A Scalable Methodology for Computing Fault-Free Paths in InfiniBand Torus Networks*

J.M. Montañana, J. Flich, A. Robles, and J. Duato

Dept. of Computer Engineering (DISCA,UPV)
Camino de Vera, 14, 46021–Valencia, Spain
jmontana@gap.upv.es

Abstract. Currently, clusters of PCs are considered as a cost-effective alternative to large parallel computers. In these systems the interconnection network plays a key role. As the number of elements increases in these systems, the probability of faults increases dramatically. Moreover, in some cases, it is critical to keep the system running even in the presence of faults. Therefore, an effective fault-tolerant strategy is needed.

InfiniBand (IBA) is a new standard interconnect suitable for clusters. Unfortunately, most of the fault-tolerant routing strategies proposed for massively parallel computers cannot be applied to IBA because routing and virtual channel transitions are deterministic, which prevent packets from avoiding the faults. A possible approach to provide fault-tolerance in IBA consists of using several disjoint paths between every source-destination pair of nodes and selecting the appropriate path at the source host. However, to this end, a routing algorithm able to provide enough disjoint paths, while still guaranteeing deadlock-freedom, is required. In this paper we address this issue, proposing a scalable fault-tolerant methodology for IBA Torus networks. Results show that the proposed methodology scales and supports up to $(2n - 1)$-faults for n-dimensional tori when using 2 VLs (virtual lanes) and 4 SLs (service levels) regardless of the network size. Additionally the methodology is able to support up to 3 faults for 2D tori with 2 VLs and only 3 SLs.

1 Introduction

Over the recent years there is a trend in using clusters of PCs for building large systems. Some examples are cluster-based Internet portal servers like AOL, Google, Amazon or Yahoo. Also, clusters of PCs are currently being considered as a cost-effective alternative for small and large-scale parallel computing. Each time, more cluster-based systems are included in the top500 list of supercomputers [2]. As an example, the Abe system [1] with 2,400 quad-core Intel Xeon 64 2.3 GHz processors (9,600 cores) is in the eighth position.

In these systems, the interconnection network plays a key role in the performance achieved. In fact, clusters are being built by using high-end interconnection networks

* This work has been jointly supported by the Spanish MEC and European Commission FEDER funds under grants "Consolider Ingenio-2010 CSD2006-00046" and "TIN2006-15516-C04-0X"; and by JCC de Castilla-La Mancha under grant PBC-05-007-2.

J. Labarta, K. Joe, and T. Sato (Eds.): ISHPC 2005 and ALPS 2006, LNCS 4759, pp. 79–92, 2008.

like Quadrics [10], InfiniBand [6], and Myrinet [4]. Among them, InfiniBand (IBA) is a standard interconnect technology for interconnecting processor nodes and I/O nodes, thus building a system area network (SAN). The InfiniBand Architecture (IBA) is designed around a switch-based interconnect technology with high-speed serial point-to-point links connecting multiple independent and clustered hosts and I/O devices. Therefore, this interconnect technology is suitable to build large clusters. As an example, the Abe system uses InfiniBand.

Often, clusters are arranged on regular network topologies when the performance is the primary concern. Low dimensional tori (2D and 3D) are one of the most widely used topologies in commercial parallel computers. Furthermore, recent proposals, such as Alpha 21364 [9] and BlueGene/L [3], use 2D and 3D tori, respectively.

In many cluster-based systems is critical to keep the system running even in the presence of faults. These systems use a very large number of components (processors, switches, and links). Each individual component can fail, and thus, the probability of failure of the entire system increases. Although switches and links are robust, they are working close to their technological limits, and therefore they are prone to faults. Increasing clock frequency leads to a higher power dissipation, and a higher heating could lead to premature faults. So, fault-tolerant mechanisms in cluster-based systems are becoming a key issue.

Most of the fault-tolerant routing strategies proposed in the literature for massively parallel computers are not suitable for clusters (see chapter 6 of [5] for a description of some of the most interesting approaches). This is because they often require certain hardware support that is not provided by current commercial interconnect technologies [4,6]. Other strategies rely on the use of adaptive routing. However, they cannot be applied to IBA because routing is deterministic, which prevents packets from circumventing the faulty components found along their paths. Also, some of these routing strategies need to perform dynamic virtual channel transitions when the packet is blocked due to a fault. However, virtual channels in IBA cannot be selected at routing time.

In IBA, routing and virtual channel (in IBA they are referred to as Virtual Lanes, VLs) selection is performed 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. As a consequence, a possible way to provide fault-tolerance in IBA would be to have several alternative paths between every source-destination pair, selecting one of them at the source host. In particular, to tolerate n faulty components it will be necessary to provide $n + 1$ disjoint paths. From a practical point of view, an appropriate methodology should be able to support a reasonable number of failures according to the network size. Obviously, the switch with the lowest degree will bound the number of possible disjoint paths in the last analysis, which will limit, in turn, the maximum number of faults that can be tolerated at the same time. For instance, in a 2D Torus network only four disjoint paths can be computed for every source-destination pair.

