Universidad Politécnica de Valencia

Arquitectura de redes de altas prestaciones

Congestion Control in Multistage Interconnection Networks

Author:
Andrew Fecheyr Lippens

March of 2010

Congestion Control in Multistage Interconnection Networks

Andrew Fecheyr Lippens

Abstract

TODO

Contents

1 Introduction

Interconnection networks in large parallel computers and computer clusters are becoming more expensive and are consuming more power. Interconnect technologies such as Myrinet 2000 [4] and InfiniBand [5] have become pricy compared to processors and other system parts. Moreover, interconnects consume a greater fraction of the total system power as link speed increases [6] while the power consumption is almost independent of link utilization. This is a serious problem for system architects as the speed of processors keeps climbing, and with it the ever increasing need for bandwidth.

Historically interconnection networks have been overdimensioned to cope with peak burst traffic. This overcapacity effectively prevented congestion of the links. However, the increasing cost and power hunger of high link speed interconnects is forcing system designers to rethink their strategy. An effective way to reduce cost and power consumption is reducing the number of network components and increasing their utilization. However, this solution will increase network contention that may lead to congestion situations. In general, congestion will quickly spread through the network due to flow control and form congestion trees. The major negative effect of these situations happens when packets blocked due to congestion prevent the advance of other packets in the same queue. This effect, referred to as head-of-line blocking, is explained in section 1.2.

Herefore, reducing the number of network components without a suitable congestion management mechanism may lead to dramatic throughput degradation when the network enters saturation. This paper explores previously published research on the subject of congestion in lossless networks and summarizes proposed techniques to control it.

Before we continue with the related works we explain some technical terms in section 1.1 and in section 1.2 we dig a little deeper in what causes performance degradation in congestion trees: head-of-line blocking.

1.1 Technical terms

This paper uses technical terms from the domain of computer networks. Some of them are explained here to help the reader.

Multistage interconnection networks (MINs) are a class of high-speed computer networks usually composed of processing elements on one end of the network and memory elements and/or other processing elements on the other

end, connected together by switching elements. The switching elements themselves are usually connected to each other in stages. MINs are often used as Network-on-a-Chip and in computer clusters to connect the individual nodes. The lossless nature of these networks makes them quite different from the regular lossy networks that make up the internet.

Congestion trees: In a lossless network, when the input flows arriving at a switch are higher than the output flows the switch buffers will completely fill up. This is called congestion. Flow control mechanisms (such as stop-and-go, or credit based systems) propagate this congestion to throughout the network, following the reversal path of the flows contributing to congestion [1]. This is called a congestion tree. The switch where the congestion originated is referred to the root of the tree and the endpoints as the leaves.

1.2 Head-of-line blocking (HOL)

Head-of-line blocking is a phenomenon that appears in buffered network switches. A switch is usually made of buffered input ports, a switch fabric and buffered output ports. Because of the FIFO nature of the input buffers and switch design, the switch fabric can only switch the packets at the head of the buffer per cycle. HOL arises when packets arriving at different input ports are destined for the same output port. If the HOL packet of a certain buffer at the input cannot be switched to an output port because of contention, the rest of the packets in that buffer are blocked by that Head-of-Line packet. HOL blocking can limit the throughput of switches up to 58.6% of its peak value [7]. Figure 1 illustrates a network switch with input buffers that suffers from HOL as two packets are destined for the same output port, needlessly blocking the packets behind them in the input buffer.

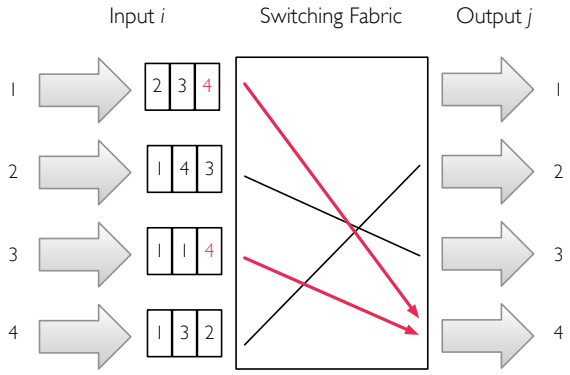

Figure 1: Head-of-line blocking

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*). From the point of view of switch architecture, HOL blocking will greatly affect IQ switches as well as 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, which uses queues only at their output ports (Output Queuing, *OQ switches*). The OQ scheme has become infeasible because it requires the switch to operate at a much faster speed than the links in order to handle all the possible incoming packets concurrently[1], and link speed in current high-speed interconnects is on the order of ten Gbps.

