A new Fault-tolerant and Congestion-aware Adaptive Routing Algorithm for Regular Networks-on-Chip

Hamed S. Kia, and Cristinel Ababei
Department of Electrical and Computer Engineering
North Dakota State University
Fargo ND, 58108-6050
Email: {hamed.sajjadikia, cristinel.ababei}@ndsu.edu

***Abstract*—In this paper, we propose a new fault-tolerant and congestion-aware adaptive routing algorithm for Networks-on-Chip (NoCs). The proposed algorithm is based on the ball-and-string model and employs a distributed approach based on partitioning of the regular NoC architecture into regions controlled by local monitoring units. Each local monitoring unit runs a shortest path computation procedure to identify the best routing path so that highly congested routers and faulty links are avoided while latency is improved. To dynamically react to continuously changing traffic conditions, the shortest path computation procedure is invoked periodically. Because this procedure is based on the ball-and-string model, the hardware overhead and computational times are minimal. Experimental results based on an actual Verilog implementation demonstrate that the proposed adaptive routing algorithm improves significantly the network throughput compared to traditional XY routing and DyXY adaptive algorithms.**

***Index Terms*—Networks on chip, Dynamic routing algorithm, Fault tolerance.**

I. Introduction

The Network-on-Chip (NoC) concept replaces design-specific global on-chip wires with a generic on-chip interconnection network realized by specialized routers that connect generic processing elements (PE). It represents a paradigm change from computation to communication centric design for Systems-on-Chip (SoCs) [1], [2]. The benefits of the NoC based SoC design include scalability, predictability, and higher bandwidth with support for concurrent communications.

Data are transfered between PEs organized as packets along paths computed by the routing algorithm. There are two types of routing strategies: deterministic and adaptive routing. In deterministic routing, the routing path is completely determined by the source and destination addresses. Its advantages include the simplicity of the router architecture and the deadlock free property. Due to the simpler hardware logic, deterministic routing offers lower flit latency when the NoC is not congested. However, as the packet injection rate increases and some of the links and routers become congested, deterministic routing is likely to suffer from throughput degradation as it can not dynamically respond to network congestion [3]. In addition, permanent link failures may render the NoC inoperable. In contrast, adaptive routing takes into consideration the traffic variations in the network and computes dynamically alternative paths to route data via less congested regions. Moreover, if the NoC architecture is equipped with link failure detection mechanisms, adaptive routing can address these failures and thereby facilitate fault tolerance [4], [5]. However, due to the hardware overhead to implement the detection mechanisms and to compute good routing paths, adaptive routing has a higher latency at low levels of network congestion. Also, dynamic routing has to be designed so that it ensures deadlock free property.

II. Related Work and Contribution

Adaptive routing has attracted a lot of attention recently. The DyAD routing algorithm proposed in [6] is a hybrid approach, which switches between deterministic routing at low packet injection rates and dynamic routing when the network congestion increases. An adaptive routing architecture based on a dynamic programming (DP) network to provide optimal path planning is proposed in [7]. It has introduced a scalable k-step look ahead routing strategy to reduce routing tables storage and to maintain a high quality of adaptation. The routing method in [8] utilizes information from all routers in the source-target path to perform traffic routing. The source units use the information on network conditions to adjust the parameters that configure the path to the target router. A dynamic routing algorithm based on monitoring the congestion status of the neighboring routers is studied in [3]. In [9] the objective is to route packets to their destination using a path that is as free as possible of congested nodes. The algorithm tries to use the situations of indecision occurring when the routing function returns several admissible output channels. In [10] a centralized monitoring system is used to locate congested links and detour them. However the proposed method is not scalable to the hundreds of cores that may soon be integrated on a SoC. Authors in [11] have proposed an adaptive routing scheme where intermediate routers make decisions locally depending on the available bandwidth in each direction to the neighboring routers and on the distance between current and the destination routers. A congestion aware routing algorithm, which sends congestion monitoring

values in parts of the network beyond adjacent routers, is proposed in [12].

Several papers focusing on fault tolerant routing algorithms have recently been published. Reconfigurable architectures have been employed in several papers to address faults. The Vicis NoC architecture proposed in [13] can tolerate the loss of routers and links due to wearout induced hard faults. Network level reconfiguration is implemented by rewriting the routing tables based on the information from the built-in-self-test (BIST) units in each router. A reconfigurable fault-tolerant deflection routing algorithm based on reinforcement learning for NoC has been proposed in [14]. The algorithm reconfigures the routing tables through reinforcement learning based on 2-hop fault information. In [15] a routing algorithm that boosts the robustness of interconnect networks by reconfiguration to avoid faulty components while maintaining connectivity and correct operation has been proposed. A lightweight fault-tolerant mechanism based on the notion of default backup paths (DBPs) has been proposed in [16]. It uses nominal redundancy to maintain network connectivity of healthy NoC routers and on-chip PEs in the presence of hard failures. Most previous works on adaptive routing report simulation results achieved with simulators developed in a programming language (e.g., C++, SystemC). Therefore, they typically do not report actual area overheads due to the lack of actual hardware implementation details. In addition, they address either congestion issues or errors (e.g., transient, intermittent, permanent failures).

