Routing in InfiniBand^{TM} Torus Network Topologies [*]

J. C. Sancho, A. Robles, P. López, J. Flich, and J. Duato
Departamento de Informática de Sistemas y Computadores
Universidad Politécnica de Valencia
P.O.B. 22012, 46071 - Valencia, SPAIN
E-mail: {jcsancho,arobles,plopez,jflich,jduato}@gap.upv.es

Abstract

InfiniBand is an interconnect standard for communication between processing nodes and I/O devices as well as for interprocessor communication (NOWs). The InfiniBand Architecture (IBA) defines a switch-based network with point-to-point links whose topology can be established by the customer. When the performance is the primary concern regular topologies are preferred. Low-dimensional tori (2D and 3D) are some of the regular topologies most widely used in commercial parallel computers. Routing in torus requires the use of virtual channels. Although InfiniBand provides support for deterministic routing and virtual channels, they are selected at each switch by service level (SL) identifiers associated to packets and do not depend on packet destination. This makes routing algorithm implementation more complex. In particular, a large number of SLs may be required, which is a scarce resource. In this paper we analyze the way several routing strategies can be applied in tori InfiniBand networks, also evaluating their resource requirements. In particular, we analyze and compare the well-known e-cube and up/down* routing algorithms and the Flexible routing algorithm recently proposed.*

Keywords: *Routing algorithms, InfiniBand networks, torus topologies, clusters, deadlock avoidance.*

1. Introduction

InfiniBand is a new standard for communication developed by many companies, including the computing industry leaders [10]. InfiniBand is designed to solve the lack of high bandwidth, concurrency and reliability of existing technologies for system area networks. Moreover, InfiniBand can be used as a platform to build networks of workstations (NOWs) and clusters of PCs [16] which have become a cost-effective alternative to parallel computers. Currently, clusters are based on different available network technologies (Fast or Gigabit Ethernet [25], Myrinet [1], ServerNet II [8], Autonet [24], etc...). However, they may not provide the protection, isolation, deterministic behavior, and quality of service required in some environments.

The InfiniBand Architecture (IBA) is designed around a switch-based interconnect technology with high-speed serial point-to-point links connecting processor nodes and I/O devices. IBA allows users to decide the network connectivity. The layout of the network can consist of regular or irregular topologies. Regular topologies are often used when performance is the primary concern [17]. This is the case when a large cluster of workstations needs to be designed to run computation intensive applications. The network would fit in a single room and the switches would be in a cabinet. In particular, most of the commercial parallel computers have been built using torus topologies with two (2D) or three (3D) dimensions, such as Intel Cavallino [3], Cray T3D [11], and Cray T3E [23]. Further, recent proposals, such as Alpha 21364 [14] and BlueGene/L [9], use 2D and 3D tori, respectively. Thus, in this paper we restrict our attention to these network topologies.

Routing in IBA is distributed, based on forwarding tables located on each switch which only consider the packet destination ID for routing [10]. IBA routing is deterministic, since routing tables only store one routing option (output port) per destination ID. IBA switches support a maximum of 16 virtual lanes (VL)[1]. VL15 is exclusively reserved for subnet management, whereas the remaining VLs are used for normal traffic. Virtual lanes provide a mean to implement multiple logical flows over a single physical link [5] and are provided to support QoS, traffic priorization, and routing. In order to route packets through a certain virtual lane, packets are marked with a certain Service Level (SL), and SL-to-VL mapping tables are used at each switch to de-

[*]This work was supported by the Spanish Ministry of Science and Technology under grant TIC2000–1151–C07, by Generalitat Valenciana under Grant CTIDIB/2002/288, and by JJ.CC. de Castilla-La Mancha under Grant PBC-02-008.

[1]In what follows, we will use the terms virtual lanes (VL) and virtual channels (VC) indistinctly.

termine the virtual lane to be used. However, VL selection does not only depend on the packet SL, but it also depends on the input and output physical ports through which the packet enters and leaves the switch. Thus, once the output physical port has been provided by the routing table, the virtual lane is obtained by taking into account both the SL of the packet and the input and output ports of the current switch.

Routing in tori can be carried out by using either an specific routing algorithm for k-ary n-cubes, such as the deterministic *e-cube* algorithm [27], or a generic routing algorithm, such as the well-known *up*/down** routing algorithm [24] or the *Flexible* routing algorithm [20]. Although, there are different adaptive routing approaches for tori proposed in the literature [12, 4, 7], they cannot be applied to InfiniBand since it only supports deterministic routing.

The *e-cube* algorithm [27] is especially designed to k-ary n-cubes and meshes, allowing packets to be routed through minimal paths. To avoid deadlocks, this algorithm routes packets in decreasing dimension order. According to the routing methodology proposed by Dally and Seitz [6], this algorithm requires two virtual lanes in torus topologies, which are selected taking into account only the destination ID. However, InfiniBand considers the SL and the input and output ports to select the virtual lane. Hence, SLs must be assigned to packets as a function of their destination, according to some strategy (see Section 2). However, a large number of service levels is required by this routing algorithm to correctly assign virtual lanes in InfiniBand (analyzed in Section 2). Further, this could seriously restrict the appliance of QoS.

On the other hand, generic routing algorithms can be defined on any topology, including torus networks. Generic routing algorithms require some tool that automatically explores the network topology and computes the routing paths. Unlike the *e-cube* algorithm, these algorithms support (with an appropriate reconfiguration protocol [2]), link and switch failures. Additionally, generic routing algorithms can be applied to InfiniBand without requiring the use of virtual lanes nor service levels. However, these routing algorithms have the drawback that they do not provide routing through minimal paths in many cases, specially with large networks. This is because of the large number of routing restrictions imposed to avoid deadlocks.

