# Destination-Based HoL Blocking Elimination *

T. Nachiondo, J. Flich, and J. Duato
Dept. Informática de Computadores y Sistemas
Universidad Politécnica de Valencia
Camino de Vera, 14, 46071–Valencia, Spain
{tnachion,jflich,jduato}@disca.upv.es

## Abstract

*Congestion management is likely to become a critical issue in interconnection networks, as increasing power consumption and cost concerns will lead to the use of smaller networks. As congested packets introduce head-of-line (HoL) blocking to the rest of packets, congestion spreads quickly. The best-known solution to HoL blocking, Virtual Output Queues (VOQs), is not scalable at all or too costly when implemented in large networks. In previous works, we proposed an efficient and cost-effective solution, referred to as Destination-Based Buffer Management (DBBM). DBBM groups destinations into different sets, and packets addressed to destinations in the same set are mapped to the same queue. DBBM eliminates most of the HoL blocking (among packets addressed to different sets), achieving very good results in terms of throughput and scalability. However, depending on the distribution of packet destinations, it may introduce an uncertain degree of unfairness among packets mapped to the same queue.*

*In order to overcome this problem we propose the Dynamic DBBM mechanism (DDBBM). DDBBM dynamically eliminates the HoL blocking among packets mapped to the same queue in DBBM. Performance results show that DDBBM keeps (and in some cases improves) the good results achieved by DBBM in terms of throughput and scalability. Moreover, DDBBM solves the unfairness introduced by DBBM. Additionally, as an example of applicability, in this paper we show that DDBBM can be applied to InfiniBand with no hardware modification.*

## 1 Introduction

Many compute-intensive applications (nuclear weapon simulations, protein folding, global climate modeling, etc.) require continued research and technology development to deliver computers with steadily increasing computing power. The required levels of computing power can only be achieved with massively parallel computers. Examples of these systems are the Earth Simulator [9] and the BlueGene/L [5]. In these systems, interconnection networks (ICTNs) with low communication latencies and high bandwidth are becoming a key component.

Also, internet portal servers and datacenter servers are emerging environments where high performance ICTNs (InfiniBand [13], Myrinet [6] ...) are being preferred to build cluster-based systems with adequate response time to applications and final users.

In these networks contention is a classic problem. Contention occurs when two packets compete for the same resource (typically a link). As the channel capacity is limited, one of the packets will be transmitted, while the other will wait. If packets belonging to different flows request the same resource, contention will occur among all the packets, and flows will advance through the network at a lower pace. Moreover, contention may derive in severe congestion if the situation persists over time. In a congested situation, packets experience high latencies and the network collapses.

Traditionally, congestion has been managed by using two different strategies: proactive techniques and reactive techniques. Proactive techniques consist on preventing the formation of congestion. Data is injected into the network in such a way that congestion should never occur. This is achieved either by reserving in advance network resources (avoidance-based techniques) [18, 4] or by limiting the routes followed by packets (prevention-based techniques) [1, 16]. In general, the use of both techniques needs a transmission scheduling that requires information about the occupancy of network resources (buffers, links, etc.) thus leading to inefficient solutions.

On the other hand, reactive techniques consist on detecting the congested situations and activating later some mechanisms to eliminate the congestion. Most of the proposed mechanisms [17, 8, 3] consist on notifying congestion to the end nodes in order to cease or reduce the injection of packets.

*This work was supported by the Spanish MCYT under Grant TIC2003-08154-C06-01, by Universidad Politecnica de Valencia under Grant 20040937, and by Junta de Comunidades de Castilla La Mancha under Grant PBC-05-005-2.

However, recently, a totally different approach has been provided [12]. This new approach relies on the idea that congestion is not a problem by itself. Simply, there are too many packets competing for the same resource. The real problem is the negative effects introduced by the congestion. Once different flows become congested within the network (forming a congestion tree) they introduce HoL blocking to the rest of non-congested flows. HoL blocking occurs when a packet at the head of a queue can not be forwarded, and it blocks the rest of packets in the queue; even if those packets could make forward progress. If the HoL blocking were removed, then congestion would be harmless. Based on this approach, in [12] a new mechanism is provided, referred to as RECN. This mechanism detects congestion within the network, and instead of eliminating the congestion, it eliminates the HoL blocking introduced by the congested packets. With this mechanism, maximum network throughput is achieved regardless of the presence of congestion. Unfortunately, RECN has been designed for Advanced Switching (AS) [15] as it relies on the use of source routing for checking if a packet belongs to a congested tree, which prevents RECN to be applied to other networks technologies such as InfiniBand (since InfiniBand uses distributed routing). Moreover, RECN requires some logic at the switches (queues and control structures) not available in commercial products.

## 2 Motivation

It is well known that VOQs at network level (referred in this paper to as $VOQ_{net}$) [2, 14] is the best-known solution to the HoL blocking problem. In $VOQ_{net}$, every switch port implements as many queues as final destinations, and each queue is used only by packets addressed to a unique destination. Although $VOQ_{net}$ completely eliminates HoL blocking, it is not scalable at all, since memory requirements increase quadratically with the number of end nodes in the network. This situation is aggravated by the increasing demand for quality of service (QoS), as every service level requires a dedicated queue per port.

An alternative is the use of VOQs at switch level (referred in this paper to as $VOQ_{sw}$). In $VOQ_{sw}$, each switch has at every input port as many queues as output ports, and packets are mapped according to the requested output port at the switch. Although the number of queues is significantly reduced, HoL blocking can still occur between flows sharing a subset of consecutive links along their paths. Indeed, in [7, 11] it was demonstrated that as network size increases, the percentage of HoL blocking not solved by $VOQ_{sw}$ increases.

