d^2-LBDR: Distance-Driven Routing to Handle Permanent Failures in 2D Mesh NoCs

Rimpy Bishnoi*, Vijay Laxmi†, Manoj Singh Gaur‡ and José Flich§
* † ‡ Malaviya National Institute of Technology Jaipur, India.
§Universitat Politècnica de València, Spain.
Email: rimpybishnoi@gmail.com,(vlaxmi, gaurms)@mnit.ac.in, jflich@disca.upv.es

***Abstract*—With the advent of deep sub-micron technology, fault-tolerant solutions are needed to keep many-core chips operative. In NoCs, Logic Based Distributed Routing (*LBDR*) proved to be a flexible routing framework for 2D meshes with link and router faults. However, to provide full coverage, *LBDR* requires a module named *FORKS* which replicates some messages. This imposes the use of virtual cut-through switching and a complex router arbiter, increasing excessively the router cost, mainly in buffer area. Also, some failure combinations require the use of a non-trivial dynamic reconfiguration strategy to avoid deadlocks. We propose *d^2-LBDR* which adds, on every router, a distance register to the closest failure. This enables the support of more failure combinations without an excessive implementation cost. Indeed, we restore the use of wormhole switching, keeping router architecture simple, while achieving the same fault coverage as the best LBDR version, without requiring complex switching strategies nor any dynamic reconfiguration strategy. Results show that a small area overhead (3%) is enough for the implementation of a fully flexible routing method without any limiting support case when compared with *LBDR*. *d^2-LBDR* reduces area overhead over the best *LBDR* approach (300% overhead against 3%) while preserving fault coverage. Results show *d^2-LBDR* performance equal to *LBDR*.**

I. Introduction

In the move from computation-centric to communication-centric designs, the NoC used in many-core chips (CMPs and MPSoCs), plays a vital role in achieving the overall system performance. Moreover, with the advent of deep sub-micron technology, and with the increasing number of cores to be embedded, no matter how much care is taken during the design process of a chip, there is an increasing probability of fault occurrence during manufacturing and/or owing to ageing, wear-out and variability. Furthermore, failures in the underlying NoC may segregate a large proportion of the network and also impact its structural regularity, leaving the chip useless. Thus, it is of most importance to make the NoC resilient to failures. This demands for a flexible routing framework (routing algorithm and its implementation), which is capable of adapting (re-configuring) in the event of faults. Once a fault is handled, the NoC should continue its operation with graceful performance degradation.

Popular dimension-order routing algorithms (*XY*,*YX*) [1] are simple to design for 2D meshes but unable to handle any NoC failure. One possible solution to handle the irregularity caused by failures is the use of forwarding tables deployed either at source nodes (*source-based routing*) or at routers (*distributed-based routing*). Indeed, these table-based solutions allow any routing algorithm and topology to be supported. However, these solutions do not scale in terms of area, power

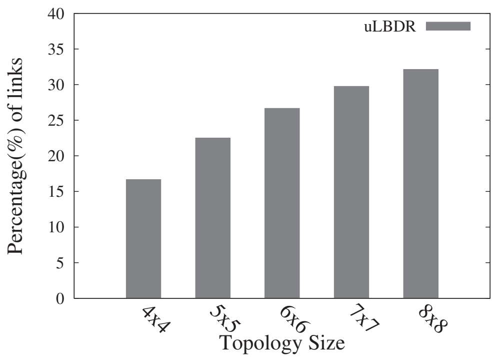


Fig. 1: % of single link failures with reconfiguration needs

consumption and latency, thus being impractical for current and future many core-designs [2].

Recently proposed scalable routing mechanism, *uLBDR* (universal Logic Based Distributed Routing) [3] offers a compact and efficient implementation of any distributed routing algorithm. *uLBDR* is an evolution of *LBDR* [2] aimed to extend its overall coverage support (number of supported link failures derived from a 2D mesh). Basically, it combines three modules: (1) *LBDR* to provide minimal path support, (2) *LBDR*$_{dr}$ to provide non-minimal path support and (3) *FORKS* to handle complex cases not covered otherwise.

With the *FORKS* module, full coverage is achieved. However, it introduces two important changes in the router design. First, the router arbiter complexity is increased to handle requests to multiple output ports, and second, the router needs to use the virtual cut-through (VCT) switching strategy, in lieu of wormhole (WH) switching. This is a significant overhead since it imposes buffers at routers to be sized to the maximum message size.

