used several bidirectional MINs (BMINs) shown in Table 1. 8-port switches are used and the interconnection pattern is the perfect shuffle pattern.

In all the experiments deterministic routing has been used. Memories of 32KB have been modeled for both input and output ports of every switch. At each port, the memory is shared by all 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. For the basic RECN mechanism only one normal queue and a maximum of eight SAQs are defined at ingress and egress ports. However, for the enhanced RECN (RECN with the two proposed enhancements) the normal queue at ingress ports has been divided in eight detection queues.

Table 1. Traffic corner cases evaluated

		normal traffic			congestion tree					
Traffic case	network	# srcs	dest	injection rate	# srcs	dest	injection rate	start time	end time	congestion type
#1	64×64	75%	rand	50%	25%	32	100%	variable	variable	incremental
#2	64×64	75%	rand	100%	25%	32	100%	variable	variable	incremental
#3	64×64	75%	rand	50%	25%	32	100%	800 μs	1100 μs	sudden
#4	64×64	75%	rand	100%	25%	32	100%	800 μs	1100 μs	sudden
#5	512×512	75%	rand	100%	6.25%	32	100%	800 μs	1100 μs	sudden
					6.25%	201	100%	800 μs	1100 μs	sudden
					6.25%	428	100%	800 μs	1100 μs	sudden
					6.25%	500	100%	800 μs	1100 μs	sudden
#6	2048×2048	75%	rand	100%	6.25%	32	100%	800 μs	1100 μs	sudden
					6.25%	515	100%	800 μs	1100 μs	sudden
					6.25%	1540	100%	800 μs	1100 μs	sudden
					6.25%	2000	100%	800 μs	1100 μs	sudden

At switches, packets cross from any input queue to any output one through a multiplexed crossbar modeled with two options: no speedup (link and crossbar bandwidth is 8Gbps), or 1.5 speedup (link bandwidth is 8Gbps, crossbar bandwidth is 12Gbps).

End nodes are connected to switches using Input Adapters (IAs). Every IA is modeled with a fixed number of N message admitance queues (where N is the total number of end nodes), and a variable number of injection queues, that follow a scheme similar to that of the output ports of a switch. When a message is generated, it is stored in the admitance 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 two versions of RECN: 1) Basic RECN: detection only at egress side and more restrictive notifications discarded, and 2) Enhanced RECN: detection at ingress ports enabled and more restrictive notifications accepted, preserving in-order delivery. For comparison purposes we also evaluate the Virtual Output Queuing (VOQ) mechanism at the switch level [2]. This method will be referred to as VOQsw.

5.2 Traffic Load

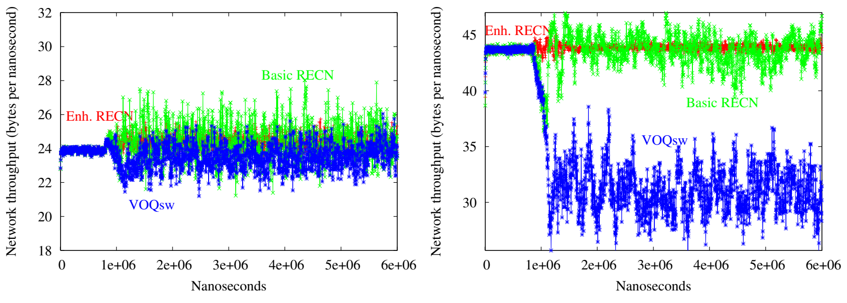
In order to evaluate the RECN mechanisms, two different scenarios will be analyzed. First, well-defined synthetic traffic patterns will be used. Table 1 shows the traffic parameters of each traffic case. For each case, there will be 75% of sources injecting traffic to random destinations during all the simulation period. These nodes will inject traffic at different rates of the link depending on the traffic case. In all the cases, the rest of sources (25%) will inject traffic at the full rate to the same destination (medium networks, traffic cases #1 to #4) or to four different destinations (large networks, traffic cases #5 and #6). Thus, congestion trees will be formed. When sudden congestion is used, all the congestion sources start injecting at the same time. In the case of incremental congestion, congestion sources start to inject one after one at intervals of 20 μs. Congestion sources inject also during 300 μs in the incremental configuration.

