Fault-Tolerant Adaptive Routing Algorithm for Network-on-Chip

Tan Bie, Yang Jiao, Zixin Wang and Rong xu

***Abstract*—For many-core chip multiprocessors (CMPs), Network-on-chip (NoC) provide high performance on chip communication and great scalability while the choice of routing algorithm plays a vital role in the performance of on-chip interconnection networks. In general, adaptive routing utilize information about the network state to select among alternative path options and offer better performance in term of the latency and throughout. However, recently published adaptive routing algorithm don't equip with a well-designed fault tolerant mechanism to handle potential link failures in the network, which induced by rapidly incensement of the circuit density as well as the extreme transistor scaling.**

Thus in our project, we propose and implement an adaptive routing algorithm using global congestion information and a runtime fault tolerant algorithm to solve multiple permanent link errors in the network. Escaped virtual channels and Up/Down restriction are applied for deadlock free.

***Index Terms*— NoC, Adaptive routing algorithm, global congestion, fault tolerance, deadlock-free**

I. INTRODUCTION

Network-on-Chip (NoC) has become the most significant communication fabric for many-core chip multiprocessors (CMPs). Also, the routing algorithms used in these networks play a vital role in determining processor performance. Meanwhile, On-chip circuits are vulnerable to errors due to transistor geometric shrinking and performance improvement, leading to serious reliability issues.

Considering the problem above, some good fault-tolerant routing algorithms have been proposed while they didn't quite consider the loading balance of network [6], leading to longer packet latencies and potential performance loss. While for routing algorithms like regional Congestion Awareness (RCA) [3] and Destination-based adaptive routing (DAR) [4], they gain good improvements on network loading balance and packet latencies by applying congestion information. However they don't equipped with a fault tolerance mechanism. Upon any link failure occurring, such routing algorithms may induce huge performance overhead and even lead the whole system to error states. Thus based on this situation, we hope to design a routing algorithm which will not only be fault tolerant, but also consider the network congestion state to improve the routing performance by adaptive path selection.

In our project, we proposed and implemented a congestion-aware adaptive routing algorithm based on the network spatial information while deadlock-free and fault-tolerant features are also ensured. The proposed routing algorithm collect the global congestion information for each node in the network and adjust path selection according to the downstream link delay. A better network balance and shorter packet latency can be achieved by applying our routing algorithm. Besides, we also add a runtime on-chip fault-tolerant mechanism to handle permanent link failures in the network by deploying routing tables and logic that are updated upon each fault occurrence. Moreover, our routing algorithm also ensure a deadlock-free configuration by using escape Virtual Channels. Finally, our project is verified and analyzed on BookSim simulator.

II. PROPOSED ROUTING ALGORITHM

Modern Network-on-chip routing algorithm could be classified into deterministic routing and adaptive routing. Different from deterministic routing where packets from a source to a destination follow the same and fixed path, the adaptive routing utilize information about network state to select among alternative path options. By utilizing these information, a good selection function is able to spread the traffic and make network load more balanced.

In our project, we focus on adaptive routing using only minimal paths in a 2D mesh topology because of its simplicity and lower latency. The congestion information generation, fault-tolerant reconfiguration and the implementation of deadlock avoidance are discussed in details below.

2.1 Global-Congestion Adaptive Routing

Traditional adaptive routing algorithms relied on local or regional congestion state to adjust the path selection function. However, such methods still face a difficult challenge of balancing remote and local congestion state and may not always accurately reflect the load on the actual paths a packet can take to its destination.