In addition, in order to provide 100% coverage for single link failures, *uLBDR* requires an additional cost in terms of reconfiguration [4]. Once a link failure is detected, the routing algorithm adapts to the new configuration. However, during this transient state, for some link failures, the network could lead to deadlock. To prevent this from happening, *uLBDR* relies on a dynamic reconfiguration process that keeps a safe order of messages during the transition. Fig. 1 shows the percentage of 1-link failure combinations requiring reconfiguration, converging to one third of links when the network size increases. Indeed, for an $N \times M$ mesh, the number of failure combinations requiring reconfiguration is $(N-2)\times(M-2)$. Notice that regardless of the number of such combinations, the NoC needs to handle them if 100% failure coverage has to be supported.

In this paper we introduce d^2-*LBDR*, which complements the lightweight $LBDR$ with a distance register and a new deroute strategy on every router. This new concept enables d^2-*LBDR* to offer the same coverage as of *uLBDR* but without requiring any complex switching technique, any complex arbiter design and without any reconfiguration need. Our main contributions, thus, are:

1) Identify the limitations of *uLBDR* and consequently proposing d^2-*LBDR* to solve it in a more efficient way.
2) Provide a distance register to failures in every router in order to take smarter decisions when routing messages.
3) Enhance the deroute strategy to increase the support for non-minimal paths in the original $LBDR_{dr}$.
4) Remove the need of reconfiguration to handle special cases with one or more link failures.

The net result achieved by d^2-*LBDR* is 100% coverage support for 1-link and 2-link failures with an overhead of 3% compared to the baseline *LBDR* approach (which achieves only 20% coverage). The *uLBDR* [3] approach has an overhead of 300% compared to baseline *LBDR*.

The rest of the paper is organized as follows. In Section II we present an overview of uLBDR. In Section III we describe d^2-LBDR. Then, we show evaluation results in Section IV. In Section V we describe the related work and provide concluding remarks in Section VI.

II. uLBDR

All *LBDR* versions use the concept of routing restrictions, an alternative way of representing a deadlock-free routing algorithm. Several turns are prohibited by the routing algorithm and these turns are represented using routing restrictions. The restriction indicates whether a message is allowed (or not) to take the turn. Fig. 3(a) shows in red the restrictions defined by the SR routing algorithm [5].

All *LBDR* versions codify the topology and the routing algorithm in two sets of configuration bits at the routers. In the case of *uLBDR*, the first set, *routing bits*, represents the routing restrictions at the neighbor routers. In total, 12 routing bits (R_{nn}, R_{ne}, R_{nw}, R_{ss}, R_{se}, R_{sw}, R_{ee}, R_{en}, R_{es}, R_{ww}, R_{wn}, R_{ws}), 3 bits for each output port, are used on every router. Routing bit R_{xy}, if set, indicates a message routed through x output port at the current router can take a turn using the y output port at the next router. The second set of bits, *connectivity bits*, keeps connectivity information of a router. Four connectivity bits (C_n, C_e, C_w, and C_s) indicate whether the router is connected with neighboring routers through its north, east, west, and south ports, respectively. To support

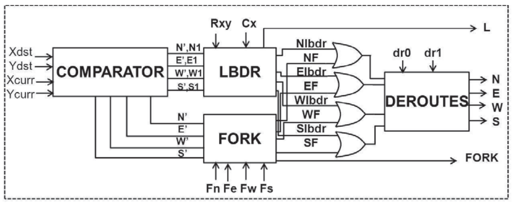


Fig. 2: uLBDR modules: LBDR, LBDR$_{dr}$ and FORKS

full coverage, *uLBDR* is based on three modules, *LBDR*, *DEROUTES* (*LBDR$_{dr}$*) and *FORKS*, as shown in Fig. 2.

