Brisk and Limited-Impact NoC Routing Reconfiguration

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

***Abstract*—The expected low reliability of the silicon substrate at upcoming technology nodes presents a key challenge for digital system designers. Networks-on-chip (NoCs) are especially concerning because they are often the only communication infrastructure for the chips in which they are deployed. Recently, routing reconfiguration solutions have been proposed to address this problem. However, they come at a high silicon cost, and often require suspending the normal network activity while executing a centralized, resource-hungry reconfiguration algorithm. This paper proposes a novel, fast and minimalistic routing reconfiguration algorithm, called BLINC. BLINC utilizes precomputed routing metadata to quickly evaluate localized detours upon each fault manifestation. We showcase the efficacy of our algorithm by deploying it in a novel NoC fault detection and reconfiguration solution, where BLINC enables uninterrupted NoC operation during aggressive online testing. If a fault seems likely to occur, we circumvent it in advance with the aid of our BLINC reconfiguration algorithm. Experimental results show an 80% reduction in the average number of routers affected by a reconfiguration event, compared to state-of-the-art techniques. BLINC enables negligible performance degradation in our detection and reconfiguration solution, while solutions based on current techniques suffer a 17-fold latency increase.**

I. Introduction

Increasing transistor densities have significantly affected the architecture of silicon chips. In particular, processor designs have transitioned to multi-core architectures, as a solution to improve performance while keeping power dissipation in check. Networks-on-Chip (NoCs) are a promising communication substrate for multi-core architectures, providing high-bandwidth and concurrent communication among chip components. NoCs are often the only communication medium among the units on chip, and hence a malfunctioning NoC may render the entire chip dysfunctional. Further, due to the waning reliability of silicon, ensuring the correct operation of NoCs at runtime is becoming increasingly challenging.

Correct NoC operation in face of unreliable silicon can be guaranteed by providing routing reconfiguration to prevent and circumvent failures. Unfortunately, the solutions available today tend to be resource-heavy and impact the entire interconnect. These techniques are based on the assumption that fault occurrences are rare events. Thus, they strive to provide optimal or quasi-optimal post-fault routing reconfiguration, at a high reconfiguration latency cost. For instance in [1], reconfiguration takes between a 1K and 10K clock cycles for an 8×8 mesh with dedicated hardware. When reconfiguration is conducted in software, it takes substantially longer [2,6]. In addition, all these solutions require suspending normal network activity while the reconfiguration is ongoing: in-flight packets are stalled because new routes may conflict with old routes, possibly triggering a deadlock situation. Overall, these solutions fail to provide uninterrupted availability in presence of faults: a property that a digital system expects from its communication infrastructure.

Distributed fault-recovery solutions [5,24] operate by enabling each router to monitor the availability of its neighbors. When it finds that a neighboring resource has become unavailable, the router avoids transmitting towards the affected direction. While solving the issue of preventing packet loss, these solutions may fall into livelock, due to the short-sightedness of their detouring approach. Indeed, the router enforcing the detour may require global connectivity information to guarantee correct delivery and deadlock-freedom, even in presence of a few faults. This requirement, though, brings along storage and route computation overheads.

978-3-9815370-2-4/DATE14/©2014 EDAA

In this work, we propose a **B**risk and **L**imited-**I**mpact **N**oC routing re-**C**onfiguration (BLINC) algorithm. BLINC deploys a topology-agnostic routing algorithm, which provides maximal connectivity and deadlock-freedom. The algorithm leverages a novel representation of the network topology, which allows to quickly perform reconfiguration locally, affecting very few routers. The representation consists of routing metadata stored at each router in a distributed fashion, and updated upon each reconfiguration event through neighbor-to-neighbor updates. The metadata is used to compute alternative (*emergency*) routes, which affect only a few routers in most cases, and can be quickly deployed upon a failure. BLINC maintains a deadlock-free network connectivity at all times, if at all possible.

We use our BLINC algorithm to develop a transparent reliability solution for NoCs, based on aggressive online testing and failure prevention. In our framework, individual components are taken offline and tested to evaluate if they are close to failing, in which case they are disabled. BLINC allows us to quickly move components offline and back online, and it provides a first-response routing solution when a failure is deemed imminent. These capabilities allow our framework to operate uninterrupted and without data loss through testing and fault reconfiguration.

Contributions. In summary, we make the following contributions.

- We propose a novel, fast, deadlock-free, distributed and localized routing reconfiguration algorithm, called BLINC. Experimental results show an 80% reduction in the number of routers affected by a reconfiguration event and a 98% reduction in reconfiguration latency, compared to existing solutions [1,22].
- We present a route computation framework that minimizes performance impact. Emergency routes provide near-optimal alternative routes without computation-intensive rerouting.
- We develop a transparent fault detection and reconfiguration solution based on BLINC. Our solution enables uninterrupted network operation during aggressive component-testing and fault reconfiguration. It presents a minimal latency increase of 6%, compared to a 17× increase for a baseline approach that stalls one link segment at a time.

