Highly Fault-tolerant NoC Routing with Application-aware Congestion Management

Doowon Lee, Ritesh Parikh and Valeria Bertacco
Department of Computer Science and Engineering, University of Michigan
{doowon, parikh, valeria}@umich.edu

***Abstract*—Silicon devices are becoming less reliable as technology moves to smaller feature sizes. As a result, digital systems are increasingly likely to experience permanent failures during their lifetime. To overcome this problem, networks-on-chip (NoCs) should be designed to, not only fulfill performance requirements, but also be robust to many fault occurrences. This paper proposes a fault- and application-aware routing framework called FATE: it leverages the diversity of communication patterns in applications for highly faulty NoCs to reduce congestion during execution. We propose a set of novel route-enabling rules that greatly reduce the search for deadlock-free, maximally-connected routes for any faulty 2D mesh topology, by preventing early on the exploration of routing configuration options that lead eventually to unviable solutions. Our experimental results show a 33% improvement on average saturation throughput for synthetic traffic patterns, and a 59% improvement on average packet latency for SPLASH-2 benchmarks, over state-of-the-art fault-tolerant solutions.**

I. Introduction

Advances in semiconductor fabrication technology have enabled the design of modern chip multiprocessors (CMPs) and systems-on-chip (SoCs) consisting of billions of transistors. They deploy tens, or even hundreds, of communicating components and, therefore, efficient on-chip communication is increasingly becoming a critical design bottleneck. Networks-on-chip (NoCs) are a promising interconnect solution because they provide massively concurrent, scalable and power-efficient communication. However, the increasing susceptibility to faults of nano-scale semiconductor devices makes it extremely challenging to maintain the correctness and low-latency characteristics that are desired for NoCs. To make matters worse, NoCs constitute a single-point-of-failure for the entire system.

Fault-tolerant NoC routing solutions [1,11,12] tackle this challenge by leveraging their inherent routing flexibility. In other words, the routing solutions react to faults by limiting communication to flow only along fault-free paths. In this context, topology-agnostic routing algorithms [6,13] offer highly flexible routing, preserving network connectivity even in the presence of many faults. Nonetheless, they often lead to severe performance degradation after only a few faults, making the continued deployment of the chip impractical. This degradation is mainly due to increased traffic congestion on the remaining healthy paths.

CMP applications [4,14] exhibit communication patterns [3] where most packets flow among only a subset of the NoC nodes. This aspect could be leveraged to focus the limited routing options available towards high-traffic communication paths. This is the key observation that led to our solution: if the high-traffic communication patterns in a faulty NoC are known (or can be estimated), then we can **select the routing function so as to provide both deadlock freedom and maximum path diversity** (and thus low congestion) among the high-traffic nodes. Consequently, we can provide high-bandwidth, low-latency communication even in faulty networks.

In this paper, we present FATE, **F**ault- and **A**pplication-aware **T**urn model **E**xtension. FATE improves the performance of faulty 2D mesh networks by leveraging the application's communication patterns. We also demonstrate a method to quickly prune the exploration of viable routes in faulty networks, in order to consider only deadlock-free options, and we reduce the number of routes evaluated by two orders of magnitude over prior application-aware solutions.

II. Related Work

Fault-tolerant routing. Fault-tolerant, deadlock-free routing solutions have been extensively investigated in the past. Glass and Ni propose three turn-models that provide adaptive, fault-tolerant routing [7]. Note, however, that their fault-tolerance is limited to only a few faults [12]. In contrast, Ariadne [1], uDIREC [11] and CBCG [12] put no constraints on the number and location of faults, and hence, they are more reliable. Similar to the above on-chip solutions, in the off-chip network domain, topology-agnostic routing algorithms [6,13] focus mostly on placing routing restrictions using topological characteristics, while ignoring traffic flow. None of the above take traffic flow into consideration to optimize for throughput.