Thus we implement a Global-Congestion adaptive routing so that every node in the network measures and maintains per-destination congestion state in the form of average delays to all other destination nodes through the possible output ports which are allowed by the minimal routing. Besides, the measured delays propagate from the destination to the source to update every node through permitted paths and thus more accurately estimating the congestion along paths. Then for every node, a set of traffic spilt ratios in which traffic for a specific destination is calculated based on combing the measured delays propagated from downstream routers and its local delay. The selection function of our routing algorithm use these ratios to decide which path to follow for a specific destination when a packet arrive at this node

Here we assume that:

1. A router only decides the distribution of traffic to its next-hop routers.
2. The ratios are per-destination basis, i.e., for a given node, all arrived packets destined for the same node use same ratio while packets using the same output ports but going to different destinations will be distributed independently by different ratio.
3. Minimal routing is used in our algorithm, thus for every node, there are at most two ports to a destination and the sum of port ratios for a destination equals to one and if there is only one permitted output port, all traffic is forced to be routed on that port.

A. *Distributed delay measurement and propagation*

Next, we illustrate the measurement and the propagation of the global delay information using an example in a 4x4 mesh topology in **Figure 1** (a). Assume all nodes in the network need to measure the delay to node 9.

Firstly, each node periodically estimates the local waiting time in the input queues for all five output ports. For every output port, this time is considered as the local queuing delay *l*[p] through port p and is approximated by counting the number of flits in the input buffers which have already requested a virtual channel to the next-hop router.

Then at the 1st clock cycle, delay from node 9 to itself is just the queuing delay on the ejection port of node 9. $Avg_9[9]$ stands for the average delay from node 0 to itself and equal to:

$$Avg_9[9] = l[Ej] \quad (1)$$

This delay information $Avg_9[9]$ is then propagate to all neighbors of node 9 at 2nd clock cycle. Node 8, 10, 5 and 13 receive $Avg_9[9]$ through their east (E), west (W), south (S) and north (N) ports respectively, as shown in **Figure 1** (b). Each of these nodes estimate their delay to node 9 by adding $Avg_9[9]$ with their locally measured delays on the port leading to node 9. For instances, at node 10, only west port could go to node 9 and the average delay from node 10 to node 9 is given as:

$$Avg_{10}[9] = l[W] + Avg_9[9] \quad (2)$$

Upon all one-hop routers finished the measurements of path delay, at 3rd clock cycle all two-hop routers 12, 14, 11, 4, 6 and 1 receive updates for the delay to node 9. For instances, node 6 receives updates about the average delay to node 9 from nodes 5 and 10 connected to the north and west port respectively. Then node 6 could estimate its average delay by computing a weighted mean of the delays through the north and west ports, the weights given by the traffic split ratio along these ports at node 6.

$$A[N][9] = Avg_{10}[9] + l[N] \quad (3)$$

$$A[W][9] = Avg_5[9] + l[W] \quad (4)$$

$$Avg_6[9] = W[N] * Avg_{10}[9] + W[W]Avg_5[9] \quad (5)$$

Here, $A[N][9]$ and $A[W][9]$ represent the delay through north and west ports respectively and W[N] and W[W] stand for the traffic split ratio at node 6 to destination node 9.

Carrying on in this manner, after some clock cycles all nodes in the network are able to measure their delay to node 9 through candidate output ports permitted by the minimal routing. This process will repeat periodically to ensure that the global congestion information stored in nodes are always up-to-date.

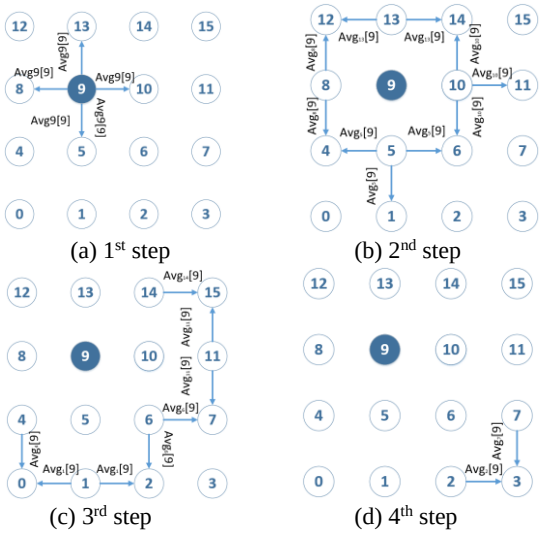


Figure 1 Example of the Distributed delay propagation

B. *Adaption of traffic split ratio*

The purpose of the traffic split ratio is to use the global congestion information, which are measured and propagated to each node, to uniformly balance the traffic load in the whole network. For each node in the network, the adaption process of the per destination traffic split ratios will be triggered upon the delay information from valid downstream routers is received by the current node. The same adaption algorithm will be repeated for all nodes in the network.

