Scalable Computing: Practice and Experience
Volume 7, Number 3, pp. 51–59. http://www.scpe.org
ISSN 1895-1767

AN EFFICIENT FAULT-TOLERANT ROUTING STRATEGY FOR TORI AND MESHES*

M. E. GÓMEZ, P. LÓPEZ AND J. DUATO†

Abstract.
In massively parallel computing system, high performance interconnection networks are decisive to get the maximum performance. While routing is one of the most important design issues of interconnection networks, fault-tolerance is another issue of growing importance in these machines, since the huge amount of hardware increases the probability of failure. This paper proposes a mechanism that provides both, scalable routing and fault-tolerance, for commercial switches to build direct regular topologies, which are the topologies used in large machines. The mechanism is very flexible and the hardware required is not complex. Furthermore, it allows a high number of faults having a minimal effect on performance.

Key words. Fault-tolerance, memory-effective routing, regular topologies, adaptive routing.

1. Introduction. In large parallel computers high-performance interconnection networks are decisive for reaching the maximum performance. While routing is one of the most important design issues of interconnection networks, fault-tolerance is another issue of growing importance, since the huge amount of hardware of large machines increases the probability of failure. Failures in the interconnection network may isolate a large fraction of the machine. Two fault models have been defined in order to deal with permanent faults: static and dynamic. In a static fault model, all the faults are known in advance when the machine is (re)booted. It needs to be combined with checkpointing techniques in order to be effective. In a dynamic fault model, once a new fault appears, some techiques are applied in order to appropriately avoid the faulty component without stopping the machine. Several approaches have been used to tolerate faults in the interconnection network. Replicating components incurs in a high extra cost. Another technique is based on reconfiguring the routing tables. This technique is extremely flexible but it may kill performance. However, most of the solutions proposed in the literature are based on fault-tolerant routing algorithms able to find an alternative path when a packet can encounter a fault. We proposed [5] a fault-tolerant routing mechanism for direct topologies that incurs in a minimal decrease of performance in the presence of faults, and reaches a reasonably high level of fault-tolerance, without disabling any healthy node and without requiring too much extra hardware.

As commented, in addition to fault-tolerance, routing is another important issue. The routing strategy determines the path that each packet follows between a source–destination pair. Routing is deterministic if only one path is provided, or adaptive, if several paths are possible between a source-destination pair. Routing strategies can be also classified depending on the place where routing decisions are taken. When using source routing, the source node calculates the path prior to packet injection and stores it in the packet header. When using distributed routing, each switch computes the next output port. The packet header only contains the destination node. Distributed routing has been used in most hardware routers for efficiency reasons.

Distributing routing is implemented following two different approaches. In the first approach, some hardware in the switches computes the output port as a function of the current and destination nodes and the status of the output port. With the use of clusters of workstations, routing based on forwarding tables was introduced. In this approach, there is a table at each switch that contains, for each destination node, the output port that must be used. The main advantage of table-based routing is that any topology and any routing algorithm can be implemented with the same commercial switches, but it is not scalable.

High-performance switch-based point-to-point interconnects are used in large cluster-based machines. In these machines, the topology is regular. Either direct networks (tori and meshes) or indirect multistage networks are the usual topology. In these large machines, source routing is inefficient due to the increased header size, and routing based on forwarding tables is also inefficient due to the table size. On the other hand, since general purpose switches are used, specifically designed routing hardware is not feasible. We proposed [6] a routing strategy for switch-based networks, Flexible Interval Routing (FIR), based on Interval Routing that allows to implement the most commonly-used deterministic and adaptive routing algorithms in meshes and tori, with a total memory requirements $O(\log N)$.

*This work was supported by the Spanish MCYT under Grant TIC2003-08154-C06-01.
†Dept. of Computer Engineering, Universidad Politécnica de Valencia, Camino de Vera, 14, 46071–Valencia, Spain, megomez@disca.upv.es

