Synergistic Use of Multiple On-Chip Networks for Ultra-Low Latency and Scalable Distributed Routing Reconfiguration

Marco Balboni
ENDIF - MPSoC Research Group
University of Ferrara, ITALY
email: marco.balboni@unife.it

José Flich
GAP - Parallel Architectures Group
Universidad Politécnica de Valéncia, SPAIN
email: jflich@gap.upv.es

Davide Bertozzi
ENDIF - MPSoC Research Group
University of Ferrara, ITALY
email: davide.bertozzi@unife.it

Abstract—**Extending the principle of partially good die allowance to manycore processors, and testing them over time to detect the onset of permanent faults, are only feasible through proper support in the on-chip interconnection network. In fact, this implies the ability to reconfigure the routing algorithm at runtime to reflect changes in network topologies. Current literature cannot avoid a large hardware and/or software overhead when tackling this challenge. This paper exploits the existence of multiple physical networks in industry-relevant manycore processors in a synergistic way, for the sake of fast and scalable distributed reconfiguration of the routing function at runtime.**

I. Introduction

While most network-on-chip (NoC) research contributions have focused on the architecture design principles or on the physical design flow so far, concerns associated with the low reliability of the silicon substrate at upcoming technology nodes are calling for new design methods for runtime management of the system interconnect. On one hand, sustaining manufacturing yield and device lifetime imply that a failure of a NoC component cannot cause the entire chip to be considered as defective. This goes beyond fault-tolerant routing algorithms [2] or flexible routing mechanisms [3], since the above techniques fail to capture the transition from one network configuration to the next one, which is potentially deadlock-prone and penalizing for instantaneous performance of network traffic. On the other hand, testing complex manycore chips cannot be only a post-manufacturing course of action, but needs to make inroads into the lifetime of the device. One clear trend is toward fault detection and reconfiguration frameworks [4], [5], where network resources are tested aggressively to detect early signs of an upcoming fault through a built-in self-testig infrastructure. The key novelty of the testing challenge lies in the fact that NoC links should be taken offline during runtime testing, while at the same time guaranteeing uninterrupted availability of the NoC. In order to maintain maximum flexibility in link deactivation, the routing algorithm must be able to change dynamically in reply to changes in system state, while preserving deadlock freedom. Current approaches to runtime network configuration suffer from large hardware/software overhead and/or lack of scalability. In general, centralized approaches have the disadvantage that some reconfiguration tasks (e.g., the computation of the new routing function) are performed in software. In contrast, distributed reconfiguration suffers from suboptimality of emergency routing solutions and overly high implementation cost and complexity. This paper moves from a different perspective: the synergistic exploitation of routing resources that are already there in many NoC implementations. In a sense, there is an overhead which is increasingly accepted in NoC design, and which is justified by other design goals, which consists of the use of multiple physical networks instead of logic ones. Although this seems to run contrary to much previous work [1], [26], it is actually motivated by how the relative costs of network design change for implementation on a single die [6]. First, wiring resources are abundantly available on chip, for realistic tile sizes. Second, logic networks do not cut down on the amount of used buffering resources. Third, building multiple physical networks via replication simplifies the design and provides more intertile communication bandwidth. This is the reason why up to 5 physical networks can be found in industrial designs. Each one can even be customized for the needs of the specific traffic class it accommodates. Given this, this paper proposes to exploit the existing multiple physical networks to spatially separate resource allocations that may close dependency cycles. The most straightforward way of accomplishing deadlock-free spatial separation is to double the number of resources used by a routing algorithm to escape from deadlock and to allow dependencies from new-epoch traffic to old traffic, but not vice versa. Whenever a switch port processing old traffic has a routing dependency with a port already migrated to the new epoch, an escape path is set up into another network plane. This way, deadlock cannot take place. The only requirement the escape network should fulfill consists of its compatibility with the network under reconfiguration from the message-dependent deadlock viewpoint, unless a specific course of action is set up to tackle this concern differently. This paper develops a reconfiguration methodology around the above basic ideas and demonstrates a substantial improvement over state-of-the-art in terms of reconfiguration latency, area overhead, impact over the performance of running traffic, and scalability to large networks.

II. Related Works

An overview of existing fault-tolerant routing techniques has been reported by [13]. On one hand, routing tables and logic can be updated upon each fault occurrence [13]–[17]. On the other hand, bypass rules can be exploited to reroute around faults using local connectivity information [18], [19].