Suppose at node *i*, there are two output ports p_x and p_y connected to the destination *j* along paths which are permitted by the minimal routing. As we discussed at part A, A[x][j] and A[y][j], which are the delay to node j through ports p_x and p_x respectively, could be estimated by the current node. Here, we assume that the delay from x port is higher that from y port, which means that

$$A[x][j] > A[y][j]$$

Then we use these information to update our traffic spilt ratio with the below equations.

$$\Delta = \min\left(0.25 * \left(\frac{A[x][j] - A[y][j]}{A[x][j]}\right), W[x][j]\right) \quad (6)$$

$$W[x][j]_{new} = W[x][j] - \Delta; W[x][j]_{new} = W[x][j] + \Delta \quad (7)$$

The basic idea of the above equation is to increase the traffic split ratio of the port with lower downstream delay and decrease the ratio of the ports with higher delay. To avoid ratios becoming negative, we chose the minimal value between the ratio difference and current higher ratio.

2.2 Runtime Fault tolerant mechanism

The mechanism to handle with soft/permanent faults in the network during the runtime is necessary for modern routing algorithm to deal with potential hard errors in the lifetime. And in our project, we propose and implement a runtime mechanism to cope with the potential permanent link failures.

Since the broken links always mean a topology change, the original routing table may lead to error state and reconfiguration is necessary to ensure the complete reachability for all surviving nodes. In general, there are two families based on their method of the reconfiguration. One is deploying the routing tables and logic that are updated upon each fault occurrence in the runtime.

The second solution based on the offline software to complete the reconfiguration upon any fault link detected and then communicate with surviving topology with a central node. Our solution is built based on the first family while combing with the global congestion information forwarding. And we assume that when a link failure occurs, the node connected with that link will detect this fault and stop the new packet/flit injection until the reconfiguration is finished. The routing table reconfiguration works as follows:

Firstly, if a link error is detected, every node in the network works as a root node, starting to broadcast a reconfiguration flag to all other nodes in the network only through the healthy links hop-by-hop. Meanwhile the delay measurement and propagation process as we discussed in 2.1 is also initialed at this node so the delay information *Avg*[i] are also transmitted.

Then, for each node received the reconfiguration flag:

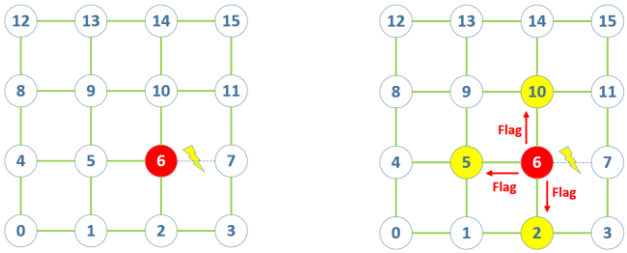
- Stall the router pipeline. If receiving a reconfiguration flag, that node should stop the pipeline and freeze the virtual channel allocation & switch allocation until the reconfiguration complete for all nodes.
- Update the routing table. For ports receiving the flag, calculate and store the new traffic split ratio W[x][i] based on the propagated delay information from downstream nodes. For ports not receiving the flag, invalid current split ratio and set to zero. Then calculate the average delay from current node to the root node. This step provides the safe paths as well as the global congestion information for the current node. This step is illustrated in **Table 1**.
- Flag forwarding. Nodes send the reconfiguration flag to its neighbors only through those ports which didn't receive a flag or connect to a faulty link.