II. Related Work

Wachter et al. [22] recently summarized existing fault-tolerant routing techniques. Most of them can be grouped into two families based on their approach to reconfiguration. The first family deploys routing tables and logic that are updated upon each fault occurrence [1,4,15,22]. This approach is topology-agnostic and, in the best case, it can tolerate an arbitrary number of faults, but suffers from high reconfiguration overhead. [16] also falls within this family: it proposes to use route-computation logic instead of routing tables to limit silicon area overhead, but requires an additional computation step to configure the routing logic.

The second family of solutions exploits bypass rules to reroute

around faults using local connectivity information [5,24]. Due to the localized nature of these solutions, they can only sustain a few faults before the network becomes disconnected. Other solutions also exist that do not belong to either of these families. For instance, [14] leverages stochastic communication to select the forwarding direction for each packet. While it can provide high fault tolerance at low silicon cost, it also suffers from high performance impact due to the use of non-minimal paths and lack of a delivery-guarantee.

Fast routing reconfiguration has been investigated mostly for off-chip interconnection networks, such as local area networks (LANs). [2] proposes a dynamic, progressive reconfiguration procedure based on the up*/down* routing algorithm [17], which uses graph manipulation operations. However, it only discusses reconfiguration principles without investigating the details and hardware required to support the reconfiguration procedure. Similarly, [18] proposes a fast dynamic reconfiguration procedure based on partial channel lists. Although both solutions claim fast reconfiguration and uninterrupted operation, they do not provide evidence supporting their claims. In the on-chip networks domain, OSR-Lite [20] proposes a fast reconfiguration solution utilizing resources to support two routing-computation logic sets based on [16], with only one of them active at a time. Upon a fault occurrence, a central manager calculates the new replacement routes, while the old ones are still in use, then the two are swapped. While OSR-Lite is reported to be faster than hardware solutions, the dedicated central manager is a single-point of failure. [21] improves [20] with a disconnection-rescuing algorithm, but it still misses potential connections due to its limited routing capability. [7] proposes a time/space-efficient reconfigurable routing algorithm, but it does not show its applicability to fault tolerance.

In Table I, we present a comparison of relevant routing reconfiguration techniques. Our proposed solution, BLINC, is very fast and provides high tolerance against a wide range of faults.

TABLE I
COMPARISON OF EXISTING ROUTING RECONFIGURATION TECHNIQUES

method	context	computation	impact	speed	fault tolerance
BLINC (our solution)	on-chip	hardware	**local**	**very fast**	**high**
ARIADNE [1]	on-chip	hardware	global	fast	**high**
MD [5]	on-chip	hardware	**local**	**very fast**	low
Sem-Jacobsen et al. [18]	off-chip	software	**local**	slow	**high**
OSR-Lite [20,21]	on-chip	software	global	moderate	moderate

III. BLINC RECONFIGURATION

The main objective of our technique is to promptly find alternative routes for packets affected by a network topology change, due to a fault or other event. Our goal is to be fast and minimalistic in the number of routing modifications, so that the network traffic is minimally perturbed.

In distributed routing, forwarding directions are selected locally at each router. Thus, rerouting entails recomputing the routing tables for all routers in the network [1,4,15]. We observe that we can limit the recomputation effort by utilizing pre-computed routing metadata, so to quickly pinpoint the affected routes. The immediate rerouting response from BLINC is fast and deadlock-free, but not necessarily minimal. However, traffic remains uninterrupted while we generate a new minimal routing configuration in the background. To this end, we also initiate concurrently a complete routing reconfiguration in software to generate new optimal routing paths in light of the new topology. Once this process is completed, the new reconfiguration is transferred to all network nodes and it replaces the emergency routing approach.

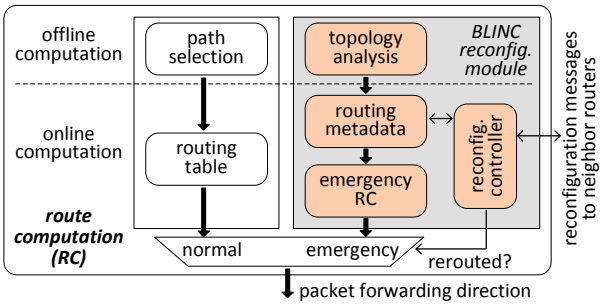


Fig. 1. **Route computation**. BLINC deploys two route computation components: routing tables (white) and reconfiguration modules (gray). On a fault occurrence, invalid routes are immediately replaced by emergency routes, while a software procedure in the background computes new optimal routes. When these are ready, they replace the emergency routes.