In [10, 7, 11] we proposed and evaluated an scalable and cost-effective solution to the HoL blocking problem, the Destination-Based Buffer Management (DBBM) mapping scheme. In DBBM packets are mapped to queues according to their destination (as in $VOQ_{net}$), however, the number of queues is significantly reduced (even lower than the ones used in $VOQ_{sw}$). Basically, in DBBM each queue stores only packets for a subset of destination ports. This can be viewed as if the physical output ports of the network were grouped into a smaller set of logical output ports, and a queue was used at each switch to allocate only packets destined to a particular logical output port (i.e., VOQ at the logical port level). Therefore, the HoL blocking among packets in different logical ports is eliminated. The simplest implementation of DBBM is to use the lower bits of the destination ID of a packet to select the queue (we refer to this as modulo mapping). DBBM achieves a trade-off among implementation cost (the set of queues is reduced and the mapping function required at switches is simple) and efficiency (DBBM has demonstrated to be an efficient mapping scheme, obtaining much better performance than $VOQ_{sw}$).

Unfortunately, DBBM can still suffer from HoL blocking (among packets addressed to the same logical port). The percentage of affected destinations (*victimized* destinations) will depend on the number of queues. For example, with DBBM with 4 queues, only 25% of destinations will potentially be affected by the congested destination. Thus, as more queues were used in DBBM, the percentage of affected destinations would decrease. Also, the percentage of HoL blocking not solved by DBBM depends on the number of congested destinations. If the number of congested destinations increases, potentially, more queues in DBBM will suffer HoL blocking, and the percentage of *victimized* destinations will increase.

As an example, Figure 5 shows the received traffic of each end node in a $64 \times 64$ multistage network when random traffic is sent to all destinations and a hotpot is formed at end node 30. In the case of DBBM with 8 queues (and modulo mapping) we can observe that every eight destinations there is one with a significant lower reception rate. This is because of the HoL blocking introduced by the congested destination, thus introducing unfairness. Notice that this problem comes from the static nature of DBBM, as it always maps the same destinations into the same queues.

In this paper we present an effective mechanism able to deal with the HoL blocking problem as DBBM does, but overcoming the DBBM fairness weakness. The proposed mechanism, referred to as Dynamic DBBM (DDBBM), will eliminate most of the HoL blocking by using the DBBM technique, and whenever a destination becomes congested, its traffic will be dynamically separated. That is, the congested flow will be separated from the non-congested ones, thus, the unfairness introduced by DBBM will be totally eliminated.

This mechanism will be also designed in such a way that can be used in InfiniBand. In this paper we will address the main aspects that must be dealt with in InfiniBand in order to implement DDBBM, without requiring any change to current InfiniBand switch architectures.

The rest of the paper is organized as follows. In Section 3 we describe the new proposed technique (DDBBM). Next,

in Section 4 we describe how the mechanism can be applied to InfiniBand. In Section 5 we analyze DDBBM in terms of scalability, throughput, robustness, and most important, fairness. Finally, we draw our conclusions in Section 6.

## 3 Dynamic DBBM (DDBBM)

DDBBM eliminates the HoL blocking not handled by DBBM (the one caused among packets addressed to the same set of destinations). Therefore, DDBBM should be seen as an improvement of DBBM, where all the good properties of DBBM still remain (most of the HoL blocking is eliminated with a low implementation complexity).

Basically, the DDBBM mechanism works as follows. When an end node detects that itself is a congested destination, it notifies that situation to the sources sending traffic to it. Upon reception of the notification, sources will inject packets addressed to the congested destination with a bit activated (at the packet header). This bit is used by switches to separate congested flows from non-congested flows. In order to do this, at each input port on every switch there is an additional queue (referred to as the dynamic queue) destined to allocate the congested flows, whereas the remaining queues (referred to as DBBM queues) are used to allocate non-congested flows according to DBBM. Thus, each time a packet is received with its bit activated it is mapped to the dynamic queue. As an example, in a DDBBM mechanism with 8 queues for DBBM and a dynamic queue, a packet addressed to destination 48 will be mapped to queue zero (modulo mapping). However, if destination 48 becomes congested, then packets addressed to that destination will be mapped to the dynamic queue. As the congested packets will be mapped to a different queue, DBBM queues will not allocate congested packets and therefore, the HoL blocking that packets addressed to destination 48 might cause will be eliminated.

DDBBM resembles a limitation-based congestion control mechanism where destination nodes detect congestion and inform sources to limit the injection (to eliminate congestion). However, notice that DDBBM neither limits source traffic injection nor eliminates congestion. Instead, it will let the sources to keep their injection rate. A well-known problem of limitation-based protocols is that their efficiency depends on network size and link bandwidth. Probably, at the moment sources are notified, the congestion may be formed within the network or may even vanished, thus introducing oscillations. However, this problem has a minor impact when DDBBM is used, as congestion will only affect to a small percentage of packets.

### 3.1 Detecting Start/End of Congestion

We need a fast and accurate detection mechanism at the end node. The mechanism should be fast in order to minimize the time during which HoL blocking takes place. Also, the mechanism should be accurate in order to prevent false positive detections. Inaccurate detections could lead to worse results, since extra HoL blocking could be introduced. For instance, if the mechanism detects that two destinations are congested, one of them wrongly detected, massive HoL blocking would be introduced to the non-congested flow (as packets for both destinations would be mapped to the dynamic queue). Thus, the key piece of DDBBM is the congestion detection mechanism at every end node.

As a first approach to the detection mechanism, each end node observes its received traffic rate. An end node considers that it is becoming congested whenever its reception rate is higher than a fixed threshold value (Detection Threshold expressed in bytes, $DT$). Similarly, an end node is no longer congested whenever its reception rate is lower than a fixed threshold value (Low Threshold expressed in bytes, $LT$). The reception rate can be easily computed at the end node. Simply the number of bytes received must be counted during a defined reception time frame ($TF$ expressed in cycles).

With this mechanism, permanent congestion at a destination will be detected sooner or later (depending on the $DT$ and $LT$ threshold values). However, experiencing a high reception rate does not necessarily means that the end node is congested. Indeed, congestion occurs at a destination end node only when there is more bandwidth demand than the offered bandwidth at the destination. Since the end node only can count the quantity of bytes received, it does not know if the high reception rate is because of a congested situation (sources are requesting more bandwidth than the offered one) or if it is because of sources are requesting just the offered bandwidth. The only way to properly identify congestion would be achieved by inspecting switches and source end nodes. If they (switches and/or end nodes) start accumulating packets then there is a congested situation. However, notice that inspecting switches and end nodes would increase the complexity of the mechanism.