The *up*/down** routing algorithm [24] is the most popular generic routing algorithm currently used in commercial interconnects. This algorithm is quite simple and easy to implement on any network technology. In [19, 18], a new methodology to compute *up*/down** routing tables was proposed. This methodology is based on obtaining a depth-first search spanning tree (DFS) from the network graph instead of the BFS spanning tree used in the original methodology [24]. The *up*/down** routing scheme based on the DFS spanning tree significantly outperformed the original *up*/down** scheme, but the behavior of BFS and DFS on regular networks was noticeably worse than the one exhibited by the specific routing algorithm [6]. More recently, we proposed the *Flexible* [20] routing algorithm, that is able to significantly outperform *up*/down** routing in regular networks. It only needs one virtual channel and one service level in order to be used on InfiniBand.

In this paper, we perform a comparative analysis of different routing algorithms on InfiniBand networks using torus topologies. In particular, we will consider the *e-cube* algorithm, the improved *up*/down* routing algorithm (using the methodology based on a DFS spanning tree), and the *Flexible* routing algorithm. The main goal of this analysis is to show which is the most cost-effective routing algorithm in tori when using InfiniBand as the network technology. For this aim, we will evaluate the performance of these routing schemes, paying special attention to the network resources required to implement them on InfiniBand networks.

The rest of the paper is organized as follows. Section 2, 3, and 4 describe the *e-cube* algorithm, the *up*/down** routing scheme, and the *Flexible* routing algorithm, respectively, also dealing with their implementation on IBA. Section 5 analyzes the routing algorithms previously described for tori by computing some behavioral routing metrics. In section 6, the IBA simulation model is described. It also shows the performance evaluation results. Finally, some conclusions are drawn.

2. Applying Dimension-Order Routing in InfiniBand

In [6], Dally and Seitz extended the well-known *e-cube* algorithm [27] for the binary n-cube to the k-ary n-cube or torus. As known, the *e-cube* algorithm routes packets in decreasing dimension order to avoid deadlocks. As stated in [6], when applied to tori, the *e-cube* algorithm requires two virtual channels to remove the cyclic channel dependencies introduced by the wraparound channels. Additionally, the use of bidirectional channels allows packets to be routed through minimal paths. To illustrate how the *e-cube* algorithm works in tori, let us assume that each physical channel is split into two virtual lanes (VL0 and VL1). A packet arriving at a node n_c and destined to node n_d, with coordinates[2] $n_{c_{n-1}}, n_{c_{n-2}}, ..., n_{c_1}, n_{c_0}$ and $n_{d_{n-1}}, n_{d_{n-2}}, ..., n_{d_1}, n_{d_0}$, respectively, will be routed through the physical channel belonging to the dimension i, where i is the position of the most significant digit in which addresses n_c and n_d differ. In each dimension i, packets will

[2]In a k-ary n-cube network, the address of a node x is formed by n coordinates $(x_{n-1}, x_{n-2}, ..., x_1, x_0)$, where $0 \leq x_i \leq k-1$ for $0 \leq i \leq n-1$.

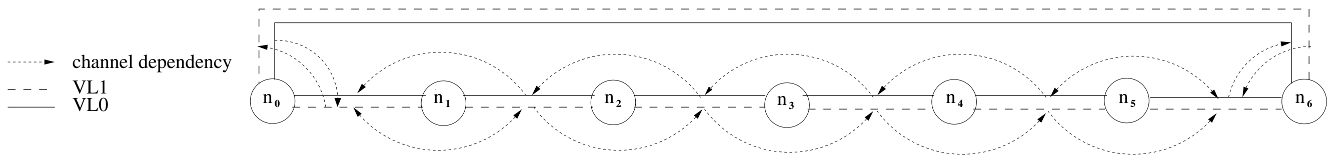


Figure 1. Channel dependencies between virtual lanes on a 1×7 torus when applying the e-cube routing algorithm.