For nodes detecting a permanent link error, repeat the above process to obtain an updated routing table with safe paths from other nodes to this faulty node as well as the network congestion information, which is used to select these safe paths adaptively.

This reconfiguration algorithm makes use of some ideas of our global congestion propagation process, both transmitting information from one destination to every possible source. Thus if any link error occurs, the reconfiguration process co-work with distributed delay propagation to obtain fully reachability to all surviving as well as the global congestion states. **Figure 2** illustrates an example while one link break in a 4x4 mesh topology network.

Table 1 Traffic split ratio update based on the flag signal and delay information during reconfigurations

Destination (i)	West	North	East	North
Ratio (W)	0.6→0.55	0.4→0	0→0.45	0
Flag received	Yes	No	Yes	No



(a) 1st step (b) 2nd step

(c) 3rd step (d) 4th step

Figure 2 Example of the reconfiguration process

III. DEADLOCK RECOVERY MECHANISM

We use the escape virtual channel to realize the deadlock-free feature in GCA. The key idea for it is to provide an escape path (escape virtual channel) for every deadlock packet. The routing algorithm for the escape path should be deadlock-free. Thus, when a packet is checked to be stuck in deadlock, we can send it on to the escape path and then the packet can use this deadlock-free path to its destination.

A. *How escape virtual channel works:*

The approach to dealing with deadlock is not to avoid it, but rather to recover from it. There are two key phases to any deadlock recovery algorithm: detection and recovery [1]. And in our algorithm, we'd like to separate it into three stages: Detection, Filtering and Recovery.

1. Detection:

In the detection phase, the network must be able to detect if itself has reached a deadlock situation. Determining exactly whether the network is in deadlock requires finding a cycle in resource wait-for graph. It's difficult and costly, so we use a conservative detection mechanism - timeout counters. Each input port of the router will be equipped with a timeout counter. There are only two cases that we will reset the counter: (1) when the input port receives a flit, (2) when we detect the deadlock and allocate an escape virtual channel for that packet. Except for the two cases above, we just increase the counter by 1 per step. When the counter gets to the specified deadlock upper bound, a filtering stage will be trigger.

2. Filtering:

In this phase, the network needs to figure out whether the recovery requests are real deadlock or just false positive. The way we do it is to check the virtual channel's state. As we know there are four states for the virtual channel: idle, routing, virtual channel allocation (vc_alloc) and active. If there is any virtual channel in idle state or there is a packet just ready for ejection, we think the deadlock is not true (false positive), otherwise, we will allocate escape virtual channel for those virtual channels in vc_alloc states (It means if all the virtual channels are in their active states, we will not allocate any escape virtual channel for this input either).

3. Recovery:

In this phase, we have selected those input virtual channels whose inner packets (head flits) have been waiting for an available virtual channel for a long time (>deadlock timeout). We apply a priority selector here to help us determine which

virtual channel should be the first to obtain the escape virtual channel. After allocating the escape virtual channel, we will clear the timeout counter on that input port. Using FSM to describe the process in **Figure 3**.

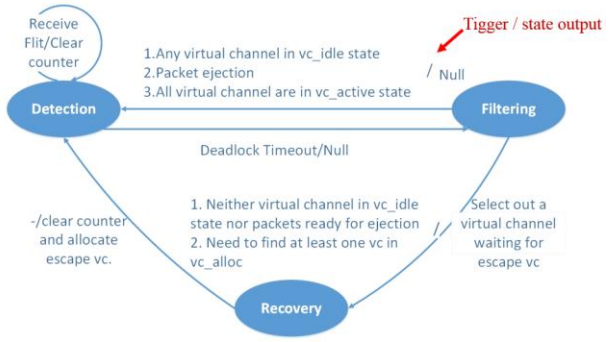


Figure 3 Deadlock recovery mechanism: Mainly has three phases: Detection (D), Filtering (F) and Recovery (R).

