IET Computers & Digital Techniques

*Research Article*

IET Journals
The Institution of Engineering and Technology

# Energy-efficient fault tolerant technique for deflection routers in two-dimensional mesh Network-on-Chips

ISSN 1751-8601
Received on 18th January 2017
Revised 11th September 2017
Accepted on 24th October 2017
E-First on 23rd February 2018
doi: 10.1049/iet-cdt.2017.0006
www.ietdl.org

*Simi Zerine Sleeba*[1] ✉, *John Jose*[2], *Maniyelil Govindankutty Mini*[3]

[1]*Department of Electronics Engineering, Government Model Engineering College, Kochi, India*
[2]*Department of Computer Science and Engineering, Indian Institute of Technology, Guwahati, India*
[3]*Department of Electronics Engineering, College of Engineering, Cherthala, India*
✉ *E-mail: simi@mec.ac.in*

**Abstract:** New generation multi-processor system-on-chips integrate hundreds of processing elements in a single chip which communicate with each other through on-chip communication networks, commonly known as network-on-chip (NoC). Routers are the most critical NoC components and deflection routing is a technique used in buffer-less routers for better energy efficiency. Massive integration of devices along with fabrication at deep sub-micron level feature sizes increases the possibility of wear out and damage to various components resulting in unreliable operation of the chip. Hence NoC fabric in general and routers, in particular, should be equipped with built-in fault tolerance mechanisms to ensure the reliability of the chip in the presence of faults. The authors propose an energy-efficient routing technique that can tolerate permanent faults in NoC links by introducing a simple logic unit placed next to the output port allocation stage of the deflection router pipeline. This technique incurs minimum wiring overheads and promises a stable network throughput for high fault rates. Evaluation of the proposed method on 8 × 8 mesh NoC for various fault rates reports reduced flit deflection rate and hop power which brings about a significant reduction in dynamic power consumption at the inter-router links compared to state-of-the-art fault tolerance techniques.

## 1 Introduction

Increasing the integration capacity of transistors in integrated circuits has made it possible to realise multi-core chips which can accommodate thousands of processing elements (Pes) in a single silicon substrate. The high processing capability of these chips demands modular communication architectures like networks-on-chip (NoC), which offers packet-based communication through a set of connected routers and links. Fig. 1*a* shows a two-dimensional (2D) NoC system having 16 routers (interconnected in a 4 × 4 mesh topology), each of which is connected to a PE. In a homogeneous multi-processor system-on-chip, each PE consists of an out-of-order superscalar processor and one or two levels of cache memory. Cache misses account for the generation of packets to the NoC framework. TERAFlops [1] and Tile64 [2] are prototype chips with 64 and 80 processor cores, respectively, interconnected using mesh NoCs.

Energy-efficient on-chip communication is achieved by eliminating router buffers [3] and using deflection routing

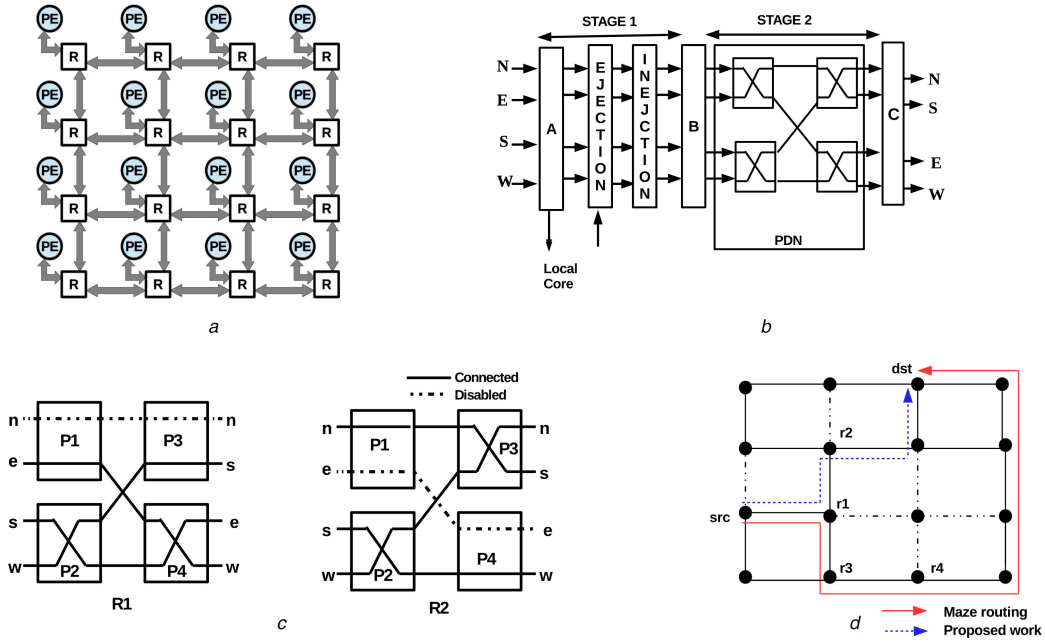


**Fig. 1** ***(a)*** *A 4 × 4 mesh NoC system,* ***(b)*** *Bufferless deflection router architecture using PDN,* ***(c)*** *Connections between permuters of PDN in FaFNoC Router with faulty north port (R1) and faulty east port (R2),* ***(d)*** *Demonstration of path traversed by flit from source (src) to destination (dst) in a 4 × 4 mesh NoC using Maze routing and proposed work*

mechanism to forward packets between various processing cores [4]. In deflection routing, packets are split into smaller units called flits (flow control units) which are routed independently by keeping destination address in the flit header. In conventional deflection routers, incoming flits stay in the router for two cycles before they are forwarded to neighbouring routers through inter-router links [4, 5]. Major functional units of a two-stage pipelined deflection router are given in Fig. 1*b*. Here A, B and C are the pipeline latches. The first stage in the router pipeline (between A and B) comprises of ejection of locally destined flits and injection of newly generated flits into the network. A permutation deflection network (PDN) [5] at the second stage of the router pipeline (between B and C) effectively assigns output ports to all the flits in B. Flits hop from one router to the next until they reach their destination router where they are ejected from the network. Some of these hops are productive, i.e. the flit advances one more step closer to its destination. Some of the hops are termed as deflections since the flit moves further away from its destination. Proper livelock mechanisms ensure that these flits get productive ports in subsequent routers so that they finally reach their destinations. Higher deflection rates lead to increased flit latency and dynamic power dissipation due to unproductive movement of flits through the NoC links. Some of the sophisticated deflection routers use a small number of side buffers to store misrouted flits and hence minimise deflections [6–8].

NoC designs need to meet tough latency and throughput targets, under stringent area and power budgets. Unfortunately, as critical dimensions shrink, the reliability of the chip degrades as well. Permanent faults caused by physical damages such as manufacturing defects and device wear-out may cause the entire chip to fail. Electromigration is an operational stress-induced mechanism which causes material deformations and loss of connections in a circuit. According to the ITRS 2009 Interconnect Report [9], the main cause of on-chip metal interconnect reliability loss in current and future CMOS technologies are electromigration. Unpredictable causes like power grid fluctuations and particle hits may cause transient faults in chips which require complex methods for error detection and correction. As the sole medium of communication in multicore chips, the NoC should be designed to tolerate transient and permanent failure of components and provide reliable communication between healthy nodes inside the chip [10].

The occurrence of faults in a router's internal functional blocks or inter-router links hinders the transit of flits through predetermined paths obtained during port allocation phase. In order to tolerate faults and facilitate flit movement, routers use additional hardware and adaptive algorithms to redirect flits through fault-free paths. The inherent path diversity of mesh topology allows most of the routers to be accessible through multiple paths from any other router in the network. In this work, we focus on tolerating permanent faults occurring in the links/ports of a 2D mesh NoC using deflection routers. We propose a simple and energy efficient *fault-tolerant logic unit* (FTLU) connected to the output of the PDN that is used for port allocation. The FTLU reallocates healthy output ports to flits which have been allotted faulty output ports by the PDN. Our algorithm promises guaranteed delivery of flits from a faulty router to the destination provided a path exists between the two. It also exploits the path diversity of mesh topology to ensure that the majority of the flits reach the destination through minimal paths. Experimental results show that our method has superior performance and energy efficiency compared to the state-of-the-art fault-tolerant techniques [11, 12] with 38% higher throughput, 16 and 20% reduction in hop count and link power, respectively, with uniform traffic conditions and 30% link fault rate.

The organisation of the remaining of the paper is as follows. In Section 2, we discuss some of the earlier works related to fault-tolerant routing in mesh NoCs and the motivation behind this work. Section 3 describes how a faulty router is modelled in our simulator and in Section 4, we explain the proposed router architecture and how fault tolerance is achieved using this technique. The experimental analysis is discussed in Sections 5 and 6 and we conclude the paper in Section 7.