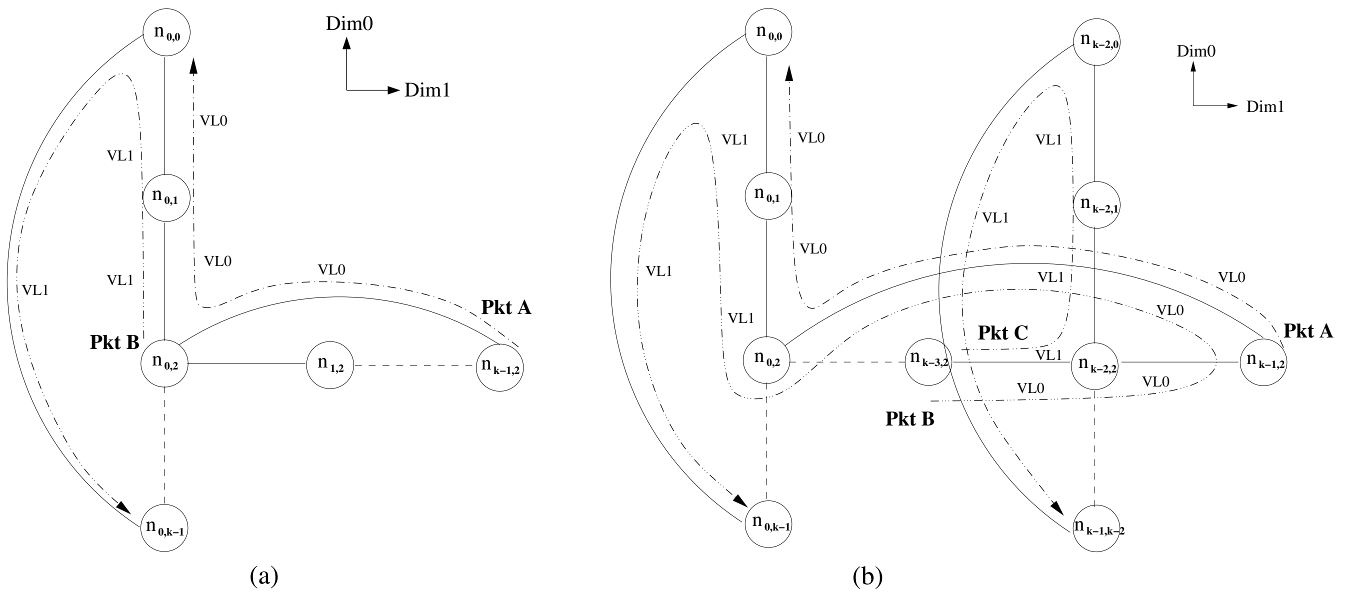


Figure 2. Conflicts on service levels in a 2D torus. (a) Packet B using wraparound channel in dimension 0 conflicts with packet A using wraparound channel in dimension 1, and (b) packet C using wraparound channel in dimension 0 conflicts with packet B using wraparound channels in dimensions 1 and 0.

be routed through VL1 if the i^{th} digit of the destination address is greater than the i^{th} digit of the current node address. Otherwise, the packet will be routed through VL0.

As an example, Figure 1 shows the channel dependencies between virtual lanes on a bidirectional torus with 7 nodes when the Dally and Seitz's routing proposal is used. As can be seen, the cyclic channel dependency $n_0 \rightarrow n_1 \rightarrow n_2 \rightarrow n_3 \rightarrow n_4 \rightarrow n_5 \rightarrow n_6 \rightarrow n_0$ among virtual lane VL1 is broken at node n_6, since packets from nodes n_i destined to nodes n_j, so that $j < i$, are routed through virtual lane VL0. Moreover, the cyclic channel dependency $n_6 \rightarrow n_5 \rightarrow n_4 \rightarrow n_3 \rightarrow n_2 \rightarrow n_1 \rightarrow n_0 \rightarrow n_6$ among virtual lane VL0 is broken at node n_0, since packets from nodes n_i destined to nodes n_j, so that $j > i$, are routed through virtual lane VL1. Therefore, cyclic channel dependencies between virtual lanes belonging to the same dimension are removed and, hence, deadlocks are avoided. The extension of the *e-cube* routing algorithm to torus networks with more than one dimension is also deadlock-free as dimensions are crossed always in decreasing order.

When the Dally and Seitz's routing methodology is being applied to InfiniBand, it must be taken into account that virtual lanes cannot be assigned as a function of the destination node ID. InfiniBand considers the input port, the output port, and the service level (SL) of the packet to select them. A possible solution to this problem is to reserve service levels and use them appropriately in order to distinguish different routing cases in the *e-cube* algorithm. To illustrate the idea, node n_5 in Figure 1 must distinguish between packets arrived from node n_4 destined to node n_0, which have to use VL0, and packets arrived from node n_4 destined to node n_6, which have to use VL1. As can be seen, both packets use the same input and output port at node n_5. Therefore, these packets must be assigned a different service level (SL) in order to distinguish them in IBA. An easy criterion to assign SLs to packets could be the following: packets which use the wraparound channel (e.g. packets from node n_4 destined to node n_0) should use SL1, otherwise they should use SL0. Moreover, the SLtoVL mapping tables are computed in such a way that packets with SL$_i$ use the appropriate VL on every switch output port. To sum up, the assignment of SLs is not made based on the VLs to be used by the packet, but depending on whether the packet has to traverse the wraparound channel or not. Note that packets that traverse wraparound channels are forced to make a VL transition. Therefore, two service levels are needed to correctly distinguish virtual lanes for one-dimensional torus according to the Dally and Seitz's routing methodology.

However, in tori with larger number of dimensions, a larger number of service levels is required, as can be observed in Figure 2(a) for a 2D torus. Let us consider two packets. The first one, B, crosses channels belonging to dimension 0, and the second, A, crosses from dimension 1 to dimension 0. Both packets (A and B) should be assigned SL1 according to the criterion described above, since they are crossing a wraparound channel. Moreover, at node $n_{0,1}$, packet A has to use VL0 whereas packet B has to use VL1. However, these packets cannot distinguish the VL to be used as long as they have assigned the same service level SL1. Therefore, an additional service level is required in order to distinguish the virtual lanes to be used by these packets. At first sight, in order to assign service levels to packets, we can extend the previous criterion to the following: use SL0 if no wraparound channel is going to be used by the packet, use SL1 if one wraparound channel is going to be used in dimension 1, or use SL2 if, on the contrary, a wraparound channel is going to be used in dimension 0.

However, three service levels are not enough, as packets may traverse wraparound channels in both dimensions. To illustrate this case, let us analyze Figure 2(b). Now, packet B traverses wraparound channels in dimensions 1 and 0, whereas packets A and C only traverse wraparound channels in dimensions 1 and 0, respectively. As stated above, packets A and C should have assigned SL1 and SL2, respectively. If packet B would have been assigned SL2, there would exist no conflict between packets A and B. However, there would be a conflict between packets B and C at node $n_{k-3,2}$. This is because the service level of both packets is the same (SL2) and they must be routed through different virtual lanes (packet B through VL0 and packet C through VL1). Therefore, an additional SL is needed. In particular, packet B should be assigned SL3 because it traverses wraparound channels in both dimensions.

