A Fault-Tolerant Routing Algorithm for a Network-on-Chip Using a Link Fault Model

Jian Wu, Zhen Zhang, and Chris Myers
Electrical and Computer Engineering Department
University of Utah
Salt Lake City, Utah, USA
Emails: jian.wu@utah.edu, zhen.zhang@utah.edu, myers@ece.utah.edu

***Abstract*—Adaptive routing is a sensible approach to enhance fault-tolerance in Network-on-Chip (NoC) architectures, but can cause deadlock if implemented improperly. Glass and Ni proposed an adaptive routing algorithm based on a turn model which is proven to tolerate at least one fault in each routing process and is free of deadlock. However, when faults happen on links of a network instead of on nodes, these two claims are no longer true. This paper proposes an improved routing algorithm based on the Glass/Ni protocol which tolerates a single link fault while still avoiding deadlock in a mesh network. Simulation results indicate that this improved algorithm provides significant improvements in network reliability with minimal cost.**

I. Introduction

With the improvements in process technology, multi-core architecture has become the norm. A new communication paradigm, called *Network-on-Chip* (NoC), has been widely investigated for future *Systems-on-Chips* (SoCs) and *Chip Multiprocessors* (CMPs). Fault-tolerance is an important feature of NoC since it usually has inherent redundancy in communication which can be used for hardware reconfiguration. For example, an alternative link can be chosen to route around a dead node to maintain the normal operation of the system. This is accomplished by dynamically redirecting packets to avoid failures in the network, so called *adaptive routing*.

A challenge of implementing adaptive routing is to avoid deadlock. Deadlock can potentially happen when a cycle forms among several nodes in a network. Adding virtual channels to routers is a method to avoid deadlock. However, virtual channels are not free. The use of virtual channels introduces extra area and energy cost. Moreover, virtual channels do not eliminate the possibility of deadlock, they only reduce their probability [1].

Glass and Ni proposed another way of realizing adaptive routing that guarantees to tolerant at least a single faulty router in a two dimensional mesh network without causing deadlock [2]. This algorithm always attempts to choose a route that allows a packet to have multiple choices in selecting its next routing node along the path to its destination. If the first node cannot be reached due to a failure, the packet may use the second choice as the alternative path. Therefore, a single failed node can always be tolerated. To avoid deadlock, the algorithm eliminates the formation of cycles by preventing particular "turns" in the network. Therefore, it is also called the *turn model*.

In the context of NoCs, a communication failure can happen due to a fault on either a router or a link between two routers [5]. The Glass/Ni algorithm only guarantees deadlock-free for the router failure, but it fails to work for all link failures, as it treats a link failure scenario as a router failure in its assumptions [2]. In the link fault model, the algorithm may not be successful to route around a single link failure, or even worse, in some cases, it can cause deadlock. Yoneda and Imai modified the Glass/Ni algorithm to achieve one-link-failure tolerance using a fault-forwarding mechanism with some additional hardware cost [6]. This method, however, cannot allow the link failure model to be applied to the west and south edges of the mesh as deadlock can occur. There have also been several related works based on the turn model which handle more than one fault by grouping multiple faults into regions. Wu uses the odd-even turn model to address convex, disjoint fault regions that do not lie on the mesh boundary [3]. In his algorithm, the fault requirements are strict. Zhang et al. proposed a routing algorithm that can handle both one-faulty-router and one-faulty-region [4]. But this algorithm does not apply to the link fault model. Fick et al. proposed another routing algorithm based on the turn model, which is not limited to any particular number of failures [7]. Rather than using adaptive routing, their approach relies on reconfigurations of the network routing tables to achieve fault tolerance and deadlock-freedom. However, this approach only handles permanent faults.

This paper proposes an improved routing algorithm based on the Glass/Ni algorithm that guarantees deadlock-freedom as well as tolerance for a single link fault at any location in the network. This method does not require any additional fault forwarding hardware. Instead, our algorithm achieves adaptive routing by allowing the turns that the Glass/Ni approach disallows, but the routers do not block on them. Namely, if it appears that deadlock may occur, it drops the packet to break the potential cycle. The proposed algorithm improves fault resilience, measured by packet loss rate, under both transient and permanent failures.

