A Reconfigurable Fault-tolerant Deflection Routing Algorithm Based on Reinforcement Learning for Network-on-Chip

Chaochao Feng[1,2], Zhonghai Lu[2], Axel Jantsch[2], Jinwen Li[1], Minxuan Zhang[1]
[1]School of Computer, National University of Defense Technology, Changsha, P.R. China
[2]Royal Institute of Technology, Stockholm, Sweden
[1]{fengchaochao, lijinwen, mxzhang}@nudt.edu.cn [2]{cfeng, zhonghai, axel}@kth.se

ABSTRACT

We propose a reconfigurable fault-tolerant deflection routing algorithm (FTDR) based on reinforcement learning for NoC. The algorithm reconfigures the routing table through a kind of reinforcement learning—Q-learning using 2-hop fault information. It is topology-agnostic and insensitive to the shape of the fault region. In order to reduce the routing table size, we also propose a hierarchical Q-learning based deflection routing algorithm (FTDR-H) with area reduction up to 27% for a switch in an 8×8 mesh compared to the original FTDR. Experimental results show that in the presence of faults, FTDR and FTDR-H are better than other fault-tolerant deflection routing algorithms and a turn model based fault-tolerant routing algorithm.

Categories and Subject Descriptors

B.8.1 [**Performance and Reliability**]: Reliability, Testing, and Fault-Tolerance

General Terms

Reliability

Keywords

fault-tolerant, deflection routing, reinforcement learning, NoC

1. INTRODUCTION

Network-on-Chip (NoC) has already become a promising solution for integrating a large number of cores on a chip to achieve high performance. However, as the CMOS technology scales down to the nanometer domain, smaller feature size, lower voltages and higher frequencies increase the number of occurrence of intermittent and transient faults, besides manufacturing defects and wear out effects which lead to permanent faults are also inevitable [1]. The inherent structure redundancy of NoC provides the potential to design a fault-tolerant routing algorithm to enhance the reliability.

NoCArc '10, December 4, 2010, Atlanta, Georgia, USA

Recently, bufferless routing, also called deflection routing, has been widely used for reducing the hardware overhead and power consumption of NoC [2, 3, 4]. In deflection routing, an incoming packet is always routed to a free output port even though it is far away from the destination. Because of its non-minimal routing characteristic, deflection routing can be easily modified to achieve fault-tolerance. This paper proposes a reconfigurable fault-tolerant deflection routing algorithm (FTDR) based on a kind of reinforcement learning—Q-learning. It is a table-based routing algorithm, which reconfigures the routing table through Q-learning and uses 2-hop fault information to make efficient routing decision to avoid faults. In order to reduce the routing table size, we also propose a hierarchical Q-learning based deflection routing algorithm (FTDR-H) with area reduction up to 27% for a switch in an 8×8 mesh compared to the original FTDR. Simulation results under both synthetic and application traffic workloads demonstrate that in the presence of faults, FTDR and FTDR-H perform better than other fault-tolerant deflection routing algorithms [5, 6] and a turn model based resilient routing algorithm [7].

The rest of paper is organized as follows: Related work is reviewed in Section 2. Section 3 describes the NoC architecture and fault model. The detailed routing algorithm and implementation are proposed in section 4. In section 5, simulation experimental results are analyzed, followed by the conclusion and future work in section 6.

2. RELATED WORK AND MOTIVATION

Depending on the shape of the fault region, fault-tolerant routing algorithms can be categorized into two classes: one can handle regular fault regions (convex and concave shapes) and the other, which is also known as topology-agnostic, can handle irregular fault regions. The regular nature of a faulty block facilitates to route packets around the fault region but it is not practical. For general interconnection network, a fault-tolerant adaptive routing algorithm using safety level concept [8] is used to find a minimal path in a mesh with rectangular faulty blocks (convex fault regions) and a software-based fault-tolerant oblivious routing algorithm [9] has been proposed to handle both convex and concave fault regions.

There are also some fault-tolerant routing algorithms for NoC to tolerate regular fault regions. Routing packets through a cycle free contour surrounding a faulty switch by a reconfigurable routing algorithm for a fault-tolerant 2D mesh NoC has been investigated in [10]. But it can only be used in one faulty switch or region topology. In [11], a deadlock free routing algorithm is presented to handle irregular mesh topology with rectangular regions. This algorithm uses the concept faulty-rings and faulty-chains to isolate the faulty

nodes from the rest of the network. A resilient routing algorithm for fault-tolerant NoC based on turn model is described in [7]. The switch can be reconfigured around faulty components while maintaining correct operation without using virtual channels. For deflection routing, a fault adaptive routing algorithm, which makes routing decision based on a cost function, has been proposed in [5]. Because it makes routing decision only based on the fault information of the current switch, the hop count field of the packet can easily overflow even in some simple fault patterns. A Fault-on-Neighbor (FoN) aware deflection switch [6], which can tolerate convex and concave fault regions (which have at most one concave point in sequence) without deadlock and livelock, makes routing decision based on 2-hop fault information and fault region shape without using routing table.