In this paper, we develop a new distributed adaptive algorithm designed to address both congestion and link failures. Our main contribution is as follows: (i) We develop a new NoC architecture which partitions the regular NoC architecture into regions controlled by local monitoring units. Each local monitoring unit runs a shortest path computation procedure to identify the best routing path so that highly congested routers or failed links are avoided, (ii) We propose the use of a ball-and-string model based shortest path computation method, which together with the decentralized region based routing approach leads to minimal hardware overhead, and (iii) The proposed NoC architecture and routing strategy are implemented in Verilog with Virtex 5 as the target FPGA fabric. The experimental results on multimedia benchmarks demonstrate the ability of the proposed routing algorithm to significantly improve the network throughput.

III. Proposed Adaptive Routing

In this section we describe the proposed dynamic routing algorithm based on the ball-and-string model for regular mesh NoC topologies.

A. General NoC Topology Description

For simplicity, we assume a regular mesh NoC topology to describe and apply the proposed dynamic routing algorithm. However, the proposed routing algorithm can also be extended to irregular NoCs. Regular mesh NoCs are 2D arrays of routers. Adjacent routers are connected via bi-directional links or channels. An example of a 3×3 mesh NoC is shown in Fig.1.a. The router has a pipelined architecture where routing computation (RC), virtual channel allocation (VA) and switch allocation (SA), and switch traversal (ST) are the main pipeline stages. The block diagram of the router is shown in Fig.1.b. To minimize the required buffering space, in this paper we assume wormhole switching. The router architecture will be modified to add support for the adaptive routing – this will be described in a later section.

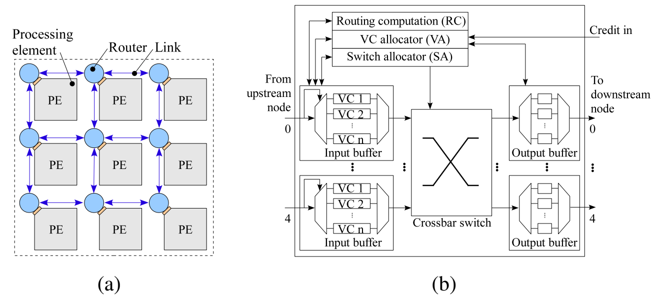

Fig. 1. (a) Example of 2D regular mesh. (b) Typical router architecture.

B. Ball-and-String Model based Shortest Path Computation Procedure

The main idea of shortest paths computation is (1) to associate a directed graph $G(V, E)$ with the NoC topology, (2) to assign edge weights proportionally to congestion, and (3) to develop a procedure to find the shortest paths in this graph for any given source node.

To compute edge weights we propose to use buffer occupancies, which are readily available in a typical NoC. More specifically, the weight is computed as the summation of the numbers of memory slots used in the output buffers of the upstream router and of memory slots used in the input buffers of the downstream router. For example, Fig.2.a illustrates the computation of the edge weight of an individual link. Fig.2.b shows the edge weights for a graph associated with a 3×2 NoC. Formally, for the purpose of computing the shortest paths the edge weights $w_{ij}, i = 1, ..., |V|, j = 1, ..., |V|$ are computed as follows:

$$w_{ij} = \begin{cases} Used\ memory\ slots & \text{if } (v_i, v_j) \in E, \forall v_i, v_j \in V \\ \infty & \text{otherwise} \end{cases} \tag{1}$$

Each time the shortest path procedure is invoked, the shortest path for each source-destination communication pair will be computed so that the path cost is minimized:

$$Min: \qquad \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} y_{ij} \times w_{ij} \tag{2}$$

where y_{ij} is a binary variable, which indicates if the link $(v_i, v_j) \in E$ is used or not as a part of the path.

The procedure for the shortest path computation is based on the parallel shortest path searching algorithm proposed in [17], which is similar to the ball-and-string model studied

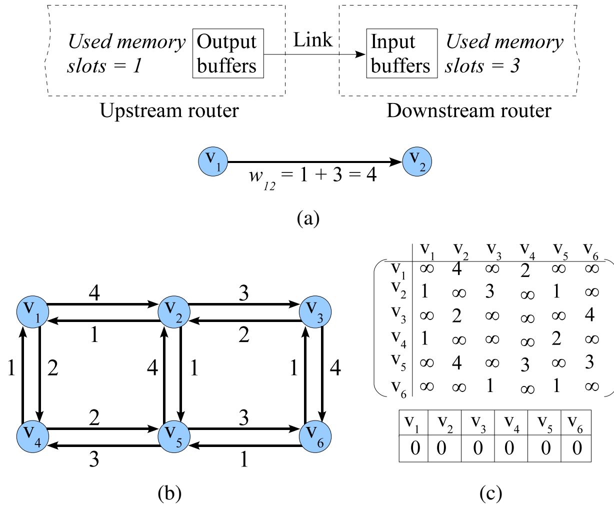

Fig. 2. (a) Computation of edge weight. (b) Edge weights for a network graph associated with a 3×2 NoC. (c) The network matrix A and the parent-array of the network graph.