As a second scenario we will use traces. The I/O traces used in our evaluations were provided by Hewlett-Packard Labs [33]. 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 answers generated by the disks. As the traces are six years old, and the technology grows quickly, allowing the use of faster devices (hosts and storage devices) that generate higher injection rates, we have applied a time compression factor to the traces.

5.3 Performance Comparison

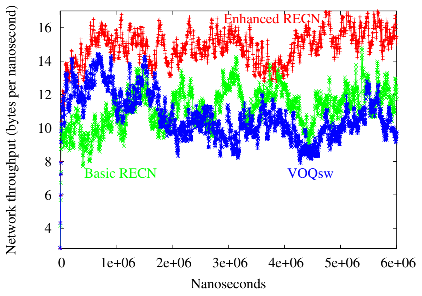
Figure 7 shows results for basic RECN, enhanced RECN and VOQsw, for the traffic cases #1 and #2. Speedup of 1.5 is used. In these two cases, the congestion tree is formed by the incremental addition of flows, and the basic RECN mechanism achieves roughly the same performance that as enhanced RECN. However, there are significant differences. First, basic RECN exhibits an oscilation in the throughput achieved, due to the fact that HOL blocking is not completely eliminated. The enhanced RECN mechanism achieves a smooth performance. In the traffic case #1, sources injecting to random destinations are injecting at half the link rate. It can be observed that when congestion tree is forming, no strong degradation is experienced. However, for traffic case #2, sources injecting random traffic are injecting at the full injection rate. It can be noticed that the basic RECN mechanism starts to degrade performance as throughput drops from 44 bytes/ns to 37 bytes/ns. However, when congestion tree disapears, it recovers, although exhibiting a significant oscillation. On the other hand, enhanced RECN behaves optimally as it tolerates the congestion tree with no performance degradation.

Taking the throughput results of VOQsw as a reference, it can be deduced that both RECN versions handle the congestion caused by traffic case #2 very well, whereas the performance achieved by VOQsw is quite poor (throughput drops from 44 to 25 bytes/ns and does not recover even when the congestion tree disappears). Therefore, we can consider that traffic case #2 can cause strong HOL blocking in the network. However, taking into account the performance of basic and enhanced RECN, we can conclude that the use of any of them virtually eliminates HOL blocking.

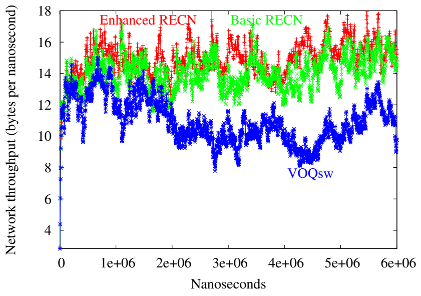


(a) Traffic case #1 (b) Traffic case #2

Fig. 7. Network throughput for traffic cases #1 and #2. Speedup is 1.5



(a) No Speedup

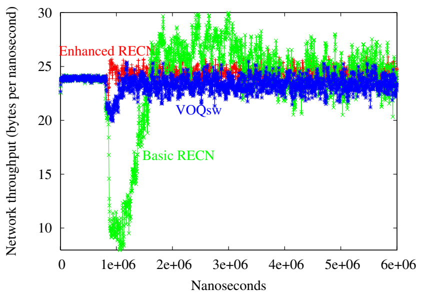


(a) Speedup 1.5

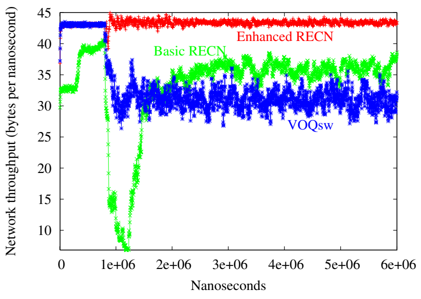
Fig. 8. Network throughput for SAN traffic

