

LinkMiser: Resource Conscious Routing and Reconfiguration in Faulty On-Chip Networks

Ritesh Parikh, Valeria Bertacco

The Department of Computer Science and Engineering, University of Michigan
{parikh,valeria}@umich.edu

ABSTRACT

As silicon technology continues to scale, transistor reliability is becoming a major concern. At the same time, increasing transistor counts are causing a rapid shift towards large chip multi-processors (CMP) designs, comprising several processors cores connected via a network-on-chip (NoC). As the sole medium of communication, a NoC should be able to gracefully tolerate an increasing number of permanent faults. In particular, it should be able to fully utilize all the functional resources to maintain decent performance.

We propose LinkMiser, a solution for reliable operation of NoCs incurring high fault rates. In contrast to previous approaches that only use a subset of the functional links, LinkMiser utilizes all the working components in the NoC to provide reliable, deadlock-free routes between nodes. To this end, it leverages fine-resolution fault diagnosis mechanisms [18], designed specifically for on-chip networks. We also propose a modified version of up*/down* routing, specifically designed for maximal utilization of the functional components. Our solution places no restriction on the number and location of faults, while its distributed nature enables an efficient implementation. Experimental results show that LinkMiser, implemented in a 64-node NoC, provides 4.4% greater connectivity and 6.7% latency improvement at 50 faults over state-of-the-art on-chip fault tolerant solutions. Moreover, the reliability improvement is even more impressive at higher fault rates.

1. INTRODUCTION

To benefit from the continued technology scaling in microprocessors, recent trend has been towards a growing number of simpler, power-efficient cores-on-chip. As a result, the corresponding increase in inter-core communication demands has rapidly sidelined traditional interconnects, such as simple buses, due to their limited bandwidth and poor scalability. NoCs, characterized by highly concurrent communication and thus better scalability, are becoming the de-facto choice for on-chip interconnects. Moreover, as the sole medium for on-chip communication, these NoCs are single points of failure, so that any permanent fault in the NoC can cause the entire system to fail. The waning reliability of silicon manufacturing at the most recent technology nodes makes the problem even worse. It is paramount to any reliability solution designed for highly unstable silicon, to fully utilize all the resources that are still functional after faults manifest.

Existing NoC reliability techniques can be broadly divided into protection against faults in router logic [5, 9] and re-routing around faulty links [19, 14, 8, 1]. The latter can also be leveraged for router logic faults using a simple fault model where a fault in a router's logic is modeled as a fault in each of the links connected to that router. Fortunately, recent on-chip diagnosis mechanisms, such as [21, 18], allow us to diagnose faulty components at a finer resolution, pointing out the faulty link or turn (the connection between two links through a router). However, current re-routing based reliability solutions do not exploit this additional opportunity of finer reconfiguration as they are distributed implementations [1, 8] of off-chip solutions [20, 10], which were in turn originally designed to route in irregular topologies. Off-chip solutions (and hence their on-chip derivatives) assumed links to be bidirectional, and a fault in one direction was assumed fatal for the entire link. This assumption

worked well for off-chip networks where faulty components could be replaced with new ones. However, for on-chip networks with fixed amount of resources, this overapproximated model disregards precious functional resources, leading to an opportunity loss in performance and connectivity, as we demonstrate in Section 3.

Moreover, unlike most approaches for NoC reliability, LinkMiser is not limited by the number or location of failures. This is important when targeting multi-cores at future technology nodes [3], where a large number of transistors may fail in an unpredictable manner. For NoCs, this might lead to deadlock-prone or even disconnected topologies, forcing surviving nodes to coordinate the discovery of the existing topology and deadlock-free routes.

1.1 Contributions

Increasingly unreliable silicon at advanced technology nodes requires techniques that can provide high reliability, with graceful degradation in performance. In addition, area overhead should be kept at a minimum. LinkMiser uses a novel routing algorithm that extends the theory of up*/down* routing to networks with unidirectional links between routers. This new unidirectional up*/down* routing (named uni-up*/down*) is uniquely designed to maximally utilize surviving links in faulty on-chip networks. Uni-up*/down* is also guaranteed to discover only deadlock-free routes without the use of any virtual channels, and provably performs better than traditional up*/down* in all possible fault scenarios. Moreover, a lightweight reconfiguration algorithm implementing uni-up*/down* provides maximum robustness, as it places no restriction on the location of faults. Given only local information, the reconfiguration algorithm executes in lockstep at each network router to route around faulty components. As transistors fail over time, LinkMiser disables components in a frugal fashion, so as to maintain the maximum working set of links leading to three major benefits:

Path Diversity: As faults accumulate over time, a greater number of links are available to route packets between nodes, creating path diversity, and hence better performance.

Node Connectivity: LinkMiser is able to connect nodes using dead-lock-free routes, including nodes that were unreachable using traditional up*/down* with bidirectional links.

Load Balance: In previous solutions, bidirectional links were completely disabled even when only one direction had become faulty. LinkMiser disables links sparingly, only in the affected directions, thus utilizing a greater number of links for node-to-node communication and reducing the stress on the healthy links.

The rest of this paper is organized as follows: Section 2 presents the related work, and Section 3 discusses the need for a reliable routing algorithm for networks with unidirectional links and the opportunities for better reliability and performance that it offers. The uni-up*/down* routing algorithm is detailed in Section 4, while the distributed reconfiguration scheme to implement it is described in Section 5. Finally, Section 6 discusses the experimental setup and results, while Section 7 concludes the paper.

2. RELATED WORK

Ensuring reliability in NoCs has been the subject of much previous research, focusing on a variety of aspects. These methods enhance NoC reliability by enabling one or a combination of the

following features: i) detection of erroneous behavior [7], ii) diagnosis of fault site [18, 9], ii) recovery from erroneous state [17], or iii) system reconfiguration to bypass the permanent faults [19, 14, 9, 1]. In this work, we specifically look at reconfiguration.

During reconfiguration, a new set of routing paths is generated (whenever a new link fault is detected) to replace the current ones. Complex, performance-oriented reconfiguration algorithms often translate to high area overhead. If implemented in software, like in most off-chip solutions [20, 16, 13], they require additional hardware to collect the surviving topology information at a central node and to relay the new routing tables back to the respective nodes once the reconfiguration algorithm completes. Similarly, if implemented in hardware, the complex optimizations would lead to prohibitive area overhead. The result is that most low-overhead hardware-only reconfiguration mechanisms can only tolerate a small number of faults, usually with constraints on fault locations. The situation is even worse for on-chip networks that have tight area and power budgets. In addition, high fault rates in upcoming silicon technology, along with the unpredictable nature of fault occurrences, result in solutions designed for a bounded number [11, 12] or constrained pattern of faults locations [22] being unfit for NoCs.

