Resource Conscious Diagnosis and Reconfiguration for NoC Permanent Faults

Ritesh Parikh, *Member, IEEE* and Valeria Bertacco, *Senior Member, IEEE*

Abstract—Networks-on-chip (NoCs) have been increasingly adopted in recent years due to the extensive integration of many components in modern multicore processors and system-on-chip designs. At the same time, transistor reliability is becoming a major concern due to the continuous scaling of silicon. As the sole medium of on-chip communication, it is critical for a NoC to be able to tolerate many permanent transistor failures. In this paper, we propose uDIREC, a unified framework for permanent fault diagnosis and subsequent reconfiguration in NoCs, which provides graceful performance degradation with an increasing number of faults. Upon in-field transistor failures, uDIREC leverages a fine-resolution diagnosis mechanism to disable faulty components very sparingly. At its core, uDIREC employs MOUNT, a novel routing algorithm to find reliable and deadlock-free routes that utilize all the still-functional links in the NoC. We implement uDIREC's reconfiguration as a truly-distributed hardware solution, still keeping the area overhead at a minimum. We also propose a software-implemented reconfiguration that provides greater integration with our software-based diagnosis scheme, at the cost of distributed nature of implementation. Regardless of the adopted implementation scheme, uDIREC places no restriction on topology, router architecture and number and location of faults. Experimental results show that uDIREC, implemented in a 64-node NoC, drops 3× fewer nodes and provides greater than 25 percent throughput improvement (beyond 15 faults) when compared to other state-of-the-art fault-tolerance solutions. uDIREC's improvement over prior-art grows further with more faults, making it a effective NoC reliability solution for a wide range of fault rates.

Index Terms—NoC, permanent faults, diagnosis, reconfiguration, reliability

1 Introduction

As silicon scales, chip multi-processors (CMP) and system-on-chip (SoC) designs are dramatically changing from limited and robust logic blocks to integrating many fragile transistors into a growing number of simple, power-efficient cores/IPs. The corresponding transition from computation-centric to communication-centric designs has compelled architects to design complex high-performance on-chip interconnects, most commonly, networks-on-chip (NoCs). Although highly scalable, NoCs as the sole medium for on-chip communication, can become a single point of failure under transistor malfunctions. A large scale processor reliability study has observed a permanent failure rate that is an order of magnitude higher than previously assumed [1]. Moreover, technology experts predict more frequent failures in the field at future technology nodes [2], [3].

Researchers have already proposed architectures that can gracefully tolerate up to a few hundred (∼500) processor-logic permanent faults [4], [5] in a 64-node CMP. However, to extend a chip's lifespan, comparable fault-tolerant solutions are required for both processors and the fabric interconnecting them. Faults in the interconnection network can potentially lead to a disconnected network and to the loss of healthy processing elements (PEs). State-of-the-art permanent fault-tolerant solutions for NoCs (fault-diagnosis: [6], reconfiguration: [7], [8]) fall significantly short of this goal, dropping the majority of potentially-healthy nodes at a high number of faults. To address this problem, we propose a novel solution, called uDIREC, which drops over 3× fewer nodes due to NoC-faults than existing solutions, and thus minimizes the network-induced loss of processing capability.

Existing NoC reliability solutions can be broadly divided into *architectural* protection against faults in the router logic [6], [9], [10] and *route-reconfiguration* solutions to bypass faulty links or routers [6], [7], [8], [11], [12]. *Reconfiguration* solutions may serve as fallback reliability schemes when router components are beyond the *architectural* repair capabilities. A common technique in this regard is to model a transistor failure in the router logic as failure of all links connected to the affected router, and then re-routing around the faulty links. In this paper, we focus on reconfiguration solutions, as they provide protection against all types of NoC faults.

Fine-resolution diagnosis of permanent faults is essential to minimize functionality or processing loss due to reconfiguration. Fortunately, majority of a router's logic faults affects only small localized portions of the router [6]. To accurately assess this phenomenon, we injected stuck-at faults in a five-port wormhole router gate-level netlist, with a spatial distribution proportional to the silicon area of gates and wires. Our analysis of the fault sites revealed that most (96 percent) faults affect only small fractions of the router logic: their functionality can be entirely masked by disabling and re-routing around a single unidirectional router link. Assuming a mechanism capable of such fine-resolution

- *R. Parikh is with Intel Corporation, Santa Clara, CA 95054. E-mail: ritesh.parikh@intel.com.*
- *V. Bertacco is with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109. E-mail: valeria@umich.edu.*

Manuscript received 21 July 2014; revised 11 July 2015; accepted 29 Aug. 2015. Date of publication 0 . 0000; date of current version 0 . 0000.
Recommended for acceptance by J.D. Bruguera.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2015.2479586

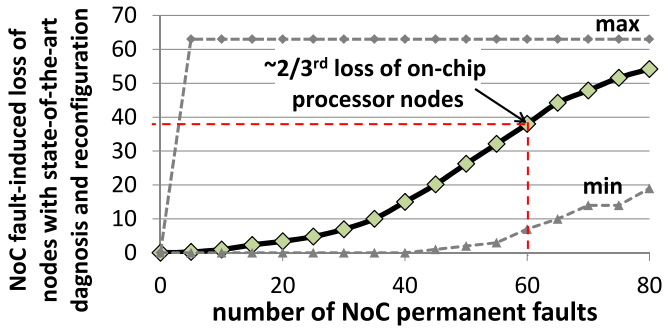


Fig. 1. Loss of healthy nodes due to faults in a 64 node mesh NoC equipped with state-of-the-art diagnosis [6] and reconfiguration [7]. Approximately 2/3*rd* nodes are unreachable at only 60 NoC faults.

diagnosis, an efficient reconfiguration scheme can potentially shield against many NoC faults, disabling only a single unidirectional link for each fault. However, existing on-chip, topology-agnostic route-reconfiguration solutions [7], [8] fail to exploit this opportunity as they can operate with bidirectional links only. For example, both Ariadne [7] and Immunet [8] view the network as an *undirected* graph, and are based on the construction of a new *undirected* spanning tree after each failure. The routes derived from such spanning tree approaches are therefore valid only when all edges in the graph provide bidirectional connectivity. Therefore, these reconfiguration schemes unnecessarily consider a fault in one direction to be fatal for the entire bidirectional link, and *cannot* benefit from the fine-grained diagnosis information. Therefore, we explore a route-reconfiguration solution capable of separately exploring unidirectional paths (routes) away and towards the root node of the spanning tree.

1.1 Motivation

We back our claim that current NoC fault-tolerance techniques are insufficiently robust with a quantitative study. Industrial CMP designs [13], [14], dedicate 6-15 percent of the chip area to the NoC and roughly 50 percent to the processor logic, while the rest is dedicated to memory, I/O, etc. Assuming uniform distribution of faults over silicon area, the NoC should be able to gracefully tolerate up to 60-150 faults to match processor-oriented reliability schemes [4], [5]. To analyze the effect of faults, we modeled a 64-node NoC, equipped with state-of-the-art fault-diagnosis [6] and route-reconfiguration [7] schemes. Fig. 1 plots the average number of processing nodes unreachable by such a NoC with increasing number of faults. This figure shows that existing solutions deteriorate considerably beyond 30 transistor faults, dropping many nodes with just a few additional faults (see slope change). At 60 NoC-related faults, almost 2/3*rd* of the nodes become isolated, many of which are expected to be functional cores. In addition, even at as few as five faults, the network can potentially drop all the nodes (max-line), corresponding to the situation when faults accumulate around the node running fault detection and reconfiguration routines. The figure also plots minimum number of lost nodes (min-line) for a given number of faults. Low number of dropped nodes on the min-line shows previous solution's heavy dependency on fault locations.

1.2 Our Approach

Existing approaches for building fault-tolerant NoCs disable whole routers and links to recover the communication fabric from permanent faults. This heavy-handed approach to recovery cannot tolerate the fault density predicted for future silicon. To this end, we first proposed uDIREC in [15]: a frugal approach to reliable NoC design, pairing a fine-resolution diagnosis and careful shutdown with a capable routing algorithm. In this work, we evolve uDIREC into an end-to-end reliability solution by integrating a fully-distributed and lightweight hardware-based reconfiguration algorithm that repairs and restarts the NoC after fault manifestation. We also present an in-depth design and analysis of various reconfiguration implementation choices, while expanding on reliability evaluations. Our experiments on a 64-node NoC with 10-60 faults show that uDIREC drops 60-75 percent fewer nodes and provides 14-40 percent higher throughput over other state-of-the-art fault-tolerance solutions. Moreover, uDIREC's distributed hardware costs less than 3 percent area over the NoC, and executes the reconfiguration in around 1M NoC hardware cycles. uDIREC (**u**nified **DI**agnosis and **REC**onfiguration) integrates the following solutions:

i) **A low-cost diagnosis scheme** that localizes faults at unidirectional link granularity.
ii) **A novel routing algorithm (MOUNT)** to maximize the utilization of unidirectional links in fault-ridden irregular networks that result from the application of our fine-grain fault diagnosis scheme to faulty NoCs.
iii) **A reconfiguration scheme** to provide end-to-end NoC reliability. It implements our MOUNT routing algorithm to discover a new set of deadlock-free routes on each fault manifestation. It places no restriction on topology, router architecture or the number and location of faults. Our reconfiguration scheme can be implemented by leveraging fully-distributed hardware or via a more flexible software solution. Each implementation comes with its pros, and the choice provides designers with the option of optimizing based on system requirements.

2 Related Work

Ensuring reliability in NoCs has been the subject of much previous research, focusing on a variety of aspects. Works such as [16], [17], [18] focus on NoC protection against soft faults. Other methods enhance NoC reliability against permanent faults by enabling one or a combination of the following features: i) detection of erroneous behavior [6], [17], [19], ii) diagnosis of fault site [6], [19], [20], [21], [22], [23], iii) recovery from erroneous state [17], [24], iv) system reconfiguration to bypass the permanent faults [6], [7], [8], [11], [12] or v) architectural protection for router logic [6], [9], [10]. In contrast, uDIREC provides unified detection, diagnosis and reconfiguration capabilities, enabling an end-to-end reliability solution.

uDIREC is orthogonal to architectural approaches that extend the lifetime of NoC links or components, such as, ECC [17], reversible transmission [25], partially-faulty links [26], and reliable components ([9], [10]). Once the components or links are beyond the repair capabilities of

TABLE 1
uDIREC's Comparison with Other Reconfiguration Solutions

solution	diagnosis support	resolution		node-drop rate	reconf area
		diagnosis	reconf		
off-chip	NO	–	bi-link	high, $>3\times$	23%
Immunet	NO	–	bi-link	high, $>3\times$	6%
Vicis	**YES**	sgmt-pair	bi-link	high,dlock	1.5%
Ariadne	NO	–	bi-link	high, $>3\times$	2%
uDIREC	**YES**	**segment**	**u-link**	**low, $1\times$**	**1-3%**

uDIREC provides unified fault diagnosis and reconfiguration at fine granularity leading to greater robustness. Moreover, uDIREC's hardware additions are small and simple. The area numbers for schemes other than uDIREC are reported from prior-work [7].

such schemes, uDIREC provides fault tolerance by routing around these faulty parts. uDIREC also differentiate itself against adaptive routing algorithms, such as [27], [28], [29], [30], that utilize escape VCs or turn-model to overcome only a few faults in regular topologies. Certain other routing algorithms [22], [31], [32] flood or deflect packets to random neighbors to provide probabilistic reliability. In this work, we specifically investigate fine-resolution diagnosis and route-reconfiguration to cope with permanent faults.

