Reachability-Based Fault-Tolerant Routing*

J. M. Montañana, J. Flich, A. Robles, and J. Duato
Dept. of Computer Engineering (DISCA)
Universidad Politécnica de Valencia
Camino de Vera, 14, 46021–Valencia, Spain
E-mail: jmontana@gap.upv.es

Abstract

Currently, clusters of PCs are being used as a cost-effective alternative to large parallel computers. In most of them it is critical to keep the system running even in the presence of faults. As the number of nodes increases in these systems, the interconnection network grows accordingly. Along with the increase in components the probability of faults increases dramatically, and thus, fault-tolerance in the system, in general, and in the interconnection network, in particular, plays a key role.

An interesting approach to provide fault-tolerance consists of migrating on fly the paths affected by the failure to new fault-free paths.

In this paper, we propose a simple and effective fault-tolerant routing methodology, referred to as Reachability Based Fault Tolerant Routing (RFTR), that can be applied to any topology. RFTR builds new alternative paths by joining subpaths extracted from the set of already computed paths, thus being time-efficient. In order to avoid deadlocks, RFTR performs, if required, a virtual channel transition on the subpath union.

As an example of applicability, in this paper we apply RFTR to InfiniBand. Evaluation results on tori show that RFTR exhibits a low computation cost and does not degrade performance significantly.

1 Introduction

Over the recent years there is a trend in using clusters of PCs for building large systems. Some examples are cluster-based commercial Internet portal servers like AOL, Google, Amazon or Yahoo. Also, clusters of PCs are currently being used 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. In fact, the Columbia system, with 10,240 Intel® Itanium® 2 processors, is in the third 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 like Quadrics, InfiniBand [1], Myrinet, and Advance Switching. Among them, InfiniBand is a standard interconnect technology for communicating 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 mentioned Colombia 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 and BlueGene/L, use 2D and 3D tori, respectively.

These systems use a very large number of components. 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. This becomes more important when the system must be kept running indefinitely.

2 Related Work

In order to deal with faults, two fault models can be used: static or dynamic. In a static fault model it is assumed that all the faults are known in advance when the machine is (re)booted. Once a fault is detected, all the processes in the system are halted, the network is emptied and a management application is run to deal with the faulty compo-

*This work was supported by the Spanish MCYT under Grant TIC2003-08154-C06-01, and the JCC de Castilla-La Mancha under Grant PBC-05-007-2.
ponent. This fault model needs to be combined with checkpointing techniques, such as in the BlueGene/L supercomputer. In a dynamic fault model, once a new fault is found, actions are taken in order to appropriately handle the faulty component without stopping the network. For instance, a source node that detects a faulty component through a path can switch to a different routing path.

Basically, there are three ways to tolerate a fault in the interconnection network: component redundancy, fault-tolerant routing algorithms, and reconfiguration. Using component redundancy has been the easiest and most expensive way to provide fault tolerance. Components in the system are replicated and once a failed component is detected, it is simply replaced by its redundant copy.

A large number of fault-tolerant routing algorithms for multiprocessor systems have been proposed, especially for mesh and torus topologies. Some of these approaches are based on block faults [9], whereas others allow individual faults [8, 10]. In the former case, several healthy nodes must be marked as faulty, reducing the system’s processing capacity. To this end, several virtual channels must be used. An approach that minimizes the number of required virtual channels and tolerates a fairly large number of faults, at the expense of disabling some healthy nodes, was proposed in [11]. This algorithm is based on a static fault model and only requires two virtual channels per link. However, its main drawbacks are that a significant number of nodes must be disabled and that it does not support adaptive routing. To overcome these drawbacks, in [7] it is proposed a fault-tolerant methodology based on routing packets through intermediate nodes together with packet misrouting. However, this approach assumes a static fault model and requires some hardware support.

To sum up, most of the fault-tolerant routing strategies proposed in the literature are not suitable for clusters. This is because they often require certain hardware support that is not provided by current commercial interconnects [1]. Other strategies rely on the use of adaptive routing. However, they cannot be applied, as routing in clusters is usually deterministic. 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 either are not supported (e.g., Myrinet) or they cannot be dynamically selected at routing time (e.g., IBA).

An alternative [2] proposed for PC clusters consists of providing a certain number of disjoint paths between every source-destination pair. The main drawback of this approach is that the number of possible disjoint paths is bounded by the switch degree and strongly depends on the routing flexibility exhibited by the applied routing scheme. Additionally, in [3] a fast way to compute disjoint paths is provided. Unfortunately this solution is also bounded by the switch degree.