Instead, DDBBM will detect congestion at the destination end nodes by considering the number of sources ($NS$) sending traffic to the destination and their injection rates (Source Injection Threshold, $SIT$). In case an end node is congested, most of its received traffic will be generated by few sources. So, some sources will be injecting more traffic than the rest.

Figure 1 shows the detection mechanism with all the parameters to be considered. A register is used per each different threshold to be considered ($DT$ , $LT$, $NS$, and $SIT$). Also, additional registers are used for the time frame ($TF$) in cycles, the overall number of bytes received (Overall bytes received, $ORB$), and a bank of registers for the number of bytes received per source (Source bytes received, $SRB$).

The mechanism works as follows. At the start of a time frame, the counter is set with the $TF$ value and the $ORB$ and all the $SRB$ registers are set to zero. During the time

frame, the counter is decremented cycle by cycle and the reception rate is quantified. Whenever a new packet arrives, the $ORB$ register and the corresponding $SRB$ register are updated accordingly.

When the counter reaches zero a time frame expires and the detection mechanism is triggered. At this point, the comparers are enabled. In particular, $ORB$ and $DT$ register are compared. Also, all the $SRB$ registers are compared in parallel with the $SIT$ register. Then, the number of $SRB$ registers with larger values than the $SIT$ register is counted, and the sum of them compared with the $NS$ register. With all this processing, the final decision on congestion is made. If the number of bytes received ($ORB$) is higher than the $DT$ register and the number of sources sending too much data is higher than the $NS$ register, then the end node is considered as congested. In that situation, the congestion bit ($CB$) is set. However, if the number of bytes received ($ORB$) is lower than the $LT$ register, the end node is not considered congested anymore and the $CB$ bit is reset. Regardless of the detected situation, at this point in time a new time frame is started and all the process repeated.

### 3.2 Notification of Congestion

Once an end node detects the start or the end of congestion (the $CB$ bit changes), it notifies the new situation to the sources that have sent packets recently. This information is available in the detection mechanism logic ($SRB$ register with non-zero value). While the $CB$ bit is set, whenever the end node receives a packet from a new source, it will send a notification to the source.

When end of congestion is detected ($CB$ bit is reset), the end node only notifies the sources that were previously notified about the congestion. Since the information located in the detection mechanism is reset each time frame, and the congested situation may last several time frames, the end node needs a data structure to keep the information regarding the notified sources. For this reason, a notification bit vector (referred to as $NBV$) is used at each end node. The notification bit vector has as many elements (bits) as end nodes in the system. Once congestion is detected, the end node sends a notification to all the end nodes that have sent traffic within the time frame ($SRB$ register with a non zero value). At the same time, the corresponding bits in the notification vector are set. During the time the $CB$ bit is set, if a new source sends data to the end node, it will be notified, and its corresponding bit will be set. When the end of congestion is detected ($CB$ bit is reset) a notification will be sent to those end nodes with their bits set in the notification vector. At the same time, the corresponding bits in the notification vector will be reset. DDBBM will use an specific control packet to notify the current congestion situation (referred to as a notification packet). Figure 1 shows the notification logic with all the parameters to be considered.

### 3.3 Injecting and Mapping Packets

Each end node has a bit vector (referred to as status bit vector) with as many bits as end nodes on the network. In the status bit vector, each bit corresponds to an end node status. When the bit for an end node is set, it means that the end node is considered as a congested destination. Initially all the end nodes are considered as non-congested. When an end node receives a notification packet (meaning start or end of congestion) it commutes the bit of the end node that sent the notification. Thus, the *status bit* mirrors the $CB$ bit at destination.

When an end node has data to transmit, a data packet is built and injected. However, a new bit at the header (the *congestion bit*) is used. The value of the congestion bit is stamped directly from the status bit vector. Therefore, only packets addressed to a congested destination have activated their congestion bit.

Switches must identify packets going to a congested destination in order to separate them from the rest of traffic. Congested flows must be mapped to the dynamic queue, while the other flows must be mapped to the DBBM queues. The allocation decision is taken based on the *congestion bit* of each arriving packet. If the bit is set to zero, the packet is mapped to a DBBM queue (DBBM modulo mapping). However, if the bit is set to one, the received packet is mapped to the dynamic queue.

The dynamic queue can be flow controlled in the same way as the other queues are. One option is to use separate credit counters for each queue, and injecting the packet to the next switch based on the selected queue. Another option is using credit-based flow control for the entire memory and status-based flow control (e.g. on/off) for each queue.

### 3.4 Out of order delivery issues

Since traffic is dynamically separated, out of order issues may appear. Out of order may be present only among packets for the same flow (source-destination pair). In DBBM, such packets are mapped to the same set of queues, thus they arrive at the destination in the same order they were injected. However, in DDBBM, two consecutive packets injected from the same source to the same destination may arrive out of order, simply because the second packet is separated from the normal traffic and thus mapped to the dynamic queue, thus potentially advancing at a faster rate than the previous packet.

In order to guarantee in order delivery (if required), the sender must ensure that packets belonging to the same message are handled in the same way. For this reason, if the sender receives a notification while a message is being injected (some packets belonging to that message have already been sent), it will not modify the *congestion bit* in the remaining packets of that message. The new value will be applied to packets belonging to the next message.

