Segment-Based Routing: An Efficient Fault-Tolerant Routing Algorithm for Meshes and Tori *

A. Mejia[1], J. Flich[1], J. Duato[1], Sven-Arne Reinemo[2] and Tor Skeie[2]

[1]Dpto de Informática de Sistemas y Computadores
Universidad Politécnica de Valencia
P.O.B. 22012, 46022 - Valencia, Spain
{andres,jflich,jduato}@gap.upv.es

[2]Simula Research Laboratory
P.O.Box 134, N-1325
Lysaker, Norway
{svenar,tskeie}@simula.no

Abstract

Computers get faster every year, but the demand for computing resources seems to grow at an even faster rate. Depending on the problem domain, this demand for more power can be satisfied by either, massively parallel computers, or clusters of computers. Common for both approaches is the dependence on high performance interconnect networks such as Myrinet, Infiniband, or 10 Gigabit Ethernet. While high throughput and low latency are key features of interconnection networks, the issue of fault-tolerance is now becoming increasingly important. As the number of network components grows so does the probability for failure, thus it becomes important to also consider the fault-tolerance mechanism of interconnection networks. The main challenge then lies in combining performance and fault-tolerance, while still keeping cost and complexity low.

This paper proposes a new deterministic routing methodology for tori and meshes, which achieves high performance without the use of virtual channels. Furthermore, it is topology agnostic in nature, meaning it can handle any topology derived from any combination of faults when combined with static reconfiguration. The algorithm, referred to as Segment-based Routing (SR), works by partitioning a topology into subnets, and subnets into segments. This allows us to place bidirectional turn restrictions locally *within a segment. As segments are independent, we gain the freedom to place turn restrictions within a segment independently from other segments. This results in a larger degree of freedom when placing turn restrictions compared to other routing strategies.*

In this paper a way to compute segment-based routing tables is presented and applied to meshes and tori. Evaluation results show that SR increases performance by a factor of 1.8 over FX and up/down* routing.*

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

1. Introduction

The demand for more computing resources steadily increases, and this demand is currently met by building larger massively parallel computers or by building computer clusters. The latter approach is becoming more common, since it can be built from commodity equipment, while the former requires a varying degree of custom made equipment. Though similar, the two approaches fulfil slightly different goals. Massively parallel computers are optimised for special cases (typically certain scientific computations), while clusters are optimised for a general set of applications. Of the worlds 100 fastest computers 60 are massively parallel computers, while 40 are clusters [4] At 5th place we find the first cluster, the Thunderbird [3]. Even if they are all different, their need for a high performance interconnection network is very similar. Again, the interconnection technology can be either commodity or custom made equipment. Clusters usually use commodity equipment, while massively parallel computers use custom made equipment. Some common interconnects are Myrinet [15], InfiniBand^{TM} [1], Quadrics [5], Gigabit Ethernet [18, 2].

Routing on these networks is deterministic. That is, all packets between a given source/destination pair will follow the same path. One of the main benefits is that in-order arrival of packets is preserved. However, deterministic routing usually makes an inefficient use of network resources. The alternative is to use adaptive routing where packets may take different paths depending on the current traffic conditions, thus, avoiding contention in the network.

It is important to select the right routing algorithm as it must be able to leverage the full potential of the topology. If the topology is regular, it is wise, with regards to performance, to use a topology dependent routing since it would be able to exploit the regularity of the topology. Dimension-Order Routing (DOR) is such an algorithm suitable for meshes. Unfortunately, those algorithms are sensitive to topology changes. A faulty switch or link will degrade the topology into an irregular one and then the algorithms will

1-4244-0054-6/06/$20.00

fail. A simple way to achieve fault-tolerance is by the use of a topology agnostic routing algorithm in combination with *static* reconfiguration. In such a static fault-model the network enters a reconfiguration phase when a fault is discovered, the system is halted and drained of packets, then all routing tables are recomputed. An alternative approach is to use a *dynamic* fault-model, where there is no need to halt the network. When a fault occurs, the faulty link or switch will be marked and another path configured. Notice that the technology needs to support the discovery of broken links and switches, as well as any features required for dynamic reconfiguration [16]. Furthermore, during reconfiguration there will be a short period of time where some routing tables are in an inconsistent state. This must be handled properly to avoid possible packet loss, deadlock and livelock. Unfortunately, such functionality is complex and not commonly available, thus static reconfiguration in combination with a topology agnostic routing algorithm is simpler, and currently the only practical approach in common interconnects.

Several high performance topology agnostic routing algorithms exist, such as LASH [20], TOR [8], LASH-TOR [19], DL [14], and multiple virtual networks [9]. Common for all these algorithms are that they require virtual channels, a feature that not all technologies support. Furthermore, if virtual channels happen to be supported the number of available channels is often limited, and often dedicated to certain purposes such as quality of service. There are, however, also topology agnostic routing algorithms that do not rely on virtual channels. Prominent examples are up*/down* [12], lturn [13], smart-routing [11], and FX [6]. These algorithms have in common that they are based on turn prohibition, a methodology which avoids deadlock by prohibiting a subset of all turns in the network. A problem with this approach is that it is unable to guarantee shortest path routing and unable to exploit any regularity in the underlying topology.

2 Motivation

When designing a routing algorithm a key requirement is that it should be deadlock free. This is guaranteed if the channel dependency graph in question is devoid of cycles and this can be achieved in several ways, such as through the use of layered routing or through turn restrictions. The former requires virtual channels whereas routing based on turn restrictions does not have this requirement, thus it is suitable for a larger range of network technologies.