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.

2 Related Work

Congestion in interconnection networks is a well-known problem, and many strategies have been proposed to alleviate their negative effects. The subject has been researched for over two decades [8]. 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. A taxonomy of hot-spot management strategies is given in [9]. The authors divide the different strategies in three categories: avoidance-based, prevention-based and detection-based strategies.

Avoidance based, or proactive congestion management techniques, use a priori knowledge of resource requirements to reserve resources before starting data transmission. They require a previous planning in order to guarantee that congestion trees will not appear. Some of these techniques are software-based [10, 11], while

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

others are hardware-oriented [12]. In general, these strategies are related to quality of service (QoS) requirements.

Prevention based strategies control traffic in a way that congestion trees should not happen. Decisions are made "on the fly", often limiting or modifying routes or memory accesses. These techniques can be implemented in software [13] or hardware [14, 15].

Detection based strategies, also referred to as reactive strategies [2], allow the forming of congestion trees. They detect these formations and activate a control mechanism in order to solve the problem. Detection is usually based on feedback information. For instance, it is possible to measure the switch buffer occupancy [19, 17] or the amount of memory access requests [15] in order to detect congestion. When congestion is detected the sources injecting traffic into the network are notified to reduce or stop the injection. There is a large amount of research in this area that can be categorized into several subclasses. Some solutions broadcast notifications to all the sources [16, 17]. These solutions consume a large fraction of the bandwidth for notification which makes them very inefficient. Other solutions send notifications only to the sources contributing to the congestion [18], making them more efficient. However, none of these solutions scales because as network size and/or link bandwidth increase, the effective delay between congestion detection and reaction (and the number of injected packets during that time) increases linearly. This results in slow response and/or typical oscillations that arise in closed-loop control systems with delays in the feedback loop. Both of these translate into significant performance degradation. Moreover, these limitations are due to physical constraints as notifications cannot propagate faster than the speed of light, thus being unavoidable. Hence, reactive congestion management alone may not be appropriate for large future systems with long round trip times.

The only scalable reactive congestion management strategies are those that react locally by reducing/stopping injection at the endpoints attached to the switch where congestion is detected [20, 21]. Although those techniques have been reported to achieve good results with synthetic traffic where all the nodes have the same injection rate, they may fail under real traffic because sources contributing to congestion are usually different from those that are throttled.

3 Dynamic Evolution of Congestion Trees

In [1] the authors analyze the different scenarios that may arise in the formation of congestion trees for networks with CIOQ switches. They focus on two key aspects that greatly influence how congestion trees are formed. The first one is the architecture of the switch and the second one is the traffic pattern.

The traditional view of congestion trees is that the congestion grows from the root to the leaves: the root of the congestion tree (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.

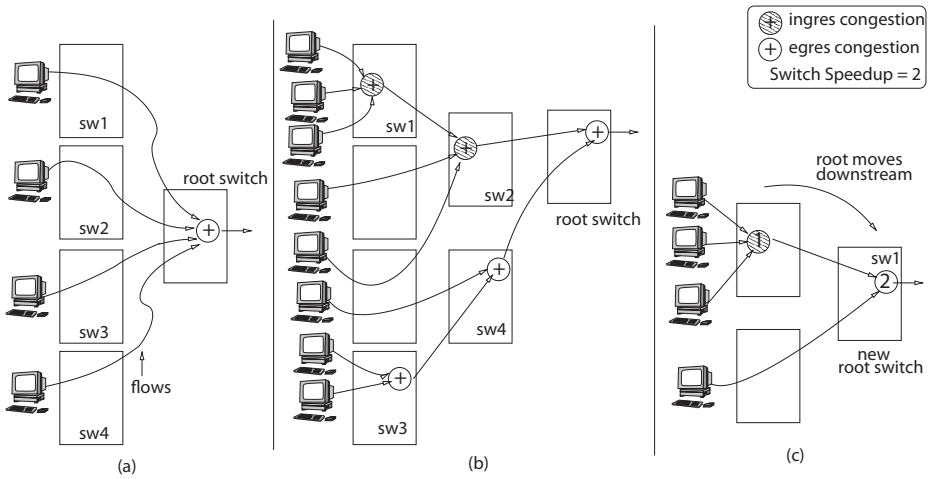

Figure 2: Different dynamic formations of trees. Speedup is two. (a) Traditional view about the formation of a congestion tree, (b) Different locations (ingress and egress) where congestion is detected for one tree, and (c) the root switch moves downstream.