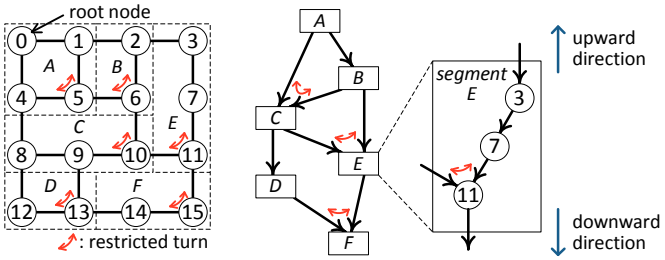
Figure 1 shows the components required for our BLINC algorithm. The white-background blocks are part of the baseline router: routing tables are generated offline [10] and ready when the network becomes operative. Upon a topology change, the gray-background reconfiguration module quickly calculates valid alternative (*emergency*) routes, so that packets affected by the change can be safely sent through them. Note that the majority of packets still utilize the original optimal routes. Thus, the deployment of our emergency routes enables fast rerouting at the expense of route optimality.

A. Alternative Route Generation Cost Analysis

Turn model: Most regular on-chip network topologies, including meshes and tori, often have many alternative routes available for each source-destination pair. For example, in a mesh, it is possible to choose between XY or YX routing, among other minimal options. With reference to Figure 2.a, node 1 can reach node 4 through node 5 or through node 0. To avoid cyclic channel dependencies [3], and thus guarantee deadlock-freedom, some of these routes should be prohibited. For instance, the turn model described in [8], disallows routes passing through a particular set of turns. A turn is defined as a connection between two links through a router. For instance, in Figure 2.a, we can prohibit the turn 1-5-4 so that no packet can traverse the path 1→5→4 or 4→5→1. Upon a fault affecting a link (or a portion of a router impacting link operability) the disabled turns must be recomputed to allow packets to go through alternative surviving routes. This effort entails a global routing reconfiguration [4], and it does not guarantee deadlock-freedom.

Up*/down* routing: Spanning tree-based routing algorithms, such as up*/down* routing [17], can be applied to route packets in any topology. The up*/down* algorithm assigns a total order to each node in the tree, from root to leaves, and it guarantees deadlock-freedom, as long as packets are not routed between two lower-order nodes via a high-order node (no down-up turn). However, upon a topology change, the up*/down* algorithm suffers from long reconfiguration latencies [1], more than 20K cycles for a 64-nodes off-chip network topology [2].

Segment-based routing: We note that it is possible to design alternative routes based on local information, if we use segment-based routing [13]. In segment-based routing, the entire network is partitioned into segments. The segmentation process starts by selecting a root node, and then identifying a segment as a sequence of nodes and links that starts and ends at the root node. Each subsequent segment is identified by building a sequence that starts and ends at nodes already included in the segmented portion of the network. Figure 2.a shows an example of segmentation. Recent work [12] has shown that it is preferable to segment a network so that each segment has exactly two links connecting it



a) sequential segmentation **b) tree of segments** **c) intra-segment tree**

Fig. 2. **Network segmentation example**. a) The segmentation process begins by identifying a segment that starts and ends at the root node (node 0). Additional segments stem from nodes already part of other segments (*e.g.*: segment B). b) The corresponding segment-to-segment tree structure reflects this construction from root to leaves. c) Nodes within each segment are organized in intra-segment trees. Note how each segment includes two connections to a parent segment.

to the already segmented portion of the network. We follow this advice in our experimental evaluation. We propose in this work to augment this algorithm and maintain metadata at each node, so that, upon a fault, it is possible to quickly modify the routing configuration locally.

B. Our Enhanced Segmentation Infrastructure

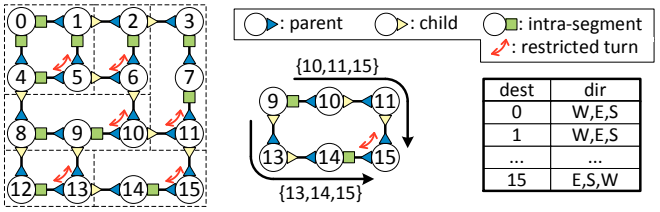
Our offline routing solution augments segment-based routing with an additional high-level tree structure, showing the connectivity between segments. The higher-level tree (Figure 2.b) is built by traversing the network segment-by-segment, following adjacency, starting from the root segment. Any two adjacent segments have a parent-child relationship if the segment under consideration (child segment) has links connecting it to one or more segments already in the tree (parent segments). For example, once segment A and B have been considered, segment C is found to be a child of both. Nodes in the tree are ordered from root to leaves based on the order in which they are included in the tree (to improve readability, Figure 2.b shows the segment order by using letters instead of numerals). Moreover, we also introduce an ordering of the nodes within each segment by building *"intra-segment trees"* (see Figure 2.c). Once all nodes are ordered, we can enforce deadlock-free routing by forbidding turns around the highest order node in each segment. Note how the forbidden turns indicated in Figure 2.a follow the approach described: for instance in segment E, the highest order node is 11, according to Figure 2.c. Thus, we disable the turn 10-11-7.

