A Multipath Routing Method for Tolerating Permanent and Non-Permanent Faults[⋆]

Gonzalo Zarza, Diego Lugones, Daniel Franco, and Emilio Luque

Computer Architecture and Operating Systems Department,
University Autonoma of Barcelona, Spain

{gonzalo.zarza, diego.lugones}@caos.uab.es
{daniel.franco, emilio.luque}@uab.es

Abstract. The intensive and continuous use of high-performance computers for executing computationally intensive applications, coupled with the large number of elements that make them up, dramatically increase the likelihood of failures during their operation.

The interconnection network is a critical part of such systems, therefore, network faults have an extremely high impact because most routing algorithms are not designed to tolerate faults. In such algorithms, just a single fault may stall messages in the network, preventing the finalization of applications, or may lead to deadlocked configurations.

This work focuses on the problem of fault tolerance for high-speed interconnection networks by designing a fault-tolerant routing method to solve an unbounded number of dynamic faults (permanent and non-permanent). To accomplish this task we take advantage of the communication path redundancy, by means of a multipath routing approach. Experiments show that our method allows applications to finalize their execution in the presence of several number of faults, with an average performance value of 97% compared to the fault-free scenarios.

1 Introduction

High-performance computer systems have opened a trend in modeling the modern society daily behavior and life style by means of applications and services such as molecular dynamics simulations, DNA sequencing, weather forecasting, geological activity studies, etc. Even a simple Google search is based on high-performance computer (HPC) systems [1].

The steady increase in complexity and number of components of HPC systems leads to significantly higher failure rates. Because of this, and due to the long execution times of computationally intensive applications, various computer systems show a Mean Time Between Failures (MTBF) smaller than the execution time of some of these applications [2]. This means that at least one failure will probably occur during the execution of such applications.

[⋆] Supported by the MEC-Spain under contract TIN2007-64974

Questions arise from the analysis of these situations such as: how do the failures (and their duration) affect these HPC systems? Are such systems able to maintain their operation and performance standards in spite of failure occurrences? If they are not, what should the solution be? What are the best options to achieve fault tolerance and system service continuity?

Undoubtedly, system performance is closely related to the robustness of the fault tolerance mechanisms of the network. For this reason, high-speed interconnection networks (HSINs) must avoid significant performance degradations and, above all, allow applications to finalize their executions while preventing abnormal behaviors, even in the presence of multiple faults regardless of their duration.

In this work, we focus on the fault tolerance problem for HSINs due to their primary role as the linking element of HPC systems. There are three main approaches that could be chosen to achieve this goal: component redundancy, network reconfiguration, and fault-tolerant routing algorithms [3]. The component redundancy approach is often used in some systems but the high extra cost of the redundant spare components is an important drawback. The second approach stops the network and reconfigures the routing tables in case of a network fault in order to adapt them to the new topology after the fault. This approach is very flexible and powerful but at the expense of killing network performance. Routing algorithms designed for fault tolerance looks for alternative paths when a fault disables the original path used to communicate a pair of source-destination nodes. This last approach could be outlined as the most interesting and suitable option but, at the same time, the design of fault-tolerant routing algorithms implies great challenges.

The failure of a single network component disables resources for variable time periods, generating congestion problems in their surroundings. Adaptive routing is a natural solution to this problem and therefore adaptive routing algorithms designed for fault tolerance could be outlined as a viable option.

For this reason, we focus our work in the problem of fault tolerance for HSINs by means of the adaptive routing approach. In this paper we present a method that exploits communication path redundancy through an adaptive multipath routing policy with the aim of solve a certain number of permanent and non-permanent link failures. The method is based on source-destination communication path information and consists of three phases. The first phase is responsible for on-line fault diagnosis and uses physical level monitoring at the intermediate nodes along the source-destination path. If a message encounters a faulty link as it progresses towards its destination, the second phase immediately reroutes the message to the destination by an alternative path. In the third and last phase, the source node is notified about the link failure in order to disable the faulty path, and to establish new paths for the following messages to be sent to that destination. At a first stage, failures are considered and treated as non-permanent. If a failure persists over time, its status changes from non-permanent to permanent.

In a previous work, we have introduced a method capable of supporting dynamic faults appearing at random during network operation [4]. The method allows the system to remain operational while measures are taken to circumvent the faulty components, however, it was not intended to treat transient and intermittent faults.