On the other hand, when using reconfiguration, once a fault is detected, a reconfiguration process is started. This process discovers the new topology and then computes the new routing information. This approach is suitable for switch-based networks (Myrinet, Quadrics, InfiniBand, and Advance Switching) in which the topology is defined by the end user. When using reconfiguration, any number of faults is tolerated as long as the network remains connected. Unlike static techniques, dynamic reconfiguration techniques do not require completely stopping the traffic in the network. However, some packets must be removed from the network and re-injected later, which could cause a strong degradation in performance during the reconfiguration time. Recent proposals [5] try to minimize the impact of the reconfiguration process on the performance of the system, at the expense of providing a specific hardware support, which prevents them from being applied to current commercial interconnects.

### 3 Motivation

In this paper, we are interested in the dynamic fault model applied to networks with deterministic routing, which is the common case in the commercial network technologies currently applied to clusters of PCs. In this scenario, there is no doubt that applying reconfiguration techniques is a good choice. However, from our point of view, reconfiguration should only be used when there is a need for changing the entire routing algorithm. This implies that all the paths for every source-destination pair must be computed again. Additionally, dynamic reconfiguration often requires to add/use new hardware resources in order to guarantee deadlock-freedom during the reconfiguration process.

However, notice that when failures appear in the network, they usually only affect to some paths. As an example, Figure 1 shows a $3 \times 3$ mesh using the Dimension Order Routing (DOR). For the sake of simplicity, we assume that end nodes will be attached only at switches A, C, and E. When the link $L$ fails, the path from A to E is affected, whereas the rest of paths are not affected by the failure. Thus, we would just need to compound a new path from A to E. However, notice that DOR is not able to provide a fault-free path from A to E. Therefore, the end nodes are logically disconnected. In this situation, a reconfiguration process should be launched.

Virtual channels can be used to improve the flexibility of a routing algorithm. In this sense, let us assume the previous example but now with several virtual channels in the network. In this situation, an illegal transition $(Y \rightarrow X)$ could be permitted (it does not lead to deadlock) if a virtual channel transition at the switch where the illegal transition would take place were performed [2, 12]. Indeed, a path performing a virtual channel transition can be viewed as two joined subpaths, each one on a different virtual channel or layer. Since each subpath does not introduce an illegal transition in its virtual channel, and virtual channels are used in an established order, deadlock-freedom is guaranteed.
Applying this concept to the example, now the new path from A to E can be computed by the original DOR routing and a virtual channel transition. In particular, the path A-B-C-D-E can be used by placing a virtual channel transition at switch C. Notice that by using virtual channel transitions the new path is fully compatible with the previous set of paths, thus guaranteeing that deadlocks can not arise. Thus, a full reconfiguration process is not needed (and simply the new routing info for the new path can be distributed without stopping the network traffic).

Using virtual channels to improve routing flexibility is not new. In fact there are proposals for routing through minimal paths by using virtual channel transitions [12]. So, these routings could be used to compute the new paths once a failure is detected. However, notice that this solution would require to compute the whole set of paths, thus not being time-efficient. Instead, in this paper we propose a different approach. Notice that the new computed path can be obtained from the set of paths previously computed (before the failure). Indeed, the new path can be viewed as the A-C and C-E subpaths joined. Thus, instead of computing the whole set of new paths (new routing algorithm) by using a reconfiguration process, we can extract from the already computed paths new ones only for those pair of nodes affected by the failure. This will let us to achieve the following benefits. Firstly, the required amount of routing information to be updated (at end nodes and/or switches) will be lower, leading to send less control data. Secondly, the traffic not affected by the failure will be left unmodified (a kind of local reconfiguration process will suffice). Thirdly, the method will take less time to compute (compared with a full reconfiguration method), as only affected paths will be recomputed. And, as a consequence, a smaller percentage of packets will be lost in the process.

Thus, the key issue consists of finding an effective methodology able to compute an alternative path for each path affected by the failure from the set of already computed paths. Moreover, this methodology should be able to compute the new paths in a time-efficient manner and use the smallest number of virtual channels.

To undertake these challenges, we propose a new fault-tolerant routing methodology, referred to as Reachability-Based Fault-Tolerant Routing (RFTR). The method will provide new paths by joining already computed subpaths and using virtual channel transitions when required. As a result, RFTR will be suitable to any topology, will tolerate dynamically a large number of failures with a very small number of virtual channels, and will exhibit a very low computational cost for any network size, minimizing the number of packets lost during the process of fixing the failure.

Finally, it has to be noted that the methodology will not depend on neither the hardware used for detecting failures, nor the way failures are notified. Anyway, as an example of applicability, the RFTR will be applied to IBA.

