Rescuing Healthy Cores Against Disabled Routers

Masoumeh Ebrahimi[1,2], Junshi Wang[3], Letian Huang[3], Masoud Daneshtalab[1,2], Axel Jantsch[1]
[1]*KTH Royal Institute of Technology, Sweden,* [2]*University of Turku, Finland,* [3]*University of Electronic Science and Technology of China, China*
masebr@utu.fi; wangjsh@std.uestc.edu.cn; huanglt@uestc.edu.cn; masdan@utu.fi; axel@kth.se

Abstract— A router may be temporarily or permanently disabled in NoCs for several reasons such as saving power, occurring faults or testing. Disabling a router, however, may have a severe impact on the performance or functionality of the entire system if it results in disconnecting the core from the network. In this paper, we propose a deadlock-free routing algorithm which allows the core to stay connected to the system and continue its normal operation when its connected router is disabled. Our analysis and experiments show that the proposed technique has 100%, 93.60%, and 87.19% network availability by 100% packet delivery when 1, 2 and 3 routers are defunct or intentionally disabled. The algorithm provides adaptivity and it is lightweight, requiring one and two virtual channels along the X and Y dimension, respectively.

Keywords: Networks-on-Chip; Router Test; Fault tolerance;

I. INTRODUCTION

One salient feature of Network-on-Chip (NoC) is its inherent path redundancy which provides a natural fault-tolerant infrastructure [1]. For instance, adaptive routing is considered to be an effective solution to tolerate disabled links and routers by allowing packets to select between multiple routes. A large number of proposals have exploited this aspect and proposed techniques to equip NoCs with fault tolerance [2]. Fault-tolerant routing algorithms including NoC-Link State (NoC-LS), Dynamic Routing Protocol for NoCs (DR-NoC) [3], Fault-on-Neighbor (FoN) [4], Q-Learning based routing [5], Simple Flooding Algorithm, Directed Flooding Algorithm, and Redundant Random Walk Algorithm [6] can be executed in NoCs in order to bypass the faulty areas. However, there has been less attention to the functioning cores either connected to the faulty routers or the routers under test. For example, when a router is faulty, the core is also disabled although it is still working correctly. Similarly, when a router is under test, the connected core should be disabled, which may result in stopping the entire system.

Our proposed algorithm, called *CoreRescuer*, satisfies three different properties: (1) it maintains the connectivity of the cores to the network when their belonging routers are disabled; (2) The routing algorithm is reconfigured to support this irregular connectivity; (3) The presented algorithm maintains the performance of NoCs through employing adaptive routing.

The reminder of this paper is organized as follows. In Section II, related work is given. In Section III, the router architecture of the proposed method is introduced. Section IV describes the CoreRescuer algorithm by giving several examples. The analytical and experimental results are reported in Section V while the summary and conclusion are given in the last section.

II. RELATED WORK

There are many fault-tolerant approaches presented for on-chip networks. These methods are able to tolerate faults in links [7][8][9][10], routers [11][12][13], or both [4][5][15].

In [7], the authors take advantage of a routing table at each router and an offline process to fill out the tables. A recursive method is used to fill out the tables. This method is based on a deterministic routing algorithm, degrading the performance of NoCs. However, it is suitable for tolerating a large number of faulty links in the network. An adaptive method is presented in [8], supporting fully adaptive routing in the absence of faults and providing some degree of adaptiveness in the presence of faults. This method is mainly based on the selection of an intermediate node for each source and destination pair. Packets are adaptively routed to an intermediate router and then they are forwarded to their destinations. In Ariadne [9], by any new fault, the network is reconfigured to replace the routing algorithm with a new one upon each failure, by discovering the underlying topology and establishing deadlock-free routes. The DBP approach [11] uses a default back-up path at each router to connect the upstream to the downstream router in the case of fault. Thereby, besides the underlying interconnection infrastructure, these backup paths connect all routers together in a form of a unidirectional cycle such as a ring. This algorithm is based on taking non-minimal routes. HiPFaR [12] targets tolerating faulty routers by avoiding non-minimal paths as along as possible.

Besides the methods that focus on tolerating either faulty routers or links, there are some methods that take both types of faults into account. TFLR [14] specifically pays attention to both types of faults by the means of routing algorithms.