Many algorithms based on turn restrictions exist, the most well known being the up*/down* routing algorithm [12]. In up*/down* we first create a breadth-first spanning tree of the topology. Then we assign a direction to each link (either *up* or *down*). Turns are restricted according to the up*/down* rule [12]: "a packet may never traverse a link in the *up* direction after having traversed one in the *down* direction." As an example, Figure 1.a shows a 3x4 mesh with two faulty links, and up*/down* restrictions applied. As up*/down* uses *bidirectional* turn restrictions it may cause an uneven distribution of traffic by having many paths crossing the same link, which results in lower performance. The FX routing scheme avoids this problem and improves performance by introducing *unidirectional* routing restrictions [6]. Common for both approaches is that, as soon as the spanning tree has been calculated, the turn restrictions are fixed. I.e. once the spanning tree root is selected, all routing restrictions are fixed. Since there are twelve root candidates, we have only twelve possible combinations of routing restrictions.

In this paper we suggest a new way to apply turn restrictions called *Segment-Based Routing* (SR). SR Works by partitioning a topology into subnets, and subnets into segments. Then we place bidirectional turn restrictions *locally* within a segment. As segments are independent, we are free to place turn restrictions within a segment independently from other segments. This *locality independence* property adds a new dimension to the placement of routing restrictions, and allows us to disentangle ourselves from the restraints found in current algorithms such as up*/down and FX. Consider Figure 1.b, here we have partitioned our example topology into four segments and labeled it with all possible turn restrictions. By placing only *one* such restriction within each segment, we are able to guarantee deadlock-free routing and a connected network. Figure 1.c shows a possible set of restrictions applied to our example topology. The main contribution of SR is that we are able to choose among 48 different combinations of routing restrictions. This new freedom gives us the possibility to optimise our selection of routing restrictions according to throughput, latency or other criteria. If we want to optimise for latency, we choose the set of restrictions maximising the number of shortest paths. While we choose the set of restrictions giving best distribution of traffic when we want to optimise for throughput.

To reckon for fault-tolerance we have designed a segmentation algorithm optimised for meshes and tori, where the algorithm is able to exploit the regularity of such topologies, while at the same time being insensitive to faults. This is possible since the algorithm really is topology agnostic, but a segmentation algorithm for totally random topologies is a topic for further research. The rest of the paper is organised as follows. In Section 3 we present the SR algorithm. Then, in Section 4 we evaluate SR in comparison with up*/down* and FX. Finally in Section 5 we conclude.

3 Segment-based Routing

3.1 Informal Description

The key concept of the SR algorithm is the partitioning of a topology into subnets, and subnets into segments. This allows us to place bidirectional turn restrictions *locally* within a segment. As segments are independent, we gain the

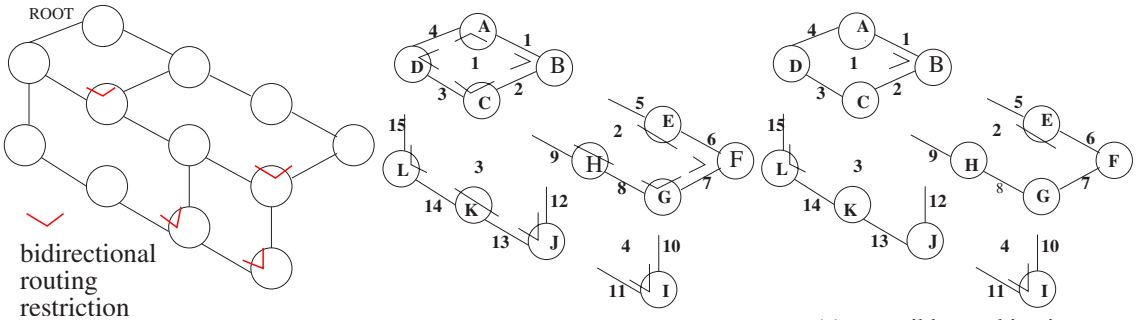


Figure 1. Possible set of routing restrictions.

freedom to place turn restrictions within a segment independently from other segments.

A segment is de ned as a list of interconnected switches and links, as shown in Figure 1.b. Here we have four segments labeled $1-4$, where segment 1 consists of the switches $\{A, B, C, D\}$ and the links $\{1, 2, 3, 4\}$. Segment 2 consists of switches $\{E, F, G, H\}$ and the links $\{5, 6, 7, 8, 9\}$, and so on for the rest of the segments. All network links and switches belong to one and only one routing segment (i.e. segments are disjoint), and every routing segment, except the initial segment, starts and ends on a switch already part of a computed segment. Furthermore, we will group segments into subnets. A subnet is a set of switches and links (i.e. one or more segments) that is connected to the rest of the network (other subnets) through only one link. The use of subnets is motivated by some special cases that will be described in section 3.4.

When the complete topology is partitioned into segments we can add routing restrictions. In Figure 1.c we have placed one routing restriction in each of the four segments from Figure 1.b, these routing restrictions guarantee deadlock-free routing and preserve connectivity. In the rst segment, the placement of a routing restriction in any one of the four switches will result in a deadlock-free routing algorithm for this segment. Furthermore, as the second segment starts and ends on switches already belonging to the rst segment, connectivity among the switches is guaranteed through the rst segment. Thus, we can place a turn restriction in the second segment, breaking the cycles that can be found through the segment, without worrying about connectivity. The main challenge lies in nding the segments, as this is critical to guarantee deadlock-freedom, preserve connectivity, and gain performance.