The rest of the paper is organized as follows. In Section 4, the RFTR methodology will be described. In Section 5, RFTR will be applied to InfiniBand. Then, in Section 6, the methodology will be evaluated in terms of fault-tolerance, performance, and resource needs. Also, RFTR will be compared to a reconfiguration method. Finally, in Section 7, some conclusions will be drawn.

4 Description of RFTR

In this Section, we will describe RFTR in detail. The methodology is based on the concepts of direct and indirect reachability. Thus, we will first introduce both concepts and then will present the methodology.

4.1 Direct and Indirect Reachability

Given a routing algorithm and a pair of switches (A and B), it is said that switch B is directly reachable from switch A if the routing algorithm provides a path from A to B. Similarly, switch A is indirectly reachable from switch B if the routing algorithm provides a valid path from A to B. Notice that reachability can be defined regardless of the type of routing algorithm used (adaptive or oblivious). However, in this paper we are interested in oblivious/deterministic routing. Moreover, an oblivious routing may provide several paths for some pair of nodes. Thus, it can be viewed also as a set of deterministic paths. We will select just one path for

---

**Table 1:** Information for A-C and C-E paths from Figure 1.

<table>
<thead>
<tr>
<th>Src</th>
<th>Dst</th>
<th>Reach</th>
<th>Trans.</th>
<th>Input Link</th>
<th>Hops</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>C</td>
<td>C</td>
<td>-</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>C</td>
<td>C</td>
<td>-</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>E</td>
<td>D</td>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>E</td>
<td>E</td>
<td>-</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

**Table 2:** Table information for A-C and C-E paths from Figure 1.

<table>
<thead>
<tr>
<th>Src</th>
<th>Dst</th>
<th>Reach</th>
<th>Trans.</th>
<th>Output Link</th>
<th>Hops</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>C</td>
<td>B</td>
<td>-</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>C</td>
<td>A</td>
<td>-</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>E</td>
<td>D</td>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>E</td>
<td>C</td>
<td>-</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
each source-destination pair. Each path will consist of a list of links and switches. Based on this view, let us define the reachability concept related to a given deterministic path: Every switch listed in a path is directly reachable from the source switch of the path (the switch which the source node is attached to) and indirectly reachable from the destination switch of the path (the switch which the destination node is attached to) 

In order to achieve a simple and fast computation method we create two tables, one for identifying directly reachable switches and one for identifying indirectly reachable switches. As an example, the first table (Figure 2.1(a)), referred to as direct reachability table (DRT) allocates entries for each path in Figure 1.(a).

Each path defines an entry on the table for every visited switch. These entries will be consecutively allocated in the table in the order the switches would be visited starting from the source switch (switch with smaller number of hops from source switch first). For each entry, the table contains the source and destination switches, the input link ID used at the reachable switch, the number of hops needed to arrive from the source to the reachable switch, and the number of transitions (if any) of virtual channels required along the path to arrive from the source to the switch (initially none).

The second table (Figure 2.1(b)), referred to as indirect reachability table (IRT), is similarly defined as DRT, but the info is acquired in opposite order, i.e., starting from the destination switch. In the same sense, each entry contains the source and destination switches, the output link ID used at the reachable switch, the number of hops needed to arrive to destination from the reachable switch, and the number of transitions (if any) of virtual channels required along the path to arrive from the switch to the destination (initially none).

4.2 RFTR Methodology

Initially, when the system is started, the method computes the DRT and IRT tables from the deterministic routing used. Then, whenever the notification of a new failure arrives, the method is triggered. The main goal of the methodology is to compute, in a time-efficient manner, alternative paths for every pair of source-destination nodes whose path have been involved in a failure. Thus, as a first step, it will identify the paths affected by the failure. This is achieved by sweeping both tables (DRT and IRT). The method will support switch and link failures. In the case of a switch failure, the switch ID will be provided. In the case of a link failure, the link ID and the ID of one switch attached to the failed link will be provided. Whenever the failed link and/or switch is found in an entry (the switch appears as reachable and the failed link appears as input link for DRT or output link for IRT), the associated path is annotated as affected. Then, all the entries defined by the given path from the failed link until destination in DRT are removed (they are no longer reachable using the path). Also the entries from source until the failed link in IRT are removed.

Once a given path is annotated as affected by the failure, RFTR will build two sets of switches. The first one will be compounded by the switches directly reachable from the source of the affected path, and the second one by the switches indirectly reachable from the destination of the affected path. In order to build the set of directly reachable switches, the method will sequentially sweep the DRT searching for directly reachable switches from the source. The set will only include the indexes to the DRT table, not the switch IDs. The same will be done with the IRT table in order to build the set of indirectly reachable switches. Note that a switch may appear in any pair several times as can be reached by the source (or indirectly reached by the destination) through different paths. Also, each replica may require different number of virtual channel transitions and may be at a different distance from the source (or destination). The method will keep the replicas on each set in order to get the best option when computing the final path.