Figure 8 shows results for the SAN traces, when using no speedup (Figure 8.a) and when using 1.5 speedup (Figure 8.b). As can be noticed, the basic RECN achieves worse performance when no speedup is used, whereas the enhanced RECN works equally with or without speedup. This result shows that it completely eliminates HOL blocking at the ingress ports. When using speedup, most of the congestion is roughly moved to the egress queues and, thus, the basic RECN behaves as the enhanced RECN. Thus, when no speedup is available, enhanced RECN makes a difference.

From these two previous results (traffic cases #1 and #2 and SAN traffic) it can be deduced that basic RECN behaves acceptably with moderated traffic and when switches with speedup are available. However, these two conditions will not be always true. Figure 9 shows performance results when no speedup is available and a sudden congestion tree forms. In Figure 9.a sources injecting random traffic inject at the half of the link rate (traffic case #3) whereas in Figure 9.b they inject at the full link rate (traffic case #4). As can be observed, the basic RECN suffers strong degradation in both cases. Network throughput drops from 25 bytes/ns to 10 bytes/ns (60% drop) for traffic case #3 and from 44 bytes/ns to 10 bytes/ns (77% drop) for traffic case #4. It can be noticed also that basic RECN recovery time is quite excesive for traffic case #4 (indeed, it never

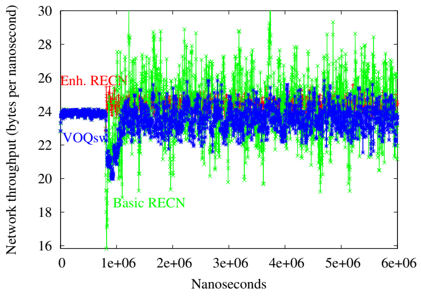


(a) Traffic case #3

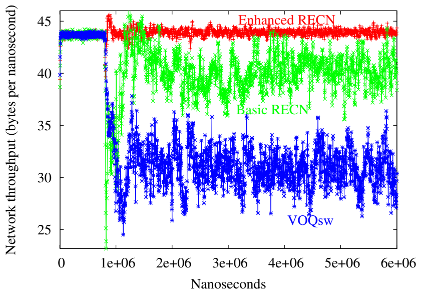


(b) Traffic case #4

Fig. 9. Network Throughput for traffic cases #3 and #4. No speedup

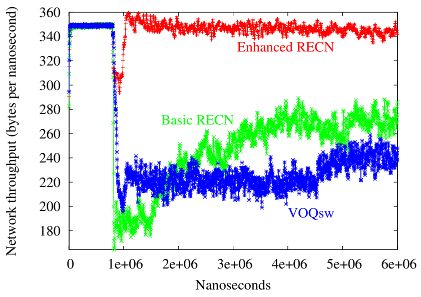


(a) Traffic case #3

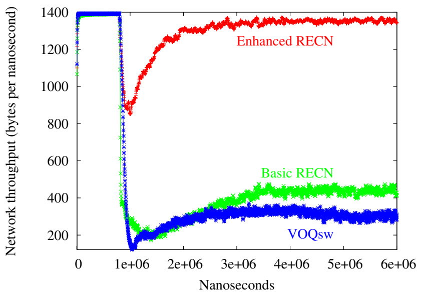


(b) Traffic case #4

Fig. 10. Network Throughput for traffic cases #3 and #4. Speedup is 1.5.