IBA provides a mechanism supported by hardware, referred to as Automatic Path Migration (APM)[6], which may be used for selecting among the available disjoint paths. According to this mechanism, at connection setup time, the source node is given two sets of path information for each destination, one for the primary path and one for

the alternate path. APM provides a fast mechanism for migration from the primary to the alternate path when a faulty component is detected. Once path migration is accomplished, the alternate path is converted into the new primary path. Therefore, the subnet manager could reload the alternate path variables with new ones and re-enable the APM mechanism.

2 Motivation

In [8] we presented a methodology referred to as TFTR (Transition-based Fault Tolerant Routing) to compute disjoint paths for an InfiniBand network. In particular, TFTR computes four disjoint paths for every source-destination pair in a 2D Torus network. To get the maximum flexibility in computing disjoint paths, the methodology relies on virtual channel transitions for some paths. To do so, the methodology computes an underlying up*/down* tree [11] and then, the paths that use illegal up*/down* transitions are enforced to switch to a new (increasing) virtual channel.

The methodology requires the use of only two virtual channels to obtain four disjoint paths for every pair of nodes in a 2D Torus. Unfortunately, the methodology requires an unbounded number of SLs. This is because, according to IBA specs [6], the SL assigned to a packet can not be changed by the switches, and this may cause mapping conflicts (see Section 3 for details). Mapping conflicts are solved by using a different path (if any) or using a new SL (not causing a mapping conflict). Taking into account that in InfiniBand the maximum number of SLs is 16 and they are used for other purposes (mainly QoS), the methodology does not scale. For instance, the methodology requires using 2 VLs and 7 SLs for a 10×10 Torus network. Therefore, only two traffic classes could be used at most in conjunction with this fault-tolerant routing methodology, which would significantly limit the QoS capabilities of IBA technology. It has to be noted that, as network size increases, the length of paths also increases. Therefore more mapping conflicts arise, leading to an increase in the number of SLs required. This is a serious problem for IBA and requires an effective solution.

When computing disjoint paths for fault-tolerance issues in a 2D Torus network with InfiniBand, it would be desirable to obtain four disjoint paths for every pair of nodes in such a way that the number of resources used (VLs and SLs) is the same and as low as possible independently of the network size.

In this paper we take on such a challenge. To this end, we need to apply a methodology to compute disjoint paths completely different from that applied in [8], introducing novel concepts, models and computation strategies. In particular, we propose a new methodology referred as SPFTR (Scalable Pattern-based Fault Tolerant Routing). SPFTR will compute four disjoint paths for a 2D Torus network by requiring only the use of 2 VLs and 4 SLs regardless of the network size. Thus, it will guarantee the existence of up to four QoS channels and, at the same time, will tolerate up to 3 faults. Moreover, the methodology will generate a knowledge database of route patterns, which will allow the computation of the disjoint routing paths in a time-efficient manner for any network size. In particular, the computation cost of the methodology will be $O(N^2)$, where N is the number of nodes in the system. Also, the methodology will be easily extended to higher dimensional Torus networks, without increasing the number of resources required.

Notice that the methodology does not depend on the hardware used for detecting failures. Also, it does not depend on the way failures are notified.

The rest of the paper is organized as follows. In Section 3, mapping conflicts in InfiniBand will be explained. In Section 4, SPFTR will be presented for a 6×6 Torus network. In Section 5, the methodology will be extended to larger 2D and higher dimensional Torus networks. In Section 6, the methodology will be evaluated in terms of fault-tolerance, performance and cost. Finally, in Section 7, some conclusions will be drawn.

3 Mapping Conflicts in InfiniBand

In InfiniBand, every switch has a forwarding table where the output port for each destination is provided. The destination ID is located at the packet header and does not change along the path. Therefore, IBA routing is a kind of source routing with the routing info distributed.

Up to 15 data virtual lanes (VLs) can be used 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 it cannot be changed by the switches. Therefore, we should also assign the proper SL to be used for a given path.

However, the fact of fixing a path with a unique SL can lead to a mapping conflict. It occurs when two packets labelled with the same SL enter a switch through the same input port, and they need to be routed to the same output port but through different VLs. The problem is that the SLtoVL mapping table does not use the input VL in order to determine the output VL. Figure 1.(a) shows an example. At switch R a mapping conflict arises as it is not possible to distinguish both paths because they are labelled with the same SL. It has to be noted that this problem arises only when there are paths with different VLs. For example, path A uses VL1 and path B uses VL0 until switch Q.

A mapping conflict can be solved only by using different service levels (SLs) for each path causing the mapping conflict. However, this often leads to an excessive number of SLs. Another solution is to use an alternative path that does not cause a mapping conflict. However, obtaining such alternative path strongly depends on the flexibility provided by the applied routing algorithm, on the available network resources (VLs), and the strategy applied to obtain SLtoVL tables.