Figure 2.a shows an example of the traditional view: 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.

3.1 Effect of Switch Architecture

The formation of a congestion tree is different depending on the number of incoming flows and the rate at which they arrive at a switch. Switch architecture,

especially the speedup[2] of a switch, is also a determining factor as will be shown in the following example. Figure 3 depicts three different situations. In the first one (a), the switch has no speedup and the two flows, injecting at the full rate, are headed to the same destination port. Because a switch without speedup can only handle one packet per cycle, packets are queued at the ingress side. Thus, congestion will occur at the ingress side of the switch. However, if speedup is used (Figure 3.b) congestion occurs at the egress side. When the number of flows arriving at the root switch is less than or equal to the speedup, the switch can forward the flows to the egress side at the same rate as they are arriving. But, because the output port link speed is only as high as one input link, packets start to queue at the egress side.

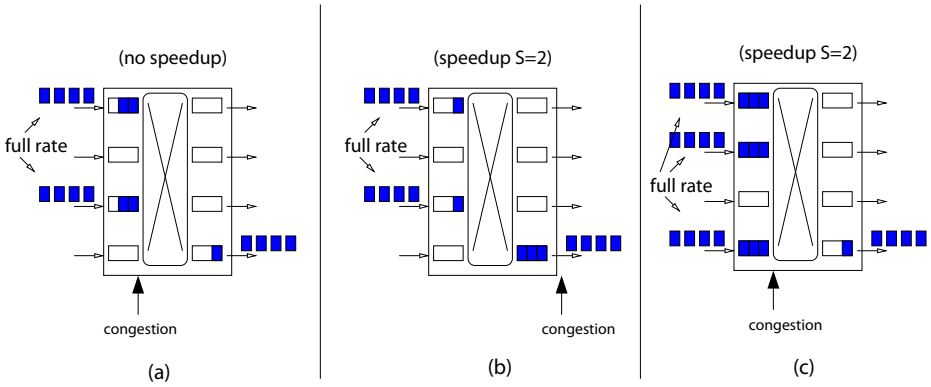

Figure 3: HOL blocking within a switch with different speedups and different flows.

Obviously the situation depends on the number of incoming flows and the speedup of the switch. Figure 3.c illustrates a number of incoming flows higher than the switch speedup, causing the congestion to once again form at the ingress side. Note that congestion will occur regardless of the destination of the incoming flows.

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

[2]A description of switch speedup is given on page 5

3.2 Impact of Traffic Patterns

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 can happen in some cases, such as a barrier synchronization among different processes in a multiprocessor system, but often is not so. Scenarios may arise where different flows contributing to the same congestion tree start at different times and inject at different rates. The authors in [1] describe the more sophisticated dynamics of these scenarios.

They show that the root of a congestion tree may move downstream by the addition of new flows, or upstream by the collapse of some branches. Also, 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.

Now that the dynamics of congestion trees have been explored a bit deeper, we can describe techniques proposed to solve their negative effects.

4 Proposed Solutions

TODO: inleiden

4.1 Basic RECN

RECN stands for regional explicit congestion notification because it detects and explicitly notifies switches about congestion throughout the region that will be affected by a given congestion tree [2]. RECN differs from local explicit congestion notification (LECN) strategies [22] proposed in that it is able to eliminate HOL blocking across multiple stages. The authors in [2] focus on removing HOL blocking in Multistage Interconnection Networks with deterministic routing, effectively making congestion trees harmless.

