Segment-Based Routing: A New Approach to Efficient Topology-Agnostic Routing *

A. Mejía, J. Flich and J. Duato
Departamento de Informática de Sistemas y Computadores
Universidad Politécnica de Valencia
E-mail: {andres,jflich,jduato}@gap.upv.es

September 23, 2005

Abstract

During the last years, Clusters of PCs are being used as a cost-effective alternative for building high-performance computing and server systems. In these systems, high-speed interconnects like Myrinet, Quadrics and InfiniBand^{TM} are currently used as they offer reliable performance at affordable cost. In these networks routing is deterministic and virtual channels are not used for routing as they simply do not exist (Myrinet), or they are reserved for other purposes like QoS (InfiniBand).

In this paper we propose a novel approach in order to get a topology-agnostic routing able to adapt to the topology. The algorithm, referred to as Segment-Based Routing (SR), partitions the topology in such a way that allows to place a bidirectional routing restriction on each partition, being its position independent of the relative positions of the remaining restrictions. This allows much greater flexibility than offered by previous proposals. We intend to use this flexibility as a way to achieve an effective (simple and efficient) routing algorithm that obtains good performance regardless of the topology.

Evaluation results show that SR is able to adapt to the topology, and therefore, it achieves better results than up*/down* and FX (in some cases, FX throughput is increased by a factor of 1.8). This promising result suggests that much greater benefits can be obtained by the proposed method.

1 Introduction

Nowadays, Clusters of PCs are becoming a cost-effective solution for high performance computing (HPC) and server systems. The use of high-performance networks and PCs allows to build large and comprehensive clusters at affordable cost. Examples of large cluster systems implemented are MareNostrum [5] (ranked 4th in top500 list [7]), and Thunder [6] (ranked 5th). Indeed, 61 clusters are included within the top 100 of the list. In these systems, the interconnection network plays a key role. For this, high-speed interconnects like Myrinet [2], InfiniBand^{TM} [4], and Quadrics [9] are preferred as they offer low latencies and high bandwidth.

Routing on these networks is deterministic. That is, a packet, once it is injected, follows a unique path until it reaches its destination. One of the main benefits of using deterministic routing is that in-order arrival of packets is preserved. However, deterministic routing usually makes an inefficient use of network resources.

The use of virtual channels [4] in these networks is also common but they are usually devoted to other purposes like quality of service. As an example, InfiniBand presents up to 16 virtual channels used in combination with 16 service levels (SLs). Moreover, Myrinet does not implement any virtual channel. Therefore, in these networks, it is necessary to provide a deterministic routing that does not rely on virtual channels. Most of the fault-tolerant routing strategies proposed in the literature for massively parallel computers are not suitable for clusters (see chapter 6 of [8] for a description of some of the most interesting approaches). A simple way to provide fault tolerance is the use of topology-agnostic routings (possibly combined with a reconfiguration process). These routings can be applied to any topology and, thus, they tolerate any combination of faults (that does not physically disconnect the network). From the set of topology-agnostic routings proposed in the literature (up*/down* [16], lturn [3], smart-routing [12], LASH [14], TOR [17], LASH-TOR [13], DL [10], multiple virtual networks [11], and FX [15]) only up*/down*, lturn, smart-routing and FX can be applied to a network with deterministic routing and no virtual channels.

In order to achieve a deadlock-free routing an acyclic channel dependency graph must be provided. To do so, the routing algorithms impose some routing restrictions. A routing restriction is defined by a switch and a pair of input and output ports of the switch. When placing a routing restriction in the network no packet can use the input port following the output port of the switch. As bidirectional links are used in the network, cycles must be broken in each direction. For this, up*/down* places routing restrictions in the same location for each direction. In this sense, FX is a novel routing scheme as it breaks cycles in each direction at different positions in the cycle. In [15] it is shown that this fact allows to achieve higher performance.

*This work was supported by the Spanish CICYT under Grant TIC2005-08154-C06.