During route-reconfiguration on fault detection, a new set of deadlock-free routes are generated to replace the current routes. The unpredictable number and location of fault occurrences, result in reconfiguration solutions designed for a bounded number [27], [33], fixed pattern [34], [35], [36] or constrained region [37] of faults being unfit for NoCs. Therefore, we only compare against solutions that put no constraints on number and location of faults. Table 1 presents a qualitative comparison of the algorithms in this domain. All previous reconfiguration algorithms, either off-chip [38], [39], [40] or on-chip [6], [7], [8], [12], are limited to the granularity of a bidirectional link, if not any coarser. Therefore, they fail to capitalize on the performance and reliability benefits of using the still-healthy unidirectional links. This is confirmed by the fact that uDIREC drops less than 1/3rd of the nodes when compared to the best performing prior-art [7].

Except for Vicis [6], which uses a costly BIST (10 percent overhead [41]) unit for diagnosis, no other previous solution presented a unified approach to diagnosis and reconfiguration. Typically, standalone route-reconfiguration schemes assume an ideal accuracy diagnosis scheme, which either localizes a fault to an entire link/router [8], [38], [39], [40], or to a *datapath segment pair* (defined in Section 3) [7].

uDIREC uses lightweight hardware additions to implement its reconfiguration, incurring an area cost comparable to the area-efficient route-reconfiguration schemes [6], [7]. Both Ariadne [7] and Immunet [8], however, do not include a diagnosis mechanism, including which would significantly increase their area footprint. On the other hand, implementing off-chip reconfiguration schemes requires dedicated reliable resources for the collection of the surviving topology and the distribution of routing tables, to and from a central node, respectively. Ariadne [7] reports that the software-managed reconfiguration algorithms for off-chip networks, lead to 23.2 percent area overhead, if implemented on-chip without any modifications.

Immunet [8] ensures reliable deadlock-free routes by reserving an escape virtual network with large buffers, which, in the worst case, reconfigures to form a unidirectional ring of surviving nodes. This leads to a high area and power overhead. On the other hand, Vicis's [6] reconfiguration can be implemented at low-cost, however, the underlying routing algorithm is not deadlock-free, and it often deadlocks at high number of faults.

3 Fine-Resolution Fault Diagnosis

NoC faults, though most only affecting the functionality of a single unidirectional link, are difficult to diagnose at this fine-granularity. Vicis [6] and [21] proposed embedding a BIST unit at each router for this purpose. However, an area overhead of >10 percent makes such a solution expensive. Although low-cost assertion checkers for fault detection have recently been proposed [42], they do not provide diagnostic information about the faulty components. Fortunately, a low-cost, passive, fine-grained scheme for detection and diagnosis of NoC link faults was proposed in [23]. Their approach leverages a centralized scoreboard that maintains the probability of each NoC link being faulty in software. The scoreboard is updated by passively monitoring traffic delivered at each network node. Ref. [23] also adds flit-level end-to-end ECC to detect and correct data faults in the transmitted packet. Upon an erroneous transmission, the work in [23] uses the ECC information to correct the packet and extract its source and destination addresses. This information is then sent to a designated "supervisor node" responsible for maintaining the scoreboard. The "supervisor node" updates all scoreboard entries corresponding to the links in the routing path of the erroneous packet (determined by analyzing the source and destination node addresses), to track the likelihood of any of these links being faulty. After accumulating the fault probabilities from just a few erroneous packets ($>=15$), the scoreboard entry with the highest accumulated fault probability is declared faulty. The diagnosis approach was reported to be accurate more than 98 percent of the time, with the accuracy increasing over more monitored packets. Note that [23] reports that this solution is successful both on deterministic and minimally adaptive routing algorithms. However, this diagnosis scheme is limited to faults affecting the links between the routers and it does not tackle faults within the router logic.

The work in [23] was extended by [19], [43], where each fault is mapped at the architectural level to one or more link failures. Consequently, [43] can localize fault manifestations in any router component (links, datapath or control logic). This solution achieves the same high accuracy of diagnosis as [23], at an area cost of less than 3 percent [43], and without introducing any test-specific traffic into the NoC. The improvements were driven from the perception of the router's datapath as an extension of the links connected to it. More specifically, faults in the following set of datapath components were considered non-differentiable from a diagnosis perspective: i) output buffer at the upstream router, ii) link between routers, iii) input buffer at the downstream router, and iv) crossbar contacts to (from) the output (input) port. Fig. 2 shows a high-level diagram of neighboring routers, and it highlights the components that are non-differentiable from a diagnosis perspective.

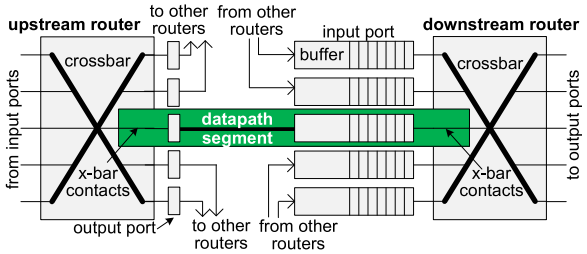


Fig. 2. **A datapath segment** includes the crossbar contacts to the output port and the output port in the upstream router, the link, the input port and the crossbar contacts from it in the downstream router.

Thus, faults in all these components can be modeled as a single link failure, and handled by re-routing around that link. We call the combined set of these datapath components, a *datapath segment*, while we refer to datapath segments corresponding to opposite unidirectional links as a *datapath segment pair*. In this work, we further the diagnosis resolution by recognizing that the two opposite unidirectional links between adjacent routers, are associated with independent datapath segments, and faults in one datapath segment do not affect the functionality of the other. We empirically studied the micro-architecture of wormhole routers to establish that a majority of faults can indeed be masked by re-routing around a single datapath segment. To this end, we synthesized a five-port baseline mesh router, and conceptually divided all router components into two pools: i) components that only affect one datapath segment's functionality, and ii) components that affect the entire router's functionality. Examples in the former category are crossbars, output ports, input ports and links, while the latter category includes arbiters, allocators and routing table. Components that only affect one datapath-segment, accounted for 96 percent of the router's silicon area, that is, 96 percent of the faults affects only one datapath segment, assuming an area-uniform fault distribution. The remaining 4 percent transistor faults typically lead to the loss of the entire router, which from the point of view of our diagnosis scheme is equivalent to the malfunctioning of all links connected to it. Our diagnosis scheme will report the links connected to the defective router faulty one-by-one, and our reconfiguration scheme will reconfigure after each new diagnosis.

Since the majority of a router's area is dedicated to datapath components, most NoC faults can be masked by disabling only single unidirectional links. For simplicity of presentation, we will predominantly use the term "unidirectional link" to refer to a datapath segment. In our experimental evaluation, we have modeled our extension of [43], which can localize most fault manifestations to the resolution of a single datapath segment. The same diagnosis information is provided to all evaluated route-reconfiguration schemes (uDIREC and prior works).

4 Routing Algorithm

In previous reconfiguration approaches, a fault affecting one unidirectional link is tagged as a failure of the entire bidirectional link. This constrains the residual network to have bidirectional links only. Reconfiguration solutions [6], [7], [8] based on such a fault model are therefore often inspired by routing algorithms designed for irregular networks with bidirectional links only [38], [44]. Up*/down* routing [38] is a classic example of a topology-agnostic algorithm for networks with bidirectional links that was adopted for providing fault-tolerance via route-reconfiguration [7]. Up*/down* works by assigning directions to all links in the network: *up* or *down*. Links towards the root node (connecting to a node closer to the root) are tagged as *up* links, while links away from the root are tagged as *down* links. Links between nodes equidistant from the root are tagged arbitrarily. All cyclic dependencies are broken by disallowing routes traversing a *down* link followed by an *up* link.

Unfortunately, up*/down* routing cannot utilize the additional unidirectional links reported as healthy by our fine-grained diagnosis scheme. Therefore, we developed a novel routing algorithm, called MOUNT, that fully utilizes the healthy unidirectional links, while still providing deadlock-free connectivity. The constraint that all network links must be bidirectional enables a desirable property: if a path between two nodes exists, irregular routing algorithms based on spanning tree construction can enable at least one deadlock-free route between them. In contrast, finding deadlock-free routes between any pair of nodes in a connected network is not always possible if the network has unidirectional links. Since the MOUNT routing algorithm must enable only deadlock-free routes, it may sacrifice the connectivity between a few nodes to achieve this goal. The connectivity and deadlock-freedom aspect of networks with unidirectional links is covered in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TC.2015.2479586.

4.1 MOUNT Routing Algorithm

MOUNT stands for **M**atched **O**rdering in **UN**idirectional **T**rees and it is a deadlock-free routing algorithm designed to maximally utilize unidirectional links. uDIREC deploys MOUNT on each fault manifestation, to quickly discover reliable routes between the still-connected nodes. The MOUNT routing algorithm works by constructing two separate spanning trees of unidirectional links: one for connections moving traffic away from the root node (down-tree), and the other for connections moving traffic towards the root node (up-tree). Each node is then assigned a unique identifier corresponding to each tree: identifiers increase numerically with increasing distance from (to) the root in the down-tree (up-tree), while equidistant nodes are ordered arbitrarily. This leads to a unique ordering of nodes (lower order = closer to root) in each tree. Thereafter, the *up* link is defined as the unidirectional link towards the node with the lower identifier in the up-tree and the *down* link is defined as the unidirectional link towards the node with the lower identifier in the down-tree.

Lockstep construction. Note that the two spanning trees cannot be constructed independently of each other. Because we use unidirectional links, such an approach could lead to a mismatch in the node ordering between the trees, and consequently a link could be assigned inconsistent tags: *up* and *down*. An example of such situation is shown in Fig. 3, where mismatched node orderings lead to link $R1 \rightarrow R2$ being tagged *up* in the up-tree and *down* in the down-tree. Thus, the construction of the two trees must proceed in lockstep, guaranteeing matched ordering by construction.

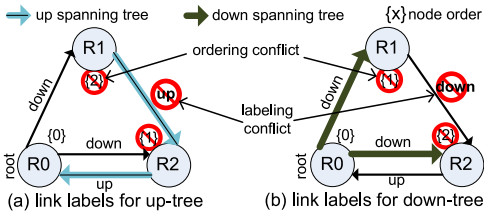


Fig. 3. The **independent construction of up-tree and down-tree** causes inconsistent labeling of link $R1 \rightarrow R2$. (a) Up-tree: link is towards the root; hence tagged *up*. (b) Down-tree: link is between nodes at the same level; hence tagged arbitrarily as *down*.

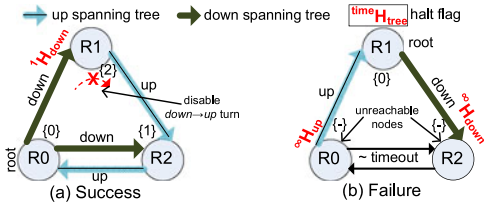


Fig. 5. **Growing the up-tree and down-tree in lockstep.** The choice of root affects connectivity. (a) Success with root *R0*: both up-tree and down-tree connect all nodes with consistent labeling. (b) Failure with root *R1*: up-tree (down-tree) halted at *R0*(*R2*).