The main contribution of this work is the ability to treat permanent and non-permanent dynamic faults while treating network congestion caused by faults. Moreover, the method only notifies the event of a failure to the sources nodes that try to send messages by faulty links, instead of distributing status-information over the network, and therefore reducing traffic overhead.

Experimental results show an average performance higher than 97% for a set of test scenarios with several faults in a 1024 nodes bidimensional torus network for standard traffic patterns.

The rest of the paper is organized as follows. Section 2 introduces the related work. Section 3 describes the multipath routing method for tolerating dynamic faults, details its behavior and explains the treatment of non-permanent faults. Evaluation environment, test scenarios and results are presented in Section 4. Finally, some conclusions and future work are drawn in Section 5.

2 Related Work

Many research studies have been published in the field of fault tolerance for interconnection networks throughout the past few decades. The vast majority assumes the existence of diagnostic techniques, and focus on how the availability of information obtained from these diagnoses can be used to develop robust and reliable routing algorithms. This means that diagnoses problem is not addressed by those methods; static fault models are used which means that all the information about faults need to be known in advance; and they assume that there are mechanisms to correctly distribute this information to network nodes.

In [5] it is presented some interesting research in the area of component redundancy. On the side of the network reconfiguration approach, there are works based on deterministic routing methodologies for tori and meshes [6], and others that achieve error detection and recovery for k-ary n-cube topologies [7]. The latter proposal tolerates non-permanent faults but relies on table based routing strategies and packet injection is required to be temporarily stopped during a global reconfiguration phase.

In [8] the author proposes a widely used methodology for designing fault-tolerant routing algorithms. Several works have achieved good performance results based on the aforenamed methodology, but almost all of them use a static fault model.

Some routing strategies using static fault models have been proposed for different network topologies like k-ary n-tree [9] and direct networks [10]. The authors of the latter work suggest the use of intermediate nodes to circumvent faults but fault detection, information distribution, and checkpointing are assumed to be provided by the interconnection network. The use of intermediate nodes was first proposed by Valiant for the purpose of traffic balancing [11].

One of the only proposals capable to deal with dynamic faults was proposed in [12] as an evolution of [13]. The method is based on a turn-model variation, needs five virtual channels to support fully adaptive routing, and packet drops are allowed under certain situations. Moreover, status-information must be distributed through control messages then rerouting decisions must be taken based on such information. All these actions have an extra cost.

In [14] the authors introduce a fault-tolerant routing methodology that sacrifices a certain number of healthy nodes in order to use no more than two virtual channels, and to reduce the routing time.

3 Multipath Fault-Tolerant Routing

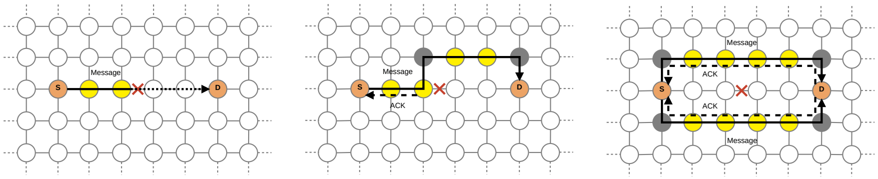
The multipath fault-tolerant routing method presented in this paper is largely based on the Distributed Routing Balancing (DRB) concept introduced in [15].

This proposal differs from the work found in the literature through its combined features. More precisely, it is able to combine support for a dynamic fault model and multipath routing. Even more, the method introduces a novelty approach, addressing system continuity functioning and network performance degradation problems at the same time. To accomplish these tasks, we use a non-global scheme to distribute real-time paths information in order to chose the best source-destination paths. Furthermore, our method does not require global reconfiguration or stopping packet injection at any time, using a limited number of virtual channels.

Conceptually, our proposal consists of three steps. In the first step, the failure in the original path is discovered when a message tries to use a faulty link, as could be seen in Fig. 1(a). In the second step, shown in Fig. 1(b), the message is rapidly rerouted to its destination through an alternative path to allow system service continuity. This action is intended to be a fast and temporary response to failures therefore may not be the optimal solution. For this reason, we include a third step which seek to reconfigure new paths to improve performance and ease routing paths. In this step, shown in Fig. 1(c), the source node is notified about the discovery of a link failure in the path, in order to disable the faulty path and reconfigure new paths for the following messages. Once those new paths have been configured, their latency values are recorded and sent back to the source node (from the destination), in order to calculate the number of alternative paths that must be used according to the network traffic burden.