## 2 Related work and motivation

Several routing techniques have been proposed so far for diagnosis and tolerance of permanent faults in 2D NoC systems. Fault-tolerant routing techniques are broadly classified as centralised or distributed. Most of the centralised methods [13–15] use central controllers for propagating fault information to all routers whereas, in distributed methods [13, 16–18], fault tolerance is achieved by local decision making at each router. A big drawback of centralised methods is that the controllers may become faulty, which may cause the entire chip to fail. Deflection routers mostly follow a distributed approach by depending on local or neighbour's fault information for taking routing decisions. Permanent link failures can occur in any random location in the network. One of the major goals of fault-tolerant routing methods is to achieve 100% fault coverage, i.e. to achieve guaranteed delivery of a packet to its destination if a path exists between the faulty router and the destination, regardless of the number and location of faults [12]. Some of the earlier works [19] restrict the number of faulty links/regions which can be gracefully tolerated by the system. Some others offer moderate fault coverage and place strict limitations on tolerable fault patterns within the network with increasing fault rate [13, 15, 17, 20]. Maze routing [12], uDirec [14] and Ariadne [18] algorithms claim 100% fault coverage whereas uLBDR [13] and Face routing [21] can tolerate fairly high fault rates.

Routing algorithms should necessarily guarantee deadlock and livelock freedom of flits [22]. In input buffered routers, fault resilient routing algorithms are based either on turn-model [16, 23] or on virtual channels [24]. In mesh NoCs, buffer-less deflection routers have an equal number of input and output ports, hence all flits entering the router are sent out at the end of a router pipeline [4, 25]. Due to this, deflection routers are inherently free from deadlock problem [11, 12, 26]. Some of the recently proposed fault-tolerant routing algorithms like FaFNoC [11] and Maze routing [12] use the golden and silver flit mechanisms used in conventional CHIPPER [5] and MinBD [6] for livelock avoidance.

FTDR is a deflection routing method which achieves fault tolerance by updating routing tables periodically so that flits bypass faulty links in their path [10]. Routing tables increase the router area significantly and hence nullify the inherent advantage of buffer-less deflection routers which is to reduce chip power and area. FaFNoC [11] is a recent fault tolerant deflection routing technique which uses a modified form of the PDN mentioned in CHIPPER [5]. As shown in the examples of Fig. 1*c*, the interconnections between the four permuter blocks (P1, P2, P3, P4) inside the PDN are adaptively enabled or disabled such that faulty output ports are not allocated to flits. The flit header is extended with additional bits to transmit information about faulty links between routers. Maze routing [12] is yet another fault resilient method that promises 100% fault coverage and guaranteed packet delivery on various topologies and deflection router architectures. It is a variant of face routing algorithm used in wireless *ad-hoc* networks, which randomly allocates output ports for flits whose productive ports are faulty [21]. Maze routing also requires flit header extension for carrying information regarding guaranteed flit delivery and unreachable nodes in the network. Since FaFNoC and Maze routing use PDN-based deflection router architecture that we use in our proposed work, we use these methods for comparison with our results. We conduct an experimental analysis of FaFNoC and Maze routing methods on an $8 \times 8$ mesh NoC and observe a few factors which are limiting them in attaining a better performance. In the proposed work, we address these issues by a superior FTLU.

*Structural inefficiency:* In the FaFNoC router, a fault handler block disables some of the connections between permuter blocks P1, P2 and P3, P4 in the PDN so that flits do not move to faulty output ports during port allocation. In router R1 of Fig. 1*c*, the north link is faulty and the connection between P1 and P3 (shown as dashed line) in the PDN is disrupted so that north output is not available for port allocation. A limitation of this technique is that a flit from the east input port of P1 will be prohibited from taking up the south output port connected to P3 if it is the flit's preferred output port. Such a flit is compelled to choose either the east or west output port, which results in deflection of the flit in an unproductive

direction. Similarly, in router R2 of Fig. 1*c*, where the east link is faulty, the link between P1 and P4 is disconnected. Due to this, north input flits are prohibited from moving to the west output port. In FaFNoC, flits dynamically switch between XY and YX routing in order to choose output ports with lesser traffic congestion. This increases the possibility of flits taking more turns (e.g. from north to east or west) compared to static XY algorithm. The defective inter-permuter connection between input and output ports in the PDN of faulty routers does not support these turns, hence deflection rate of flits increases. From our evaluations on $8 \times 8$ mesh, we find that under low traffic injection rates (<0.1 flits/cycle/core) and 10% faulty links, 22% of overall flit deflections is due to this structural inefficiency of the PDN explained above.

*Algorithmic inefficiency:* We implement Maze routing algorithm on an NoC with deflection routers that uses a PDN for output port allocation. In each router, the preferred output port for each input flit is computed. Maze routing algorithm identifies the flits whose desired output ports are faulty and allots them at random to other healthy output ports either towards the right (clockwise direction) or the left (anti-clockwise direction) of its desired port. To achieve livelock freedom, the chosen direction is stored in the flit header and this direction rule is followed by the flit in all subsequent routers through which it traverses. Fig. 1*d* shows a possible path of a flit from the source(src) to the destination(dst) using Maze routing. From src, a flit traverses along the *x*-direction towards dst and reaches router r1. From r1, the preferred output for the flit is in the east which is assumed to be faulty (shown as a black dotted line). So the flit randomly chooses a fault-free output port either to the left (north port) or to the right (south port) of the east port with equal probability. The solid red line in Fig. 1*d* shows the path followed by a flit which deflects through the south port of r1 to router r3. In all subsequent hops, this flit chooses the healthy port to the right of the line joining the current router to the destination router. Accordingly, the flit reaches its destination in eight hops. In our evaluation of Maze routing technique, we find that 34% of flits which deflect due to faulty output ports traverse non-minimal paths to reach their destinations on account of this random selection of direction. These additional deflections increase network activity factor and consume dynamic power. Larger network sizes and higher fault rates result in increased flit latency and throughput degradation. In order to overcome this algorithmic inefficiency, we propose a routing algorithm that explores shortest available paths in the presence of faulty links and delivers the flit from src to dst in four hops (shown as a blue dotted path in Fig. 1*d*).

*Area overhead:* Fault-tolerant router designs should adhere to strict area and power budgets. We choose FaFNoC and Maze routing techniques for comparison with our work as these routers do not employ routing tables and hence occupy very less chip area. In Maze routing, information such as Manhattan distance between current router and destination, the direction of deflection (left or right) and co-ordinates of the current node are coded in a flit's header using additional bits. In an $8 \times 8$ mesh NoC, flit header extension increases the flit channel width by 10%. In FaFNoC, hop count and fault status of a flit are also transmitted in the header which increases the channel width by 8%. Widening of flit channels leads to an almost quadratic increase in the router area [27]. The proposed method reduces this area and wiring overhead by minimising the fault information transmitted in the flit header.

## 3 Faulty-router model

In our simulations of NoCs with faulty routers and links, we use a coarse grain model which effectively represents faults in bidirectional flit channels as well as component failures inside routers. A fault in either of the input or output channels of a router is represented by disabling both input and output channels in that direction. This ensures that an equal number of active input and output ports are there for a router which is essential for deflection routing [4]. A failure in any of the datapath elements of a router is effectively modelled by converging it to a fault in a specific input–output port. The link corresponding to this port is then disabled by setting appropriate ports of the two routers at its ends as faulty. Failures occurring in crucial functional units of a router are fatal to router operation. Such routers are fully disconnected from the network by disabling all its input–output ports. Four fault flags (1 bit each) are used in each router to represent the functional correctness of output ports in north, south, east and west directions. The fault flag corresponding to a faulty input–output port of a router is set. At the same time, the fault flag of the corresponding input–output port of the adjacent router is also set. Fault flags corresponding to healthy ports of routers are reset. The functional blocks used for port re-allocation read the status of the fault flags for activating their circuits (to be explained in the next section).

The values of fault flags are set or reset during fault diagnosis phase. Diagnosis of faults is beyond the scope of this paper and it is assumed that the flags are updated using online test methods or during system reboot.