Matched-ordering by construction. MOUNT builds the two trees using a breadth-first search, but advances the construction of the two trees in lockstep, expanding to new nodes only if a node order matching on both trees exists. To this end, each leaf node reached in the network expands the two trees to its descendants only when the node itself is reachable by both the up-tree and the down-tree. Otherwise, the up-tree (down-tree) construction is halted until both tree constructions reach that node. 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. As shown in Fig. 5a, starting from the root node (*R0*), both the up-tree and the down-tree expand to *R2* using the *bidirectional link* $R0 \leftrightarrow R2$; 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 the end of construction, both trees reach all nodes, while agreeing on node ordering; consequently the links are assigned consistent tags. Thereafter, all routes traversing a *down* link followed by an *up* link (*down* $\rightarrow$ *up* turn) are disallowed. Finally, a route search algorithm finds the minimal route(s) between each source-destination pair. The pseudo-code of the MOUNT routing algorithm is shown in Fig. 4. The proofs for MOUNT's connectivity and deadlock-freedom characteristics are detailed in Appendix B, available in the online supplemental material.

Root node selection. The structure of both trees greatly depends on the root node selection. However, as shown in Fig. 5, this aspect may also affect the connectivity characteristics of the network. In that example, if instead of *R0* (Fig. 5a), *R1* (Fig. 5b) is chosen as root, MOUNT is unable to find deadlock-free routes to any other node in the network. With *R1* as root in Fig. 5b, 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 the down-tree (up-tree) never reaches *R0* (*R2*) in this configuration. Therefore, optimal root selection can improve the connectivity characteristics of the network when using the MOUNT routing algorithm. Notice this is in contrast of networks with bidirectional links (traditional up*/down*), where connectivity within a subnetwork is independent of the choice of the root node.

```
ROOT = pick_root()
newly_reached = up-tree = down-tree = ROOT;
/*Begin up-tree and down-tree construction*/
do:
| for(NODE in newly_reached):
| | up-tree += nodes_with_link_to(NODE)
| | down-tree += nodes_with_link_from(NODE)
| newly_reached = new_overlap(up-tree,down-tree)
while(newly_reached != NULL)
disable_unreached_nodes()
order = order_nodes_reachable_by_both_trees()
apply_down_up_turn_restrictions(order)
find_minimal_routes_with_turn_restrictions()
```

Fig. 4. **MOUNT routing algorithm** to determine deadlock-free routes in networks with unidirectional links. To guarantee a matched node ordering, nodes expand the trees to their neighbors only if both up-tree and down-tree have reached them. The resulting matched-ordering governs the turn restrictions, and the evaluated minimal routes adhere to these turn restrictions.

5 Reconfiguration

Reconfiguration in uDIREC is invoked upon a permanent failure and it implements the MOUNT routing scheme. Remember from Fig. 5 that the structure of the constructed network depends on the choice of root node, and therefore, the optimal root node is selected by considering them all. During reconfiguration, first, each node in turn (starting from the node detecting the failure), is appointed as the temporary root node and the number of nodes it can connect is calculated. The optimal root-selection is finalized when one of the following two conditions occur: (i) a root node that provides deadlock-free connectivity among all nodes is found; or (ii) all nodes have been considered as root node. The pseudo-code of our reconfiguration algorithm is shown in Fig. 6. The rest of this section describes and compares two versions of the reconfiguration algorithm: i) a centralized software implementation and ii) a distributed hardware implementation.

5.1 Software-Based Reconfiguration

We first proposed the software solution in [15] as a low-cost and flexible way to implement reconfiguration. In this paper, we provide a short summary of that approach to be able to contrast against a more robust hardware solution, which is the subject of this work. For the software implementation, we designate any one node in the multi-core system as the "supervisor". This node performs the reconfiguration in software. The fault diagnosis scheme we utilize already stores the topology information in a software-maintained scoreboard at the supervisor node. In this manner, we avoid hardware overhead incurred by conventional software-based reconfiguration solutions [38], [39] to reliably collect the topology information at a central node. Upon a new fault detection, the supervisor node transmits

```
/* Root Selection by connectivity evaluation */
ROOT_win = -1; max_connectivity = 0
for (ROOT in all_nodes):
| connectivity = eval_uDIREC_connectivity(ROOT)
| if (connectivity == num_nodes):
| | ROOT_win = ROOT; break;
| if (connectivity > max_connectivity):
| | ROOT_win = ROOT;
| | max_connectivity = connectivity
/* Route Construction for winner root */
apply_MOUNT_routing_algo(ROOT_win)
```

Fig. 6. **uDIREC's reconfiguration algorithm.** All nodes are evaluated as root, and the root that provides maximum connectivity is chosen to build the new network. Within each root trial, the MOUNT routing algorithm determines the deadlock-free routes.

a reserved message to all routers/nodes in the system, informing them about recovery initiation. At the reception of this message, all routers suspend their current operation and wait for routing function updates from the supervisor, while the nodes stop new packet injections. In the meantime, the supervisor computes the optimal root node and deadlock-free routes for the surviving topology in software, using the MOUNT routing algorithm described in the previous section. Finally, our software-based reconfiguration algorithm conveys the computed routing tables to all routers, following which the supervisor node broadcasts a trigger message to resume normal operation.

The software solution drastically reduce the overhead of distributing the newly computed routing tables by noting that permanent faults are rare occurrences and all reconfiguration-specific transmissions can be done serially over a single wire. To this end, our software solution leverages a lightweight routing table distribution network that is implemented as 2-bit wide unidirectional ring consisting of one control and one data wire, with snooping controllers at each hop. This lightweight network trades-off reconfiguration latency for area and power savings. The wiring overhead of this network is a mere 0.34 percent over our baseline mesh NoC.

5.2 Hardware-Based Reconfiguration

During reconfiguration, the root node that maximizes connectivity is identified via exhaustive search. This forms the first phase of our reconfiguration algorithm, which we call the (root) *selection* phase. At the end of the *selection* phase, each node is aware of the winner-root, i.e., the node that could connect the maximum number of nodes when chosen as root. The steps of the *selection* phase are shown graphically in Fig. 7a and discussed in the next section. Following *selection*, the winner-root deploys the MOUNT routing algorithm to connect the network using deadlock free routes: routes to each node are explored in turn, starting from finding the routes to the winner-root. We call this phase the *construction* phase, and its steps are outlined in Fig. 7b and Section 5.2.2. Our algorithm is fully distributed, i.e., given only local information, state-machines at each router run in lockstep to collectively reconfigure the network. All nodes locally compute/maintain just enough information to orchestrate the reconfiguration.

To enable distributed reconfiguration, each node broadcasts two 1-bit flags to all other nodes in a co-ordinated fashion. The first 1-bit 'notification' flag is broadcasted to notify

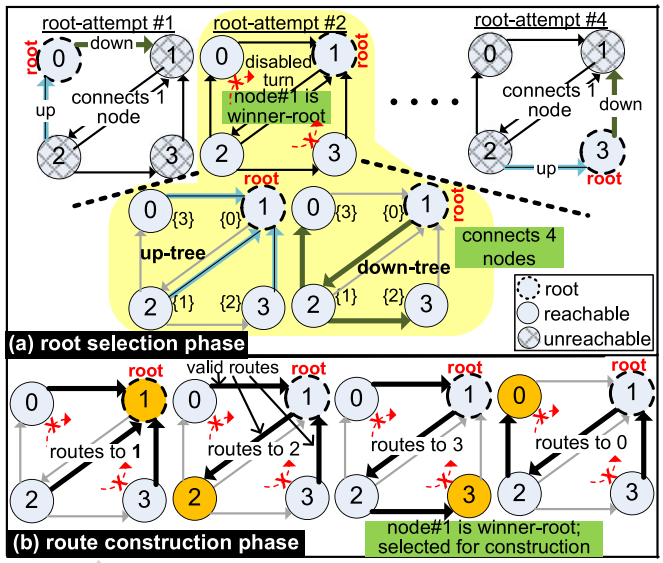


Fig. 7. **uDIREC's hardware-based reconfiguration algorithm.** a) *Root selection phase*: all nodes are tried as root and the root that provides maximum connectivity is chosen to build the new network (node#1 in the figure). Within each root trial, both up-tree and down-tree are constructed using the MOUNT routing algorithm to assess the number of connected nodes. b) *Route construction phase*: after disabling deadlock-causing turns (2→0→1 and 2→3→1 in the figure) corresponding to the winner-root, valid routes to each node are discovered.

all nodes about recovery initiation, whereas the second 1-bit 'connection' flag is broadcasted to discover the underlying topology and build deadlock-free routes.

5.2.1 Selection Phase

The selection phase is partitioned into epochs, one for each choice of root node (N, in an N-nodes network). Each epoch is further partitioned into broadcasts by each node (N in total), starting from the node serving as root during the current epoch. Remember, the goal of a *selection* epoch is to evaluate the connectivity of all choices of root. Therefore, within one *selection* epoch, all nodes should try to discover routes to all other reachable nodes, so that the connectivity information is available locally at each node. To this end, within a *selection* epoch, each node broadcasts in turn, letting other nodes know the route(s) (if it exists) to reach it.

After the completion of one *selection* epoch, the next one is initiated by a new root node by broadcasting the 1-bit 'notification' and 'connection' flags to all other nodes. A node performs different operations upon receiving different flags (notification or connection), as discussed in Section 5.2.3. Fig. 8 summarizes the actions performed during each *selection* epoch. This figure also clarifies the timing of each epoch and all broadcasts within one epoch.

Since each broadcast completes within 2*N cycles (as explained later), one *selection* epoch deterministically completes in $2*N^2$ cycles (broadcasts by N nodes). All nodes automatically switch to the *construction* phase after the last (N^{th}) *selection* epoch. Therefore, the *selection* phase all together takes a total of $2*N^3$ cycles (N epochs, containing N broadcasts each, and 2N cycles for each broadcast). However, if any of the root nodes is found to connect all nodes in the network, the system terminates the reconfiguration process (ending *selection* and skipping *construction*). Thus the minimum amount of time spent in selection is $2*N^2$,

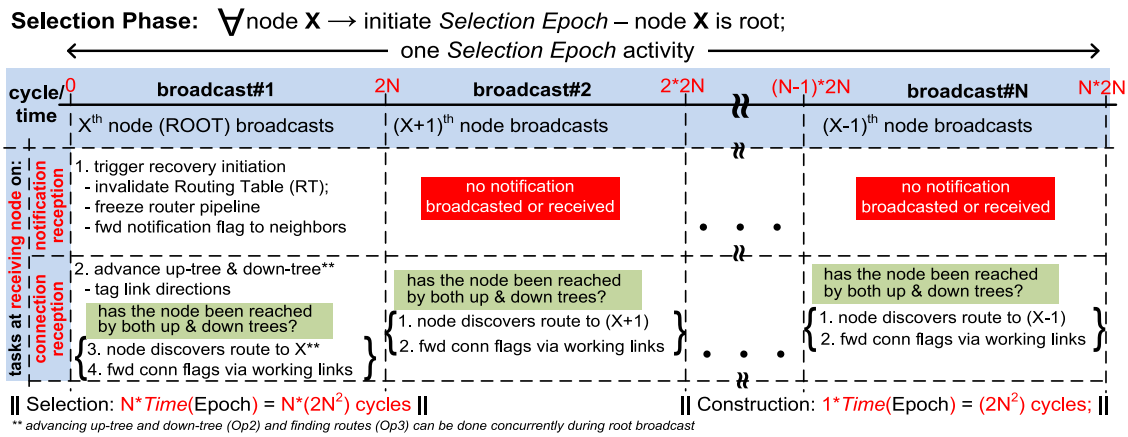


Fig. 8. **Epoch activity**. Both selection and construction phases are partitioned into epochs, one for each choice of root. Each epoch is further partitioned into broadcasts by all nodes, starting from the node serving as root during the current epoch. Different actions are performed during 'notification' and 'connection' flag reception, as well as during root and non-root broadcasts. Each epoch completes in $2N^2$ cycles.

corresponding to the case when the first *selection* epoch leads to full connectivity.