The configuration and use of simultaneous alternative paths between source and destination nodes allow the method to deal with link failures. The use of such simultaneous paths provides path redundancy and allows performance improvements by means of communication balancing and distribution.

The alternative paths are created using intermediates nodes which can be used for two different purposes. One of these purposes is to allow the segmentation of the original source-destination path when encountering faults on the



(a) Step 1 (b) Step 2 (c) Step 3

Fig. 1. Example of behavior.

fly (due to the dynamic fault model), in order to circumvent the faulty areas as shown in Fig. 1(b). The second purpose of intermediate nodes is to be used as scattering and gathering areas from source and destination nodes when knowing the location of the faults, as shown in Fig. 1(c). From these scattering/gathering areas, alternative paths are built based on intermediate nodes carefully chosen to ensure that they are not in the original path, using the available links in routers. The set of alternative paths between each source-destination pair is called *multipath* or *metapath* [15].

The intermediate nodes are chosen according to their distance to the nodes that have detected the faults or to the source and destination nodes, as appropriate. The nodes of 1-hop distance are considered first, then nodes of 2-hop distance, etc. If necessary, e.g. if a link fails, the multipath could be expanded in order to include additional alternative paths. This case is shown in Fig. 1(c), where two alternative paths were included.

In this work, we consider only two intermediate nodes so that the path is divided in three segments: the first ranges from the source (S) to the first intermediate node (*In1*), the second between the two intermediate nodes, and the third from the second intermediate node (*In2*) to the destination (D). This segmented path is called a *multistep path* (MSP), and uses minimal static routing in each segment. When using *multistep paths* deadlock freedom becomes a key issue. In our proposal, deadlock freedom is ensured by having a separate virtual channel for each step. As we are considering two intermediate nodes, one extra virtual channel is used (if required) from S to *In1*, another from *In1* to *In2*, and a third one from *In2* to D. This way, each step defines a virtual network, and the packets change virtual network at each intermediate node. Although each virtual network relies on a different virtual channel, they all share the same adaptive channel(s). Therefore, a total of 4 virtual channels are need.

3.1 Method Behavior

The behavior of the method, including all its functionalities, could be seen in Fig. 2. The behavior diagram shown in Fig. 2 consists of four main blocks: *Source endnode*; *Message routing*; *ACK routing*; and *Destination endnode*. The source and destination endnode blocks contain the actions implemented at the source and destination nodes, respectively; while message and ACK routing blocks rep-

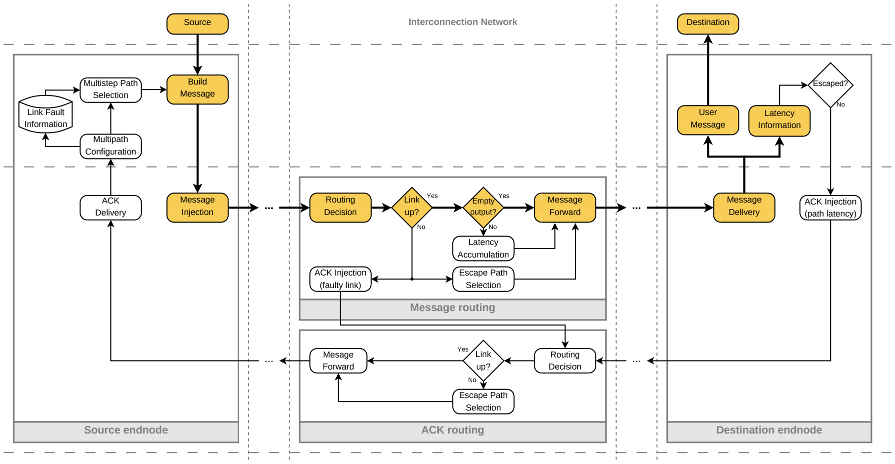


Fig. 2. Method behavior diagram.

resent actions carried out by the routers along the source-destination paths. Each block is composed by several elements (stage boxes and decision elements), where the colored elements represent the set of actions performed in the absence of failures and congestion, and the colorless correspond to the additional features for fault tolerance and congestion control.

When a source node injects a message in the interconnection network, it traverses a set of routers before reaching the destination node. Two monitoring actions are conducted at each router along the source-destination path: link state and traffic load monitoring. Link state monitoring is performed directly over router physical channels, while traffic load monitoring is accomplished by the router over the message. These actions are represented in Fig. 2 by the two colored decision elements in the *Message routing* block.