The remainder of this paper is organized as follows. Section II introduces the network topology which our routing

algorithm is based on, and the modeling of link faults using an asynchronous handshake protocol. Section III presents the problem with the Glass/Ni algorithm when it is used in a network under the link fault model. Our proposed routing algorithm is described in section IV, followed with simulation results. Finally, conclusions and some future works are given in Section V.

II. Network Architecture and Link Faults Modeling

In this paper, a 2-dimensional (2D) mesh topology is employed as the on-chip network architecture. The 2D mesh topology has been widely used in previous research due to reasons such as simplicity in physical layout, redundancy in routing, and regularity [8]. Fig. 1 (a) shows a four-by-four 2D mesh network with 16 nodes. The nodes are labeled by their positions (x,y), and connected by pairs of input/output links. In a NoC, a node usually consists of a router and a *intellectual property* (IP) core as shown in Fig. 1 (b). The router is responsible for sending and receiving data packets among the nodes. The IP core is responsible for packet generation and processing. Besides routing packets to/from its adjacent IP core, a router can directly route packets between its neighbors through ports North (N), South (S), West (W), and East (E), respectively.

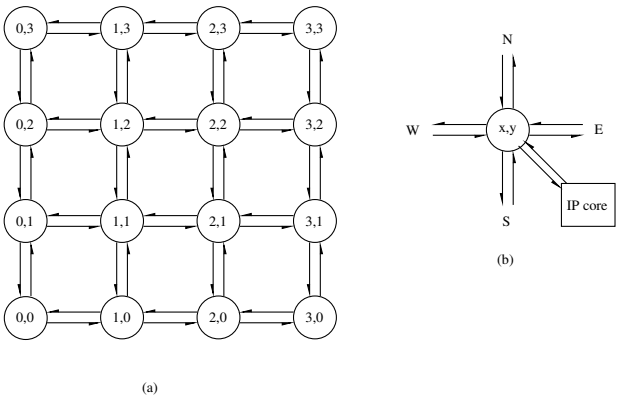

Fig. 1. (a) A four-by-four 2D mesh network with 16 nodes. (b) The connections to an individual router node.

To model the links between routers, our method employs asynchronous channels implemented in a VHDL package [9]. In addition to defining a channel data structure, this package provides the functions *send*, *receive*, and *probe* to enable communication of data over a channel. A VHDL process executes a *send* or *receive* function when it wishes to request communication over the channel. A communication occurs when one process connected on one *port* of a channel requests to send while another process connected to the other port of the channel request to receive. The data is then transmitted over the channel using a handshaking protocol. The *probe* function can be utilized to determine if the process connected on the other port of a channel has a pending request on the channel.

Each link shown in Fig. 1(a) includes a buffer which is shown in more detail in Fig. 2. In order to model link failures, this buffer has a tunable probability of failure. In particular, this buffer actually includes two channels, a *Good* channel and a *Bad* channel. The *Channel select* signal is used to randomly choose for each packet transaction whether to use the *Good* or *Bad* channel. If the *Bad* channel is selected, the packet cannot be transmitted to the next router (i.e., the link is faulty). Otherwise, the link is functional and data is communicated over the *Good* channel to the *Buffer Out* channel to the next router. The probability of selecting between the two channels can be set to different values so that different levels of link reliability can be simulated. In order to use this faulty buffer, before sending out a packet, a router first uses the *probe* function to check whether the buffer at its output port is ready and able to accept a new packet. If the buffer is full, then neither the *Good* nor the *Bad* channel is ready. Once the buffer becomes empty, a random selection is made and either the *Good* or *Bad* channel becomes ready. If the *Good* channel is ready, the router can transmit the data to the buffer. Otherwise, the router must either send the data in another direction or drop the packet if no viable alternative exists. In this case, the router also handshakes with the *Bad* channel which causes the buffer to make a new channel selection.