5.2.2 Construction Phase

The *construction* phase activity is similar to that of the *selection* phase, except that only the winner-root constructs the new deadlock-free routes. All nodes are independently aware of the end of the *selection* phase and at the start of the *construction* phase, the winner-root tries to initiate the construction of the network, with itself as root. The winner-root starts by broadcasting the notification and connection flags, similarly to a *selection* epoch. Note that it takes exactly N broadcasts to complete the *construction phase*.

5.2.3 Reconfiguration via Flag Broadcasts

The 'notification' flag is broadcasted only during the root broadcast at the beginning of each epoch. It is **not** broadcasted in the successive N-1 broadcasts by non-root nodes within each epoch. If a node receives the notification flag while in "normal" state, then it invalidates its routing paths, freezes the router pipeline and sets the router's state to "selecting". Additionally, upon reception, the node recognizes the beginning of a new epoch and sets a local epoch-status register (ESR). The node will automatically reset the ESR after $2*N^2$ cycles, indicating the end of the current epoch. Finally, the notification flag is forwarded to all port (s) from which the flag was not previously received. Note that notification flags are forwarded across all ports, irrespective of the condition of the corresponding network link (failed or working). The timeline on the top-half of Fig. 8 summarizes the actions taken by a node upon reception of the notification flag.

The 'connection' flag is used to build up and down trees. Specifically, upon receiving a connection flag, a node performs the following operations (bottom-half of Fig. 8):

1) *Tagging link directions*. This is performed only during the root broadcast at the start of each epoch. Distinct *to* (broadcasting node) and *from* (broadcasting node) connection flags are used to build the up-tree and the down-tree, respectively (Section 4.1). Note that we use a single wire to forward these two types of connection flags in a time-multiplexed fashion and each broadcast is bound to reach all connected nodes within 2*N cycles. Therefore, we allocate a fixed 2*N cycle time window for broadcast by each node. Unidirectional links towards a node that is closer to the current root are *up* links, while links towards a node that is farther from the root are *down* links. Naturally, the outgoing link corresponding to the port that receives the *to* connection flag, is tagged *up* (Fig. 9). Similarly, the outgoing link corresponding to the port that forwards the *from* connection flag is tagged *down*. Ties between nodes at equal distance from the root are broken arbitrarily, as discussed in Section 4.
2) *Routing table update*. During a broadcast by any node (including root), the broadcasting node informs the other nodes how it can be reached. The purpose of the non-root nodes' broadcast in each epoch is that of updating the routing table. A *to* connection flag reception at a port implies that the reverse path of flag broadcast can be followed to reach the broadcasting node, and hence the port of reception is recorded in the node's routing table. Since two-way connectivity is a must, both *to* and *from* flag receptions (up-tree and down-tree) are required at a node during root node broadcast for it to record the routes to any other node. Moreover, broadcasts should spread only through working links and enabled turns. This means that a *to* connection flag is forwarded only if there is a working unidirectional link in the reverse direction of forwarding, whereas a *from* connection flag is forwarded only if there is working unidirectional link in the direction of forwarding. Also, each node during connection flag forwarding only forwards flags through enabled turns. A connectivity counter (CC), maintained locally at each node, is incremented upon discovery of a route to a new node. It is used to evaluate the number of nodes connected by each choice of root.
3) *Flag forwarding*. Each node initiates the broadcast by forwarding the *to* connection flag, followed by forwarding the *from* connection flag in the next cycle. The nodes receiving the flags, forward them on subsequent cycles, only via port(s) that did not receive a flag earlier. Both connection flags are forwarded on a

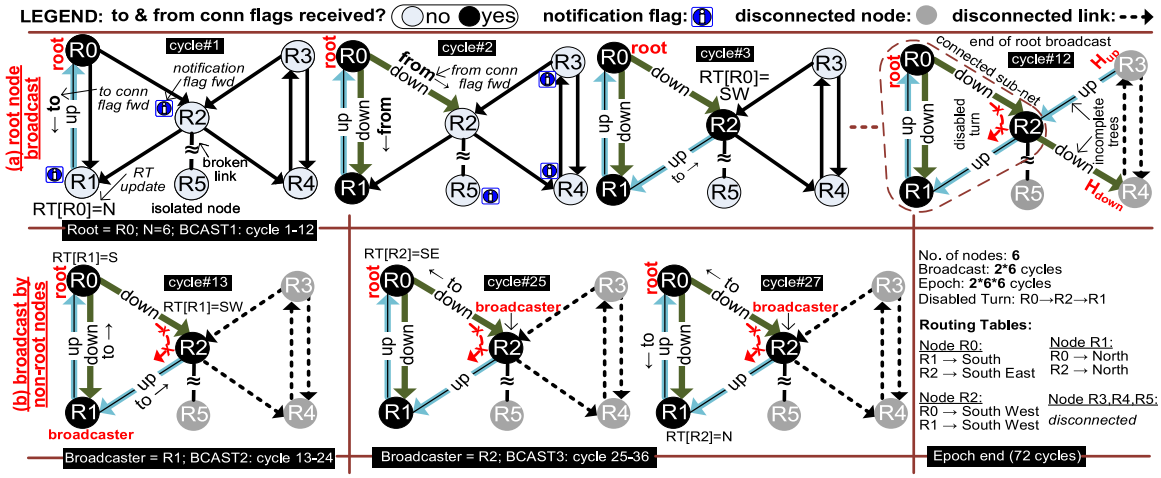


Fig. 9. **Reconfiguration example** showing (a) root broadcast: *R0*, (b) non-root node broadcast: *R1*, *R2*; and routing table at end of epoch.

single wire and thus, each forwarding event is completed in two cycles. All nodes decipher the type of flag received (*to* or *from*), based on whether the current cycle number is odd or even. Additionally, flags are forwarded only through enabled turns: *to*connection flags received from ports with outgoing link marked as *up* (*up* ports) are not forwarded to other *up* ports, because this would enable a *down* → *up* path in the reverse direction. Each node forwards the *to* and *from* flags only after receiving them both during the root broadcast. This is to enforce the constraint of growing the *up* and *down* spanning tree in lockstep, as discussed in Section 4.1.

5.2.4 Orchestration of Epochs

At the end of an epoch, each node performs a series of checks and status updates to orchestrate the reconfiguration. Each node pro-actively tries to initiate a new epoch (with itself as root) after the end of the current epoch, if the node has not been tried as root before. A node temporarily stops attempting to be root if it receives a notification flag indicating that some other node has already initiated an epoch. All nodes, if connected, resume normal operation at the end of the *selection* epoch. A connected network can be easily detected by checking the contents of the routing table. As our resilient routing algorithm only provides two-way connectivity, either all nodes have full routing tables or all nodes have incomplete routing tables. This implies that all nodes in the network are connected only if the connectivity counter is equal to the number of nodes in the network (CC = N), at the end of a *selection* epoch. Remember that the connectivity counter counts the number of connected nodes for each choice of root in the network. Otherwise, if the routing tables are incomplete (CC ≠ N), the value of CC is compared to the maximum CC achieved during all previous *selection* epochs (different roots). If the current CC value is greater than the maximum CC value from previous trials, then the corresponding $<$ROOT, CC$>$ value pair is stored locally at each node in-place of the previous maximum $<$ROOT, CC$>$ pair. The routes discovered during the *selection* epoch are erased from the routing tables. A separate counter, named the epoch counter (EC) is maintained to count the number of completed epochs. The *construction* phase is automatically initiated when all root trials are completed (EC = N).

5.2.5 Timing

We are able to use low-cost single bit flags for reconfiguration by relying on atomic broadcasts based on cycle numbers. Each node is assigned interleaved time windows of 2*N cycles (bound on one broadcast) each, during which the remaining nodes are prevented from broadcasting. Note that broadcasts by a single node are guaranteed to be completed in 2*N cycles. **Therefore, the broadcasting node's ID can be inferred by the cycle number at which the broadcasted flag is received.** The time slots of all nodes in the network are interleaved to provide equal opportunity to all nodes to initiate a broadcast. In this way, each node waits for its slot to broadcast, thus avoiding overlap. We could also improve the timing of the reconfiguration by making global broadcast slots programmable and reducing their number and duration as nodes are lost. That being said, permanent faults, even when up to tens or hundreds, are not frequent enough for reconfiguration to affect overall performance.

The worst case reconfiguration time in hardware-based reconfiguration translates to only 1ms overhead for 1 GHz 64-node NoC. Our current implementation is based on the assumption of a synchronous clock domain across the communication infrastructure. In case of asynchronous clock domains, node IDs will have to be piggybacked with the broadcasted flags and a token-based arbitration mechanism can be used to prevent overlapping of broadcasts.

5.2.6 Reconfiguration Example

Fig. 9 shows the activity during one *selection* epoch in a six-node network. For this epoch, *R0* is the root and each broadcast takes 12 cycles to complete (2*6-nodes). During root broadcast (Fig. 9a), both notification and connection flags are forwarded. Also, link directions are tagged and routes to the root are updated at each node. Finally, the link tags determined during the root broadcast are used to disable deadlock-causing turns. For instance, during cycle#1, the notification flag is forwarded to nodes *R1* and *R2*, and the *to* connection flag is forwarded to node *R1*. At the same time, link *R1*→*R0* is marked as *up* and the routing table at node *R1* is updated. During cycle#2, the notification flag is further

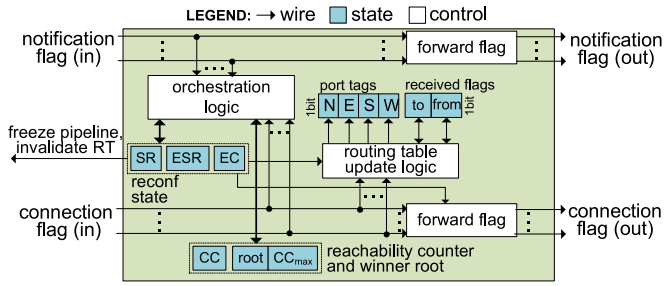


Fig. 10. **Hardware additions at each router for the implementation of uDIREC's hardware-based reconfiguration**. Notification and connection wires are snooped by *orchestration logic* and *routing table update logic* at each router. Further, broadcasted flags are *forwarded* to neighboring routers. The reconfiguration status registers (*SR*, *ESR*, *EC*) are updated based on received flags, while ports are tagged and routing table is filled in accordance to the MOUNT routing algorithm. Finally, the orchestration logic computes the winner root by comparing the connectivity counter values.

forwarded to all remaining nodes irrespective of the broken connections and the *from* connection flag is forwarded to nodes *R1* and *R2* (the corresponding links marked *down*). At the end of the root node broadcast (cycle#12), only nodes *R0, R1* and *R2* are connected and the turn *R0→R2→R1* is disabled to avoid deadlock. Node *R3* (*R4*) is disconnected as it is unreachable by the down-tree (up-tree).

During the broadcasts by other nodes (Fig. 9b), only the connection flags are forwarded. At cycle#13 node *R1* is broadcasting, and it forwards the *to* connection flag to nodes *R0* and *R2*, which update their routing tables. In a similar manner, node *R2* broadcasts during cycle#25-36 and lets other nodes find routes to itself. At the end of the epoch (6-broadcasts*2*6-nodes cycles), routing tables at each node store valid routes to the other connected nodes.

5.2.7 Hardware Additions

uDIREC-specific hardware is lightweight and it has only minimal connections to the router pipeline. Fig. 10 shows the schematic of uDIREC's hardware additions. The hardware additions at each router node can be classified into:

Wires: 2 1-bit wires per port per router are required for connection and notification flags. These wires replicate the baseline topology, i.e., a mesh in our case.