In summary, in order to apply the *e-cube* algorithm in 2D tori four service levels are required. Service levels are

Service level	Dimension 1	Dimension 0
SL0	no wraparound	no wraparound
SL1	wraparound	no wraparound
SL2	no wraparound	wraparound
SL3	wraparound	wraparound

Table 1. Service level assignment based on the use of the wraparound channel on each dimension in a 2D torus.

assigned depending on whether packets have to traverse or not the wraparound channel in each of the dimensions they cross, as it is illustrated in Table 1. The same reasoning can be easily generalized to tori with an arbitrary number of dimensions. The number of service levels required in n-dimensional tori will be 2^n. In particular, for 3D tori, 8 service levels will be required. This value is very significant as long as InfiniBand only provides 15 SLs and they are mainly intended for QoS. Therefore, in many cases, the number of available SLs in InfiniBand could not be enough to fulfill the requirements for deadlock avoidance in torus networks when using the *e-cube* routing algorithm.

3. Up*/down* Routing

Up/down** routing is the most popular routing scheme currently used in commercial networks, such as Myrinet [1]. Unlike the *e-cube* algorithm, *up*/down** routing is a generic routing algorithm valid for any network topology.

The *up*/down** routing algorithm avoids deadlocks by restricting routing in such a way that cyclic channel dependencies are avoided. In order to avoid deadlocks while still allowing all links to be used, *up*/down** builds an spanning tree and assigns a direction ("up" or "down") to each output port based on the spanning tree. To compute the final paths it uses the following rule: a legal route must traverse zero or more links in the "up" direction followed by zero or more links in the "down" direction.

In order to compute the *up*/down** routing tables, different methodologies can be applied. These methodologies differ in the type of spanning tree to be built. The original methodology is based on BFS spanning trees, as it was proposed in Autonet [24], whereas an alternative methodology is based on DFS spanning trees [19].

The DFS methodology provides more minimal paths and a better traffic balance than the BFS one, resulting in a significant increase in network performance (both in regular [20] and irregular [18] networks). Like in the BFS spanning tree, an initial switch must be chosen as the root before starting the computation of the DFS spanning tree. The selection of the root is made by using heuristic rules [18]. In particular, the root with the highest average topological distance to the rest of the switches will be selected as the root node. The rest of switches are added to the DFS spanning tree following a recursive procedure. Unlike the BFS spanning tree, adding switches is made by using heuristic rules, as proposed in [18]. Starting from the root, the switch with the largest number of links connecting to switches that already belong to the tree is selected as the next switch in the tree. In case of tie, the switch with the highest average topological distance to the rest of the switches will be selected first.

Next, in order to assign direction to links, switches in the network must be labeled with positive integer numbers. When assigning directions to links, the "up" end of each link is defined as the end whose switch has the label with the highest value.

The *up*/down** routing algorithm cannot be applied to InfiniBand networks in a straightforward manner because it does not conform to IBA specifications. The reason for this is the fact that this routing algorithm takes into account both the input port and the destination ID for routing, whereas IBA switches only consider the destination ID. We have proposed two simple and effective strategies to solve this problem [22, 13]. In [13] routing restrictions are avoided with the destination renaming technique, which uses the IBA virtual addressing scheme. Basically, when a switch provides two different paths for packets destined to the same host which have arrived at the switch through different input channels, the destination of one of them is renamed, selecting a valid address (not used) from the range assigned to the destination host. In [22] the problem is solved by occasionally modifying those paths with routing conflicts. In this paper, we will use the destination renaming technique. As this technique does not require the use of additional network resources, *up*/down** routing can be implemented on InfiniBand by using one VL and one SL. However, additional VLs could be used to improve performance [21].

4. Flexible Routing

Like *up*/down** routing, the flexible routing algorithm [20] is a generic routing scheme that avoids deadlock by breaking cycles in the network graph. However, cycles are broken at different nodes for each direction in the cycle, thus providing better traffic balance than that provided by the *up*/down** routing algorithm.

To briefly illustrate this idea, consider the 4-switch network depicted in Figure 3. Solid arrows represent the "up" direction assigned to each link by the *up*/down** routing algorithm. Also, removed channel dependencies by *up*/down** are shown in dashed arrows. Each routing path crossing a channel is represented by $[x, y]$, where x and y represent the source and destination switches of the routing

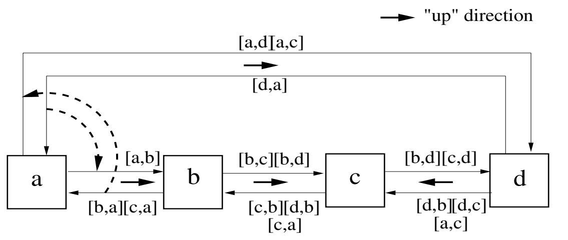


Figure 3. Link direction assignment and cyclic channel dependencies removed by using the up*/down* routing scheme.

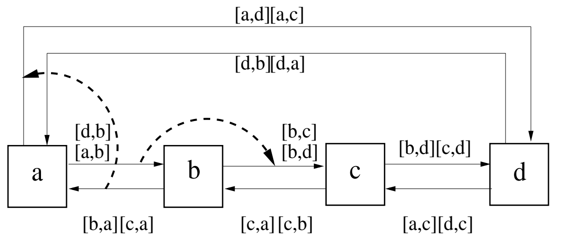


Figure 4. Independently removing cyclic channel dependencies for each direction of the cycle and balancing traffic.

path, respectively. Every routing path is computed by selecting a single path between every pair of switches, minimizing the number of routing paths crossing each channel. We can observe that the *up*/down** routing algorithm unevenly distributes the routing paths among channels, since there are some channels crossed by three routing paths, whereas other channels are crossed by one routing path.