A fault-tolerant approach based on reinforcement learning is presented in [5], using a routing table at each router. The routing tables are updated based on the statuses of all routers and links in the network and the local routing decision is made based on the gathered information. The presented algorithm in [4] is able to tolerate both faulty links and routers but at the cost of complexity and considerable hardware overhead.

Vicis [15] suggests a framework with the overhead of 40% targeting permanent faults and capable of detecting and diagnosing errors, system reconfiguration and system recovery. In this framework, after detecting any faults, the build-in-self-test mechanism is used to locate faults, and finally the faulty components are disabled. Most of the proposed methods are based on a common assumption that the core is disabled if the local router is faulty. These algorithms try to keep the network working by disconnecting both the faulty router and the core from the network. However, this assumption is not realistic and may have a severe impact on the functionality of the entire

system or the performance (assuming that the tasks of the core are distributed among the other cores).

NoRD [16] takes this issue into account in the realm of power-efficient NoCs. In this method, a separate virtual channel is used to connect all cores to each other through a ring. Thereby, the cores connected to powered-off routers can communicate with the rest of the network by routing packets through this ring. All the other cores also should deliver packets using this ring in order to access the core connected to the powered-off routers. The main shortcoming of NoRD is that, in large networks, the ring will be very long, resulting in significant performance loss. NoRD requires two virtual channels along both X and Y dimensions.

DRAIN [17] uses a system-level recovery mechanism by transferring the architectural state and cached data from disconnected cores to nearby connect cashes.

In this paper, we introduce a routing algorithm which is able to survive the core connected to a disabled router (either because of permanent faults or testing the router). When disabling a router, it is reconfigured to act as a bypassing router, maintaining the connectivity between the routers in horizontal and vertical direction. This characteristic avoids performance drop in networks. Although, the focus of this paper is to tolerate the disabled routers and to survive the cores, the faulty links can be easily tolerated by small modifications in the algorithm that we left it as our future work. Beside routing-based approaches, link redundancy or architectural modifications are also suitable approaches to tolerate faulty links without dealing with the underlying routing algorithms.

III. THE ROUTER ARCHITECTURE

A. *Default Router Architecture*

The proposed CoreRescuer algorithm takes advantage of an adaptive routing algorithm with a low hardware overhead. In this configuration, the physical channel along the Y dimension is shared between two virtual channels while the physical channel along the X dimension is not shared. It implies that this algorithm requires one and two virtual channels along the X and Y dimension, respectively. This is the minimum amount of virtual channels to provide fully adaptiveness in 2D mesh networks. As shown in Figure 1(a), each router has seven pairs of channels, i.e. Local (L), East (E), West (W), North-vc1 (N1), North-vc2 (N2), South-vc1 (S1), and South-vc2 (S2). By default, the input and output channels are connected through a crossbar unit (Figure 1(a)).

CoreRescuer improvises some default paths such that when the router is disabled, the input channels are directly connected to the output channels (Figure 1(b)). Using these bypassing connections, the local core delivers its packets to the north neighboring router through the N1channel (i.e. L-to-N1). The core receives packets from the north neighboring router but using the N2 channel (i.e. N2-to-L). As shown in Table 1, the other static connections are as: N1-to-S1, S2-to-N2, S1-to-S2, E-to-W, and W-to-E.

Table 1. Default bypassing connections except top borderline routers

Input Channel	L	E	W	N1	N2	S1	S2
Output Channel	N1	W	E	S1	L	S2	N2

Default connections are valid for all the routers except the top-borderline routers as they do not have any north neighboring router to make a connection with. In this case, the local core sends and receives packets through its south neighboring router. Figure 1(c) shows the default connections of the top borderline routers, also listed in Table 2. The bypassing connections not only are beneficial to support the network with disabled routers but also are useful for reducing the latency. This is due to the fact that the bypassing connections directly connect the neighboring routers together without processing the packets in the disabled router. The north or south neighbor of the disabled router, which helps the packets to reach the core of the disabled router, is implicitly called *ladder router*.