As an optimization, notice that both sweeps (the one for detecting affected paths and the one for computing the sets) can be done at the same time.

Once sets are computed for a given affected path, the method will select the appropriate intermediate switch from the intersection of both sets. The new path will consist of the union of two subpaths, one from the source to the intermediate switch, and the another one from the intermediate switch to destination.

The method uses the following criteria for selecting the intermediate switch: the final path will have the minimum number of virtual channel transitions and, in case of a tie, the shortest path will be selected. In order to compute the number of required virtual channel transitions, the method will take into account the number of virtual channel transitions already used in each subpath (extracted from both tables) and if a virtual channel transition is required (according to the routing algorithm applied) 

Once a new path is obtained, the old path is removed and the new reachability info provided by the new path is added to DRT and IRT. Once the new paths are computed, the new routing info must be distributed to switches and end nodes. However, how info is distributed and routing tables are updated depend on the RFTR implementation and the used network technology. Finally, it has to be noted that once all the new paths are computed, DRT and IRT tables are ready to be used in the presence of a new failure.

In the next section, we will apply RFTR to InfiniBand.

Notice that the definition does not include all the reachability info provided by the paths, as reachability among intermediate switches is not considered. However, as it will be seen in the evaluation, this will not impact on the fault tolerance properties of RFTR.

Initially, paths have no virtual channel transitions. They may appear later as long as the methodology computes new paths to cope with failures.

To compute this, the method needs to know the input and output ports used at the intermediate switch. They are allocated in the DRT and IRT tables.
5 RFTR on InfiniBand

In the previous section, we described the methodology independently of the architecture of the network. As an example of applicability, in this section we will adapt the RFTR methodology to InfiniBand (IBA). For this, we will first describe how routing and virtual channels are managed in IBA. Then, we will describe how failures are detected and managed, and how the entire process can be integrated with RFTR.

5.1 Routing and Mapping Conflicts

In IBA, routing and virtual channel (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. Every switch has a forwarding table which provides only one output port (and always the same) for each destination.

In order to allow different paths from the same source-destination pair, IBA allows the use of virtual addresses [1]. Therefore, the same destination node can be identified with different IDs. From the subnet point of view, each ID is different (the forwarding table may supply a different output port to each ID), however, for the destination point of view each ID is the same. IBA allows up to 7 bits of the local ID (LID) to be used as virtual address (masked at destination). Therefore, up to 128 virtual addresses can be used per destination.

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 it cannot be changed by the switches. Therefore, we should also assign the proper SL that must be used for a given path.

However, the fact of fixing a path with an unique SL and the use of several Virtual Lanes may lead to a mapping conflict. It occurs when two packet s labeled with the same SL enter a switch through the same input port, and they need to be routed through the same output port but along different VLs. The problem is that the SLtoVL mapping table does not consider the input VL in order to determine the output VL. Figure 3 shows an example. At switch R a mapping conflict arises due to the fact that it is not possible to distinguish both paths because they are labeled with the same SL. It has to be noted that this problem arises only when there are paths that use different VLs. For example, the path B uses VL0 until switch Q and then uses VL1.

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 mapping tables.

Figure 3. Mapping conflict example.

5.2 Fault Detection in InfiniBand

An IBA network is divided into subnets (connected through routers). On each, end nodes and routers are connected through switches. The Subnet Manager (SM) is the entity that discovers all the devices on an IBA subnet, configures them, and detects any change in the subnet’s topology. The SM is allocated in a particular node in the subnet. A change in the topology can be due to devices being added or removed, or because of a failure. In each network device there exists a Subnet Manager Agent (SMA), which is responsible for monitoring port’s link integrity.

The IBA standard defines two complementary mechanisms for detecting changes. On the one hand, the SM will perform periodic sweeps of the subnet requesting information to each SMA associated to each component. The frequency of these sweeps is not defined by the IBA standard, thus it can be adjusted accordingly to parameters like the size of the subnet or the desired detection time for changes. On the other hand, each SMA could actively notify to the SM whenever a change is detected. The second method can be optionally implemented by vendors.

5.3 Applying RFTR to InfiniBand

Figure 4 shows the steps followed by the methodology once integrated in IBA. At the first stage the SM carries out sweeps in order to detect changes or failures in the subnet. When the SM encounters a failure, it will launch the RFTR methodology. The method will work as described in section 4.2. However, as SLs are used in InfiniBand, a new criteria for selecting the appropriate intermediate switch will be used. In particular, higher priority will be given to those paths that do not introduce a mapping conflict (thus not requiring an additional SL). In case of a tie, the switch that leads to a shorter final path will be preferred.