A better traffic balance may be obtained if instead of breaking cyclic channel dependencies at the same point for each direction, they are broken at different points. This is the case of the *Flexible* routing algorithm. In Figure 4, we can observe that the number of dependencies removed from the network is the same as the one removed in Figure 3. However, the routing restrictions are independently placed for each direction of the cycle. We can observe that, with this approach, the maximum number of routing paths crossing every channel decreases down to 2. Moreover, the *Flexible* routing scheme may apply new routing restrictions to the network in addition to those imposed to remove cyclic dependencies. This allows to achieve, in a direct manner (i.e. without applying a traffic balancing algorithm), a better traffic distribution, avoiding disconnecting the network. For example, in Figure 5, we achieve the same traffic balance as in Figure 4 but without requiring the use of a traffic balancing algorithm.

The *Flexible* routing scheme is based on computing a DFS spanning tree on the network graph, which provides a suitable underlying graph to detect cycles. Then, the *Flex-*

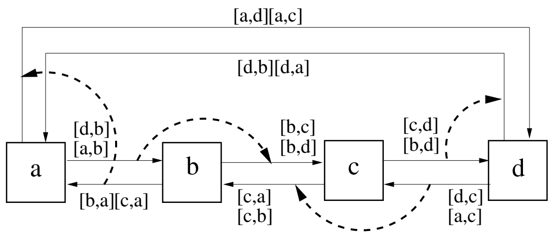


Figure 5. Balancing paths by removing additional channel dependencies with *Flexible*.

ible routing applies some rules to remove channel dependencies in order to guarantee deadlock freedom. See [20] for further details.

Like the *up*/down** routing algorithm, the *Flexible* routing cannot be applied in a straightforward manner to InfiniBand due to the fact that this routing algorithm also takes into account the input port and the destination ID to route packets. Therefore, we will also use the destination renaming technique to implement the *Flexible* routing on InfiniBand. Like *up*/down** routing, the *Flexible* routing algorithm only requires a single VL and SL to be applied on InfiniBand. Also, additional VLs could be used to improve performance [21].

5. Comparing Routing Algorithms

In this section, we analyze and quantify the benefits of using the routing algorithms described above in 2D and 3D tori. Since IBA routing is deterministic, a single path has to be selected for each source-destination pair among all the available ones. We will select this path by using the traffic balancing algorithm proposed in [18]. The analysis is performed by comparing some behavioral routing metrics: (1) The *average distance* metric, which is the average number of crossed links in the shortest path between any pair of nodes[3]; (2) the *crossing path* metric, which shows the maximum number of routing paths crossing through any network channel; and (3) the *deviation crossing* metric, which refers to the standard deviation of the crossing path metric.

Table 2 shows the values of these behavioral routing metrics computed for each routing algorithm for 2D torus networks with sizes of 4×4, 6×6, and 8×8 switches, and 3D tori with sizes of 3×3×3 and 4×4×4 switches. These network configurations provide up to 64 switches in the network, which seems adequate in small and medium scale clusters or NOWs. As can be seen, the *Flexible* and the *up*/down** routing algorithms cannot guarantee routing through minimal paths for large 2D tori (6×6 and 8×8 switches). However, the *e-cube* algorithm provides minimal paths for all destinations and for all network sizes.

[3]When paths are computed by assuming that there are no routing restrictions, then, the average distance is called *average topological distance*.

	Average Distances	Crossing path	Deviation Crossing
4×4 torus (average topological distance = 2.13)			
UD_DFS	2.13	12	2.77
FX	2.13	8	0.00
e-cube	2.13	10	0.75
6×6 torus (average topological distance = 3.08)			
UD_DFS	3.31	56	13.60
FX	3.31	43	10.55
e-cube	3.08	30	1.35
8×8 torus (average topological distance = 4.06)			
UD_DFS	4.57	209	43.30
FX	4.57	106	28.50
e-cube	4.06	70	2.19
3×3×3 torus (average topological distance = 2.07)			
UD_DFS	2.07	15	2.81
FX	2.07	9	0
e-cube	2.07	9	0
4×4×4 torus (average topological distance = 3.04)			
UD_DFS	3.04	75	14.53
FX	3.04	32	0
e-cube	3.04	48	8.89

Table 2. Comparing the Up*/down* routing algorithm based on the DFS methodology (UD_DFS), the Flexible routing (FX), and the e-cube algorithm for 2D and 3D tori.

Concerning traffic balance, the *e-cube* routing algorithm achieves the lowest value of the *crossing path* and the *deviation crossing* metrics when the topology radix is high. This is the case of the 6 × 6 and 8 × 8 tori. On the other hand, for low radix networks, the *Flexible* routing scheme provides the best traffic balance, because the value of the *deviation crossing* metric is equal to zero, which means that all the channels are uniformly balanced. This is due to the fact that this routing scheme performs a perfect traffic balance on 4-node cycles [20].

6. Performance Evaluation

In this section, we evaluate by simulation the performance of the improved *up*/down** routing algorithm based on the DFS methodology (UD_DFS), the *Flexible* routing (FX), and the *e-cube* algorithm for 2D and 3D tori. In particular, we have considered the same network topologies analyzed in the previous section.

For the *e-cube* algorithm, we have considered 2VL/4SL for 2D tori and 2VL/8SL for 3D tori. Moreover, for the *up*/down** and *Flexible* routing algorithms, we have considered 1VL/1SL (referred to as DFS_1VL and FX_1VL, respectively) and, for comparison purposes, also 2VL/2SL (referred to as DFS_2VL and FX_2VL, respectively).

The evaluation will be performed by using a simulator that models an InfiniBand network at the register transfer level, taking into account the timing parameters from InfiniBand switches in order to obtain realistic simulation results.