(a) Traffic case #5, 512 $\times$ 512 network



(b) Traffic case #6, 2048 $\times$ 2048 network

Fig. 11. Network Throughput for traffic cases #5 and #6. Speedup is 1.5.

fully recovers). On the other hand, the enhanced RECN mechanism is able to filter the inefficiencies induced by the absence of speedup and by the dynamics of congestion trees, achieving maximum performance. Thus, it virtually eliminates HOL blocking.

Figure 10 also shows performance results for a sudden congestion tree formation, when switch speedup is 1.5. Again, network throughput for traffic cases #3 and #4 are shown in Figure 10.a and Figure 10.b, respectively. With both traffics, the behavior of the basic RECN mechanism is better than in the case of no-speedup switches, due to the fact that congestion tends to appear at egress sides. However, the basic RECN mechanism still exhibits worse performance than the enhanced one. Whereas the enhanced RECN keeps almost constantly network throughput at maximum, the basic RECN troughput drops significantly (33% for traffic case #3 and 45% for traffic case #4) and exhibits strong oscillations. Moreover, when sources inject at full rate (traffic case #4), network troughput does not completely recover from the drop. When the congestion tree disappears, the basic RECN performance is far better than the VOQsw one.

Figure 11 shows performance results for basic RECN and enhanced RECN when used on larger networks. Figure 11.a corresponds to a 512 $\times$ 512 MIN network (512 hosts, 640 switches), whereas Figure 11.b corresponds to a 2048 $\times$ 2048 MIN network (2048 hosts, 3072 switches). In both cases, switch speedup of 1.5 has been considered. Due to the big size of these networks, instead of producing just one tree, four congestion trees are suddenly formed by sources injecting at the full link rate (traffic cases #5 and #6, respectively). It can be seen in the figures that the differences on the performance of basic and enhanced RECN grow with network size. The enhanced version keeps network throughput close to the maximum independently of network size, whereas the basic RECN throughput drops dramatically and practically does not recover.

6 Conclusions

We have analyzed the complex nature of the formation of congestion trees in networks with CIOQ switches. We have presented examples showing that, contrary to the common belief, a congestion tree can grow in a variety of ways, depending on switch architecture (crossbar speedup) and traffic load. Also, we have analyzed the importance of considering such variety when designing congestion management strategies for lossless networks, specifically those strategies focused on eliminate HOL blocking. Moreover, taking into account the previous analysis, we have proposed two significant enhancements to a recently proposed congestion management mechanism (RECN) in the aim of handling congestion trees regardless the way they form. The comparative results presented in the paper show that network performance degrades dramatically in several scenarios when using basic RECN, whereas the enhanced RECN is able to keep almost maximum performance in all the cases. So, we can conclude that the proposed enhancements allow RECN to correctly eliminate the HOL blocking produced by congestion trees independently of the way they grow.

References

1. G. S. Almasi, A. Gottlieb, “Highly parallel computing”, *Ed. Benjamin-Cummings Publishing Co., Inc.*, 1994.
2. 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.
3. E. Baydal, P. Lopez and J. Duato, “A Congestion Control Mechanism for Wormhole Networks”, in *Proc. 9th. Euromicro Workshop Parallel & Distributed Processing*, pp. 19–26, Feb. 2001.
4. E. Baydal and P. Lopez, “A Robust Mechanism for Congestion Control: INC”, in *Proc. 9th International Euro-Par Conference*, pp. 958–968, Aug. 2003.
5. R. Bianchini, T. J. LeBlanc, L. I. Kontothanassis, M. E. Crovella,“Alleviating Memory Contention in Matrix Computations on Large-Scale Shared-Memory Multiprocessors”, Technical report 449, Dept. of Computer Science, Rochester University, April 1993.
6. W. J. Dally, P. Carvey, and L. Dennison, “The Avici Terabit Switch/Router”, in *Proc. Hot Interconnects 6*, Aug. 1998.
7. W. J. Dally, “Virtual-channel flow control”, *IEEE Trans. on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194–205, March 1992.
8. 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.
9. S. P. Dandamudi, “Reducing Hot-Spot Contention in Shared-Memory Multiprocessor Systems”, in *IEEE Concurrency*, vol. 7, no 1, pp. 48–59, January 1999.