3.2 Segment-Based Routing Algorithm

The complete algorithm[1] is shown in Figure 2. It consists of the procedure `compute_segments`, which searches for all segments. And the procedure `find`, which tries to nd a new segment starting in the switch received as an argument. Throughout the execution of the algorithm, the switches and links can be in the following states:

- *not visited*. Initially all the switches and links are in the state *not visited*. This is denoted by the variable `.visited` being *false*.
- *visited* . A switch or link becomes *visited* once it belongs to an already computed routing segment. This is denoted by the variable `.visited` being *true*.
- *tvisited*. 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*. This is denoted by the variable `.tvisited` being *true*.
- *starting*. A switch is marked as the *starting* switch if it is the rst switch chosen to compute a segment within a subnet. This is denoted by the variable `.starting` being *true*.
- *terminal*. A switch is marked as *terminal* if, through at least one of its links, no new segment is found. This is denoted by the variable `.terminal` being *true*.

The `compute_segments` procedure (Figure 2) searches for all segments. First, it chooses a random switch as the starting point of the rst segment in the rst subnet[2]. Then the selected switch (*sw*) is marked as *starting* and *visited*, and added to the rst subnet. Second, the `find` procedure is used to nd a segment starting in switch *sw*. Such a segment only exists if it is possible to arrive back at *sw* through a set of switches and links not already *visited*. On success, the `find` procedure updates all the switches and links belonging to the new segment, i.e. all of them are marked as *visited* and as belonging to the current subnet. On fail, the `find` procedure leaves all links