## 4 Router architecture

In this work, we propose a fault-tolerant deflection router architecture with a two-stage pipeline. The input stage (stage 1) of the pipeline consists of functional blocks for route computation, ejection and injection of flits. The output stage (stage 2) extends from registers B to C. Fig. 2 depicts the output stage of a router having four bidirectional ports in a mesh NoC. It consists of a PDN whose four output lines are connected to the *FTLU*. Latches (L1, L2, L3, L4) are placed between each pair of output lines of the FTLU. In order to facilitate fault tolerance, we adopt the method of reallocating healthy output ports to flits which are allocated to faulty ports by the PDN. The proposed architecture enables two types of displacements of a flit from a faulty to a healthy port: (i) in an orthogonal direction using FTLU and (ii) in the geometrically opposite direction using a latch. Either of these two displacements or a combination of both may be used for reallocation such that all flits reside in healthy ports of register C at the end of the router pipeline. The following subsections explain these mechanisms in detail.

### *4.1 Permutation deflection network*

At the end of the input stage, four flits that are ready for output port allocation are available in a pipeline register B as shown in Fig. 2. These are either flits in transit through the router or newly injected flits from the local processing core. The PDN reads these four flits and maps them to the four output lines in north, south, east and west directions using four permuter blocks P1, P2, P3 and P4 as per the logic used in CHIPPER, respectively [5]. Each permuter block has two input and two output ports. At each of these blocks, the flit with higher priority is allotted to an output port of its choice and the flit with lower priority is allotted to the remaining output port. In this paper, higher priority is assigned to the flit with lesser number of hops to its destination.

### *4.2 Fault-tolerant logic unit*

The FTLU consists of two sections which are placed in parallel in the router pipeline. The upper section consists of two permuter blocks viz. P5 and P6. P5 connects flits from north and south output lines of the PDN to east and west output lines of the FTLU. Similarly, P6 maps flit from east and west outputs of PDN to north and south outputs of the FTLU. In the lower section of FTLU, two swapping blocks viz. SWAP1 and SWAP2 are provided. In SWAP1 block, a flit in the faulty north or south port of PDN is swapped with a flit in the healthy east or west port. Similarly, SWAP2 interchanges flit between faulty east or west ports and healthy north or south ports. In each router, four fault flags NF, SF, EF and WF represent the healthiness of the output ports in the north, south, east and west directions, respectively. The flag bit corresponding to a faulty output port will be set. Port allocation by the PDN is done on the basis of computed route of a flit and it does not consider the fault status of the router's output ports. As a result, some flits may be assigned to the faulty ports at the output of PDN. The FTLU reallocates such flits to healthy ports in an orthogonal direction with respect to the faulty port. In a router with four output ports, the north and south ports are orthogonal to the east and west ports. Algorithm 1 (see Fig. 3) describes the steps followed in the FTLU.

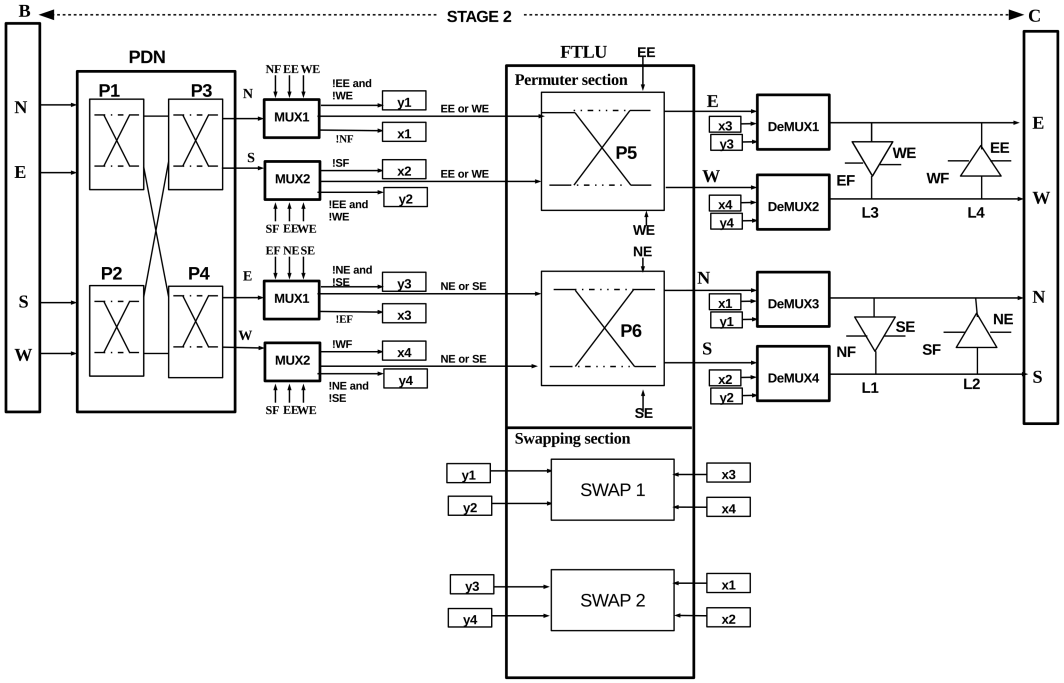


**Fig. 2** *Output stage of the proposed router architecture*

1: Inputs : pdnOutput(N,S,E,W), Faultflags(N,S,E,W), Emptyflags(N,S,E,W);
2: Outputs : ftluOutput(N,S,E,W);
3: **for** <each port in N,S,E,W> **do**
4: **if** (faultflag(port)) **then**
5: **if** port is (N or S) and EmptyFlag(E or W) **then**
6: $P5Input(port) \leftarrow pdnOutput(port)$ // Assigning PDN outputs to a permuter P5 or P6 inputs
7: **else**
8: $SWAP1Input(y1 \quad or \quad y2) \leftarrow pdnOutput(port)$ // Assigning PDN outputs to swap circuit inputs as in Figure 2
9: **end if**
10: **if** port is (E or W) and EmptyFlag(N or S) **then**
11: $P6Input(port) \leftarrow pdnOutput(port)$
12: **else**
13: $SWAP2Input(y3 \quad or \quad y4) \leftarrow pdnOutput(port)$
14: **end if**
15: **else**
16: $(x1, x2, x3, x4) \leftarrow pdnOutput(port)$ // Assign PDN output to x1 or x2 or x3 or x4 as in Figure 2.
17: **end if**
18: **end for**
19: $Emptyflag(P5Output1 \quad or \quad P5Output2) \leftarrow HighPriority(P5Input1 \quad or \quad P5Input2)$
20: $Remaining(P5Output1 \quad or \quad P5Output2) \leftarrow LowPriority(P5Input1 \quad or \quad P5Input2)$
21: $FLB(P5Output1) = 1; \quad FLB(P5Output2) = 1$ // FLB set to YX routing
22: $Emptyflag(P6Output1 \quad or \quad P6Output2) \leftarrow HighPriority(P6Input1 \quad or \quad P6Input2)$
23: $Remaining(P5Output1 \quad or \quad P5Output2) \leftarrow LowPriority(P5Input1 \quad or \quad P5Input2)$
24: $FLB(P6Output1) = 0; \quad FLB(P6Output2) = 0$ // FLB reset to XY routing
25: $SWAP1(y1) \leftrightarrow (x3 \quad or \quad x4); \quad SWAP1(y2) \leftrightarrow (x3 \quad or \quad x4); \quad FLB(x3 \quad or \quad x4) = 1;$
26: $SWAP2(y3) \leftrightarrow (x1 \quad or \quad x2); SWAP2(y4) \leftrightarrow (x1 \quad or \quad x2); \quad FLB(x1 \quad or \quad x2) = 0;$ // Swapping circuit
27: $ftluOutput(N) \leftarrow (P5Output(N) \quad or \quad x1 \quad or \quad y1); ftluOutput(S) \leftarrow (P5Output(S) \quad or \quad x2 \quad or \quad y2);$ // Demux outputs
28: $ftluOutput(E) \leftarrow (P6Output(E) \quad or \quad x3 \quad or \quad y3); ftluOutput(W) \leftarrow (P6Output(W) \quad or \quad x4 \quad or \quad y4);$

**Fig. 3** *Algorithm 1: FTLU*

Entry of flits into the FTLU is restricted by multiplexers that are controlled by fault flags as shown in Fig. 2. If a flit is present at a faulty output of the PDN, the corresponding fault flag is 1 and this gates the flit into one of the two sections of the FTLU (line numbers 4–14 in Algorithm 1).