In order to differentiate from the subnet point of view both paths (the new one and the failed one), the methodology will assign a new virtual address (LID) to each one.
Evaluation

According to the new routing info. new LIDs, thus being appropriately routed by the switches on the reception, the new injected data packets will use the end nodes the new LIDs corresponding to the new paths. When the updated switch, it will send to the SMAs placed in all the might be discarded.

On the other hand, if the size of the information sent will be different for each switch or end node (even some switches will do not receive new routing information). To send such an information a special routing mechanism (Directed-Route) is used. According to the IBA specs, Directed-Route is used for routing control packets through a reserved VL. The entire path of each packet is established at the header by specifying all the output ports along the switches to be crossed. As it defines the routing paths per port, the failure will be avoided.

When the SM has completed the sending of the new routing info, it will wait for the reception of a confirmation from all the switches whose tables have been modified. This is required in order to ensure that messages can be appropriately routed when using the new LIDs. Otherwise, they might be discarded.

Once the SM has received the confirmation from every updated switch, it will send to the SMAs placed in all the end nodes the new LIDs corresponding to the new paths. On the reception, the new injected data packets will use the new LIDs thus being appropriately routed by the switches according to the new routing info.

The method will be fully analyzed in the next section.

6 Evaluation

In this section, we will evaluate the proposed fault-tolerant methodology (RFTR) when applied to InfiniBand. To this end, we will analyze its fault tolerance degree, the required resources and computation time, and the exhibited performance. Also, we will compare RFTR with other fault tolerant routing methodologies, such as TFTR and SPFTR [2, 3], and with respect to the alternative of applying a reconfiguration process, such as the Simple Reconfiguration method [4]. TFTR and SPFTR are fault-tolerant mechanisms that statistically provide a limited number of disjoint paths (they are previously computed) to cope with failures. However, RFTR is able to dynamically provide new paths as long as failures appear. Simple reconfiguration is a dynamic network reconfiguration method which has proved to be a fast mechanism that works for any topology and between any pair of old and new routing functions.

To evaluate the mechanism, we have developed a detailed simulator that allows us to model the network at the cycle level. The simulator models an IBA network, following the IBA specifications [1]. In what follows, we will first present the evaluation model, describing all the simulation parameters and the main features of the IBA. Then, we will present the analytical and evaluation results.

6.1 Simulation and Environment Model

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. The routing time at each switch will be set to 100 ns. This time includes the time to access the forwarding tables, the crossbar arbiter setup time, the time to set up the crossbar connections.

VLs can be used to form separate virtual networks. We assume that the crossbar supplies separate ports for each VL. Buffers will be used both at the input and the output side of the crossbar. Buffer size will be fixed in both cases to 1 KB.

Links in InfiniBand are serial. In the simulator, the link injection rate will be fixed to the 1X configuration [1]. 1X cables have a link speed of 2.5 Gbps. Therefore, a bit can be injected every 0.4 ns. With 8/10 coding [1] a new byte can be injected into the link every 4 ns.

The IBA specification defines a credit-based flow control scheme for each virtual lane with independent buffer resources. Additionally, the virtual cut-through switching technique is used.

We have considered torus network topologies. In particular, we have analyzed 2D tori with different sizes, from 16 switches (4x4) up to 196 switches (14x14). Also 3D tori with 64 switches (4x4x4) and 216 switches (6x6x6) have been analyzed. In all the cases, two nodes are attached to each switch.

RFTR is evaluated using up*/down* routing as the underlying routing algorithm. We have selected this routing scheme because, unlike dimension-order routing, it is able to provide some alternative paths between every source-destination pair. In most of the cases, two VLs will be used by RFTR, thus, at maximum, only one transition will be allowed to each path. In order to take maximum benefits from the two available VLs, the VL used for packets being routed through paths that do not require to perform a VL transition (initially, in the absence of failures, all the paths have no VL transitions) will be randomly selected. For paths with a VL transition, all the packets will be injected only into the first VL.

In the analysis, we will only consider faults of links connecting switches. RFTR will be evaluated in terms of fault tolerance (under different sequences of link failures) and network performance. In particular, network perfor-
The fault tolerance degree of RFTR is obtained by analyzing, for a certain number of faults, all the fault combinations that keep the network physically connected. The method is \( n \)-fault tolerant if it provides for any combination of \( n \) failures a valid path for each source-destination pair. Thus, we should analyze all the possible fault combinations for every number of faults. However, as the number of faults increases, the number of possible fault combinations increases exponentially. Thus, from a particular number of faults upwards, it is impossible to explore all the fault combinations in a reasonable amount of time, specially in medium- and large-sized networks. To overcome this problem, we will evaluate all the combinations on small network sizes, as performed in [7]. When the number of fault combinations to be analyzed is very large from a computational point of view, we will perform a statistical analysis (as followed in [7]), in which a representative subset of the total number of fault combinations is analyzed.