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

[2]As any switch could be selected as the starting switch, the procedure can end up with different solutions. Later, we will indicate a better criteria for computing segments for 2D meshes.

```
procedure compute_segments()
var
  s : segment list
  sw : switch
  c,n : integer # current subnet and 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 ( nd(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  nd(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
                  nd(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.

and switches at their initial state. If the procedure fails, it means that there are no new segments reachable from this switch, and the switch is marked as *terminal*. Third, the procedure `next_visited` is used to search for a switch marked as *visited*, belonging to the current subnet, and with at least one link not marked as *visited*. If such a switch is found it is used to search for new segments as just described. Otherwise, the procedure `next_not_visited` is used to search for a switch that is not marked as *visited*, not marked as *terminal*, and attached to a terminal switch. If successful, a new subnet is started and a new segment is searched for from this switch. On failure, we know that all switches have been searched and that all switches and links are part of a segment and subnet.

The procedure `find` is responsible for nding, from a given starting point, a segment ending on a visited switch and made of switches and links not visited. During the search the current switch is marked as *tvisited* and is added to the current segment *segm*. Next, a set of links attached to the current switch is built (*suitable_links*). This set only includes links not marked as *visited*, nor as *tvisited*. If the set is empty, then there are no suitable links and the *nd* procedure has failed in nding a new segment. Otherwise, the links in the set are considered in the order found. Order is important, since the segments found may be different if the order of search is changed. When the links are searched, they are rst marked as *tvisited*, then added to the current segment *segm*. Then the switch at the other end of the link is inspected. If this switch is marked as *visited*, or if a recursive call of the `find` procedure from the neighbour switch returns true (i.e. such a switch is found at a later stage), then a new segment has been found. If we are unable to nd a new segment we return with failure.

Figure 3 shows an example run of the algorithm. The topology is made of 12 switches connected with 14 links. We start by randomly selecting a switch, which yields switch I. From switch I we nd segment ns_1 consisting of the following links and switches: $\{I, 8, E, 5, F, 9, J, 12\}$. This is the only segment that can be found from I as a segment should always end in a visited switch. Next, we search for a switch marked as *visited*, which includes $\{I, E, F, J\}$, and that has a link not visited, which reduces the candidates to $\{F\}$. However, from F no new segment can be found, and therefore, the switch is marked as *terminal*. All switches belonging to the segments computed so far belong to the rst subnet (SN_0). Next, we search for a switch not visited and attached to a terminal switch. The unique solution is switch G. At this step, a new subnet is started (SN_1), and G is marked as *starting* and *visited*. From G a new segment is found: $\{G, 3, C, 2, D, 4, H, 11, L, 13, K, 10\}$. Next, we search for a switch marked as *visited* and with one or more attached links not visited, which results in $\{G, C, H\}$. G is searched and a new segment containing only link 7 is found. Next, from C no new segment is found and the switch is marked as *terminal*.

We continue by inspecting B, as it is attached to a terminal switch and not marked as *visited*, and decides that no new segment is found. Thus, it is marked as *terminal*. Finally we perform the same action on A. When nished, three segments and four subnets are found, including four starting switches and three terminal switches.

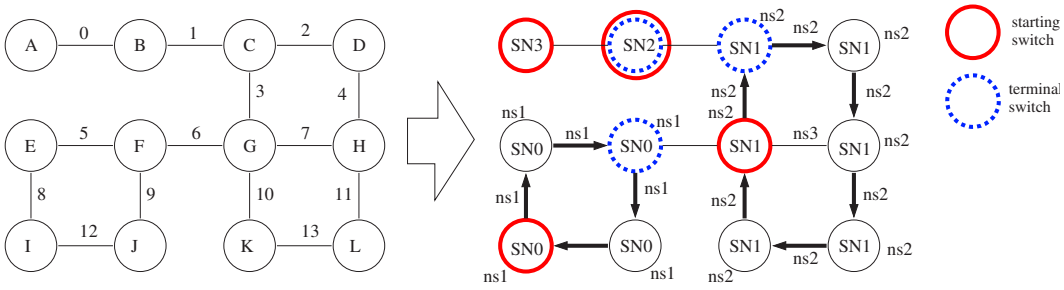


Figure 3. Example of computing routing segments.

Through the previous steps we will nd three types of routing segments:

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

In order to ensure deadlock-freedom and preserve connectivity, the routing algorithm must place routing restrictions in each routing segment as shown in Figure 4.a. In particular, for a starting segment, the cycle can be broken by placing a bidirectional routing restriction on any switch except the starting one (as it could introduce a cycle between two subnets, refer to section 3.4). For regular segments, cycles are broken by placing one bidirectional routing restriction on any switch belonging to the segment. Finally, for unitary segments, no traf c can be allowed to cross the link in order to avoid deadlock. Thus, on one side of the link a bidirectional routing restriction must be placed between this link and every other link attached to switch.

3.3 Deadlock Freedom and Connectivity within a Subnet

We will start by demonstrating deadlock-freedom and connectivity for a single subnet. This means that only one *starting* switch exists in the network, and that no *terminal* switches are present. Later on we will extend this to cover multiple subnets.

Definition 1 A cycle is defined as a set of n switches and n channels, where channel c_i connects switches sw_i and sw_{i+1} and channel c_n connects switches sw_n and sw_1. Thus, a cycle can be represented as sw_1,c_1,sw_2,c_2,sw_3,c_3,...,sw_n,c_n.