Researchers have recently proposed a few hardware-only reconfiguration solutions [19, 9, 1] that tackle the problem of unconstrained faults to a certain extent. However, they either impose significant area overhead, and/or fail to adequately address all failure scenarios. For example, Immunit [19] incurs large area overhead and Vicis [9] deadlocks on fault occurrences.

Ariadne [1] proposes a low-overhead distributed implementation of up*/down* [20] routing for NoCs. Ariadne has all desirable features of up*/down* routing, being able to provide deadlock-free routes between connected nodes, regardless of the number and location of faults. However, Ariadne, as a direct implementation of off-chip solutions, does not exploit finer reconfiguration opportunities offered by the unique nature of faults in NoCs. Specifically, Ariadne can only work on bidirectional links and it disables healthy unidirectional links if the link in the opposite direction is faulty. Similarly, Ariadne has no support for disabling faulty turns within a router, and must disable both links that form a turn to continue error-free operation.

In this work, we propose LinkMiser, based on a novel routing mechanism, uni-up*/down*, that extends the theory of up*/down* routing to use in networks with unidirectionally faulty links and turns. Moreover, LinkMiser incorporates a low-overhead distributed reconfiguration algorithm, implementing this novel routing algorithm. LinkMiser offers the same unlimited robustness of Ariadne, while outperforming all previous approaches, as it does not disable healthy links to be able to use off-chip reconfiguration solutions.

3. BACKGROUND AND MOTIVATION

In this section, we first discuss unique characteristics of fault manifestation in NoCs and opportunities they present for improved reliability and performance. Routers in an NoC are connected by bidirectional links, where each unidirectional link is implemented using separate sets of parallel wires. Most previous reconfiguration approaches assume a very simple fault model, where a broken link wire in one direction is represented as a failure in the whole bidirectional link. All faults within the router logic are modeled by tagging all communication links connected to it as faulty. Vicis [9] first improved on this simplistic fault model by mapping gate-level faults within a router logic to link-level failures. This coarse grain fault model (denoted *CG_FModel*) used by Vicis (and Ariadne [1]), however, modeled each link failure as a bidirectional link failure, so to leverage off-chip irregular routing algorithms [4, 20] for recon-

figuration in faulty on-chip networks without significant changes. Up*/down* routing [20] is a classic example of a routing algorithm that was originally designed for off-chip irregular networks and, because of its simplicity, it was adopted by Ariadne [1], an on-chip fault tolerant solution. Up*/down* routing is a deadlock-free algorithm that assigns directions to all surviving links in the network: *up* or *down*. First, a spanning tree is constructed after choosing a root node, as shown in Figure 1a. Starting from the root, the rest of the nodes in the network are arranged on a single spanning tree. Considering that each link is made of two opposite unidirectional links, the *up* link is defined as: i) the unidirectional link towards a node that is closer to the root in the spanning tree; ii) or the unidirectional link towards the node that has the lower identifier, if nodes at either end of the link are at the same tree level. Cyclic dependencies are broken by disallowing routes that include traversing a *down* link followed by a *up* link (Figure 1b). However, due to the overapproximation of faults in the *CG_FModel*, up*/down* is unable to fully utilize all functional resources available.

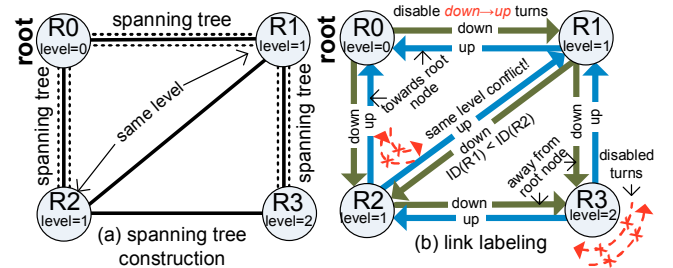


Figure 1: **Up*/down* routing.** Links are marked either *up* or *down*. Links are marked *up* if they point towards the node that is closer to the root, while in case of nodes being at the same level, the link towards a lower ID node is marked as *up*. Finally, all *down*→*up* turns are disabled.

Authors of [18] further refined the diagnosis problem for on-chip networks by mapping all link or router (datapath and control) level faults to *OP/IP* (output port/input port) *links* or *turns*. [18] defines an *OP/IP link* as the combination of a link between routers, the corresponding output port in the upstream router, and the input port/FIFO in the downstream router. As packets that enter an *OP/IP link* always traverse all its components, all faults within any *OP/IP link* can be modeled as a failure in the corresponding unidirectional link. This fine grain fault model is denoted as the *FG_FModel*. Other faults (such as within the crossbar) can be modeled as malfunctioning *turns*. Note that, instead of a failed connection between links, a malfunctioning *turn* is modeled as two bidirectional link failures in the *CG_FModel*. This motivates our research to find an efficient algorithm that could incorporate this new fault model, *i.e.*, find opportunities for better reliability and performance using working unidirectional links/turns. Figure 2 graphically represents the two fault models, *i.e.*, *CG_FModel* and *FG_FModel*. Fault locations shown in Figure 2a are modeled differently in the *CG_FModel* (2b) and the *FG_FModel* (2c).

Improved Performance: The *FG_FModel* can improve path diversity over the *CG_FModel*, as working unidirectional links that were disabled in the *CG_FModel* can be used to transmit packets. As shown in Figure 3a, the unidirectional link $R1 \rightarrow R0$ is faulty. With the *CG_FModel*, only a single path will remain from $R0$ to $R1$, via link $R0 \rightarrow R2$ and then link $R2 \rightarrow R1$. However, using the *FG_FModel*, an additional deadlock-free route can be utilized from $R0$ to $R1$, via the unidirectional link $R0 \rightarrow R1$.

Improved Reliability: The *FG_FModel* can provide better load balance than the *CG_FModel*, as the number of working links shrinks faster in the *CG_FModel* than in the *FG_FModel*, as can be noted

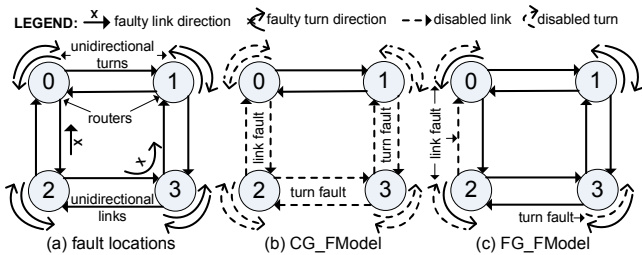


Figure 2: **Fault Modeling.** (a) Example fault manifestation. (b) CG_FModel : fault in one direction blocks communication in both directions via the faulty link. Faulty turns are modeled as two bidirectionally faulty links. (c) FG_FModel : fault in one direction only blocks communication in the faulty direction. Faulty turns are modeled by disallowing packet routes from input to output link of that turn.

in Figure 3a. Moreover, the FG_FModel can enable routes to and from a node that was considered disconnected in the CG_FModel , thus providing better connectivity. In case of Figure 3b, $R1$ is disconnected from the rest of the network if the CG_FModel is used. However, the FG_FModel can provide deadlock-free routes between any pair of nodes by simply disabling the turn $R0 \rightarrow R1 \rightarrow R2$.