In all the presented results, we will plot the average packet latency[4] measured in nanoseconds versus the average accepted traffic[5] measured in bytes/ns/switch. It must be noticed that both measures are dependent variables on injected traffic. This is a compact way of presenting the results. However, it has the drawback that sometimes two different values of latency correspond to the same accepted traffic rate. The explanation is that both of them actually correspond to different traffic injection rates (the lowest injection rate corresponds to the one that obtains the lowest latency and vice versa).

First, we will describe the main InfiniBand network model features defined by the specs together with the main simulator parameters. Then, we will show the simulation results.

6.1. The InfiniBand Network Model

The network is composed of a set of switches and hosts, all of them interconnected by a single link. The evaluated 2D torus network uses 8-port switches, with 4 ports used to connect to hosts and leaving 4 ports to connect to other switches. For 3D torus, 10-port switches are considered, also using 4 ports to connect to hosts and leaving the rest of the ports to connect to other switches.

Packets are routed at each switch by accessing the forwarding table. This table contains the output port to be used at the switch for each possible destination. If there is sufficient buffer capacity in the output buffer, the packet is forwarded. Otherwise, the packet must wait at the input buffer. Buffer size will be fixed in both cases to 2KB.

In the simulator, each switch will have a crossbar connecting the input ports to the output ports, allowing multiple packets to be transmitted simultaneously without interference. Each output port has a separate arbiter that will select the next packet to be transmitted from the set of packets requesting the output port. The delay of the crossbar will be set accordingly to the value of the injection rate of the links. Switches can support up to a maximum of 15 virtual lanes (VL). Each VL provides separate guaranteed buffering resources. service level network. The SL identifier with the input port the output port has a separate VL round-robin arbiter that selects the next VL that contains a packet to be transmitted over the physical output link.

The routing time at each switch will be set to 100 ns. Links in InfiniBand are serial. In the simulator, the link injection rate will be fixed to the 1X configuration (2.5 Gbps) [10]. Therefore, a bit can be injected every 0.4 ns. With 8/10 coding [10] a new byte can be injected into the link every 4 ns. Also, the fly time[6] will be set to 100 ns, that corresponds to 20 m copper cable length with a propagation delay of 5 ns/m.

According to IBA specification, we use the virtual cut-through switching technique and a credit-based flow control scheme for each virtual lane.

We will use two different packet lengths in all the evaluations. We will use short packets with 32 bytes and long packets with 512 bytes. Also, we have considered different synthetic traffic patterns in order to analyze their influence on system performance. In particular, uniform, bit-reversal, and matrix transpose packet destination distributions will be considered.

6.2. Simulation Results

Figure 6 shows the simulation results for 2D tori of 4×4, 6×6, and 8×8 switches, when packet length is 512-bytes and uniform packet distribution is used. As can be seen, the *Flexible* routing algorithm significantly improves the performance with respect to other routing algorithms evaluated for 4×4 tori. Moreover, despite using only one virtual lane, performance improvement is noticeable. In particular, FX_1VL/SL increases throughput by 11 % and 20 % with respect to *e-cube* and DFS_2VL/SL, respectively. In addition, performance of the *Flexible* algorithm increases by 16 % when using two virtual lanes (FX_2VL/SL). As it was analyzed in Section 5, the reason for this improvement must be found in its ability to better balance network traffic.

When network size increases, the *e-cube* algorithm achieves a higher throughput than the *Flexible* routing algorithm. In particular, it achieves a throughput 25 % higher than FX_1VL/SL in a 8×8 torus. However, remember that the *e-cube* algorithm has to use 4 service levels and 2 virtual channels. On the other hand, the differences in throughput improvement with respect FX_2VL/SL is reduced to a 10 %.

Also, we can observe that the *e-cube* algorithm reduces the latency with respect to FX_2VL/SL. In particular, a reduction of 5.0 % and 9.8 % in latency is observed for 6×6 and 8×8 at low traffic rates, respectively. Remember that in these cases, the *e-cube* algorithm is the one that always provides minimal paths, thus decreasing packet latency, espe-

[4]Latency is the elapsed time between the generation of a packet at the source host until it is delivered at the destination end-node.

[5]Accepted traffic is the amount of information delivered by the network per time unit.

[6]Time required by a bit to reach the opposite link side.

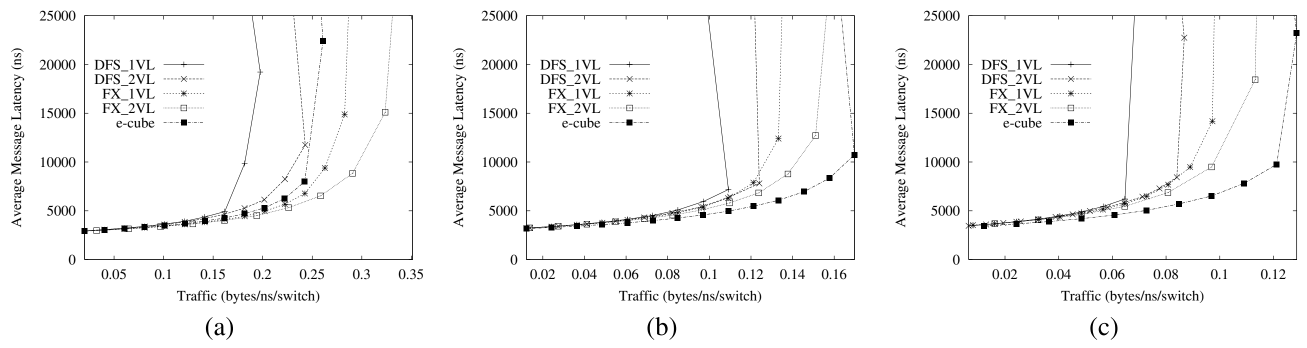


Figure 6. Average packet latency vs. accepted traffic for the improved up*/down* routing, the Flexible routing algorithm, and the e-cube algorithm. Torus of (a) 4×4, (b) 6×6, and (c) 8×8 switches. Packet length is 512 bytes. Uniform packet distribution.

cially with short packets. However, when using long packets the differences in latency between *e-cube* and *Flexible* routing algorithms are not so significant.