To provide an intuitive understanding of our approach, we leverage the fact that each segment is connected to segments closer to the root through two links. Thus, upon a link failure within a segment, it is possible to use the two links to reach the two disconnected portions of the segment. The tree structure remains unchanged, and it is sufficient to find a different route through one or more segments. As an example, in Figure 2.a, assume that packets going from node 4 to node 13 traverse nodes 8 and 9. If the link 8-9 fails, it is possible to find an alternate route via 5, 6, 10 and 9. Note that the alternate route may be non-minimal. However, as shown in Section IV, routes generated using our algorithm are only slightly longer than minimal.

C. Routing Metadata

To quickly find emergency routes upon a failure, the BLINC reconfiguration module leverages routing metadata that we compute while segmenting the network. Routing metadata is embedded at each router, and it includes three types of information:

• **Port type**: A router port can be connecting the router to a lower-order node (parent port), or to a higher-order node in the same segment (intra-segment port), or to a higher-order node in a different segment (child port). Figure 3.a indicates the type of all ports for our example network.



a) port type **b) children sets** (node 9) **c) preference list** (node 9)

Fig. 3. **Routing metadata**. a) Port types are assigned based on our hierarchical segmentation process. b) Each port in a node has an associated children set through child and intra-segment ports. c) Preference direction lists associated with each node are optional.

• **Children set**: the set of reachable nodes along downward routes for each port. Figure 3.b shows the children sets for node 9.

• **Preference list** (optional): An ordered list indicating the preferred output directions. If available, this list is used to improve the quality of the emergency routes generated. The list can be user-provided or generated automatically. In our evaluation we prioritize based on distance to destination. Figure 3.c shows an example of preference list.

Note that, if an output port includes the destination node in its children set, the node has at least one valid route to destination, since downward traversal is always allowed. To store the routing metadata at each router we need: 2 bits to encode the port type, a bit array to indicate the children set for each port, and 6 bits per destination to encode the preference list (at most 3 directions, 2 bits to encode each direction). Thus, for an 8x8 mesh, we need at least 264 bits per router (384 additional bits if the preference list is provided).

D. Reconfiguration Process

Upon a link failure, BLINC leverages the metadata described above to quickly generate alternative routes for the affected packets. Figure 4 illustrates the process with a high-level schematic: each segment can be represented as a chain of nodes, and thus the segment affected by the failure will find itself partitioned. The localized reconfiguration process will re-establish connectivity for all nodes by exploiting the additional routing paths that had earlier been disabled to avoid deadlock. Indeed, because of the construction described in Section III-B, each segment contains exactly one disabled turn which, at this point, will be re-enabled. Then all the children sets within the segment must be updated, so that each node is reachable from the segment boundary. This goal entails adding children to some ports' children sets and removing children from other ports. In the example of Figure 4, Y was originally reachable via X only, but after the failure, it becomes reachable via Z instead. Finally, the additions and subtractions to the children sets are propagated outside the segment, until a common ancestor is reached. The reason to keep the children sets updated is that, during a topology change, packets whose routing is affected by the change will use the children sets to determine their new paths. This decision guarantees that even packets undergoing a detour will reach their destination in the minimum number of hops possible in the new topology. Moreover, the elements added or removed from the children sets are maintained in a separate bit array so that it is possible for a packet to determine when it is undertaking an "emergency route". Note that this emergency route is also deadlock-free because the rest of the network maintains the same turn restrictions as before. Note also that a new optimal routing configuration is being generated in the background in software; once computed, it overwrites all emergency routes.

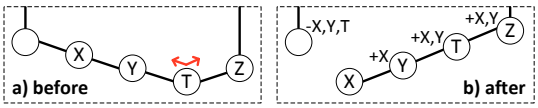


Fig. 4. **Children set update on reconfiguration**.

Algorithm 1 provides an outline of the reconfiguration algorithm. We describe each step in detail below and illustrate the process with an example in Figure 5. If the faulty link is adjacent to the node with the disabled turn within the segment, then only steps 1 and 5 of the algorithm must be executed. When that is not the case, all steps of the algorithm must be completed.

Algorithm 1 BLINC reconfiguration procedure

```
1: (parent_node, child_node) = disabled_link
2: if(Not HasTurnRestriction(child_node))
3:     turn_node = FindTurnRestrictedNode(child_node)
4:     ReversePortType(from child_node to turn_node)
5:     added_set = GetNewChildren(turn_node)
6:     curr_node = Parent(turn_node)
7:     while(Not IsEmpty(added_set))
8:         AddChildrenSet(curr_node, added_set)
9:         Update(added_set), curr_node = Parent(curr_node)
10:    WaitForAck(child_node)
11: removed_set = GetUnreachableChildren(parent_node)
12: curr_node = Parent(parent_node)
13: while(Not IsEmpty(removed_set))
14:     RemoveChildrenSet(curr_node, removed_set)
15:     Update(removed_set), curr_node = Parent(curr_node)
```