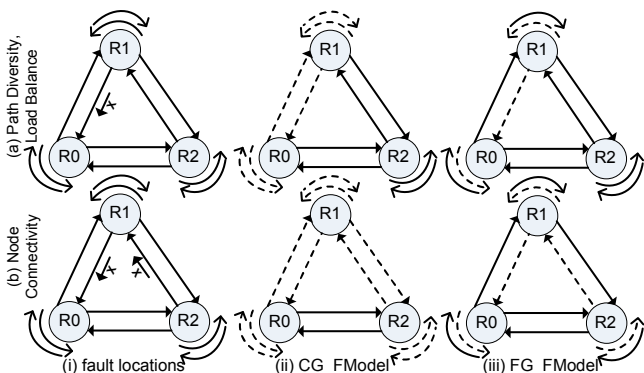


Figure 3: Additional working links in the FG_FModel can provide: (a) improved path diversity and load balance, (b) additional node connectivity.

4. RESILIENT ROUTING ALGORITHM

As mentioned earlier, all previous versions of up*/down* routing assume only bidirectional links (CG_FModel). This constraint, although it forces reconfiguration algorithms like Ariadne to disable working unidirectional links, it enables a low-overhead implementation with a desirable property: if a path between two nodes exists, the algorithm will enable at least one deadlock-free route between them, irrespective of the number and location of faults. It is easy to visualize that if paths exist between all nodes in an undirected graph, then a tree spanning all nodes without forming cycles always exists. Following paths in this tree can guarantee deadlock-free connectivity among all nodes. However, finding deadlock-free routes between any pair of nodes, even if the network is connected, is not always possible in a directed network. Therefore, we design the new uni-up*/down* routing mechanism in such a manner that enabled routes are always deadlock-free and at the same time the algorithm tries to maximize connectivity. In other words, uni-up*/down* never enables networks where deadlocks are possible, sacrificing a few connected nodes if connecting them would lead to possibility of deadlock. Additionally, uni-up*/down* provably performs better than traditional up*/down* in all possible fault scenarios, that is, uni-up*/down* provides more path diversity, better connectivity and load balance in any fault configuration. It also allows for unlimited robustness as it places no restriction on the location of faults.

4.1 Separate Up and Down Trees

Traditional up*/down* routing is considerably restricted, as a single spanning tree is expanded only via links that are bidirectionally working. No additional connectivity among nodes can be achieved using this version of up*/down* routing on networks using the FG_FModel . Referring to the example of Figure 3b(iii), even though deadlock-free connectivity is possible, a traditional up*/down* spanning tree only consists of $R0$ and $R2$. The network has to unnecessarily sacrifice $R1$, severely affecting the available computational resources. In addition to connectivity, current up*/down* approaches have no provision for assigning directions (up or down) to unidirectional links, and hence, cannot be used to provide better path diversity or load balance.

To this end, we present uni-up*/down* routing, specifically designed to provide maximum connectivity with deadlock freedom in networks with unidirectional links. To maximally utilize all the unidirectional links, we construct separate spanning trees for connections away from the root node (down-tree) and for connections towards the root node (up-tree). In other words, the down-tree marks all the down links and the up-tree marks all the up links. Note that both spanning trees are made of strictly unidirectional links. Links are marked in accordance to the up*/down* principle (Section 3) and in a consistent manner, meaning no unidirectional link is marked as both up and down. Due to consistent marking of links in accordance to the up*/down* principle, all dependency cycles are broken by disabling down-up turns and hence the new routing function is guaranteed to be deadlock-free. Additionally, if both down-tree and up-tree are able to reach all nodes, then the network will be connected in a deadlock-free fashion, as all nodes can reach the root following the links in up-tree and the root can forward the messages to respective destinations following the links in down-tree, all without taking a down-up turn.

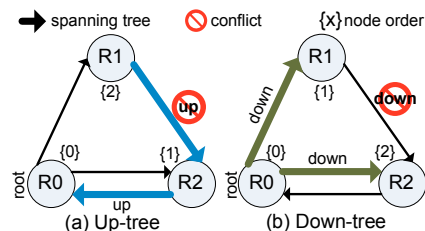


Figure 4: Independent breadth-first construction of up and down spanning trees with conflict in the labelling of link $R1 \rightarrow R2$. (a) Up-tree: link $R1 \rightarrow R2$ is directed towards node that is closer to the root; hence marked up. (b) Down-tree: link $R1 \rightarrow R2$ is between nodes at the same level and it is directed towards a node with larger identifier; hence marked down.

Down-tree and up-tree are not always required to have the same shape in order to have a consistent link marking, as in this case our novel uni-up*/down* would degenerate to traditional up*/down*. The two spanning trees are considered to be of the same shape when unidirectional links in both trees have one-to-one correspondence, i.e., for each unidirectional link in the down-tree the opposite direction link of the same bidirectional link is a part of the up-tree. On the other hand, the two spanning trees cannot be extended in an oblivious fashion, without knowledge of each other. For example, if we build independent breadth-first search (BFS) spanning trees for up and down links, then, due to the uneven structure of directed networks, consistent labeling of all the unidirectional links is not guaranteed. An example of such an independent construction is shown in Figure 4, where link $R1 \rightarrow R2$ is marked differently in two trees. In the up-tree (Figure 4a), link $R1 \rightarrow R2$ is towards a node ($R2$) that is closer to the root and hence marked up, while in the down-tree (Figure 4b), link $R1 \rightarrow R2$ is between nodes equi-distant

from the root node, but it is marked *down* as it is towards a node with a larger ID ($R2 > R1$). The assignment of directions to links strongly depends on the hierarchical structure of the BFS spanning tree, which is established once the root is selected, and hence independent construction of trees may lead to inconsistent labeling of links. Uni-up*/down*, therefore, builds the two trees concurrently using the BFS methodology but expands to new branches only if consistent link labelling is possible, ensuring consistent labeling by construction. Links between nodes that are at the same level for both trees, are marked based on static node IDs, similar to traditional up*/down*.