B. Up/down deadlock free routing algorithm:

We choose Up/down routing algorithm as our deadlock free algorithm applying on escape virtual channel. Since our 8x8 mesh network has several permanent faults on it, we cannot use some simple deadlock free algorithms like x-y dimension order algorithm for escape path. To take fully advantage of the DAR table generated for GCA algorithm, we finally choose the up/down algorithm.

The paper [2] introduces the up/down routing, a deadlock-free algorithm that can operate on any irregular topology. Up/down requires each link to be assigned a direction: up or down. It then disallows those paths that include traversing a down link followed by an up link. In this way, all cyclic dependencies are broken. In this paper, we take fully advantage of our GCA algorithm to generate a pseudo-up/down algorithm which can work correctly but may lose a little performance.

Instead of choosing the root node when coming across a fault, we simply fix our root at a certain node at the very beginning. And then based on this root node we can use the GCA algorithm directly to realize the deadlock-free algorithm. Now, let's see how this pseudo-up/down routing algorithm works.

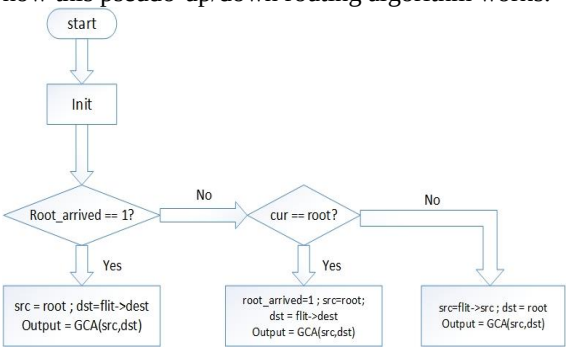


Figure 4 Up/down routing algorithm based on GCA: 'Cur' represents the ID of the current router. After the packet has passed through the root node (root arrived bit has been set 1), we will just use GCA table to find the head flit's next direction to its destination. If it is on the root node, we will set the root arrived bit to be 1 and use GCA table to find an output port from root to its destination. If it hasn't gone through the root node (root arrived bit is 0), we will set this packet's destination to be the root node and use GCA table to find the next output port

In order to implement this algorithm, we need firstly add a bit (named *root_arrived*) in flit which indicates whether the flit has passed through the root node.

The reason why this algorithm is deadlock free is that we are based on GCA table which will always give us a closer-to-dest direction even when there are permanent faults in NoC. So when we use GCA to send flit from source to root and then from root to destination, we actually disallow those paths that include traversing a down link followed by an up link. In this way, the algorithm implemented is deadlock-free.

IV. HARDWARE IMPLEMENTATION

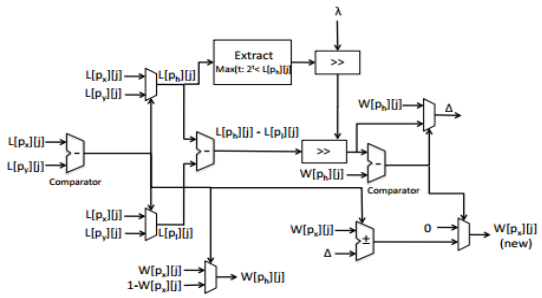


Figure 5 Delay Measurement and Propagation Logic

We implemented logics needed by the router in Verilog HDL to measure the storage overhead of our routing algorithm. In the DAR [1], they achieve 4.5% storage overhead over baseline router. In our design, we prove that the fault-tolerant feature cost is also reasonable, which leads to 6.1% overall storage overhead compared to baseline router. Here is some major logics we added to the router.

A. Port Pre-Selection

Because our router is designed for 2D mesh network, minimal adaptive routing is used to pre-select outputs ports. A packet arriving at an input port can have a choice of at most two output ports which maps to one of the four quadrants. As one hot port representation used, the Port Pre-Selection part introduced 10 bits storage for each destination including the current node itself.