Application-aware routing. Application-aware routing solutions tune the routing function to traffic flow. APSRA [10] strives to meet the bandwidth requirement of an application, while achieving deadlock freedom by breaking cyclic dependencies. However, deploying this solution to identify optimal routes in faulty networks has an extremely high computational cost. In contrast, BSOR [9] selects load-balanced, oblivious paths via either mixed integer linear programming (MILP) or Dijkstra's shortest path algorithm to approximate optimally balanced paths. BSOR, however, suffers from low performance as they do not manage runtime congestions well. Although all of these application-aware solutions can be deployed in faulty networks, they often require intractable computation.

III. Overview and Background

FATE is a software-based solution that generates a deadlock-free routing function, while maximizing the number of distinct paths available between nodes with high communication requirements, for applications running on faulty 2D mesh networks. Figure 1 shows how FATE operates: it is triggered by a new fault occurrence or an application launch. It then uses the CMP's idle cores to compute a new routing function. FATE's efficient heuristic quickly discovers a near-optimal routing function based on an iterative exploration, where turn

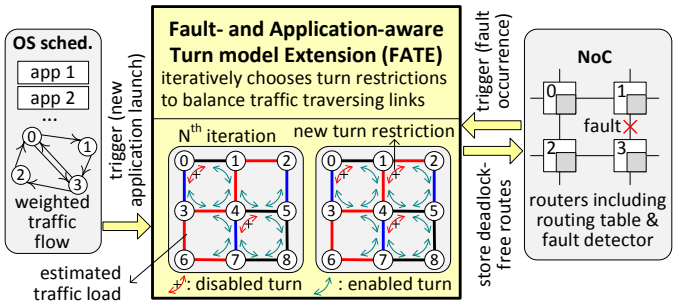


Fig. 1: **Overview of FATE**.

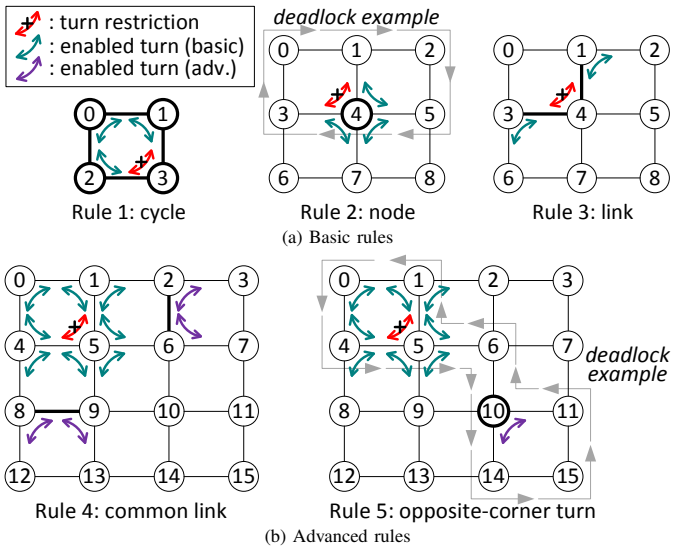


Fig. 2: **FATE's turn-enabling rules**.

restrictions are placed one at a time. After placing each turn restriction, FATE deploys its *turn-enabling rules* (Section IV) to enable turns that must be active in order to maintain connectivity in light of the most recent turn-restriction choice. Moreover, in choosing the location of each new turn restriction, FATE leverages traffic load estimates that it derives from the communication patterns extracted via application analysis (Section V). Finally, the network's routers are re-programmed using the new routing function.

FATE assumes that information about the application's communication patterns is available: indeed, these can be observed in CMP and SoC applications with runtime profiling, and then modeled through Markov chains [2,3]. As in many fault-tolerant routing solutions, we also assume that the NoC is equipped with a fault diagnosis solution. In addition, we assume that our solution is deployed in systems where the OS is notified of new fault detections and can update routing tables accordingly.