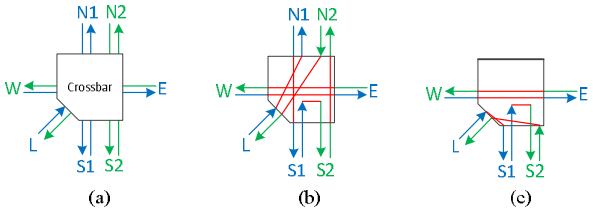

Figure 1. (a) A router using crossbar unit (b) default bypassing connections (c) default bypassing connections of top borderline routers

Table 2: Default bypassing connections of the top borderline routers

Input Channel	L	E	W	S1	S2
Output Channel	S1	W	E	S2	L

Figure 2 shows an example where three routers (i.e. at locations 4, 7, and 14) are disabled and the bypassing connections are employed. The router 12 performs as the ladder router to send and deliver packets to/from the core 7 while the router 9 is the ladder router of both the cores 4 and 14.

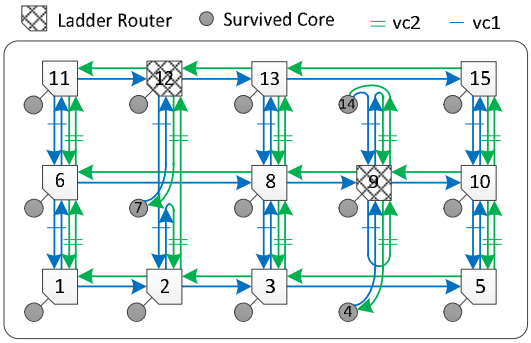

Figure 2. An example of three disabled routers and use of bypassing connections

It is worth mentioning that we assume that in a case of fault, a fault-detection mechanism is employed at each router and we use this information in CoreRescuer. In the reconfiguration phase, the input buffers of the router should be emptied before going to a bypassing mode. In this period, if the disabled router delivers packets to a wrong direction, packets still have a high chance to reach their destinations because of the non-minimal nature of the CoreRescuer algorithm. However, the packets will be dropped whenever a suitable path cannot be found at a router. After the reconfiguration, the algorithm should be able to deliver all packets to their destinations without any drop which is the goal of this paper.

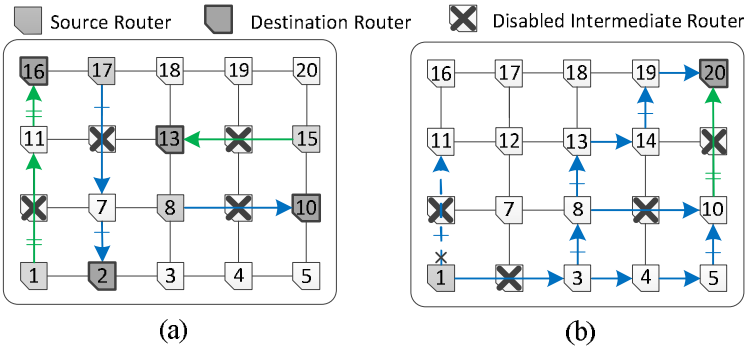

Figure 3. Different locations of a disabled intermediate router

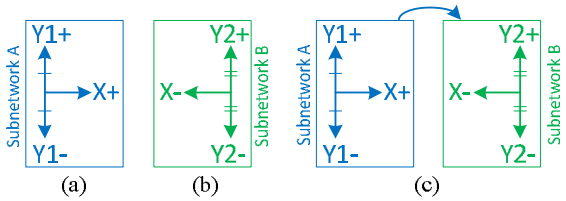

Figure 4. Two disjoint subnetworks formed by dividing the channels

B. *Rules Guranteeing Deadlock and Livelock Freedom*

In the CoreRescuer algorithm, the network can be partitioned into two disjoint subnetworks covering disjoint sets of channels as: ((X+)(Y1+)(Y1-) and (X-)(Y2+)(Y2-), called as subnetwork A (Figure 4(a)) and subnetwork B (Figure 4(b)), respectively.