Table 1 shows the results obtained when considering up to eight failures in \( 4 \times 4 \) and \( 3 \times 3 \times 3 \) tori. For every number of faults, the number of fault combinations analyzed can be observed. As can be seen, RFTR is able to tolerate all the fault combinations. Exactly, we can guarantee (100\% of fault combinations analyzed) that RFTR is 8-fault tolerant in the \( 4 \times 4 \) Torus, and 4-fault tolerant in the \( 3 \times 3 \times 3 \) Torus, whereas from 5 faults upward we only can state it in statistical terms in the \( 3 \times 3 \times 3 \) Torus. This is a reasonable fault-tolerance degree taking into account the network sizes considered in this paper (it represents more than 5\% of failed links), and that the mean time between failures is much greater than the mean time to repair. When compared RFTR with respect to TFTR and SPFTR, we can observe that RFTR is able to support an unbounded number of failures, whereas TFTR and SPFTR only can support up to 3 failures in 2D tori and 5 failures in 3D tori.

### 6.2 Analytical Results

#### 6.2.1 Fault Tolerance

The fault tolerance degree of RFTR is obtained by analyzing, for a certain number of faults, all the fault combinations that keep the network physically connected. The method is \( n \)-fault tolerant if it provides for any combination of \( n \) failures a valid path for each source-destination pair. Thus, we should analyze all the possible fault combinations for every number of faults. However, as the number of faults increases, the number of possible fault combinations increases exponentially. Thus, from a particular number of faults upwards, it is impossible to explore all the fault combinations in a reasonable amount of time, specially in medium- and large-sized networks. To overcome this problem, we will evaluate all the combinations on small network sizes, as performed in [7]. When the number of fault combinations to be analyzed is very large from a computational point of view, we will perform a statistical analysis (as followed in [7]), in which a representative subset of the total number of fault combinations is analyzed.

Table 1 shows the results obtained when considering up to eight failures in \( 4 \times 4 \) and \( 3 \times 3 \times 3 \) tori. For every number of faults, the number of fault combinations analyzed can be observed. As can be seen, RFTR is able to tolerate all the fault combinations. Exactly, we can guarantee (100\% of fault combinations analyzed) that RFTR is 8-fault tolerant in the \( 4 \times 4 \) Torus, and 4-fault tolerant in the \( 3 \times 3 \times 3 \) Torus, whereas from 5 faults upward we only can state it in statistical terms in the \( 3 \times 3 \times 3 \) Torus. This is a reasonable fault-tolerance degree taking into account the network sizes considered in this paper (it represents more than 5\% of failed links), and that the mean time between failures is much greater than the mean time to repair. When compared RFTR with respect to TFTR and SPFTR, we can observe that RFTR is able to support an unbounded number of failures, whereas TFTR and SPFTR only can support up to 3 failures in 2D tori and 5 failures in 3D tori.

### 6.2.2 Resources Needed

Also, Table 1 shows the number of resources required by RFTR, TFTR, and SPFTR on IBA. As shown, only 2 VLs and up to 6 SLs are required by RFTR for tolerating 4 failures in 2D tori. However, a third VL is required from 5 faults upward, also increasing the number of SLs up to 8 (tolerating 8 failures). On the other hand, only 2 VLs are required in 3D tori, regardless of the number of faults. Further...

---

Table 1. Fault tolerance achieved and resources required by RFTR, TFTR and SFTR.

<table>
<thead>
<tr>
<th>Num. of faults</th>
<th>RFTR</th>
<th>TFTR</th>
<th>SPFTR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Evaluated cases</td>
<td>Percentage of non supported cases</td>
<td>Max VLs</td>
</tr>
<tr>
<td>1</td>
<td>81*</td>
<td>0.00%</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3,240*</td>
<td>0.00%</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>85,320*</td>
<td>0.00%</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1,663,740*</td>
<td>0.00%</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>25,621,596*</td>
<td>0.00%</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>6,490,804*</td>
<td>0.00%</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>17,386,083*</td>
<td>0.00%</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>16,082,127*</td>
<td>0.00%</td>
<td>2</td>
</tr>
</tbody>
</table>

*Server faults have been analyzed.
ther, the maximum number of required SLs is 5. Considering that IBA specifies a maximum of 15 VLs and 16 SLs, it can be concluded that the proposed methodology can be used to support a reasonably large number of failures, using a number of resources smaller than the maximum provided by IBA. We have corroborated by statistical analysis that resource requirements in larger 2D and 3D tori are the same. From the comparison with other methodologies we can conclude that RFTR is able to achieve a much higher fault-tolerant degree, at the expense of requiring a number of SLs slightly greater than that required by TFTR and SFTR.