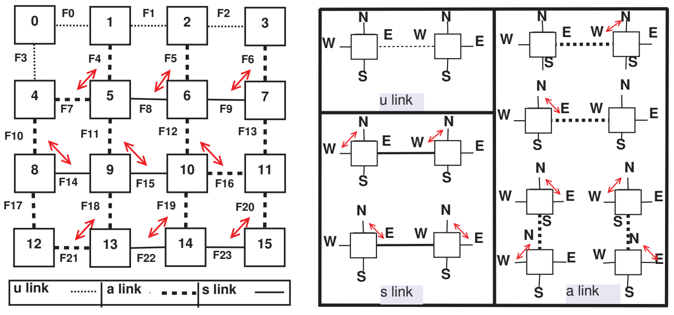


Fig. 3: (a) SR routing on 4×4 topology (b) Link types

uLBDR is located at each input port of a router and uses the routing and connectivity bits to implement the routing logic. Routing is performed based on the relative position of the current router and the destination router in the mesh (the *COMPARATOR* module). The *LBDR* module, by using routing and connectivity bits, provides minimal path support whenever possible. However, it leads to low coverage support, thus exhibiting the need for non-minimal path support. Thus, *LBDR$_{dr}$* is added (*DEROUTES* module). At each input port, a deroute option (an output port) is encoded using two bits, dr_0 and dr_1. When *LBDR* module does not provide a valid output port, the deroute is taken. To ensure deadlock freedom, the computation of a deroute option (performed offline) relies on the rules of the underlying routing algorithm. Thus, no deroute option should cross a routing restriction. uLBDR supports two approaches for deroutes. In the first one, a single global deroute option is encoded per router. As this option reduces flexibility, in the second approach a deroute option is encoded per input port. Even with the support of deroutes, there are still uncovered failure combinations. In particular, cases where two destinations located at the same quadrant from a router, require a different output port to successfully reach each destination. To support this, the *FORKS* module was added. It uses four additional configuration bits, F$_n$, F$_s$, F$_e$, and F$_w$ per router to encode these conflictive quadrants. *FORKS* replicates the message through the output ports of the quadrant whenever the message destination is located in that quadrant. However, the addition of *FORKS* poses severe limitations on the flexibility offered by *LBDR* as mentioned earlier. The arbiter is redesigned to handle requests to multiple output ports. But, most important, VCT is needed to guarantee deadlock freedom induced by new extra dependencies between branches of two fork operations. For a detailed description of uLBDR, please refer to [3].

III. d^2-LBDR

We assume the use of the Segment-Based routing algorithm (SR) [5] which groups routers and links in disjoint segments and applies an independent routing restriction on each segment. SR can be applied to any topology and is represented by its set of routing restrictions (Fig. 3.(a)). We assume a centralized infrastructure for fault detection and notification, as proposed in [6] and assumed also in LBDR.

A. The Premise

In order to efficiently offer fault tolerance while overcoming limitations of uLBDR, we perform an analysis of the types of

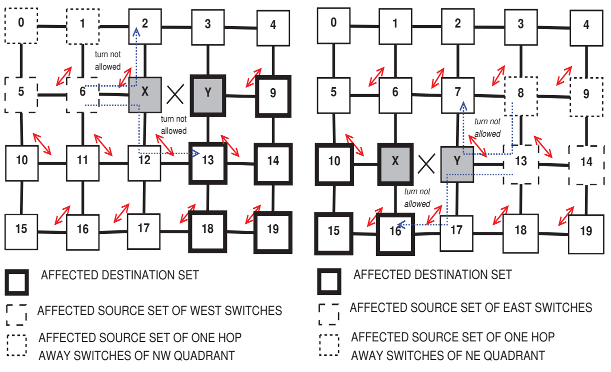
1-link failures that may be present in a 2D mesh when using SR (Fig. 3.a). Links are classified as shown in Fig. 3.(b):

1) **Unrestricted link** (u-link): No bidirectional routing restriction is located on any port of the routers connecting the link $\{F_0, F_1, F_2, F_3\}$.
2) **Anti-symmetric link** (a-link): One bidirectional routing restriction is located in one of the routers attached to the link $\{F_4, F_5, F_6, F_7, F_{10}, F_{13}, F_{16}, F_{17}, F_{20}, F_{21}\}$ or a bidirectional routing restriction is located on each router connected to the link but in different relative positions $\{F_{11}, F_{12}, F_18, F_{19}\}$.
3) **Symmetric link** (s-link): Two bidirectional routing restrictions are located on the same relative position and each on a different router connected to the link. These can be further divided into two groups:
    a) s^{NW} **link**: Bidirectional routing restrictions between $N \leftrightarrow W$ ports as in $\{F_8, F_9, F_{22}, F_{23}\}$.
    b) s^{NE} **link**: Bidirectional routing restrictions between $N \leftrightarrow E$ ports as in $\{F_{14}, F_{15}\}$.