Deadlock avoidance. Deadlock situations can happen when packets wait for each other in a cyclic manner. Such situations can be avoided by breaking cyclic resource dependencies [5], disabling at least one of the turns that contribute to the cycle. This deadlock-avoidance technique has been proposed in the past: for instance, the turn models [7] forbid certain turns in 2D meshes to break every possible cycle. Similarly, FATE achieves deadlock freedom by disabling turns, but it provides much more flexibility in the turn restriction than the turn models.

IV. FATE's Turn-Enabling Rules

This section discusses the rules that determine which turns must be enabled as a consequence of another turn being disabled. These rules, grouped into *basic* and *advanced*, can be applied in any order, and are illustrated in Figure 2. We first describe how the rules operate in regular meshes, and then extend them to faulty topologies. Note that our approach minimizes the number of turn restrictions, and thus maximize cumulative bandwidth to all destinations.

1) Basic rules identify which turns must be enabled because of a turn restriction on the surrounding cycle, node and links, and they are illustrated in Figure 2a.

Rule 1 (**cycle**) — *Once a turn in a cycle is disabled, all other turns in the same cycle should be enabled*, so that all nodes in the cycle can still communicate.

Rule 2 (**node**) — *Once a turn in a node (router) is disabled, all other turns insisting on the same node should be enabled.*

Rule 3 (**link**) — *Once a turn adjacent to a link is disabled, the turn to the same link, on the opposite side with respect to this turn and not on the same router should be enabled.*

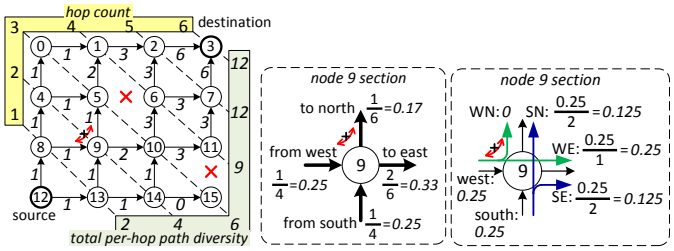
Note that any violation of these rules would lead to a superfluous number of turn disabling. For instance, if both the turns 1-4-3 and 1-4-5 were disabled in the middle network of Figure 2a, then the network would still allow a dependency along the gray-arrow path in the figure.

2) Advanced rules force FATE to enable turns that are remote with respect to the restricted turn. They allow to aggressively prune the search space towards a solution with minimal disabled turns. Figure 2b illustrates the two rules below.

Rule 4 (**common link**) — *If a cycle has only two undecided turns that share a common link, then all turns that are adjacent to that link and lie outside the cycle, should be enabled*. This rule can be inferred from Rules 1 and 3.

Rule 5 (**opposite-corner turn**) — *If two turns are located on opposite nodes in a cycle, and the two turns are not adjacent to any of the cycle's links, then only one of them can be disabled.* The reasoning behind this rule can be understood by considering the example on the right part of Figure 2b: if we had disabled both the opposite-corner turns 1-5-4 and 11-10-14, then we could no longer avoid a deadlock situation as an example shown in the figure.

Turn-enabling Rules in Faulty Topologies. Among the basic rules, Rule 1 is the one that is most affected by faults: cycles may become merged because of faults. Moreover, faulty networks may have both concave and convex cycles, and Rule 1 must be appropriately applied by considering turns common to both a concave cycle and a convex cycle as breaking only one of these cycles, not both. Rules 2 and 3 remain unchanged for faulty topologies. In addition, we limit the application of Rule 4 to links contributing to cycles that have not been affected by a fault. The reason we limit the application of Rule 4 is because it could become complex to identify which turns should be enabled when a link spans multiple routers. Finally, we simply apply Rule 5 as we described for non-faulty networks. However, some turns enabled are not necessarily



(a) Path diversity (b) Link load (c) Turn load

Fig. 3: **Link-load and turn-load estimation example**.

causing deadlock in faulty networks, and thus they should not be enabled. If the enabled turns should have been disabled, our backtracking step would correct the situation.