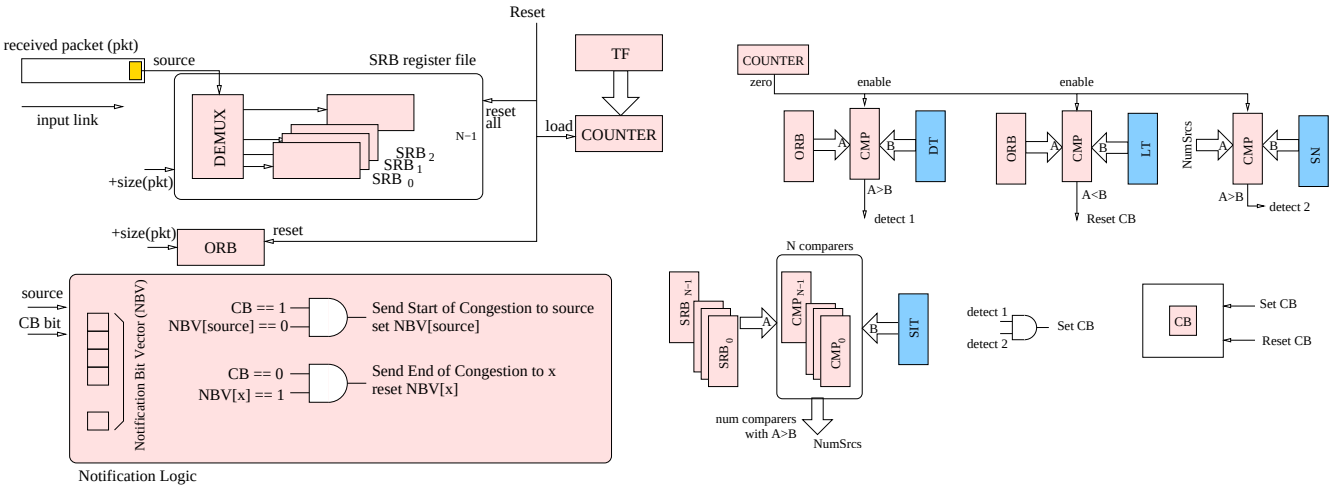


**Figure 1. Logic for measuring, notifying, and detecting congestion.**

## 4 Implementation of DDBBM on InfiniBand

One of the goals of the paper is to adapt DDBBM to InfiniBand (IBA) with no switch modification. Indeed only tables provided by the standard and located at switches will be modified.

In IBA, routing and virtual channel (they are referred to as Virtual Lanes, VLs) selection is based on the destination local ID (DLID) and the service level (SL) fields of the packet header. These two fields are computed at the source node and do not change along the path. Every switch has a forwarding table which provides only one output port (and always the same) for each destination. Thus, IBA implements deterministic routing.

Up to 15 data virtual lanes can be implemented in IBA. Virtual lane selection is based on the use of service levels (SLs). By means of SLtoVL mapping tables, located on every switch, SLs are used to select the proper VL at each switch. This table returns, for a given input port and a given SL, the VL to be used at the corresponding output port. For this, the SL is placed at the packet header and cannot be changed by the switches. Therefore, we should also assign the proper SL that must be used for a given path. VLs and SLs were initially defined in IBA for providing QoS and deadlock avoidance.

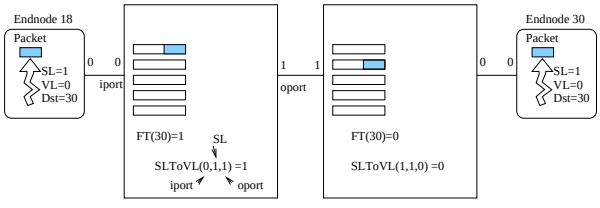


**Figure 2. Example of the VLs used in IBA.**

Figure 2 shows a routing example in InfiniBand. Whenever a packet is injected into the network, the end node (end node 18) computes the SL of the packet (there are up to 16 SLs) and the initial VL to use. In the example, the SL is set to 1 and the initial VL is set to 0. At the first switch the VL used to map the packet is VL0 (as the initial VL was set to zero). At that switch the next VL to use is changed according to the SLtoVL table. In particular, the SLtoVL table at the first switch indicates that at the next switch the VL to use will be VL1. Notice also that the output port selected at each switch is extracted from the forwarding table (FT function in the example).

To implement DDBBM we will use SL identifiers and different VLs. A particular VL (referred to as $VL_i$) can be used along any path by using a SL identifier (referred to as $SL_i$) and all the SLtoVL tables being programmed in such a way that whenever the SL of the packet is $SL_i$, regardless of the input and output ports, the provided VL is $VL_i$. Thus, if we want to implement DDBBM with 5 queues (4 queues for DBBM, and one for the dynamic queue) we must reserve 5 SL identifiers and fill the corresponding SLtoVL tables.

Now, at the end nodes we need to compute for each packet the proper SL. We need to implement the status bit vector and the modulo selection at the end node. For efficiency reasons the vector should be included at the network interface. Simply whenever a packet is going to be injected, the status bit for the corresponding destination is inspected. If the bit is set, then the SL that forces the use of the dynamic queue ($SL_4$ for a DDBBM implementation with 5 queues (VL0 through VL4)) is selected. If not, then lower bits of the destination (modulo mapping) indicate the SL. In our example, the lower two bits. Notice that the selection may be implemented in the subnet manager located on every channel adapter in InfiniBand.

Finally, we need to implement the detection mechanism at the end nodes and the notification of congestion. Regard-

ing the detection mechanism, it must be implemented at the network interfaces. Again, at the end nodes we may rely on the subnet manager in order to keep the statistics of traffic and to issue notification packets to the end nodes. For notifications we can use the VL15. This virtual lane is reserved for control packets in InfiniBand.

# 5 Performance Evaluation

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

## 5.1 Simulation Model

The simulator models an ICTN with switches, end nodes, and links. Buffers up to 1KB are modeled for both the input and the output ports of every switch. The buffer capacity is statically divided by the number of queues defined by each of the evaluated mapping schemes, resulting in a fixed size per queue.

At every switch packets are forwarded from any input queue to any output queue through a multiplexed crossbar. We have considered a crossbar bandwidth of 1.5 GB/s (speedup of 1.5 when compared to link bandwidth). The crossbar is controlled by a scheduler that receives requests from the packets at the head of any input queue. A requesting packet is forwarded only if the corresponding crossbar input and crossbar output are free. At each output port a weighted round-robin arbiter selects the output queue to be served.

For links we assume serial full-duplex pipelined transmissions with 1 GB/s raw bandwidth. The link-level flow control (LL-FC) protocol is credit-based; a packet can be transmitted downstream only if a credit is available. Whenever a packet frees an input buffer location a new credit is sent to the output port upstream. A similar flow control scheme has been implemented for the internal switch traversal (input-output packet forwarding). The maximum number of credits per output (input) port depends on the buffer size at the next input (output) port and the total number of queues. The LL-FC packets share the link bandwidth with data traffic.