A flit at a faulty port of the PDN either passes through the permuter section or the swapping section for port reallocation. The permuters P5 and P6 in the upper section are activated for port reallocation only if any one of their output ports are empty (line numbers 19–24 in Algorithm 1). Four 1 bit flags, NE, SE, EE and WE represent whether the output ports in north, south, east and west directions, respectively, are empty or not. To avoid livelock problem, it is necessary to reallocate a flit to an output port other than in its input direction. Based on this restriction, a flit may try to reallocate to an orthogonal port in the FTLU, but the desirable port maybe occupied by another flit from the PDN. For energy-efficient reallocation in the FTLU, flits from faulty ports are given higher priority to occupy fruitful ports. By activating a swapping block, such flits are swapped with the flits that reside in their productive ports (line numbers 25 and 26 in Algorithm 1). The flits from the healthy ports of the PDN are available to the swapping blocks at registers $x1$, $x2$, $x3$ and $x4$ and flits from faulty ports are available at $y1$, $y2$, $y3$ and $y4$. The individual outputs of the permuter section and swapping section of the FTLU are merged into four output lines using demultiplexers. In a fault-free router whose four input–output ports are healthy, flits from the output of PDN bypass the FTLU and move to their respective output ports through the demultiplexers (line numbers 27 and 28 in Algorithm 1).

In a mesh NoC, the routers at the corners and edges have two and three pairs of input–output ports, respectively. In order to maintain simplicity, we use homogeneous router architecture throughout the NoC, i.e. the architectures of the corner and edge

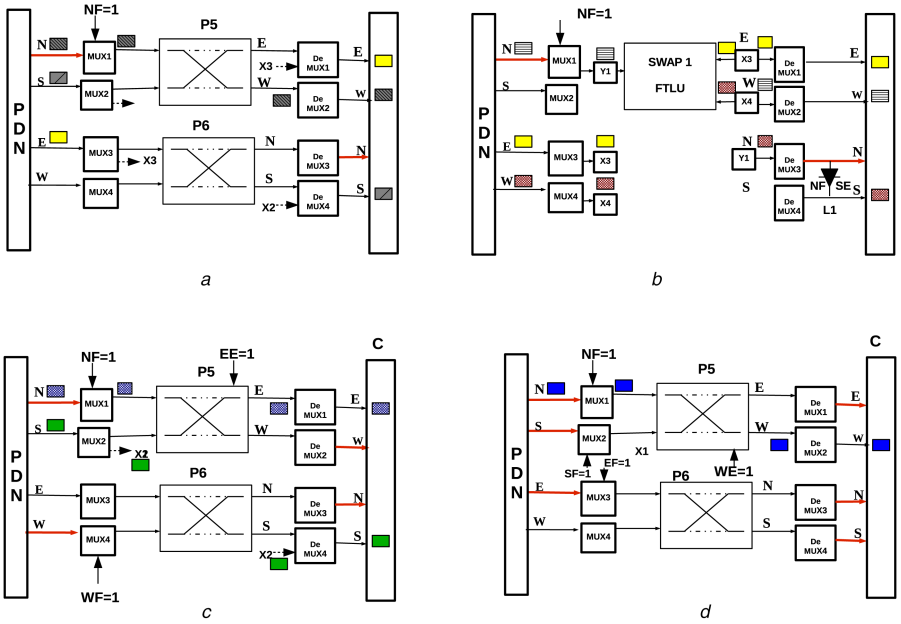


**Fig. 4** *Reallocation of output ports in routers with*
***(a)*** One faulty port (north) using permuter (P5), ***(b)*** One faulty port (north) using swap circuit (SW1) and latch (L1), ***(c)*** Two faulty ports (north and west), ***(d)*** Three faulty ports (north, south and east)

routers are similar to that of the central routers. Route computation of input flits using XY and YX methods is in such a way that the forbidden output ports of edge and corner routers are not allocated by the PDN. In the FTLU also, the output ports in the forbidden directions are treated as faulty by setting the corresponding fault flag. For example, in the router at the top right corner of the mesh, north and east output ports are set as faulty (NF = 1, EF = 1). Similarly, for a router at the left edge of the mesh, the west output is set as faulty (WF = 1). The total area and power consumed by the NoC can be reduced by using customised architectures for routers at the edges and corners.

### *4.3 Port reallocation using latches*

A latch is used to transfer flits from one port to another when it is enabled. We use the term 'latch' to refer to a circuit that performs this function. Edge triggered flip-flops or level triggered latches could be suitably used for implementing it in hardware. In the proposed method, we use a latch to reallocate a flit from a faulty port to a healthy port in the geometrically opposite direction. In a router with four input–output ports, the two pairs of geometrically opposite ports are east and west, north and south. When a flit deflects to a direction which is geometrically opposite to its productive port, the distance to its destination increases. Hence, this kind of port reallocation is done at the final stage of the router pipeline to reallocate flits that remain in faulty ports at the output of FTLU. As shown in Fig. 2, latches L1 and L2 are connected between north and south output lines of the router whereas L3 and L4 are connected between east and west output lines. As a result of swapping operation in the FTLU, a flit from a healthy port is transferred to a faulty port in its orthogonal direction. A latch from the faulty port is enabled to transfer this flit to a healthy port in the geometrically opposite direction.

We explain the proposed port reallocation mechanism in routers with at most three faulty output ports using examples.

*Routers with one faulty port:* In Fig. 4*a*, we assume that the north port of a router is faulty (NF = 1). At the output of PDN, the north, south and east ports are occupied by flits. The flit from the north port is directed to the permuter P5 in the FTLU by MUX1. P5 allocates this flit to the west output port which is vacant (line numbers 5 and 6 in Algorithm 1). The flits in the south and east output ports of PDN bypass the FTLU through registers $x2$ and $x3$, respectively, and move to the pipeline register C via demultiplexers (DeMUX4 and DeMUX1, respectively).

A special case of a router with one faulty port is shown in Fig. 4*b*. Here, flits occupy the north (NF = 1), east and west output ports of PDN and south port is empty. We assume that the flit in the north port entered the router through the south input port. This flit is restricted from tracing back through the south port due to reasons which are explained in Section 4.6. Therefore, east or west ports are the only suitable locations for reallocating this flit. Assuming that the north flit prefers the west port which is non-empty, the swapping circuit (SWAP1) in the FTLU is activated to interchange flits between north (in register $y1$) and west ports (in register $x3$) (line number 25 in Algorithm 1). After swapping, outputs of the FTLU are available in registers $y1$, $x3$ and $x4$. The flit in $y1$ is joined to the north output line by DeMUX3 and then reallocated to the healthy south port by enabling latch L1.

*Routers with two faulty ports:* In Fig. 4*c*, the north and west ports of a router are faulty (NF = 1, WF = 1). Here, a maximum of two flits enter the router through the south and east ports. After port allocation by the PDN, each flit occupies one among the four output ports of the PDN. The different situations that may arise are categorised here.

*Case 1: The two flits are at north and south output ports of PDN.* Since the north port is faulty (NF = 1), MUX1 is enabled and the flit is passed to permuter P5 in the FTLU. As shown in Fig. 4*c*, P5 reallocates this flit to the east port which is healthy and vacant (EF = 0, EE = 1). The flit at the south port bypasses the FTLU through register $x2$ and moves to its position in pipeline register C.

*Case 2: The two flits are at east and west output lines of PDN.* Here, MUX4 is enabled since fault flag of the west port is high (WF = 1). The flit passes through MUX4 and the permuter P6 in the FTLU reallocates it to the south port which is healthy and vacant.

*Case 3: One flit is at the north and the other flit is at west output line of PDN.* Permuter P5 reassigns the north flit to the east port which is healthy and empty. Similarly, the west flit is reassigned to the south port through permuter P6. After reallocation, these flits are moved to the south and east positions in pipeline register C.

*Routers with three faulty ports:* When three pairs of input–output ports in a router are faulty, one flit needs to be routed through the single healthy port. The constraint placed on port reallocation is overridden in this special case. At the output of PDN, if the flit occupies a faulty port, it is re-allocated to the single healthy port either by the FTLU or latches. Fig. 4*d* shows a router where all three ports except the west port are faulty. A flit at the north output of PDN is reassigned to the west port by permuter P5 in the FTLU.

### 4.4 Disconnected routers

When all the four output ports in a router are faulty, no flit enters or leaves the router and injection of new flits from the local core is also throttled. Such routers are apparently disconnected from rest of the network. We propose a scheme to eliminate flits that are destined to disconnected routers in the network. A flit qualifies for elimination if it fails in its attempt to enter the destination router through all the four input ports due to a fault. The flit header is extended by a four-bit field called Expiry field (EX field). Each bit in the EX field represents an attempt to reach destination router through the north, south, east and west ports, respectively. Initially, the four bits are reset to zero. If a flit reaches a router at one hop distance from the destination and gets deflected to an unproductive direction due to a faulty port, the corresponding bit of EX field is set as 1. When the four bits in EX field of a flit are set, it is ready for elimination. A functional block called 'Kill Block' in each router examines the EX field of all input flits and erases the qualified flits from the input buffer. Since Kill block can be placed as a parallel block in the input stage of the router pipeline, it does not introduce additional delay in the router's critical path. However, an extension of flit header by four bits increases the width of flit channels which increases the dynamic power dissipation across the links. During simulations, we consider cases of routers having up to three faulty ports only. So the EX field is not appended to the flit header while evaluating the proposed fault-tolerant technique.