Runtime reconfiguration of the routing function has been first investigated in high-performance local area networks, spurred by the need to deliver incremental expansion capabilities. Static reconfiguration (SREC) has long been the dominant solution. With SREC, no packets can be routed according to the new routing function while there are still packets in the network routed according to the old one [9]. Dynamic reconfiguration (DREC) techniques overcome this limitation [10]–[12]. However, they are applicable only to a limited set of routing functions, or rely on dropping packets to avoid deadlocks, or appear to be more complex than the straightforward static approach, or have requirements on the minimal set of hardware resources implemented. For instance, the double scheme proposed in [7] proposes the spatial and/or temporal separation of the routing resources used by each routing function into two sets, and allows dependencies to exist from one set of resources to the other but not from both at any given time. Unfortunately, it requires the network to implement two sets of data virtual channels. Other approaches strike a trade-off between SREC and DREC. For instance, the

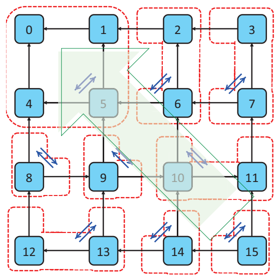


Figure 1. Segments and scroll-up token propagation in a 2D mesh.

work in [8] describes how the various phases of SREC can be overlapped in order to increase parallelism.

When it comes to on-chip networks, a few main approaches stand out. ARIADNE [14] is fully distributed, however it undergoes subtle effects: its latency badly scales with network size, and it does not guarantee a transparent transition between configurations. MD [18] routes packets adaptively through the shortest paths in the presence of a faulty link, as long as a path exists. The local visibility of this mechanism causes the network to become rapidly disconnected as the number of faults increases. OSRLite [25] has been proposed as an embodiment of native OSR into an on-chip environment. [20] improves [25] with an algorithm that extends coverage of fault patterns. The main issue with OSRLite is its centralized nature, which causes overhead for manager notifications, and puts software computation of the global manager on the critical path for the reconfiguration process. This latter disadvantage is also shared by the work in [13]. Recently, BLINC has significantly raised the bar for fast, deadlock-free, distributed and localized routing reconfiguration [4]. BLINC uses precomputed routing metadata to quickly evaluate localized detours upon each fault manifestation. Unfortunately, the complexity of this scheme is significant (more than 500 bits per router in an 8x8 mesh, with poor scalability as the network size increases). However, since [4] has demonstrated superior reconfiguration latency and fault tolerance with respect to the main competing schemes in literature, we consider BLINC as the reference solution for comparison.

This paper aims at a distributed routing reconfiguration method at runtime for NoCs. It borrows the same deadlock-avoidance principle from OSR/OSRLite, that is, separation of old and new packets with a token. However, the scheme is then augmented to become fully distributed, which was possible due to the same spatial separation concept for deadlock freedom proposed by the double scheme [7]. However, the difference with the original schemes is significant. Differently than OSR, we do not have a centralized control function, since our reconfiguration process is fully distributed. Differently than the double scheme, we do not envision a dedicated physical network only for reconfiguration, but we exploit existing ones, therefore we need to meet the additional constraint of minimum performance perturbation of the background traffic in the escape network through a smart escape strategy. The outcome is a new design point for runtime and distributed reconfiguration of the routing function in NoCs.

III. Reconfiguration Mechanism

A. Background

The paper relies on segment-based routing (SR). SR is topology-agnostic in nature, and works by partitioning a topology into segments (an example is in Fig.1). This allows to place bidirectional turn restrictions locally and independently within a segment, thus making the network deadlock-free.

Without lack of generality, we assume the uLBDR (Universal Logic-Based Distributed Routing) routing mechanism as proposed in Rodrigo et al. paper [22]. It has several routing configuration bits at each switch (26) that enable to take the proper routing decision based on the destination coordinates of the packet at hand, and on the routing restrictions posed by SR. uLBDR supports non-minimal paths through the use of deroutes.

This paper moves from the OSRLite runtime routing reconfiguration function, and ultimately augments it to overcome its global and centralized nature. Before delving into the proposed method, some OSRLite basics are recalled. In OSR, a global controller is in charge of initiating reconfiguration, either because of a planned decision (e.g., power management) or of an unexpected event (e.g., the likely onset of a permanent fault). In the latter case, the event needs to be notified to the manager. The manager computes the configuration bits for the new routing function to activate, and updates the corresponding registers in NoC switches through a dual NoC. However, the transition from the old to new routing function occurs in a controlled way, so to avoid deadlock. In practice, a separation token crosses the network in the order of its channel dependency graph, starting from a root node. As the token is received at switch input ports, the new routing function is activated as those ports are emptied by old traffic. Similarly, output ports evolve to the new epoch as the input ports that have no routing restrictions toward them have moved into the new epoch. In practice, the network evolves to the new routing function progressively, by enabling concurrent local and static reconfigurations at its switch ports.