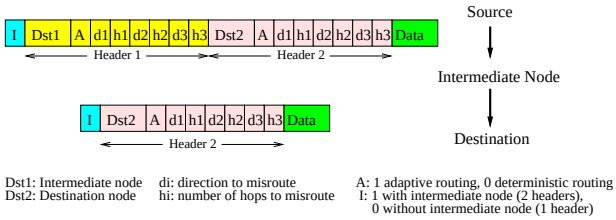

FIG. 2.1. *Packet info for the fault-tolerant methodology.*

In this paper we present an integration proposal, combining the FIR routing strategy [6] and the aforementioned fault-tolerant mechanism [5] to provide a fault-tolerant and memory-efficient routing strategy for direct networks (meshes and tori). As a result, we will provide a distributed, fault-tolerant and scalable routing strategy for general purpose switches. The rest of the paper is organized as follows. To make the paper self-contained, section 2 briefly presents the fault-tolerant methodology and section 3 describes the FIR scheme. Section 4 presents Fault-Tolerant FIR, the integration proposal. Section 5 evaluates it and finally, some conclusions are drawn.

2. Fault-Tolerant Methodology. The methodology provides fault-tolerance both in meshes and tori. It assumes a static fault model. The methodology is focused only at the computation of the new routing info[1]. The initial (i. e., without faults) routing algorithm routes packets by using Duato's fully adaptive routing with at least two virtual channels (at least one adaptive and one escape) per physical channel. The adaptive channel(s) enables routing through any minimal path. The escape channel guarantees deadlock freedom based on the bubble flow control mechanism. The methodology provides a fault-free path for each source-destination pair. To do that in the presence of faults, it uses intermediate nodes for routing between some source-destination pairs. Packets are first sent from the source node (S) to an intermediate node (I), and later, from this node to the final destination node (D)[2]. In both subpaths, minimal adaptive routing is used. If possible, the I node is selected inside the minimal adaptive cube defined by S and D. In this way, both subcubes defined by S and I, and by I and D, are inside this cube, but they are smaller and in this way the failure is avoided. In both subcubes (S-I and I-D) packets can be adaptively routed. In order to avoid deadlocks between both subpaths at the I node, we propose the use of two different escape channels. One of them will be used as escape channel for the S-I subpath and the second one for the I-D subpath. With an I node, the methodology is the 1-fault tolerant. In order to tolerate more than one failure we use two additional mechanisms: misrouting and switching off adaptive routing. Misrouting forces routing packets several hops along different directions (up to three directions in our methodology). Once misrouting is consumed, then minimal routing (adaptive or deterministic if adaptive routing is switched off) is applied. To guarantee deadlock-freedom, misrouting must use the directions according to the order established by the deterministic routing. The methodology uses the $X+Y+Z+X-Y-Z-$ direction-order routing, which is deadlock-free and provides routing flexibility since it allows routing packets in both directions of the same dimension providing non-minimal paths.

A packet routed through an I node requires two subheaders (see Figure 2.1). The first one is used in the subpath towards the I node, and the second one in the one towards the final destination. At the I node, the first subheader is removed. Packet subheaders also include control info about misrouting (direction and hops, up to three misroutings are allowed) and switching off adaptive routing (one bit). The routing info must also be stored at each source node, but not at the switches. For every destination, the possible I node (if required) and info about misrouting and switching off adaptive routing must be stored. The amount of required memory is low.

3. FIR: Flexible Interval Routing. Interval routing (IR) was proposed in [7]. IR groups those destination nodes reachable from the same output port into an interval. The intervals corresponding to different output ports of a switch are non-overlapping. Each packet is forwarded through the output port whose interval includes the destination of the packet. It is sufficient to store the bounds of each interval and to perform a paralell comparision to implement it, thus, memory space required is $2 \times d \times \log(N)$ bits per switch[3]. In a

[1] Detection of faults, checkpointing, and distribution of routing info is out of the scope of the methodology.
[2] Packets are not ejected from the network at the I node.
[3] d being the number of switch ports or switch degree and N the network size.

previous paper[6], we proposed an extension of the IR scheme, the Flexible Interval Routing (FIR), which can implement deterministic and adaptive routing on meshes and tori. Each output port has an associated interval, indicated by two registers, First Interval (FI) and Last Interval (LI). But, in order to add flexibility, we associate additional registers to the output ports. In regular topologies, each node is identified by its coordinates in each network dimension, to check if a given dimension can be used for routing, the coordinates of the current and destination nodes are compared. So in FIR, with each output port is associated a Mask Register (MR). This register has the size of n bits[4] and selects the bits of the packet destination address corresponding to the dimension of the output port. These bits are set to 1 in the MR and are the ones that will be compared with FI and LI. This is showun in Figure 3.1 taking into account cyclic intervals. If the masked destination address is inside the interval, then the hardware returns 1. This operation is done in parallel in all the output ports and the results of the comparisons for all the output ports can be stored in the Allowed Register (AR), unique for the switch, with a size of d bits.