### 4.5 Fault loop bit (FLB)

In deflection routing, each flit is routed independently. For this, every flit contains a few extra bits (called header) that carry its source and destination addresses. In the proposed method, the flit header is extended by an additional bit called the fault loop bit (FLB). Productive output ports of a flit in a router are computed by referring to the value of FLB in the flit header. The values 0 and 1 for FLB represent the XY and YX route computation methods, respectively. In the following section, we explain how FLB is toggled during port reallocation to prevent livelock.

### 4.6 Livelock problem

In a deflection router, a flit maybe reallocated to an unproductive port due to a fault in its productive direction. If a single deterministic method (e.g. XY routing) is used for route computation in every router, the flit maybe routed back to the same fault location repeatedly. This results in livelock, a problem which should be completely eliminated by the routing algorithm. The proposed technique ensures livelock freedom for mesh NoCs with all types of fault patterns except those with gateway routers explained in Section 4.6.2. For this, we impose two constraints during port reallocation:

i. Disallowing the reallocation of flits to output ports in the same direction as their input ports.
ii. Adaptive switching between XY and YX routing methods to change the direction of traversal of the flit after port reallocation.

The first constraint says that a flit is not allowed to back trace to its input direction after port reallocation. For example, a flit which entered the router through the east input port will not be assigned to the east output port during port reallocation by the FTLU or latches. Such flits may choose one of the two remaining ports that are eligible for reallocation. In a router with three faulty ports, a flit entering through the single healthy port is exempted from this restriction. Initial route computation of a flit is based on XY routing, i.e. the default value of FLB is 0. On obtaining a faulty port at the output of PDN, the flit passes through FTLU or latch for reallocation to a healthy port. After reallocation, if the flit is assigned to a port in the horizontal direction (east or west) of the mesh, its FLB value is set to 1. Accordingly, productive ports for this flit are computed using YX method in the next router, which implies that the flit makes a vertical hop to a router in the adjacent row. FLB of a flit traversing through a port in the vertical direction (north or south) of the mesh will be reset to 0. Hence, XY method is used for route computation in the next router, which moves the flit to a router in the next column. Whenever a flit bypasses the port reallocation stage in a router, its FLB is reset to the default value, i.e. 0 irrespective of the port to which it is allocated.

With the example in Fig. 5*a*, we illustrate how toggling of FLB and restricting reverse movement of flits guarantees livelock freedom of flits in a network with multiple faulty links. A flit which is destined to router(6, 6) starts from router(3, 0) and initially follows XY routing. At (3, 2), the east output port is faulty. By port reallocation using FTLU, the flit is reassigned to the south output port. Since the flit traverses to (4, 2) through the south port of (3, 2), FLB of flit remains at the value 0. So, next route computation at (4, 2) is based on XY routing and the flit hops to router(4, 3) through the east output port. Again, the flit encounters a fault in the productive east port at router(5, 3). This flit cannot be reallocated by the FTLU since the south port of (5, 3) is faulty and the north port is not allowed. So the flit is reallocated to the west port by a latch and its FLB is set to 1. This implies that route computation in router(5, 2) is based on YX routing. Port reallocation for the flit at router(6, 4) is similar to that in router(3, 2). The path followed by the flit up to its destination router(6, 6) is shown in red line in Fig. 5*a*.

*4.6.1 Proof of livelock freedom:* For establishing livelock freedom of the proposed fault-tolerant routing technique, we assume that there exists at least one fault-free path between the current router and destination router of the flit. We prove that a flit traversing the network from router $(x1, y1)$ reaches its destination at router $(x2, y2)$ in a finite number of hops given in the following steps:

*Step 1:* The flit starts from $(x1, y1)$ in the horizontal direction (FLB = 0, XY routing) and encounters a fault in the productive port (east or west) of an intermediate router $(x, y)$.
*Step 2:* Since reallocation of the flit to its input direction is forbidden, only ports in the vertical direction (north or south) are eligible for reallocating the flit. Moving along one of the vertical ports, the flit reaches a router $(x \pm 1, y)$ with FLB = 0 (line numbers 22–24 or 26 in Algorithm 1).
*Step 3:* In router $(x \pm 1, y)$, the flit tries to obtain a port in the horizontal direction (XY routing). If the desired port is healthy, the flit succeeds in traversing horizontally. The co-ordinates of the next router are $(x \pm 1, y \pm 1)$ which is nearer to $(x2, y2)$ than $(x, y)$ and FLB = 0.
*Step 4:* If the desired port in the horizontal direction is faulty, the flit is again reallocated to the vertical port other than its input (port reallocation in the orthogonal direction by FTLU). This places the flit at router $(x \pm 2, y)$ with FLB = 0. Again Step 3 or 4 is repeated till Step 3 succeeds and the flit reaches a router which lies in the same column as the destination, i.e. $y = y2$.
*Step 5:* Now the flit tries to obtain a port in the vertical direction (north or south). If it encounters a faulty port in an intermediate router $(x, y2)$, the FTLU reallocates the flit to the east or west port (in the orthogonal direction) and FLB = 1. The co-ordinates of the next router are $(x, y2 \pm 1)$.
*Step 6:* The next route for the flit is computed using YX method. If the north or south port of the router is healthy, the flit traverses along the vertical direction to the next router $(x \pm 1, y2 \pm 1)$ and FLB = 0. Again Step 3 or 4 is repeated till Step 3 succeeds.
*Step 7:* If the desired port in the vertical direction is faulty, the flit is reallocated to a horizontal port other than its input (port reallocation in the orthogonal direction by FTLU). This places the flit at router $(x, y2 \pm 2)$ with FLB = 1. Again Step 6 or 7 is repeated till Step 6 succeeds and $x = x2$ and in Step 3, $y = y2$. Thus the flit reaches the destination $(x2, y2)$.

In general, if a flit is deviated from its productive path due to a faulty port, it is again guided towards its productive direction by adaptive switching between XY and YX routing methods. Since the flit is not allowed to back trace in its input direction after port

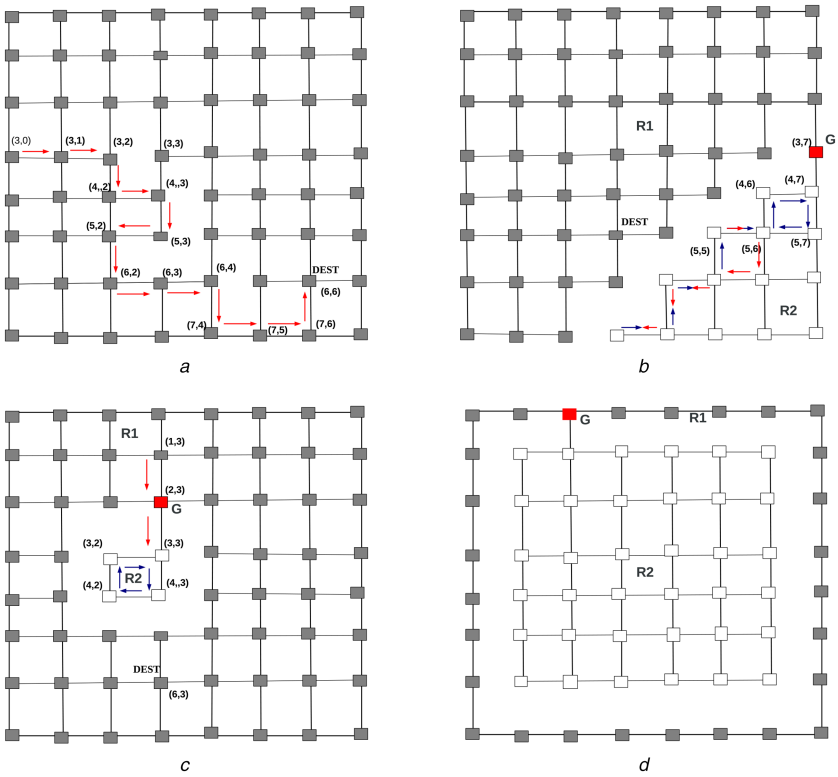


**Fig. 5** *Complex fault patterns*

***(a)*** Livelock free routing in an 8 × 8 mesh with multiple faults in spatially distributed locations, ***(b)–(d)*** Fault pattern dividing the network into two regions R1 and R2 connected to a gateway router, G

reallocation, it will progress towards the destination and reach there in a finite number of hops.