If there are no faults in the source-destination path, each message registers and transports the accumulated latency information about the path it traverses (either the original or an alternative), by means of the *Latency Accumulation* element in *Message routing* block. When the message reaches the destination node, the accumulated latency value is obtained from the packet and, if the path is fault-free, sent back to the source node by means of an ACK message (*ACK Injection (path latency)* element in the *Destination endnode* block) in order to notify the source node about the network traffic burden.

On the other hand, two actions are triggered if a message tries to use a faulty link while it traverses the source-destination path. Firstly, the message is rerouted to its destination through an escape path (*Escape Path Selection* element in the *Message routing* block). At the same time, the *ACK Injection (faulty link)* element sends back a special ACK message to the source node. This ACK message –sent by means of the *ACK routing* block– carries information about the fault location (e.g. node and port identifiers) to avoid the use of the faulty path. Those triggered actions were previously illustrated in Fig. 1(b).

Those two kinds of ACK messages have higher priority in the routing unit, and their size is less than 1% of the data messages because they only transport control info: a latency value or failure information. Notice that only one of those ACK messages is sent for each data message, as appropriate.

Using the link fault information together with the set of collected latencies, the number of alternative paths needed for a specific source-destination pair is determined. From this action, performed at the *Multipath Configuration* element in the *Source endnode* block, the method avoids the use of faulty paths and fairly distributes the communication load over the multipath. The communication load distribution is accomplished by selecting the appropriate MSPs at the *Multistep Path Selection* element.

The outcome of this phase is then used by the source node to distribute the load among all the MSPs in base of their latency. The path with lower latency is most frequently used, then messages are distributed over the MSPs according to their relative latency values.

The set of actions at node level of our proposal have not a high overhead because they are simple (comparisons and accumulations), locally performed, and do not delay send/receive primitives. As shown in Fig. 2, message is forwarded without any overhead when the output link is non-faulty. The escape path mechanism is invoked only when faults are detected, and latency accumulations are performed when messages are waiting in the queue. Hence, computing these operations is performed concurrently with packet delivery. Furthermore, interconnection networks usually are not planned to continuously operate at their saturation point, thus small overheads could be tolerated if necessary to avoid faults.

Our method relies on physical level information about links state. This information is already available on almost all modern network devices. Current devices test and control their ports and links by means of physical parameters such as potential difference, impedance, etc. For example, the InfiniBand architecture offers four link states: LinkDown, LinkInitialize, LinkArm and LinkActive [16]. Even the simplest Ethernet router makes available the link state information.

3.2 Non-permanent Faults

Our method considers faults as non-permanent at a first stage, to prevent the misuse of resources. If faults persist over time, their status is changed to permanent.

In order to achieve this functionality, information about the location and status of faults is stored in the *Link Fault Information* element (*Source endnode* block). From this information, source nodes may be able to use fault-free paths which otherwise would incorrectly appear as faulty (due to non-permanent faults).

The information about faults is obtained from the ACK messages sent back to the source nodes from routers that have detected the faults. The node and port identifiers obtained from these ACK message are used for storing and indexing the fault information in the *Link Fault Information* element. Each entry is composed by these two identifiers and two additional numbers used to manage the fault status: the *stage number* and the *attempt number*.

The information in the *Link Fault Information* element is updated each time a fault ACK message arrives to the source node. If the ACK carries information about a new fault, a new entry is included. In turn, if there is already an entry for the fault, the *stage number* is increased. A fault is only considered as permanent if its *stage number* is greater than or equal to three. Notice that this means a fault is considered to be permanent only after receiving at least three different notifications about the fault.

On the other hand, the *attempt number* has been included to treat differences in the duration of non-permanent faults. In fact, it is intended to be used as a timer to delay the use of a path which has been notified as faulty. This seeks to reduce the possibility of considering a non-permanent fault as a permanent one.

In our method, as stated above, the *Multistep Path Selection* element must select the MSPs before the injection of each new message. There is where the information stored in the *Link Fault Information* element is used.

When selecting the MSP, the *Multistep Path Selection* element looks for entries corresponding to the links along the path between the source-destination pair. Notice that a link can be identified by means of the identifiers of the router and port to which it is connected.

The path is fault-free and can be selected if there are no entries for all the links along the path. On the other hand, if there are faults along the path, it may or may not be used depending on faults status. If at least one of the entries corresponds to a permanent fault, the path cannot be used and an alternative path should be selected. If none of the entries correspond to permanent faults, the *attempt number* must be considered for the selection. The path can be selected only if *attempt number* of all the links along the path are lower ten. Otherwise, the *attempt number* is increased for all the links along the path, but the path cannot be used. By means of this action, the *attempt number* eventually will reach ten and the path would be selected.