Few works focus on handling network with irregular fault regions because it needs more fault information to make correct routing decision. A Dependable routing for parallel computer systems is described in [12]. It can be applied to arbitrary topologies and tolerate any number of failures regardless of their spatial and temporal distributions. For NoC, a region-based routing has been proposed to handle irregular networks [13]. This algorithm groups destinations into regions to make routing decision. Its main overhead is the execution time of the region computation process.

Q-routing based on Q-learning is first proposed to handle rectangular regions in dynamic Network-on-Chip [14]. The proposed FTDR algorithm inspired by Q-learning is to overcome the constraint of the regular fault region. It is insensitive to the shape of the fault region and guarantees communication between any pair of switches as long as a path exists between them.

3. NOC ARCHITECTURE AND FAULT MODEL

3.1 NoC Architecture

The NoC architecture is based on Nostrum NoC [15], which is a 2D mesh topology. Each process element (PE) is attached to a switch (S), as shown in Fig. 1. The difference from the ordinary 2D mesh is that the boundary output is connected to the input of the same switch, which can be used as a packet buffer. All incoming packets are prioritized based on its hop counts which record the number of hops the packet has been routed. The switch makes routing decision for each arriving packet from the highest priority to the lowest. If a desired output port has already been occupied by a higher priority packet, a free port with the smallest stress value, which is the traffic load of neighbor switches in last 4 cycles, will be chosen, which means the packet has to be deflected.

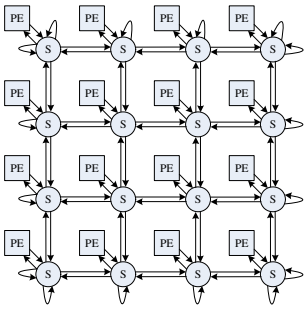

Figure 1: NoC architecture.

The packet format which is compatible for a multi-core NoC platform [16] is shown in Fig. 2. A packet, which is 114 bits, contains a 34 bits head and an 80 bits payload. A valid bit (V) is used to mark a packet valid or not. Relative addressing is used for the source and destination address fields (SA and DA) which are 12 bits respectively. The HC field (9 bits) records the number of hops the packet has been routed.

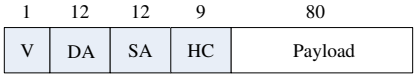

Figure 2: Packet format.

3.2 Fault Model

In this paper, faults can be categorized as faulty links and switches which are permanent faults (wear out faults). For deflection switch, the number of input ports should be equal to the number of output ports. So link failures are assumed to be bidirectional. For the focus on the routing algorithm, we also assumed that there exists a fault diagnosis mechanism to detect faults. How to detect faults is beyond the scope of this paper. In each switch, a 4-bit fault vector is used to represent the fault status of its four links. A switch fault is modeled by making all four links attached on it faulty. Packets are only destined to fault-free switches. The fault region can be any shape as long as it does not disconnect the network.

A 2-hop fault information transmission mechanism [6] is used to reduce the average hop counts. In the 2-hop fault information transmission mechanism, four additional signals (*fault_from[d] (1 bit), fault_to[d] (1 bit), FoN_from[d] (3 bits), FoN_to[d] (3 bits)*), which are 8 bits in total for each direction of a switch, are used to transmit fault information, as shown in Fig. 3. Each switch is not only responsible for transmitting its own link status to four neighbors but also collecting the link status from its three neighbors and transmitting to the fourth neighbor. For example, switch A can get the status of 16 links within 2 hops, as shown in Fig. 3. The signal *FoN_to[d]* collected by the current switch is a 3-bit vector to denote link status along the other three directions except *d* and is transmitted to the neighbor along *d*.

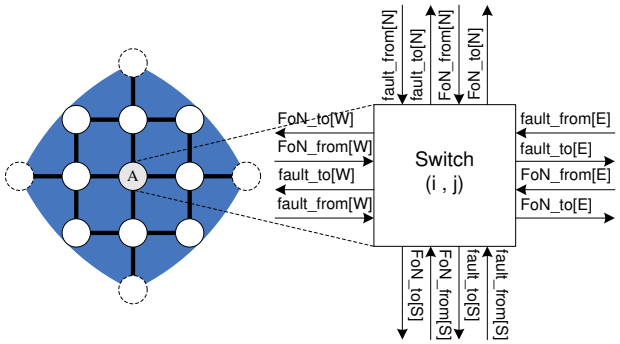

Figure 3: Fault information transmission mechanism.