Fig.1 shows the direction of token propagation across the network. Only the scroll-up phase is shown. The scroll-down phase, which causes the remaining links to receive the token, is omitted for lack of space. Fig.2a also shows the scroll-up token propagation tree. A switch with a given ID can fire a token in its output ports only when all tokens from the input ports have been received, and have internally propagated to the output ports through the stated rules.

The main issues with OSRLite are:

- the global manager is on the critical path of the reconfiguration process.
- the token propagation starts from a root node.
- a separate control network or virtual channel is needed for communication with the global manager.

B. Key Idea

We overcome the main limitations of OSRLite in the direction of a fully distributed and dynamic reconfiguration mechanism by relying on the following key intuitions:

First, considering results of Triviño et al. [21], we can identify a region around a NoC fault that is affected by it. In practice, for each fault in the NoC, we don't have to update all the routing configuration bits of all switches in the NoC, but only of a limited subset of them. This means that it is possible to border the region where the routing function has to be changed.

Second, since each switch is involved only in a limited number of fault regions, the modifications of the uLBDR configuration registers for those cases could be encoded in a small table for each router. The size of the table would be 26 bits for each relevant fault. This way, the routing function should not be computed by a global controller, but would be encoded in distributed tables. The fault coverage of this approach will be addressed in section IV-D.

Third, making OSRLite a distributed mechanism implies that not only the root node, but also every node in the network can trigger token propagation, as an effect of a detected risk of malfunctioning in a link, or of a dedicated testing phase which is about to start in the link under test. Consider for instance Fig.2b, where the indicated link needs immediate disconnection. Switch with ID 10 needs to avoid routing traffic through the critical link. To do that, OSRLite would require tokens in the two input ports from 9 and 14. Our scheme mimics the same behavior by redirecting the links of those input ports into an escape network. When this happens, and switch 10 is drained by old traffic, no packets will cross the critical link any more, and a token will appear at the south port of switch with ID 6. For the same reason, a regular token will be concurrently triggered to the east. However, this way the token propagation would stop, because for instance switch 11 needs also a token from south to fire. Similarly for switches 4 and 5. Therefore, we need to open more tunnels. In the next section, this mechanism will be further optimized to reduce inter-network tunneling and speed-up the reconfiguration.

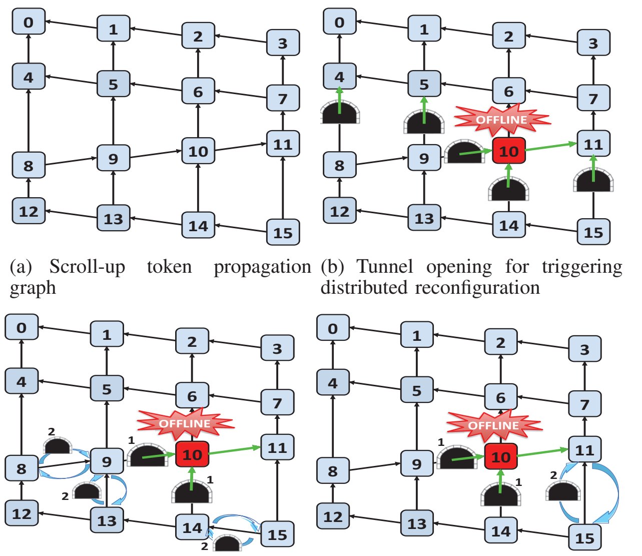
The proposed method has the key requirement of an escape network. For instance, networks carrying intra-partition or inter-core traffic in a manycore processor could be reconfigured on top of a global network (for I/O or memory controller communication). Alternatively, a network carrying one message type could be reconfigured on top of a network carrying a different message type, provided the two message types do not form a dependency chain rising the risk of message-dependent deadlock. For instance, memory requests messages cannot be tunneled into a response network, and vice versa. Nonetheless, another case falls within reach of this paper, that is, multiple networks with multiple virtual channels each. For instance the memory request VC of physical network 0 could be tunneled into a memory request VC of physical network 1. This is message-dependent deadlock safe. Last but not least, the mechanism is complementary, that is, the role of the network under reconfiguration and the escape one can be flipped for the sake of exhaustive testing.

Finally, in order to enable the two coupled links during reconfiguration to have different routing functions, escape paths are assumed to go through the local ports of escape switches, hence ending up being multiplexed with the traffic from IP source cores.