If a path containing a non-permanent fault is selected and used, two situations may arise. In the best case, the fault will have disappeared and a latency value will be received from the destination node. In this case, all the entries of this path will be removed from the *Link Fault Information* element, and the resources utilization would be improved. In the worst case, the message will be rerouted to its destination and a new fault notification will be received.

4 Performance Evaluation

This section describes the test scenarios used to evaluate our proposal and provides the explanation of experimental results.

The simulation environment is provided by the commercial modeling and simulation tool OPNET Modeler [17]. This tool gives support for modeling communication networks, and allows the injection of faults in model components. The whole actions and functionalities of our proposal have been modeled using this tool.

Experimentation is based on two-dimensional torus chosen mainly due to its current popularity and multiple alternative paths between nodes. The network was modeled based on interconnection elements, connected among them through links; and endnodes that provide the interface to connect processing nodes to the network.

The simulations were conducted for a 1024 nodes network arranged in a 32x32 torus topology. We have assumed Virtual Cut-Through flow control and several standard package sizes with a constant packet injection rate. Link bandwith was set to 1 Gbps, and the size of routers buffers to 2 MB.

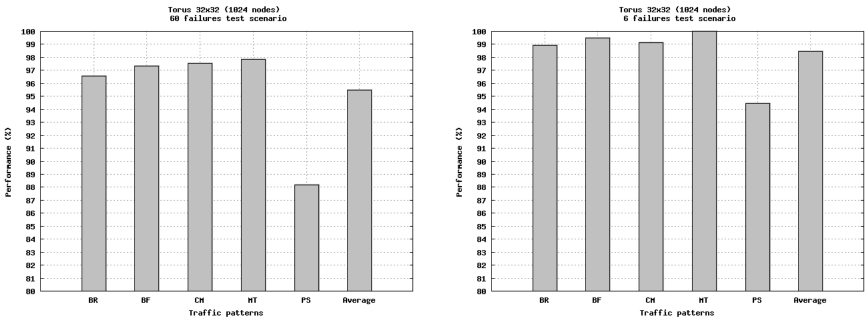
In order to evaluate the behavior and measure performance, experiments were conducted using standard communication patterns with up to 60 simultaneous link failures randomly injected. Permanents faults were used in order to validate the method in the worst-case scenarios. Standard communication patterns were used due to their application in computational intensive scientific applications [18]. These patterns are: *Bit Reversal (BR)*, *Perfect Shuffle (PS)*, *Butterfly (BF)*, *Matrix Transpose (MT)*, and *Complement (CM)*.

In order to evaluate our proposal, as a first step, a set of fault-free scenarios were simulated several times to get average latency values in the absence of failures. Later, faults were injected in the scenarios used in the first step to measure the average latency values of each approach. Up to 60 network links were simultaneously failed during the evaluation of test scenarios. Then, performance degradation was measured as the difference between latency values of faulty and fault-free scenarios.

The vast majority of previous work in literature use static models, therefore, it is not possible to make direct performance comparisons against them. One of the only proposals dealing with dynamic faults ([12]) achieves a throughput performance average of 85.5% using a *Uniform* traffic pattern and 88.8% using *Permutation* traffic, both in the presence of 7 random link failures and allowing packet drops.

The performance results for the standard traffic patterns in the 60-failures test scenario are shown in Fig. 3(a). In order to compare the ratio between the number of failures and the performance of our proposal, the results of the 6-failures test scenario are also included and shown in Fig. 3(b).

As shown in Figs. 3(a) and 3(b), our method obtains very high performance values. An important point to emphasize is the fact that with a linear increase of 10 times the number of faults, the average performance degradation value is just about 3%. In the worst case the performance is about 88% for the *Perfect shuffle* pattern with 60 simultaneous faults, and 100% in the best case for *Matrix transpose* pattern with 6 simultaneous faults. Performance values are even better if we consider the average values, obtaining a 97% performance value in the worst case (60 faults).



(a) 60 failures

(b) 6 failures

Fig. 3. Results of test scenarios.

5 Conclusions

In this paper, we have proposed multipath fault-tolerant routing method designed to deal with the fault tolerance problem for high-speed interconnection networks. This method is able to support a dynamic fault model, while at the same time not requiring network reconfigurations or stopping packet injection at any time, using a limited number of virtual channels. Our proposal needs few additional hardware resources and is able to treat the intermittent and transient faults as well as the permanent ones, maximizing the resources utilization. Unlike other fault-tolerant approaches, our method does not degrade at all the system performance in the absence of faults.