4. RECONFIGURABLE FAULT-TOLERANT DEFLECTION ROUTING ALGORITHM

4.1 Basic Idea of Q-routing

Q-routing is an adaptive routing algorithm based on a variant of the reinforcement learning—Q-learning, which makes routing decision using only local information without having to know the network topology in advance [17]. In Q-routing, a switch x sends a packet to a switch d through

a neighbor switch y with the lowest estimated delivery time $Q^x(d, y)$ which is stored in a two dimensional table. After the packet is sent to y, x receives the minimum estimated delivery time $\min_z Q^y_{t-1}(d, z)$ from y. Then the estimated delivery time $Q^x(d, y)$ can be updated with formula (1).

$$Q^x_t(d, y) = (1 - \alpha)Q^x_{t-1}(d, y) + \alpha(b^x_t + \min_z Q^y_{t-1}(d, z)) \quad (1)$$

where b^x_t is the time the packet spent in the buffer of switch x and α is the learning rate which determines to what extent the newly acquired information will override the old information. A factor of 0 will make no learning, while a factor of 1 would consider only the most recent information.

4.2 FTDR Algorithm

For deflection routing, we use the number of hops to destination instead of the estimated delivery time to build the routing table Q^x in each switch. There are $n \times m$ entries in the routing table, where n is the number of switches in the network and m is the number of neighbor switches. For 2D mesh, m is 4. For a given topology, the initial routing table is fixed. For example, Table 1 shows a routing table of switch 5 in a 3×3 mesh. In order to achieve fault-tolerance in deflection routing, the formula (1) can be modified to formula (2) to update the routing table. The packet does not have to wait in a switch in deflection routing, so b^x_t is zero. In order to reduce the learning period, we set α to 1. The switch chooses a direction with the smallest number of hops to destination to send a packet. In the case of several directions with equal number of hops to destination, the switch chooses one of them with the smallest stress value. When a switch x sends a packet to d through y, y returns the minimum number of hops to d back to x. Then x updates the corresponding entry with 1 *hop* plus the minimum number of hops to d from y ($\min_z Q^y_{t-1}(d, z)$).

$$Q^x_t(d, y) = 1 + \min_z Q^y_{t-1}(d, z) \quad (2)$$

Table 1: Routing table of switch 5 in a 3×3 mesh

	North	East	South	West
Number of hops to S1	2	4	4	2
Number of hops to S2	1	3	3	3
Number of hops to S3	2	2	4	4
Number of hops to S4	3	3	3	1
Number of hops to S5	0	0	0	0
Number of hops to S6	3	1	3	3
Number of hops to S7	4	4	2	2
Number of hops to S8	3	3	1	3
Number of hops to S9	4	2	2	4

The routing table update function is shown in Fig. 4. If there is no fault in the network, the routing table can not be updated. If one link of the switch is broken, all table entries corresponding to this direction are set to ∞ (step 2-4). After a learning period, the table entries will converge to a fixed value which denotes the minimum number of hops from each port to each destination. Additionally, we use the 2-hop fault information to reduce the average hop counts. If a switch detects that one of its neighbors y along direction d has only one link not faulty based on the 2-hop fault information, the table entries from d to all destinations except y are set to ∞ (step 5-7). If a 2-hop link is faulty ($FoN\_from(d)(j) = 1, j \in \{North, East, South, West\}$), the table entries from d to all destinations along j are updated with the previous Q-value plus 2 (step 8-11).

The pseudo code of the algorithm is shown in Fig. 5. There are at most four packets reaching a switch at the same time. The switch makes routing decision from the highest priority packet to the lowest. The switch first calculates the destination ID of the packet and looks up the routing table to check if the packet has reached the destination (step 1-4). If the packet has not reached the destination, the switch looks up the productive direction(s) with the minimum number of hops to destination from the routing table and then chooses a free productive port with the smallest stress value to route the packet (step 7-9). If there is no free productive port, the switch chooses a free port with the smallest stress value to route the packet which can balance the network traffic loads (step 10,11). The switch also sends the minimum number of hops to destination back to the neighbor switch, which sends the packet, to update its routing table (step 6).