4 Description of SPFTR

In this section we will describe SPFTR. For the sake of simplicity and ease of understanding we will apply the methodology to a 6×6 Torus network. In the next section we will extend the methodology to larger networks and n-dimensional Torus networks. The aim of this methodology is to compute the forwarding and SLtoVL tables for IBA Torus networks in such a way that they provide the maximum number of disjoint routing paths ($2n$) between every pair of nodes in a n-dimensional Torus in order to tolerate up to $2n-1$ failures. The methodology must deal at the same time with different issues:

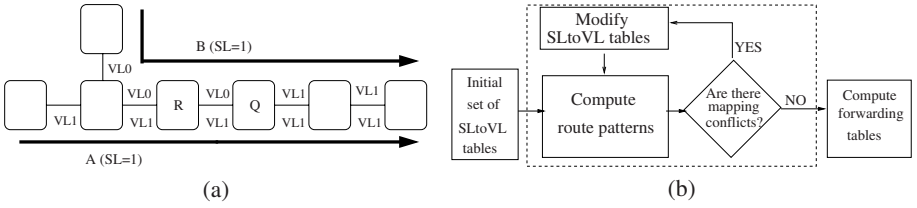


Fig. 1. (a) Mapping conflict example, (b) and steps followed by SPFTR

- It must ensure that any combination of the computed disjoint paths ($2n$ for every source-destination pair) does not lead to deadlock. Deadlock-freedom is ensured by the routing algorithm used.
- It must use only 4 SLs and 2 VLs, regardless of network size. This is achieved by computing in an appropriate manner the SLtoVL tables.
- It must be scalable. For this end, network regions whose switches require the same content for their SLtoVL tables will be defined.
- It must be simple enough in order to be applied in a time-efficient manner to larger networks. For this aim, we will obtain a set of route patterns to be used as templates to define the set of final disjoint paths.

All these issues have direct dependencies between them, and this fact makes obtaining the methodology quite challenging. The choice of the routing algorithm will influence the route patterns that can be used. At the same time, the route patterns used will influence the definition of the SLtoVL tables. In the other way, the definition of SLtoVL tables will influence the route patterns that can be applied.

4.1 Steps Followed by SPFTR

Figure 1.(b) shows the three steps followed by the methodology. Firstly, SLtoVL tables are initialized according to some initial rules: two virtual lanes will be used and transitions among virtual lanes will be done in increasing order (first VL0, then VL1) in order to avoid deadlocks.

In the second step, we look for a set of route patterns that accomplishes the restrictions imposed by the SLtoVL tables. This step will try to get paths as short as possible. The methodology performs as many iterations as required in order to get the final set of paths without introducing mapping conflicts. For this, once route patterns are computed, mapping conflicts are searched. If there is at least a mapping conflict, then the set of SLtoVL tables are changed accordingly (and manually), and a new iteration is performed.

Obviously, the computational cost of this procedure is not bounded and could become very high. However, this methodology takes advantage of the fact that once all the routes are successfully computed, it generates a unique definition of SLtoVL tables and route patterns that can be used in any 2D Torus regardless of its size. Applying the methodology to larger networks (once it was successfully applied to a small 2D Torus) will exhibit a low computational cost. In particular, the computational cost will be

$O(n^2)$, where n is the number of nodes. This is because the SLtoVL tables are already computed for each source-destination pair. Therefore, we only have to apply the route patterns to obtain the forwarding tables.

4.2 Routing Algorithm and SLtoVL Table Initialization

The methodology assumes a deterministic routing algorithm which guarantees deadlock-freedom by building an acyclic channel dependency graph (CDG). This is achieved by enforcing some routing restrictions at the switch level, which prevents packets from traversing some consecutive links (forbidden transitions).

To do this, an underlying deadlock-free routing algorithm will be used. This routing algorithm will indicate where the forbidden transitions are placed. In order to increase the routing flexibility when looking for disjoint paths, the applied routing scheme will traverse some forbidden transitions. The routing scheme will carry out a virtual lane transition each time a forbidden transition is traversed. As commented, the proposed methodology will use only 2 VLs. This means that every routing path can traverse at most one forbidden transition (i.e. only one virtual lane transition can be carried out by a packet at most). In addition, virtual lanes must be used in an ordered way (for example, first VL0, then VL1), in order to avoid cycles.

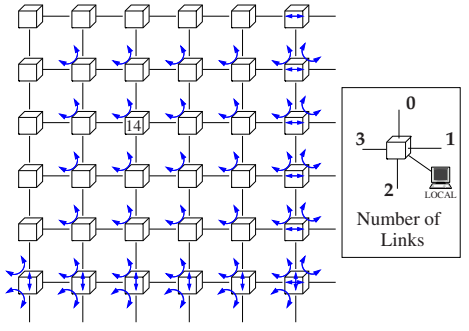
Thus, the key factor on defining our routing scheme is the selection of the appropriate underlying routing algorithm. This algorithm should be selected in such a way that it guarantees obtaining four disjoint paths for every pair of nodes and also it allows the scalability of the paths (the route patterns are valid regardless of the network size).