Each node in the network expands the two trees to its descendants, only when that node is reachable by both *up* and *down* trees. Otherwise, the growth of the up-tree (down-tree) beyond a node is halted till the down-tree (up-tree) reaches that node. If expanding the trees to their neighbors takes a single unit of time, then link labeling automatically terminates within a deterministic amount of time which is bounded by the number of nodes in the network. All nodes that are reachable by both the up-tree and the down-tree can communicate among themselves by enabling deadlock-free routes. All other (unreachable) nodes timeout after waiting for one or both tree(s) to reach them. Thereafter, links between nodes reachable by both trees are used to enable deadlock-free routes. As shown in Figure 5a, with $R1$ as root, the up-tree uses the link $R0 \rightarrow R1$ to expand to $R0$ and the down-tree takes the link $R1 \rightarrow R2$ to expand to $R2$. Both trees halt at their frontier nodes ($R0$ for up-tree; $R2$ for down-tree) waiting for their counterpart trees. The algorithm terminates with $R1$ connected to no other node, as down-tree (up-tree) never reaches $R0$ ($R2$) in this configuration.

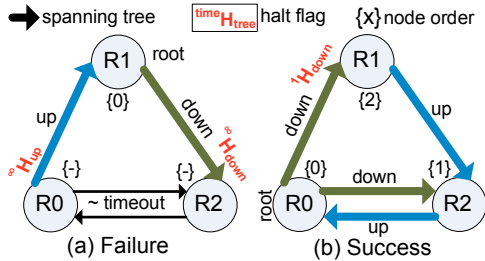


Figure 5: (a) Failure to find deadlock-free connectivity when $R1$ is chosen as root. (b) Successfully provides deadlock-free connectivity if $R0$ is chosen as root. Both up-tree and down-tree are complete.

However, the structure of both trees greatly depends on the choice of root and as shown in Figure 5, this aspect may affect the connectivity of the network. In this example, if instead of $R1$ (Figure 5a), $R0$ (Figure 5b) is chosen as root, then uni-up*/down* is able to find deadlock-free routes among all nodes in the network. In Figure 5b, starting from the root node both trees expand to $R2$ using *bidirectional* link $R0 \leftrightarrow R1$ and hence $R2$ can expand to its descendants. At the same time, the down-tree expands to $R1$ and halts at $R1$ for the up-tree to catch-up. In the next iteration, $R2$ expands the up-tree to $R1$, cancelling the halting status of $R1$. At this time, both trees are complete (reach all nodes), and hence the network is connected and deadlock-free. This is different from traditional up*/down* where connectivity is independent from the choice of root node. Thus, in order to provide maximum deadlock-free connectivity, uni-up*/down* tries all nodes as potential roots and selects the one that provides maximum connectivity. A pseudo-code of the uni-up*/down* routing algorithm is provided in Figure 6.

5. RECONFIGURATION ALGORITHM

Reconfiguration in LinkMiser is invoked upon a permanent link failure and it implements our novel uni-up*/down* routing scheme

```

for (ROOT in all_nodes): //try all nodes as roots
  unreached_nodes = all_nodes - ROOT
  while (tree_expansion_possible(ROOT)):
    for (node in unreached_nodes):
      if (is_reachable_by_both_trees(ROOT,node)):
        mark_link_towards_root_up(ROOT,node)
        mark_link_away_from_root_down(ROOT,node)
        expand_up-tree_if_backward_link(ROOT,node)
        expand_down-tree_if_forward_link(ROOT,node)
        unreached_nodes = unreached_nodes - node
    if (no_unreached_nodes_remaining()): //connected
      exit() //no need to try other nodes as roots
ROOT_max = get_root_with_max_connectivity();
apply_up_down_link_marking(ROOT_max);
disable_all_down_up_turns(ROOT_max);

```

Figure 6: **Uni-up*/down* Routing Algorithm.** All nodes are tried as root and the root that provides maximum connectivity is chosen to build the new network. Within each root trial, nodes expand the trees to their neighbors only if both up-tree and down-tree have reached them.

to provide topology agnostic routing in networks with unidirectional links. Moreover, in a disconnected network, our reconfiguration automatically discovers the largest connected topology. In contrast, most previous distributed reconfiguration approaches, particularly Ariadne [1], may construct a sub-optimal (in terms of number of nodes connected) network. In Ariadne, the node that detects the fault is fixed as the root node of the up*/down* re-routing process, and thus the construction of new routes is limited to the nodes that are connected to the node detecting failure. However, in disconnected networks, the node detecting the fault maybe a part of the sub-network that is smaller as compared to the other surviving sub-topologies. In addition, Ariadne forwards the recovery initiation flags only to the connected neighbors, and thus, nodes that are not a part of the fault-detecting node's sub-network are unaware of the reconfiguration process.

Therefore, we propose a scheme where all nodes are made aware of the reconfiguration process, irrespective of their connection status and a selection process determines the optimal choice of the root. To this end, during reconfiguration, each node broadcasts two 1-bit flags (instead of a single 1-bit flag in Ariadne) to all other nodes. The first 1-bit 'notification' flag is used to notify all nodes about recovery initiation, whereas the second 1-bit 'connection' flag is used to discover the underlying topology and build resilient deadlock-free routes. LinkMiser uses hardware that is independent from the router pipeline and is turned-on only during reconfiguration and hence, is less susceptible to wear-out faults. Moreover, even if reliability of reconfiguration logic becomes a bottleneck, simple area-inefficient resiliency schemes such as logic duplication [5] can be leveraged without significant penalty due to the low area requirement of the reconfiguration logic. Resilient LinkMiser hardware design and its independence from the router pipeline allows reliable broadcasting of reconfiguration flags even to disconnected nodes, notifying them of the reconfiguration process.

During LinkMiser reconfiguration, each node, in turn, is appointed as the temporary root node and the selection is finalized when one of the following two conditions occur: (i) a root node is found which provides deadlock-free connectivity among all nodes; or (ii) all nodes completed their attempt to serve as root node. During the trials, all nodes note the root node for which the corresponding tree could connect maximum nodes in a deadlock-free fashion. This forms the first phase of our reconfiguration algorithm, which we call the *selection* phase. Following *selection*, the root node(s) that could connect maximum nodes (winners), form separate sub-networks and each node becomes the part of the largest sub-network to which it is connected. Note that, there can be multiple winner root nodes in a disconnected network. We call this the

construction phase.

Our algorithm is fully distributed, *i.e.*, given only local information, the algorithm runs in lockstep at each network router to collectively reconfigure the network. Based on simple broadcast by network nodes, the reconfiguration procedure consists of N broadcasts in the best case to $2*N^2$ broadcasts in the worst case, when operating on a network with N nodes. Each broadcast deterministically ends in $2*N$ cycles and thus the algorithm is bounded by $4*N^3$ cycles. Each node has the local knowledge of when the reconfiguration finishes.