Routing table update function

RoutingTableUpdate(*dest_ID, Q_value_from, FoN_from*)
1: ***for*** *dir* ∈ {*North*, *East*, *South*, *West*}
2: ***if*** *link*(*dir*) is faulty ***then***
3: ***for*** *i* = 1 to *N* ***and*** *i* ≠ *Swich_ID*
4: *table_entry*(*i*)(*dir*) ←∞
5: ***if*** *Neighbor*(*dir*) has only one link not faulty ***then***
6: ***for*** *i* = 1 to *N* ***and*** (*i* ≠ *Swich_ID* ***and*** *i* ≠ *Neighbor_ID*(*dir*)) then
7: *table_entry*(*i*)(*dir*) ←∞
8: ***for*** *j* ∈ {*North*, *East*, *South*, *West*}
9: ***if*** *FoN_from(dir)(j)*=1 ***then***
10: ***for*** *n* ∈ {all switches along *j* through *Neighbor*(*dir*)}
11: *table_entry*(*n*)(*dir*)=*table_entry*(*n*)(*dir*) + 2;
12: ***if*** *Q_value_from*(*dir*, *dest_ID*) ≠ 0 ***and*** *dest_ID* ≠ *Swich_ID* ***then***
13: *table_entry*(*dest_ID*)(*dir*) ←*Q_value_from*(*dir*, *dest_ID*)

Figure 4: Routing table update function.

FTDR Algorithm

For each input packet (from the highest priority to the lowest priority)
1: *dest_ID*← get_dest_ID(d_x, d_y)
2: *Hops_to_Dest*←table_lookup(*dest_ID*)
3: ***if*** *Hops_to_Dest* = (0,0,0,0) ***then***
4: Route packet to local port
5: ***else***
6: *Q_value_to*(*input_Dir, dest_ID*) ←1 + min(*Hops_to_Dest*) //send Q-value
7: {$d_{productive}$}←get_prefer_Dir(*Hops_to_Dest*)
8: ***if*** there are free ports in {$d_{productive}$} ***then***
9: Choose a free productive port with the smallest stress value to route the packet
10: ***else*** //all productive ports are not free
11: Choose a free port with the smallest stress value to route the packet

Figure 5: FTDR algorithm.

4.3 FTDR-H Algorithm

The routing table is the main overhead of FTDR algorithm. In order to reduce the table size, we also propose a hierarchical Q-learning based deflection routing algorithm (FTDR-H). The $n \times n$ mesh can be divided into several subregions with equal size. Each switch contains a local and a region routing table. When a packet reaches a switch, the switch first checks whether the destination is in the same region as the current switch or not. If it is, the switch makes routing decision based on the local routing table. If the destination is not in the same region, the switch makes routing decision based on the region routing table. The local and region routing tables are also updated by formula (2). However, it is assumed that each region is not disconnected by faults for FTDR-H algorithm. Here, an 8×8 mesh, which is divided into four 4×4 regions, is used as an example. Each switch contains a local routing table of 16×4 entries and a region routing table of 4×4 entries. So the total size of the routing table is reduced from 64×4 to 20×4 entries.

4.4 Deadlock and Livelock Avoidance

Deflection routing is inherently deadlock free since packets never have to wait in a switch. However, it must avoid livelock by limiting misrouting. FTDR algorithm makes routing decision based on the packet priority and routing table. First, the algorithm always gives the highest priority to the oldest packet. Given a network size and different fault patterns, the length of the hop count field must be enough to guarantee the priority can not saturate. Second, it can be proved that the routing table entry will converge to the minimum hops to each destination from the following proof. Packet priority and converged routing table, which limit the number of misrouting, guarantee a packet can eventually advance towards its destination to achieve freedom from livelock.

Theorem: In FTDR algorithm, the routing table entry will converge to the minimum number of hops to destination in the presence of fault regions which do not disconnect the network.

Proof: Without loss of generality, assume x sends packets to d through y. The initial Q-value $Q_0^x(d, y)$ is the minimum number of hops from x to d through y.

In the case of no faults, according to the equation (2), the term $\min_z Q_{t-1}^y(d, z)$ equals to the minimum number of hops to d from y, so $1 + \min_z Q_{t-1}^y(d, z)$ still equals to the minimum number of hops from x to d through y ($Q_0^x(d, y)$). So the routing table entry will not be updated.

If there are faulty links on the shortest path from y to d, the routing table entry of the switches with faulty links will be updated immediately. It will take some time to propagate the updated routing table entry to reconfigure the routing table entry $Q_{t-1}^y(d, z)$ of y. After that, $\min_z Q_{t-1}^y(d, z)$ is still the minimum number of hops to d from y. The corresponding table entry $Q^x(d, y)$ of x will be reconfigured to minimum number of hops to d after $Q_{t-1}^y(d, z)$ converges.