### Table 2. Memory requirements.

<table>
<thead>
<tr>
<th>Topology</th>
<th>Raids</th>
<th>Memory for tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D Torus</td>
<td>7</td>
<td>38 KB</td>
</tr>
<tr>
<td>8</td>
<td>75 KB</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>136 KB</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>232 KB</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>584 KB</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1267 KB</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Topology</th>
<th>Raids</th>
<th>Memory for tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D Torus</td>
<td>3</td>
<td>6350 Bytes</td>
</tr>
<tr>
<td>4</td>
<td>50 KB</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>239 KB</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>890 KB</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2661 KB</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>6145 KB</td>
<td></td>
</tr>
</tbody>
</table>

We have also evaluated the memory resources required by the Subnet Manager to support the IRT and DRT tables, as they are the most important data structures used by the RFTR methodology. The memory requirements will directly depend in the number of entries on the tables. As commented in Section 4.1, each table provides an entry for every visited switch along each routing path. Therefore, the total amount of entries on each table will be imposed by the average path length times the number of paths. Therefore, the size will be similar to the size required to store all the routing paths. As shown in Figures 5.(a) and 5.(b), the average path length depends on the number of faults. However, its increment as the number of faults increases is not significant. Hence, in practical terms, it would reasonably allow us to define the table size independently of the network failures. The total amount of memory required by RFTR to support both reachability tables is shown in Table 2.

### 6.2.3 Computation Time

Figures 5.(a) and 5.(b) show the computation time (based on an AMD Opteron 1.4 GHz) of RFTR for different consecutive link failures, for 2D and 3D torus with different sizes, respectively. Plotted values correspond to the average computation time obtained from evaluating 1000 fault combinations for every number of faults (when the number of combinations is greater than 1000). Error bars are plotted. As can be observed, the computation time is very low (lower than one second), and linearly increases with the number of failures. This is because as long as the number of faults increases, the number of affected paths increases too. The low computation time exhibited by RFTR contrasts with that exhibited by reconfiguration techniques. In particular, assuming DFS up*/down* routing [6], simply the computation time of new routing tables required by reconfiguration dramatically increases with the network size, as shown in Figure 5.(c) for 2D and 3D tori under one link failure.

### 6.2.4 Length of Paths

Figure 6.(c) shows for 3D tori, the average length of paths provided by RFTR. Also, for comparison purposes, the average length of paths provided by the reconfiguration process is plotted. As a reference, the average topological distance between switches is also shown. We can observe that the difference between the average path length and the average topological distance slightly increases as the number of failures increases. Moreover, the average path length values obtained by reconfiguration are slightly lower than that obtained by RFTR. This is due to the fact that the methodology tries to minimize the number of VLs used (usually 2 VLs, and in some cases up to 3 VLs). This leads to select, in some cases, intermediate switches that are not in the minimal path, from source to destination, therefore increasing the average path length.

### 6.3 Performance Evaluation

Now, we analyze how RFTR influences network performance for different numbers of link failures. In particular, we will evaluate the performance degradation after the occurrence of a certain sequence of failures. For comparison purposes, we will also evaluate the performance achieved by the reconfiguration process launched after every fault. To maximize performance, we assume that the configuration process computes a new spanning tree (required by up*/down* routing) by selecting as root the switch with the highest average distance to the rest of switches [6].

We have run 100 simulations for each number of faults and for each analyzed strategy (RFTR and reconfiguration). In each of them, failed links were selected randomly. Average obtained throughput and error bars can be seen in Figure 6.(a). As can be observed, throughput decreases in both cases as the number of faults increases. However, notice that the difference is negligible. This is a good result because, applying our methodology (RFTR), we are achieving similar network performance to that provided when applying reconfiguration, but without needing to compute new routing tables. In RFTR, only some routing table entries need to be updated (see Figure 6.(b)). Notice that DFS has been used to compute the initial paths of RFTR. This algorithm exhibits a high computation time because it searches for a set of paths that provides the best traffic balance in the network, thus achieving high throughput. By using RFTR in combination with DFS we can still guarantee high network throughput in the presence of failures with a very low computation time.