Lemma 1 Given a cycle in the network, there is at least one complete segment included in the cycle.

Proof 1 Assume that there is a cycle sw_1,c_1,sw_2,c_2,sw_3,c_3,...,sw_n,c_n.

If all the switches sw_i and channels c_i in the cycle belong to the same segment, then, the segment is a starting segment.

Otherwise, if there are some switches or channels belonging to different segments we must distinguish between two general cases.

In the first case all the switches belong to the same segment NS_a, but at least a channel belongs to another segment NS_b. This means that there is a link belonging to NS_b attached to two switches belonging to NS_a. Thus, this link forms a unitary segment and there is a complete segment in the cycle.

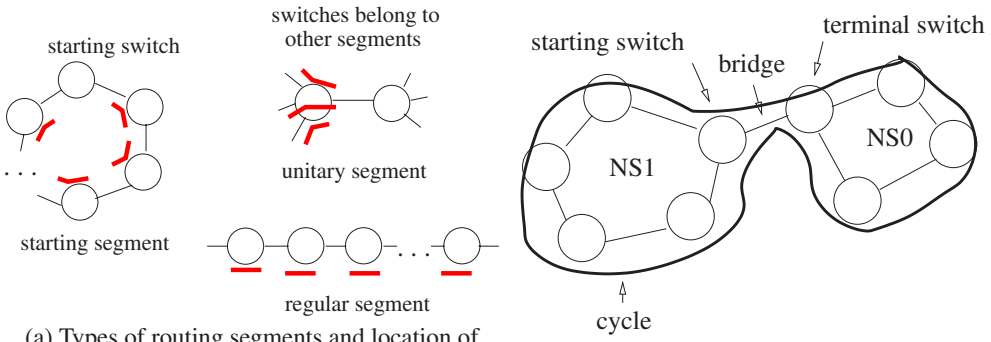
In the second case there are two neighbour switches in the cycle that belong to different segments NS_a and NS_b.

If the link connecting both switches belongs to a different segment (NS_c), then it forms a unitary segment and, therefore, the cycle includes one complete segment.

If the link belongs to the same segment as one of the switches (i.e. NS_b), the link is the last element of the segment and, therefore, NS_b has been computed after NS_a (as it departs from another segment). In this situation, the NS_b segment will have consecutive elements (taking the direction contrary to NS_a in the cycle) and the last element will be a link or a switch. If the last element of NS_b in the cycle is a link, then NS_b is a regular segment included in the cycle. Therefore, there is a complete segment in the cycle.

Otherwise, if the segment NS_b ends in a switch, then this switch will be attached to another segment in the cycle (lets say this new segment is NS_c) and it follows that NS_c is computed after NS_b. As NS_b was computed after NS_a, NS_c also has been computed after NS_a. This deductive process is repeated until the cycle is closed.

At the end, a complete segment is found within the cycle (based on the previous deductions) or a contradiction is reached. In the extreme case, a switch belonging to a network segment (NS_x) is attached through a link to a switch belonging to NS_a (closing the cycle). In this situation, the



(a) Types of routing segments and location of routing restrictions

(b) Example of links connecting subnets

Figure 4. (a) Segments and restrictions, and (b) subnets.

link belongs to a different segment (a unitary segment is found), or it belongs to NS_x (NS_x is a regular segment and is completely included in the cycle), or it belongs to NS_a. In the latter case, NS_a should be computed after NS_x. However, NS_a was computed before NS_b, which was computed before NS_x. Thus, we have a contradiction. Figure 5.a shows a summary of all the possibilities.

Lemma 2 Given any topology, the SR algorithm will always find a deadlock-free solution.

Proof 2 The way routing restrictions are placed in segments ensures that no cycle can be formed through an entire segment. In order to introduce a deadlock, a cycle must be present in the network. From the previous lemma it is deduced that a cycle includes at least one complete segment. Thus, as no cycle can be formed across segments and a cycle contains at least one segment, it is deduced that no cycle can be formed.

Lemma 3 Given any topology, the SR routing will always keep connectivity among all switches.

Proof 3 At the starting segment computed by the SR algorithm, connectivity among all the switches of the segment is ensured. Since, as channels are bidirectional, the segment starts and ends at the same switch and only one routing restriction is placed in the segment, all switches can be reached from all other switches within the segment.

The next segment starts and ends at switches belonging to the initial segment. For a regular segment (contains at least one switch), a routing restriction is placed within the segment. This ensures that all switches on one side of the routing restriction can reach all switches on the other side of the routing restriction through the initial segment. Thus, connectivity is guaranteed among all switches belonging to the initial and second segment. For a unitary segment, the connectivity is ensured as no switch is included in a unitary segment. For each added segment the connectivity among switches in the new segment is ensured as described above. Thus, in the end, all switches are connected.

3.4 Deadlock Freedom and Connectivity among Subnets

Now, we extend the above to networks where SR detects different subnets. Whenever a *terminal* switch is present in a subnet, it will be attached to the *starting* switch of another subnet and the corresponding link will not belong to any subnet nor any segment. This link will be referred to as a *bridge*. Figure 4.b shows an example.

Lemma 4 From a starting switch of a subnet, the same switch can not be reached within its subnet.

Proof 4 The starting switch is used as the starting point to find the first routing segment for the subnet (a starting segment). As the first routing segment ends in the starting switch, and a routing restriction is placed in a switch different from the starting switch, the starting switch can not be reached from itself within the routing segment.

For every new segment (regular or unitary) computed within the same subnet, restrictions will be added in such a way that the switches attached to endpoints of the segment will be unconnected through the new segment. Therefore, the starting switch can not reach itself by using any new segment. Furthermore, the starting switch can not reach itself through any of the switches and links of its subnet.