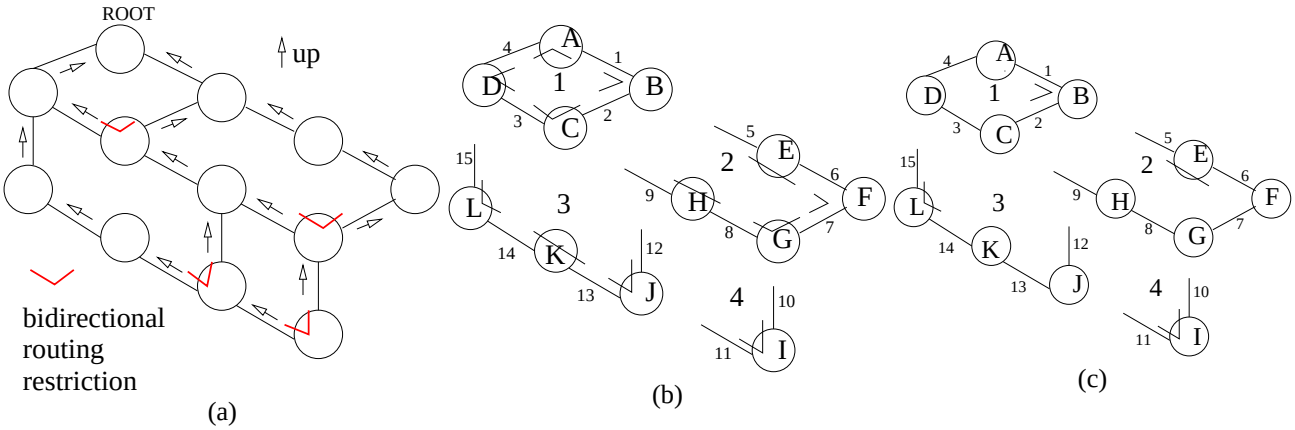


Figure 1: Possible set of restrictions. (a) Established by up*/down*, (b) different alternatives, and (c) a possible combination.

Placing routing restrictions in the network must be done carefully as deadlock-freedom must be ensured while keeping network connectivity. To do so, routing algorithms like up*/down*, lturn, and FX establish some rules beforehand[1]. These rules will indicate where cycles will be broken. As an example, Figure 1.a shows a network where the up*/down* routing is applied. As a first step, the up*/down* selects one node as the root. From this root, a BFS spanning tree is computed and the links are labeled as *up* or *down* accordingly. Then, routing restrictions are placed at the pair of consecutive links that perform a *down-up* transition. Thus, once the root is selected the routing restrictions (four bidirectional routing restrictions in the example) are fixed (assuming the BFS up*/down*). As there are 12 candidates to become root, only 12 combinations of routing restrictions are allowed. Additionally, for each combination, the relative position of each routing restriction is fixed.

The FX routing also suffers from this fact. Indeed, it builds a depth-first spanning tree (DFS) for labeling the nodes, then it uses such labelings in order to place the unidirectional routing restrictions. Thus, once the DFS tree is computed, the exact position of the routing restrictions is fixed. Notice that the rules used by up*/down* and FX (the same for lturn) do not take into acount the topology.

In thi spaper we take a different and new approach, we present the Segment-based Routing Algorithm (SR). As mentioned before, any deterministic routing algorithm can be viewed as a set of routing restrictions in the network. Routing restrictions must be carefully placed in the network in order to guarantee at the same time deadlock freedom and full connectivity among all the endnodes of the system.

It will be based on the partitioning of the network in such a way that on every partition a bidirectional routing restriction will be placed, but, contrary to the previous routing algorithms, the exact position of the routing restriction on a partition will be completely independent of the rest of partitions. This will give some degree of freedom in order to achieve a good set of paths. For this, the SR algorithm will, on a first stage, compute routing segments. A routing segment is defined as a list of interconnected switches and links. Figure 1.b shows an example. In this topology we can define several routing segments. For instance, the network is made up of four routing segments. In particular, switches A, B, C, and D, and links connecting those switches (1, 2, 3, 4) define the first routing segment. The second routing segment is formed by switches E, F, G, H and links 5, 6, 7, 8, and 9. The third segment is formed by 12, J, 13, K, 14, L, and 15. Finally, the last segment is formed by 10, I, and 11. We can observe that all the network components (switches and links) are included in only one routing segment (disjoint segments).

Taking into account the routing segments defined in Figure 1.b we can add a bidirectional routing restriction to every routing segment and, as a result, we will obtain a deadlock-free routing algorithm while still guaranteeing full connectivity among switches. Figure 1.a shows an example. Obviously the way segments are computed is critical in order to guarantee deadlock-freedom and full connectivity. Also, there will be some special cases that require a different treatment.