The end nodes are connected to switches using Input Adapters (IAs). Every IA is modeled by (i) a *fixed* number $N$ of message *admittance* queues organized in VOQ; (ii) and a *variable* number of *injection* queues organized similarly to the output ports of a switch. When a new message is generated, first it is stored completely in the admittance queue assigned to its destination; then it is segmented into 64-byte packets before being transferred to an injection queue. The transfer from admittance queues to injection queues are controlled by a round-robin arbiter. The transmission of packets from injection queues into the network is controlled by a weighted round-robin arbiter.

## 5.2 Topologies and Traffic Patterns

DDBBM will be evaluated in different bidirectional multistage networks (BMINs) and 2D/3D meshes. In particular, $64 \times 64$, $512 \times 512$, and $1024 \times 1024$ BMINs will be used, each one built using 8-port switches interconnected in a perfect shuffle connection pattern. The routing algorithm is deterministic. For the regular topologies, $4 \times 4$, $8 \times 8$, $16 \times 16$, $8 \times 8 \times 4$, and $4 \times 4 \times 4$ meshes will be evaluated. In this case we will use the Dimension Order Routing (DOR). Evaluated topologies are shown in Table 1, where cases #XB represent BMINs, and cases #XM represent meshes.

| | Traffi c to: | random dests | hotspot |
|---|---|---|---|
| Case | Network Evaluated (end nodes) | % of injecting sources (IR) | % of injecting sources (IR) [destinations] |
| #1B | $64 \times 64$ (4) | 100% (100%) | - (-) [-] |
| #2B | $64 \times 64$ (4) | 70% (60%) | 30 % ( 60%) [1] |
| #3B | $64 \times 64$ (4) | 70% (60%) | 30 % (100%) [1] |
| #4B | $64 \times 64$ (4) | 70% (90%) | 30 % ( 60%) [1] |
| #5B | $64 \times 64$ (4) | 70% (60%) | 30 % (100%) [5] |
| #6B | $512 \times 512$ (4) | 70% (60%) | 30 % ( 60%) [1] |
| #7B | $1024 \times 1024$ (4) | 70% (60%) | 30 % ( 60%) [1] |
| #1M | $8 \times 8$ (1) | 100% (20%) | - (-) [1] |
| #2M | $4 \times 4$ (4) | 100% (20%) | - (-) [1] |
| #3M | $8 \times 8$ (1) | 70% (20%) | 30% (100%) [1] |
| #4M | $8 \times 8 \times 4$ (1) | 70% (20%) | 30% (100%) [1] |
| #5M | $4 \times 4$ (4) | 70% (20%) | 30% (100%) [1] |
| #6M | $16 \times 16$ (1) | 70% (20%) | 30% (100%) [1] |
| #7M | $8 \times 8$ (4) | 70% (20%) | 30% (100%) [1] |
| #8M | $4 \times 4$ (16) | 70% (5%) | 30% (100%) [1] |
| #9M | $4 \times 4 \times 4$ (4) | 70% (20%) | 30% (100%) [1] |

**Table 1. Topologies and traffic patterns. IR means Injection Rate per end node.**

For the BMIN networks we have defined 7 different synthetic traffic patterns (shown in Table 1). In #1B case all the sources inject at full injection rate (100%) with a uniform distribution of packet destinations. Cases #2B-#7B introduce at least a congestion tree by oversubscribing a hotspotted end node: 30% of sources injecting to the same randomly selected hotspot destination. The rest of traffic (background traffic) is made of the remaining sources (70%) injecting traffic to randomly-selected destinations. Cases #2B through #7B differentiate by the injection rates of end nodes and the number of hotspots. As the background traffic shares links and queues with the flows belonging to the congestion tree, substantial HoL blocking will be introduced in multiple switches.

On the other hand, in regular networks we have defined 9 different scenarios based on synthetic traffic patterns (see Table 1). All the cases cause a congestion tree by oversubscribing the hotspotted end node. In this case, the background traffic is made of 70% of the sources injecting to randomly selected destinations, while the remaining 30% of sources injecting to a common randomly-selected hotspot destination.

We will analyze the behavior of DDBBM with different number of queues. For comparison purposes we will also evaluate the $VOQ_{net}$, $VOQ_{sw}$, and the DBBM mapping policies. For DBBM we will also use different number of queues.

## 5.3 Evaluation Results

### 5.3.1 DDBBM Sensitivity to Thresholds

In order to evaluate the robustness of DDBBM to $DT$, $LT$, and $SIT$ parameters, cases #1B, #3B, #2M and #5M have been evaluated with the following values: $DT$ 0.8 and 0.95, $LT$ 0.2 and 0.4, $SIT$ 0 and 6. For all the cases, the $NS$ parameter has been set to one. The $DT$ and $LT$ parameters are expressed as percentages of the maximum reception rate. However, the $SIT$ parameter is expressed as the number of times the source exceeds the injection rate of $100/N$ where $N$ is the number of end nodes. For instance, $SIT = 6$ means that the destination must receive $(600/N)\%$ of the total received traffic from a particular source to decide that this source is contributing to the congestion. A value of zero means that the $SIT$ parameter is disabled.

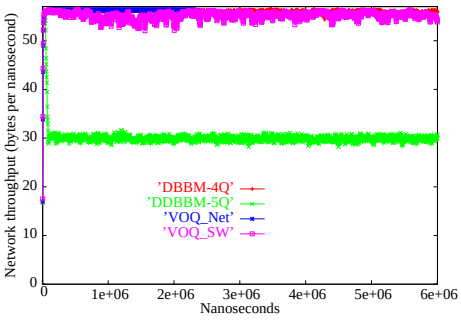


**Figure 3. Accepted traffic. #1B case. DT = 0.95, LT = 0.2, SIT = 0.**

Figure 3 shows results for the #1B case with $DT = 0.95$, $LT = 0.2$, and $SIT = 0$. In this case, all the evaluated mapping schemes except DDBBM achieve the maximum throughput. In this case, as the $SIT$ parameter is disabled, most of the end nodes detect congestion (although uniform traffic is being used). All the end nodes receiving at a rate higher than 95% of their link bandwidth will be considered as a hotspotted end node, being false positives. After notifying sources, all the injected packets are mapped to the dynamic queue thus introducing massive HoL blocking. Therefore, DDBBM exhibits poor performance levels. We have analyzed topology/traffic case #1B with increasing values of $SIT$. Results show that as we increase the value of $SIT$, DDBBM performance increases. Concretely, when $SIT$ is set to 6, DDBBM achieves the same performance as $VOQ_{net}$, as the false positives detected by end nodes are filtered. Figure 4 shows results for topology/traffic case #1B with $SIT$= 6.