5.1 Selection Phase

The node detecting permanent failure initiates reconfiguration by entering the *selection* mode and broadcasting two 1-bit flags to all other nodes. This node works as the temporary root node of the reconfiguration process, and corresponds to the first root node trial (first *selection epoch*). Each *selection epoch* involves broadcasts by the remaining $N-1$ nodes, after the root node completes its first broadcast. However, during these $N-1$ broadcasts, the notification flag is not broadcasted and the nodes that receive the connection flag perform only a subset of operations that were performed during the root node broadcast. Each broadcast takes $2*N$ cycles to complete and thus one epoch of *selection* deterministically completes in $2*N^2$ cycles. A node performs different operations on receiving different 1-bit flags (notification or connection), as discussed in detail later in this section.

After the first epoch, each (remaining) node, in turn, acts as temporary root and initiates a new epoch. Thus, there are N total *selection* epochs, one for each choice of root. Meanwhile, during *selection*, all networks nodes keep track of the largest sub-network that they are connected to, and the corresponding root node of that sub-network. All nodes automatically switch to *construction* phase after the N^{th} *selection epoch*, and therefore *selection* takes a total $2*N^3$ cycles (N epochs, N broadcasts each, $2N$ cycles each broadcast). However, the state switches back directly to normal (skipping *construction*), if any root node trial finds deadlock-free connectivity among all nodes in the network. Thus, *selection* and the entire reconfiguration could potentially be completed in one *selection epoch*, *i.e.*, $2*N^2$, if the network is still connected. The activity at all nodes during one *selection epoch* are shown in Figure 7.

The notification flag is broadcasted only during root broadcast. If the notification flag is received in “normal” state, then the node invalidates the routing paths, freezes the router pipeline and sets the router’s state to “selecting”. Additionally, the node notes the beginning of a new *selection epoch* (by setting the micro-status register (MSR)). The node will automatically reset the MSR after $2*N^2$ cycles, indicating the end of current epoch. Finally, the notification flag is forwarded to all port(s) from which no flag was received earlier. Note that, notification flags are forwarded across all ports, irrespective of the condition of the corresponding network link (failed or working). However, upon receiving the connection flag, each node performs three more involved operations, the details of which are given below:

(i) Tagging Link Directions. This action applies the uni-up*/down* routing restrictions and is performed only during the broadcast by the root node. Separate *to* (broadcasting node) and *from* (broadcasting node) connection flags are used to build up-tree and down-tree, respectively. The outgoing link corresponding to the port that received the *to* connection flag is marked as *up* (towards a node closer to root), whereas the outgoing link corresponding to the port that forwarded the *from* connection flag is marked as *down* (towards a node farther from root). However, nodes at equal distance from the root can forward flags to each other in the same cycle. In such a

scenario, ties can be broken by comparing the statically assigned node IDs: the node with the higher ID marks the outgoing link as *up*, while the other node marks the outgoing link as *down*. Time multiplexing is used to forward *to* and *from* flags separately on a single connection wire and thus broadcasts are bounded by $2*N$ cycles (instead of N cycles for a single flag transfer).

(ii) Routing Table Update. During broadcast by any node (including root), the broadcasting node informs the other nodes how it can be reached. A (*to*) connection flag reception at a port implies that the reverse path of flag broadcast can be followed to reach the broadcasting node. Since bidirectional connectivity is a must, both *to* and *from* flag receptions are required at a node during root node broadcast for it to record the routes to any other node. A bidirectionally connected node records the ports where the *to* connection flag was received from in its routing tables, learning the path to the broadcasting node. Additionally, the reachability counter (RC) at the node is incremented to indicate the discovery of a route to a new node. Moreover, broadcasts should spread only through working links and enabled turns. This means that *to* connection flags are forwarded only if the link is working in the reverse direction of forwarding, whereas *from* connection flags are forwarded if the link is working in the direction of transmission. Also, each node during connection flag forwarding only forwards flags to ports on enabled turns.

(iii) Flag Forwarding. Each node initiates the broadcast by forwarding the *to* connection flag if a link is functional in the reverse direction of forwarding, followed by *from* connection flag forwarding if a link is working in the same direction as forwarding. Thus, each forwarding event is completed in two cycles. All other nodes decipher whether they receive a *to* flag or *from* flag depending on odd and even cycle numbers respectively. In the next couple of cycles, the receiving node forwards the *to* and *from* flags one by one only to those port(s) that did not receive a flag earlier. Additionally, flags are forwarded only across enabled turns: *to* connection flags received from *up* ports are not forwarded to *up* ports, because this will allow a *down*→*up* path in the reverse direction. In addition, nodes forward flags only if they had received both *to* and *from* connection flags during the root node broadcast. This applies the constraint of growing *up* and *down* spanning tree together, as discussed in Section 4.1.

Orchestration of Selection Epochs. At the end of each selection epoch (indicated by resetting of MSR), each node performs a series of checks and status updates to orchestrate the selection phase. All nodes, if connected, resume normal operation at the end of *selection epoch*. A connected network can be easily detected by checking the contents of the routing table. As our resilient routing algorithm only provides bidirectional connectivity, either all nodes have full routing tables or all nodes have incomplete routing tables. This implies that the entire network is connected only if $RC=N$ (reachability counter counts the number of connected nodes), at the end of any *selection epoch*. Otherwise, if routing tables are not complete ($RC \neq N$), the value of RC is compared to the maximum RC achieved during all previous *selection epochs* (different roots). If current RC value is greater than the maximum RC value from previous trials, then the corresponding $\langle \text{ROOT}, \text{RC} \rangle$ value pair is stored locally at each node in-place of the previous maximum $\langle \text{ROOT}, \text{RC} \rangle$ pair. A separate counter, named the epoch counter (EC) is maintained to count the number of finished epochs. *Selection* phase automatically switches to *construction* phase when all root trials are finished ($EC=N$). The *construction* phase, discussed in the next section, utilizes the maximum $\langle \text{ROOT}, \text{RC} \rangle$ values at each node to connect best effort (large) sub-networks. At the end

ROOT SELECTION – EPOCH ACTIVITY

\forall node X; node X is ROOT;