A cycle cannot be formed within each subnetwork since to form a cycle at least four directions are needed (X+, X-, Y+, and Y-). However, each of the subnetwork A and subnetwork B cover three directions as (X+, Y1+, and Y1-) and (X-, Y2+, and Y2-), respectively. So there is no possibility of forming a complete cycle within each subnetwork. It is also necessary to prove that subnetworks are disjoint from each other, so a cycle cannot be formed between subnetworks. As shown in Figure 4(a) and (b), two subnetworks are completely disjoint and they do not share any common channel. As a result, the network is deadlock free if packets are solely belonging to either the subnetwork A or the subnetwork B. To improve the routing flexibility, packets can also switch between subnetworks in one way. This will not lead to a deadlock as to complete a cycle, switching should be done in both ways. As shown in Figure 4(c), in the CoreRescuer algorithm, packets in the subnetwork A can safely switch to the subnetwork B but not vice versa.

The CoreRescuer algorithm is livelock free as in the worst case packets belonging to the subnetwork A reach to the easternmost column and continue routing by switching to the subnetwork B. Since there is no chance of switching to the subnetwork A, the movement is limited and the algorithm is livelock free.

IV. THE CORERESCUER ALGORITHM

In this section, we first introduce the basic form of the CoreRescuer algorithm in a network without any disabled router. Then, we investigate all situations where an intermediate router, a source router, or a destination router is disabled. It will be shown that, all situations are covered by the CoreRescuer algorithm and no packet is dropped in the network.

A. *Default Path Choices of the CoreRescuer Routing Algorithm*

In CoreRescuer, the E, S, NE, and SE-bound packets are routed in the subnetwork A in which they can freely use any channels belonging to this subnetwork as the E, N1, and S1 channels (Figure 4(a)). The W, N, NW, and SW-bound packets are routed in the subnetwork B and they can use any of the W, N2, and S2 channels (Figure 4(b)) without any limitation. In addition, all packets in the subnetwork A can switch to the subnetwork B but not vice versa. As shown in Figure 5(a), the E, W, N, and S-bound packets route normally, respecting the channel assignment. For example, the S-bound packets use the S1 channels belonging to the subnetwork A while the N-bound packets use the N2 channels belonging to the subnetwork B. The NE, SE, NW, and SW-bound packets, however, are first routed to a position which is one hop apart from the destination router in both X and Y direction. Then packets are delivered to a ladder router which is located one hop away from the destination router in the Y dimension. All packets use these default paths as long as they do not face any disabled router from the source core to the destination core.

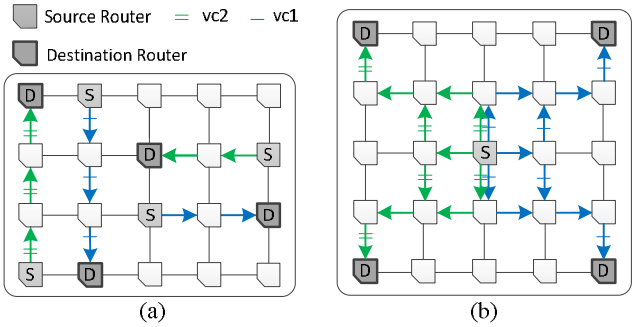

Figure 5. The default path choices of the CoreRescuer algorithm (The blue and green lines indicate the channels of the subnetwork A and the subnetwork B, respectively)

B. *Tolerating a Disabled Intermediate Router by CoreRescuer*

In this subsection, we investigate the cases that the disabled router is located in an intermediate router rather than the source and destination one. Figure 3(a) shows examples where a disabled router is simply bypassed by the E, W, N, and S-bound packets. Consistent with the router architecture of the