State elements: A 2-bit status register (SR) records the operational state ("selection", "construction" or "normal") and a 1-bit epoch-status register records the end of a *selection* or *construction* epoch. To implement MOUNT, we require one 1-bit register per port to store the port's direction (*up* or *down*) and a 2-bit register per node to keep track of the type of connection flags received during root broadcast (*to* and *from*). In addition, two $log_2 N$ bit counters, epoch counter and connectivity counter, are required to keep track of the reconfiguration process. Finally, the maximum value of CC and corresponding root node is stored in two separate $log_2 N$ bit registers.

Control logic: Notification and connection flags are treated differently, and hence separate logic is required to implement the operations that are performed upon receiving each flag. The logic corresponding to notification flags implements: status register update, epoch-status register update and flag forwarding. Upon reception of the connection flags, logic is required to tag link directions, to fill the routing tables and to forward flags across enabled links/turns. Finally, EC, CC and ESR updates are performed at the end of each epoch to orchestrate the *selection* phase. The details of the tasks performed are provided in Section 5.2.1 and illustrated schematically in Fig. 8.

6 Hardware vs. Software Implementation

We proposed two versions of our reconfiguration algorithm: i) a distributed hardware-based implementation, and ii) a software-based implementation. The hardware implementation is truly-distributed in the sense that all network nodes execute the same state machine, and therefore, no one node is preferred over others and there is no single-point-of-failure. In contrast, the software-based reconfiguration collects the topology information at a central node, and executes the reconfiguration algorithm in software. As our diagnosis scheme is also implemented in software, the software implementation can be tightly integrated with our diagnosis scheme, as was discussed earlier in Section 5.1. Naturally, the software implementation has a lower complexity and is more easily extensible to sophisticated root search algorithms. However, the hardware implementation leads to a faster reconfiguration (~1 million NoC cycles for a 64 node CMP in the worst case), which could be important for highly critical or fault-prone systems. In contrast, the software-based solution could take up to a few seconds complete. In the rest of this section, we discuss important reconfiguration parameters, and how the choice of hardware vs. software implementation affects them.

Distributed Implementation. Our hardware solution is fully-distributed as all nodes run the same state-machine. Nodes rely on timed broadcasts for synchronization during the reconfiguration operation. As a result of the distributed implementation, there is no single point of failure. In contrast, the software-based implementation collects the topology information at a centralized location, therefore, it is not truly distributed. It also means that, in certain situations, the software-based scheme can have a single point of failure, for instance, if a fault leads to the isolation of the "supervising" node. Therefore, the software-based implementation requires duplication of the centralized orchestration entity to provide greater robustness.

On the flip side, the hardware additions themselves can possibly fail. However, all robustness schemes are vulnerable when the reliability hardware itself malfunctions. Therefore, uDIREC's hardware was designed to minimize such events: i) dedicated wires are used for reconfiguration, eliminating the chance of wearout-induced permanent failures during normal operation, and ii) a small silicon footprint allows uDIREC's hardware to be protected using triple modular redundancy without increasing the total overheads significantly. Finally, uDIREC protects itself against faults by inferring them as faults in the corresponding network links. In other words, if a reconfiguration wire between two state machines at adjacent routers is broken, then the corresponding link between these routers will be tagged faulty by both state machines. The reconfiguration will proceed exactly as if the network link is faulty, which might lead to dropped nodes.

Complexity and Integration. Even though the software-based implementation is slower and provides a single-point-of-failure under rare scenarios, it is easier to implement and modify. Particularly, it can be fully integrated with our software-based diagnosis scheme, leading to a low overhead implementation as discussed in Section 5.1. In addition, the software-based reconfiguration implementation is extensible to more sophisticated root search algorithms. In contrast, the hardware-based reconfiguration implementation involves intricate operations such as timed broadcasts and synchronization among nodes, which are adopted to keep the area overhead of implementation at a minimum at the cost of considerable additional complexity.

Reconfiguration Duration. As discussed earlier, the hardware-based reconfiguration is faster compared to the software-based reconfiguration. Therefore, for highly critical and fault-prone systems, the hardware-based reconfiguration is recommended. In either implementation, we perform an exhaustive search of the optimal root, which could be further optimized for performance. However, in our solution we traded reconfiguration time for simplicity of the algorithm. This is particularly important for the hardware implementation, which is not extensible to sophisticated search algorithms. Even in the software-based reconfiguration, the exhaustive search leads to a negligible overall performance overhead for most systems. This is because permanent faults (even when up to tens or hundreds) are not frequent enough for reconfiguration duration to affect overall performance. Tree-based routing algorithms can be efficiently implemented in software, and typically take only hundreds of milliseconds to complete ($\sim$170 ms [38]). Even though we run multiple iterations of the tree-based MOUNT routing algorithm, we expect the reconfiguration overhead to be within a few seconds at worst. Assuming an aggressive life-span of two years for high-end servers and consumer electronics, and 150 NoC faults (Section 1) in the worst case, a NoC would suffer one fault every five days. Therefore, an overhead of a few seconds per fault manifestation is negligible.

Reconfiguration-Induced Deadlock. Either version of our reconfiguration algorithm can cause routing deadlocks even if both the initial (before fault manifestation) and final (after reconfiguration) routing functions are independently deadlock-free [45], [46]. We avoid such deadlocks by identifying the packets that request an illegal turn according to the updated (after reconfiguration) routing function. These packets are then ejected to the network interface of the router in which they are buffered at the time of reconfiguration. After reconfiguration, these packets are re-injected into the network upon buffer availability. Other state-of-the-art reconfiguration techniques [7], [8] utilize a similar technique to overcome reconfiguration-induced deadlocks.

Early Diagnosis. Few faults in a routers' datapath can be corrected by the end-to-end ECC embedded in our diagnostic method. This presents the opportunity to keep using the network (for some time) even after the fault has been diagnosed, as the initial few faults in the router datapath are within the correction capacity of the ECC. This pre-emptive approach enables us to salvage the processor and memory state of the about-to-be-disconnected nodes while the network is still connected. The operating system will then intervene to restart the process from a correct state, such that no tasks are assigned to disconnected nodes. Traditionally, computer architects have relied on checkpointing support [47] for salvaging processor and memory state, while a recent research proposal [48] adds emergency links to this end. Using our technique, it is possible to greatly simplify this additional reliability-specific hardware. uDIREC does not guarantee the integrity of packets that are traversing the network after the fault manifestation and before fault detection, but it relies instead on orthogonal recovery schemes [17], [49], [50] for that. However, the property of early diagnosis can greatly reduce the likelihood of fatal data corruptions and reduce the reliance on such recovery schemes.

Optimal Root and Tree. The choice of root node also affects the network latency and throughput characteristics [51]. In addition, tree-based routing algorithms' performance is sensitive to the way trees are grown (breadth-first versus depth-first), and the order in which nodes are numbered [51]. Further, the surviving set of on-chip functionalities may also differ with different root selection and tree growth schemes. However, the corresponding analysis is beyond the scope of this work and we do not consider these metrics in our root selection or tree-building process. uDIREC chooses the optimal node solely on the basis of number of connected nodes, breaking ties by comparing statically-assigned node IDs, while the trees are built in a breadth-first fashion.

7 Experimental Results

We evaluated uDIREC by modeling a NoC system in a cycle-accurate C++ simulator [52]. The baseline system is an 8×8 mesh network with generic four-stage pipelined (including link traversal) routers and with two message classes, one VC per message class. Each input channel is 64-bits wide and each VC buffer is 8-entry deep. In addition, the NoC is augmented with uDIREC reconfiguration capabilities. For comparison, we also implemented Ariadne [7], which outperformed all previous on-chip reconfiguration solutions, including the state-of-the-art Vicis [6] and Immunet [8]. Ariadne [7] reports 40 percent latency improvement over Immunet, which falls back to a high-latency ring to provide connectivity, and 140 percent improvement over Vicis, which occasionally deadlocks. Since both Ariadne and Immunet guarantee connectivity if routes (using only bidirectional links) between pairs of nodes survive, they show identical packet delivery rates. They also deliver a higher fraction of packets compared to Vicis, especially at high number of faults when Vicis tends to deadlock. Our results demonstrate uDIREC's substantial improvements over Ariadne, and thus uDIREC's improvements over Vicis and Immunet are expected to be even more impressive. Note that Ariadne assumed a perfect diagnosis mechanism, and therefore, for fair comparison, we have paired it with the diagnosis scheme of [43] in our implementation.

Our framework is analyzed with two types of workloads (Table 2a): synthetic uniform random traffic, as well as applications from the PARSEC suite [53]. PARSEC traces are obtained from the Wisconsin Multifacet GEMS simulator [54] modeling a fault-free network and configured as detailed in Table 2b. After fault injections, we ignore

TABLE 2
(a) Simulation Inputs (b) GEMS configuration

(a)

traffic	uniform PARSEC
packet	1flit (control) 5flits (data)
simulation	1M cycles
warm-up	10K cycles

(b)

processor	in-order SPARC
coherence	MOESI
L1 cache	Private: 32KB/node ways:2 latency:3
L2 cache	Shared: 1MB/node ways:16 latency:15

messages originating from and destined to the disconnected nodes: this could lead to some evaluation inaccuracies for parallel collaborating benchmarks running on a partitioned multi-core. However, the traces are intended to subject the faulty NoC to the realistic burstiness of application traffic, and provide a simple and intuitive comparison. The metrics provide valuable insights considering that a particular fault manifestation in uDIREC and prior work(s) could lead to vastly different configurations in terms of number/location/functionality of working cores/IPs.

Algorithmic complexity. Our diagnosis scheme is based on analyzing routing information of faulty packets. Therefore, its complexity is proportional to the number of routes, i.e., $\mathcal{O}(N^2)$ in a N-node network. In MOUNT routing algorithm, each node explores the routes to all the other nodes, and therefore its complexity is again $\mathcal{O}(N^2)$. Finally, our reconfiguration algorithm invokes the routing algorithm one time for each choice of root node, leading to a timing complexity of $\mathcal{O}(N^3)$.

uDIREC Variants. We implemented three uDIREC variants: i) uDIREC_sw, the software-based reconfiguration algorithm from Section 5.1, ii) uDIREC_hw, the hardware-based reconfiguration solution from Section 5.2 and iii) uDIREC_nv, a naïve variant of the hardware-based solution. uDIREC_hw utilizes dedicated notification flags to notify all nodes of recovery initiation, and a separate root *selection* phase to find the optimal-root. uDIREC_nv provides a simpler design alternative that has no support for discovering the largest sub-network and simply fixes the node detecting the fault to be the root. uDIREC_nv provides faster reconfiguration because of the absence of the *selection* phase), and lower area and power as it does not require notification flags. However, uDIREC_nv leads to sub-optimal reconfiguration in certain scenarios.