Failures on u-links or a-links can be supported by the *LBDR* approach. To deal with such failures, the only modification required, at the routing algorithm level, is to remove the routing restriction located in the same segment of the failed link. The routing restriction can now be removed from such segments because now within the failed segment, cycle is broken by the failed link itself. This is the key property of SR algorithm and identified and well proved against deadlock freedomness in [3]. No reconfiguration process is needed since both algorithms (before and after the failure) are compatible in the placement of routing restrictions. However, dealing with failures on s-links is complex and requires more advanced modifications on the baseline routing algorithm [4]. Fig. 4(a) shows one example for an s^{NW}-link failure between routers X and Y. With this combination, messages from router 5 or 6 and with destinations located in either east direction or south-east quadrant of the failed link *(affected destination* set in figure), will stuck at node X. For these flows there is no deadlock-free path (either minimal or non-minimal) available from router X due to the two routing restrictions, one located at router X and the other at the south neighbor (router 12). Similarly, messages originated from routers 0 and 1 and forwarded to the same destination set will also stuck at router X. A similar scenario can be built for s^{NE}-links between routers X and Y as shown in Fig. 4(b). One solution for the above scenario is to shift the routing restrictions of neighboring segments as proposed in [4]. But as this solution requires moving routing restrictions, deadlock may occur during the transition, thus, needing an additional and non-trivial reconfiguration mechanism [7].

B. Proposed Methodology

In order to handle s-link failures, we analyze how the restrictions located at router X and its south neighbor affect connectivity. We notice that some routing restrictions (coded in routing bits) by the underlying routing algorithm should become a forbidden turn for a particular set of flows but remain open for the rest of flows. For instance, as shown in Fig. 4(a), the turn south-to-east from router 1 (R_{se} bit at router 1) should become a forbidden turn for destination 13 and allowed for destination 12. In this way we prevent messages reaching a blocking point at router X. Thus, masking routing bits depending on the flows would help.



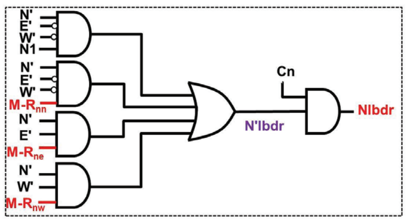
turn not allowed
turn not allowed
AFFECTED DESTINATION SET
AFFECTED SOURCE SET OF WEST SWITCHES
AFFECTED SOURCE SET OF ONE HOP AWAY SWITCHES OF NW QUADRANT
AFFECTED DESTINATION SET
AFFECTED SOURCE SET OF EAST SWITCHES
AFFECTED SOURCE SET OF ONE HOP AWAY SWITCHES OF NE QUADRANT

(a) $slink^{NW}$ (b) $slink^{NE}$

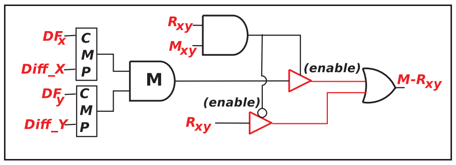
Fig. 4: Blocking situation at router a) X and b) Y

For such purpose, d^2-*LBDR* adds at each router one set of mask bits M, with one M_{xy} bit per R_{xy} routing bit, and two registers per router called distance bits (DF_x, DF_y). M_{xy}, when set to one, indicates the routing bit R_{xy} needs to be evaluated (possibly masked) before being used by the routing logic. Only R_{xy} bits set to one can be masked. Indeed, only when the two bits (R_{xy} and M_{xy}) are set to one, the routing bit may change, depending on the distance and relative location of the current router to the failed link.

The two registers, DF_x and DF_y, are used to keep the distance to the faulty link from the current router and help to identify at what extent the remaining path until destination will be fault-free. For a healthy chip, M_{xy} bits are set to zero and DF_x and DF_y registers are set to the maximum distance in X and Y dimensions, respectively. In the presence of at least one s-link failure, both mask bits and distance registers are modified only at affected routers. Those changes are computed offline.



(a) Routing Logic for port N



(b) Routing bit Masking Logic

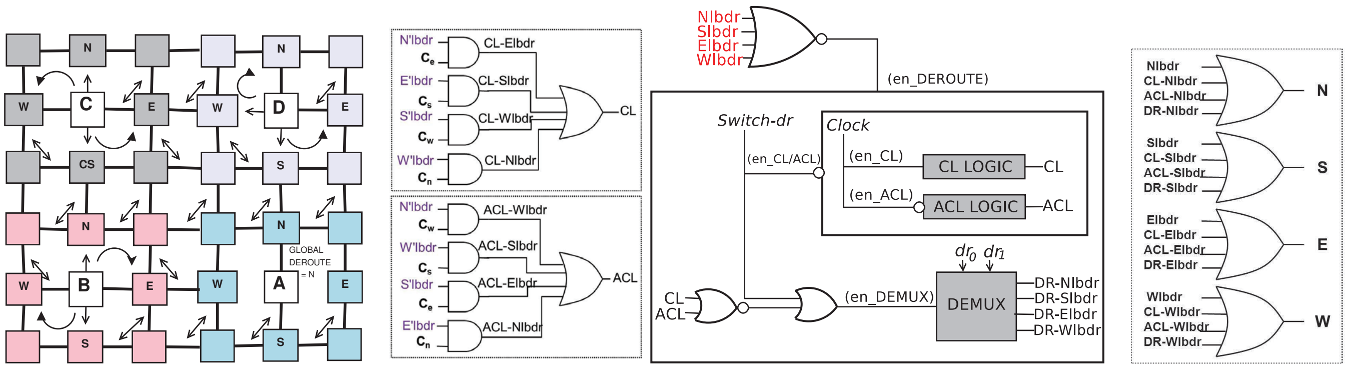
R_{xy}	M_{xy}	M	M-R_{xy}
0	X	X	0
1	0	X	1
1	1	0	0
1	1	1	1