For this, we have started from up*/down* routing algorithm [11]. Up*/down* is based on an assignment of directions (”up” and ”down”) to the links. In our case the labeling of links will be slightly different from that performed by the original up*/down* routing. This assignment of directions is performed by building a certain spanning tree from the network graph. To do this, the node in the corner is selected as the root. Unlike original up*/down* routing, which builds a spanning tree from the complete network graph, the proposed strategy proceeds to remove all the wraparound links of the Torus before building the BFS spanning tree. Once the BFS spanning tree is built, the wraparound links are added, and then we impose a particular set of routing restrictions in order to break cyclic channel dependencies in the CDG. In particular, we have identified all the possible cycles that can be formed and proceeded to remove them by imposing the corresponding routing restrictions. The distribution of the routing restrictions can be seen in Figure 2.a. Deadlock-freedom is guaranteed by verifying that the resulting CDG is acyclic.

The main advantage of this algorithm is that, in most places, the orientation of the routing restrictions is the same, as can be seen in Figure 2.(a), which definitely will contribute to ease the computation process of SLtoVL tables carried out by the methodology [1].

Notice that this algorithm has not been designed with the aim of providing minimal paths between every pair of nodes nor minimizing the number of routing restrictions.

[1] We tried other routing algorithms, such as e-cube and up*/down*. However they were not able to provide a scalable methodology using the minimum number of SLs.



(a)

In Port	Out Port	SL	VLOut
0	1	X	N/D
0	2	X	N/D
0	3	X	**1**
1	X	X	N/D
2	X	X	N/D
3	0	X	**1**
3	1	X	N/D
3	2	X	N/D
local	X	X	N/D
X	local	X	N/D

(b)

Fig. 2. Underlying routing algorithm applied: (a) Imposed routing restrictions, (b) SLtoVL table initialization for switch 14. N/D means Not Defined

Instead, it is only used as an underlying routing to guarantee deadlock-freedom of the final routing scheme and provide the symmetry required to simplify the methodology. Notice that this scheme will be able to provide minimal paths by traversing some forbidden transitions and carrying out the corresponding virtual channel transition.

As commented above, the first step of the methodology is to initialize the SLtoVL tables. In particular, SLtoVL tables must initially contain those entries corresponding to link transitions (In Port - Out Port) that require performing in turn a virtual lane transition (i.e., transitions forbidden by the underlying routing algorithm). For instance, Figure 2.(b) shows the initial SLtoVL table corresponding to a switch in the centre of the network. The numbering of the links[2] is shown in Figure 2.(a). Specifically, the entries corresponding to link transitions 0 - 3 and 3 - 0 will be forced to use VL1 (shown in bold face in Figure 2.(b)) because both link transitions are forbidden, as can be seen in Figure 2.(a).

However, note that setting the SLtoVL table in such a way does not guarantee on its own deadlock-freedom. In order to enforce deadlock-freedom packets using these entries should enter the switch exclusively through VL0. This fact must be taken into account when computing the routing paths. The rest of table entries are not enforced in this step by the routing algorithm. The methodology will update those entries accordingly in the next step.

4.3 Network Regions and Route Patterns

The second step of the methodology aims at computing a set of route patterns. Route patterns constitute a kind of templates that can be used to obtain all the disjoint routing paths between every pair of nodes. A route pattern is defined by a sequence of movements (e.g., ”go to next switch on the left” or ”go to the left until the column of destination”) together with a SL. This sequence of movements must be compatible with the routing scheme applied in order to guarantee deadlock-freedom. Moreover, the SL must be selected in such a way that mapping conflicts are avoided.

[2] This numbering will be used throughout the rest of the paper.

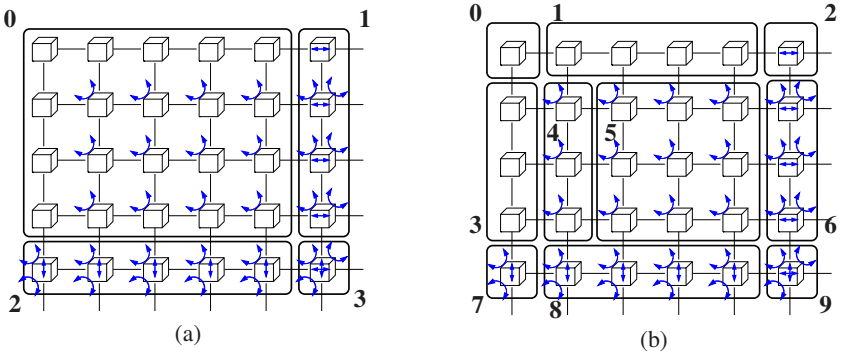


Fig. 3. (a) SPFTR regions (b) A-SPFTR regions in a 2D Torus Networks

Additionally, the methodology defines network regions. A network region will be formed by neighbour switches with the same SLtoVL table definition. Network regions are defined taking into account two conditions. The first one is that all the switches in a region must have the same routing restrictions (this is obvious as they must have the same SLtoVL table). And, second, in order to help the methodology to scale, the same distribution of regions should be kept regardless of network size. Figure 3.(a) shows the final regions defined for the 6×6 Torus network. As can be observed, 4 regions have been defined so that fulfil the conditions referred above. Each of them has a different SLtoVL table for every switch within the region.