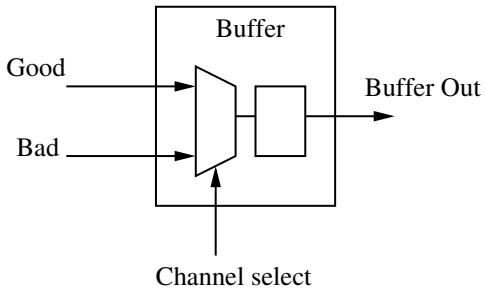

Fig. 2. Link buffer with tunable probability of failure.

Note that the probability of choosing a *Bad* channel using the random selection function is assumed to be p_f which is the probability of the link being faulty. By setting p_f to be the same for every buffer, faults are distributed randomly on the network. These faults are transient faults, since the fault state is determined for each individual transaction. For example, for every 1 million packets with an average of 5 hops, about 50 thousands faults are injected when p_f is 1%.

III. Problem Description

Glass and Ni use a systematic method called the *turn model* to prevent deadlock [2]. In the turn model, the west and south boundaries of the network are defined as negative edges, and the north and east boundaries are defined as positive edges. A packet that is routed toward these two kinds of edges are called the negative and positive phases of routing, respectively. In a 2D mesh topology network, there are four corners in each grid which form a cycle potentially causing deadlock. To avoid this, the Glass/Ni algorithm requires a packet first to proceed along the negative phase, if it is available, until it reaches farther west and south than the destination. Then the packet turns to the

positive phase. Once a packet is in its positive phase, it cannot switch back to the negative phase. This restriction allows the packet be fully adaptive along any of its paths, and never needs to make turns at northeastern corners of each grid, which are referred to as "illegal turns".

The Glass/Ni routing algorithm can tolerate one node fault in a 2D mesh network. This robustness is achieved by ensuring that a packet always has at least two choices of its next move at each node along its path. When on its negative phase, a packet can either stay in this phase or switch to the positive phase. If a packet is already on its positive phase, it still has two choices: go north or go east. There is though one special case that needs to be considered when a packet traverses a negative edge (i.e., the south edge or the west edge) of the network and encounters a faulty node. In this case, a packet only has one choice to route. For example, Fig. 3 shows a portion of a mesh network where routers (0,0), (1,0), and (2,0) are on the south edge of the network. Let us denote the forward channel from router (x_1, y_1) to router (x_2, y_2) as $(x_1, y_1) \rightarrow (x_2, y_2)$. Suppose that a packet located at router (0,0) is destined for router (2,0). If no fault is encountered, the packet would follow the route $(0,0) \rightarrow (1,0) \rightarrow (2,0)$. If router (1,0) has a fault, the packet instead must route around the faulty router using the route $(0,0) \rightarrow (0,1) \rightarrow (1,1) \rightarrow (2,1) \rightarrow (2,0)$. This route though includes the illegal turn from (1,1) to (2,0). For this special case, this illegal turn is allowed. This does not have the potential to introduce deadlock because no cycle can be formed in this case as the faulty node, (1,0) in this example, breaks the potential cycle.

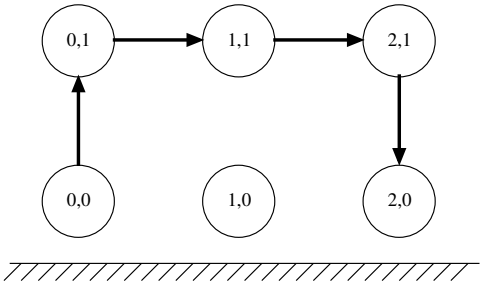

Fig. 3. A special case for negative edges in the Glass/Ni algorithm.