Figure 7 shows the simulation results for 3D tori of 3×3×3 and 4×4×4 switches. As can be seen, the performance improvement achieved by the *Flexible* routing algorithm for 3D tori is higher than that achieved for 2D tori. In particular, FX_1VL/SL increases throughput by a factor of 1.1 with respect to the *e-cube* algorithm for 4×4×4 tori. Moreover, FX_2VL/SL increases throughput by a factor of 1.33 with respect to *e-cube* in 4×4×4 tori, whereas in 3×3×3 tori the achieved throughput is improved by a factor of 1.25. Furthermore, the benefits of using the *Flexible* routing algorithm are more significant as long as it requires only one SL and one or two VLs, whereas the *e-cube* needs 2VLs and 8SLs, for 2D and 3D tori, respectively.

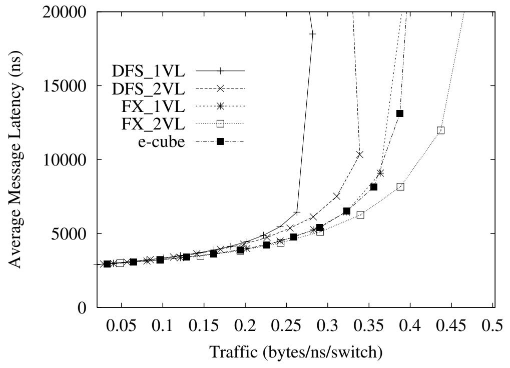
Table 3 shows the factors of throughput increase of the *Flexible* routing algorithm using 1VL/1SL and 2VL/2SL with respect to the *e-cube* routing algorithm for 2D and 3D tori when using different packet distributions and 512-byte packets. As can be seen, when bit-reversal and matrix transpose packet distributions are used, the obtained results are qualitatively similar to those obtained for uniform distribution. For these packet distributions, all the packets from a given host are sent to the same destination. This fact could influence the relative behavior exhibited by the analyzed routing algorithms. However, the results allow us to corroborate the benefits of using the *Flexible* routing algorithm in tori.

In Table 4 we can observe that when short packets are used (32 bytes), the benefits of using the *e-cube* algorithm in 6×6 and 8×8 tori increases with respect to the ones achieved with long packets in 6×6 and 8×8 torus. This is due to the fact that the latency of short packets is more sensitive to the distance between hosts. Therefore, the advantage of following shorter paths achieved by the *e-cube* algorithm increases for short packets. In general, these results corroborate those obtained with long packets.

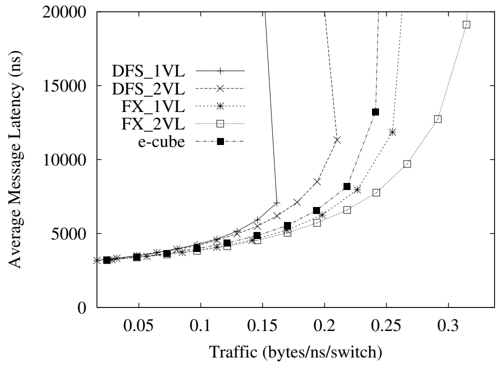
7. Conclusions

In this paper, we have analyzed the performance of several routing algorithms for torus networks in InfiniBand. The *e-cube* algorithm is compared with two generic routing algorithms, such as the improved *up*/down** and the *Flexible* routing algorithms, paying attention to both their performance and the network resources (InfiniBand VLs and SLs) required to support them. Medium sized networks with up to 64 switches have been evaluated.

Evaluation results show that the *Flexible* routing algorithm achieves the best performance for 3D tori and small 2D tori, whereas the *e-cube* algorithm achieves the best performance for large 2D tori. The main drawback of the *e-cube* algorithm is that it requires 2VLs and 4SLs (case of 2D tori) or 8SLs (case of 3D tori) to be implemented on InfiniBand. This could become a serious problem if the number of available SLs (which are mainly intended to QoS) is not enough to provide deadlock avoidance. However, the Flexible routing algorithm only requires one SL/VL in all cases. Taking into account that in the worst case (8×8 tori) the throughput achieved by the *Flexible* routing algorithm (using 2 SL/2VL) is only a 8.33% lower than that of the *e-cube* algorithm, we can conclude that the *Flexible* routing algorithm is the most cost-effective algorithm to be implemented on InfiniBand using torus networks. Further, the *Flexible* routing algorithm could achieve additional performance by using more VLs and SLs. Also, unlike the *e-cube* algorithm, it could continue to be applied in case of failures in the network.



(a)



(b)

Figure 7. Average packet latency vs. accepted traffic for the improved up*/down* routing, the Flexible routing algorithm, and the e-cube algorithm. Torus of (a) 3×3×3, and (b) 4×4×4 switches. Packet length is 512 bytes. Uniform packet distribution.

	uniform		*bit-reversal*		*matrix transpose*	
torus size	1VL/1SL	2VL/2SL	1VL/1SL	2VL/2SL	1VL/1SL	2VL/2SL
4×4	1.11	1.35	1.08	1.08	1.05	1.05
6×6	0.81	0.96	0.75	0.89	0.75	0.78
8×8	0.77	0.92	0.73	0.91	1.07	1.15
3×3×3	0.98	1.22	1.02	1.12	1.24	1.36
4×4×4	1.11	1.36	0.79	0.82	1.23	1.40

Table 3. Factors of throughput increase of the Flexible routing algorithm using 1VL/1L and 2VL/2L with respect to the e-cube algorithm. Torus with 4×4, 6×6, 8×8, 3×3×3, and 4×4×4 switches. Uniform, bit-reversal, and matrix transpose packet distributions. Packet length is 512 bytes.