The main advantage of using regions is that it minimizes the number of required route patterns. Route patterns depend on the regions where source and destination are located (the same or different) and their relative positions (i.e., they may be located either in the same row/column or in different rows and/or columns). To generate the route patterns, we try to obtain four disjoint paths (as short as possible) between every pair of nodes. To this end, it is necessary to establish the path followed by the pattern (sequence of movements) and the SL to be used. The former is accomplished by following the restrictions imposed by the applied routing algorithm (required VL transitions were already introduced in the initial SLtoVL tables), thus guaranteeing deadlock-freedom. The latter is carried out according to the current SLtoVL table entries and the bounded number of SLs used (4 SLs), checking that mapping conflicts are not introduced.

If it is not possible to obtain a set of valid route patterns, SLtoVL tables must be slightly updated (manually) and a new try to obtain route patterns will be carried out. Thus, an iterative process is performed with the two steps until a valid set of route patterns is found.

Each route pattern must be assigned a SL, checking the corresponding entries in the SLtoVL tables to verify that mapping conflicts are not introduced.

Table 1 shows the final SLtoVL table for every region. These tables have been computed taking into account the route patterns applied for every source-destination pair. The contents of the final SLtoVL tables are the same for all the switches belonging to the same region. According to the defined regions and the possible relative positions of every pair of nodes, 153 route patterns have been obtained.

Table 1. SLtoVL tables for every region in 2D Torus network (values imposed by routing restrictions in each region are written in bold face, X means any value)

		Region 0	Region 1	Region 2	Region 3			Region 0	Region 1	Region 2	Region 3
In	Out	SL0-1-2-3	SL0-1-2-3	SL0-1-2-3	SL0-1-2-3	In	Out	SL0-1-2-3	SL0-1-2-3	SL0-1-2-3	SL0-1-2-3
0	1	0 1 0 0	**1 1 1 1**	0 1 0 0	**1 1 1 1**	2	0	1 1 0 1	1 1 0 1	**1 1 1 1**	**1 1 1 1**
0	2	0 1 0 0	0 1 0 0	**1 1 1 1**	**1 1 1 1**	2	1	0 1 0 1	1 1 0 1	0 1 0 1	0 1 0 1
0	3	**1 1 1 1**	**1 1 1 1**	**1 1 1 1**	**1 1 1 1**	2	3	1 1 0 1	1 1 0 1	**1 1 1 1**	**1 1 1 1**
1	0	1 1 0 1	**1 1 1 1**	1 1 0 1	**1 1 1 1**	3	0	**1 1 1 1**	**1 1 1 1**	**1 1 1 1**	**1 1 1 1**
1	2	0 1 0 1	0 1 0 1	1 1 0 1	0 1 0 1	3	1	0 1 0 0	**1 1 1 1**	0 1 0 0	**1 1 1 1**
1	3	1 1 0 1	**1 1 1 1**	1 1 0 1	**1 1 1 1**	3	2	0 1 0 0	0 1 0 0	**1 1 1 1**	**1 1 1 1**
loc.	X	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	loc.	loc.	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
X	loc.	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1						

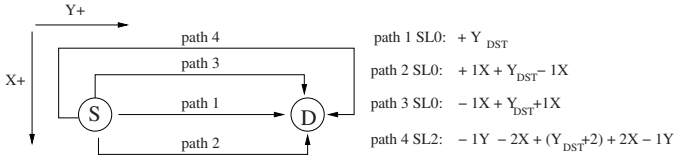


Fig. 4. Example of route pattern

To illustrate the process of computing the routing paths from the route patterns, without lose of generality, let us consider a pair of switches located in the same row in region 0 (the relative position of the switches and the regions will determine the pattern to use). Then, the patterns defined for computing the paths can be shown in Figure 4. The computed routing paths in a 6×6 Torus network can be shown in Figure 5.(a).

As an additional result from the methodology, we have successfully obtained an alternative set of route patterns and SLtoVL table definition that only requires 2 VLs and 3 SLs. This has been achieved by sacrificing the scalability in terms of higher dimensional Torus network, thus being only valid for 2D tori. However, scalability in terms of number of nodes is guaranteed (see next section). The resulting algorithm will be referred to as A-SPFTR (Asymmetric scalable patter-based fault tolerant routing). The regions defined are shown in Figure 3.(b).

5 Extending the Methodology

In this Section we will show how the methodology can be applied to other networks. In a first effort, the methodology will be extended to larger 2D Torus networks, thus scaling the methodology. In a second effort, we will extend it to 3D Torus networks.