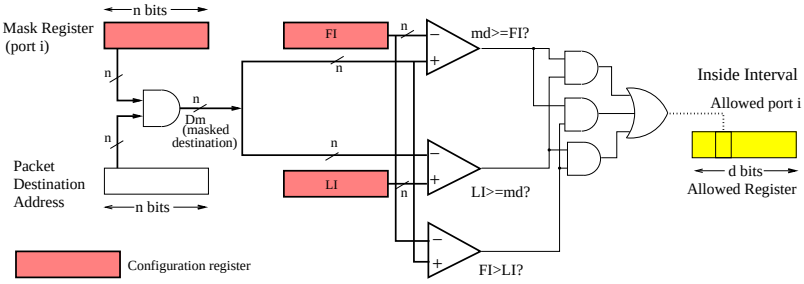

FIG. 3.1. *Hardware to generate the Allowed bit for each output port.*

Unlike IR, our proposal permits that the different intervals associated to the different output ports of a given switch overlap and, therefore, more than one output port can be allowed[5]. However, to guarantee deadlock freedom, some routing restrictions must usually be applied. In regular topologies, this is usually ensured by traversing network dimensions following some order. These routing restrictions are implemented in FIR by means of an additional register associated to each output port, the Routing Restrictions Register (RRR). It establishes, for each output port, which other output ports of the switch should be chosen prior to this one if they are allowed. This register has one bit per each output port. In a given output port i, the j bit in its RRR indicates if the output port j has highest preference (bit set to 1) or not (0) than output port i. Thus, the routing decision for a given output port i is obtained taking into account the Allowed bits of the other output ports and the bits in its RRR. If there is other output port with highest preference (bit set to 1 in the RRR) that also has its Allowed bit set to 1, this output port will not be returned to route the packet. Figure 3.2 presents the required hardware for output port i. This is done in paralell in all the output ports of the switch. The results can be stored in a Routing Register (RR), unique per switch, with a size of d bits. The RR indicates the result of the routing function. In case of adaptive routing, it may contain more than one 1.

If virtual channel multiplexing [2] is used, the configuration registers (LI, FI, MR and RRR) will be associated to virtual channels. Moreover, the RRR will have one bit per virtual channel and it will define the preferences among virtual channels. This is also the case for the registers associated to the whole switch (the Allowed and Routing Registers), that will represent virtual channels. Therefore, FIR requires, at each switch, $d \times v$ FI, LI, MR and RRR configuration registers, and one RR and AR registers of size $d \times v$, where v is the number of virtual channels per output port. The configuration of the configuration FIR registers establishes the routing algorithm. In [6] we present how to set them for the most popular routing algorithms used in meshes and tori networks. Following, we present some illustrative examples for this paper.

3.1. Illustrative Examples. To begin, a simple example explaining the register configuration for a 16-node 2-D mesh (4×4) and deterministic routing is presented. The destination addresses use 4 bits, and also the LI, FI and MR. Assuming no virtual channel multiplexing, the RRR requires one bit per each output port, so for a 2-D mesh, 4 bits are needed. Figure 3.3.(a) shows the configuration of the FIR registers when XY

[4] $n = \log(N)$, N being the network size
[5] This can be used to provide adaptive routing.