Step 1. Disabling the link: Nodes adjacent to the faulty link stop sending packets through it (line 1 in the pseudo-code).
Step 2. Re-enabling the turn: Before the fault, the node with the disabled turn (T in Figure 4) was the leaf in the intra-segment tree. After the fault, the portion of the segment between the turn-disabled node and the faulty link becomes isolated (portion between X and T in Figure 4). To reconnect it, we need to i) re-enable the turn at T, ii) swap the port types for each link in the isolated portion (Figure 5.2), and then iii) create an *"added-children set"* for each port in the isolated portion, which includes all the nodes downstream towards X (lines 3-5 in the pseudo-code). The added-children set is used to detect when a packet must detour because of the fault: if the destination is in the original children set, the packet does not experience a detour, otherwise a detour is necessary.
Step 3. Enabling alternative routes: Once the added-children set for the turn-restricted node (T in Figure 4) is computed, the set is propagated toward the root node, across segment boundaries, to instruct every node of the new route to reach the destinations next to the faulty link. For each node towards the root, the current children set of that node is compared against the incoming added-children set, then the latter is reduced to include only nodes not already present in the children sets of the node under consideration. Indeed, if a destination was already in the children set of a node, no routing change should be applied at that node. The process stops when the added-children set becomes empty. In Figure 5.3, the added-children set is propagated from node 10 all the way to the root node and node 4 (lines 6-9).
Step 4. Waiting for ack: The last recipients of the added-children set generate an acknowledgment message that is propagated all the way back to the node adjacent to the link failure (line 10).
Step 5. Disabling invalid routes: Finally, the other portion of the segment (the one connected to the parent port side of the faulty link) generates a *"removed-children set"* to indicate that it cannot reach the other end of the link. The removed-children set is propagated towards the root in a similar fashion to the added-children set: the only difference is that nodes are eliminated from

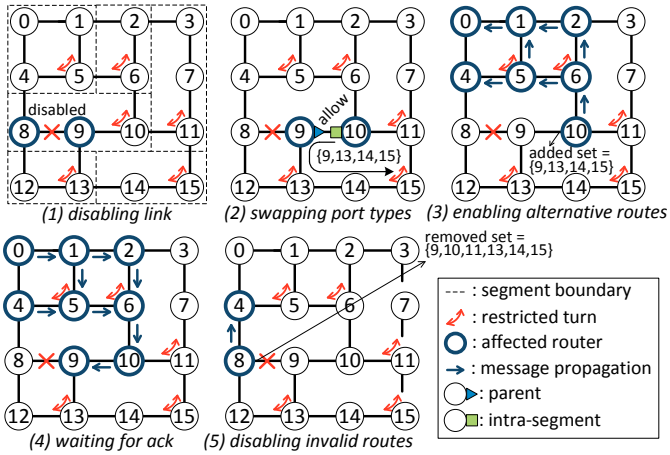


Fig. 5. **Reconfiguration example**. The turn restriction in the segment with a fault is eliminated and the new intra-segment leaf is the node next to the faulty link (node 9). An added-children set is created to indicate the new routes to reach the downward nodes after the fault, and it is propagated towards the root node. Once this step completes, acknowledgment messages are propagated back down. A similar process is followed to propagate removed-children sets starting from the other end of the fault (node 8).

the removed-children set when they already exist in the union of the children sets of all router's ports other than the port receiving the remove-children message (lines 11-15). Each router tracks its children's modification through a bit vector.

Messages carrying the port swapping and the added- or removed-children sets can be transmitted on the same network using a dedicated virtual channel. Note that this minimally impacts the router's frequency because it only adds a 2-input multiplexer in front of the crossbar. In addition, packets in reconfiguring routers are stalled until the reconfiguration process is completed. Then packets are routed as before, simply following the turn restriction rules. However, when a packet is forwarded through a port that includes that packet's destination in its added- or removed-children set, the packet enters "emergency routing", and a corresponding flag is set in its header flit. Once a packet is in emergency routing, it routes using a minimal-hop tree-routing algorithm by always selecting the port containing the destination in its children set. When multiple options are available, the preference list is used. When no choices are available, the packet is routed towards the upward direction, in the worst case, to the root node.

E. Discussion

BLINC reconfiguration is capable of tolerating a single link failure per segment. Once a segment is damaged by a fault, the next fault in the same segment may not be recovered by BLINC. We deploy a background rerouting algorithm to handle this. The background rerouting computes an optimal routing function for the surviving network topology. Subsequently, the new metadata can be overwritten safely when the network is idle. Although this rerouting process usually takes longer than our fast reconfiguration [6], it is still much shorter than the plausible time to the next failure. Indeed, BLINC supplements existing solutions by providing a fast emergency response against the fault.

BLINC's metadata adds to the router's area footprint, so it can be affected by faults. However, since the metadata consists mostly of storage, it can be easily protected by error-correcting codes.