In summary, after a learning period, the routing table entry will converge finally. □

4.5 Hardware Implementation

The structure of FTDR switch is shown in Fig. 6. Each switch contains an $n \times 4$ routing table. Each entry of the routing table contains 6 bits, so the size of the whole table is $n \times 24$ bits. An entry of all '1' denotes ∞. The stress value, fault information and Q-value are transmitted between two switches. Routing controller makes routing decision based on the above three kinds of information. For FTDR-H switch, the routing table includes local and region routing tables.

To make a comparison, we also implement the FoN [6] and cost-based deflection switches [5]. All switches are synthesized using TSMC 65nm standard-cell library by Synopsys Design Compiler. All switches are implemented in an 8×8 mesh and optimized for speed. The results of the performance, area and power consumption for the four switches are shown in Table 2. The FoN switch can achieve highest frequency and smallest area and power consumption because it is not table-based. But it can only handle regular fault regions. Although the cost-based switch does not use routing table, it has high hardware overhead because it has to find the best permutation among all permutations of input and output ports based on a cost function. Compared to the original FTDR, FTDR-H can reduce area up to 27%.

5. EXPERIMENTAL STUDIES

In this section, FTDR and FTDR-H algorithms are compared with the FoN and cost-based deflection routing algorithms and a turn model based resilient routing algorithm [7] in terms of throughput and hop count applying both synthetic and application workloads on an 8×8 2D mesh.

Table 2: Comparison of four deflection switches

	Frequency (MHz)	Area (μm^2)	Power per MHz (mW/MHz)
FoN	500	39076	0.01
Cost-based	166	82277	0.042
FTDR	400	101754	0.028
FTDR-H	400	74323	0.022

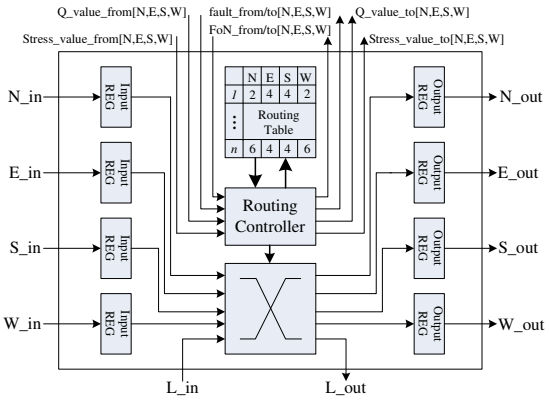

Figure 6: FTDR switch structure.

5.1 Simulation Platform

The switch is developed in VHDL and performed a cycle-accurate RTL-level simulation. For synthetic workload, a packet generator is attached to each switch and there is a FIFO in it to buffer the packets which can not be injected into the network because of no free output port. Six common synthetic traffic patterns (uniform random, transpose, bit complement, bit reverse, shuffle and tornado) are used. For application workload, we use a Distributed Shared Memory (DSM) based multi-core NoC architecture, in which a LEON3 processor and a local memory are attached to each switch through a Dual Microcoded Controller [16]. The fault pattern is randomly generated before the simulation.

5.2 Results with Synthetic Workload

First, we will show the advantage of the FTDR algorithm with 2-hop fault information against 1-hop information. Fig. 7 plots the average hop count vs. simulation time under uniform random traffic with a fault pattern of 10% link faults and the packet injection rate of 0.1 packets/cycle/node. It can be seen that at the beginning of the simulation the average hop count increases quickly. After an initial learning period which is about 350 cycles in this figure, the average hop count will slightly decrease and remain stable. The length of the learning period depends on the fault pattern. Using 2-hop fault information, some unnecessary misroutings can be avoided, so the average hop count is smaller than using only 1-hop fault information.

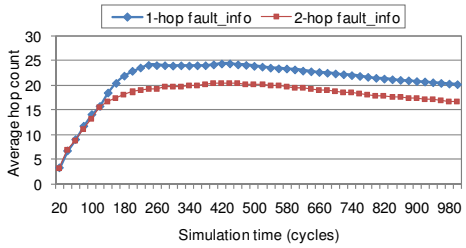

Figure 7: Comparison of the FTDR algorithm with 1-hop and 2-hop fault information.

Fig. 8 (a)-(f) show the throughput of the network with link faults from 0 to 30% achieved by four routing algorithms under six synthetic traffic patterns respectively. In the case of no faults, the throughput of the cost-based switch is slightly higher than other switches, because it can always find the best permutation for each packet in a congested network instead of unnecessary deflections. In the presence of link faults, the cost-based switch achieves the lowest throughput among the four switches, while FTDR and FTDR-H achieve almost the same throughput. The throughput of FTDR is 14% and 23% higher on average than FoN and cost-based deflection switch respectively. Based on the throughput data of the turn model based resilient routing algorithm in [7], we also compare it with FTDR and FTDR-H algorithms. For 8×8 2D mesh with 10%, 20% and 30% faulty links, the throughput of the turn model based resilient routing algorithm is 0.1, 0.08 and 0.06 packets/cycle/node under uniform traffic pattern respectively, while FTDR and FTDR-H can achieve 0.16, 0.13 and 0.1 packets/cycle/node respectively.