If there is at least one bridge in the cycle, then that bridge must appear in the cycle twice. This is because there is only one connection between different subnets, as shown in Figure 4.b.

At least one of the switches attached to the bridge is a *starting* switch. In order to get such a cycle, the starting switch must be able to reach itself from within its subnet. As has been demonstrated, this is not possible and therefore, no cycle can be formed when using bridge links. Additionally, terminal switches and bridge links ensure connectivity as no routing restriction is added to those components.

3.5 SR Algorithm Applied to 2D Meshes

It is obvious that the way segments are computed and routing restrictions are placed, will have a great impact on performance. A random search for segments may result in performance worse than what could be achieved with up*/down*. Therefore, segments and routing restrictions must be computed with care, taking into account some parameters from the topology. As a first effort, we will show an example of applying SR on 2D meshes.

Figure 5.b shows a 5×5 mesh already partitioned into segments. These segments are computed by searching from top to bottom, each row in a different direction. The first row from left to right, the second row from right to left and so on. Each segment will be made as short as possible by exploring links closer to already visited switches first. The routing restrictions are then placed in such a way that all routing restrictions on the same row will be located in the same relative position (Figure 5.b).

This way of searching for segments and placing routing restrictions also works when the topology has link failures, as shown in Figure 5.c. Here the method is applied to a 5×5 mesh with three failures. When dealing with the failures the algorithm will obtain different segments, but all segments will still be kept as short as possible.

3.6 Computational Cost

SR can be viewed as an algorithm executed in three phases. In the first phase segments are computed. The way segments are computed will influence the computational cost of the algorithm. E.g. a random search with a recursive function may exhibit an excessive computational cost however the search algorithm proposed in the previous section will lower the cost as links are only visited once. Thus, for a 2D mesh, the cost for this phase becomes $O(m)$, where m is the number of links in the network.

In the second phase routing restrictions are placed. As proposed above, the computational cost of this phase will be $O(s)$, where s is the total number of segments. Finally, in the third phase, paths are computed according to the routing restrictions. The complexity of this phase may vary greatly depending on the desired final performance. A straightforward random path selection will have 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. FX, for instance, uses an algorithm that searches for a set of paths that minimises the maximum number of paths that cross any link (minimising the crossing path metric). If an optimisation like this is applied, then the cost will be driven by the complexity of such algorithm. As stated in [7] the best case scenario for the computational cost of FX is $O(n^3)$ not considering the selection of the best DFS tree.

4 Performance Evaluation

In this section we will compare the SR algorithm, both analytically and through simulations, with the UD and FX schemes. Analytical studies are performed by looking at path and routing statistics of the three algorithms, while empirical performance numbers are obtained through packet level simulations.

4.1 Simulation Tool and Network Model

Our simulator aims to model arbitrary switch-based networks with point-to-point links. Each switch has a non-blocking crossbar connecting input and output ports, which allows multiple packets to be simultaneously transmitted. The crossbar arbiter is based on FIFO request queues, with one queue per output port, and buffers of 1 KB at both the input and output side. The crossbar is able to transmit one byte per connection, per cycle. A routing decision is made at every switch, by accessing the local routing table. This table maps destinations to output ports, with one mapping for each possible destination. When the routing decision is complete the packet is forwarded to the output buffer if there is sufficient buffer capacity. Otherwise, the packet will remain in the input buffer until the buffers became available, modelling an input-output buffered switch. The routing time at each switch is set to 100 ns. This includes the time to access the routing table, the crossbar arbitration time, and the time to set up the crossbar connections.

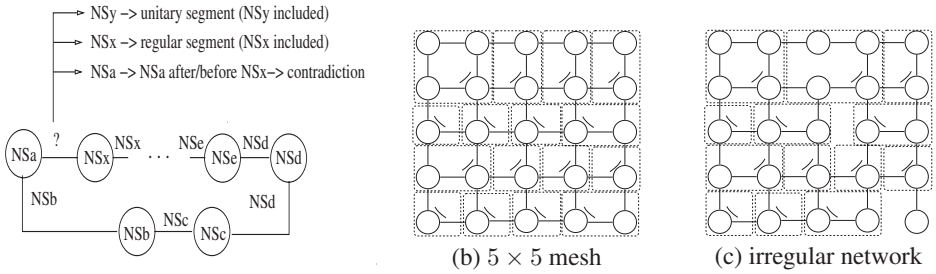
Our switch uses virtual cut-through switching [17] and credit-based flow control. In our simulations we use a constant packet size of 32 bytes, and a credit size of 64 bytes. We also model the fly time,which depends on the link length and the propagation delay of the cable. We model 12 meter copper cables with a propagation delay of 5 ns/m, leading to 60 ns of fly time. For each simulation run, we assume the same constant packet rate for each end-node. Once the network has reached steady state, the packet generation rate will be equal to the packet reception rate. We will evaluate a range of loads, from low load to saturation, with uniform, bit reversal, and hotspot traffic patterns. For hotspot traffic, 4% of all sources will inject packets with the same destination, while the rest will inject packets to random destinations. We have evaluated meshes and tori of sizes 4×4, 8×4 and 8×8. Additionally, we have modeled these topologies with 5% of randomly-injected link failures.