*4.6.2 Fault patterns with gateway routers:* The livelock avoidance mechanism using a single additional bit (FLB) in the flit header works well with fault locations that are spatially distributed in the mesh network like the one shown in Fig. 5*a*. However, certain fault patterns partition the mesh NoC into two or more non-overlapped regions. A gateway router is a router that acts as a single common connection point of such regions in the NoC. A flit from one region can reach its destination in another region only by traversing through the gateway router. Figs. 5*b–d* depict fault patterns that divide the network into two regions, R1 and R2 which are connected through a gateway router, G (shown as a red square). In Fig. 5*b*, a flit from router(5, 5) which is at a two-hop distance from its destination router(5,3) has to be routed through the gateway router(3, 7) due to the fault pattern. In router(4,7), a flit may roam around in an endless loop (shown as a blue line) due to its southward path which does not connect to its destination. Fig. 5*c* shows how a flit moving south towards destination gets stuck in an infinite loop in region R2 as it cannot trace back through the gateway router to region R1. In Fig. 5*d*, region R2 is surrounded by region R1 due to the complex fault pattern. To ensure livelock freedom in such cases, additional bits are required in the flit header to trace back from one region to the gateway router so that a flit can reach its destination located in another region. Our random fault generator shows that at 30% fault rate, the probability of formation of fault patterns consisting of a gateway router is only 0.000001. Considering the overhead incurred, the proposed method does not implement livelock safety measures for such fault patterns.

## 5 Experimental method

We evaluate our proposed technique by comparing various performance parameters with FaFNoC and Maze routing methods explained in Section 2. Experiments are conducted in three phases, i.e. simulations, dynamic power analysis and hardware synthesis.

### *5.1 Simulation framework*

We model FaFNoC, Maze routing and our proposed router architecture using a flit level, cycle accurate simulator, Booksim [28]. Booksim is an open source NoC simulator which models the conventional virtual channel router with two cycle latency. We modify its router pipeline to accurately model the PDN-based deflection routing mechanism and FTLU as mentioned in Section 4. All simulations are done for 8 × 8 mesh NoC. Faults are modelled by disabling a fixed number of bidirectional links in the NoC as mentioned in Section 3. Fault rate is the percentage of faulty links out of the total links in the mesh network. For an 8 × 8 mesh NoC, there are a total of 112 links and a fault rate of 10% denotes that ports corresponding to 11 links are disabled. In our fault model, the disabling of ports does not represent link faults alone. We consider the fact that whenever there is a failure in a datapath element inside a router, it is extrapolated to one of the router's input–output ports. This port is treated as faulty by setting the corresponding fault flag. In order to get a fair comparison, we choose the faulty links by a random selection process. However, we do not consider fault patterns which lead to disconnected routers or gateway routers in the network.

The performance of the NoC is evaluated using various synthetic traffic profiles. Traffic is generated by the processor cores at the rate of 0.1 flits/cycle/core for all simulations. In a fault-free NoC, flit injection rate is approximately equal to its generation rate. As the fault rate in the network rises to 30%, there is a proportionate decrease in the number of fault-free channels in the router. As a result, flits must wait in the processor's core buffer for a longer time before getting injected into the network through vacant channels. Hence, flit injection rate becomes lesser than the generation rate. A set of 20 simulations are conducted for a particular fault rate by random choosing of faulty channels. Average throughput and average hop count are computed from each set of three distinct fault rates for plotting graphs. Simulations are also conducted using real application traces. Combinations of various low, medium and high MPKI applications from SPEC CPU 2006 benchmark suite [29] are run on a multi-core platform, Multi2Sim [30]. The application mixes (M1–M6) and the %MPKI in each mix are given in Table 1. The network packets generated by

**Table 1** Percentage of applications of various network injection intensity in benchmark mixes M1–M6

| Benchmark mix | M1 | M2 | M3 | M4 | M5 | M6 |
|---|---|---|---|---|---|---|
| % of low MPKI | 100 | 0 | 0 | 50 | 0 | 50 |
| % of medium MPKI | 0 | 100 | 0 | 0 | 50 | 50 |
| % of high MPKI | 0 | 0 | 100 | 50 | 50 | 0 |

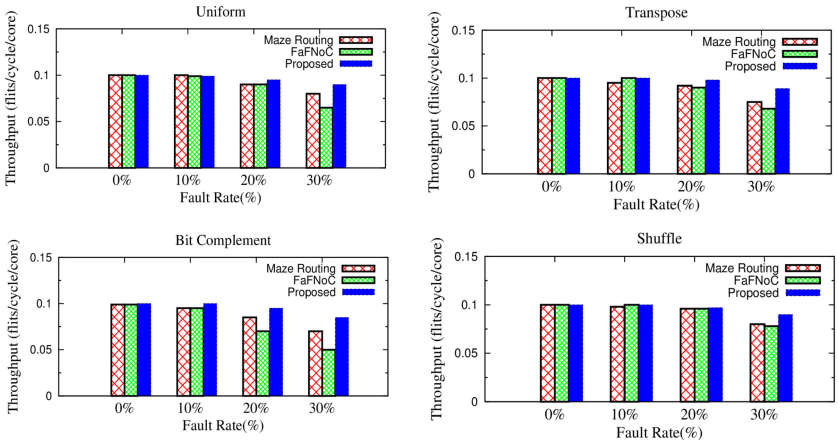


**Fig. 6** *Average throughput versus fault rate under various synthetic traffic patterns in* $8 \times 8$ *mesh network*

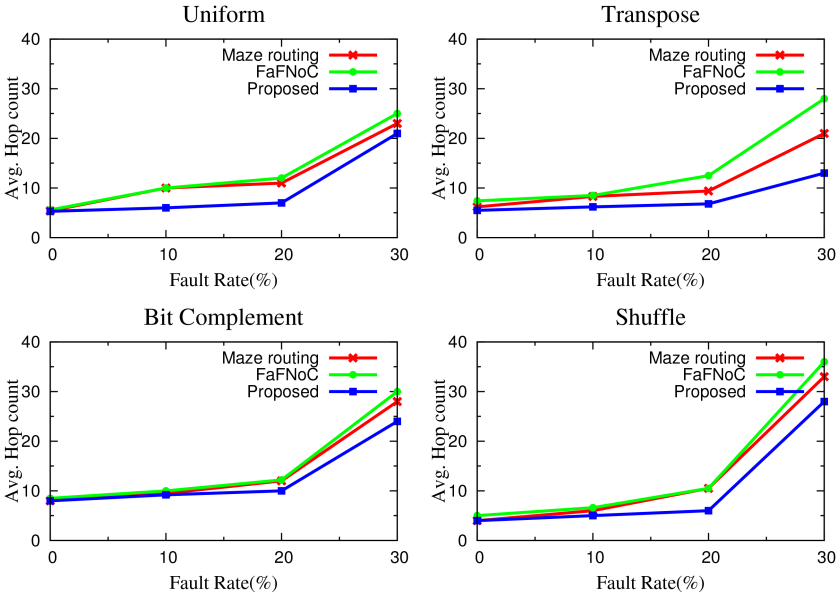


**Fig. 7** *Average hop count versus fault rate under various synthetic traffic patterns in* $8 \times 8$ *mesh network*

these benchmark mixes are injected into the booksim model and network parameters are analysed.

## 6 Analysis of network level parameters

### *6.1 Average throughput, hop count and latency for synthetic traffic patterns*

The throughput of a network is the number of flits delivered successfully per cycle per core and has a maximum value of 1. Hop count is the total number of productive hops and deflections of a flit from source to destination. Fig. 6 shows the average throughput for varying fault rates under uniform, transpose, bit complement and shuffle traffic patterns for the three fault-tolerant routing mechanisms. For a fault-free NoC, it is seen that the throughput value is equal to the network injection rate (which is taken as 0.1 flits/cycle/core). For higher fault rates, the flit injection rate reduces due to congestion in the network. Since deflection routers do not buffer any of the flits in transit, the throughput under various fault rates also decreases in accordance with the injection rate. Fig. 6 shows that our method meets higher performance for all synthetic traffic patterns compared to the other two. At 30% fault rate, it delivers 38 and 13% higher throughput than FaFNoC and Maze routing, respectively, for uniform traffic. Our method efficiently utilises the FTLU along with an adaptive route computation technique to route flits through the shortest fault-free path. In Maze routing, flits encountering faulty ports are reassigned to ports in random directions. Due to this, a large number of flits traverse through longer paths even when shorter paths exist. This increases the hop count and latency for higher fault rates. Even though FaFNoC switches between XY and YX routing to avoid congestion, the lack of adequate structural connectivity at the PDN results in increased deflections and latency. Hence average hop count is found to be the highest in FaFNoC compared to Maze routing and our method as shown in Fig. 7. It also shows the lowest throughput values for all fault rates. At uniform traffic conditions and 30% fault rate, hop count for our method is <9% compared to Maze routing and 16% compared to FaFNoC.