V. FATE ROUTING

In this section, we propose FATE's heuristic algorithm to identify a routing function with deadlock-free routing and minimal routing restrictions. FATE relies on the information it receives about the application's communication patterns to strive to place turn restrictions on low-traffic links. When the FATE algorithm begins, all turns are undecided. Turn restrictions are then placed one at a time, starting from the regions transferring the most traffic. Upon placing each restriction, the turn-enabling rules are applied to enable the related set of turns. This process is repeated until each turn is either enabled or disabled.

A. Link, Turn and Cycle Load Estimates

To estimate the load on each network's link, turn and cycle, we consider one source-destination pair provided by the application at a time. For each pair, we compute all the possible paths that packets can take from source to destination, and we then derive the fraction of traffic that would go through each link. The computation of all the loads proceeds with the four steps below (see Figure 3).

Step 1 — Compute path diversity. We calculate the number of different routes (*i.e.*, path diversity) to reach each link from the source node.

Step 2 — Compute link-load estimates. We now use the results of Step 1 to estimate the load on each link based on the path diversity available. We calculate the total path diversity at each hop from the source (as illustrated in Figure 3a), and divide the link's path diversity by the total diversity.

Step 3 — Compute turn-load estimates. To estimate the load at each turn, we divide the input load from the source direction of the turn by the number of output links allowed.

Step 4 — Compute cycle-load estimates. For each cycle, the load is computed by simply summing the loads on all the turns in the cycle.

B. FATE Route-calculation Algorithm

Once all load estimates have been computed, we can apply the FATE routing algorithm, as illustrated in Figure 4. Note that we weigh the load estimates by multiplying each estimate by the traffic weight associated to its source-destination pair. The algorithm starts by selecting a turn to disable, choosing the turn with the lightest impact on the heaviest link load, among

```
1: repeat until there is no undecided turn
2:    compute_link,turn,cycle_loads()
3:    cycle = cycle_with_heaviest_load()
4:    disabled_turn = turn_with_smallest_link_load_increase(cycle)
5:    enabled_turns = apply_turn_enabling_rules(disabled_turn)
6:    if( not (check_deadlock() or check_disconnected()) )
7:       disable(disabled_turn), enable(enabled_turns)
8:    else update_conflict_history(), backtrack()
```

Fig. 4: **FATE routing algorithm**

those in the highest-load cycle (lines 2-4). Once the turn to be disabled is selected, we apply the turn-enabling rules to enable as many other turns as possible (line 5). If the set of turns enabled/disabled leads to a deadlock or a disconnected network (line 6), we backtrack, and update the list of conflicting selections (line 8).

VI. EXPERIMENTAL EVALUATION

We evaluated FATE with a cycle-accurate NoC simulator [8] modeling an 8×8 2D mesh NoC. The network's 3-stage routers are input-buffered, and each input port within a router contains 2 VCs per protocol class, and 5 flits per VC. We deploy a minimal adaptive routing approach and a local congestion monitoring scheme, so that output channels towards non-congested input buffers are favored. We analyzed FATE's performance on faulty 2D mesh networks by injecting a varying number of link faults: 1 (1%), 3 (3%), 6 (5%), 11 (10%) and 17 (15%) faulty links. We evaluated our testbed with both synthetic traffic [8] and traces from the SPLASH-2 benchmark suite [14]. We used five different synthetic patterns shown in Figure 5. Our synthetic traffic consists of a mix of equal amounts of 1- and 5-flit packets. The eleven SPLASH-2 traces were collected for 10 million cycles after spawning threads and initializing caches. The traffic weights were calculated by counting the number of flits per each source-destination pair.