The special case for the Glass/Ni algorithm just described makes the assumption that all faults result in an entire node becoming permanently unusable. If faults are transitory or occur on a link, this algorithm cannot ensure robustness to a single link failure. Even worse, the algorithm can result in deadlock. For example, Fig. 4 illustrates a packet encountering a faulty link between nodes $(0,0)$ and $(1,0)$. As described earlier, the Glass/Ni algorithm detours packets from $(0,0)$ to $(2,0)$ using the route $(0,0) \rightarrow (0,1) \rightarrow (1,1) \rightarrow (2,1) \rightarrow (2,0)$. Since it is the link $(0,0) \rightarrow (1,0)$ that is considered as faulty rather than node (1,0), cycles, such as $(1,0) \rightarrow (1,1)$, $(1,1) \rightarrow (2,1)$, $(2,1) \rightarrow (2,0)$, $(2,0) \rightarrow (1,0)$, can still be formed. Suppose each of these four channels has a one place buffer, and link $(1,0) \rightarrow (1,1)$, $(1,1) \rightarrow (2,1)$, $(2,1) \rightarrow (2,0)$, and $(2,0) \rightarrow (1,0)$ are obtained by four packets, which are respectively going to router $(2,1)$, $(2,0)$, $(1,0)$, and $(1,1)$. None of the four packets can make any progress, since each packet is blocked by a packet in its outgoing channel. The result is a deadlock.

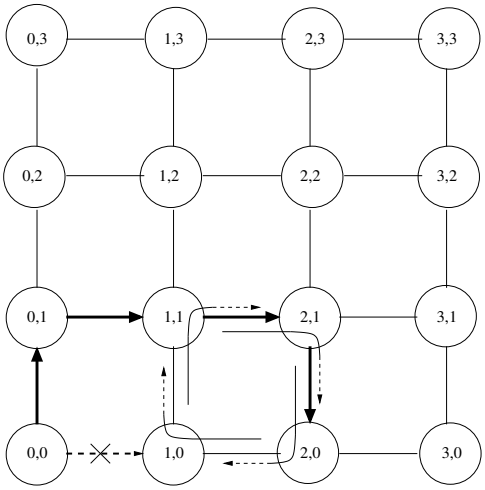

Fig. 4. Deadlock caused by a link fault.

There is another special situation that cannot be handled by the Glass/Ni routing algorithm when assuming link faults. When a packet encounters a fault that is one hop away from its destination node, the Glass/Ni algorithm must drop the packet since it assumes the fault happens on the destination node. However, in the link-fault model, there are still available routes to the destination node. In [6], Yoneda and Imai address this problem using a fault lookahead mechanism as shown in Fig. 5. In this example, a packet is at node $(1,1)$, and its final destination node is $(2,2)$. Before sending out the packet, node $(1,1)$ checks links $(2,1) \rightarrow (2,2)$ and $(1,2) \rightarrow (2,2)$. If there is a fault on one of the links, node $(1,1)$ sends the packet towards the other link to avoid this link fault. While this method can handle these one-away link faults, it introduces additional hardware cost. Moreover, it must assume that only routers can fail on negative edges, since it cannot handle link faults in these situations.

IV. Proposed fault-tolerant routing

Our proposed routing algorithm achieves tolerance to any single link fault while avoiding deadlock. To realize one fault tolerance, all nodes must be able to communicate on over one alternative channel. In our proposed algorithm, non-blocking illegal turns are allowed to handle the two cases mentioned above. For example in the case showed in Fig. 3, the packet is allowed to make the turn from east to south at node $(2,1)$. A deadlock may occur only when all the four links have been obtained by packets in a cycle as illustrated in Fig. 4. Our method though does not block the packet moving from $(1,1)$ to $(2,0)$. Instead, if the buffer on the channel between $(2,1)$ and $(2,0)$ does not clear in a sufficient time, the packet from $(1,1)$ is simply accepted and dropped effectively breaking the cycle, as illustrated in Fig. 6.