The rest of the paper is organized as follows. Section 2 describes the routing method and demonstrates that it achieves a valid solution for any topology. Then, in Section 3, SR will be compared in performance against FX and up*/down*. Finally, in Section 4 we will draw some conclusions and future work.

2 Segment-based Routing

Figure 2 shows the algorithm for computing the routing segments[2]. Throughout the computation, the algorithm marks switches and links with the following status:

[1]Smart-routing takes a different approach to compute the routing restrictions. In fact, smart-routing starts an iterative process of computing paths and checking deadlock avoidance until it gets the best set of paths. Thus, it exhibits a high computational cost.

[2]The algorithm assumes that a packet will never enter and leave a switch through the same link.

```
procedure compute_segments()
var
  s : segment list
  sw : switch
  c : integer # current subnet
  n : integer # current segment
  end : boolean
begin
  c = 0; n = 0
  sw = random
  sw.starting = true
  sw.subnet = c
  sw.visited = true
  s[n] = empty
  end = false
  repeat
    if (find(sw,s[n],c))
      n++
    else
      sw.terminal = true
    sw = next_visited()
    if (sw == nil)
    begin
      sw = next_not_visited()
      c++
      sw.starting = true
      sw.subnet = c
      sw.visited = true
    end
    if (sw == nil) end = true
  until (end)
end procedure
```

```
procedure find(sw, segm, snet) : bool
var
   nsw : switch
begin
  sw.tvisited = true
  segm = segm + sw
  links = suitable_links(sw)
  if (links==nil) begin
    sw.tvisited = false
    segm = segm - sw
    return false
  end
  for each link ln in links begin
    ln.tvisited = true
    segm = segm + ln
    nsw = aTop[sw,ln]
    if ((nsw.visited and nsw.subnet = snet) or
                  find(nsw, segm, snet)) begin
      ln.visited = true
      sw.visited = true
      ln.tvisited = false
      sw.tvisited = false
      return true
    end
    else begin
      ln.tvisited = false
      segm = segm - ln
    end
  segm = segm - sw
  sw.tvisited = false
  return false
end procedure
```

Figure 2: Main procedure for searching segments.

- *visited* and *tvisited*. A switch or link becomes *visited* once it belongs to an already computed routing segment. During the process of computing a routing segment, a switch or link may change to the state *temporarily visited*. Only switches and links not marked as *visited* may be marked as *tvisited*.

- *starting* and *terminal*. A switch is marked as *starting* if it is the first one chosen to compute network segments within a subnet. A switch is marked as *terminal* if through at least one of its links no new segment is found.

Additionally, the algorithm will group switches and links in the network within subnets. A subnet will be formed by a group of switches and links that will be connected to the rest of the network (other subnets) through one link. All the components of a routing segment will belong to the same subnet. The use of subnets is motivated by some special cases that will be described later.

The *compute_segments* procedure (Figure 2) searches for all the segments. For this, a random switch is chosen in order to be the starting switch of the first segment within the first subnet[3]. The selected switch (*sw*) is marked as *starting* and *visited*, and added to the first subnet. Then, the procedure searches all the possible segments and subnets. For this, it calls the *find* procedure that will try to find a new segment starting from the first switch (*sw*). On success, the *find* procedure updates all the switches and links belonging to the new segment accordingly (all of them are marked as *visited* and belonging to the current subnet). On fail, the *find* procedure keeps the switches and links unmodified. If the procedure fails, then from the switch there is no new segment and therefore, the switch is marked as *terminal*.

Next, the procedure searches (*next_visited* function) a switch that is marked as *visited*, belongs to the current subnet, and has at least one link not marked as *visited*. From this switch a new segment is searched. However, in case there is no such switch, the procedure searches (*next_not_visited* function) a switch that is not marked as *visited* nor as *terminal* and is attached to a terminal switch. On success, a new subnet is initiated, the switch is marked as *starting* and *visited*, is included in the new subnet and a new segment is searched from the switch. In the case there is no such switch, then all the switches have

[3]As any switch could be selected as starting switch, SR could drive to different solutions. In [18], a better criteria for computing segments for 2D meshes is described

been tested and the procedure ends. Notice that this procedure ensures that all the switches and links in the network will be considered to belong to a segment.