IV. Experimental Evaluation

We evaluated our BLINC algorithm with a cycle-accurate NoC simulator, Booksim [3]. Our baseline design uses wormhole, 3-stage pipelined routers with buffers for eight 64-bit flits per input

port, connected in a mesh topology. Packets are 10 flits long, injected using random traffic at a 0.05 flits/cycle/router rate. We first evaluated the performance of our solution, and then considered its value in the context of a fault detection/reconfiguration solution for uninterrupted availability.

Fault Model. We assume that the faults' spatial distribution is uniform over the component's area [1]. Thus, we derive area values for each major module (input buffers, crossbar, route computation unit, *etc.*) of the router in [3], using Synopsys DC and the Nangate 45nm target library. The faults' impact noted in this synthesized logic model is then mapped to the link-failure model of the network: i) faults in the router's control logic (9.4%) disable the entire router, affecting all surrounding links; ii) faults in the router datapath (90.6%) only disable one link, including the two I/O ports connected at its ends. Our evaluation considers 4 baseline systems, with the following fault's nominal rates: 0%, 1%, 5% and 10%. The rate corresponds to injecting a number of faults equal to the fraction of links in the topology (an 8x8 mesh has 112 links, thus the 25% rate would be 28 faults). Note that some faults may affect several links.

To evaluate BLINC, we create 10 distinct faulty topologies, using different random seeds for each fault rate, and then generate one more failure at 10 random different sites. In total, each fault rate is evaluated with 100 distinct fault situations (10 baseline topologies $\times$ 10 failure locations).

A. Characterization

Number of affected routers. The left part of Table II reports the average number of routers affected by a reconfiguration event over a range of fault densities and network sizes. The affected number of routers increases slowly with network size, showing that BLINC localizes the fault manifestation to a small region. Compared to existing methods [1,20], BLINC achieves more than 80% reduction in the number of affected routers, across a wide range of fault densities.

Reconfiguration latency. Reconfiguration latency was computed assuming that each node takes 5 cycles to process an add/remove children-set message and 1 cycle to propagate the acknowledgement messages. Our findings are reported on the right part of Table II. While reconfiguration latency is minimally sensitive to network size, it does show a steady increase with growing fault density. We believe this is due to the naturally occurring longer segments in faulty networks, which in turn, impose more hops in the transmission of reconfiguration messages. Overall, BLINC's reconfiguration latency is 98% shorter than previous hardware-based techniques [1]. Note that [20,22] have the disadvantage of performing reconfiguration in software, thus competing with applications for CPU-time.

TABLE II
AFFECTED ROUTERS AND RECONFIGURATION LATENCY

method	initial faults	impact of next fault: # affected routers 6×6	8×8	10×10	reconf. latency (cycles) 6×6	8×8	10×10
BLINC	0%	7.0	9.0	9.9	21.0	26.0	30.1
	1%	6.8	9.3	10.3	21.0	28.1	31.1
	5%	7.1	9.1	10.0	23.9	28.7	30.6
	10%	6.6	8.9	10.0	24.9	30.0	34.4
ARIADNE [1]	-	all routers			1.3K	4.1K	10K
OSR-Lite [20]	-	all routers			-	~569*	-
Wachter et al. [22]	-	all routers			-	-	0.2K-208K

Quality of emergency routes. Figure 6 shows the hop count increase and the utilization of emergency routes when the number of disabled links over the baseline topology varies from 1 to 10. The average hop count increases as the number of disabled links increases, up to 11.4%, compared to the performance of the baseline topology. In addition, the plot indicates only a 3.0% increase in hop count compared to optimal routes. Further, emergency routing is applied to 7.7% of the packets at 1 fault, and up to 48.7% of the packets at 10 faults. These results suggest that emergency routing provides near-optimal routes.

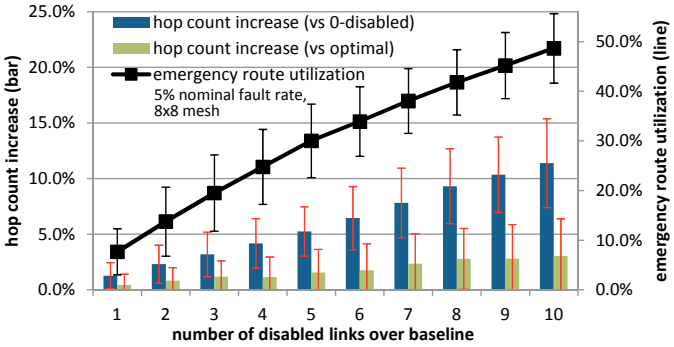


Fig. 6. **Effect of multiple disabled links**. The network performance degrades gracefully as additional links are disabled. The emergency routes shows only a slightly worse hop count than optimal ones.