Area and power results. uDIREC_sw incurs the least area and power overhead of all the uDIREC variants. uDIREC_sw introduces only 0.34 percent wiring area overhead and a negligible logic overhead due to the routing table distribution network. All other aspects of uDIREC_sw are implemented in software, and therefore, they do not incur any area overhead. We implemented uDIREC_hw and uDIREC_nv in Verilog and synthesized the designs using Synopsys' Design Compiler targeting the Artisan 45 nm library. uDIREC_hw's logic costs only 1.88 percent over the baseline router (area = 44,800 μm^2), whereas uDIREC_nv's logic overhead is only 1.35 percent. In addition, uDIREC_hw introduces a wiring area overhead of 1.36 percent, while uDIREC_nv incurs a wiring cost of only 0.68 percent. All the wire lengths in our analysis are calculated assuming the chip and tile dimensions from Intel's 45 nm SCC chip [55]. The low-area overhead of our hardware-based schemes can be attributed to two factors: i) our schemes introduce only 1-2 additional wires per router port, compared to the already existing 64 wires for data transmission, and ii) the area of on-chip networks is dominated by packet buffers, which are not required in our schemes as received flags are forwarded immediately without storage. The area overhead of uDIREC_hw is only slightly more than Ariadne (1.1 percent overhead) and Vicis (0.83 percent overhead), while it is considerably less than Immunet, which reserves a separate virtual network with large input buffers for deadlock freedom. uDIREC's reconfiguration components (optionally power-gated during normal operation) introduce negligible power overhead. For power analysis, we modeled uDIREC as a 2-bit wide mesh network at 45 nm in Orion2.0 [56] and assuming an activity factor of 0.3. uDIREC_hw consumes 6.2 mW power on average (6.8 mW peak), which is only 2.2 percent of the full router power of 280 mW (320 mW peak). Wiring with 62 percent contribution dominates uDIREC's power consumption.

Fault injection. Our architectural-level fault injection technique randomly injects gate-level faults in network components with a uniform spatial distribution over their silicon area. Each fault location is then analyzed for mapping to one or more links, modeling the fault diagnosis scheme proposed in [43]. The links that are marked as affected by the fault are bypassed using the route-reconfiguration schemes we evaluate. Further, our baseline NoC is not equipped with any reliability feature beside uDIREC. In our experiments, we injected a varying number of permanent faults (0-60 transistor failures) into the NoC infrastructure, and analyzed uDIREC's reliability and performance impact. For each number of transistor failures, the experiment was repeated 1,000 times with different fault spatial locations, selected based on a uniformly random function.

Scope of experiments. We restricted the scope of our experiments by injecting faults only in the NoC infrastructure because the system-level performance and reliability depend on a vast number of factors: fault location, fault timing, memory organization, programming model, processor and memory reliability schemes, and architecture specific characteristics. As a result, the surviving system functionality might not be directly comparable. Hence, we have provided a generalized evaluation across a wide range of faults, consisting of insightful performance (latency, throughput) and reliability (number of dropped nodes, packet delivery rate) metrics for fault-tolerant NoCs.

7.1 Reliability Evaluation

As faults accumulate, networks may become disconnected. Performance of parallel workloads running on a multi-core chip with a faulty network directly depends on the number of connected processing elements and other on-chip functionality, e.g., memory controllers, caches. 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. In this section, we study: a) the average number of dropped nodes, i.e., the nodes that are not part of the largest connected sub-network, and b) the packet delivery rate for uniform traffic, as

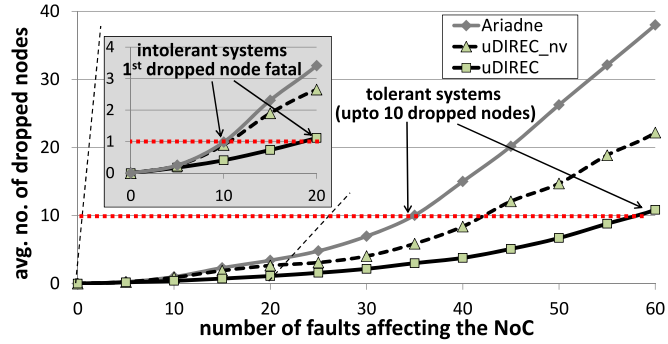


Fig. 11. **Average number of dropped nodes**. Compared to Ariadne, uDIREC drops 3× fewer nodes and approximately doubles the number of faults tolerated before the same number of nodes are dropped. uDIREC_nv drops significantly fewer nodes compared to Ariadne at the same number of faults, but more than uDIREC.

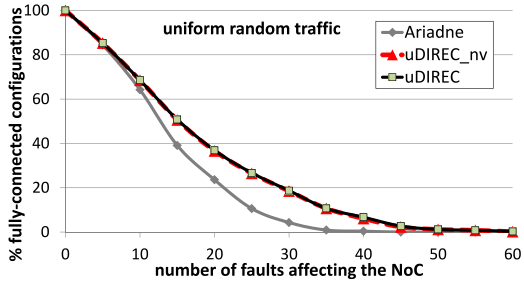


Fig. 12. **Probability of completely connected configurations** decreases rapidly with increasing faults. uDIREC provides fully connected networks with a higher probability.

faults accumulate in the NoC. The number of dropped nodes indicate the loss of on-chip processing elements and vital functionality, while the packet delivery rate captures the number of packets delivered over the number of packets generated. Both reflect the reliability of the network: a more reliable network will drop fewer nodes and will deliver a higher percentage of packets.

Both uDIREC_sw and uDIREC_hw leverage the same underlying diagnosis, routing and reconfiguration algorithms. As a result, the healthy network instances produced after the application of uDIREC_sw and uDIREC_hw to faulty networks are indistinguishable. Our reliability metrics assess the quantity of the resources kept functional after reconfiguration, and therefore both schemes lead to the same evaluation. Hence, we present the reliability results for both schemes under one tag, i.e., uDIREC.

It can be noted in Fig. 11 that Ariadne consistently drops over three times more nodes than uDIREC. Even with just a few faults (5-20), uDIREC shows substantial improvement over Ariadne, dropping one node against Ariadne's 3 at 20 faults, as shown in the zoomed section of Fig. 11. This showcases uDIREC's advantage across a wide range of reliability requirements. For highly intolerant systems, where the average of one node loss is considered fatal, uDIREC can tolerate an average of 20 faults, as compared to Ariadne's 10 faults, as shown in the zoomed section of Fig. 11. Fig. 12 shows the probability of a completely connected network (when network connectivity is equivalent to a fault-free system) with a varying number of transistor faults. As can be noted from the figure, uDIREC consistently leads to more than 3× completely connected networks between 25 and 50 faults, when compared to Ariadne, beyond which the probability of still having a fully connected network is near-zero. Also, both uDIREC and uDIREC_nv show an almost similar probability of a completely connected network (overlapped lines in Fig. 12). For more tolerant systems, for instance, those that can operate with an average loss of up to 10 nodes, uDIREC can keep the system functional even when experiencing up to 60 faults on average, as compared to 35 faults in Ariadne's case. uDIREC_nv, a naïve variant of uDIREC, drops more nodes than uDIREC but still shows substantial improvement over Ariadne. With lower complexity and shorter reconfiguration, uDIREC_nv is a good trade-off for commodity systems.

A partitioned network is unable to deliver packets originating in a sub-network different from the destination sub-network. Analyzing Fig. 13, both uDIREC and Ariadne deliver the majority (or all) of the packets up to 10 faults. Beyond 15 faults, Ariadne starts partitioning into multiple sub-networks, and hence its delivery rate drops substantially below that of uDIREC. At 25 faults, uDIREC delivers 7 percent more packets than Ariadne, and the gain goes up to ∼3× at 60 faults. uDIREC's ability to deliver a large fraction of packets even at a high number of faults makes it an excellent solution for fault-ridden NoCs. Again, uDIREC_nv delivers fewer packets than uDIREC, but considerably more packets than Ariadne. The number of dropped nodes in Immunet and its packet delivery rate, are both identical to Ariadne [7], whereas Vicis delivers a lower fraction of packets at higher number of faults.

7.2 Reconfiguration Timing

As mentioned earlier, uDIREC_hw has a separate *selection* and *construction* phase, while uDIREC_nv and Ariadne fix the node detecting the fault as the root, and proceed directly to construct the network. Thus, both uDIREC_nv and Ariadne complete in a fixed time window of one epoch (Section 5.2), while uDIREC_hw takes longer. Note that, one uDIREC_hw and uDIREC_nv epoch is $2N^2$ (∼8K for 8×8 network) cycles, whereas one Ariadne epoch is N^2 (∼4K for 8×8 network) cycles long. Fig. 14 plots the average number of cycles needed to reconfigure the network upon a fault occurrence. For uDIREC_hw, the reconfiguration time increases almost linearly up to 30 faults (∼450K cycles), after which the rate of increase drops. This behavior can be explained by analyzing Fig. 12, showing that the probability of obtaining a fully-connected network decreases rapidly up to ∼30 faults. Recall from Section 5.2.1 that reconfiguration terminates early when a root can connect all nodes (CC = N). Therefore, the reconfiguration time increases due to the initial rapid decrease in the number of fully-connected networks. Beyond 30 faults, the increase in reconfiguration time is primarily due to segregation of the network into separate sub-networks. As each sub-network requires one epoch during the *construction* phase, this increase is proportional to the increase in the number of sub-networks. Even though the reconfiguration duration is much higher compared to Ariadne, the upper bound of ∼1M cycles is a massive improvement over the software-based implementation, uDIREC_sw, which could take up to a few seconds for each reconfiguration event.

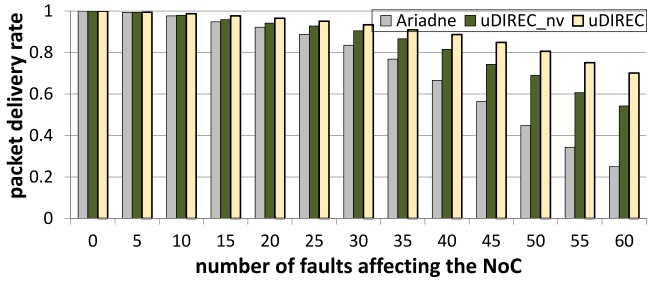


Fig. 13. **Packet delivery rate**. Higher network partitioning in Ariadne causes a steep decrease in delivery rate beyond 10 faults. In contrast, both uDIREC and uDIREC_nv degrade gracefully.

7.3 Performance After Reconfiguration

After reconfiguration, the NoC should keep functioning adequately experiencing only a graceful performance degradation. In our evaluation we report two performance metrics: average network latency and saturation throughput, after the network is affected by transistor faults. Latency and throughput measures are reported for the largest connected sub-network, assuming nodes disconnected from each other cannot work collaboratively. First, we report the average zero-load network latency, that is, the steady-state latency of a lightly loaded network (0.03 flits injected per cycle per node). It reflects the average delivery time of a network without congestion, and hence, in a sense, it indicates the average length of routes between NoC nodes. A reconfiguration technique that provides greater path diversity will have a lower zero-load latency.

uDIREC_sw and uDIREC_hw utilize the same diagnosis mechanism, and leverage the same underlying routing and reconfiguration algorithms. Therefore the performance parameters, such as latency and throughput, that are measured on network instances post reconfiguration, present no difference between uDIREC_sw and uDIREC_hw. Thus, we present the performance results for both variants together under the uDIREC label.

Analyzing Fig. 15, both uDIREC and Ariadne initially show an increase in latency because the number of paths affected increases with the number of faults, while very few nodes are disconnected from the network. Beyond approximately 30 faults, Ariadne's latency starts falling. This effect is easily understood by analyzing Fig. 11 beyond the 30-faults mark: a substantial number of nodes are dropped by Ariadne. As a result, packets now travel shorter routes to their destinations, and thus the average network latency is reduced.

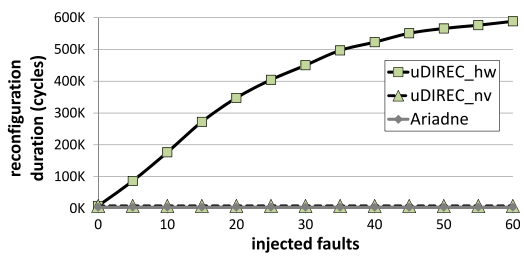


Fig. 14. **Reconfiguration duration with uDIREC's hardware implementations**. Both uDIREC_nv and Ariadne reconfigure in constant time, while uDIREC_hw's reconfiguration time increases with the number of faults, saturating beyond 40 faults.