in [18], [19]. The shortest path procedure will identify the best paths for packets to travel to their destinations under the congestion conditions that exist at the time of edge weights computation. To account for the changes in these conditions (due to the changes in network traffic) the procedure will be called periodically multiple times. Therefore, it is important for the implementation of such a procedure to be fast and to require minimal hardware resources. Our custom implementation of this algorithm utilizes the adjacency matrix $[A]_{n\times n}$ of the network graph – referred to as the *network matrix* because each entry stores the corresponding edge weight (i.e., $a_{ij} = w_{ij}$). In addition, it utilizes a specialized array – referred to as the *parent-array* – which stores the IDs of predecessor node (or the parent node) of each node along the shortest path. For example, the network matrix and the parent-array initialized to zero of the network graph from Fig.2.b are shown in Fig.2.c.

To illustrate how the algorithm works, we use the example from Fig.2.b. Let us assume node v_1 as the source. In the first step of the algorithm, all entries in the first column of the network matrix are set to infinity. Also, the minimum value in the first row is found and then subtracted from each entry of the first row (see Fig.3.a). Because $a_{14} = 0$, the shortest path to v_4 is already determined and v_1 is recorded in the forth column of the parent-array. Then, all entries in the fourth column of the network matrix are also set to infinity as shown in Fig.3.a. In the next step of the algorithm, the minimum value among the entries of the first and fourth rows is identified and subtracted from the entries of these rows as shown in Fig.3.b. Because $a_{12} = 0$ and $a_{45} = 0$, the shortest paths to v_2 and v_5 are also determined at this time and these two nodes are recorded in the second and fifth columns of the parent-array. Also, all entries in the second and fourth columns are set to infinity. This process is repeated until all entries in the network matrix are set to infinity. At this time, all entries in the parent-

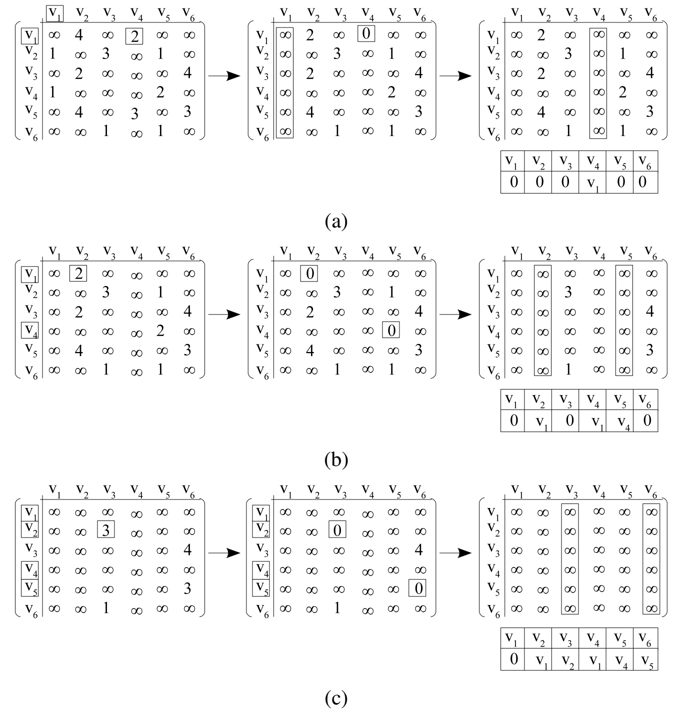

Fig. 3. Applying the algorithm to network matrix.

array store the predecessors of each node, which can be back-traced to construct the shortest path from the source v_1 to any node in the graph.

Fig.4 illustrates how the network matrix and the parent-array are implemented. To minimize the memory usage for storing and manipulating matrices, in our actual hardware implementation (described in detail in a later section) we use registers to store only the entries that are initially non-infinity in the network matrix. In other words, for example in Fig.2 we know that a_{16} and a_{61} will remain equal to infinity throughout the shortest paths computation process – due to the regular mesh NoC topology which tells us that there is no direct connection between nodes v_1,v_6. Therefore, there is no need to allocate and manipulate memory for these entries of the network matrix. However, we need to use two additional registers shown as "Fixed" and "Flag" in Fig.4. Each time when the entries of a given column must be set to infinity only the corresponding entry of the "Fixed" register is set to 1. This eliminates the need for the infinity value, which is difficult to define in a simple hardware implementation. The "Flag" register is used to mark the rows which are processed currently. For example, once it is known that the first and fourth rows are to be processed next, their corresponding entries in the "Flag" register are set to 1. In our hardware implementation, even though the design time was slightly longer this custom implementation of the network matrix reduced significantly the memory usage.