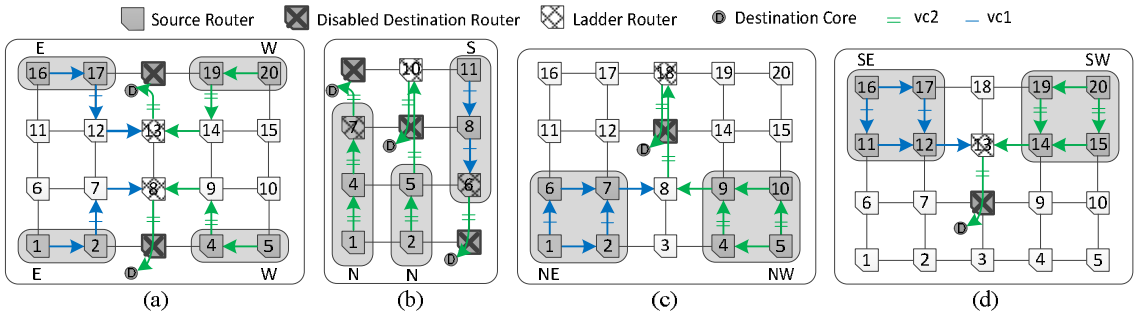

Figure 6. Different locations of a disabled destination router

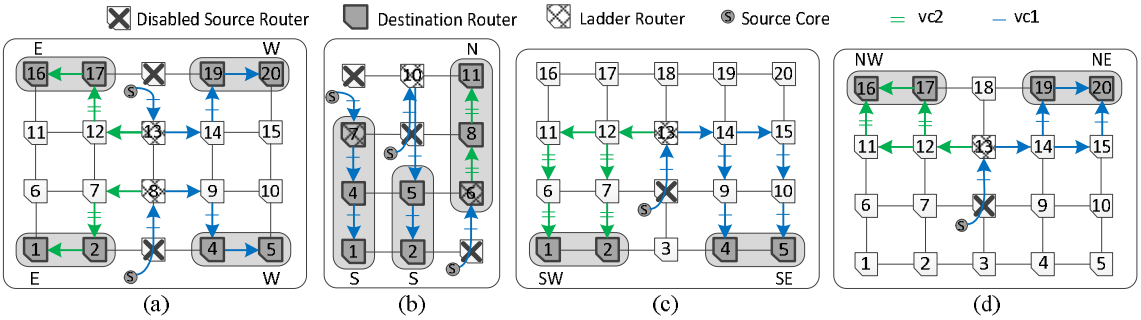

Figure 7. Different locations of a disabled source router

bypassing router, shown in Figure 1(b), northward packets are able to bypass the disabled router using the N2 channel while southward packets use the S1 channel in order to bypass the disabled router. Figure 3(b) shows an example where a NE packet is delivered from the source core 1 toward the destination core 20. Along the path, at each intermediate router, a functioning neighboring router is always chosen for the packet (e.g. at the router 4, the east neighbor is selected, and at the router 8, the north neighbor is chosen) unless both routers in minimal directions are disabled (e.g. at the router 1). In this case, the packet should be sent to the east direction to bypass the disabled router. The reason for this choice is that, if the north direction is selected, the packet has to be routed through N2, meaning that the packet switches to the subnetwork B. However, the subnetwork B does not cover the east direction and thus the packet cannot reach the destination from the router 11. Another important point in this example is that, even if the packet has two options at the router 13, the packet is sent to the east direction. This is the default choice as the router 14 is located in one hop distance from the destination router in both X and Y direction. Using the bypassing connections, the packet should be sent to the east direction at the router 14. However, since the router 15 is disabled, the packet is sent to the north direction. Similar examples can be used for the SE, NW, and SW packets. In short, a disabled router can be avoided by selecting a functioning neighbor along the path. If both neighbors in a minimal path are disabled, then packets are routed through the bypassing connections. This example shows that all packets are able to reach from any source core to any destination core regardless of a disabled intermediate router.

C. Tolerating a Disabled Destination Router by CoreRescuer

Figure 6 covers all locations from where a packet is sent to a core connected to a disabled destination router. Figure 6(a) shows the cases where a destination is located to the east and west of a source router. The survived core is accessible through its south neighboring router (using the N2 channel) when standing in the top borderline, otherwise through the north neighboring router (using the S2 channel).

Westward packets belong to the subnetwork B where the W, N2, and S2 channels can be taken without any limitation. Eastward packets belong to the subnetwork A where the E, N1, and S1 channels can be freely taken. Packets must switch to the subnetwork B to be able to take N2 or S2 and access the destination core. As it has been already discussed, switching from the subnetwork A to the subnetwork B is allowable.

Figure 6(b) shows examples of northward and southward packets. Southward packets, by default, are routed through the S1 channel and they can bypass the disabled router using the same channel. However, if the destination router is disabled, packets legitimately switch to the S2 channel which is directly connected to the destination core. For northward packets, by default, the N2 channel is used. Thereby if the disabled router is in the top borderline, the packet can directly reach the destination core. Otherwise, at first the disabled router is bypassed to reach the ladder router and then the packet is forwarded to the destination core.