(c) Truth table

Fig. 5: d^2-LBDR: Routing and Routing bit Masking Logic

C. d^2-LBDR Routing and Routing Bit Masking Logic

Fig. 5 shows the d^2-*LBDR* routing logic and the routing bit masking logic. The routing logic shown in Fig. 5(a) is similar



(a) Example of deroute cases (b) CL/ACL logic blocks (c) Complete Deroute Strategy (d) Final selection logic

Fig. 6: d^2-LBDR deroute strategy and final selection logic

to the one used in uLBDR but in our case, instead of directly feeding the routing bits into the logic, we mask them against faults (shown in red).

For masking, we have incorporated an additional masking logic as shown in Fig. 5(b). First of all, the distance between the current router and the destination router along X and Y dimensions ($Diff\_X$ and $Diff\_Y$) are compared against the fault distance (registers DF_x and DF_y). If the failed link is beyond the area defined between the current router and the destination router, the AND gate M is set. In this case, the routing bits will not be masked. If, however, the failed link is within the area, the routing bits (if set) will be masked (if the associated mask bit is set). Only when both M_{xy} and R_{xy} bits are set, the routing bit R_{xy} is masked. The truth table of the masking logic is shown in Table 5(c).

As an example, on link failure between routers X and Y in Fig. 4(a), routers 0 and 1 mask the routing bit R_{se}, as these turns become forbidden for destinations beyond the faulty link. Router 0 configures its DF_x register to 3 while router 1 to 2. In both routers, mask bit M_{se} is set to one. Notice that messages intended to any destination from the *affected destination* set will be sent through the east output port, taking over the failed link.

D. Deroute Strategy

With the previous mask bit strategy, *d^2-LBDR* successfully covers all 1-link failure cases without moving restrictions (which requires a reconfiguration strategy in uLBDR). Now, we delve into more complex cases where 2-link failures of any shape emerge. For this, we extend the deroute strategy of uLBDR to support non-minimal path and 100% coverage for all 2-link failures independent of any specific fault model. Notice that uLBDR covers some specific 2-link failures by moving routing restrictions and needing reconfiguration.

In d^2-LBDR, we opt for a single global deroute option for the entire router coded with two bits (dr$_0$, dr$_1$). In addition to this, we propose a different deroute strategy to support the following cases (see Fig. 6(a)):

- At router B, in order to avoid routing restrictions, messages destined to north need a deroute to east and messages destined to south need a deroute to west. We can solve this ambiguity by forcing a 90° clockwise rotation of the intended output port.
- Similarly, at router C, messages destined to north need a deroute to west and messages destined to south require a deroute to east. We can set a 90° anti-clockwise rotation of the intended output port.
- Unfortunately, we may find more complex cases. At router D, messages destined to west require a north deroute and messages destined to south require an east deroute, which means 90° clockwise for one case and 90° anti-clockwise for the other.

All these cases can not be handled by setting a single deroute option for the entire router and even not by having a deroute option per input port [6]. This inspired us to propose a deroute strategy that can be used to cover all cases shown in Fig. 6(a). Indeed, our deroute strategy is driven by the intended output port whereas the deroute strategy in LBDR$_{dr}$ is driven by the input port.

Fig. 6 shows the proposed deroute strategy of d^2-LBDR. It relies on CLockwise (CL) and Anti-CLockwise (ACL) deroutes. When the routing logic does not provide any valid minimal path for any flow, this logic is enabled (*en_DEROUTE* signal) and selects a particular deroute option according to the values set in *switch-dr* and *clock* bits. Different values of *switch-dr* and *clock* bits are used to configure a particular deroute mode as shown in Table I. Indeed, the deroute option can be configured to use any of the following modes:

TABLE I: Truth Table for deroute strategy of d^2-LBDR

en_DEROUTE	Clock	switch-dr	CL	ACL	en_DEMUX
1	X	X	X	X	X
0	X	1	X	X	Yes
0	1	0	1	0	No
0	1	0	0	0	Yes
0	0	0	0	1	No

- **Single global deroute option for the entire router** (Router A). The deroute is set to north by setting *switch-dr* to one as given in row 2 in Table I.
- 90° **clockwise rotation from the intended output port** (Router B). A clockwise deroute is configured by setting *switch-dr* to zero and *clock* to one (row 3 in the table). It enables messages destined to north (south) direction to use an east (west) deroute.
- 90° **anti-clockwise rotation from the intended output port** (Router C). An anti-clockwise deroute is configured

by setting *switch-dr* to zero and *clock* to zero (row 5 in the table). It enables messages destined to north (south) direction to use a west (east) deroute.