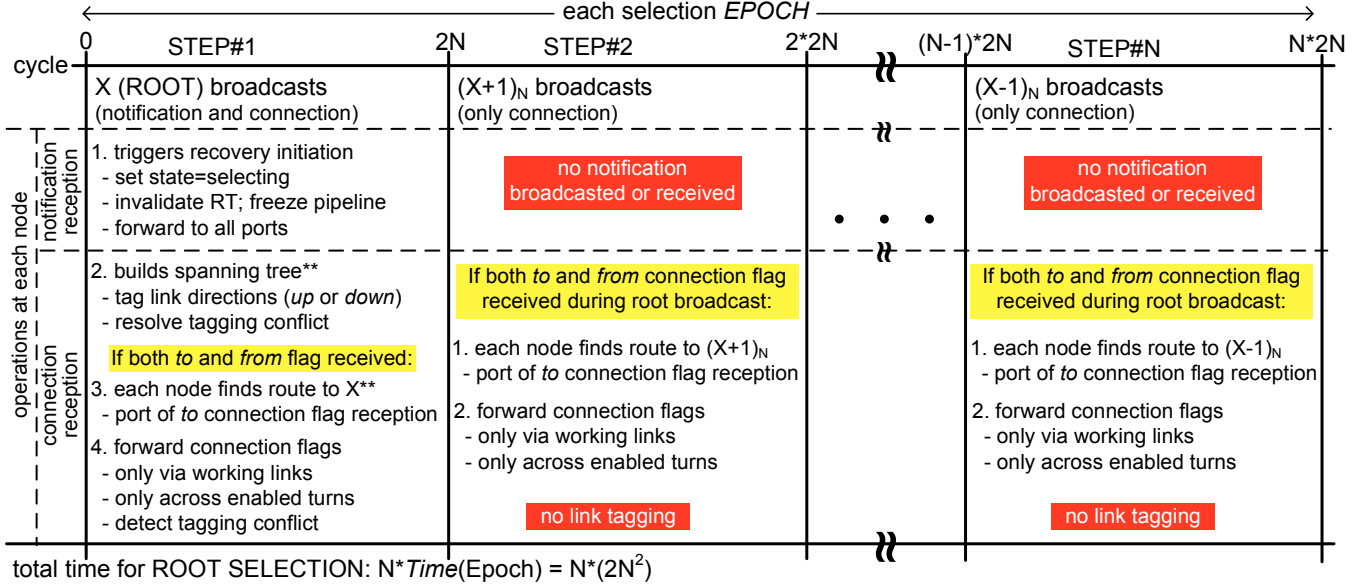


Figure 7: **Selection Epoch Activity**. Each epoch completes in $2N^2$ cycles.

of each epoch, each node also pro-actively tries to initiate a new epoch (with itself as root), if the node has not been tried as root before. A node stops attempting to be root if it receives a notification flag indicating that some other node has already initiated an epoch.

5.2 Construction Phase

The working of *construction* phase is similar to the *selection* phase, however, only the winner root nodes (instead of all nodes) construct separate sub-networks and routes discovered by this phase are final, *i.e.*, they are not erased in favor of a possibly larger sub-network. Note that there can be multiple winners in a disconnected network as nodes only have local knowledge about the sub-network they are connected to. All nodes are independently aware of the end of *selection* phase and at the start of *construction* phase, all winners pro-actively try to initiate constructing their own sub-network, with themselves as root. The winner root that gets the first construction slot starts constructing its sub-network by broadcasting the notification and connection flags, almost in a similar fashion to one *selection epoch*. However, as shown in Figure 8 individual nodes (e.g. R2) or groups of nodes, can be part of multiple sub-networks. Thus, to enable the largest surviving topology, each node chooses the largest sub-network it is connected to and discards all communication from other sub-networks during *construction* phase. The effective size of other sub-networks is reduced, as can be seen in the example of Figure 8 where sub-network#1 is reduced to just two nodes (R0 and R1). Again, it takes exactly N broadcasts to complete one *construction epoch*. After each *construction epoch*, the remaining winner roots start their *construction* process, one by one, in available time slots. Reconfiguration deterministically terminates when no flags are received at nodes for $2*N^2$ cycles (N broadcasts of $2*N$ cycles each), as then all nodes infer that no new winner root remains. Thus, in the worst case, when all network nodes are isolated due to faults, there will be N winner root nodes and the entire *construction* process will take $2*N^3$ cycles (N epochs, N broadcasts each, $2*N$ cycles each broadcast).

We implement the timing of our reconfiguration algorithm us-

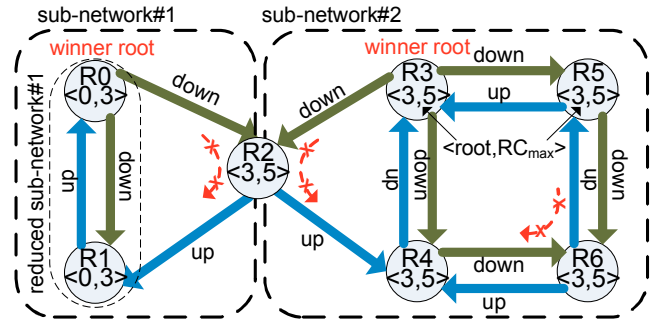


Figure 8: **LinkMiser: Construction Phase**. Multiple winner roots are possible after the *selection* phase if the network is disconnected. Individual nodes (e.g. R2) or groups of nodes, can be a part of multiple sub-networks. Each node chooses the largest sub-network it is connected to and discards all communication from other sub-networks during *construction* phase. For example, sub-network#1 reduces to just R0 and R1, as R2 is also a part of a larger sub-network (#2) rooted at R3.

ing atomic broadcasts based on cycle numbers, where broadcasting node's nodeID is inferred by the cycle number at which the broadcast is initiated. Each node is assigned different time windows of $2*N$ cycles each, for broadcasting, during which the remaining nodes are prevented from broadcasting. The time slots of all nodes in the network are interleaved to provide equal opportunity to all nodes to initiate broadcast. Thus, each node has the knowledge of when to broadcast so that no other broadcasts collide and all the flag recipients have exact knowledge of the broadcast initiator node. This is similar to the implementation of timing in Ariadne's [1] reconfiguration.

6. EXPERIMENTAL RESULTS

We evaluate LinkMiser by modeling a NoC system in a cycle-accurate C++ simulator based on [6]. The baseline system is an 8x8 mesh network with generic 5-stage pipeline 2-VC per port routers. Each input channel is 64 bits wide and each VC buffer is 5-entry wide. In addition, the NoC is augmented with Link-

Miser reconfiguration capabilities described in Section 5. Moreover, Ariadne, which reportedly outperformed all previous on-chip distributed reconfiguration proposals, is implemented for comparison. The framework is analyzed with two different types of workloads (1a): directed random traffic (uniform random), as well as applications from the PARSEC suite [2]. PARSEC application traces are obtained from Wisconsin Multifacet GEMS simulator [15] modeling a similar network and configured as detailed in Table 1b.