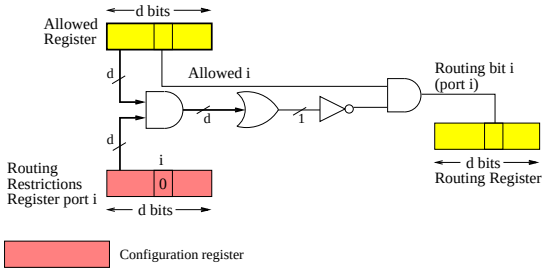

FIG. 3.2. *Routing bit for port i.*

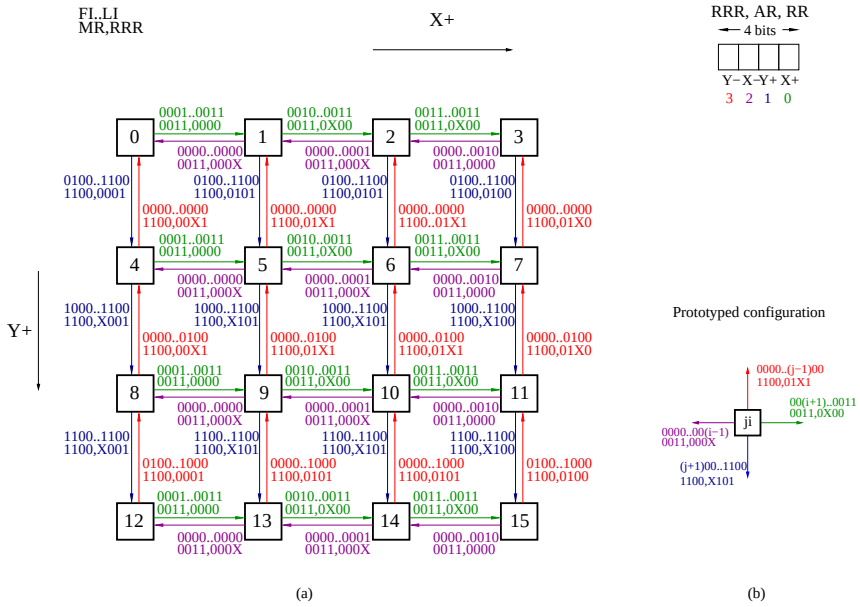

FIG. 3.3. *FIR registers configuration in a 2-D mesh with XY routing.*

routing is used. Figure 3.3.(b) shows the prototyped configuration for a central switch ji, i being the two least significant bits of the switch address and j the two most significant bits. The MR chooses the dimension bits in the destination address. This is, the two most significant bits of the destination address for the Y output ports, and the two least significant bits for the X links. Finally, the RRRs implement the XY routing. To do that, the RRR in the Y output port has the bits corresponding to the X output ports set to 1 to give preference to these output ports.

There are mainly two alternatives to prevent deadlocks in the rings of each dimension in torus networks. An alternative is the use of two virtual channels[6]. Another alternative that does not require the use of virtual channels is the bubble flow control [1]. With the bubble mechanism, when a packet is injected into the network or moves from one dimension to another dimension, two free buffers are necesary to guarantee deadlock freedom. In our proposal, an additional register associated with each output port is required to support the bubble mechanism (the Bubble Register -BR-), with one bit per input port (including the injection ports). The input ports that only need one free buffer to send through the current output port will have its associated bit set to 1. If two buffers are required by the input port in the output port, then the corresponding bit must be set to 0. Figure 3.4 shows the hardware that implements the bubble condition in the routing decision. So, the Routing bit corresponding to output port i will be obtained from the AR, its RRR, the bit associated to the input port in its BR and the space available at the output port.

FIR can implement adaptive routing. In this case, more than one bit in the RR can be set to 1. We consider fully adaptive routing based on Duato's protocol [4]. In this routing algorithm, each physical link is

[6]See [6] for a detailed FIR registers configuration.

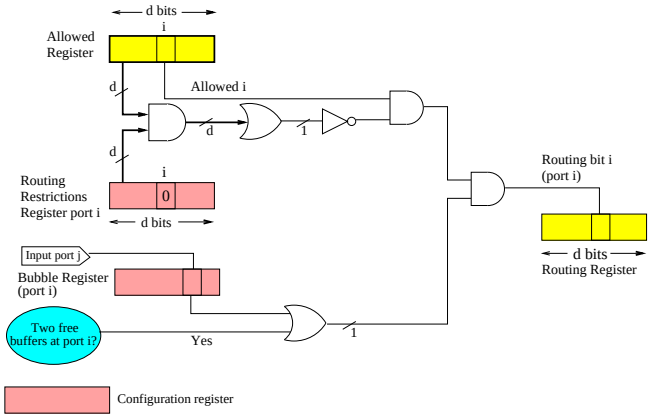

FIG. 3.4. *Bubble condition hardware in output port i.*

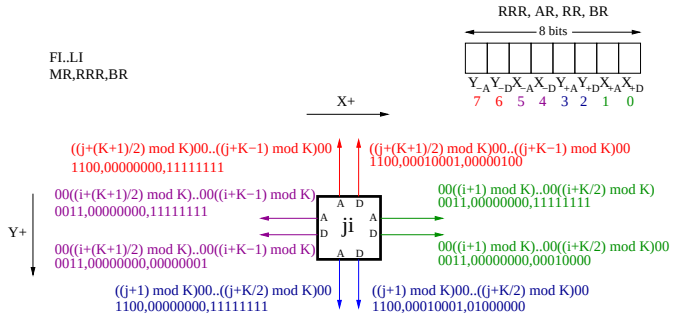

FIG. 3.5. *LI, FI, MR, RRR and BR for a* 4×4 *Torus with adaptive routing and the bubble flow control mechanism.*