A. Performance on Faulty Networks

We compared our solution against fault-tolerant, application-oblivious routing solutions that are based on the construction of spanning trees: up*/down* with breadth-first search (BFS) [1], and up*/down* with depth-first search (DFS) [13]. We also compared our solution against two application-aware routing solutions: Application-Specific Routing Algorithm (APSRA) [10] and Bandwidth-Sensitive Oblivious Routing (BSOR) [9].

Figure 5 reports the average saturation throughput across various fault rates and traffic patterns. In the left half of the figure, we observe that the performance of our scheme degrades more gracefully than both application-oblivious spanning-tree solutions (BFS and DFS) as faults increase. FATE achieves a 10% improvement over BFS when there is only one faulty link. FATE's improvement over BFS increases to 33% when 15% of the links are faulty. FATE also achieves consistently higher throughput at all fault rates than both APSRA and BSOR. FATE provides a 9% higher throughput at the 15% faulty-link rate. Finally, BSOR provides lower throughput than both APSRA and FATE. FATE also performs better than others across various traffic patterns at the 15% faulty-link rates, as shown in the right half of Figure 5. We observe that traffic

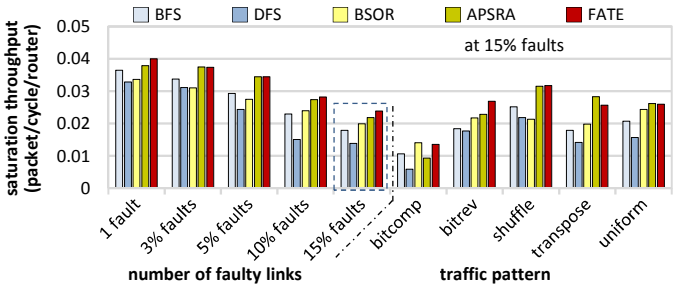


Fig. 5: **Saturation throughput for synthetic traffic patterns over various fault rates and traffic patterns**.

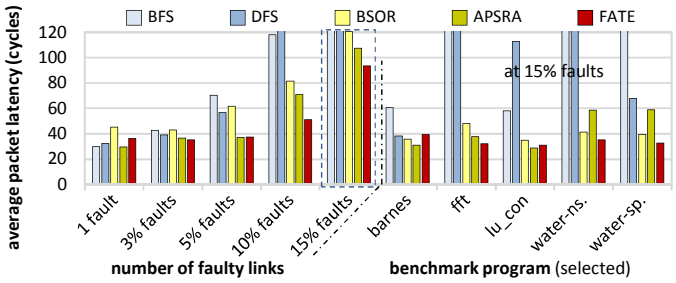


Fig. 6: **Packet latency for SPLASH-2 traces over various fault rates and benchmarks**.

patterns where packets utilize a few turns more frequently (*e.g.*, *transpose* and *bitrev*) are those that benefit the most from FATE.

Figure 6 reports the average packet latency from our trace-driven SPLASH-2 simulations across various fault rates and benchmarks. As shown in the left half of the figure, FATE experiences negligible latency increase up to the 5% faulty-link rate, while the latency increase is more significant beyond that point. In addition, FATE attains lower latency than APSRA and BSOR at almost all fault rates except one faulty link. Finally, we show results for five selected benchmarks at the 15% faulty-link rate on the right side of the figure. FATE consistently provides much lower latency than BFS and DFS. For instance, when running *fft*, each node accesses frequently memory nodes located at the corners of the mesh, and communicates with a few other nodes. As a result, FATE enables more routes among these frequently communicating nodes.

B. Overheads

Software computation overhead. We evaluated the overhead of computing the routing function compared against APSRA [10]. Table I reports the average computation time to derive a routing function, for both FATE and APSRA. Execution times were measured on an Intel Xeon E5520. Overall, FATE is a significantly faster solution than APSRA. In the right portion of the table, we also compare the average number of attempts of turn-disabling placement. Note that FATE's number of attempts is minuscule compared to APSRA: we owe this result to the turn-enabling rules, which greatly prune the search space for an optimal set of turn-disabling placements. We capped those algorithm's runs at 200,000 placement attempts and we report the fraction of occurrences where we reached the cap value.