A larger 2D Torus network can be viewed as a 6×6 Torus network with additional rows and/or columns of switches. These new rows or columns can be placed in the middle of the Torus network (taking as a reference Figure 3.(a)). Therefore, the new components will belong to regions 0 and 2 in the case of a column or to regions 0 and 1 in the case of a row. By doing this, the distribution of regions will be the same. Therefore, SLtoVL tables of new switches will be already defined. Also, the route patterns to use for every new switch to all the destinations have been already computed as each switch can use the route patterns of one of its neighbours. As an example, Figure 5.(b)

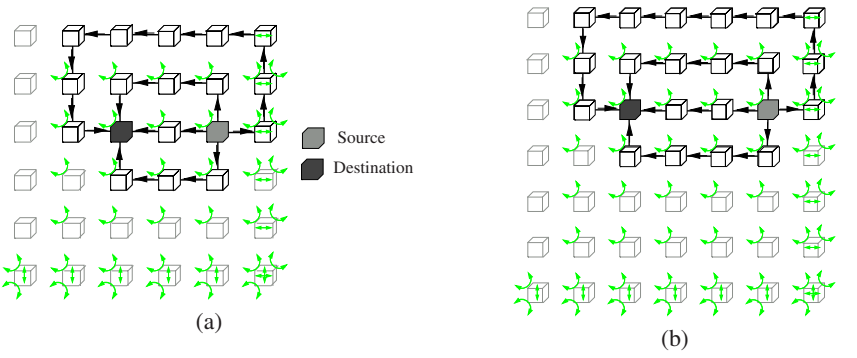


Fig. 5. Example of pattern applied in a (a) 6 × 6 and (b) 7 × 7 Torus network

shows the paths computed for a given pair of switches for a 7 × 7 Torus by using the patterns shown in Figure 4.

Notice also that A-SPFTR (2 VLs and 3 SLs) also scales. In this case, new rows could be added along regions 3, 4, 5, and 6, and new columns along regions 1, 5, and 8, maintaining the same region structure shown in Figure 3.(b).

In order to extend the methodology to n-dimensional Torus, we will use the same routing algorithm described in Section 4.2. It will define the routing restrictions (Figure 6.(a) shows an example for a 3 × 3 × 3 Torus network). In particular, for every plane in the 3D Torus we can find an orientation at which all restrictions are allocated in the same positions as in the 2D Torus shown in Figure 2.(a).

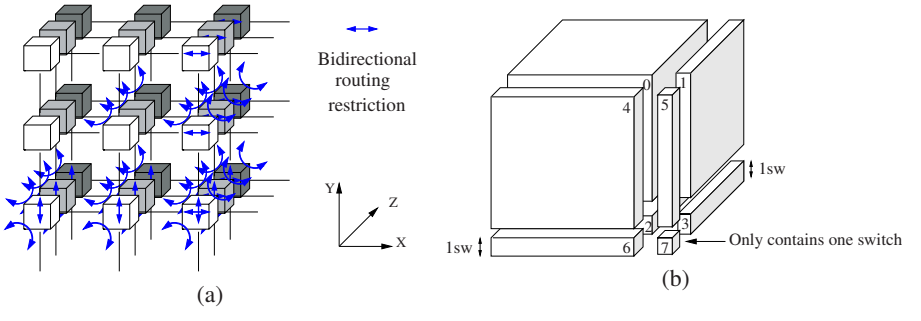


Fig. 6. (a) Frontal view of routing restrictions in the plain X-Y of a 3D Torus. (b) SPFTR Regions in a 3D Torus

Then we can define the SLtoVL table (defining the output VL for each pair of links), considering that each pair of links can be allocated in a plane and using the SLtoVL tables for the corresponding 2D plane, as shown in Figure 6.(a). By doing this in all 2D planes, all the entries for the SLtoVL tables (for the 3D case) will be filled.

Notice that every pair of links which are in the same direction will be included in two planes. Therefore, the SLtoVL values for them must be the same in both planes. That is, SLtoVL tables must be compatible plane by plane. The final regions in a 3D Torus can be shown in Figure 6.(b).

In the case of a 3D Torus the number of route patterns for each plane will be the same, then we will have 3 times the quantity of patterns defined for a plane (153 patterns). Additionally it is needed to add the route patterns for the pair of nodes not included in the same plane. Finally, the total amount of route patterns is 726.

6 Evaluation

In this section, we will evaluate SPFTR and A-SPFTR. Notice that both models are scalable in size but only SPFTR is scalable in dimensions too. For comparison purposes we will also evaluate the up*/down* routing scheme. First, we will present the evaluation model, describing all the simulation parameters and the Torus networks we have used. Finally, we will present the evaluation results.

6.1 Evaluation Model

We will evaluate the routing algorithms in 2D and 3D Torus networks. We have analyzed 2D tori with different sizes from 16 switches (4x4) up to 400 switches (20x20). Also 3D tori will be analyzed, from 64 switches (4x4x4) up to 512 switches (8x8x8).

In the analysis, we will only consider faults of links connecting switches. Note that a switch failure can be viewed as if all its links had failed. Moreover, a failure in a link connecting a host to a switch does not change the topology. The number of disjoint paths depends on the minimum degree of any switch in the network. Therefore, at maximum, there will be 4 disjoint paths in a 2D Torus (6 in a 3D Torus). Hence, at most, 3 faults can be tolerated (5 in a 3D Torus). The methodology will also be evaluated in terms of network performance. In particular, network performance degradation due to faults will be analyzed when applying the proposed methodology. The network performance is evaluated using the simulation tool and simulation parameters presented in [7]. The simulator models an IBA network, following the IBA specifications [6].