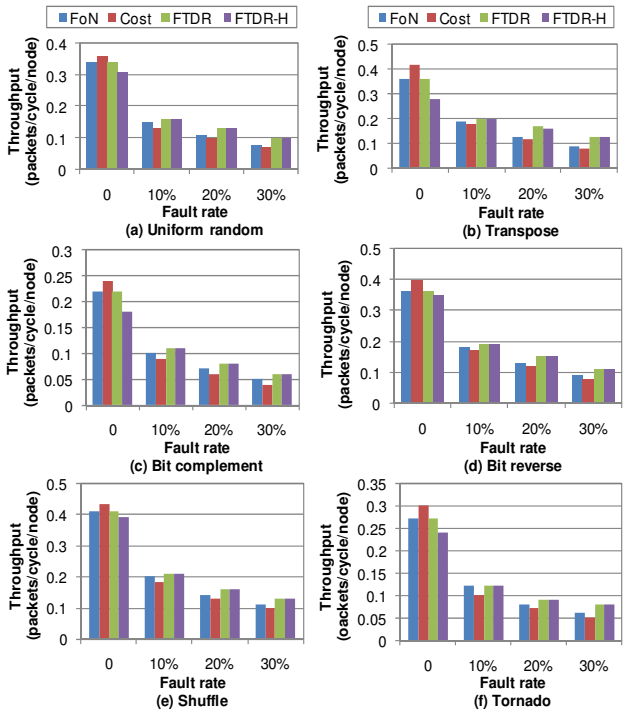

Figure 8: Throughput with various link fault rates under synthetic workloads.

Fig. 9 (a)-(f) present the average hop count of the four routing algorithms with various link faults under six synthetic traffic patterns respectively. The packet injection rate is 0.1 packets/cycle/node. FTDR-H can achieve 2.5x, 1.7x, 1.2x, 2x, 2.8x and 1.3x less hop counts on average than the cost-based algorithm for each traffic pattern respectively. The cost-based algorithm makes routing decision only based on local fault status, so even in some simple fault patterns the hop count field of the packet can easily overflow. FTDR and FTDR-H perform better than FoN because after a learning period the routing table will converge. For uniform random, bit reverse and shuffle traffic patterns, the average hop count of FTDR-H is 18%, 10%, 15% less than FTDR respectively. For the rest of patterns, FTDR-H performs similar as FTDR.

5.3 Results with Application Workload

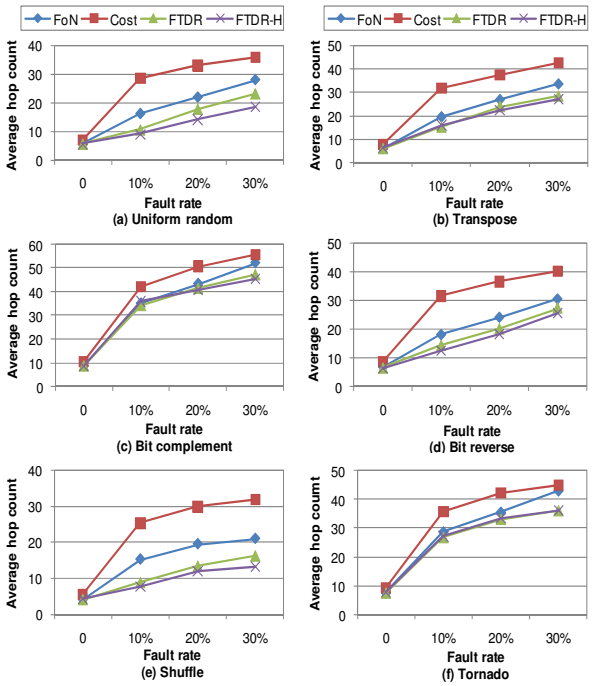

Figure 9: Average hop count with various link fault rates under synthetic workloads.

For application workloads, three applications, matrix multiplication, 1D radix-2 1024-point parallel DIF FFT and wavefront computation, are mapped manually on LEON3 processors. The matrix multiplication calculates the product of two matrices, A(128×1) and B(1×128), resulting in a C(128×128). Three matrices are distributed stored in the shared memory. The data of FFT are equally partitioned into 128 groups of 8 complex numbers each and stored in the local shared memory of 64 nodes respectively. In wavefront computations, given a 64×64 matrix, the left and top edges of which are all '1', the computation of each remaining element depends on its left, above and above-left neighbors.