Using the additional registers, the pseudocode of the shortest path algorithm is presented in Algorithm 1.

Flag		v_1	v_2	v_3	v_4	v_5	v_6
Fixed		1	0	0	1	0	0
1	v_1		4		2		
0	v_2	1		3		1	
0	v_3		2				4
1	v_4	1				2	
0	v_5		4		3		3
0	v_6			1		1	

Fig. 4. Illustration of the hardware implementation of network matrix.

Algorithm 1: Shortest path computation

1: $\forall i, Fixed(i) \leftarrow 1, Flag(i) \leftarrow 1$ if node i is source, 0 otherwise
2: $[Parent]_{1\times n} \leftarrow [0]_{1\times n}$
3: Initialize network matrix $[A]_{n\times n}$ based on network graph
4: **while** $(Flag \neq [1]_{1\times n})$ **do**
5:   **for** $i \leftarrow 1$ to n **do**
6:     **for** $j \leftarrow 1$ to n **do**
7:       $A(i,j)^* = Flag(i) \times Fixed(j)' \times A(i,j)$
8:     **end for**
9:   **end for**
10:   Find the non-zero minimum of $A^*(i,j)$
11:   **for** $i \leftarrow 1$ to n **do**
12:     **for** $j \leftarrow 1$ to n **do**
13:       **if** $Flag(i) = 1$ **then**
14:         $A(i,j) = A(i,j) - minimum$
15:       **end if**
16:       **if** $A(i,j) = 0$ **then**
17:         $Flag(i) = 0, Fixed(j) = 0, Parent(j) = i$
18:       **end if**
19:     **end for**
20:   **end for**
21: **end while**

Fig. 5. The pseudocode of the shortest path computation procedure.

C. Adaptive Routing

To minimize the required extra hardware we propose a distributed (or decentralized) scheme for the implementation of the adaptive routing. The NoC is partitioned into several partitions (or regions) and each partition is managed by a local monitoring unit (LMU). LMUs represent the controllers responsible with routing packets that enter their partitions. For example, the 4×4 NoC from Fig.6 is partitioned into four equal regions controlled by four LMUs. Even though in this example the partitions have equal size, they may have different sizes too.

Each LMU is in charge with routing data to routers within its own partition and to the first-order neighboring routers adjacent to its own partition. To compute edge weights for links that connect routers from different partitions, adjacent LMUs are interconnected to be able to share information. For example in Fig.6, LMU_1 is responsible for routing packets injected within the partition formed by the routers $\{1, 2, 5, 6\}$ and the packets that arrive from adjacent routers $\{3, 7, 9, 10\}$. This LMU will implement the shortest path computation procedure described in the previous section, which will utilize the network matrix of the sub-graph corresponding to routers $\{1, 2, 3, 5, 6, 7, 9, 10\}$ and all edges between these routers except the two edges corresponding to the links between

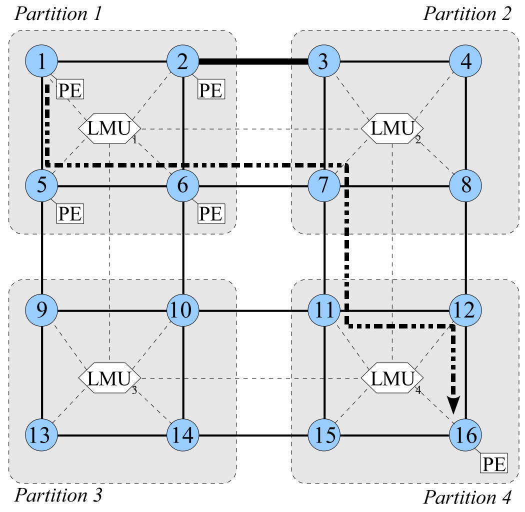

Fig. 6. NoC partitioned into four partitions controlled by four LMUs.

routers $\{3, 7\}$ and $\{9, 10\}$.

As an example consider a situation when a packet arrives to router 2 in partition 1 via the boundary-crossing link between routers $\{3, 2\}$ (shown in thicker line in Fig.6). In this case, the source node in the shortest path procedure will correspond to router 2. LMU_1 will extract the destination address from the header flit. If the destination router is located inside partition 1 or is one of the adjacent routers $\{9, 10\}$, then LMU_1, which has already computed the shortest paths, will update the header flit (Fig.9) with the shortest path routing information. If the destination is in partition 3, then the header flit will be updated with the routing information toward one of the routers $\{9, 10\}$ – to the one with the shortest path – and packets will be routed accordingly. Then, LMU_3 will be responsible with routing to the final destination inside partition 3.

As another example, let us consider the source-destination pair v_1, v_{16}. Because the destination is in partition 4, the algorithm will first find the shortest path to either of the routers $\{3, 7, 9, 10\}$ in partitions 2 and 3. Assuming that the shortest path is to router 7, the packets may be routed as shown in Fig.6. Then, LMU_2 will be responsible with routing packets toward partition 4. This will be done by utilizing the shortest path from the source 7 to either of the routers $\{11, 12\}$. If this path is to the router 11 as shown in Fig.6, then LMU_4 will be responsible with routing along the shortest path from the local source 11 to the final destination 16.