6.2 Evaluation Results

Fault Tolerance. We define a *singular case* as the fault combination that can not be tolerated by the routing algorithm, while still maintaining the network connected. In other words, those fault combinations for which the routing algorithm is not able to obtain a valid path for a particular source-destination pair.

We will evaluate the fault tolerance degree analyzing which combinations of faults are supported and which are not (singular case), for different amounts of link failures.

Figure 7 shows the percentages of singular cases when using up*/down*, SPFTR, (results for A-SPFTR are similar to SPFTR) for different number of faults. As can be observed, SPFTR (same happens for A-SPFTR) do not present any singular case up to any combination of $(2n - 1)$ faults in n-dimensional tori, as it is able to provide $2n$

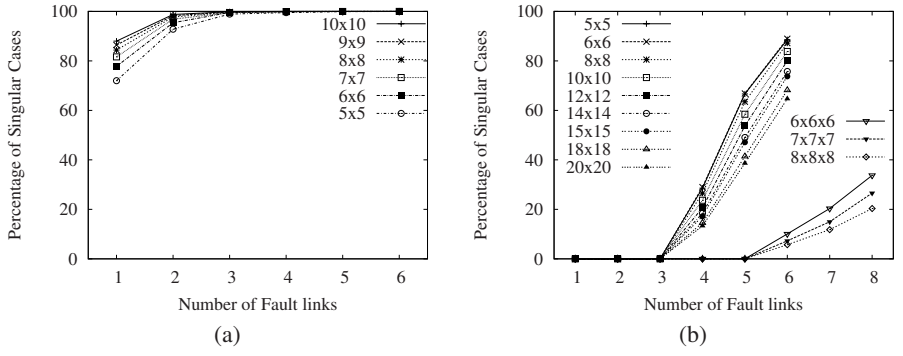


Fig. 7. Singular cases for (a) Up*/Down* and (b) SPFTR in 2D and in 3D (Maximum of 100.000 fault combination evaluated at each point when number of fault combinations is higher than 100.000)

disjoint paths. Therefore, the methodology is $(2n-1)$-fault tolerant with 2 VLs and 4 SLs for n-dimensional tori (SPFTR) or with 2 VLs and 3 SLs for 2D tori (A-SPFTR). On the other hand, we can see that up*/down* does not tolerate even a single failure.

From four faults and beyond we can see that none of the routing methods is able to tolerate all the failure combinations. Indeed, for 4 faults in the network, 30% of failure combinations are not tolerated in the worst case (5x5 Torus network). However, this is a reasonable fault-tolerance degree taking into account the analyzed network sizes, and that the mean time between failures is much greater than the mean time to repair.

However, notice that the up*/down* routing algorithm has a pretty worse behaviour. Practically, all fault combinations, even with one fault, are not tolerated by this algorithm. Also, notice that when a singular case arises the only solution will be to launch a network reconfiguration process to compute new routing tables.

Length of Paths. Figures 8.(b) and 8.(c) shows the average length of the shortest paths for the SPFTR, A-SPFTR (for 2D tori) and up*/down* routing, also showing the average topological distance between switches. When considering only the shortest path for every pair of switches (the common case in the absence of failures), SPFTR and A-SPFTR achieve, on average, shorter paths than up*/down*. Additionally, as network sizes increases, the difference between the average path length and the average topological distance slightly increases. This is due to the fact that only one forbidden transition is allowed (2 VLs are used) and some route patterns would need to take longer paths to reach destination. Also, in Figures 8.(b) and 8.(c) we can see for 2D and 3D tori, respectively, the average lengths of the shortest paths and the average length when the complete set of alternative paths is considered.

Performance degradation with faults. Figure 8.(a) shows the performance degradation suffered by the network in the presence of faults when the SPFTR algorithm is used (similar results have been obtained for A-SPFTR). In the presence of faults every source node will use the shortest disjoint path that does not traverse any faulty link.