- **Both CL and ACL deroute** (Router D). Messages destined to west require a north deroute (90° clockwise rotation) and messages destined to south require an east deroute (90° anti-clockwise rotation). For the clockwise deroute, *clock* is set to one. In addition, *switch-dr* is set to zero to enable the clockwise/anti-clockwise circuit as shown in row 4 in the table. The anti-clockwise deroute is encoded with the dr_0, dr_1 bits. By setting *clock*, a correct deroute (N) will be provided for messages destined towards west as they will disable the DEMUX as CL is set to one. On the other hand, for messages destined to south and requiring a deroute to east, CL signal will be set to zero. Thus, CL along with ACL helps to enable the DEMUX and to use the deroute encoded in dr_0, dr_1 bits.

E. Deadlock-freedomness and Connectivity of d^2-LBDR

The proposed d^2-LBDR approach relies on the concept of routing restrictions defined by the routing algorithm [2]. As d^2-LBDR is an implementation mechanism of a given routing algorithm (SR in this paper), the guarantee of deadlock-freedomness relies on the underlying routing algorithm. In particular, in the location of routing restrictions. In case of no failure, d^2-LBDR ensures deadlock-free condition by preventing messages to cross any routing restriction by setting their corresponding R_{xy} bits to zero. Routing turns whose corresponding routing bits are set to zero will be discarded by the routing logic (AND gates in the routing logic). This guarantees that the mechanism is free from deadlocks. However in case of failures, d^2-LBDR does not remove any restriction from the network. Indeed, none the R_{xy} bits set to zero by the routing algorithm are set to one by d^2-LBDR logic. To avoid faulty routes, few additional routing restrictions are added for some part of topology affected from failure. Therefore, only connectivity could be compromised and not deadlock-freedomness. Indeed, the channel dependency graph is kept acyclic by ensuring the existence of all restrictions imposed by underlying routing. Connectivity is also guaranteed by the fact that the proposed d^2-LBDR mechanism is bundled with the checker tool (see the evaluation section) that tests connectivity property for every possible failure combination.

IV. Evaluation

In this section we provide coverage, hardware implementation analysis, and performance evaluation of d^2-LBDR. We use different versions of LBDR for comparison purposes.

A. Coverage Analysis

For coverage, we coded d^2-LBDR, LBDR$_{dr}$ and uLBDR in the checker tool used in [4] which tests the correctness (deadlock-freedom and connectivity) for every single and double link failure combination. For each failure combination the checker tool considers a topology to be supported only if the underlying mechanism provides always valid paths from each router to the rest of network routers.

As already mentioned, d^2-LBDR achieves full coverage for all 1-link failures. In the case of 2-link failures, as Fig. 7 shows, d^2-LBDR achieves also 100% coverage. uLBDR achieves the

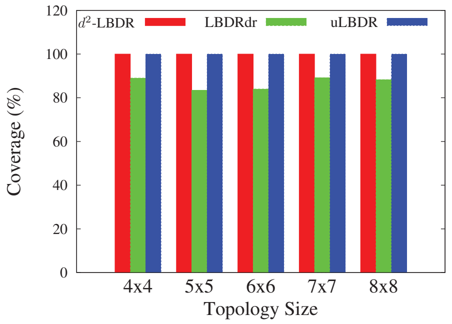


Fig. 7: Coverage of all mechanisms under 2-link failures

same results but it needs a built-in reconfiguration mechanism for some combinations. This is not the case for d^2-LBDR. On the other hand, uLBDR without forks module (LBDR$_{dr}$) is unable to provide 100% support.