(a) System Input		(b) System Configuration (GEMS)	
synthetic traffic	uniform random	processors	in-order SPARC cores
benchmark	PARSEC	coherence	MOESI protocol
packet length	1flit (control) 5flits (data)	L1 cache	private 32KB/node ways:2 latency:3
simulation time	250K cycles	L2 cache	shared 1MB/node ways:16 latency:15
warm-up period	10K cycles		

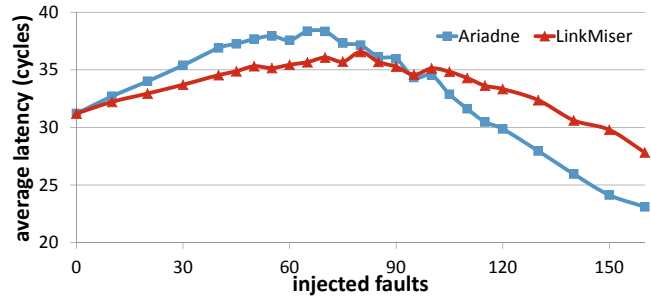
Table 1: **Simulation Infrastructure.** (a) GEMS configuration used to generate PARSEC traces. (b) Simulation inputs.

To accurately assess the impact of transistor failures in an NoC, we developed an architectural level fault model similar to Vicis [9] and Ariadne [1]. Their fault model maps gate-level faults to link-level faults by randomly injecting faults in a network netlist, and counting the number of dysfunctional links thereafter. After a statistically significant number of fault experiments, the model provides a probabilistic mapping of number of gate faults to number of link failures. Although this fault model is fairly accurate for scenarios where all permanent faults occur at once, it can be fairly off when modeling real world fault manifestations. In reality, permanent faults due to wear-out cause devices to fail one-by-one over time, and after each failure reconfiguration mechanisms (leveraged by diagnosis information) bypass the faults by disabling parts of the network. However, due to the limited resolution of reconfiguration, often healthy components are disabled too. Specifically, as soon as a fault manifests in one unidirectional link in *CG_FModel*, the fault-free opposite unidirectional link is also disabled, and hence wear-out faults cannot further affect this unidirectional link, as it is no longer used. Thus, to accurately model faults, we do not map faults to a unidirectional link if the opposite direction link is already declared faulty and bypassed by the reconfiguration mechanism.

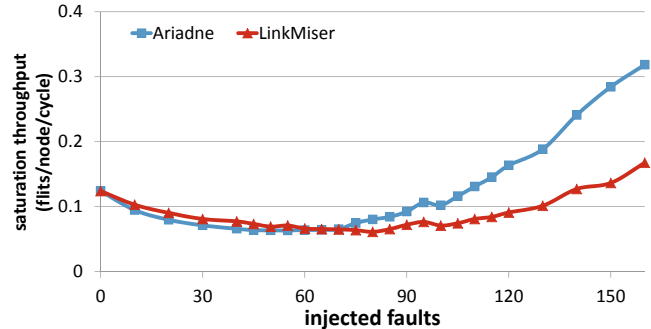
6.1 Performance Evaluation

To analyze LinkMiser’s performance impact we injected a varying number of permanent faults (0-160 transistor failures) into the NoC infrastructure at (100) different random locations each. All our performance results are reported for the largest surviving topology, under the assumption that after occurrence of faults the operating system maps processes over the largest surviving topology only. In our evaluation we report two performance metrics: average network latency and saturation throughput.

First, we report the zero-load latency, that is the steady-state latency of a lightly loaded network (0.01 flits injected per cycle per node). It reflects the average delivery time of a network without resource congestion, and hence in a sense indicates the average length of routes between network nodes. Analysing Figure 9a, both Ariadne and LinkMiser initially show increase in latency with increasing faults, although LinkMiser degrades gracefully as it is resource conscious and provides greater path diversity. For example, at 50 faults, LinkMiser on average has 6.7% less latency than Ariadne. However, as more faults accumulate, the network gets disconnected into multiple sub-networks, and thus average latency drops as packets have to route within smaller network sections. As Ariadne loses nodes more quickly than LinkMiser (Section 6.2), its latency shows a greater improvement. Similar trends were observed when a LinkMiser-equipped network is simulated with PARSEC benchmark traces, as plotted in Figure 10.



(a) **Zero Load Latency** increases gracefully in LinkMiser. However, latency decreases as the surviving topology gets smaller. At high fault rates, LinkMiser shows worse latency characteristics than Ariadne as it connects significantly more nodes.



(b) **Saturation Throughput** is initially higher for LinkMiser (compared to Ariadne), but as faults accumulate LinkMiser loses due to a significantly smaller surviving topology in Ariadne.

Figure 9: **Performance under uniform random traffic.**

Figure 9b plots the packet throughput delivered by the network when heavy traffic is injected. Similar to the latency characteristics, before the network gets disconnected, LinkMiser shows graceful degradation in delivery capacity. For example, at 50 faults, LinkMiser delivers 9.1% more packets than Ariadne. However, as more faults cause the network to be segmented, throughput improves because packets are routed within small subnetwork partitions. As expected, the throughput improvement is greater for Ariadne equipped networks.

6.2 Reliability Evaluation

As faults accumulate, networks often become disconnected. Performance of parallel workloads running on a multi-core chip with a faulty network directly depends on the size of the largest surviving sub-network. A bigger surviving topology enables more processing cores for applications. Thus, the ability of an algorithm to maximize the connectivity of a faulty network is critical, since if no routes are available between two nodes, they cannot work collaboratively. Figure 11 plots average size of the maximum surviving topology against the number of injected faults. LinkMiser, due to its resource consciousness, loses fewer nodes as compared to Ariadne. At 50 faults, LinkMiser on average connects 4.4% more nodes, a gain which goes up to 36% at 100 faults and 83% at 160 faults. Due to the significantly large surviving topology, LinkMiser has higher latency and lower throughput at high fault rates.

After faults manifest, only a few connected components within the network are able to provide full functionality of multi-cores, as there are certain nodes (main memory controllers, IO controllers) that are vital to their functioning. Thus, it is essential for a connected sub-network to include at-least each one of these vital nodes. However, the probability of a connected sub-network having all vital functionalities decreases rapidly as the network gets more seg-

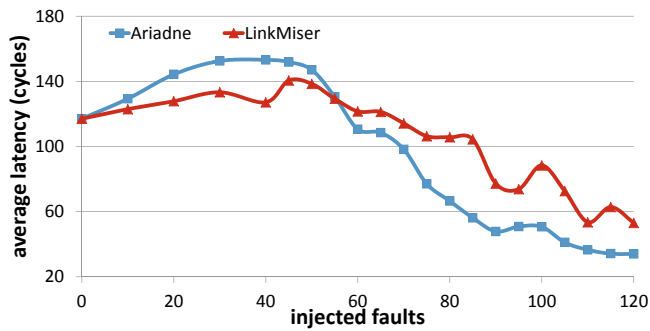


Figure 10: **Performance under PARSEC benchmark traces.** Average latency across all benchmarks and fault configurations is plotted against number of injected faults. LinkMiser provides additional path diversity (and hence graceful degradation) at low fault rates and greater node connectivity at high fault rates (causing only moderate decrease in latency).