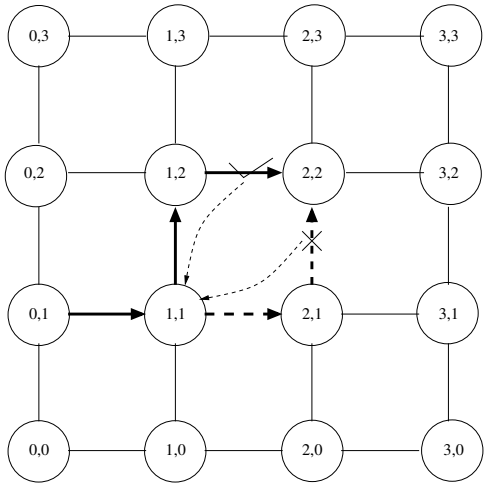

Fig. 5. A fault lookahead mechanism.

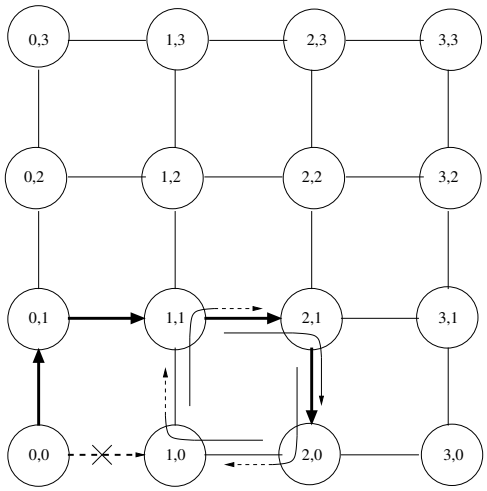

Fig. 6. Packet is dropped to break deadlock.

To handle the one-away faults, our method expands the non-blocking of illegal turns idea to every node in the network. For example, in Fig. 7, a packet is delivered from node $(0,1)$ to node $(2,2)$ and encounters a fault on link $(2,1) \rightarrow (2,2)$. In our algorithm, the packet is redirected to node (3,1), and it follows the route $(3,1) \rightarrow (3,2) \rightarrow (2,2)$. This route requires an illegal turn on node $(3,2)$. However, if the buffer between nodes $(3,2)$ and $(2,2)$ does not become free in a sufficient amount of time, the packet is dropped at node $(3,2)$ avoiding the potential deadlock.

To examine the importance of tolerance to a single link fault, let us consider the 2-by-2 mesh network shown in Fig. 8. Suppose that the probability of a link being faulty is p_f. The network reliability is the probability of a packet being dropped due to link faults, and it is denoted by P_{fault}. Consider a packet that is injected at node $(0,0)$ and destined for node $(1,0)$. In a non-adaptive algorithm, only one choice of route is possible (i.e., $(0,0) \rightarrow (1,0)$), so $P_{fault} = p_f$. If two choices can be made at node $(0,0)$ (i.e., $(0,0) \rightarrow (1,0)$ or $(0,0) \rightarrow (0,1) \rightarrow (1,1) \rightarrow (1,0)$), the probability is calculated to be:

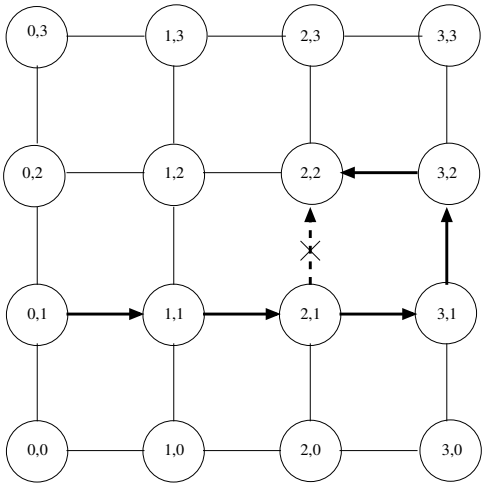

Fig. 7. One-away fault handling example.

$$\begin{aligned} P_{fault} &= p_f^2 + p_f(1-p_f)[p_f + (1-p_f)p_f] \\ &= 3p_f^2 - 3p_f^3 + p_f^4 \\ &\approx 3p_f^2 \end{aligned}$$