B. Delay Measurement and Propagation Logic

Seen from **Figure 5**, delay measurement contains two parts: local queuing delay count and average delay calculation. Since we are mainly interested in the relative delays to destination node through the candidate output ports, the local queuing delay for output port p is approximated by the number of flits in the input buffers that have already acquired a VC at the next-hop router connected to port p.

Since port pre-selection logic has selected at most two output ports for each destination node, Average delay from corresponding downstream node will be used to compute delay to destination node through pre-selected ports. Then traffic split ratio will be used to compute weighted average delay from current node to destination node. Then computed average delay can be propagated to upstream node. Local queuing delay and average delay both have 6 bits, and every router have to store 2 local delay and one average delay for each destination node.

In order to reduce storage overhead, we only store one 5-bit traffic split ratio for each destination node since a packet at an

input buffer can have a choice of at most two output ports which maps to one of the four quadrants, and split ratios are normalized such that they always add up to one.

C. Adapt Split Ratio

The computations involved with adaptation of split ratios are given as follows:

$$\Delta = min(\lambda \left(\frac{L[p_h][j] - L[p_l][j])}{L[p_h][j]} \right), W[p_h][j])$$

$$W[p_x][j] = W[p_x][j] \pm \Delta \quad ()$$

To simplify the implementation of these computations in hardware we always assume λ = 0.25 which reduces the multiplication to a shift operation. The division is also avoided by extracting only the most significant bit of L[ph][j] that is set and ignoring the remaining less significant bits. This reduces division to a shift operation.

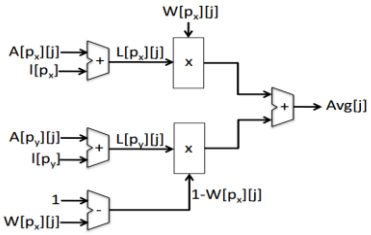


Figure 6 Logic for Adaption of Weights

D. Reconfiguration flag forwarding unit

For proposed fault tolerant algorithm, additional hardware unit is needed to receive and forward the flag signal where the calculation and the updates of traffic split ratio could be done using the hardware sources introduced in IV (B) and (C).

The reconfiguration flag forwarding unit is consisted of two major parts: an arbiter combination logic and a N^2 size buffer for an N x N mesh network. Thus the area overhead for this unit is quite small compared to the modern router architecture.

The arbiter identifies the id (indicate the destination router) of routers which have link errors and send signals to trigger the split ratios updates of the corresponding routing table. Besides, the arbiter selects the ports to forward the reconfiguration flag and send to the output buffer.

The buffer is used to indicate whether the reconfiguration has been done or not for a specific root node. Upon the router receive a reconfiguration flag of a specific root node at the first time, the corresponding buffer set high and the reconfiguration will not be trigger again if the router receives that flag signal again in the future. This mechanism avoids redundant reconfigurations as well as the potential livelock to some degree.

V. Evaluation

We evaluated our GCA algorithm with a cycle-accurate NoC simulator, BookSim. An 8x8 mesh network is utilized for evaluation with several different traffic patterns considered. And for the evaluation of faults within network, a random fault generator is added to the original BookSim for generating random faulty network without isolating any node by specifying the number of fault. For this evaluation part, an evaluation on performance of the GCA algorithm in non-faulty network is conducted in comparison with some extant routing algorithms in BookSim, as well as a comparison in saturation throughput. For evaluation of performance on faulty network, an increasing number of fault is inserted into network with the random fault generator at a fixed injection rate, thus fault tolerance of the proposed routing algorithm is tested.

A. Evaluation of GCA algorithm in non-faulty network

Dimension-order, *min-adaptive* and *xy_yx-adaptive* are used for a comparison with the proposed routing algorithm in non-faulty network, as they are the typical deterministic/adaptive routing algorithms on mesh network. For four different traffic patterns – uniform, shuffle, bitrev and transpose, average packet latency is measured for the three extant algorithms as well as the proposed GCA routing algorithm. The result is shown in ***Figure 7***.