The technique basically works by separating the packets from congested flows from the packets belonging to uncongested flows. By doing so, congested packets no longer interfere with uncongested packets and even with packets from congested flows belonging to another tree. This is important since different congested flows usually experience significantly different degrees of congestion. Therefore, the technique focuses on quickly detecting congested flows and immediately mapping

them to separate queues. The deallocation of those queues when they are no longer required is also important in order to minimize the number of resources needed to eliminate HOL blocking.

4.1.1 Congestion Detection

The congestion detection mechanism is simple, fast and accurately identifies the sources that are contributing to congestion. RECN's congestion detection mechanism consists of detecting when the occupancy of a given output queue reaches a certain threshold. This technique works for both congestion at the endnodes of the network as well as internal network congestion. Fully determining the sources that are contributing to congestion was deemed infeasible due to data RAM bandwidth consumption and the corresponding interference with packet transmission. As such, the authors opted for another approach. The switch simply propagates notifications to the input ports that are contributing to congestion. Each output port has a flag associated with each input port, which records the transmission of the corresponding notification to avoid repeated congestion notifications.

4.1.2 Congestion Notification and Queue Allocation

When congestion notifications are received at some input port, a set aside queue (SAQ) is locally allocated for the packets leading to the congested output port. All the incoming packets that request the congested output port will be stored in that SAQ, thus preventing them from blocking other packets stored in the main queue for that input port.

Path information is stored in a CAM. Each CAM line contains the path information associated with the corresponding SAQ. The reason for using a CAM is that the switch has to check the destination address field in the header of every incoming data packet and compare it against all the path informations stored in the CAM to determine whether this packet is going to cross the root of some congestion tree. In such a case, it will be stored in the corresponding SAQ. It should be noted that packets from uncongested flows that will use the same output port at the current switch as other packets from congested flows will not match any CAM line, and therefore, will not be stored in any SAQ, thus avoiding HOL blocking. It should be noted that two congestion trees may overlap. Moreover, one of them could be a subtree of the other tree. All of these cases are automatically solved when using a CAM to store path information and incoming notification packets are matched against it.

Figure 4 shows the CAM structure required to store path information in a switch for PCI Express Advanced Switching[3]. 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.

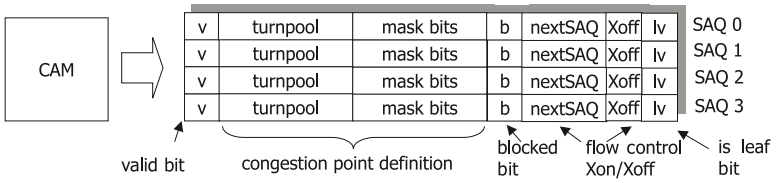

Figure 4: CAM structure.

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. 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 and mask bits are stored in the CAM lines.

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 also implements for each individual SAQ a special Xon/Xoff

[3]AS is an open standard for fabric-interconnection technologies developed by the ASI Special Interest Group [23].

flow control, that follows the stop-and-go model.

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.

4.2 Enhanced RECN

In [2] the authors propose two enhancements to RECN. Both enhancements allow RECN to keep track of the dynamics of congestion trees, and thereby, significantly increase performance.

The first enhancement is allowing RECN to detect congestion at switch ingress ports. Basic RECN defines SAQs at ingress and egress ports, but it only detects congestion at egress ports. However, Section 3.1 on page 7 shows that congestion may first occur at ingress ports. In these cases, basic 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. So, RECN will not react quickly to eliminate HOL blocking at an important part of the tree.