Note that since $p_f << 1$, $3p_f^3$ and p_f^4 can be ignored. For node $(0,0)$'s two other possible destinations, node $(0,1)$ and node $(1,1)$, P'_{fault} is still about $3p_f^2$. Although the Glass/Ni algorithm can tolerate single faults in most cases, this is not true in some special cases in the link fault model as mentioned above. Therefore, the packet loss rate in the Glass/Ni algorithm is dominated by p_f, i.e. the probability of a packet being dropped is increased significantly even if there is only one case that a single fault cannot be tolerated. If transient faults are randomly distributed in the network, these unsupported situations can easily be encountered and determine the overall packet loss rate. Our experimental results demonstrate that the packet loss rate is significantly higher if one-link-fault tolerance cannot be maintained.

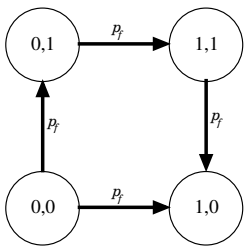

Fig. 8. A 2-by-2 mesh network with fault probability of p_f on each link.

Our algorithm, however, may introduce additional packet loss to avoid deadlock when attempting to make illegal turns. Therefore, the overall packet loss rate, $P_{overall}$, is given as follows:

$$P_{overall} = P_{fault} + P_{deadlock},$$

where $P_{deadlock}$ is the additional packet loss due to deadlock avoidance. It is important to note that while P_{fault} is independent of the packet injection rate in the network, $P_{deadlock}$ is clearly a function of this rate. Therefore, $P_{deadlock}$ can be minimized by limiting the network traffic load, adding additional buffers, or resending these dropped packets.

Since $P_{deadlock}$ is a function of packet injection rate, it is important to determine the injection rates of interest. Fig. 9 shows our network's throughput under different packet injection rates. The packet injection rate is measured by the number of packets injected into the network per router delay. Note that all our simulations are run for about 1000 ms with an average of 80 million packets injected following a uniform traffic pattern. The plot shows that the network's throughput reaches its maximum at an injection rate of about 1.8 packets/router delay. After that, the throughput drops due to network saturation. Since the network only works efficiently when it is not saturated, our evaluations of fault-tolerance performance are focused on injection rates below about 2.0 packets/router delay.

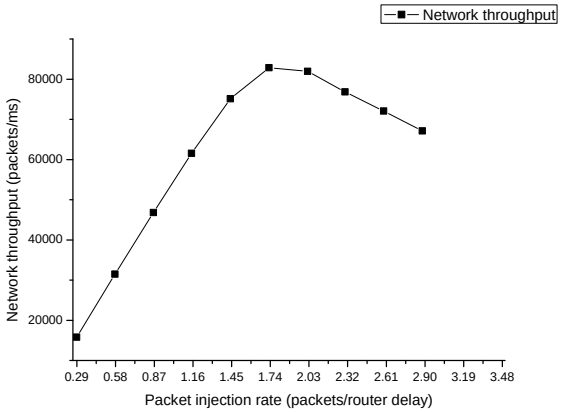

Fig. 9. Network throughput vs. packet injection rate.

Fig. 10 presents a comparison of packet loss rates for our algorithm with a non-adaptive routing algorithm and a variation on the Glass/Ni algorithm that avoids the deadlock problems on the negative edges. This variation does not handle one away link faults. The packet loss rate is measured as the percentage of dropped packets out of the total number of injected packets. These results demonstrate that the packet loss rate of the proposed routing algorithm is significantly better. Even though one away faults are only part of all single fault cases, failure to handle these types of faults increases the packet loss rate by 20 times compared to our proposed algorithm. This justifies our previous analysis that the existence of any single link fault problems ends up dominating the overall packet loss rate. Since our algorithm gives priority to deadlock avoidance, the trade-off is that packets get dropped when there is a potential for deadlock. To see this tradeoff, Fig. 10 includes both the overall packet loss rate as well as the loss rate due to link faults only. While for low packet injection rates most packet loss is due to link faults, as the injection rate increases, the packet loss due to deadlock avoidance begins to dominate.