Due to the fact that the cost-based deflection routing can not guarantee livelock-free under some fault patterns, we compare the FoN, FTDR and FTDR-H algorithms in the application platform. Fig. 10 (a)-(c) reveal the average hop count of the three algorithms with an increasing link fault rate from 0 to 30% for the three application workloads. As it can be observed, FTDR and FTDR-H algorithms perform better than FoN especially under high fault rate. The average hop count of FTDR-H is 25%, 18% and 18% less than FoN for the three application workloads respectively. For matrix multiplication and wavefront computation, FTDR-H performs slightly better than FTDR. For FFT, the average hop count of FTDR-H is 10% less than FTDR in the presence of link faults. This is because the local routing table of FTDR-H will converge more quickly than FTDR.

5.4 Area Cost Evaluation

Fig. 11 compares the switch area in different network sizes for different switches (non fault-tolerant, FoN, cost-based, FTDR and FTDR-H). All synthesized results are optimized for area. For FTDR-H algorithm, the 8×8, 12×12 and 16×16 meshes are divided into 4, 9 and 16 4×4 sub-meshes respectively. The areas of FoN and cost-based switches do not increase with the network size. As the network size increases, the area of FTDR switch increases significantly. FTDR requires 2 to 8 times the area of other switches as

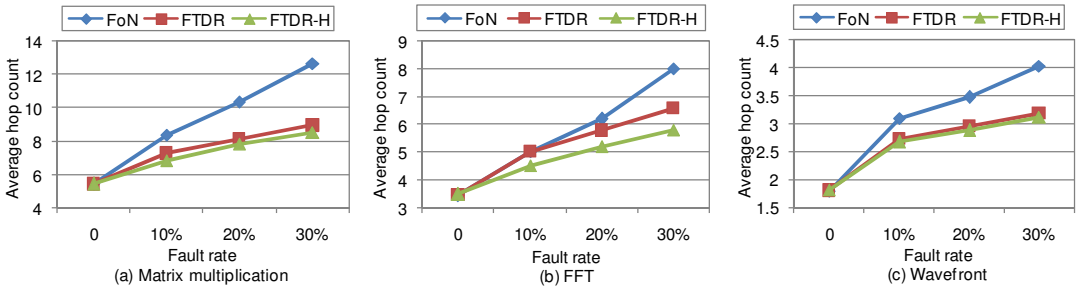

Figure 10: Average hop count with various link fault rates under application workloads.

the network size increases from 4×4 to 16×16. The area of the routing table is the main overhead of FTDR switch. For an $n \times n$ mesh, assume each routing table entry has d bits, so the routing table has a cost of $n^2 \times d$ bits. In FTDR-H algorithm, an $n \times n$ mesh is divided into n^2/m^2 $m \times m$ sub-meshes, so the routing table cost can reduce to $(m^2 + n^2/m^2) \times d$ bits. $(m^2 + n^2/m^2)$ has a minimum value $2n$ (when $m = \sqrt{n}$). The routing table cost of FTDR-H can reduce up to $n/2$ times less than FTDR in theory.

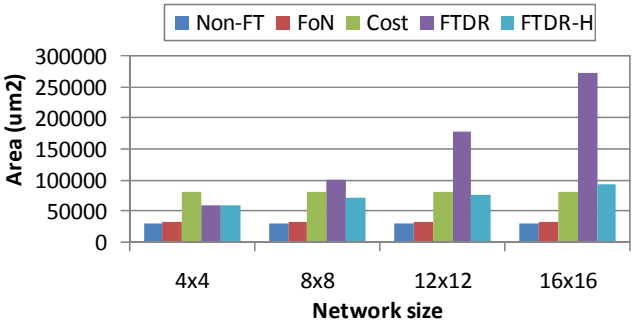

Figure 11: Switch area in different network sizes.

6. CONCLUSION AND FUTURE WORK

In this paper, we propose a fault-tolerant deflection routing algorithm which uses a Q-learning method to reconfigure the routing table to avoid faults. It is a topology-agnostic routing algorithm which is insensitive to the shape of fault regions. An optimized routing algorithm based on hierarchical Q-learning is also proposed to reduce the routing table size. We compare the proposed FTDR and FTDR-H switches with other fault-tolerant deflection switches for both simulation and synthesize results. Although FoN switch has smaller hardware overhead, it can only handle limited fault patterns with the regular shape, which is not practical. Because of the large area overhead of the routing table, FTDR switch is suitable for small and medium network sizes. Compare to FTDR switch, FTDR-H switch can save the area up to 27%. Simulation results show that in the presence of link faults, FTDR and FTDR-H switches outperform the other two fault-tolerant deflection switches and a turn model based fault-tolerant switch.