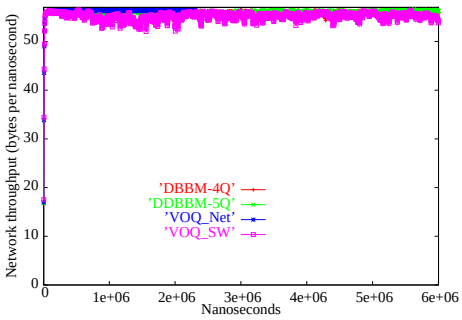


**Figure 4. Accepted traffic. #1B case. DT = 0.95, LT = 0.2, SIT=6.**

We have analyzed all the combinations for $DT$, $LT$, and $SIT$ thresholds. The obtained results (not shown) are similar for the presented case, showing that $DT$ and $LT$ thresholds have low impact on network performance. The same evaluation has been performed with topology/traffic cases #3B, #2M and #5M (not shown). Again, no significant differences were encountered. Therefore, we conclude that the DDBBM mechanism is largely insensitive to $DT$, and $LT$ parameters. Only with high injection rates with uniform traffic there exists a dependence on the $SIT$ parameter. Based on these results, in the following evaluations we will fix the parameters to the following values: $DT$=0.95, $LT$=0.2, and $SIT$=6.

### 5.3.2 Fairness of DDBBM

In this Section we will analyze the unfairness introduced by the different mapping policies. From previous works we know that DBBM introduces certain levels of unfairness to some destinations. This is because DBBM maps different flows to the same queue, and thus HoL blocking is not eliminated completely. Notice that this problem is the main motivation for designing DDBBM.

In Figure 5 we can see the traffic received by each end node for topology/traffic case #3B ($64 \times 64$ BMIN) when $VOQ_{net}$, $VOQ_{sw}$, DBBM-4Q and DDBBM-5Q schemes are used. The end node with the highest traffic reception rate corresponds to the hotspot (end node 30), which reaches 90% of reception rate (axes are truncated at 50%).

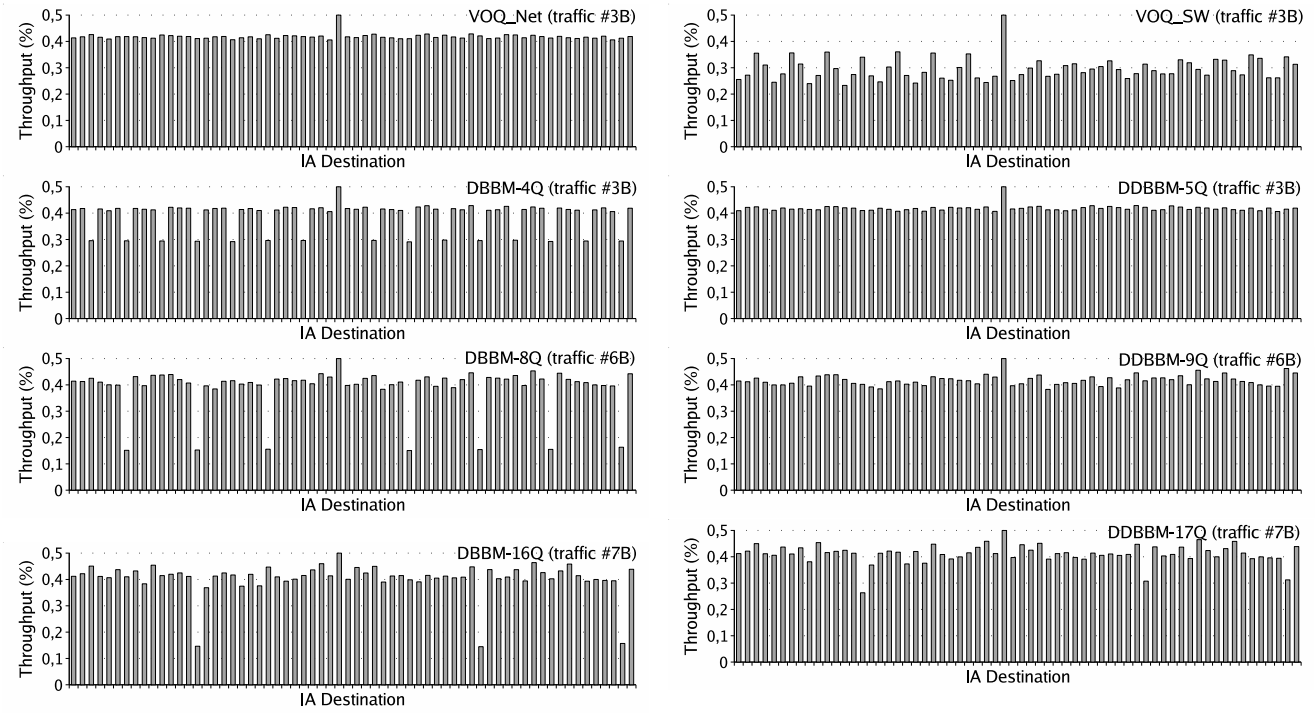


**Figure 5. BMIN Accepted traffic per destination.**

With $VOQ_{net}$ every destination, except the hotspot, receives roughly the same goodput. However, with $VOQ_{sw}$, all the flows that share any queue along their path with the congested flow suffer from HoL blocking. The bar graph of $VOQ_{sw}$ in Figure 5 shows that every 4 consecutive end nodes similar percentages of accepted traffic are obtained. This pattern is consequence of the number of links from a switch connecting the next stage $(4)$ together with the routing algorithm applied. The routing algorithm will determine the output link for each packet and consequently which packets will be mapped to the same queue as the packets addressed to the congested destination. Hence the reduction in the number of received packets by the *victimized* destinations. With DBBM (DBBM-4Q bar graph in Figure 5) the number of affected flows depends on the number of queues, but neither on routing nor the number of links per switch.