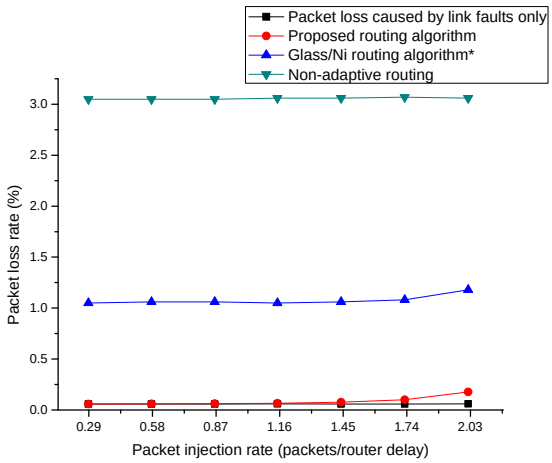

Fig. 10. Packet loss rates for different routing algorithms.

In order to obtain simulation results in a reasonable time, it is necessary to use an artificially high link fault probability. In reality, the probability of a link fault is extremely low. In order to evaluate our algorithm using a realistic link fault probability would require a prohibitively long simulation time. Using a technique known as *importance sampling*, reliability can be estimated for a low link fault probability using simulation results found at a relative high link fault probability [10]. In particular, importance sampling can be used to estimate properties of a particular distribution using samples generated from a different distribution. The basic idea is to change the probability of P_{fault} to make the estimation of $\mathbf{E}[X; P_{fault}]$ easier. To achieve this, a random variable L is chosen to define another probability $P_{fault}^{(L)} = LP_{fault}$ such that:

$$\mathbf{E}[X; P_{fault}] = \mathbf{E}[\frac{X}{L}; P_{fault}^{(L)}]$$

For simplicity, let us again consider the 2-by-2 mesh as an example. As mentioned above, the formula of estimating the packet loss rate of the 2-by-2 mesh is:

$$P_{fault} \approx 3p_f^2$$

To calculate P_{fault} under a low link fault probability using importance sampling, p_f can be weighted by a ratio w. Then the new packet loss rate, denoted as P'_{fault}, becomes:

$$\begin{aligned} P'_{fault} &\approx 3(w * p_f)^2 \\ &= w^2 P_{fault} \end{aligned}$$

Since $P_{deadlock}$ is caused by single link fault, its value is determined by p_f. Therefore, the overall failure rate is:

$$P_{overall} = w^2 P_{fault} + wP_{deadlock}$$

Using this equation, an estimate of the packet loss rate in a 4-by-4 mesh network under 0.1 percent link fault probability can be found using a 1 percent link fault probability and $w = \frac{1}{10}$.

Fig. 11 shows a comparison of the estimated results with actual simulation results demonstrating that this approach provides a reasonable estimate.

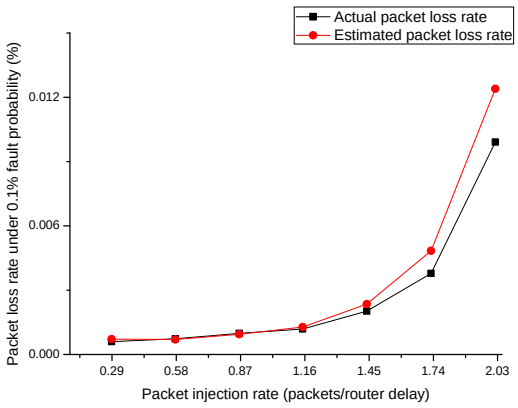

Fig. 11. Packet loss rate estimation based on the importance sampling technique.

Fig. 11 shows that under a lower probability of link fault the impact of deadlock avoidance becomes more significant. This is because $P_{deadlock}$ is determined by p_f (i.e., the probability of failure of a single link). Therefore, the impact of network congestion on packet loss rate increases as the link fault probability decreases. The effect of network congestion can be reduced by adding larger buffers to the links. The results shown in Fig. 12 indicate that by using two buffers instead of one in each link of the network, the packet loss rate can be largely eliminated.