split into several virtual channels, the adaptive and the escape ones. Figure 3.5 shows the prototyped register configuration for a 4×4 torus using a dimension order routing as deterministic subfunction in the escape channels. The escape channels must meet the bubble condition, so the BR must be configured. RRRs associated to the adaptive channels do not establish any order among VCs. Thefore, the routing function returns the adaptive channels that provide minimal routing and the deterministic channel returned by the deterministic subfunction.

4. FT-FIR: Fault-Tolerant FIR. In this section, we show how FIR can provide fault-tolerance using the above described fault-tolerant methodology at the expense of a few extra amount of memory and hardware. As commented before, the fault-tolerant methodology requires at least three virtual channels per physical channel. Two of them are used as escape channels with the bubble flow control, E1 and E0. E0 is used by the packets in the S-I phase, and E1 is used by the packets in the I-D phase. At least an additional adaptive channel is required to provide adaptive routing. Figure 4.1 shows the FIR registers configuration in a switch of a 2-D torus network. RRRs establish the deterministic routing in the escape channels ordering the different directions. $X+Y+X-Y-$ routing is used in order to be able to misroute packets. $X+$ direction is given the highest preference by setting the bits corresponding to the $X+$ escape channels set to 1 in the RRRs corresponding to the other directions. $Y-$ direction has the lowest preference, by setting the bits corresponding to the escape channels of the other directions set to 1 in its RRR. The new switch hardware will do the selection of the proper escape channel considering the I bit in the packet header as shown below. Only the escape channel corresponding to the current routing phase will be allowed. The RRRs associated with the adaptive channels do not establish any preference among the VCs. This means that the routing function returns all the adaptive channels that provide minimal routing and the corresponding deterministic channel following the deterministic routing subfunction. The bubble condition in the escape channels is implemented by the BR[7].

We have presented the FIR register configuration, now we explain how the FIR switch hardware must be modified to provide fault-tolerance. As commented above, packet subheaders provides information about

[7]The bits corresponding to the injection channels are not shown. They are set to 0, since two buffers are required.

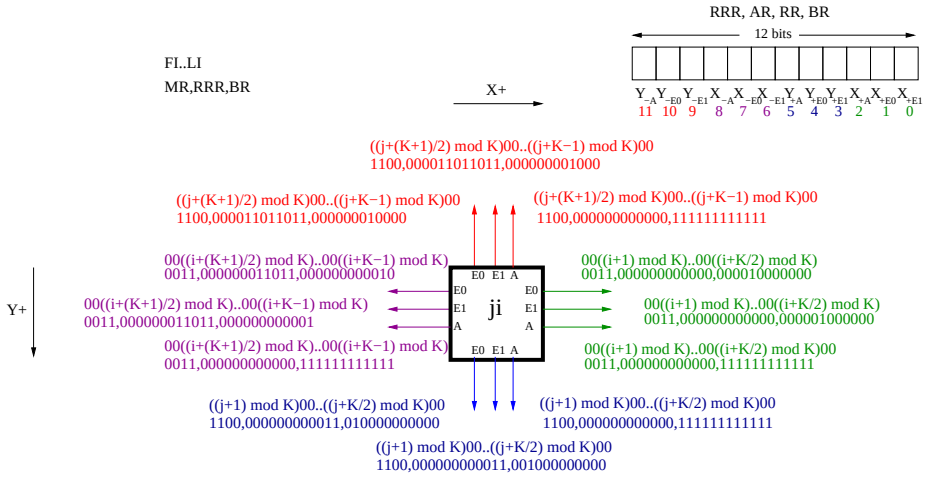

FIG. 4.1. *FI, LI, MR, RRR and BR configuration for a* 4×4 *torus and two escape channels.*