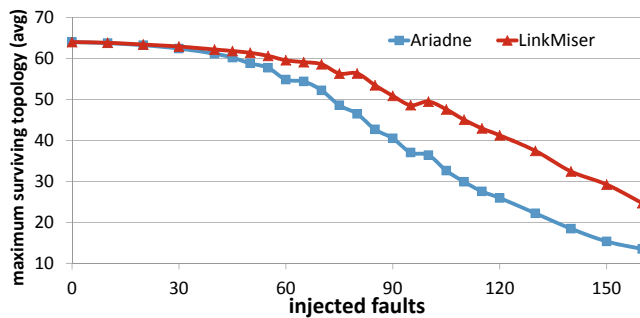


Figure 11: **Average Size of Largest Surviving Topology.** Connectivity in LinkMiser degrades gracefully.

regated, that is, the more the number of sub-networks, the less the probability of any one having a full set of functionalities. Figure 12 plots the number of separate sub-networks that are formed as an NoC is injected with faults. At 50 faults, LinkMiser segregates into 34% fewer connected sub-networks as compared to Ariadne, whereas beyond 100 faults Ariadne has almost twice as many sub-networks as LinkMiser.

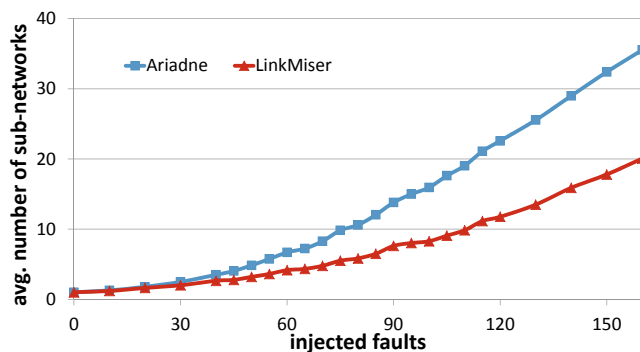


Figure 12: **Average Number of Connected Sub-Networks.** With increasing faults, LinkMiser segregates into fewer sub-networks.

7. CONCLUSIONS

We have presented LinkMiser, a solution for reliable operation of NoCs incurring high fault rates. LinkMiser leverages uni-up*/down* routing, a novel variant of traditional up*/down* routing, to maximally utilize all the working links in the NoC. Moreover, an area efficient implementation is enabled by our novel distributed reconfiguration algorithm, that places no restriction on the number or location of faults. Simulations show that for a 64-node NoC, LinkMiser provides 4.4% connectivity and 6.7% latency benefit (50

faults) over state-of-the-art on-chip reconfiguration solution. Reliability characteristics further improve at high fault rates, making LinkMiser an excellent choice for aggressively scaled silicon.

8. 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] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. In *Proc. PACT*, October 2008.
- [3] S. Borkar. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *Micro, IEEE*, 25(6), 2005.
- [4] G.-M. Chiu. The odd-even turn model for adaptive routing. *IEEE Trans. Parallel and Distributed Systems*, 11(7), 2000.
- [5] K. Constantinides, S. Plaza, J. Blome, B. Zhang, V. Bertacco, S. Mahlke, T. Austin, and M. Orshansky. BulletProof: A defect-tolerant CMP switch architecture. In *Proc. HPCA*, 2006.
- [6] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., 2003.
- [7] A. Dutta and N. A. Touba. Reliable network-on-chip using a low cost unequal error protection code. In *Proc. DFT*, 2007.
- [8] 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*, 2009.
- [9] D. Fick, A. DeOrio, J. Hu, V. Bertacco, D. Blaauw, and D. Sylvester. Vicis: A reliable network for unreliable silicon. In *Proc. DAC*, 2009.
- [10] C. Glass and L. Ni. The turn model for adaptive routing. In *Proc. ISCA*, 1992.
- [11] C. J. Glass and L. M. Ni. Fault-tolerant wormhole routing in meshes without virtual channels. *IEEE Trans. Parallel and Distributed Systems*, 7, 1996.
- [12] M. Gomez, J. Duato, J. Flich, P. Lopez, A. Robles, N. Nordbotten, O. Lysne, and T. Skeie. An efficient fault-tolerant routing methodology for meshes and tori. *Comp. Arch. Letters*, 3(1), 2004.
- [13] M. Koibuchi, A. Funahashi, A. Jouraku, and H. Amano. L-turn routing: An adaptive routing in irregular networks. In *Proc. ICPP*, 2001.
- [14] M. Koibuchi, H. Matsutani, H. Amano, and T. M. Pinkston. A lightweight fault-tolerant mechanism for network-on-chip. In *Proc. NoCs*, 2008.
- [15] M. Martin, D. Sorin, B. Beckmann, M. Marty, M. Xu, A. Alameldeen, K. Moore, M. Hill, and D. Wood. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *ACM SIGARCH Computer Architecture News*, 33(4), 2005.
- [16] 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.
- [17] S. Murali, et al. Analysis of error recovery schemes for networks on chips. *IEEE Design & Test*, 22(5), 2005.
- [18] R. Parikh, A. Ghofrani, V. Bertacco, and K.-T. Cheng. Comprehensive online defect diagnosis in on-chip networks. In *WRA*, 2011. <http://eecs.umich.edu/~parikh/documents/wra11.pdf>.
- [19] V. Puente, J. A. Gregorio, F. Vallejo, and R. Beivide. Immunit: A cheap and robust fault-tolerant packet routing mechanism. In *Proc. ISCA*, 2004.
- [20] M. Schroeder, A. Birrell, M. Burrows, H. Murray, R. Needham, T. Rodeheffer, E. Satterthwaite, and C. Thacker. Autonet: A high-speed, self-configuring local area network using point-to-point links. *IEEE Trans. Selected Areas in Communication*, 9(8), 1991.
- [21] S. Shamshiri, A. Ghofrani, and K.-T. Cheng. End-to-end error correction and online diagnosis for on-chip networks. In *Proc. VTS*, 2011.
- [22] J. Wu. A fault-tolerant and deadlock-free routing protocol in 2d meshes based on odd-even turn model. *IEEE Trans. Computers*, 52(9), 2003.