As can be seen in Figure 6(c) and Figure 6(d), packets can reach the survived core at the destination by simply routing toward the ladder router. This example shows that all packets are able to reach from any source core to any destination core regardless of the disabled destination router and without violating any rule.

The CoreRescuer algorithm respects the general rules of channel assignments in all conditions without exception. This guarantees that the algorithm is deadlock and livelock free. In the other words, in all cases, packets are either belonging to the subnetwork A or the subnetwork B. In addition, packets

belonging to the subnetwork A can switch to the subnetwork B but not vice versa.

D. Tolerating a Disabled Source Router by CoreRescuer

Figure 7 shows all cases to prove that a packet can reach to any core in the network if delivered from a source core connected to a disabled router. In all cases, at first the packet is delivered to the ladder router. For this transmission, the N1 or S1 channel is firstly used which belongs to the subnetwork A. The E, S, NE, and SE-bound packets continue in the subnetwork A to reach the destination router while the W, N, NW, and SW-bound packets are required switching to the subnetwork B which is permitted. Thereby, all packets are able to reach from any source core to any destination core regardless of the disabled source router.

V. RESULTS AND DISCUSSION

To evaluate the efficiency of the CoreRescuer approach, a NoC simulator is developed with VHDL. For all the routers, the data width is set to 32 bits. An input port can accommodate 12 flits in each virtual channel. Moreover, the packet length is set to 5 flits and the mesh size is 8×8. As a performance metric, we use latency defined as the number of cycles between the initiation of a message issued by a core and the time when the message is completely delivered to the destination. The simulator is warmed up for 12,000 cycles and then the average performance is measured over another 200,000 cycles. CoreRescuer is designed based on using one and two virtual channels along the X and Y dimensions, respectively. In order to evaluate the reliability of CoreRescuer, we implemented the HiPFaR algorithm [12] which is a high performance fault-tolerant algorithm based on using the shortest paths. This algorithm requires the same number of virtual channels as CoreRescuer.

A. Reliability Evaluation Based on Supported Patterns

The uniform traffic profile is used to evaluate the reliability where each core generates data packets and sends them to another core using a uniform distribution. To evaluate the reliability of CoreRescuer, the number of disabled routers increases from one to three. For extracting the reliability based on the number of supported patterns, all patterns of disabled routers are examined in the simulation. Meaning that all possible combinations of two disabled routers (i.e. 2,016) and three disabled routers (i.e. 41,664) in a 8×8 mesh network are verified. A pattern is counted as supported if all packets reach their destinations. In the other words, even a single packet drop renders the pattern unsupported.

The reliability based on the number of supported patterns is illustrated in Figure 8. Both CoreRescuer and HiPFaR are reliable by 100% when there is a single disabled router in the network. When disabling two routers, CoreRescuer can tolerate 93.60% of total combinations while HiPFaR is able to tolerate two disabled routers by the probability of 95.48%. Although HiPFaR tolerates 3% more patterns than CoreRescuer, but it completely disconnects the cores connected to the disabled routers. It is interesting that the simulation results reveals a better reliability values than what it has been obtained by analytical formulas (i.e. 92.36% reliability by analytical analysis versus 93.60% reliability by simulation results). The reason for this difference is that the location of patterns (e.g. in borderline routers) may eliminate some positions of source and destination routers to ever existed regarding the disabled routers (e.g. preventing a southwest packet encountering a diagonal patterns in its paths).By disabling three routers in the network, the reliability of CoreRescuer and HiPFaR is obtained as 81.78% and 87.19%, respectively.

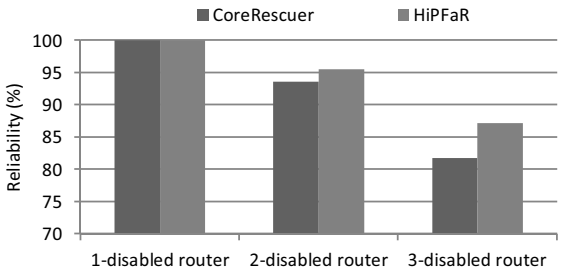

Figure 8. Reliability based on the number of supported patterns

B. Reliability Evaluation Based on Successful Packet Arrival

As an alternative metric, we measure the average number of successful packet arrivals at destinations as a fraction of the total number of injected packets. The average is taken over all combinations of patterns regardless of whether the pattern is supported or not. As can be seen from the results shown in Figure 9, on average CoreRescuer is able to successfully deliver 100%, 99.73%, and 99.21% of packets to their destinations under one, two and three disabled routers, respectively. In HiPFaR the reliability values are as 100%, 93.80%, and 87.19%. This obviously better reliability of CoreRescuer over HiPFaR is mainly due to the fact that HiPFaR cannot send and receive any packets by the cores connected to the disabled routers.