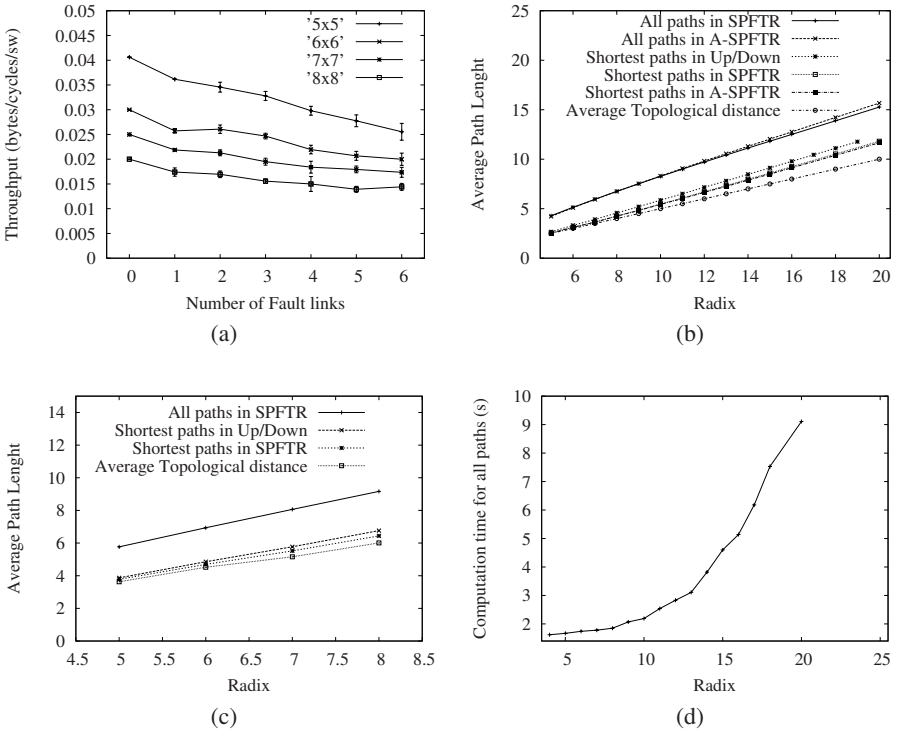


Fig. 8. (a) Degradation of performance with faults for SPFTR and A-SPFTR. (b) Average length of paths in 2D Torus. (c) Average length of paths in 3D Torus. (d) Computation Cost in 2D Torus. Radix stands for the number of switches per dimension.

For a particular number of faults, different random combinations of faults are injected. For every combination of faults the network is simulated, obtaining its throughput and displaying the average results. Error bars are shown for every number of faults.

As can be observed, throughput decreases as the number of faults increases. However, it can be noticed that performance degradation is relatively lower in larger networks. For instance, with 6 faulty links, throughput decreases in SPFTR up to 28 % for the 4×4 Torus and a 23 % for the 8×8 Torus. This is because the same number of faulty links affects a lower percentage of paths in larger networks. For up*/down* no results can be plotted as practically it does not tolerate any failure combination.

Computation Time. Finally, Figure 8.(d) shows the computation time of SPFTR (for A-SPFTR same results are obtained) for different 2D tori with different sizes. As can be observed, the computation time grows quadratically with the number of switches in the network. Notice that only the computation of forwarding tables is required. This is because SLtoVL tables and route patterns are already computed (tables for 2D Torus in Table 1). As an example, all routing information for a 20x20 Torus network was computed in less than 10 seconds using an Intel Xeon 3.06 GHz.

7 Conclusions

In this paper, we have proposed a scalable and effective methodology to design fault-tolerant routing strategies for IBA that is able to provide several disjoint paths between every source-destination pair, taking advantage of the APM mechanism provided by IBA. The proposed methodology uses some network resources (VLs and SLs) and guarantees deadlock-freedom by removing cycles in the CDG.

The resulting fault-tolerant routing strategy (referred to as SPFTR) on n-dimensional Torus networks uses only 2 VLs and 4 SLs, regardless of the network size. Furthermore, if we focus on 2D tori, the number of SLs required is reduced down to 3 to tolerate up to $2n - 1$ link failures. This can be considered an interesting result if we take into account that the up*/down* routing algorithm is not able to tolerate one single failure.

As future work, we plan to analyze other regular topologies as meshes or Multistage networks (MINs). We also plan to use this methodology in conjunction with a reconfiguration process to support a larger number of failures.

References

1. Abe supercomputer, `http://www.ncsa.uiuc.edu/`
2. Top500 supercomputer list (June 2007), `http://www.top500.org`
3. Adiga, N., Blumrich, M., Chen, D., et al.: Blue Gene/L torus interconnection network. IBM Journal of Research and Development 49 (March 2005)
4. Boden, N.J., et al.: Myrinet: A Gigabit-per-second local area network. IEEE Micro 15(1), 29–36 (1995)
5. Duato, J., Yalamanchili, S., Ni, L.: Interconnection Networks. An Engineering Approach. Morgan Kaufmann, San Francisco (2003)
6. Architecture. Specification Release 1.0, InfiniBand Trade Association^{TM} (October 2004)
7. Lysne, O., et al.: Simple Deadlock-Free Dynamic Network Reconfiguration. In: 11th Int. Conference on High Performance Computing (HiPC), December 19-22, 2004, Bangalore, India (2004)
8. Montañana, J.M., Flich, J., Robles, A., Duato, J.: A Transition-Based Fault-Tolerant Routing Methodology for InfiniBand networks. In: IPDPS 2004. Proc. of the 2004 Int. Parallel and Distributed Processing Symp, IEEE Computer Society Press, Los Alamitos (2004)
9. Mukherjee, S., Bannon, P., Lang, S., Spink, D.W.A.: The Alpha 21364 network architecture. IEEE MICRO (January-February 2002)
10. Petrini, F., et al.: The quadrics network (qsnet): High-performance clustering technology. In: HotI 2001. Proceedings of the 9^{th} IEEE Hot Interconnects, Palo Alto, California (August 2001) (original version), IEEE Micro January-February 2002 (extended version)
11. Schroeder, M.D., et al.: Autonet: A high-speed, self-configuring local area network using point-to-point links. Journal on Selected Areas in Comm. 9(8) (October 1991)