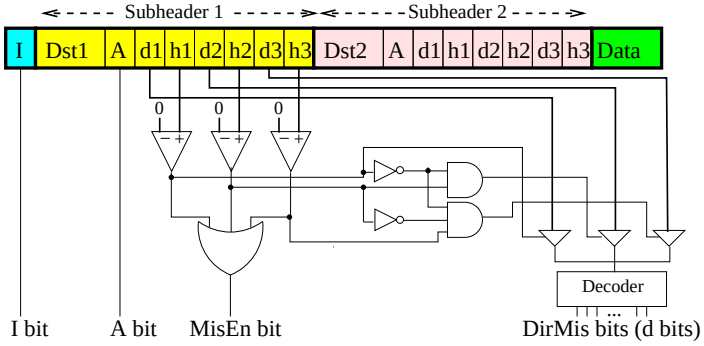

FIG. 4.2. *Hardware to obtain the fault-tolerance control bits from the packet header.*

intermediate nodes, disabling adaptivity and misrouting. From the packet header, in addition to the destination identifier, four control bits are generated (see Figure 4.2). These bits are used by the switch hardware associated with fault-tolerance. The I and A bits are obtained from the same bits of the packet subheaders. The $MisEn$ bit is set to one if the packet must be misrouted. This is true if any of the hops fields is different of 0 in the subheader corresponding to the current routing phase. $MisEn$ can be obtained by using three comparators, one per hops field in the packet subheader. The $DirMis$ field indicates the direction to misroute the packet. It has a number of bits equal to the switch degree and contains a "1" in the position that corresponds to the current direction (port) that must be used to misroute the packet. A decoder can be used to obtain the $DirMis$ bits. Its input is the first direction to misroute, which is obtained by using a multiplexer (implemented by the tristate gates and their control logic in the Figure). Figure 4.2 shows the processing of the first subheader. When the I is reached, this subheader is deleted and then the second subheader is processed as the first one.

Packets must select the proper escape channel considering the I bit in the packet header. To do that, we propose the extension shown in Figure 4.3.(a) for Figure 3.1. In this figure the Allowed bit is obtained for the escape channels, E0 and E1, considering the I bit. This bit indicates the routing phase in which the packet is. E0 is allowed only if packet is traveling to the I node, that is if I is 1. E1 only if packet is traveling to its final destination, that is if I is 0. The H bit shown in Figure 4.3 is a new configuration bit associated to each virtual channel. It indicates which escape channel corresponds to the current virtual channel. H is set to 0 in E0 escape channels and to 1 in E1 escape channels. On the contrary, the adaptive channels are not affected by the I bit.

Other mechanism consists in disabling adaptivity. If the A bit is set to 0 in any of the packet subheaders, then the packet must be routed in a deterministic way in that phase and therefore it must use the deterministic virtual channel corresponding to the proper escape channel. So, as shown in Figure 4.3.(b), if the A bit is

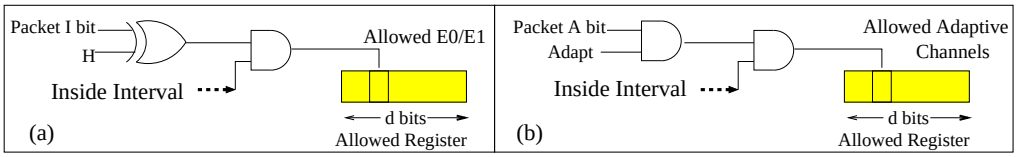

FIG. 4.3. *(a). Selection of the adequate escape channel taking into account the I bit. (b). If A=0, then the adaptive channels are not allowed.*

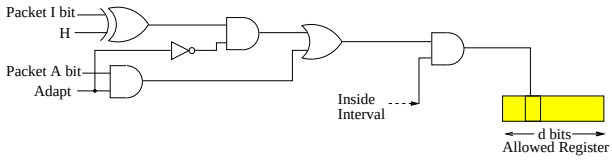

FIG. 4.4. *Combination of Figure 4.3.(a) and Figure 4.3.(b): Allowed bit taking into account the I and A bits of the packet header.*

set to 0, the adaptive channels are disabled. *Adapt* is another configuration bit associated with each virtual channel. It indicates whether the current virtual channel is adaptive (set to 1 in the adaptive virtual channels, and to 0 in the escape ones). The hardware shown in Figure 4.3.(a) is applied in the escape virtual channels. Figure 4.4 extends Figure 3.1 merging Figure 4.3.(a) and Figure 4.3.(b) to obtain the hardware for a generic virtual channel.