We present results for UD, FX/DOR[3] and SR, and we plot the average packet latency[4] versus the average accepted traffic [5] measured in bytes/ns/switch. SR segments are computed as described in section 3.5 and source/destination paths are computed using the path balancing algorithm developed in [10]. This method minimises the deviation of

[3] For regular topologies FX equals DOR.

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

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



(a) deadlock-freedom (b) 5×5 mesh (c) irregular network

Figure 5. Example of (a) deadlock-freedom and (b) SR applied to a different topologies.

	SR					UD				FX			
			APL	ALW			APL	ALW			APL	ALW	
Topology	#Segm	#Rest	Avg	Avg	STD	#Rest	Avg	Avg	STD	#Rest	Avg	Avg	STD
Mesh 4x4	9	18	2.50	13.33	3.10	20	2.50	13.33	4.80	36	2.50	13.33	1.91
Mesh 8x4	21	42	3.87	38.15	15.49	42	3.87	38.15	20.09	84	3.87	38.15	15.01
Mesh 8x8	49	98	5.25	96.00	31.31	98	5.25	96.00	43.40	196	5.25	96.00	27.77
	SR					UD				FX			
			APL	ALW			APL	ALW			APL	ALW	
Topology	#Segm	#Rest	Avg	Avg	STD	#Rest	Avg	Avg	STD	#Rest	Avg	Avg	STD
Torus 4x4	25	64	2.10	8.43	2.02	62	2.00	8.00	3.96	96	2.00	8.00	0.00
Torus 8x8	65	194	4.58	73.50	28.07	160	4.50	72.00	50.35	288	4.50	72.00	28.06
	SR					UD				FX			
			APL	ALW			APL	ALW			APL	ALW	
Topo-Seed	#Segm	#Rest	Avg	Avg	STD	#Rest	Avg	Avg	STD	#Rest	Avg	Avg	STD
Irreg - 1	45	90	5.39	98.53	45.37	86	5.67	109.55	63.16	170	5.54	106.96	60.32
Irreg - 2	43	86	5.34	103.34	49.95	86	5.39	104.19	66.96	176	5.55	107.29	71.29
Irreg - 3	43	86	5.33	103.26	43.19	86	5.46	105.66	63.19	171	5.43	104.92	59.06
Irreg - 4	43	86	5.38	104.06	40.99	86	5.45	105.40	63.50	170	5.43	104.93	48.79
Irreg - 5	43	86	5.39	104.23	40.84	86	5.62	108.58	60.39	176	5.63	108.72	50.55

Table 1. Analytical results for different network topologies. APL stands for Average Path Lenght whereas ALW stands for Average Link Weight. Irreg - x is derived from a 8×8 mesh with 5% of link failures

link weight. Therefore, it is similar to the one used in FX. Due to lack of space we only present results for a subset of all simulations[6].

4.2 Analytical Results

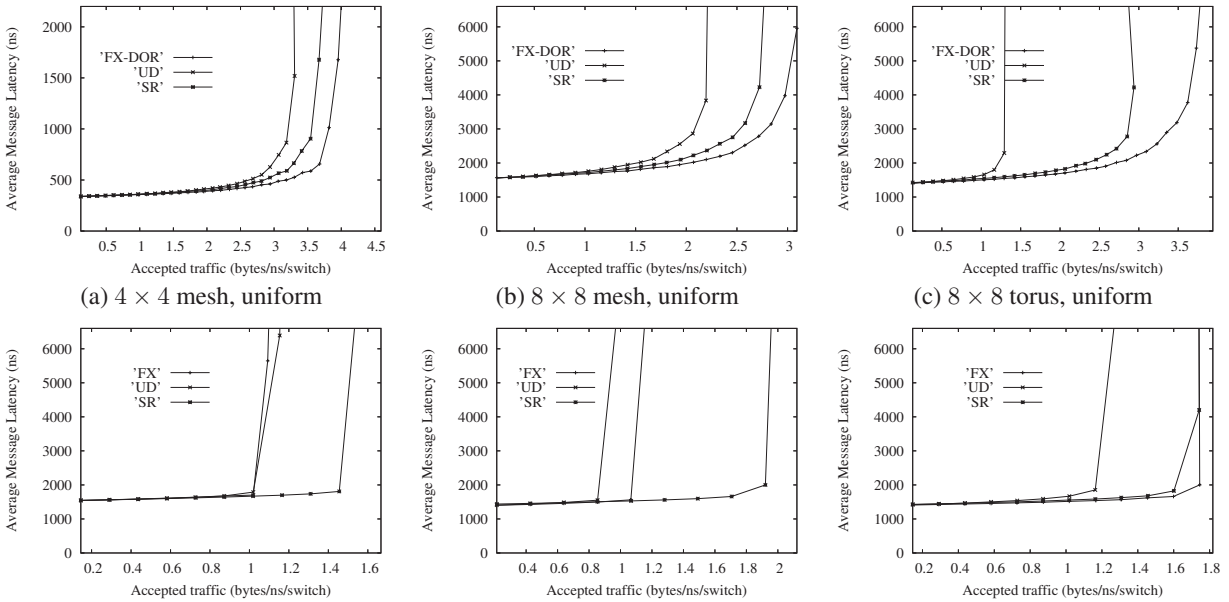
We have compared three properties of SR, UD and FX algorithms: *average path length*, *average link weight* (i.e. average number of paths crossing a link), and the total number of *unidirectional turn restrictions* applied. For SR, we also included the *number of segments* computed. These statistics are shown in Table 1 for every scenario considered.

For meshes without faults, both the average path length and the average link weight for all algorithms are the same, since all algorithms always use minimal paths. The standard deviation, however, is higher in SR compared to FX. It is reasonable, thus, to assume that FX will perform better than SR when using a uniform traffic pattern (see Section 4.3). Also UD has a standard deviation higher than SR, thus we expect SR to outperform UD for uniform traffic. The high standard deviation observed for UD is due to UD's tendency to concentrate traffic around the root of the UD tree.

For tori without faults, Table 1 shows that SR achieves a slightly higher average path length compared to UD and FX. This indicates that not all paths are minimal, and this is due to a weakness in SR. In its current form the segmentation algorithm is unable to exploit the wraparound links present in tori, while UD and FX are able to use these links. Still, we should expect SR to outperfrom UD when using uniform traffic, since the observed standard deviation for UD is almost twice that of SR. Looking at the number of segments and turn restrictions for SR, we observe that we prohibit more turns than the number of segments. This reveals that there exist many unitary segments. These unitary segments are formed from the wraparound links.

In our last set of results we consider 8×8 meshes with faults, In this situation SR really shows its benefits. The SR algorithm performs better than both UD and FX for all metrics. It has the shortest average path length, lowest average weight, and the lowest standard deviation. This is

[6]Similar results have been obtained for all the topologies.



(a) 4×4 mesh, uniform (b) 8×8 mesh, uniform (c) 8×8 torus, uniform

(d) 8×8 mesh bit-reversal pattern (e) 8×8 torus bit-reversal pattern (f) 8×8 torus with 4% hot-spot traffic

Figure 6. Average packet latency vs throughput for different topology/traffic patterns.

due to the use of segments, which makes it possible for SR to retain the regularity of the topology in the areas where there are no faults. A feature that makes SR a very good alternative for semi-regular (regular networks with failures) topologies, as will be shown in section 4.3.

When comparing FX with SR, one should remember that the use of unidirectional routing restrictions in FX plays a key role in balancing traffic in regular topologies. FX achieves perfect balancing for a 4×4 torus. However, the gap between SR and FX, for regular topologies, narrows as the number of switches increases. This happens even if SR only uses bidirectional routing restrictions, which leaves room for future improvement. When considering regular topologies with faults, FX looses much of its advantage over SR, which can be seen from the values in Table 1.

4.3 Simulation Results

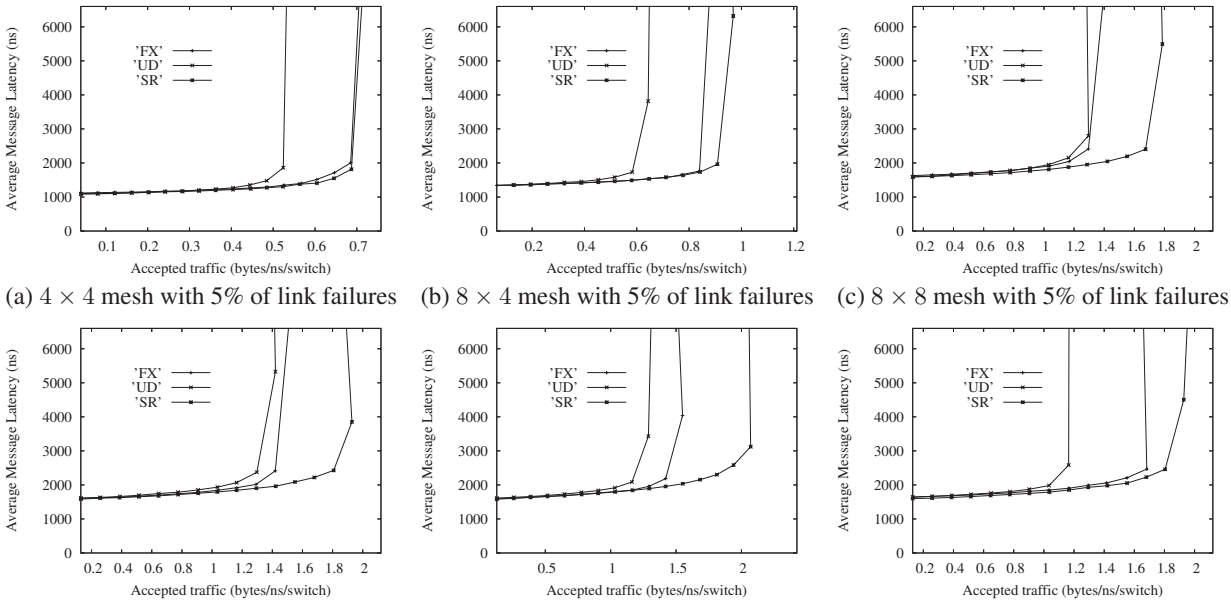