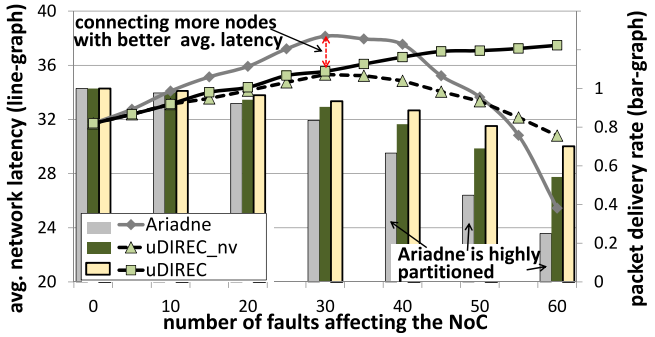
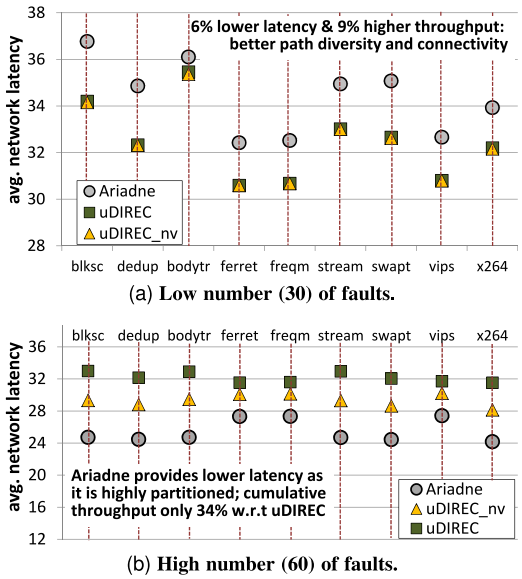


Fig. 15. **Zero-load latency**. Initially, latency degrades more gracefully for uDIREC as it provides more path diversity. Beyond 40 faults, Ariadne becomes highly partitioned and hence latency drops steeply. The packet delivery rate is much lower for Ariadne, confirming its excessive partitioning. uDIREC_nv offers lower latency compared to uDIREC, but it shows greater partitioning.

uDIREC exhibits better latency characteristics when compared to Ariadne. Initially, uDIREC degrades gracefully as it is resource-conscious and provides greater path diversity. For example, at 30 faults, uDIREC on average has 7 percent lower latency than Ariadne. However, as more faults accumulate, Ariadne drops nodes more quickly than uDIREC (Section 7.1), and because of shorter routes in partitioned networks, Ariadne's latency shows a greater decrease. The crossover between the two latency graphs is at ~40 faults, and as noted from the packet delivery rate chart copied over in Fig. 15, the difference between the delivery rate of the two techniques is large (35 percent more in uDIREC) and grows rapidly beyond that point. uDIREC would show even bigger latency improvements over Immunet and Vicis, as they show 40 and 140 percent worse latency than Ariadne, respectively [7]. Finally, uDIREC_nv exhibits lower latency compared to uDIREC throughout the fault range, but it also suffers from greater partitioning. Still, the NoC partitioning with uDIREC_nv is not as severe as in the case of Ariadne.

Similar trends are observed in simulations using PARSEC benchmark traces, as plotted in Fig. 16. uDIREC's ability to provide path diversity leads to lower latencies at few faults (Fig. 16a). At 30 faults, uDIREC shows 5.7 percent lower latency when compared to Ariadne, while delivering 8.8 percent higher cumulative throughput. At even higher faults, uDIREC loses nodes gracefully, and hence its latency decrease is not as rapid as Ariadne (Fig. 16b). Note that at 60 faults, Ariadne's cumulative throughput is only 34 percent of uDIREC.

In addition to latency, we also measured saturation throughput of the largest surviving sub-network. Saturation throughput is the measure of maximum sustained bandwidth provided by the network and it is measured in flits received at each cycle. Fig. 17 plots the packet throughput delivered by the network. uDIREC consistently delivers more packets per cycle as it uses additional unidirectional links to connect more nodes and to enable more routes. uDIREC delivers 25 percent more packets per cycle than Ariadne at 15 faults. This advantage further increases to 39 percent at 60 faults. While not performing as well as uDIREC, uDIREC_nv still has 11 percent higher throughput compared to Ariadne at 15 faults. Again, both Immunet and Vicis



(a) **Low number (30) of faults.**

(b) **High number (60) of faults.**

Fig. 16. **Average network latency with PARSEC benchmark traces**. The figure also shows cumulative throughput delivered by Ariadne (normalized against uDIREC). At few faults, uDIREC latency is lower compared to Ariadne as it provides additional path diversity, even though Ariadne delivers 8 percent fewer packets. At high number of faults, uDIREC connects significantly more nodes compared to Ariadne, with Ariadne delivering only 30 percent of uDIREC's throughput. Therefore, packet latency with uDIREC is larger (packets traverse longer routes to destinations).

can sustain a considerably lower maximum throughput compared to Ariadne [7], and hence their performance is even worse against uDIREC.

Performance, Energy and Fault-Tolerance (PEF) Metric [10]. Traditional NoC metrics, such as energy-delay product (EDP), do not capture the importance of reliability and its relation to both performance and power. To this end, [10] proposed a composite metric which unifies all three components: latency, energy, and fault-tolerance. They defined PEF as shown in Equation (1). In a fault-free network *Packet Delivery Rate* $= 1$; thus, PEF becomes equal to EDP. We assume total network energy to be proportional to the number of active routers in the network, as we use identical routers in our setup. Therefore, we estimate *Energy Per Packet* as in Equation (2). Fig. 18 shows PEF values for uDIREC variants normalized against PEF values for Ariadne. Note that a lower value of PEF is better. The relative difference in the PEF values between uDIREC variants and Ariadne monotonically increases with an increasing number of faults. At 15 faults, uDIREC (uDIREC_nv) has 24 percent (17 percent) lower PEF

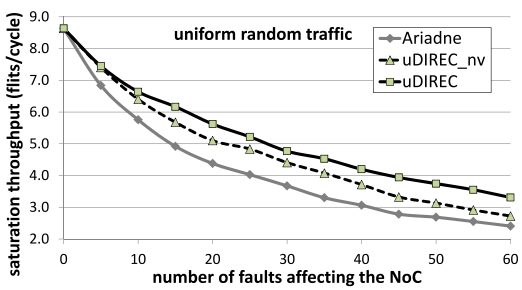


Fig. 17. **Saturation throughput**. uDIREC consistently delivers more packets per cycle as it uses additional unidirectional links to connect more nodes and enable more routes.

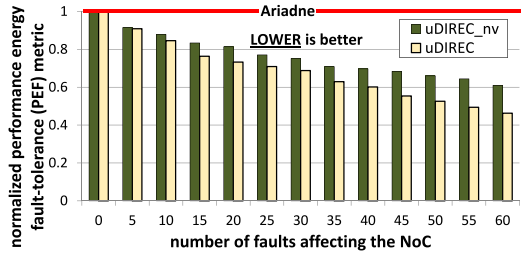


Fig. 18. **Performance-energy-fault tolerance metric [10]**. uDIREC monotonically improves with increasing faults. uDIREC (uDIREC_nv) shows 2× (1.6×) improvement at 60 faults.

than Ariadne, while showing more than 2× (1.6×) improvement at 60 faults. The reported PEF values confirm the benefits of using uDIREC across a wide range of fault rates. At few faults, the additional paths provided by uDIREC lead to reduced latency, while with more faults, uDIREC delivers a greater fraction of the packets to their intended destinations

$$PEF = \frac{(Average\ Latency) \times (Energy\ per\ Packet)}{Packet\ Delivery\ Rate} \quad (1)$$

$$Energy\ Per\ Packet \propto \frac{Number\ of\ Active\ Routers}{Packet\ Throughput} \quad (2)$$

8 Conclusions

We have presented uDIREC, a solution for the reliable operation of NoCs providing graceful performance degradation even with a large number of faults. uDIREC leverages MOUNT, a novel deadlock-free routing algorithm to maximally utilize all the working links in the NoC. Moreover, uDIREC incorporates a novel fault diagnosis and reconfiguration algorithm that places no restriction on topology, router architecture or the number and location of faults. Simulations show that for a 64-node NoC at 15 faults, uDIREC drops 68 percent fewer nodes and provides 25 percent higher bandwidth over state-of-the-art reliability solutions (diagnosis - Vicis [6]; reconfiguration - Ariadne [7]). A combined performance, energy and fault-tolerance metric, that integrates the energy-delay product with the packet delivery rate, indicates 24 percent improvement at 15 NoC faults, an improvement that more than doubles beyond 50 NoC faults, showing that uDIREC is beneficial over a wide range of fault rates.

Acknowledgments

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

References

[1] E. Nightingale, J. Douceur, and V. Orgovan, "Cycles, cells and platters: An empirical analysis of hardware failures on a million consumer PCs," in *Proc. EUROSYS 6th Conf. Comput. Syst.*, 2011, pp. 343–356.

[2] J. Srinivasan, S. Adve, P. Bose, and J. Rivers, "The impact of technology scaling on lifetime reliability," in *Proc. Int. Conf. Dependable Syst. Netw.*, 2004, pp. 177–186.

[3] S. Borkar, "Designing reliable systems from unreliable components: The challenges of transistor variability and degradation," *IEEE Micro*, vol. 25, no. 6, pp. 10–16, Nov./Dec. 2005.

[4] A. Pellegrini, J. Greathouse, and V. Bertacco, "Viper: Virtual pipelines for enhanced reliability," in *Proc. 39th Annu. Int. Symp. Comput. Archit.*, 2012, pp. 344–355.

[5] S. Gupta, S. Feng, A. Ansari, J. Blome, and S. Mahlke, "The StageNet fabric for constructing resilient multicore systems," in *Proc. 41st IEEE/ACM Int. Symp. Microarchit.*, 2008, pp. 141–151.

[6] D. Fick, A. DeOrio, J. Hu, V. Bertacco, D. Blaauw, and D. Sylvester, "Vicis: A reliable network for unreliable silicon," in *Proc. 46th ACM/IEEE Des. Autom. Conf.*, 2009, pp. 812–817.

[7] K. Aisopos, A. DeOrio, L.-S. Peh, and V. Bertacco, "ARIADNE: Agnostic reconfiguration in a disconnected network environment," in *Proc. Int. Conf. Parallel Archit. Compilation Techn.*, 2011, pp. 298–309.

[8] V. Puente, J. Gregorio, F. Vallejo, and R. Beivide, "Immunet: A cheap and robust fault-tolerant packet routing mechanism," in *Proc. 31st Annu. Int. Symp. Comput. Archit.*, 2004, pp. 198–209.

[9] 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. 12th Int. Symp. High-Perform. Comput. Archit.*, 2006, pp. 5–16.

[10] J. Kim, C. Nicopoulos, D. Park, V. Narayanan, M. Yousif, and C. Das, "A gracefully degrading and energy-efficient modular router architecture for on-chip networks," in *Proc. 33rd Annu. Int. Symp. Comput. Archit.*, 2006, pp. 4–15.

[11] M. Koibuchi, H. Matsutani, H. Amano, and T. Pinkston, "A lightweight fault-tolerant mechanism for network-on-chip," in *Proc. 2nd ACM/IEEE Int. Symp. Netw.-on-Chip*, 2008, pp. 13–22.