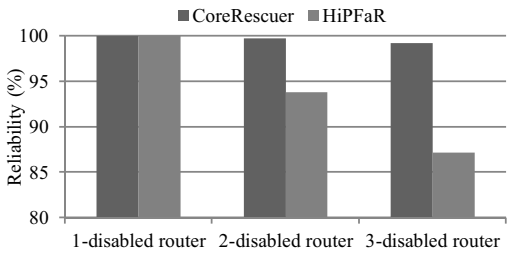

Figure 9. Reliability based on the number of successful packets arrival

C. Performance Analysis under Uniform Traffic Profile

The performance analysis under uniform random traffic is shown in Figure 10 where the average communication latencies of CoreRescuer and HiPFaR are evaluated. The average communication latency of the CoreRescuer and HiPFaR routing algorithms are measured under zero, one, and two disabled routers in the network. The average latency is taken over all the supported patterns. As predicted, results show the increased latency by disabling more routers in the network while HiPFaR always performs slightly better than CoreRescuer. However there is a big difference between these two methods. In HiPFaR, since the core and router are disconnected together from the network, the ratio of the number of injected packets into the network resources remains almost the same. However, in CoreRescuer, a router is disabled

while the core functions normally. This means that the traffic load remains the same while the network resources are decreased (i.e. the routers and some of their links are disconnected). Considering this difference, the CoreRescuer algorithm is perfectly able to control the traffic and avoid the performance drop. So, even if the number of packets over the network resources gets larger, it is compensated by using the bypassing connections and a lower number of hops.

D. Hardware Analysis

To assess the area overhead and power consumption, CoreRescuer and HiPFaR are synthesized using Synopsys Design Compiler. We compared the area overhead and power consumption of a router in both algorithms. For synthesizing, we use the TSMC45nm technology at the operating frequency of 1GHz and supply voltage of 0.9V. As indicated in Table 3, the power consumption and area overhead of CoreRescuer and HiPFaR are comparable. The small differences are because of employing the bypassing connections in CoreRescuer which is not used in HiPFaR.

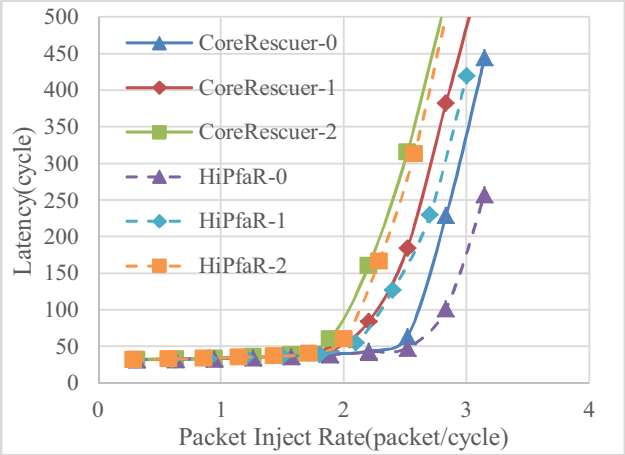

Figure 10. Latency analysis

Table 3. Power and area analysis

	CoreRescuer	HiPFaR
Dynamic Power	669.9 µW	669.4 µW
Static Power	12.4 µW	12.3 µW
Total Power	682.3 µW	681.8 µW
Area	46,16µm^2	46,13µm2

VI. SUMMARY AND CONCLUSION

In this paper, we proposed an approach to tolerate disabled routers in NoCs, either because of occurring faults, testing the core, or saving power. In traditional methods, a disabled router led to disconnecting the core from the rest of the network. However, this is a waste and may have a severe impact on the functionality or performance of the entire system. CoreRescuer allows the core to send/receive packets to/from all cores in the network without any packet loss. In addition to this capability, CoreRescuer is based on an adaptive algorithm, resulting in a better performance than static methods. Moreover, by simply bypassing the disabled router in horizontal and vertical direction, CoreRescuer avoids taking non-minimal routes to bypass disabled routers and the shortest paths are used as much as possible.