Figure 6 shows the performance achieved by UD, FX, and SR for meshes of size 4×4, 8×8 and for a 8×8 torus, all with uniform traffic. We see that the standard deviation of the average link weight shown in Table 1 dictates the performance achieved by each algorithm. In particular, FX beats SR with a factor of 1.075 on the 4×4 mesh, and a factor of 1.11 on the 8×8 mesh[7], while SR outperforms UD by a factor of 1.14 in a 4×4 mesh and 1.25 in a 8×8 mesh. In this scenario FX performance is outstanding, since FX is identical to dimension order routing (DOR). It is known that DOR behaves well with uniform traffic, even better than adaptive routing in meshes, but performance drops with other traffic patterns. Let us therefore analyse the bit reversal and hotspot case shown in Figure 6. We see that for bit reversal traffic, SR shows an increase in throughput, compared to FX, by a factor of 1.42 for a 8×8 mesh, and a factor of 1.8 for a 8×8 torus. For hotspot traffic both algorithms exhibit similar performance. These results underline the potential of SR in regular networks.

When considering regular topologies with faults, the performance of SR is further strengthened. Figure 7 shows the results for three different meshes with 5% of randomly-injected link faults and uniform traffic. For the 4×4 mesh SR and FX are equal, but as the network size grows so does the performance advantage of SR. For the 8×8 mesh, SR achieves an increase in throughput by a factor of 1.4 compared to FX. Altogether, this shows that the SR algorithm performs well as a fault-tolerant routing algorithm for regular topologies. With the main strength lying in the locality independence property for the placement of routing restrictions, and its ability to exploit semi-regular topologies. Furthermore, it indicates that SR has the potential of becoming a topology agnostic routing methodology.

5 Conclusions

We have proposed a segment-based routing algorithm, where the novelty resides in the introduction of a *locality independence* property. This property adds a new dimension to the placement of routing restrictions, and allows us to disentangle ourselves from the restraints found in current algorithms such as up*/down and FX. The SR algorithm's use

[7] Similar results are also achieved in 8×8 torus



(a) 4×4 mesh with 5% of link failures (b) 8×4 mesh with 5% of link failures (c) 8×8 mesh with 5% of link failures

(d) 8×8 mesh with 5% of link failures (e) 8×8 mesh with 5% of link failures (f) 8×8 mesh with 5% of link failures

Figure 7. Average packet latency vs throughput. Uniform distribution of packet destinations.

of segments, makes it possible to exploit the semi-regularity found in irregular topologies, such as meshes and tori with faults. Results show that SR performance is similar to the one obtained by FX when used in regular topologies, while it supersedes FX when we introduce link failures.

As future work we plan to improve the segmentation algorithm to give better results with tori, as well as for highly irregular topologies. As an optimisation we will use unidirectional routing restrictions as used in FX.

References

[1] Infiniband ^{TM} architecture. In *Specification Available at http://www.infinibandta.com*, volume 1, June 2001.

[2] Ieee std. 802.3ae-2002 10gb/s ethernet. In *IEEE Standards Association*, 2002.

[3] Thunderbird supercomputer. sandia national laboratories. Nov. 2005.

[4] Top500 supercomputer website. supercomputer's ranking. In *Available at www.top500.org/*, Nov. 2005.

[5] e. a. F.Petrini. The quadrics network (qsnet): high performance clustering technology. In *IEEE Hot Interconnects*, pages 125–130, Aug. 2001.

[6] e. a. J.C.Sancho. A flexible routing scheme for networks of workstations. In *International Conference on High Performance Computing*, Oct. 2000.

[7] e. a. J.C.Sancho. Contribucion al diseo de algoritmos de encaminamiento en redes de estaciones de trabajo. In *Tesis Doctoral, Universidad Politecnica de Valencia Ref. B-TES 3681 B*, Dec. 2002.

[8] e. a. J.C.Sancho. Effective methodology for deadlock-free minimal routing in infiniband networks. In *International Conference On Parallel Processing*, Aug. 2002.

[9] e. a. J.Flich. Improving infiniband routing through multiple virtual networks. In *International Symposium on High Performance Computing*, Dec. 2002.

[10] e. a. J.Flich, J.Duato. Combining in-transit buffers with optimized routing schemes to boost the performance of networks with source routing. In *2000 International Symposium on High Performance Computing*, Oct. 2000.

[11] V. L.Cherkasova and T.Rokicki. Fibre channel fabrics: Evaluation and design. In *International Conference on System Sciences*, Feb. 1995.

[12] e. a. MD. Schroeder. Autonet: A high-speed, self-configuring lan using point-to-point links. In *Journal on Selected Areas in Communications*, volume 9, Oct. 1991.

[13] e. a. M.Koibuchi. Lturn routing: An adaptive routing in irregular networks. In *International Conference on Parallel Processing*, volume 3, pages 374–383, Sept. 2001.

[14] e. a. M.Koibuchi. Descending layers routing: A deadlock-free deterministic routing using virtual channels in system area networks with irregular topologies. In *International Conference on Parallel Processing*, Oct. 2003.

[15] e. a. N.J.Boden. Myrinet: A gigabit per second lan. In *IEEE Micro*, number 15 in 1, pages 29–35, 1995.

[16] e. a. O.Lysne. Simple deadlock-free dynamic network reconfiguration. In *Lecture Notes in Computer Science*, volume 3296, Dec. 2004.

[17] P.Kermani and L.Kleinrock. Virtual cut-through: A new computer communication switching technique. In *Computer Networks*, volume 3, pages 267–286, 1979.

[18] R.Seifert. Gigabit ethernet. In *Addison-Wesley, ISBN: 0-201-18553-9*, Apr. 1998.

[19] e. a. T.Skeie. Lash-tor: A generic transition-oriented routing algorithm. In *IEEE International Conference on Parallel and Distributed Systems*, July 2004.

[20] O. T.Skeie and I.Theiss. Layered shortest path (lash) routing in irregular san. In *IEEE Communication Architecture for Clusters*, Sept. 2002.