The procedure *find* is responsible to find, from a given visited switch, a segment ending on a visited switch and made of switches and links not visited. For this, the current switch is marked as *tvisited* and is added to the current segment. Then, a set of links attached to the switch is build (function *suitable_links*). In particular, links (attached to the switch) not marked as *visited* nor as *tvisited* are included in the set. In the case that the set is empty (there are no suitable links) then, the procedure has failed in finding a new segment.

However, in the case the set is not empty, links are considered in a established order. Notice that the order followed may impact on the future performance of the algorithm as different sets of segments will be discovered. Thus, here is where we can consider some order based on the topology and some objectives like maximizing bandwidth and minimizing latency. In [18], we propose some rules to compute effective segments in meshes. For every link in the set the procedure marks the link as *tvisited*, adds the link to the segment and inspects if the switch at the other side is marked as *visited* and belongs to the current subnet. If not, the procedure is called recursively from the switch attached to the other side in order to find the remaining of the segment. If the called procedure succeeds (returns true) or the switch at the other side is marked as *visited*, then the segment has been found. In case of not finding a new segment through the link, the link is unmarked, is removed from the segment and a new link from the set is tested. Finally, in the case no segment is found through any of the links, then the procedure has failed finding a segment from the switch, the switch is removed from the segment and the procedure returns.

Once the routing segments are computed, the algorithm will place routing restrictions on every routing segment accordingly. The previous algorithm will find three types of routing segments:

- Starting segment. This routing segment will start and end on the same switch, thus forming a cycle. This routing segment will be found every time a new subnet is initiated.
- Regular segment. This type of segment will start on a link, will contain at least one switch, and will end on a link.
- Unitary segment. This type of routing segment will be made only by a link.

In order to ensure deadlock freedom while guaranteeing connectivity, the routing algorithm must place in each type of routing segment routing restrictions. In particular, for a starting segment, a bidirectional routing restriction can be placed on any switch except the starting one (the reason for not placing a routing restriction in the starting switch is described in [18]). Notice that by doing this, the cycle is broken. For regular segments, one bidirectional routing restriction can be placed on any of the switches belonging to the segment. This is done in order to break any possible cycle using the segment. Finally, for unitary segments, all the traffic crossing the link must be avoided in order to avoid deadlocks. Thus, in one side of the links bidirectional routing restrictions must be placed for every link attached to the switch.

2.1 Computational Cost

SR can be viewed as an algorithm performing three phases. In the first one segments are computed. A random search through a recursive function may exhibit an excessive computational cost (as the recursive function searching for segments is not guided). Therefore, in order to get an efficient segmentation it is required to guide the procedure. A simple and efficient rule to apply is keeping segments as shortest as possible. But you can also have more elaborated criterions based on traffic (perfect traffic balancing, hotspot optimization) or topology considerations (prioritizing weight of links in case any weighted networks). Taking into account the shortest segment search proposed in [18], the cost will be lower as links are visited only once, thus, being the cost for this phase O(m) where m is the number of links in the network (in the case of a mesh, the cost will be $O(2n)$ being n the number of switches).

At the second phase routing restrictions are placed. In a straight forward method, as proposed in [18], the computational cost of this phase will be $O(s)$ where s is the number of segments.

Finally, at the third phase, paths are computed taking into account the routing restrictions. The complexity of this phase may greatly vary depending on the desired final performance. Notice that the SR routing algorithm may provide several paths for some source-destination pair of nodes. A straight forward search like random path selection will exhibit a computational cost of $O(n^2)$ where n is the number of switches. However, more sophisticated methods have been proposed in order to get the best set of paths. For instance, FX uses an algorithm that searches the best set of paths that minimizes the maximum number of paths that cross any link (minimizing the crossing path metric). It has to be noted that the final selection of paths does not depend on the routing algorithm. For instance, up*/down* does not state any specific rule to select the final set of paths. However, the same algorithm used in FX could be applied among all the available paths from up*/down*.

Therefore, the computational cost of SR will be driven by the third stage. If the method is based on random path selection (among all the possible paths), then its cost will be $O(n^2)$ where n is the number of switches. On the other hand, if an optimization function is used like the one used in FX, then its cost will be the same of FX.