Therefore, two new switch configuration bits, H and *Adapt*, are required per virtual channel in order to configure the different virtual channels either as escape (E0 or E1) or as adaptive. To do that, we propose adding two registers to each switch with a number of bits equal to the number of virtual channels of the switch, that is, $d \times v$ bits. In the first register, the Adaptive Register, the bits corresponding to the adaptive channels are set to 1. In the second register, the Escape Register, the bits corresponding to escape channels E1 are set to 1.

Next, we consider the last mechanism of the fault-tolerant methodology: misrouting. When misrouting a packet, it is routed in a deterministic way, so the adaptive channels will be disabled, and the corresponding escape channel will be used instead. Figure 4.5.(b) obtains the Allowed bit for the adaptive channels when misrouting. The escape channels to use depends on the current routing phase (E0 or E1) and belongs to the first direction to misroute, even if the packet destination is not inside the interval bounds of that direction, since misrouting can provide non-minimal paths. As the packet travels along its path, the number of hops associated to that direction is decreased at the packet subheader, and when the number of hops is 0, then the next direction to misroute indicated in the packet subheader is followed. Figure 4.5.(a) shows how escape channels are allowed taking into account the $MisEn$ and $DirMis$ bits. Once the hops of all the directions to misroute at the current routing phase are consumed, the $MisEn$ bit is set to 0, and therefore packets are routed following minimal path, taking into account the FI..LI registers. Figure 4.6 combines Figure 4.5.(b) and Figure 4.5.(a).

Therefore we can say that FIR allows to implement the fault-tolerant methodology. To do that two registers (Adaptive and Escape Registers) are added to each switch with a number of bits equal to the number of virtual channels in the switch. The switch hardware is only a little more complex. The hardware shown in Figure 4.6 must be added to Figure 3.1 to obtain the Allowed bits in the new switch. Remember that in any routing strategy, some logic is also required to manage the three mechanisms the fault-tolerant methodology is composed of.

5. Evaluation of FTFIR. In this section, we evaluate the Fault Tolerant FIR strategy. We are interested in analyzing its fault-tolerance, its performance, the amount of memory required and the routing delay.

The fault-tolerant degree and performance of the proposal is the same obtained by the fault-tolerant methodology (evaluated in [5]) since the proposed FT-FIR scheme is a different hardware implementation. This methodology is 7-fault tolerant. However, the percentage of tolerated fault combinations is greater than 99,9% up to 10 failures. Concerning performance, in the presence of failures, the performance degradation is small. As an example, network throughput degrades less than 5.5% when injecting 6 random failures in a 512-node Torus.

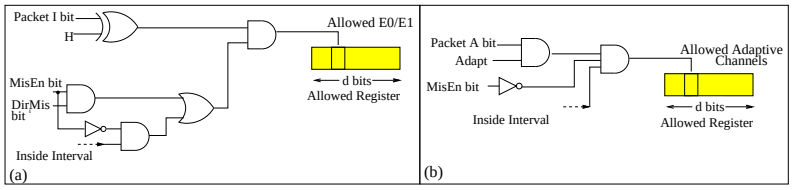

FIG. 4.5. *(a). When using misrouting, if the first direction indicated in the packet subheader corresponds to the direction of the current output channel, then the corresponding escape channel is allowed. (b). The adaptive channels are not allowed, If the packet must be misrouted in the current routing phase (MisEn=1).*

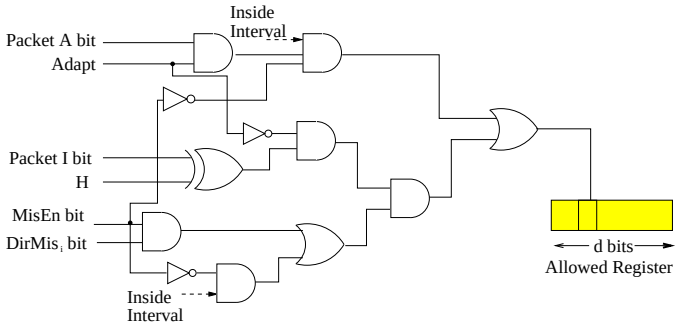

FIG. 4.6. *Combination of Figures 4.5.(a) and 4.5.(b): Allowed bit taking into all the fault-tolerant info contained in the packet header.*