Evaluation results show an average performance value higher than 97% for several test scenarios ranging from 1 up to 60 number of faults, using the standard communication patterns. From these results we conclude that our method is capable of reroute messages to their destinations through fault-free paths with a negligible performance degradation even in the presence of a high number of faults.

We are currently working on improving the method to tolerate an unbounded number of failures. In addition, future work includes the enlargement of the current fault models to address information losses and stuck caused by the failures of network devices.

References

1. Barroso, L., Dean, J., Holzle, U.: Web search for a planet: The google cluster architecture. Micro, IEEE **23**(2) (March-April 2003) 22–28
2. Adiga, N., Almasi, G., Almasi, G., Aridor, Y., Barik, R., et al.: An overview of the BlueGene/L supercomputer. In: Supercomputing ACM/IEEE 2002 Conference. (Nov. 2002) 60–82
3. Abd-El-Barr, M.: Design and analysis of reliable and fault-tolerant computer systems. Imperial College Press, London, UK. (2007)

4. Zarza, G., Lugones, D., Franco, D., Luque, E.: A multipath fault-tolerant routing method for high-speed interconnection networks. In: 15th International European Conference on Parallel and Distributed Computing (EuroPar). (2009) 1078–1088
5. Sem-Jacobsen, F., Skeie, T., Lysne, O., et al.: Siamese-twin: A dynamically fault-tolerant fat-tree. In: International Parallel and Distributed Processing Symposium (IPDPS 2005), IEEE Computer Society (April 2005) 100b–100b
6. Mejia, A., Flich, J., Duato, J., Reinemo, S.A., Skeie, T.: Segment-based routing: an efficient fault-tolerant routing algorithm for meshes and tori. In: International Parallel and Distributed Processing Symposium (IPDPS 2006), IEEE Computer Society (April 2006) 10 pp.–
7. Puente, V., Gregorio, J.A.: Immucube: Scalable fault-tolerant routing for k-ary n-cube networks. IEEE Transactions on Parallel and Distributed Systems **18**(6) (2007) 776–788
8. Duato, J.: A theory of fault-tolerant routing in wormhole networks. IEEE Transactions on Parallel and Distributed Systems **8**(8) (1997) 790–802
9. Gómez, C., Gómez, M.E., López, P., Duato, J.: An efficient fault-tolerant routing methodology for fat-tree interconnection networks. In: ISPA. Volume 4742 of Lecture Notes in Computer Science., Springer (2007) 509–522
10. Gómez, M.E., Nordbotten, N.A., Flich, J., Lopez, P., Robles, A., Duato, J., Skeie, T., Lysne, O.: A routing methodology for achieving fault tolerance in direct networks. IEEE Transactions on Computers **55**(4) (2006) 400–415
11. Valiant, L.G., Brebner, G.J.: Universal schemes for parallel communication. In: STOC '81: Proceedings of the thirteenth annual ACM symposium on Theory of computing, New York, NY, USA, ACM (1981) 263–277
12. Nordbotten, N.A., Skeie, T.: A routing methodology for dynamic fault tolerance in meshes and tori. In: International Conference on High Performance Computing (HiPC). LNCS 4873, Springer-Verlag (2007) 514–527
13. Skeie, T.: Handling multiple faults in wormhole mesh networks. In: Euro-Par '98: Proceedings of the 4th International Euro-Par Conference on Parallel Processing, London, UK, Springer-Verlag (1998) 1076–1088
14. Ho, C.T., Stockmeyer, L.: A new approach to fault-tolerant wormhole routing for mesh-connected parallel computers. IEEE Transactions on Computers **53**(4) (April 2004) 427–438
15. Franco, D., Garcés, I., Luque, E.: Distributed routing balancing for interconnection network communication. In: HIPC '98. 5th International Conference On High Performance Computing. (1998) 253–261
16. InfiniBand Trade Association: InfiniBand architecture specification: release 1.2. Volume 1. InfiniBand Trade Association, Portland, OR (2004)
17. OPNET Technologies: Opnet modeler accelerating network R&D (July 2009)
18. Duato, J., Yalamanchili, S., Ni, L.M.: 9. In: Interconnection networks. An Engineering Approach. Morgan Kaufmann (2003) 475–558