Area overhead. We evaluated the area overhead of routing tables and route-computation logic, targeting the Nangate 45nm library using Synopsys DC at an operating frequency of 400MHz. With this configuration, our routing computation adds approximately 6% area overhead, mostly for the routing table. Note that the other fault-tolerant, adaptive routing solutions we compared against (BFS, DFS and APSRA) entail similar overhead, as they utilize similar routing intrastructures as FATE.

TABLE I: **computation overhead for FATE and APSRA**

	average time (sec)		average number of placements attempted		% runs reaching 200k cap	
	APSRA	FATE	APSRA	FATE	APSRA	FATE
fault-free	>500	3.61	71,321	117	19%	0%
1 fault		3.27	94,639	107	31%	0%
3% faults		3.29	107,956	107	41%	0%
5% faults		3.21	120,877	105	48%	0%
10% faults		3.62	151,802	118	69%	0%
15% faults		2.93	159,667	96	74%	0%

VII. Conclusions

We proposed FATE, a high-performing, adaptive routing solution for faulty networks-on-chip, that leverages the knowledge of an application's communication patterns. We developed turn-enabling rules to quickly determine the optimal set of turns that should be disabled to break all deadlocks. Our application-aware heuristic balances load evenly among network resources using our novel load-estimation metrics, and chooses the most promising turn-restriction locations. Experimental results show improvements in throughput (up to 33%) and latency (up to 59%) for synthetic traffic patterns and SPLASH-2 benchmark traces over state-of-the-art solutions. Finally, FATE keeps a low cost profile (6% area overhead).

Acknowledgements. This work was supported by STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

References

[1] K. Aisopos, A. DeOrio, L.-S. Peh, and V. Bertacco. Ariadne: agnostic reconfiguration in a disconnected network environment. In *Proc. PACT*, 2011.
[2] M. Badr and N. Jerger. SynFull: synthetic traffic models capturing cache coherent behavior. In *Proc. ISCA*, 2014.
[3] N. Barrow-Williams, C. Fensch, and S. Moore. A communication characterisation of Splash-2 and Parsec. In *Proc. IISWC*, 2009.
[4] C. Bienia, S. Kumar, J. Singh, and K. Li. The PARSEC benchmark suite: characterization and architectural implications. In *Proc. PACT*, 2008.
[5] W. Dally and C. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. Computers*, C-36(5), 1987.
[6] J. Flich *et al*. A survey and evaluation of topology-agnostic deterministic routing algorithms. *IEEE Trans. PDS*, 23(3), 2012.
[7] C. Glass and L. Ni. The turn model for adaptive routing. In *Proc. ISCA*, 1992.
[8] N. Jiang *et al*. A detailed and flexible cycle-accurate network-on-chip simulator. In *Proc. ISPASS*, 2013.
[9] M. Kinsy *et al*. Optimal and heuristic application-aware oblivious routing. In *IEEE Trans. Computers*, 2013.
[10] M. Palesi *et al*. Design of bandwidth aware and congestion avoiding efficient routing algorithms for networks-on-chip platforms. In *Proc. NOCS*, 2008.
[11] R. Parikh and V. Bertacco. uDIREC: unified diagnosis and reconfiguration for frugal bypass of NoC faults. In *Proc. MICRO*, 2013.
[12] P. Ren *et al*. Fault-tolerant routing for on-chip network without using virtual channels. In *Proc. DAC*, 2014.
[13] J. Sancho, A. Robles, and J. Duato. An effective methodology to improve the performance of the Up*/Down* routing algorithm. *IEEE Trans. PDS*, 15(8), 2004.
[14] S. Woo *et al*. The SPLASH-2 programs: characterization and methodological considerations. In *Proc. ISCA*, 1995.