References

[1] N. J. Boden et al., "Myrinet - A gigabit per second local area network", *IEEE Micro*, vol. 15, Feb. 1995.

[2] R. Casado, et al., "Performance evaluation of dynamic reconfiguration in high-speed local area networks", 6^{th} *International Symposium on High Performance Computing Architecture*, December 2000.

[3] J. Carbonaro and F. Verhoorn, "Cavallino: The TeraFlops router and Nic", *International Symposium on High Performance Interconnects*, 1996.

[4] A. A. Chien and J. H. Kim, "Planar-adaptive routing: Low-cost adaptive networks for multiprocessors," in *Proceedings of the 19th International Symposium on Computer Architecture*, May 1992.

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

[6] W. J. Dally and C. L. Seitz, " Deadlock-free Message Routing in Multiprocessor Interconnection Networks" *IEEE Transactions on Parallel and Distributed Systems*, 1987.

[7] J. Duato and P. López, "Performance evaluation of adaptive routing algorithms for k-ary n-cubes," in *Parallel Computer Routing and Communication*, K. Bolding and L. Snyder (ed.), Springer-Verlag, pp. 45–59, 1994.

[8] D. García and W. Watson, "Servernet II", *1997 Parallel Computer, Routing, and Communication Workshop*, June 1997.

	uniform		*bit-reversal*		*matrix transpose*	
torus size	1VL/1SL	2VL/2SL	1VL/1SL	2VL/2SL	1VL/1SL	2VL/2SL
4×4	1.13	1.31	1.17	1.18	1.03	0.97
6×6	0.76	0.86	0.82	0.90	0.81	0.86
8×8	0.72	0.80	0.81	0.89	1.08	1.16
3×3×3	1.00	1.17	0.98	1.00	1.20	1.27
4×4×4	1.11	1.28	0.94	0.96	1.25	1.39

Table 4. Factors of throughput increase of the Flexible routing algorithm using 1VL/1L and 2VL/2L with respect to the e-cube algorithm. Torus with 4×4, 6×6, 8×8, 3×3×3, and 4×4×4 switches. Uniform, bit-reversal, and matrix transpose packet distributions. Packet length is 32 bytes.

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

[10] InfiniBand^{TM} Trade Association, *InfiniBand^{TM} architecture. Specification Volumen 1. Release 1.0.a.* Available at http://www.infinibandta.com.

[11] R. E. Kessler and J. L. Schwarzmeier, "Cray T3D: A New Dimension for Cray Research", *Digest of Papers, COMPCON Spring '93*, IEEE Computer Society Press, February 1993.

[12] D.H. Linder and J.C. Harden, "An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes," *IEEE Trans. Comput.*, vol. C-40, no. 1, pp. 2–12, Jan. 1991.

[13] P. López, J. Flich, and J. Duato, "Deadlock-free Routing in InfiniBand^{TM} through Destination Renaming", *2001 International Conference on Parallel Processing*, September 2001.

[14] S. S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb, "The Alpha 21364 Network Architecture", IEEE Micro, vol. 22, no. 1, pp. 26-35, Jannuary 2002.

[15] W. Qiao and L. M. Ni, "Adaptive routing in irregular networks using cut-trough switches", *Proceedings of the 1996 International Conference on Parallel Processing*, August 1996.

[16] G. Pfister, *In search of clusters*, Prentice Hall, 1995.

[17] R. Riesen et al., "CPLANT", *Second Extreme Linux Workshop*, June. 1999.

[18] J.C. Sancho and A. Robles, " Improving the Up*/Down* Routing Scheme for Networks of Workstations", *Euro-Par 2000*, August 2000.

[19] J.C. Sancho, A. Robles, and J. Duato, " New Methodology to Compute Deadlock-Free Routing Tables for Irregular Networks", *Fourth Workshop on Communication, Architecture and Applications for Network-based Parallel Computing*, January 2000.

[20] J.C. Sancho, A. Robles, and J. Duato, "A flexible routing scheme for networks of workstations", *fourth International Symposium on High Performance Computing*, Oct. 2000.

[21] J.C. Sancho, J. Flich, A. Robles, P. López and J. Duato, "Analyzing the Influence of Virtual Lanes on InfiniBand Networks", *Workshop on Communication Architecture for Clusters*. Apr. 2002.

[22] J.C. Sancho, A. Robles, and J. Duato, "Effective Strategy to Compute Forwarding Tables for InfiniBand Networks", *2001 International Conference on Parallel Processing* , September 2001.

[23] S. L. Scott and G. M. Thorson, "The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus", *Symposium on High Performance Interconnects*, August 1996.

[24] M. D. Schroeder et al., "Autonet: A high-speed, self-configuring local area network using point-to-point links", *SRC research report 59*, DEC, Apr. 1990.

[25] R. Sheifert, *Gigabit Ethernet*, Addison-Wesley, Apr. 1998.

[26] F. Silla and J. Duato, "Improving the Efficiency of Adaptive Routing in Networks with Irregular Topology", *International Conference on High Performance Computing*, Dec. 1997.

[27] H. Sullivan and T. R. Bashkow, "A large scale, homogeneous, fully distributed parallel machine," *Proceedings of the 4th International Symposium on Computer Architecture*, March 1977.