D. Addressing Link Failures

As CMOS fabrication technologies move to nano-scale feature sizes, integrated circuits become more susceptible to manufacturing faults, transient faults, and aging mechanisms that can lead to permanent faults. In NoC architectures without fault tolerance mechanisms, permanent link failures can render the NoC inoperable. The adaptive algorithm proposed in this paper, can easily address link failures and thereby facilitate fault tolerance. When a link failure is detected inside a given partition, the corresponding LMU can remove that link from

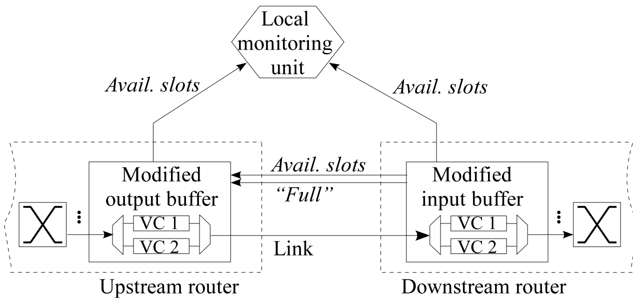

Fig. 7. Block diagram of the communication between two adjacent routers.

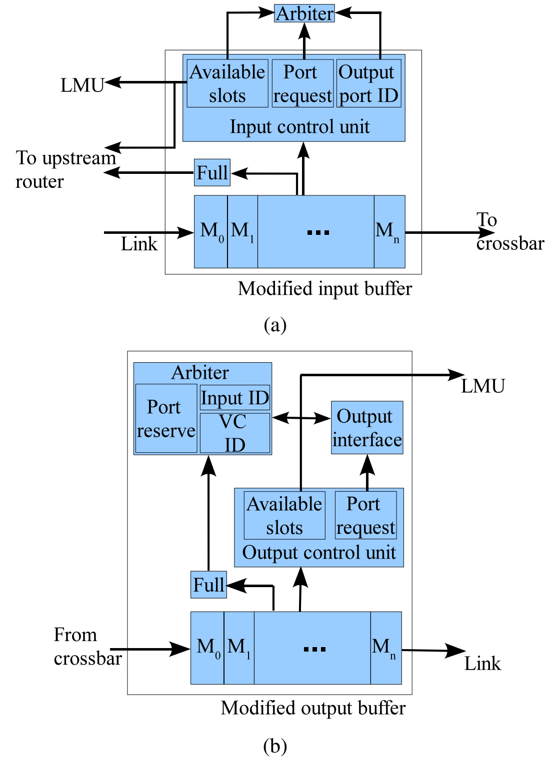

Fig. 8. (a) Block diagram of the input buffer. (b) Block diagram of the output buffer.

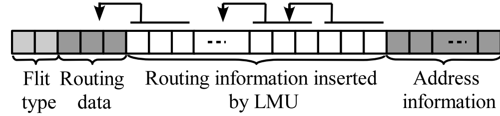

Fig. 9. Header flit description.

the adjacency matrix. Hence, the shortest path computation procedure will compute thereafter paths formed by the remaining healthy links only.

E. Deadlock

Deadlock occurs when packets are unable to move forward because they are waiting on one another to release resources (i.e., there is a cyclic dependency between packets). This is undesirable because it can paralyze the operation of the network. Therefore, routing algorithms must be designed so that deadlock is avoided [1]. While the proposed routing algorithm is not designed to directly guarantee the deadlock free property[1], it indirectly minimizes the likelihood of deadlock occurrence. If a deadlock situation occurs, the affected links (which do not see activity for long periods of time) can be interpreted as if they were congested or broken. Because, the proposed adaptive algorithm is called periodically, the new computed routing paths will avoid the affected links, thereby most likely eliminating the deadlock situation. In other words, in the event that a packet dependency occurs, it will be eliminated during the next call of the shortest path computation procedure, which will find a different path (using other links) along which packets can move forward.

IV. Hardware Implementation

The implementation of the adaptive routing requires changes in the NoC architecture. First, we added the local monitoring units and their connections as discussed in the previous section and as illustrated in Fig.7. Second, we design a new router architecture to provide support for the mechanics of the proposed routing algorithm as described below.

A. Modified Router Architecture

Because we use input and output buffers occupancies to compute edge weights, we modify the input and output buffers by adding local input and output control units as shown in Fig.8.

To minimize the router area, all buffers are implemented using registers instead of SDRAM structures. Input and output ports use 2 virtual channels. Messages are divided into packets, which are further divided into 3 flits (header, body, and tail). The header flit contains the routing information and destination address. Fig.9 shows the format of the header flit. Once a header flit arrives to the input port of a given router, the routing information (i.e., the output port ID to which the flit should be forwarded to) is provided by the local monitoring unit. When the header flit is received the routing bits are shifted as shown in Fig.9. Routers use the routing data bits to determine the output port. In this way the usage of routing tables is avoided. After gaining access to the output port and before the transfer to the output buffer is started, the input control unit (ICU) saves the destination port ID in the port ID control bits (see Fig.8.a) where it will be stored until after the tail flit of the same packet will traverse this router. The ICU also sets the "Port request" bit whenever a flit requests access to any output port.