Acknowledgments

This work was supported by VINNOVA-MarieCurie within the CUBRIC and ERoT projects and Academy of Finland.

References

[1] M. Palesi and M. Daneshtalab (Eds.), "Routing Algorithms in Networks-on-Chip", Springer 2014.

[2] M. Radetzki, Ch. Feng, X. Zhao, and A. Jantsch, “Methods for fault tolerance in networks-on-chip”, *ACM Computing Surveys*, 46(1):8:1-8:38, 2013.

[3] M. Ali, M. Welzl, and S. Hessler, "A Fault tolerant mechanism for handling Permanent and Transient Failures in a Network on Chip", In Proc. ITNG, 2007.

[4] Chaochao Feng, Zhonghai Lu, Axel Jantsch, Jinwen Li, and Minxuan Zhang, "FoN: Fault-on-Neighbor aware Routing Algorithm for Networks-on-Chip", In Proc. SOCC, pp. 441-446, 2010.

[5] Ch. Feng, Zh. L, A. Jantsch, J. Li, M. Zhang, “A Reconfigurable Fault-tolerant Deflection Routing Algorithm Based on Reinforcement Learning for Network-on-Chip”, in Proc. of NoCArc, 2010.

[6] M. Pirretti et al. "Fault tolerant algorithms for network-on-chip interconnect." In Proc. VLSI, pp.46-51, 2004.

[7] D. Fick. A. DeOrio, G. Chen, v. Bertacco, D. Sylverster, D.Blaauw, “A highly resilient routing algorithm for fault-tolerant NoCs”, in Proc. of DATE, pp. 21-26, 2009.

[8] M.E. Gómez, J. Duato, J. Flich, P. López, A. Robles, N.A. Nordbotten, O. Lysne, T. Skeie, “An Efficient Fault-Tolerant Routing Methodology for Meshes and Tori”, Computer Architecture Letters, V. 3 , I. 1, 2004.

[9] K. Aisopos, A. DeOrio, L.S. Peh, V. Bertacco, “ARIADNE: Agnostic Reconfiguration in a Disconnected Network Environment”, In Proc. PACT, pp. 298-309, 2011.

[10] M. Ebrahimi, M. Daneshtalab, J. Plosila, F. Mehdipour, "MD: Minimal path-based Fault-Tolerant Routing in On-Chip Networks", in Proceedings of 18th Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 35-40, Jan 2013, Japan.

[11] M. Koibuchi, H. Matsutani, H. Amano, T.M. Pinkston, “A Lightweight Fault-Tolerant Mechanism for Network-on-Chip”, in Proc. of NoCS, pp. 13-22, 2008.

[12] M. Ebrahimi, M. Daneshtalab, J. Plosila, "High Performance Fault-Tolerant Routing Algorithm for NoC-based Many-Core Systems", in Proc. PDP, pp. 462-469, 2013.

[13] M. Ebrahimi, M. Daneshtalab, J. Plosila, and H. Tenhunen, "Minimal-Path Fault-Tolerant Approach Using Connection-Retaining Structure in Networks-on-Chip", in Proceedings of 7th ACM/IEEE International Symposium on Networks-on-Chip (NOCS), pp. 1-8, April 2013, US.

[14] M. Ebrahimi, M. Daneshtalab, "A Light-weight fault-tolerant routing algorithm tolerating faulty links and routers," Journal of Computing, 2014.

[15] D. Fick, A. DeOrio, J. Hu, V. Bertacco, D. Blaauw, D. Sylvester, “Vicis: a reliable network for unreliable silicon”, In Proc. DAC, 2009.

[16] L. Chen, and T. M. Pinkston, “NoRD: Node-Router Decoupling for Effective Power-gating of On-Chip Routers”, In Proc. of MICRO, pp. 270-281, 2012.

[17] A. DeOrio, K. Aisopos, V. Bertacco, and L.S. Peh, “DRAIN: distributed recovery architecture for inaccessible nodes in multi-core chips”, In Proc. of DAC, pp. 912-917, 2011.