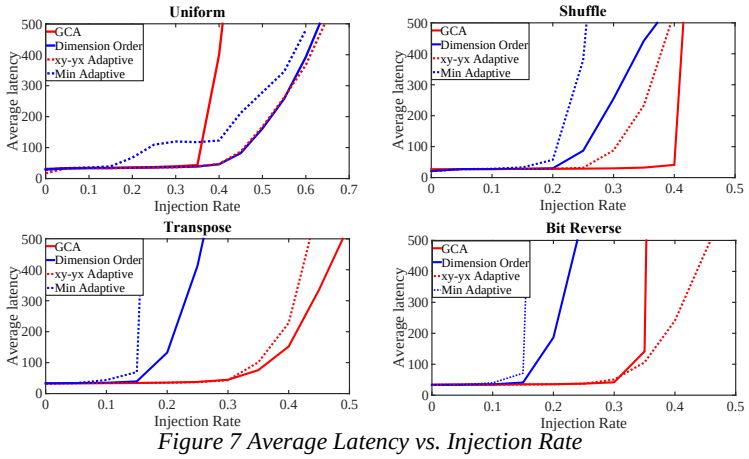


Figure 7 Average Latency vs. Injection Rate

Seen from **Figure 7**, average latency will increase dramatically at a certain point for each routing algorithm and each traffic pattern. Such certain point on injection rate is named as saturation throughput. GCA algorithm performs best in shuffle and transpose traffic pattern but worst in uniform, the reason is that GCA is aimed at keeping the traffic balanced in mesh network, for shuffle and transpose traffic, GCA algorithm is always the most efficient one among these algorithms , but for uniform, GCA loses some performance as a trade-off.

Saturation throughput can be estimated from **Figure 7** by measuring the inject rate at which average latency is triple of the zero-injection latency. And a comparison in saturation throughput is illustrated in **Figure 8**.

Seen from **Figure 8**, the saturation throughput for GCA is preferable among all routing algorithms except the uniform traffic pattern, but in reality, the uniform scenario is rare, in all for network without fault, GCA is a proper choice for routing algorithm.

B. Simulation of GCA algorithm in faulty network

Actually we have several choices on the fault-tolerance solution, the simplest choice is random-walk. After implementing and evaluation of random-walk routing algorithm on BookSim, the poor efficiency and deadlock problem prevent us from research deeper on such topic.

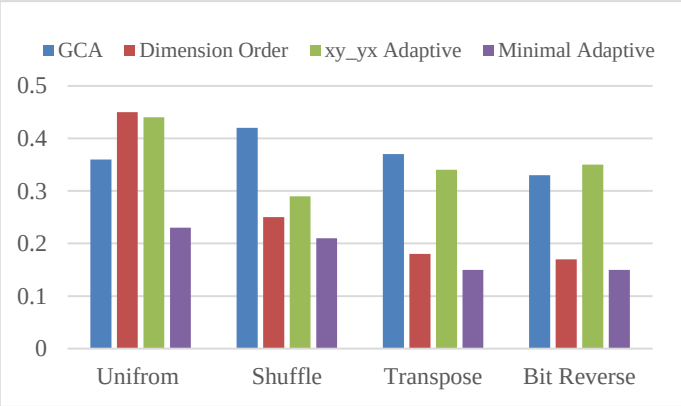


Figure 8 Comparison of saturation throughput

For the Up/Down routing algorithm we have discussed in III. B, deadlock-free as it is, the comparatively long latency also prevents us from taking it as a main routing algorithm. Alternatively, due to its metric in deadlock-free, it can be used complementarily as the routing algorithm for escape virtual channel as deadlock situation is rare but indeed exists in network.

For faulty network, random error generator is utilized for simulation. Injection rate for this part is fixed to be 0.2. As the number of fault within network increases from 1 to 10 in 8x8 mesh network, by measuring the average latency for specific number of fault 10 times, average latency for each scenario can be obtained as below. Note that four different traffic patterns are also considered for this part.