At each positive edge of the clock the ICU computes the number of occupied slots in the input buffer. This information is sent to LMU, upstream router and arbiter. The "Full" bit (see Fig.8.a) is set to 1 when there is at least an empty slot available and this information is communicated to the upstream router.

Whenever there is an empty slot in the output buffers, the

[1]Specialized adaptive algorithms can be designed to guarantee the deadlock free property, but at the expense of increased complexity and larger area penalty [20], [21].

output control unit (OCU) sets the “Full” bit in Fig.8.b and sends this info to the arbiter, which continuously monitors the output buffers. If an empty slot is available in the output buffer the arbiter will check if it is reserved or not. When both virtual channels of an output port are available the arbiter will select and grant access first the one with more empty slots. When the output buffer receives a header flit the arbiter will set the reserved bit to logic 1, which will be kept until after the tail flit will be received. The OCU also computes the number of used buffer slots (i.e., the output buffer occupancy), which is sent to the local monitoring unit, arbiter, and output interface unit. In addition, OCU also sets the “Request port” bit whenever a flit requests access to physical link. This bit is continuously monitored by the output interface unit.

The output interface unit shown in Fig.8.b functions as an arbiter. A packet in the output buffer will be sent to a virtual channel with more empty slots in the input port of the downstream router. When both output virtual channels compete over the physical link, the output interface unit will select and grant access first to the VC that has the least available memory. When the input buffer of the downstream router receives a header flit it will be marked as being reserved.

The shortest path computation requires eight clock cycles. Therefore, every other eight clock cycles the LMUs update the shortest paths. This period is small enough to ensure rapid response to changes in traffic as observed in our experiments.

V. Experimental Results

To validate and test the proposed adaptive algorithm, we have coded in Verilog a 4×4 NoC prototype with an architecture similar to that shown in Fig.6. The NoC design is synthesized using the Xilinx ISE compiler [22] and the RTL implementation is verified via dynamic simulation. The target hardware platform is a Virtex 5 FPGA. The ISE tool is utilized to estimate total area. The static timing analysis feature of the ISE tool is used to measure and compute the average flit latency.

A. Adaptive Routing to Address Congestion

In the first set of experiments, we compare the proposed adaptive routing algorithm against the traditional XY routing. We also compare the proposed routing algorithm against DyXY adaptive algorithm [3] due to its popularity and ease of implementation. Even though DyXY was studied only based on simulations, we have implemented it in Verilog using our own adapted router architecture. The hardware implementation of other previously proposed adaptive routing algorithms is not available. Moreover, because of the complexity of these adaptive algorithms, their Verilog implementation is very challenging. Therefore, we restrict our experiments to comparisons against XY and DyXY algorithms.

We report our results for two multimedia benchmarks. The communication task graph (CTG) and optimized mapping of the first benchmark are from [23] and are shown in Fig.10. The injected traffic at all sources of the CTG is generated by local generators. The average number of injected packets at each

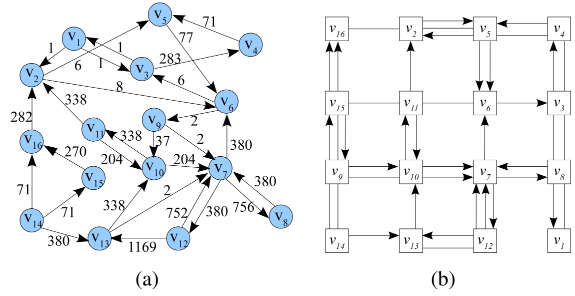

Fig. 10. CTG and optimized mapping of the first multimedia benchmark.

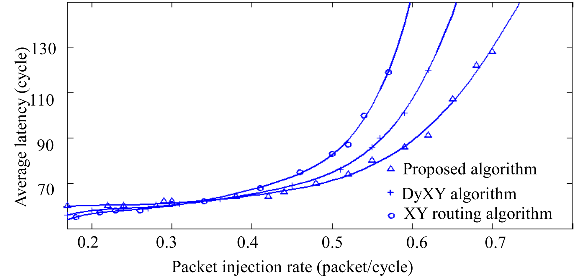

Fig. 11. Comparison of the average latency achieved by the proposed adaptive routing, the traditional XY routing algorithm and DyXY algorithm for the first benchmark.

source is proportional to the communication volume of each source-destination pair shown in Fig.10.a. The average latency is computed under the assumption that packets are consumed immediately upon arrival to their destinations. Fig.11 presents the average latencies achieved using the proposed adaptive routing algorithm, the traditional XY routing algorithm, and DyXY algorithm respectively. It can be observed that the proposed adaptive routing improves the network throughput at high packet injection rates. However, at low packet injection rates latency is slightly degraded compared to XY routing due to the delay penalty incurred in the hardware for adaptive routing support.