Notice that in $VOQ_{sw}$ none of the end nodes is receiving the expected traffic (0.4%), thus all of them experience HoL blocking. However, DDBBM-5Q completely eliminates the HoL blocking, and hence the unfairness. The DDBBM-5Q bar graph in Figure 5 shows that the traffic received by each end node is roughly the same. Therefore, the HoL blocking experienced by DBBM has been completely eliminated by DDBBM. Thus, the detection mechanism has identified correctly the congested destination (end node 30) and the packets addressed to the congested destination have been mapped to the dynamic queue.

Bar graphs of DBBM-8Q/16Q and DDBBM-9Q/17Q in Figures 5 show similar results with topology/traffic cases #6B and #7B. In these cases, the mechanism is much more stressed as the network is larger ($512\times512$ and $1024\times1024$ BMIN networks). However, only in case #7B ($1024 \times 1024$ BMIN) DDBBM shows some kind of unfairness. This is because although congestion tree is isolated, the reduced number of queues applying DDBBM scheme respect to the total number of end nodes will introduce a slight HoL blocking among the non congested flows.

Fairness has been also analyzed in 2D/3D meshes. In this case, similar results have been obtained. Figure 6 shows received traffic for each destination over the different mapping schemes ($VOQ_{net}$, $VOQ_{sw}$, DBBM, and DDBBM, respectively) for topology/traffic cases #5M ($4\times4$ mesh with 64 end nodes), and #8M ($4 \times 4$ mesh with 256 end nodes). In all the analyzed cases (cases #5M, #7M (not shown), #8M, and #9M (not shown)) DDBBM successfully eliminates all the unfairness that would be introduced by the congested destination. This behavior confirms that DDBBM does not present unfairness regardless of the topology and network size.

To conclude, results show that only when packets addressed to the congested destination are isolated in a queue, unfairness is eliminated. Excepting $VOQ_{net}$ and DDBBM schemes, all the other schemes introduce some degree of unfairness under high load and congestion. However, DDBBM with a very low number of queues eliminates un-

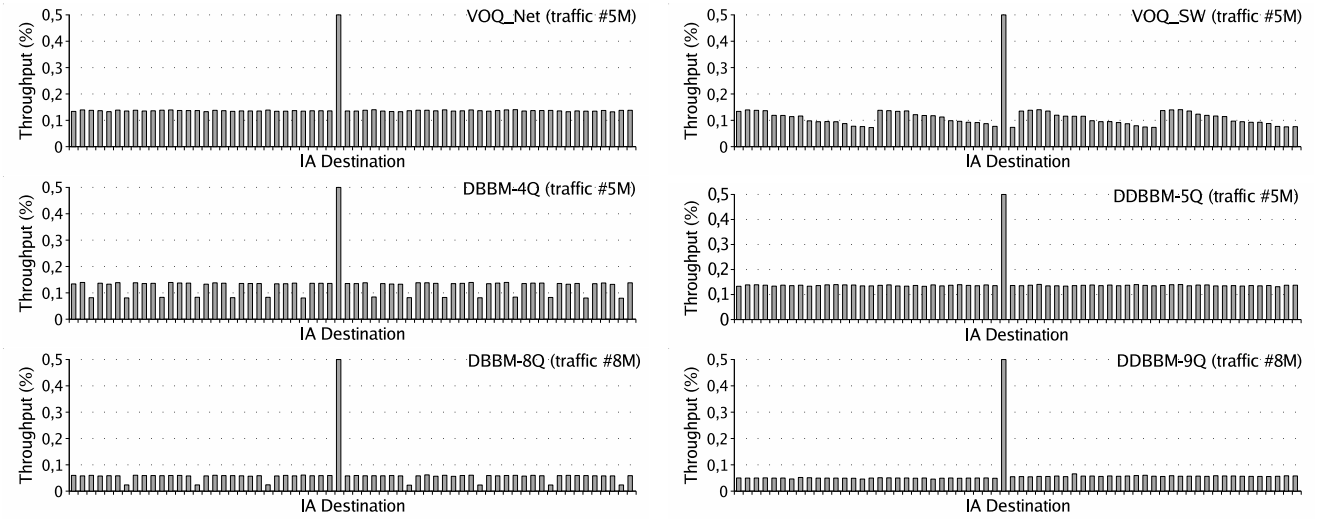


**Figure 6. Mesh Accepted traffic per destination.**

fairness completely.

### 5.3.3 Throughput Analysis

Finally we evaluate DDBBM in terms of achieved network throughput. Figure 7 shows the throughput achieved for BMIN cases (#2B, #3B, #4B, and #5B).

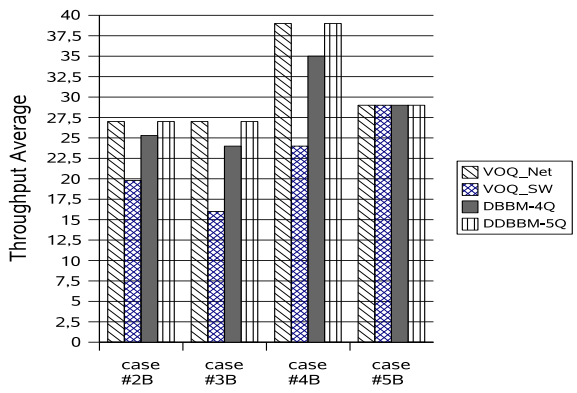


**Figure 7. Throughput for BMIN cases.**

For case #2B we can observe that DDBBM-5Q achieves the maximum throughput whereas DBBM-4Q and $VOQ_{sw}$ show lower throughput numbers. $VOQ_{sw}$ achieves 70% of the DDBBM throughput while DBBM-4Q achieves 92%. Similarly, for topology/traffic cases #3B and #4B, DDBBM-4Q performs equal than $VOQ_{net}$, and DBBM-4Q achieves 88% and 89% of DDBBM throughput, respectively, whereas $VOQ_{sw}$ only achieves 59% and 61% of DDBBM throughput. Finally, in case #5B, where 5 lower intense congestion trees are formed, all of the evaluated mapping schemes achieve the maximum throughput. As a summary, DDBBM-5Q achieves $VOQ_{net}$ throughput in all the hot spot evaluated cases.