The latency of a flit is the total number of cycles it takes to traverse from its source to destination. In Fig. 8*a*, average flit latency for an $8 \times 8$ mesh NoC is shown as a function of flit injection rate for uniform traffic for various fault rates. For all the three methods, we observe that the network tends to saturate early

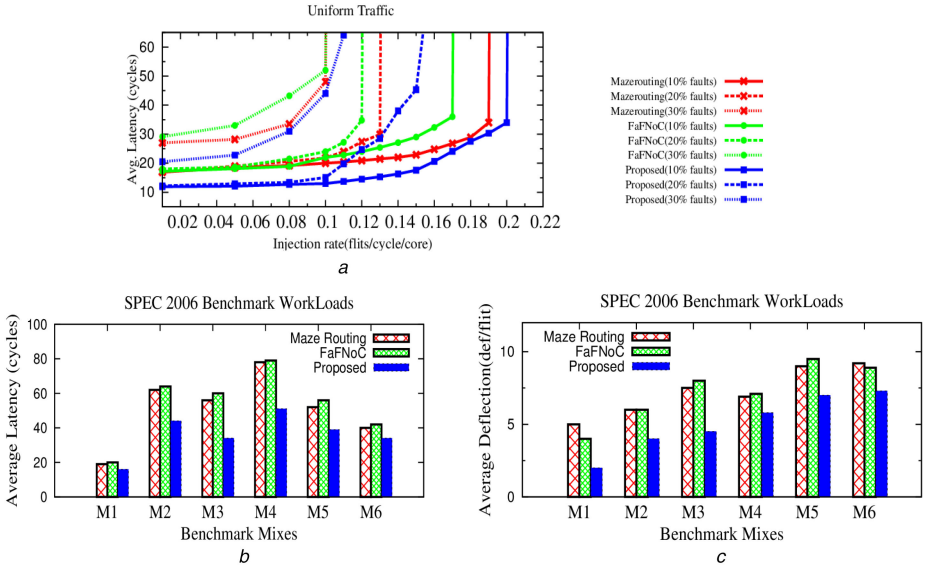


**Fig. 8** *Simulation results of*

***(a)*** Average latency versus injection rate under various fault rates for uniform traffic in $8 \times 8$ mesh network, ***(b)*** Average latency and ***(c)*** Deflection for 10% fault rate under various mixes from SPEC CPU 2006 benchmark applications in $8 \times 8$ mesh network

with increasing fault rate. The proposed method shows 3.8 and 8.7% improvement in network saturation point with respect to Maze routing for 10 and 30% fault rates, respectively. At 30% fault rates and injection rate of 0.1, Maze routing and FaFNoC show a sharp rise in average latency and both networks saturate. From the graph, it is observed that our routing technique is capable of tolerating higher fault rates with a gradual increase in latency for injection rates above 0.1.

### 6.2 Average latency and deflection for real applications

Figs. 8*b* and *c* show the graphs of average latency and deflections obtained for benchmark application mixes (M1–M6) for a fault rate of 10% on flit channels. For real applications, the injection rates are much lower than that chosen for synthetic traffic simulation. For a faulty NoC, at very low injection rates, there is lesser congestion in the network and more than one productive ports may be eligible for reallocation. The merit of our algorithm lies in its capability to reallocate ports that result in shortest possible distance to the flit's destination. We get an average of 37 and 33% lower latency for our proposed method across all mixes compared to FaFNoC and Maze routing, respectively. Using our technique, the majority of the flit hops are productive; hence deflection rates are considerably lower for all benchmark mixes as seen from the figure.

### 6.3 Dynamic link power estimation

Increase in the flit deflection rate leads to increased activity and dynamic power dissipation across the inter-router links. Hence power efficiency of such NoCs is proportional to the link activity factor (LAF) which is derived from the average hop count of flits through these channels during simulations. LAF is a measure of the number of packets per cycle per link and is calculated using the formula

$$\text{LAF} = \frac{(h * n)}{(c * k)}$$

where $h$ is the average hop count (sum of hop count of all packets/$n$); $n$ is the total number of packets injected into the network throughout the simulation at the rate of 0.1 per cycle per core; $c$ is the total number of simulation cycles (taken as 100,000); $k$ is the total number of links (112 for $8 \times 8$ mesh).

Orion 3.0 tool [31] is used to estimate the dynamic power dissipated at the inter-router links. For all the three fault-tolerant routing techniques, the values of LAF corresponding to various hop count values at various fault rates are computed using the above formula. The number of active links in the network decreases with increasing fault rate. Since we calculate LAF as an average value across all links in the mesh, the total number of links is taken as $k$ (irrespective of the number of active links). The calculated value of LAF is given as the input load rate in Orion to determine the dynamic power. We assume the link length to be 2.5 µm and a baseline flit width of 128 bits. For $8 \times 8$ networks, FaFNoC and Maze routing extend the flit header by 12 and 14 bits, respectively, whereas our method uses only one additional bit for FLB.

In Fig. 9, we plot the link power (in mW) obtained from Orion against varying fault rates for four synthetic traffic functions. From the graph of uniform traffic function, dynamic power due to our method is 20% lower than FaFNoC at 30% fault rate. FaFNoC consumes the highest energy among the three methods. The reason for the energy efficiency of our technique is the reduction in hop count by efficient fault-tolerant routing.

### 6.4 Hardware synthesis

We implement Verilog models of the three-router architecture and synthesise using Synopsys design compiler with 65 nm CMOS library. The values of router pipeline latency, static power, and area obtained for the three architectures normalised with respect to the conventional CHIPPER architecture are shown in Table 2. Router delay is the time taken by a flit to move from its input to the output port through various functional units. This can be divided into two stages. The first stage of the three routers has similar functional units, hence all of them have the same delay for the first stage. The second stage consists of the PDN and fault-tolerance mechanisms. The critical path length inside each router and operating frequency of the network are determined by the complexity of the fault-tolerant logic. The output stage of CHIPPER does not include any fault-tolerant logic, so its delay is taken as 1. The FTLU in our proposed router increases the delay of the second stage by 26% when compared with CHIPPER. Area and static power consumed by the control logic of our router are 40 and 35% higher than CHIPPER, respectively. Maze routing incurs the least area and power among the three architectures as it uses fewer functional units for fault tolerance. Even though NoCs using our newly proposed router operate at a lesser frequency compared to the other two methods, a significant reduction in dynamic power dissipation at the inter-router links is achieved by our energy-efficient fault-tolerant routing mechanism. In an $8 \times 8$ mesh NoC, both FaFNoC and Maze routing require 8–10% wider channels to accommodate the extended flit headers [11, 12]. The channel width increases with growing network size, resulting in a larger area for the router's datapath and inter-router links. Our router is advantageous in this

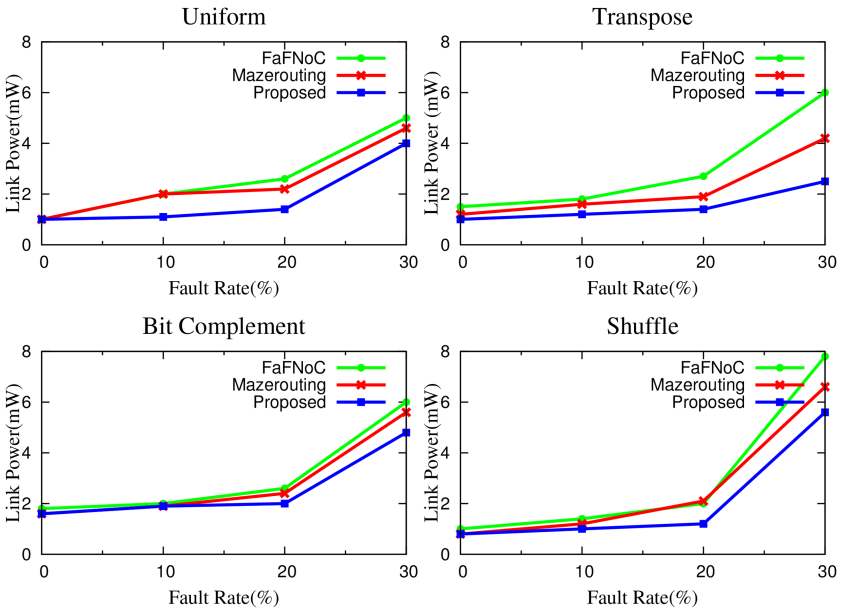


**Fig. 9** *Link power versus fault rate under various synthetic traffic patterns in* $8 \times 8$ *mesh network*

**Table 2** Router delay, area and static power for Maze router, FaFNoC and proposed method normalised w.r.t CHIPPER

| | CHIPPER | Maze routing | FaFNoC | Proposed |
|---|---|---|---|---|
| router delay | 1 | 1.08 | 1.15 | 1.26 |
| static power | 1 | 1.2 | 1.26 | 1.35 |
| router area | 1 | 1.24 | 1.3 | 1.4 |

aspect, since our flit header uses only one additional bit for FLB irrespective of the network size.