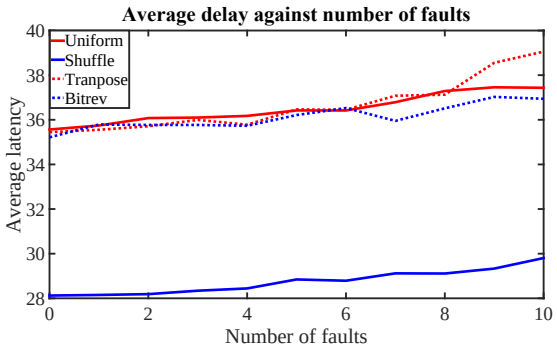


Figure 9 Average delay vs. number of fault

Seen from **Figure 9**, due to the effective reconfiguration stage in dealing with faults, the average latency increases slowly as the number of fault increases.

C. Comparison between proposed work and some published routing algorithms

At last, a table is presented for a comparison between the proposed GCA routing algorithm and some published routing algorithms.

As we can see from **Table 2**, the proposed routing algorithm works well even in comparison with some published work.

Table 2 Comparison between proposed work and published work

	This work	[3]	[4]	[5]	[7]
Algorithm	Adaptive	Adaptive	Deterministic	Adaptive	Adaptive
Fault tolerant?	Yes	No	No	No	Yes
Saturation Throughput for different traffic pattern					
Uniform	0.36	0.35	0.36	0.32	0.34
Shuffle	0.42	-	-	-	-
Transpose	0.37	0.33	0.21	0.27	-
Bit-comp	0.22	0.21	0.22	0.16	-

VI. CONCLUSION

In this paper, the proposed routing algorithm – GCA (Global-Congestion Adaptive) – is designed based on Destination-based Adaptive Routing (DAR) with relatively hardware overhead of 6.1%. Besides, a deadlock recovery mechanism using escape virtual channel which is equipped with a deadlock free up/down algorithm. Comparing our results with some other algorithms like improved random walk and original up/down algorithm, our algorithm has a better for different traffic patterns. In the future, some research can be taken into explore the possibility in improving the fault-tolerance performance by bringing in software-based off-line reconfiguration mechanism.

References

[1] R. Ramanujam and B. Lin, "Destination-based congestion awareness for adaptive routing in 2D mesh networks", ACM Transactions on Design Automation of Electronic Systems, vol. 18, no. 4, pp. 1-27, 2013.

[2] K. Aisopos, A. DeOrio, L. Peh, and V. Bertacco, "ARIADNE: Agnostic Reconfiguration In A Disconnected Network Environment", International Conference on Parallel Architectures and Compilation Techniques (PACT), Galveston Island, TX, October 2011.

[3] P. Gratz, B. Grot and S. Keckler, "Regional Congestion Awareness for Load Balance in Networks-on-Chip", HPCA, 2008.

[4] D. Seo, A. Ali, W. Lim, N. Rafique and M. Thottethodi, "Near-Optimal Worst-Case Throughput Routing for Two-Dimensional Mesh Networks", ACM SIGARCH Computer Architecture News, vol. 33, no. 2, pp. 432-443, 2005.

[5] A. Singh, W. Dally, B. Towles and A. Gupta, "Globally Adaptive Load-Balanced Routing on Tori",IEEE Comput. Arch. Lett., vol. 3, no. 1, pp. 2-2, 2004.

[6] S. Jovanovic, C. Tanougast, S. Weber, and C. Bobda, "A new deadlock-free fault-tolerant routing algorithm for NoC interconnections", in Proc. Int. Conf. Field Program. Logic Appl., Aug.–Sep. 2009, pp. 326–331.

[7] R. Parikh, V. Bertacco, "ForEVeR: A complementary formal and runtime verification approach to correct NoC functionality". ACM Trans. Embedded Comput. Syst.13(3s): 104:1-104:30 (2014)