10. 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.
11. J. Duato, S. Yalamanchili, and L. M. Ni, *Interconnection Networks: An Engineering Approach* (Revised printing), Morgan Kaufmann Publishers, 2003.
12. J. Duato, I. Johnson, J. Flich, F. Naven, P.J. Garcia, T. Nachiondo, "A New Scalable and Cost-Effective Congestion Management Strategy for Lossless Multistage Interconnection Networks", in *Proc. 11th International Symposium on High-Performance Computer Architecture* (HPCA05), pp. 108–119, Feb. 2005.
13. 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.
14. P. T. Gaughan and S. Yalamanchili, "Adaptive Routing Protocols for Hypercube Interconnection Networks", *IEEE Computer*, vol. 26, no. 5, pp. 12–23, May 1993.
15. W.S.Ho, D.L.Eager, "A Novel Strategy for Controlling Hot Spot Contention", in *Proc. Int. Conf. Parallel Processing*, vol. I, pp. 14–18, 1989.
16. InfiniBand Trade Association, "InfiniBand Architecture. Specification Volume 1. Release 1.0". Available at http://www.infinibandta.com/.
17. M. Karol, M. Hluchyj, and S. Morgen, "Input versus Output Queueing on a Space Division Switch", in *IEEE Transactions on Communications*, vol. 35, no. 12, pp.1347-1356, 1987.
18. 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.
19. 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.
20. S. Konstantinidou and L. Snyder, "Chaos Router: Architecture and Performance", in *Proc. 18th International Symposium on Computer Architecture*, pp. 79–88, June 1991.
21. 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.
22. J. Liu, K. G. Shin,C. C. Chang, "Prevention of Congestion in Packet-Switched Multistage Interconnection Networks", *IEEE Transactions on Parallel Distributed Systems*, vol. 6, no. 5, pp. 535–541, May 1995.
23. P. Lopez 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.
24. Myrinet 2000 Series Networking. Available at http://www.cspi.com/multicomputer/products/2000_series_networking/ 2000_networking.htm.
25. "Advanced Switching for the PCI Express Architecture". White paper. Available at http://www.intel.com/technology/pciex press/devnet/AdvancedSwitching.pdf
26. "Advanced Switching Core Architecture Specification". Available at http://www.asi-sig.org/specifications for ASI SIG.
27. 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.
28. Quadrics QsNet. Available at http://doc.quadrics.com
29. S. L. Scott, G. S. Sohi,"The Use of Feedback in Multiprocessors and Its Application to Tree Saturation Control", *IEEE Transactions on Parallel Distributed Systems*, vol. 1, no. 4, pp. 385–398, Oct. 1990.
30. 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.
31. 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.

32. A. Smai and L. Thorelli, “Global Reactive Congestion Control in Multicomputer Networks”, in *Proc. 5th Int. Conf. on High Performance Computing*, 1998.
33. *SSP homepage*, http://ginger.hpl.hp.com/research/itc/csl/ssp/
34. 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.
35. 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.
36. 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.
37. 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.
38. W. Vogels et al., “Tree-Saturation Control in the AC3 Velocity Cluster Interconnect”, in *Proc. 8th Conference on Hot Interconnects*, Aug. 2000.
39. M. Wang, H. J. Siegel, M. A. Nichols, S. Abraham, “Using a Multipath Network for Reducing the Effects of Hot Spots”, *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no.3, pp. 252–268, March 1995.
40. 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.
41. P. Yew, N. Tzeng, D. H. Lawrie,“ Distributing Hot-Spot Addressing in Large-Scale Multiprocessors”, *IEEE Transactions on Computers*, vol. 36, no. 4, pp. 388–395, April 1987.