[12] F. Chaix, D. Avresky, N.-E. Zergainoh, and M. Nicolaidis, "A fault-tolerant deadlock-free adaptive routing for on chip interconnects," in *Proc. Des. Autom. Test Eur. Conf. Exhib.*, 2011, pp. 1–4.

[13] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. Brown, and A. Agarwal, "On-chip interconnection architecture of the tile processor," *IEEE Micro*, vol. 27, no. 5, pp. 15–31, Sep. 2007.

[14] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar, "An 80-tile sub-100-w teraFLOPS processor in 65-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 43, no. 1, pp. 29–41, Jan. 2008.

[15] R. Parikh and V. Bertacco, "uDIREC: Unified diagnosis and reconfiguration for frugal bypass of NoC faults," in *Proc. 46th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2013, pp. 148–159.

[16] D. Park, C. Nicopoulos, J. Kim, N. Vijaykrishnan, and C. R. Das, "Exploring fault-tolerant network-on-chip architectures," in *Proc. Int. Conf. Dependable Syst. Netw.*, 2006, pp. 93–104.

[17] S. Murali, T. Theocharides, N. Vijaykrishnan, M. Irwin, L. Benini, and G. De Micheli, "Analysis of error recovery schemes for networks on chips," *IEEE Des. Test Comput.*, vol. 22, no. 5, pp. 434–442, Sep./Oct. 2005.

[18] D. Bertozzi, L. Benini, and G. De Micheli, "Low power error resilient encoding for on-chip data buses," in *Proc. Des. Autom. Test Eur. Conf. Exhib.*, 2002, pp. 102–109.

[19] A. Ghofrani, R. Parikh, A. Shamshiri, A. DeOrio, K.-T. Cheng, and V. Bertacco, "Comprehensive online defect diagnosis in on-chip networks," in *Proc. IEEE 30th VLSI Test Symp.*, 2012, pp. 44–49.

[20] J. Raik, R. Ubar, and V. Govind, "Test configurations for diagnosing faulty links in NoC switches," in *Proc. 12th IEEE Eur. Test Symp.*, 2007, pp. 29–34.

[21] E. Cota, F. Kastensmidt, M. Cassel, M. Herve, P. Almeida, P. Meirelles, A. Amory, and M. Lubaszewski, "A high-fault-coverage approach for the test of data, control and handshake interconnects in mesh networks-on-chip," *IEEE Trans. Comput.*, vol. 57, no. 9, pp. 1202–1215, Sep. 2008.

[22] A. Kohler and M. Radetzki, "Fault-tolerant architecture and deflection routing for degradable NoC switches," in *Proc. 3rd ACM/IEEE Int. Symp. Netw.-on-Chip*, 2009, pp. 22–31.

[23] S. Shamshiri, A. Ghofrani, and K.-T. Cheng, "End-to-end error correction and online diagnosis for on-chip networks," in *Proc. IEEE Int. Test Conf.*, 2011, pp. 1–10.

[24] T. Pinkston, "Flexible and efficient routing based on progressive deadlock recovery," *IEEE Trans. Comput.*, vol. 48, no. 7, pp. 649–669, Jul. 1999.

[25] M. Al Faruque, T. Ebi, and J. Henkel, "Configurable links for runtime adaptive on-chip communication," in *Proc. Des. Autom. Test Eur. Conf. Exhib.*, 2009, pp. 256–261.

[26] M. Palesi, S. Kumar, and V. Catania, "Leveraging partially faulty links usage for enhancing yield and performance in networks-on-chip," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 29, no. 3, pp. 426–440, Mar. 2010.

[27] M. Gomez, et al., "An efficient fault-tolerant routing methodology for meshes and tori," *Comput. Archit. Lett.*, vol. 3, no. 1, p. 3, 2004.

[28] C.-C. Su and K. Shin, "Adaptive fault-tolerant deadlock-free routing in meshes and hypercubes," *IEEE Trans. Comput.*, vol. 45, no. 6, pp. 666–683, Jun. 1996.

[29] C.-T. Ho and L. Stockmeyer, "A new approach to fault-tolerant wormhole routing for mesh-connected parallel computers," *IEEE Trans. Comput.*, vol. 53, no. 4, pp. 427–438, Apr. 2004.

[30] M. Ebrahimiand M. Daneshtalab, "A light-weight fault-tolerant routing algorithm tolerating faulty links and routers," *Computing*, vol. 97, no. 6, pp. 631–648, 2015.

[31] M. Pirretti, G. Link, R. Brooks, N. Vijaykrishnan, M. Kandemir, and M. Irwin, "Fault tolerant algorithms for network-on-chip interconnect," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, 2004, pp. 46–51.

[32] P. Bogdan, T. Dumitras, and R. Marculescu, "Stochastic communication: A new paradigm for fault-tolerant networks-on-chip," *VLSI Des.*, vol. 2007, 2007, Article ID 95348, doi:10.1155/2007/95348.

[33] C. J. Glass and L. M. Ni, "Fault-tolerant wormhole routing in meshes without virtual channels," *IEEE Trans. Parallel Distrib Syst.*, vol. 7, no. 6, pp. 620–636, Jun. 1996.

[34] S. Chalasani and R. Boppana, "Communication in multicomputers with nonconvex faults," *IEEE Trans. Comput.*, vol. 46, no. 5, pp. 616–622, May 1997.

[35] Y. Fukushima, M. Fukushi, and S. Horiguchi, "Fault-tolerant routing algorithm for network on chip without virtual channels," in *Proc. 24th IEEE Int. Symp. Defect Fault Tolerance VLSI Syst.*, 2009, pp. 313–321.

[36] Z. Zhang, A. Greiner, and S. Taktak, "A reconfigurable routing algorithm for a fault-tolerant 2D-mesh network-on-chip," in *Proc. 45th Acm/IEEE Des. Autom. Conf.*, 2008, pp. 441–446.

[37] J. Flich, A. Mejia, P. Lopez, and J. Duato, "Region-based routing: An efficient routing mechanism to tackle unreliable hardware in network on chips," in *Proc. 1st Int. Symp. Netw.-on-Chip*, 2007, pp. 183–194.

[38] M. Schroeder, A. D. Birreli, M. Burrows, H. Murray, R. M. Needham, T. L. Rodeheffer, E. H. Satterthwaite, and C. P. Thacker, "Autonet: A high-speed, self-configuring local area network using point-to-point links," *IEEE J. Sel. Areas Commun.*, vol. 9, no. 8, pp. 1318–1335, Oct. 1991.

[39] 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. 20th Int. Parallel Distrib. Process. Symp.*, 2006, p. 105.

[40] J. C. Sancho, A. Robles, and J. Duato, "A flexible routing scheme for networks of workstations," in *Proc. 3rd Int. Symp. High Perform. Comput.*, 2000, pp. 260–267.

[41] A. DeOrio, D. Fick, V. Bertacco, D. Sylvester, D. Blaauw, J. Hu, and G. Chen, "A reliable routing architecture and algorithm for NoCs," *IEEE Trans. Comput.-Aided Des. ICs Syst.*, vol. 31, no. 5, pp. 726–739, May 2012.

[42] A. Prodromou, A. Panteli, C. Nicopoulos, and Y. Sazeides, "NoCAlert: An on-line and real-time fault detection mechanism for network-on-chip architectures," in *Proc. 45th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2012, pp. 60–71.

[43] R. Parikh, A. Ghofrani, V. Bertacco, and K.-T. Cheng, "Comprehensive online defect diagnosis in on-chip networks," in *Workshop Resilient Archit.*, 2011, http://web.eecs.umich.edu/~parikh/documents/wra11.pdf

[44] G.-M. Chiu, "The odd-even turn model for adaptive routing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 11, no. 7, pp. 729–738, Jul. 2000.

[45] O. Lysne, J. Montañana, J. Flich, J. Duato, T. Pinkston, and T. Skeie, "An efficient and deadlock-free network reconfiguration protocol," *IEEE Trans. Comput.*, vol. 57, no. 6, pp. 762–779, Jun. 2008.

[46] T. Pinkston, R. Pang, and J. Duato, "Deadlock-free dynamic reconfiguration schemes for increased network dependability," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 8, pp. 780–794, Aug. 2003.

[47] M. Prvulovic, Z. Zhang, and J. Torrellas, "ReVive: Cost-effective architectural support for rollback recovery in shared-memory multiprocessors," in *Proc. 29th Annu. Int. Symp. Comput. Archit.*, 2002, pp. 111–122.

[48] A. DeOrio, K. Aisopos, V. Bertacco, and L.-S. Peh, "DRAIN: Distributed recovery architecture for inaccessible nodes in multi-core chips," in *Proc. 48th ACM/EDAC/IEEE Des. Autom. Conf.*, 2011, pp. 912–917.
[49] K. Aisopos and L.-S. Peh, "A systematic methodology to develop resilient cache coherence protocols," in *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2011, pp. 47–58.
[50] R. Parikh and V. Bertacco, "Formally enhanced verification at runtime to ensure NoC functional correctness," in *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2011, pp. 410–419.
[51] J. Sancho, A. Robles, and J. Duato, "An effective methodology to improve the performance of the up*/down* routing algorithm," *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, no. 8, pp. 740–754, Aug. 2004.
[52] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, W. J. Dally, "A detailed and flexible cycle-accurate network-on-chip simulator," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2013, pp. 86–96.
[53] C. Bienia, S. Kumar, J. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proc. 17th Int. Conf. Parallel Archit. Compilation Techn.*, Oct. 2008, pp. 72–81.
[54] M. Martin, D. Sorin, B. Beckmann, M. Marty, M. Xu, A. Alameldeen, K. Moore, M. Hill, andD. Wood,"Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *ACM SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 92–99, 2005.
[55] J. Howard, et al., "A 48-core IA-32 message-passing processor with DVFS in 45nm CMOS," in *Proc. IEEE Int. Solid-State Circuits Conf. Dig. Techn. Papers*, 2010, pp. 108–109.
[56] A. Kahng, B. Li, L.-S. Peh, and K. Samadi, "ORION 2.0: A fast and accurate NoC power and area model for early-stage design space exploration," in *Proc. Des. Autom. Test Eur. Conf. Exhib.*, 2009, pp. 423–428.
[57] C. Glass and L. Ni, "The turn model for adaptive routing," in *Proc. 19th Annu. Int. Symp. Comput. Archit.*, 1992, pp. 278–287.
[58] W. Dally and C. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. Comput.*, vol. C-36, no. 5, pp. 547–553, May 1987.

Ritesh Parikh (S'09-M'14) received the BTech and MTech degrees in electronics engineering from the IIT Kharagpur, India, in 2008 and 2009, respectively, and the PhD degree in computer engineering from the University of Michigan, Ann Arbor, MI, in 2014. He was a postdoctoral research fellow at the University of Michigan, Ann Arbor. He is currently a performance architect with Intel's server engineering division. His research interests are in networks-on-chip, heterogeneous computer architecture, and in ensuring the correctness of multicore computer designs, including verification and reliable system design.

Valeria Bertacco (S'95-M'03-SM'10) received the Laurea degree in computer engineering from the University of Padova, Italy, and the MS and PhD degrees in electrical engineering from Stanford University in 2003. She was at Synopsys, Mountain View, CA, for four years. She is currently a professor of electrical engineering and computer science with the University of Michigan, Ann Arbor, MI. Her current research interests include complete design validation, digital system reliability, and hardware-security assurance. She received the IEEE CEDA Early Career Award and served on the program committees of DAC, DATE, and MICRO.