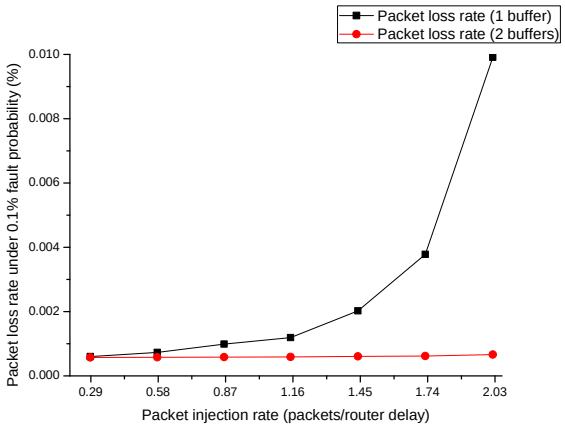

Fig. 12. Packet loss rate in network with 2 buffer size.

V. Conclusions

This paper proposes an improved version of the Glass/Ni routing algorithm which significantly enhances the network reliability in the presence of transient link faults. Meanwhile, it successfully solves the deadlock problem which the traditional turn model encounters in networks with link faults. Our algorithm guarantees deadlock-freedom by trading-off additional packet loss. However, this can be tolerated as long as the network is not saturated. Simulation results show that the packet loss rate has been largely decreased by using our proposed algorithm compared to a modified Glass/Ni algorithm. It is also shown that the packet loss rate does not increase significantly due to deadlock, indicating the trade-off is reasonable, especially when buffer size is increased. This paper also considers the use of importance sampling to estimate a network's actual reliability performance using a reasonable simulation effort.

The 4-by-4 mesh network has been implemented and simulated under 1 percent and 0.1 percent link fault probabilities. Future work includes deriving analytical packet loss rates for the 4-by-4 mesh and providing more substantial support for our experimental results. We may further investigate the technique to reveal other factors that have impact on the estimated results. Other fault models, such as permanent faults may be applied to measure the network's fault resilience ability. We also plan to verify the network using stochastic model checking.

Acknowledgment

We would like to thank Professor Hao Zheng from the University of South Florida and Professor Tomohiro Yoneda of the National Institute of Informatics, Tokyo, Japan. This work is supported by the National Science Foundation under Grant CNS-0930225.

References

[1] A. Agarwal, C. Iskander, H. Multisystems, and R. Shankar, Survey of network on chip (noc) architectures & contributions, 2009, Journal of Engineering, Computing and Architecture.
[2] Glass, C.J. and Ni, L.M., Fault-tolerant wormhole routing in meshes, 1993, In Proceedings of the 23rd International Symposium on Fault-Tolerant Computing (FTCS-23).
[3] Wu, J., A fault-tolerant and deadlock-free routing protocol in 2D meshes based on odd-even turn model, 2003, IEEE Trans. on Computers, 52(9).
[4] Zhang, Z.; Greiner, A.; Taktak, S., A reconfigurable routing algorithm for a fault-tolerant 2D-Mesh Network-on-Chip, 2008, 45th ACM/IEEE Design Automation Conference (DAC).
[5] Tudor D., Sam K., Radu M., Towards On-Chip Fault-Tolerant Communication, 2003, In Proceedings of the 2003 Asia and South Pacific Design Automation Conference (ASP-DAC'03).
[6] Imai, M.; Yoneda, T., Improving Dependability and Performance of Fully Asynchronous On-chip Networks, 2011, In Proceedings of the 17th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC).
[7] Fick, D.; DeOrio, A.; Chen, G.; Bertacco, V.; Sylvester, D.; Blaauw, D., A highly resilient routing algorithm for fault-tolerant NoCs, 2009, In Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE'09).
[8] W.J. Dally and B. Towles, Principles and practices of interconnection networks, 2004, Morgan Kaufmann.
[9] Myers, C., Asynchronous Circuit Design, 2001, John Wiley & Sons, Inc.
[10] Srinivasan, R., Importance Sampling: Applications in Communications and Detection. 2002, Springer.