B. Uninterrupted Availability with BLINC

To showcase the value of BLINC's approach, we evaluated its deployment in a fault detection and reconfiguration solution that provides uninterrupted availability. The methodology tests network resources aggressively to detect early signs of an upcoming fault. Each link, in turn, is taken offline for testing, which is performed through transmission of testing packets generated by a test pattern generator, such as the one in [9]. This approach can detect a majority of router faults [4]. For this methodology to be valuable, i) the network should be available and connected while a link is being tested and ii) the testing approach should be capable of detecting early signs of link failure (*e.g.*: increased delay, *etc.*), so that the network can reconfigure around the upcoming fault with no loss of packets. The first requirement can be accommodated by our BLINC algorithm: as shown in the previous section, it can provide emergency routes with minimal overhead. The latter requirement has been solved in the context of microprocessor designs [19,23] but, to date, no solution of this kind has been developed for NoCs. Because of BLINC's fast and localized reconfiguration, it is possible to select each link in turn, take it offline, test it in a harsh operating environment to mimic circuit aging (such as lowered supply voltage [19]), and then bring it back online. BLINC can simply reconfigure the NoC to avoid the target link before the testing phase, and then reactivate the original routing after the test completes. If a link is found at risk of experiencing failure, emergency routing is maintained until a new segment-based routing solution can be computed in the background.

The testing flow is characterized by two parameters: the length of the test duration for each link (L) and the network testing rate (f), as illustrated in Figure 7. One complete testing cycle entails testing each link in turn. Note that the network should remain completely available even throughout testing.

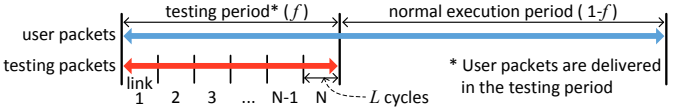


Fig. 7. **Online testing flow**. The network should remain completely available even during testing, so that an aggressive testing frequency does not degrade network performance.

Figure 8 reports our findings: it plots average packet latency

under a range of testing rates and test durations. Viable testing rates and durations were derived from [9,11,23]. For instance, the rightmost data point in our plot corresponds to one complete test period every 112,000 cycles. We compare our measurements against a routing solution (called *Stall*) that does not benefit from BLINC. In Stall, packets are simply stalled in their buffers whenever they are trying to use a link undergoing testing. We only compared against Stall, because other solutions [1,4,15,22] have reconfiguration latencies longer than the test durations we are evaluating. From Figure 8, we observe that with BLINC, average packet latency is minimally affected (6% in the worst case) by the ongoing testing and reconfiguration process, regardless of test durations. Moreover, Stall cannot provide uninterrupted availability beyond a test duration of 500 cycles, even at very low testing rates (the latency increase is 17× for L=1,000, f=1%).

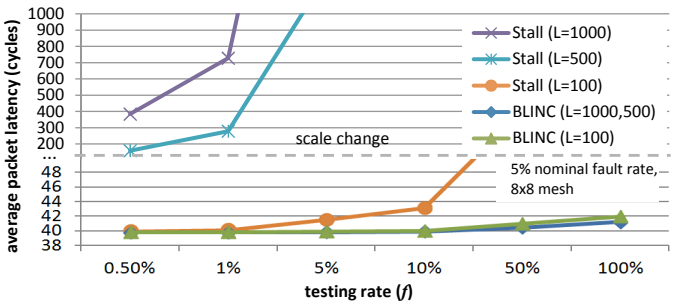


Fig. 8. **Average packet latency under online testing**. BLINC reconfigures routing quickly, supporing online testing with only negligible performance degradation.

Figure 9 shows the accepted flit rate during testing. We evaluated this under a randomly chosen 8×8 mesh topology and 1,000 random traffic patterns with $L = 500$. BLINC provides a steady packet delivery capability. The accepted flit rate shows only little fluctuations as shown in the figure. On the contrary, Stall shows a decreased delivery capability with a periodic increase when the link under test changes. We also conducted the same experiment for BLINC with a doubled injection rate. The result shows a similar result as the one in the figure, except for a slightly lowered accepted rate at the fifth link period (12,000-12,500 cycles), shortly recovered in the next period.

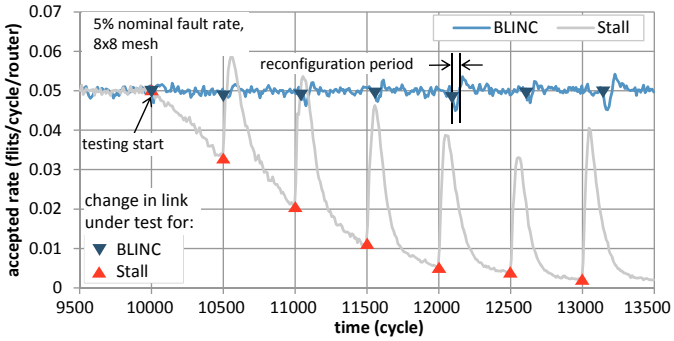


Fig. 9. **Accepted flit rate during testing**. BLINC shows a steady rate with little fluctuations during reconfiguration (near 12,100 cycle and 13,150 cycle). Stall shows a constantly decreasing rate with periodic peaks.