				SR			UD			FX		
				Distance	Weight		Distance	Weight		Distance	Weight	
Topol	Seed	# Segm	# Restr	Avg	Avg	STD	Avg	Avg	STD	Avg	Avg	STD
Irreg.	1	45	45	5.39	98.53	45.37	5.67	109.55	63.16	5.54	106.96	60.32
Irreg.	2	43	43	5.34	103.34	49.95	5.39	104.19	66.96	5.55	107.29	71.29
Irreg.	3	43	43	5.33	103.26	43.19	5.46	105.66	63.19	5.43	104.92	59.06
Irreg.	4	43	43	5.38	104.06	40.99	5.45	105.40	63.50	5.43	104.93	48.79
Irreg.	5	43	43	5.39	104.23	40.84	5.62	108.58	60.39	5.63	108.72	50.55

Table 1: Analytical results for different network topologies: five different 8×8 meshes with 5% of link failures (Irreg-x).

3 Performance Evaluation

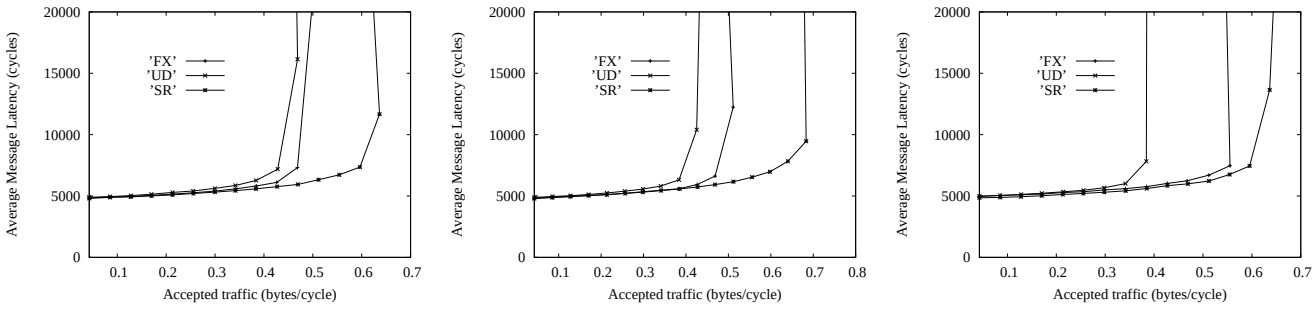
In this section we will evaluate the performance of the SR algorithm. For this, we will obtain analytical results of the paths computed by SR compared with the ones achieved by up*/down* (UD) and the efficient FX. In the case of SR we will compute the segments and apply the routing restrictions the same way described in [18]. In the third stage of the algorithm, computation of the final paths for every source-destination pair, we will use the path balancing algorithm described in [1], similar to the one used by FX.

3.1 Simulation Results

We have modeled different regular topologies with 5% of randomly-injected link failures. Due to lack of space only results for some topologies will be shown[4]. Table 1 displays analytical results for SR, FX and UD in different network topologies. In particular, it shows the average routing distance, the average number of paths crossing each link (denoted as weight), and its standard deviation (STD). It also shows for SR the number of segments computed, and the number of bidirectional routing restrictions applied.

As can be seen, SR achieves better paths than FX and UD in terms of average path length and link weight distribution. This is due to the fact that SR is able to keep the regularity of the network (in those places where there are not failures) when computing the routing segments, therefore obtaining more suitable paths. As we will see, this fact will drive network performance. Also notice that for all the irregular networks, SR obtains always routing segments that are not unitary as the number of restrictions is equal to the number of routing segments.

It has to be noted that FX looses considerable ground when applied to irregular topologies. SR achieves more than 20% of reduction in the standard deviation of link weight and also in the average distance between nodes as it allows more flexibility.



(a) 8×8 mesh seed 2 (b) 8×8 mesh seed 3 (c) 8×8 mesh seed 4

Figure 3: Average packet latency vs throughput. Meshes with 5 % of link failures. Uniform distribution of packet destinations.

Now let us drive your attention to the performance results. [5] Figure 3 shows results for some mesh networks with 5% of randomly-injected link failures and uniform traffic. For all the cases, SR beats FX and in all the cases UD is outperformed by both of them. In particular, for some 8×8 meshes SR increases FX and UD throughput by a factor of 1.4 and 1.65, respectively. Therefore these results suggest that SR is on the right track to obtain the maximum from the network regardless of its topology.

[4] Similar results have been obtained for the rest of topologies.