Finally, we are interested in analyzing RFTR performance during the entire process, since a fault is injected...


Figure 5. Average computation time (a,b) for RFTR, (c) for DFS and RFTR.

Figure 6. (a) Network throughput degradation. (b) Average routing info sent after the first failure. (b) Average path length when using RFTR and Simple Reconfiguration.

Figure 7. (a,b) Accepted traffic when using RFTR and Simple Reconfiguration algorithms, respectively. (c) Accumulated lost bytes using RFTR and Simple Reconfiguration algorithms.

until routing tables are updated. In particular, we will analyze how the network traffic is temporarily affected by the process of detecting the fault, computing new paths, updating forwarding and SLtoVL tables at switches, and notifying the new LIDs to all the nodes. For comparison purposes, we will also analyze the transitory network behavior when applying the reconfiguration process.

Figure 7.(a) shows the accepted and lost traffic over time when a failure is injected and RFTR is applied for an injection rate near saturation. On the other hand, Figure 7.(b) shows the traffic evolution when the same failure is injected and the reconfiguration process is applied for the same injection rate. Figure 7.(c) shows the accumulated lost bytes along the process for both algorithms. The meaning of the vertical lines is the following: the first one represents the time when the failure is injected and the time when the SM receives the notification of the failure (only one line because the information is received 18.25 μs after the injection of the failure, and the difference is too small to be represented with two lines in the plot). The second line represents the time when the SM finishes the computation of the new routing info and the time when all switches and nodes have received all the new routing information (only one line because the information is received 14.60 μs and 66.46 μs for RFTR and Simple Reconfiguration, respectively, after computation process finishes, and it is also too small difference as to be represented with two lines in the plot). These computation times required by RFTR and Simple Reconfiguration have been estimated from real experiments (computed on an AMD Opteron 1.4 GHz.).

The times between injection and detection of the failure are the same in RFTR and Simple Reconfiguration, because both simulations assume the same traffic rate, the same frequency of sweeps for fault detection, and the failure is injected at the same time in both simulations. We have observed that the time from the end of computing the new routing info until the end of the global process, keeps constant for RFTR, independently of the traffic rate, while for Simple
Reconfiguration increases with the traffic rate because it requires to solve all dependencies with old traffic. For the evaluation we have considered that the control packets are sent in an independent virtual channel with the highest routing priority. Also, notice that the useful traffic is not significantly affected by the process. It has to be noted that the shown case corresponds to a worst case, that is, the failed link is near the root switch of the spanning tree used by up*/down*, thus affecting a large percentage of paths.

Using RFTR the amount of routing information sent will depend on the number of paths affected by the failure. Figure 6.(b) shows the percentage of the routing information sent to every switch after the occurrence of a link failure in the 7x7 Torus. As assumed before for the link failure selection, we are considering a worst case. Also, Figure 6.(b) shows the amount of routing info sent to every switch for a total reconfiguration. We can observe that RFTR sends a very small fraction of routing info compared to the routing info sent by a full reconfiguration.

7 Conclusions

In this paper, we have proposed a new fault-tolerant routing algorithm suitable for clusters of PCs. The methodology, referred to as Reachability-Based Fault-Tolerant Routing (RFTR), is based on the direct and indirect reachability concepts. The proposed methodology consists of computing new paths only for those source-destination pairs affected by failures, by joining two valid subpaths. These subpaths are joined in a certain intermediate switch, which is obtained by intersecting the set of directly reachable switches from source and the set of indirectly reachable switches from destination. At the intermediate switch a virtual channel transitions may be required in order to guarantee deadlock freedom.

The main contribution of this paper is to show that the proposed fault-tolerant routing methodology is able to cope with network failures in an effective and time-efficient manner. This methodology avoids the delays and traffic overhead usually associated to the application of network reconfiguration techniques (widely used in clusters of PCs), while still being suitable to be implemented on commercial network technologies currently used in clusters of PCs. Only a few alternative paths must be computed and some routing table entries must be updated.

Analysis results show that RFTR is able to tolerate a reasonable number of faults (up to 4 failures in 2D tori and 8 failures in 3D tori) using just two virtual channels. However, unlike previous proposals, it could tolerate dynamically a larger number of faults, as long as the network remains physically connected, if more virtual channels were used. Moreover, performance degradation due to fault occurrence is similar to that exhibited by a full reconfiguration process.

In the present work, we are updating the new routing info, when the computation process finishes. As a future work, we plan to optimize the RFTR, sending the routing info of each new path as soon as it is computed, in order to reduce the lost byte rate during the computation time of RFTR. Also, we will consider the possibility of preventing the nodes from injecting packets (using paths which visit a failure) after the detection of the failure.

References