B. Router Area Overhead

For router area overhead, we have implemented a 5-port router (45nm technology) with different routing versions: LBDR, LBDR$_{dr}$, uLBDR, and d^2-LBDR. Flit size is set to 64 bits and buffer slots are set to 4 flits modeling a WH router for d^2-LBDR, LBDR and LBDR$_{dr}$. For uLBDR we have modeled 16 slot buffers, assuming a VCT router. Table II shows the area results for the whole router designs. As we can observe, the

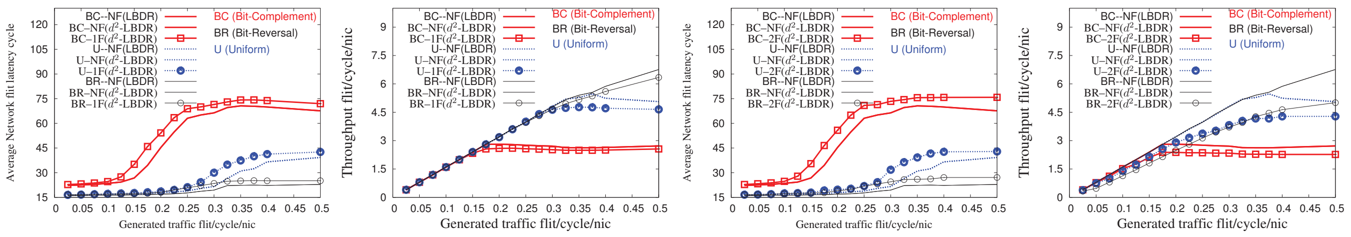
TABLE II: Area overhead analysis

ROUTER	Area(μm^2)	Normalized
LBDR - WH	20660	1.00
LBDR$_{dr}$ - WH	21011	1.017
d^2-LBDR - WH	**21429**	**1.0372**
uLBDR - VCT	65670	3.179

area overhead of d^2-LBDR when compared to LBDR baseline approach is 3.72%. This is in-line with the added resources needed by d^2-LBDR. However, this overhead is negligible when compared to uLBDR with 16-slot queues. In this case, uLBDR router is 3 times larger (as it requires VCT instead of WH), thus being much higher than d^2-LBDR. Notice that reconfiguration overhead in uLBDR has not been considered. Therefore, d^2-LBDR is a very appealing strategy to stay in WH switching whereas covering all single link and double link failures with no reconfiguration needs. Indeed, d^2-LBDR area overhead is similar to the one of uLBDR when the router buffer sizes are the same (however, this does not hold for VCT switching). Moreover, in terms of delay performance, d^2-LBDR falls within the slack provided by the arbiter stage in a pipelined router design similar to LBDR versions. Hence no additional delay penalty has been observed for d^2-LBDR.

C. Performance Evaluation

We have evaluated all the 1-link combinations and a representative set of 2-link cases. We show the average performance results for all those cases in Figs. 8. We assume 4×4 mesh (results hold for 8×8 meshes; not shown for space reasons). We compare LBDR with No Failures (NF-LBDR) and d^2-LBDR with no failures, 1-link failure, and 2-links failures. We use gMemNoCsim[8], with varying message sizes (3 and



(a) latency for 0-1-link failures (b) throughput for 0-1-link failures (c) latency for 0-2-link failures (d) throughput for 0-2-link failures

Fig. 8: Performance of all topologies derived from any single and double link failures under various traffic profiles

10 flits long) and with 40K transient and permanent messages loads. As we can observe, d^2-LBDR performance is identical to that of LBDR (same plots) for no failure cases. Performance smoothly degrades as link failures appear. This performance degradation is, however, due to topology degradation.

V. Related Work

Low-cost and efficient routing solutions based on Dimension-Ordered Routing, like FDOR [9], or on turn models [10], [11] have been proposed to provide coverage on irregular topologies. However, they either provide low coverage (single link faults) [9], [10] or disable a large number of fault-free nodes to extend the coverage [10], [11]. Recently, a ZoneDefence [12] routing is proposed that tries to reduce the number of disabled fault-free nodes by spreading the fault location information to nearby routers. However, still, these implementations are unable to offer full coverage support.

Solutions based on the use of routing tables at destinations [13], [14], [15] are proposed to provide full coverage support but are expensive in terms of area overhead, routing delay and power consumption, thus may not be suitable for large scale NoCs [2]. A routing table-based solution using deflection routing [13] reduces the area overhead by partitioning the network into several regions. However, deflection routing is prone to live-locks and tables do not scale. ARIADNE [16], based on the reconfiguration of routing tables utilizes up*/down* routing algorithm and reconfigures it upon fault. Though ARIADNE provides an improvement over area overhead of previous similar table based solutions, its underlying routing algorithm is not optimized for regular networks.
In the other direction, LBDR approaches [2] avoid tables by providing small logic blocks to extend coverage in 2D meshes. However, full coverage is achieved by adding large overheads (e.g. VCT switching). Therefore, most of the proposals either offer low coverage (turn model based) or do not scale (routing table based). Also, solutions which offer high coverage and scale limit the flexibility of the underlying mechanism (LBDR based) and require non-area optimized switching strategies. In contrast, d^2-LBDR achieves full coverage without imposing such limitations.