[5] simulation environment can be found in [18]

4 Conclusions

In this paper we have proposed a novel approach to achieve a topology-agnostic routing algorithm. The novelty resides in the fact that it offers greater flexibility when placing the routing restrictions in order to guarantee deadlock-freedom while keeping network connectivity. Moreover, SR uses the regularity of the topology when computing the paths even in the presence of failures. This approach offers greater flexibility when computing the possible best set of paths for a given topology. Preliminary results (applying bidirectional routing restrictions and using a preliminary way of computing segments in meshes) shows encouraging results. SR outperforms UD. Also, in some scenarios SR increases FX throughput up to a factor of 1.8.

As future work we plan to increase the performance of the algorithm. An issue that we would like to explore is the development of better methods for computing segments and the use of unidirectional routing restrictions as FX does.

We wish to thank Dr. Tor Skeie for his constructive comments and suggestions that has helped improving the paper.

References

[1] J. Flich, P. Lopez, M.P. Malumbres, J. Duato, and T. Rokicki, Combining In-Transit Buffers with Optimized Routing Schemes to Boost the Performance of Networks with Source Routing. in *Proc. of Int. Symp. on High Performance Computing*, Oct. 2000.

[2] N.J.Boden and et al. Myrinet: A Gigabit per Second Local Area Network. *IEEE Micro, 15(1):29 35*, 1995.

[3] M.Koibuchi, A.Funahashi, A.Jouraku, and H.Amano. Lturn routing: An adaptive routing in irregular networks. in *Proc. of ICPP, pages 374 383*, Sept. 2001.

[4] InfiniBand^{TM} Trade Association, InfiniBand^{TM} architecture. Specification Volumen 1. Release 1.0.a. *Available at http://www.infinibandta.com.* june 2001

[5] MareNostrum supercomputer, Barcelona Supercomputing Center. *Available at http://www.bsc.org.es/.*

[6] Thunder supercomputer, Lawrence Livermore National Laboratory. *Available at http://www.llnl.gov/.*

[7] Top500 Supercomputer Sites, *Available at www.top500.org/.*

[8] J. Duato, S. Yalamanchili, and L. Ni. Interconnection Networks: An Engineering Approach. 1997.

[9] F. Petrini, W. Feng, and A. Hoisie. The Quadrics network (QsNet): high performance clustering technology. In *Proc. of Hot Interconnects, pages 125 130*, Aug. 2001.

[10] M.Koibuchi, A.Jouraku, K. Watanabe, and H.Amano Descending Layers Routing: A Deadlock-Free Deterministic Routing using Virtual Channels in System Area Networks with Irregular Topologies. em International Conference on Parallel Processing, 2003.

[11] J. Flich, P. Lpez, J.C. Sancho, A. Robles, J. Duato Improving InfiniBand Routing through Multiple Virtual Networks *Procceding of 2002 International Symposium on High Performance Computing*, 2002.

[12] L. Cherkasova, V. Kotov, and T. Rokicki, Fibre channel fabrics: Evaluation and design, in *Proc. of 29th Int. Conf. on System Sciences*, Feb. 1995.

[13] T. Skeie, O. Lysne, J. Flich, P. Lopez, A. Robles, J. Duato, LASH-TOR: A Generic Transition-Oriented Routing Algorithm, in *Proc. of IEEE International Conference on Parallel and Distributed Systems*, 2004.

[14] T. Skeie, O. Lysne and I. Theiss, Layered Shortest Path (LASH) Routing in Irregular System Area Networks, in *Proc. of Communication Architecture for Clusters, IEEE Computer society* 2002.

[15] J.C. Sancho, A. Robles, and J. Duato, A Flexible routing Scheme for Networks of Workstations, in *Proc. of 2000 International Conference on High Performance Computing (ISHPC'01)*, Sept. 2000.

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

[17] J.C. Sancho, A.Robles, J. Flich, P.Lopez, and J. Duato, Effective methodology for deadlock-free minimal routing in InfiniBand networks *Proc. of the 2002 Int Conf. On Parallel Processing, IEEE Computer Society*, 2002.

[18] A.Mejia, J. Flich and J. Duato, A Novel Methodology for Efficient Routing in Irregular Networks *DISCA, Universidad Politcnica de Valencia, 2005*, Research Report, DISCA/0066-2005..