## 7 Conclusions

In this paper, we analysed the existing state-of-the-art fault-tolerance mechanism adopted in deflection router-based mesh NoCs. We propose a novel fault-tolerance technique to improve the performance and energy efficiency of NoCs at high fault rates. Our technique consists of an FTLU at the output stage of deflection routers which reallocates flits from faulty ports to healthy output ports. We also use adaptive switching between XY and YX routing algorithms to avoid livelock and route flits to the destination through the shortest available path. This method offers the advantage of simple design and smaller link area compared to similar methods proposed recently. We experimentally show that our method brings about a substantial reduction in flit hop count and dynamic link power consumption of NoC compared to the earlier methods.

## 8 Acknowledgments

This work is supported in part by grant from UGC, Government of India, under the Maulana Azad National Fellowship scheme and in part by DST, Government of India, SERB-ECR Scheme (project number ECR/2016/212).

## 9 References

[1] Hoskote, Y., Vangal, S., Singh, A., *et al.*: 'A 5-GHz mesh interconnect for a teraflops processor', *IEEE Micro*, 2007, **27**, (5), pp. 51–61

[2] Bell, S., Edwards, B., Amann, J., *et al.*: 'TILE64-processor: a 64-core SoC with mesh interconnect'. Proc. IEEE Int. Solid-State Circuits Conf., San Francisco, CA, USA, February 2008, pp. 588–598

[3] Dally, W., Towles, B.: '*Principles and practices of interconnection networks*' (Morgan Kaufmann Publishers Inc., USA, 2003)

[4] Moscibroda, T., Mutlu, O.: 'A case for bufferless routing in on-chip networks'. Proc. Int. Symp. Computer Architecture, Austin, TX, USA, June 2009, pp. 196–207

[5] Fallin, C., Craik, C., Mutlu, O.: 'CHIPPER: a low complexity bufferless deflection router'. Proc. Int. Symp. High Performance Computer Architecture, Washington, DC, USA, February 2011, pp. 144–155

[6] Fallin, C., Nazario, G., Yu, X., *et al.*: 'MinBD: minimally-buffered deflection routing for energy-efficient interconnect'. Proc. NOC Symp., Denmark, May 2012, pp. 1–10

[7] Jose, J., Nayak, B., Kumar, K., *et al.*: 'DeBAR: deflection based adaptive router with minimal buffering'. Proc. 2013 Design, Automation and Test in Europe Conf. and Exhibition (DATE), Grenoble, France, March 2013, pp. 1583–1588

[8] Joanna, G.R., Jose, J., Radhakrishnan, R., *et al.*: 'MinBSD: minimally buffered single cycle deflection router'. Proc. 2014 Design, Automation and Test in Europe Conf. and Exhibition (DATE), Dresden, Germany, March 2014, pp. 1–4

[9] 'Int. Technology Roadmap for Semiconductors Interconnect, 2009', available at http://public.itrs.net/, accessed 20 April 2016

[10] Feng, C., Lu, Z., Jantsch, A., *et al.*: 'Addressing transient and permanent faults in NoC with efficient fault-tolerant deflection router', *IEEE Trans. VLSI*, 2013, **21**, (6), pp. 1053–1066

[11] Rounge, A.: 'Fault tolerant network on chip based on fault aware flits and deflection routing'. Proc. NOC Symp., Vancouver, Canada, September 2015, pp. 9–16

[12] Fattah, M., Airola, A., Ausavarungnirun, R., *et al.*: 'A low overhead, fully distributed, guaranteed delivery routing algorithm for faulty network-on-chips'. Proc. NOC Symp., Vancouver, Canada, September 2015, pp. 1–8

[13] Rodrigo, S., Medardoni, S., Flich, J., *et al.*: 'Efficient implementation of distributed routing algorithms for NoCs', *IET Comput. Digit. Tech.*, 2009, **3**, (5), pp. 460–475

[14] Parikh, R., Bertacco, V.: 'uDIREC: unified diagnosis and reconfiguration for frugal bypass of NoC faults'. Proc. Int. Symp. Microarchitecture, Davis, CA, December 2013, pp. 148–159

[15] Bishnoy, R., Laxmi, V., Gaur, M.S., *et al.*: 'd2-LBDR: distance-driven routing to handle permanent failures in 2D mesh NoCs'. Proc. 2015 Design, Automation and Test in Europe Conf. and Exhibition, Dresden, Germany, March 2015, pp. 800–805

[16] Fick, D., DeOrio, A., Chen, G., *et al.*: 'A highly resilient routing algorithm for fault-tolerant NoCs'. Proc. 2009 Design, Automation and Test in Europe Conf. and Exhibition, Nice, France, April 2009, pp. 21–26

[17] Wachter, E., Erichsen, A., Amory, A., *et al.*: 'Topology- agnostic fault-tolerant NoC routing method'. Proc. 2013 Design, Automation and Test in Europe Conf. and Exhibition, Grenoble, France, March 2013, pp. 1595–1600

[18] Aisopos, K., DeOrio, A., Peh, L., *et al.*: 'ARIADNE: agnostic reconfiguration in a disconnected network environment'. Proc. Int. Conf. Parallel Architectures and Compilation Techniques, Texas, USA, October 2011, pp. 298–309

[19] Zhang, Z., Greiner, A., Taktak, S.: 'A reconfigurable routing algorithm for a fault-tolerant 2D-mesh network-on-chip'. Proc. ACM/IEEE Design Automation Conf., CA, USA, 2008, pp. 441–446

[20] Balboni, M., DeOrio, A., Peh, L., *et al.*: 'Synergistic use of multiple on-chip networks for ultra-low latency and scalable distributed routing reconfiguration'. Proc. 2015 Design, Automation and Test in Europe Conf. and Exhibition, Grenoble, France, March 2015, pp. 806–811

[21] Bose, P., Morin, P., Stojmenovic, I., *et al.*: 'Routing with guaranteed delivery in *ad hoc* wireless networks', *Wirel. Netw.*, 2001, **7**, (6), pp. 609–616

[22] Dally, W.: 'Virtual-channel flow control', *IEEE Trans. Parallel Distrib. Syst.*, 1992, **3**, (2), pp. 194–205

[23] Zou, Y., Pasricha, S.: 'NARCO: neighbor aware turn model-based fault tolerant routing for NoCs', *IEEE Embedded Syst. Lett.*, 2010, **2**, (3), pp. 85–89

[24] Iordanou, C., Soteriou, V., Aisopos, K.: 'Hermes: architecting a top-performing fault-tolerant routing algorithm for networks-on-chips'. Proc. IEEE Int. Conf. Computer Design, Seoul, Korea, October 2014, pp. 424–431
[25] Sleeba, S.Z., Jose, J., Mini, M.G.: 'WeDBless: weighted deflection bufferless router for mesh NoCs'. Proc. Great Lakes Symp. VLSI, Houston, TX, USA, May 2014, pp. 77–78
[26] Feng, C., Lu, Z., Jantsch, A., *et al.*: 'FoN: fault-on-neighbor aware routing algorithm for networks-on-chip'. Proc. SoC Conf., Incheon, Korea, September 2010, pp. 441–446
[27] Lee, J., Nicopoulos, C., Sung, J.P., *et al.*: 'Do we need wide flits in networks-on-chip?'. Proc. IEEE Computer Society Annual Symp. VLSI, Natal, Brazil, August 2013, pp. 2–7
[28] 'Booksim 2.0 User's Guide, 2010', available at http://nocs.stanford.edu, accessed 01 February 2014
[29] Henning, J.: 'SPEC CPU 2006 benchmark descriptions'. SIGARCH Computer Architecture News, 2006
[30] Ubal, R., Sahuquillo, J., Petit, S., *et al.*: 'Multi2Sim: a simulation framework to evaluate multicore-multithreaded processors'. Proc. Intl. Symp. Computer Architecture and High Performance Computing, Brazil, October 2007, pp. 62–68
[31] Kahng, A.B., Li, B., Peh, L., *et al.*: 'Orion 2.0: a fast and accurate NoC power and area model for early stage design space exploration', *IEEE Trans. VLSI*, 2012, **20**, (1), pp. 191–196