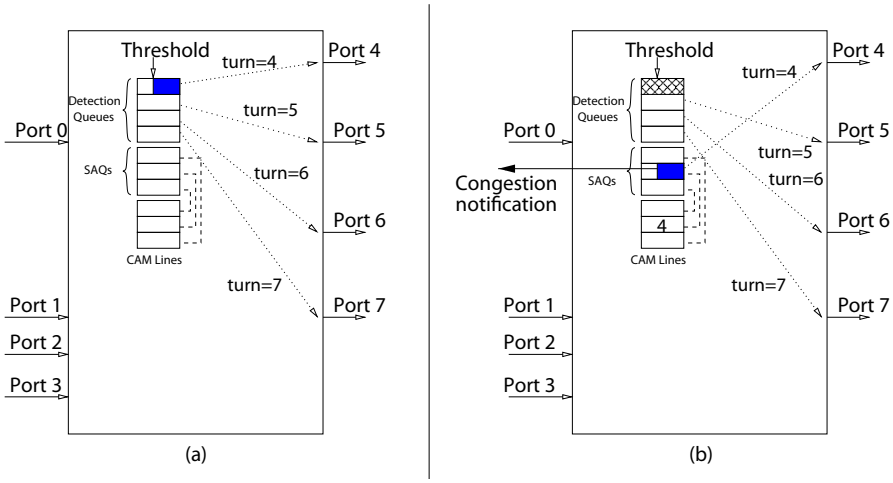

Figure 5: Mechanism for detecting and handling congestion at the ingress side: (a) Queue status at the detection moment (b) Queue status after detection.

The first enhancement is accomplished by replacing the normal queue at each ingress port by a set of small buffers (referred to as detection queues). 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. 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. 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 5 shows the proposed mechanism.

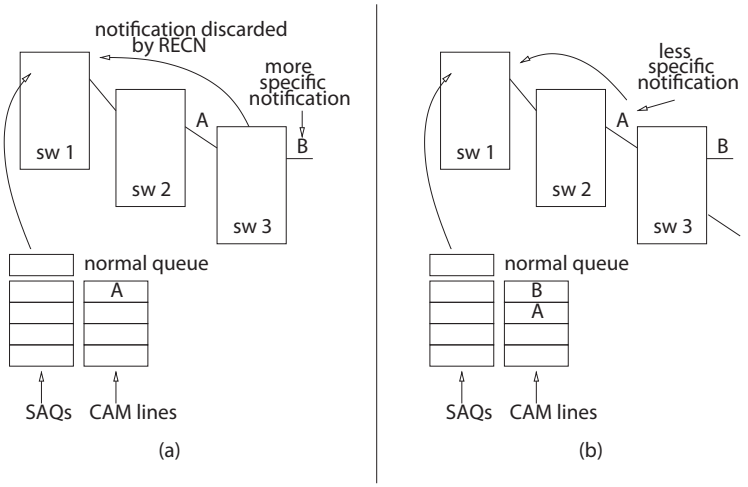

Figure 6: Basic RECN treatment for (a) more specific and (b) less specific notifications.

The second enhancement is related to the actions taken upon reception of notifications. Basic 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 6.a shows an example where RECN does not allocate a SAQ when receiving a notification. First, *sw1* is notified that point A is congested. Then, it allocates a new SAQ for that congested point. 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).

This fact can lead 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) than 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.

The second enhancement 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 a pointer to the new allocated SAQ in the "old" SAQ.

Both enhancements introduced to RECN allow it to detect congestion at ingress ports and allow the system to correctly accommodate for movements of the root switch that are traditionally not taken in to consideration.

5 Performance evaluation

6 Conclusion

References

[1] P.J. García, J. Flich, J. Duato, I. Johnson, F.J. Quiles, F. Naven, *Dynamic Evolution of Congestion Trees: Analysus and Impact on Switch Architecture*, 2005.

[2] J. Duato, I. Johnson, J. Flich, F. Naven, P. García, and T. Nachiondo *A New Scalable and Cost-Effective Congestion Management Strategy for Lossless Multistage Interconnection Networks*, 2005.

[3] J. Escudero-Sahuquillo, P.J. García, F.J. Quiles, J. Flich, and J. Duato, *FBICM: Efficient Congestion Management for High-Performance Networks Using Distributed Deterministic Routing*, 2008.

[4] Myrinet 2000 Series Networking.
Available at `http://www.myricom.com/myrinet/overview/`.

[5] InfiniBand Trade Association, *InfiniBand Architecture. Specification Volume 1. Release 1.0*. Available at `http://www.infinibandta.org`.

[6] 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.

[7] M. Karol, M. Hluchyj, S. Morgan, *Input Versus Output Queuing on a Space-Division Packet Switch*, IEEE Transactions on Communications, Volume 35, Issue 12, pp. 1347–1356, Dec 1987.

[8] 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.

[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] 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.

[11] 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.

[12] 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.

[13] 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.

[14] G.S. Almasi, A. Gottlieb, *Highly parallel computing*, 1994.

[15] 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.

[16] 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.

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

[18] 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.

[19] 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.

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

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

[22] 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.

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