The communication task graph (CTG) and optimized mapping of the second benchmark are from [24] and are shown in Fig.12. Again the proposed adaptive routing improves the network throughput at high packet injection rates (Fig.13). The improvement in throughput and the extra hardware needed to implement the proposed algorithm is shown in Table I. Throughput is defined at the point where the latency is twice as the low packet injection rate latency. The extra hardware to implement the proposed algorithm is around 17%, which is less than 42% of [13]. The area penalty is slightly higher than the area penalty for the DyXY routing algorithm. However, note that DyXY routing algorithm is designed to address only congestion, while the proposed routing algorithm addresses both congestion and link failures for fault tolerance.

B. Adaptive Routing to Address Link Failures

In this section we investigate the performance of the proposed adaptive routing algorithm in the presence of link failures. We investigate the fault tolerance of the proposed routing

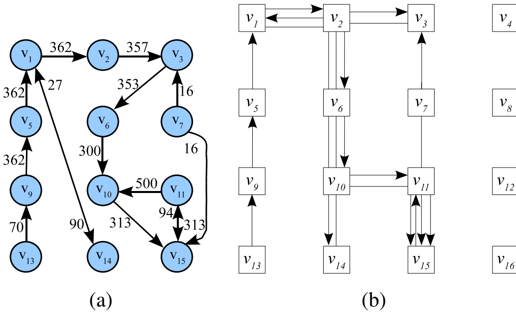

Fig. 12. CTG and optimized mapping of the second multimedia benchmark.

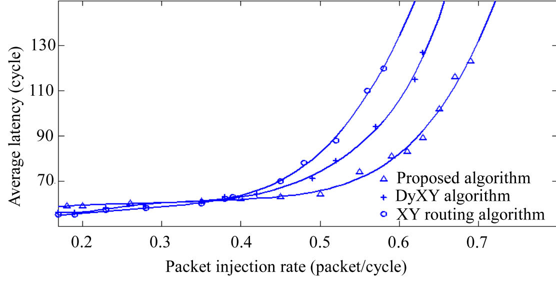

Fig. 13. Comparison of the average latency achieved by the proposed adaptive routing and the traditional XY routing algorithms for the second benchmark.

TABLE I
COMPARISON AGAINST XY ROUTING

Routing algorithm	Extra hardware	Test Case	Throughput improvement
Proposed algorithm	17 %	Test case 1	21%
		Test case 2	20%
DyXY algorithm	12 %	Test case 1	11%
		Test case 2	10%

algorithm for a number of injected link failures varied between 1 and 4. For each of these numbers, we randomly inject link failures several times and then we compute the average throughput. To keep the CPU computational runtimes of ISE tool within reasonable limits we study a simpler testcase whose mapping is shown in Fig.14. Each of the injected set of faults are handled by the proposed algorithm as described in Section III.D. The network throughput degradation as a function of the number of injected faults is shown in Fig.15. It can be noted that the network throughput degrades gracefully, which demonstrates the ability of the proposed routing algorithm to address link failures.

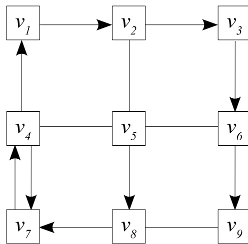

Fig. 14. The mapping of the third benchmark.

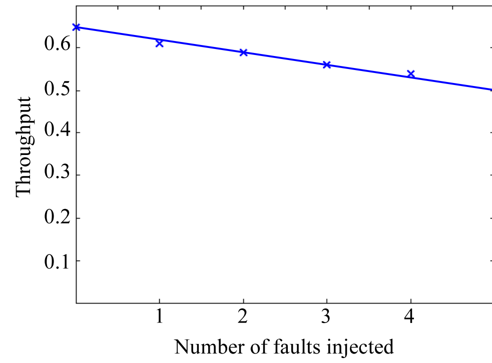

Fig. 15. Throughput depredation for different amount of fault injection.

VI. CONCLUSION

We proposed a new fault-tolerant and congestion-aware adaptive routing algorithm for NoCs. To implement the proposed algorithm the NoC architecture is partitioned into regions controlled by local monitoring units. Each local monitoring unit runs a shortest path computation procedure to identify the best routing path so that highly congested routers are avoided. To dynamically react to continuously changing traffic conditions the procedure is invoked periodically. Because the procedure is based on the ball-and-string model, the hardware overhead and computational times are minimal. Experimental results based on an actual Verilog implementation demonstrate that the proposed adaptive routing algorithm improves significantly the network throughput compared to traditional XY routing and DyXY adaptive algorithms.

REFERENCES

[1] W. J. Dally, and B. Towles, Principles and Practices of Interconnection Networks, Morgan Kaufmann, 2004.