In future work, we will explore fault detection mechanism and extend the fault-tolerant routing algorithm for irregular NoC.

7. ACKNOWLEDGMENTS

The research is partially supported by the National 863 Program of China(No. 2009AA01Z102), and the National Natural Science Foundation of China (No. 60873212 and No. 60873016).

8. REFERENCES

[1] C. Constantinescu, "Trends and challenges in VLSI circuit reliability," *IEEE Micro*, 23(4):14–19, July-August 2003.

[2] Z. Lu, M. Zhong and A. Jantsch, "Evaluation of on-chip networks using deflection routing," *In Proceedings of ACM Great Lakes Symposium on VLSI*, pages 296–301, May 2006.

[3] T. Moscibroda and O. Mutlu, "A case for bufferless routing in on-chip networks," *In Proceedings of International Symposium on Computer Architecture*, pages 196–207, June 2009.

[4] M. Hayenga, N.E. Jerger and M. Lipasti, "SCARAB: a single cycle adaptive routing and bufferless network," *In Proceedings of International Symposium on Microarchitecture*, pages 244–254, June 2009.

[5] A. Kohler and M. Radetzki, "Fault-tolerant architecture and deflection routing for degradable NoC switches," *In Proceedings of IEEE International Symposium on Networks-on-Chip*, pages 22–31, May 2009.

[6] C. Feng, Z. Lu, A. Jantsch, J. Li and M. Zhang, "FoN: Fault-on-Neighbor aware routing algorithm for Networks-on-Chip," *In Proceedings of the 23rd IEEE International SoC Conference*, pages 441-446, September 2010.

[7] D. Fick, A. Deorio, G. Chen, V. Bertacco, D. Sylvester and D. Blaauw, "A highly resilient routing algorithm for fault-tolerant NoCs," *In Proceedings of Design, Automation and Test in Europe Conference and Exhibition*, pages 21–26, April 2009.

[8] J. Wu, "Fault-tolerant adaptive and minimal routing in mesh-connected multicomputers using extended safety levels," *IEEE Transactions on Parallel and Distributed Systems*, 11(2):149–159, February 2000.

[9] Y.J. Suh, B.V. Dao, J. Duato and S. Yalamanchili, "Software-based rerouting for fault-tolerant pipelined communication," *IEEE Transactions on Parallel and Distributed Systems*, 11(3):193–211, March 2000.

[10] Z. Zhang, A. Greiner and S. Taktak, "A reconfigurable routing algorithm for a fault-tolerant 2D-mesh Network-on-Chip," *In Proceedings of ACM/IEEE Design Automation Conference*, pages 441–446, June 2008.

[11] R. Holsmark, M. Palesi and S. Kumar, "Deadlock free routing algorithms for irregular mesh topology NoC systems with rectangular regions," *Journal of Systems Architecture*, 54(3):427–440, March 2007.

[12] V. Puente, J.A. Gregorio, F. Vallejo and R. Beivide, "Immunet: dependable routing for interconnection networks with arbitrary topology," *IEEE Transactions on Computers*, 57(12):1676–1689, December 2009.

[13] A. Mejia, M. Palesi, J. Flich, S. Kumar, P. Lopez, R. Holsmark and J. Duato, "Region-based routing: a mechanism to support efficient routing algorithms in NoCs," *IEEE Transactions on VLSI Systems*, 17(3):356–369, May 2009.

[14] M. Majer, C. Bobda, A. Ahmadinia and J. Teich, "Packet routing in dynamically changing networks on chip," *In Proceedings of IEEE International Parallel and Distributed Processing Symposium*, pages 154b, April 2005.

[15] E. Nilsson, M. Millberg, J. Oberg and A. Jantsch, "Load distribution with the proximity congestion awareness in a network on chip," *In Proceedings of Design, Automation and Test in Europe Conference and Exhibition*, pages 1126–1127, March 2003.

[16] X. Chen, Z. Lu, A. Jantsch and S. Chen, "Supporting distributed shared memory on multi-core network-on-chips using a dual microcoded controller," *In Proceedings of Design, Automation and Test in Europe Conference and Exhibition*, pages 39–44, March 2010.

[17] J.A. Boyan and M.L. Littman, "Packet routing in dynamically changing networks: a reinforcement learning approach," *Advances in Neural Information Processing Systems*, Vol. 6, pages 671–678, 1994.