V. Conclusions

We proposed BLINC, a brisk and local-impact NoC routing reconfiguration algorithm. BLINC utilizes a combination of online route computation procedures for immediate response, paired with an optimal offline solution for long term routing. To achieve its goal, BLINC employs compact and easy-to-manipulate routing metadata. Our evaluation shows more than 80% reduction in the number of routers affected by reconfiguration, and 98% reduction in reconfiguration latency, compared to state-of-the-art solutions. We also discussed how BLINC enables uninterrupted availability for networks-on-chip, by allowing individual network links to be taken offline for testing at high frequency. BLINC maintains stable network performance with only a 6% increase in latency during testing, in contrast with a 17-fold latency increase for a baseline approach that stalls one link segment at a time.

Acknowledgements. This work was supported by STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA, and NSF grant #0746425.

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] R. Casado, A. Bermudez, F. Quiles, J. Sanchez, and J. Duato, "Performance evaluation of dynamic reconfiguration in high-speed local area networks," in *Proc. HPCA*, 2000.
[3] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2003.
[4] A. DeOrio et al., "A reliable routing architecture and algorithm for NoCs," *IEEE Trans. CAD*, vol. 31, no. 5, 2012.
[5] M. Ebrahimi, M. Daneshtalab, J. Plosila, and F. Mehdipour, "MD: minimal path-based fault-tolerant routing in on-chip networks," in *Proc. ASPDAC*, 2013.
[6] J. Flich et al., "A survey and evaluation of topology-agnostic deterministic routing algorithms," *IEEE Trans. PDS*, vol. 23, no. 3, 2012.
[7] B. Fu, Y. Han, J. Ma, H. Li, and X. Li, "An abacus turn model for time/space-efficient reconfigurable routing," in *Proc. ISCA*, 2011.
[8] C. Glass and L. Ni, "The turn model for adaptive routing," in *Proc. ISCA*, 1992.
[9] C. Grecu, A. Ivanov, R. Saleh, and P. Pande, "Testing network-on-chip communication fabrics," *IEEE Trans. CAD*, vol. 26, no. 12, 2007.
[10] M. Koibuchi, A. Jouraku, and H. Amano, "The impact of path selection algorithm of adaptive routing for implementing deterministic routing," in *Proc. PDPTA*, 2002.
[11] T. Lehtonen, D. Wolpert, P. Liljeberg, J. Plosila, and P. Ampadu, "Self-adaptive system for addressing permanent errors in on-chip interconnects," in *IEEE Trans. VLSI Systems*, 2010.
[12] A. Mejia, J. Flich, and J. Duato, "On the potentials of segment-based routing for NoCs," in *Proc. ICPP*, 2008.
[13] A. Mejia, J. Flich, J. Duato, S.-A. Reinemo, and T. Skeie, "Segment-based routing: an efficient fault-tolerant routing algorithm for meshes and tori," in *Proc. IPDPS*, 2006.
[14] M. Pirretti et al., "Fault tolerant algorithms for network-on-chip interconnect," in *Proc. ISVLSI*, 2004.
[15] V. Puente, J. Gregorio, F. Vallejo, and R. Beivide, "Immunet: a cheap and robust fault-tolerant packet routing mechanism," in *Proc. ISCA*, 2004.
[16] S. Rodrigo et al., "Addressing manufacturing challenges with cost-efficient fault tolerant routing," in *Proc. NOCS*, 2010.
[17] M. Schroeder et al., "Autonet: a high-speed, self-configuring local area network using point-to-point links," *IEEE Journal of Selected Areas in Communications*, vol. 9, no. 8, 1991.
[18] F. Sem-Jacobsen and O. Lysne, "Topology agnostic dynamic quick reconfiguration for large-scale interconnection networks," in *Proc. CCGrid*, 2012.
[19] J. Smolens, B. Gold, J. Hoe, B. Falsafi, and K. Mai, "Detecting emerging wearout faults," in *Proc. SELSE*, 2007.
[20] A. Strano et al., "OSR-Lite: fast and deadlock-free NoC reconfiguration framework," in *Proc. SAMOS*, 2012.
[21] F. Trivino, D. Bertozzi, and J. Flich, "A fast algorithm for runtime reconfiguration to maximize the lifetime of nanoscale NoCs," in *Proc. INA-OCMC*, 2013.
[22] E. Wachter, A. Erichsen, A. Amory, and F. Moraes, "Topology-agnostic fault-tolerant NoC routing method," in *Proc. DATE*, 2013.
[23] B. Zandian et al., "WearMon: reliability monitoring using adaptive critical path testing," in *Proc. DSN*, 2010.
[24] Z. Zhang, A. Greiner, and S. Taktak, "A reconfigurable routing algorithm for a fault-tolerant 2D-mesh network-on-chip," in *Proc. DAC*, 2008.