[2] G.D. Micheli, and L. Benini, Networks on Chips: Technology and Tools, Morgan Kaufmann, 2006.

[3] M. Li, Q.A. Zeng, and W.B. Jone, "DyXY: a proximity congestion aware deadlock-free dynamic routing method for network on chip," *ACM/IEEE Design Automation Conference (DAC)*, 2006.

[4] R. Marculescu, "Networks-on-chip: the quest for on-chip fault-tolerant communication," *IEEE Computer Society Annual Symposium on VLSI*, 2003.

[5] M. Yang, T. Li, Y. Jiang, and Y. Yang, "Fault tolerant routing schemes in RDT(2,2,1)/a-based interconnection for networks on chip designs," *Int. Symposium on Parallel Architectures, Algorithms and Networks*, 2005.

[6] J. Hu and R. Marculescu, "DyAD: smart routing for networks-on-chip," *ACM/IEEE Design Automation Conference (DAC)*, 2004.

[7] T. Mak, P.Y.K. Cheung, W. Luk, and K.P. Lam, "A DP-network for optimal dynamic routing in Network-on-Chip," *ACM/IEEE Int. Conference on Hardware Software Codesign*, 2009.

[8] L. Tedesco, F. Clermidy, and F. Moraes, "A path-load based adaptive routing algorithm for Networks-on-Chip," *ACM Annual Symposium on Integrated Circuits and System Design*, 2009.

[9] G. Ascia, V. Catania, M. Palesi, and D. Patti, "Implementation and analysis of a new selection strategy for adaptive routing in networks on chip," *IEEE Trans. on Computers*, vol. 57, no. 6, pp. 809-820, 2008.

[10] F. Ge, N. Wu, and Y. Wan, "A network monitor based dynamic routing scheme for network on chip," *IEEE Asia Pacific Conference on Microelectronics and Electronics*, 2009.

[11] M.A. Al Faruque, T. Ebi, and J. Henkel, "Run-time adaptive on-chip communication scheme," *ACM/IEEE Int. Conference on Computer Aided-Design (ICCAD)*, 2007.

[12] P. Gratz, B. Grot, and S.W. Keckler, "Regional congestion awareness for load balance in networks-on-chip," *IEEE Int. Symposium on High Performance Computer Architecture*, 2008.
[13] D. Fick, A. DeOrio, J. Hu, V. Bertacco, D. Blaauw, and D. Sylvester, "Vicis: a reliable network for unreliable silicon," *ACM/IEEE Design Automation Conference (DAC)*, 2009.
[14] C. Feng, Z. Lu, A. Jantsch, J. Li, and M. Zhang, "A reconfigurable fault tolerant deflection routing algorithm based on reinforcement learning for network-on-chip," *Int. Workshop on Network on Chip Architectures (NocArc)*, 2010.
[15] D. Fick, A. Deorio, G. Chen, D. Sylvester, and D. Blaauw, "A highly resilient routing algorithm for fault tolerant NoCs," *ACM/IEEE Design Automation and Test in Europe Conf. (DATE)*, 2009.
[16] M. Koibuchi, H. Matsutani, H. Amano, and T.M. Pinkston, "A lightweight fault tolerant mechanism for Network-on-Chip," *ACM/IEEE Int. Symposium on Networks-on-Chip (NoCS)*, 2008.
[17] H. Ishikawa, S. Shimizu, Y. Arakawa, N. Yamanaka, and K. Shiba, "New parallel shortest path searching algorithm based on dynamically reconfigurable processor DAPDNA-2," *IEEE Int. Conference on Communications*, 2007.
[18] P. Narvaez, K.Y. Siu, and H.Y. Tzeng, "New dynamic SPT algorithm based on a ball-and-string model," *ACM/IEEE Trans. on Networking (TON)*, vol. 9, no. 6, pp. 706-718, 2001.
[19] T. Shi and J.J. Lee, "An O(L) parallel shortest path algorithm," *Int. Conference on Computer Design (CDES)*, pp. 119-124, 2009.
[20] A.D. Choudhury, G. Palermo, C. Silvano, and V. Zaccaria, "Yield enhancement by robust application-specific mapping on Network-on-Chips," *Int. Workshop on Network on Chip Architectures (NocArc)*, 2009.
[21] C. Seiculescu, S. Murali, L. Benini, G. De Micheli, "A method to remove deadlocks in Networks-on-Chips with wormhole flow control," *ACM/IEEE Design Automation and Test in Europe Conf. (DATE)*, pp. 1625-1628, 2010.
[22] Xilinx ISE Tools, http://www.xilinx.com
[23] M. Lai, L. Gao, N. Xiao, and Z. Wang, "An accurate and efficient performance analysis approach based on queuing model for Network on Chip," *ACM/IEEE Int. Conference on Computer-Aided Design (ICCAD)*, 2009.
[24] E.B. van der Tol, E.G.T. Jaspers, "Mapping of MPEG-4 decoding on a flexible architecture platform," *SPIE, Media Processors*, pp. 1-13, 2002.