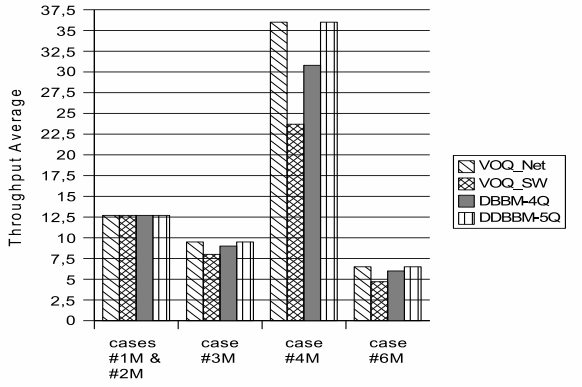


**Figure 8. Throughput for mesh cases.**

For 2D meshes, Figure 8 shows the throughput achieved for topology/traffic cases #1M, #2M, #3M, #4M, and #6M. We see that topology/traffic cases #1M and #2M show equal characteristics. All the mapping schemes achieves the maximum performance. This is because the network is able to absorb the injected traffic at the rate that is injected (not hotspotted end node appears). However, when a congestion tree is formed, as in cases #3M, #4M, and #6M, results show that only DDBBM is able to achieve $VOQ_{net}$ throughput. Likewise, we see that although DBBM is not able to reach $VOQ_{net}$ throughput, it shows better performance than $VOQ_{sw}$ as this method perform the worst in all the cases. Notice that this behavior keeps despite the increase in the number of end nodes (cases #4M and #6M).

These results confirm previous conclusions from BMIN networks: regardless of the topology DDBBM is able to reach maximum performance while using a reduced set of queues.

It has to be noted that we also evaluated the DDBBM mechanism with real traffic (extracted from IP and SAN traces). In all the cases, the DDBBM mechanism achieved maximum performance (the one achieved by $VOQ_{net}$) with very low queue requirements, and completely eliminating the unfairness introduced by DBBM.

## 6 Conclusions

In this paper we have proposed a mechanism able to dynamically eliminate the HoL blocking caused by congested end nodes. This is achieved by designing a detection mechanism at the end nodes and then notifying sources in order to separate the congested traffic. The mechanism has been designed in order to simplify the switch design. Therefore, it allows its use on commercial products. In this sense, we have presented a method to apply DDBBM in InfiniBand without hardware modification.

The results presented clearly demonstrate that the proposed scheme, DDBBM, achieves similar performance than $VOQ_{net}$ scheme but with much fewer resources, and better performance than the other analyzed schemes (DBBM and $VOQ_{sw}$). These good results are in terms of throughput and fairness, under a wide range of traffic patterns and topologies. Also, DDBBM sensitivity to threshold values has been evaluated. Results have shown that only under high uniform traffic and BMIN topologies there exists a dependence on the $SIT$ parameter. However, with a fine tuning of the parameter, the adverse case is solved. Also, detection thresholds ($DT$ and $LT$) are highly insensitive to DDBBM final performance.

As future work we plan to implement the DDBBM mechanism on a real system. For this, we plan to implement the mechanism on InfiniBand and to use real InfiniBand platforms for its test.

## References


[1] G. S. Almasi and 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," in *ACM Trans. Computer Systems*, Nov. 1993.

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

[4] R. Bianchini, T. J. LeBlanc, L. I. Kontothanassis, and M. E. Crovella, "Alleviating Memory Contention in Matrix Computations on Large-Scale Shared-Memory Multiprocessors," *Tech. report 449*, Computer Science Dept., Rochester University, April 1993.

[5] IBM BG/L Team, "An Overview of BlueGene/L Supercomputer," in *High Performance Networking and Computing (SC2002)*, Nov. 2002.

[6] N.J. Boden et al, "Myrinet - A gigabit per second local area network," in *IEEE Micro*, Feb. 1995.

[7] T. Nachiondo, J. Flich, and J. Duato, "Efficient Reduction of HOL blocking in Multistage Networks," in *Proc. 2005 Int. Parallel and Distributed Processing Symposium*, April 2005.

[8] W. J. Dally and H. Aoki, "Deadlock-Free Adaptive Routing in Multicomputer Networks Using Virtual Channels," in *IEEE Trans. on Par. and Distr. Systems*, vol. 4, no. 4, April 1993.

[9] Earth Simulator Center. http://www.es.jamstec.go.jp/esc/eng/index.html.

[10] J. Duato, J. Flich, and T. Nachiondo, "Cost-Effective Technique to Reduce HOL-blocking in Single-Stage and Multistage Switch Fabrics," in *Proc. Euromicro Conf. on Par., Distr. and Network-based Processing*, Feb. 2004.

[11] T. Nachiondo, J. Flich, J. Duato, and M. Gusat, "Cost/Performance Trade-offs and Fairness Evaluation of Queue Mapping Policies," in *Proc. International Euro-Par Conference*, Lisbon, Aug. 2005.

[12] J. Duato, I. Johnson, J. Flich, F. Naven, P. Garcia, and T. Nachiondo, "A New Scalable and Cost-Effective Congestion Management Strategy for Lossless Multistage Interconnection Networks," in *Int. Symp. on High-Performance Comp. Arch.*, Feb. 2005.

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

[14] N. McKeown, "Scheduling algorithms for input-queued cell switches," Ph.D. Thesis, University of California at Berkeley, 1995.

[15] "Advanced Switching for the PCI Express Architecture," White paper.

[16] S. L. Scott, and G. S. Sohi, "The Use of Feedback in Multiprocessors and Its Application to Tree Saturation Control," in *IEEE Trans. on Parallel Distr. Systems*, vol. 1, no. 4, Oct. 1990.

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

[18] P. Yew, N. Tzeng, and D. H. Lawrie, "Distributing Hot-Spot Addressing in Large-Scale Multiprocessors," in *IEEE Trans. Computers*, vol. 36, no. 4, April 1987.