VI. Conclusions and Future Work

We have presented a Distance Driven LBDR (d^2-LBDR) to handle permanent failures on 2D meshes. d^2-LBDR offers full fault coverage for all topologies derived as a result of single and double link failures. d^2-LBDR removes the need of VCT switching, thus, being compatible with the more area efficient WH switching mechanism. Results demonstrate negligible area and performance overheads and full coverage support of all single and two link failures. As future work, we plan to investigate the applicability of this mechanism for other topology agnostic routing algorithms as well analyzing the coverage for cases with more than two link failures.

Acknowledgment

We would like to acknowledge support received from Indo-Spain DST project DST /INT /Spain/P35/11/1 and Spanish Ministerio de Economa y Competitividad (MINECO) under grant PRI-PIBIN-2011-0989.

References

[1] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[2] S. Rodrigo, S. Medardoni, J. Flich, D. Bertozzi, and J. Duato, "Efficient implementation of distributed routing algorithms for nocs," *Computers Digital Techniques, IET*, vol. 3, no. 5, pp. 460–475, 2009.

[3] S. Rodrigo, J. Flich, A. Roca, S. Medardoni, D. Bertozzi, J. Camacho, F. Silla, and J. Duato, "Cost-efficient on-chip routing implementations for cmp and mpsoc systems," *IEEE Trans.Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 4, pp. 534–547, April 2011.

[4] R. Bishnoi, V. Laxmi, M. S. Gaur, J. Flich, and T. Francisco, "A brief comment on "a complete self-testing and self-configuring noc infrastructure for cost-effective mpsocs," *ACM TECS*, in press.

[5] A. Mejia, J. Flich, and D. J., "On the potentials of segment-based routing for nocs," in *Proc. ICPP'08*, 2008, pp. 594–603.

[6] A. Ghiribaldi, D. Ludovici, F. Triviño, A. Strano, J. Flich, J. L. Sánchez, F. Alfaro, M. Favalli, and D. Bertozzi, "A complete self-testing and self-configuring noc infrastructure for cost-effective mpsocs," *ACM TECS*, vol. 12, no. 4, pp. 106:1–106:29, 2013.

[7] A. Strano, D. Bertozzi, F. Trivino, J. Sanchez, F. Alfaro, and J. Flich, "Osr-lite: Fast and deadlock-free noc reconfiguration framework," in *Proc. SAMOS'12*, 2012, pp. 86–95.

[8] M. Lodde and J. Flich, "Memory hierarchy and network co-design through trace-driven simulation."

[9] T. Skeie, F. Sem-Jacobsen, S. Rodrigo, J. Flich, D. Bertozzi, and S. Medardoni, "Flexible dor routing for virtualization of multicore chips," in *Proc SOC'09*, Oct 2009, pp. 073–076.

[10] C. Glass and L. Ni, "Fault-tolerant wormhole routing in meshes," in *Proc FTCS-23'93*, June 1993, pp. 240–249.

[11] J. Wu, "A fault-tolerant and deadlock-free routing protocol in 2d meshes based on odd-even turn model," *IEEE Trans. Comput.*, vol. 52, no. 9, pp. 1154–1169, Sep. 2003.

[12] B. Fu, Y. Han, H. Li, and X. Li, "Zonedefense: A fault-tolerant routing for 2-d meshes without virtual channels," *IEEE TVLSI*, vol. 22, no. 1, pp. 113–126, Jan 2014.

[13] Y. B. Kim and Y.-B. Kim, "Fault tolerant source routing for network-on-chip," in *Proc. DFT '07*, 2007, pp. 12–20.

[14] C. Feng, Z. Lu, A. Jantsch, M. Zhang, and Z. Xing, "Addressing transient and permanent faults in noc with efficient fault-tolerant deflection router," *IEEE TVLSI*, vol. 21, no. 6, pp. 1053–1066, June 2013.

[15] D. Fick, A. DeOrio, G. Chen, V. Bertacco, D. Sylvester, and D. Blaauw, "A highly resilient routing algorithm for fault-tolerant nocs," in *Proc. DATE '09.*, April 2009, pp. 21–26.

[16] K. Aisopos, A. DeOrio, L.-S. Peh, and V. Bertacco, "Ariadne: Agnostic reconfiguration in a disconnected network environment," in *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, Oct 2011, pp. 298–309.