Now we compare the amount of memory required at the switches by the FT-FIR strategy and an implementation based on forwarding tables[8]. Assume that we have a network composed of N nodes, build with switches with d ports. The links attached to each port may be split into up to v virtual channels. Finally, the routing algorithm offers a maximum of r routing options. The FT-FIR approach needs to associate five configuration registers with each virtual channel, three of them (FI, LI and MR) of size $\log(N)$ bits and two (RRR and BR) of size $d \times v$ bits. No matter if the routing is deterministic or adaptive. In addition, two configuration registers are associated to the whole switch, Adaptive and Escape Registers, of size $d \times v$ bits in order to provide fault-tolerance. Therefore, the total number of bits required to implement the FT-FIR strategy is $C_{\text{FTFIR}} = d \times v \times (3 \times \log(N) + 2 \times d \times\ v) + 2 \times d \times\ v$ bits. So, its cost remains being $O(\log(N))$ as in FIR. On the other hand, routing based on forwarding tables requires a table with as many entries as the number of nodes, and each entry must contain the port(s) returned by the routing function. Hence, the cost of this alternative is $C_{\text{FT}} = N \times \log(d \times v) \times r$ bits in each switch, which means that the cost is $O(N)$, and not scalable with the network size.

Routing delay of FIR is given by the time required to check if the destination address is inside the interval (all the ports make these comparisons concurrently) thus obtaining the Allowed bits plus the time to merge these bits with the routing restrictions (RRR). In the FT-FIR, this delay is slightly increased by the additional constraints of the fault-tolerant methodology. However, as part of this hardware may work in parallel with the interval comparison, the increase in routing delay will be small. On the other hand, some hardware would be required to support the fault-tolerance mechanism in a conventional routing scheme. This additional hardware will also increase the routing delay.

6. Conclusions. This paper proposes Fault-Tolerant Flexible Interval Routing (FT-FIR), a fault-tolerant adaptive routing strategy for commercial switches, . The strategy provides both, fault-tolerance and scalable routing for commercial switches in regular direct topologies. It is scalable because it requires relatively few amount of hardware and memory. It uses only 7 registers per each virtual channel to route packets, so the total requirements of memory is $O(\log(N))$. Moreover, it tolerates a relatively large number of faults while inflicting a minimal decrease of performance in the presence of faults.

[8]The fault-tolerant information is stored in the source nodes, but not at the switches.

REFERENCES

[1] C. Carrion, R. Beivide, J. A. Gregorio, and F. Vallejo. *A Flow Control Mechanism to Avoid Message Deadlock in K-ary N-Cube Networks*. Forfth International Conference on High Performance Computing, pp. 332-329, December 1997.

[2] W. J. Dally, Virtual-channel flow control, *IEEE Trans. on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194-205, March 1992.

[3] W. J. Dally and H. Aoki. *Deadlock-free adaptive routing in multicomputer networks using virtual channels*. IEEE Trans. on Parallel and Distributed Systems, vol 4, no 4. pp 466-475, April 1993.

[4] J. Duato, S. Yalamanchili and L. Ni. Interconnection Networks. An Engineering Approach. *Morgan Kaufmann*, 2004.

[5] M. E. Gómez, J. Duato, J. Flich, P. López, A. Robles, N. A. Nordbotten, T. Skeie, and O. Lysne. *A New Adaptive Fault-Tolerant Routing Methodology for Direct Networks*, in Proc. International Conference on High Performance Computing, 2004.

[6] M. E. Gómez, P. López, J. Duato. *A Memory-Effective Routing Strategy for regular Interconnection Networks*, to appear in Proc. Int. Parallel and Distributed Processing Symposium, 2005. Best Paper Award in the Architecture Track.

[7] N. Santoro and R. Khatib. Routing without routing tables. *Tech. report SCS-TR-6, School of Computer Science*, Carleton University, 1982. Also as: Labelling and Implicit Routing in Networks, Computer Journal 28(1), 1985, pp. 5-8.

Edited by: M. Tudruj, R. Olejnik.
Received: February 24, 2006.
Accepted: July 30, 2006.