C. The Baseline Mechanism

When putting together the three ideas from the previous section, we get the following reconfiguration methodology. A switch can enforce the fast disconnection of an attached link by triggering the token propagation process. It will handshake the opening of inter-network tunnels with nearby switches. Normal tokens are then fired by the target switch, which will trigger the scroll-up phase of the token propagation. Once completed, the scroll-up phase will trigger the scroll-down phase. Once the scroll-down phase reaches the OSRLite root node, then the missing scroll-up phase (since the token propagation started somewhere in the middle of the network) will be triggered, till the tokens reach the tunnels and these latter are closed. This completes the reconfiguration process.

Once reached by a token, the input port of a switch will behave like in vanilla OSR, with the difference that a fault identifier needs to travel with tokens. Fault IDs will be searched



(a) Scroll-up token propagation graph (b) Tunnel opening for triggering distributed reconfiguration

(c) Tunnel propagation (d) Eager token request

Figure 2. Token OSR..

in a local CAM memory, indicating whether that fault ID requires routing bit modifications at the switch under test or not. For this reason, the token can be a single-flit special packet carrying the fault ID in its body. There is a fault ID for every bidirectional link in the network (say M), however the number of CAM entries in a switch is limited, since not all faults affects its routing bits. The methodology does not need to affect the network as a whole. In fact, thanks to the notion of fault region around a faulty link, only switches in the fault region should be affected by the token propagation. Incoming links from boundary switches may be assumed to already exhibit a token, since incoming traffic will be eventually derouted inside the fault region.

D. Optimized Mechanism

The three phases of the process (partial scroll-up, scroll-down, residual scroll-up) make it overly long in time. This motivates our next optimizations.

Tunnel Propagation

The switch directly attached to the link to disconnect requests tunnel opening to upstream switches with channel dependencies with the link under test. These latter open such tunnels right after pending packets are completed. See for instance switches 9 and 14 in Fig.2c. However, tunnels between 8-4, 9-5 and 15-11 are not opened, thus avoiding the need for dedicated multi-hop signaling. The novelty is that these switches then iterate the mechanism with their upstream switches in the scroll-up token propagation graph. That is, they pretend that tunnels are fired tokens, and try to fulfill the requirements to fire these tokens. For instance, switch 9 will handshake with switches 8 and 13 the opening of tunnels on the connecting links. Once this is completed, and switch 9 has internally processed all pending old traffic, tunnels between 9-10 and 14-10 will be closed, since all the traffic routed across those links will belong to the new routing function (that is, no traffic at all). Overall, tunnels are propagated backwards along the token propagation graph, till they reach the OSRLite root node. This mechanism overlaps the token partial and residual scroll-up phases.

Eager tunnel Request

When switch 11 in Fig.2d receives the token from switch 10, it temporarily misses a token from south to fire. In order to avoid multi-hop dedicated signaling between 10 and 15 to open a tunnel in that location, switch 11 directly asks switch 15 for the missing token through an eager token injection handshaking. Switch 15 opens the tunnel, then in turn applies tunnel propagation with its upstream switches in the token propagation graph. This mechanism is applied by all switches in the network as they receive an incoming token, and speeds up the reconfiguration process significantly. In particular, it is applied also by switch 6 at the opposite side of the link to be put offline. In that case, the switch receives the token from the link under test, and handshakes tunnel opening on those input links that have routing dependencies with the target link. The relevant aspect here is that a few such input links will belong to the scroll-down phase. In turn, switches opening tunnels will propagate them further upstream, thus speeding up the partial scroll-up phase, and overlapping it with the scroll-down phase. Eager tunnel request raises a new condition for stopping tunnel propagation. This latter finishes not only when tunnels reach the OSRLite root node, but also when tunnel backward propagation is requested for a link which has (or is firing) a token. In that case, the tunnel is closed. An example is illustrated in Section III-E.

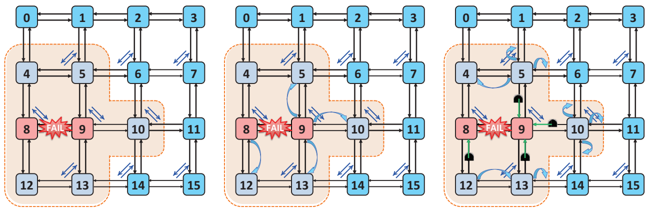
The ultimate effect of our optimizations can be understood as though several spots were enlarging simultaneously on a white surface, thus contributing to cover the whole surface in the smallest possible time. Tunnels are the borders of such spots. As the tunnels propagate as a wave, the incident traffic is derouted to the escape network, since it is old traffic that is trying to enter a new domain.

E. *The mechanism at Work*

Let us consider a 4×4 mesh and the link between routers 8 and 9 becoming faulty (although still operational) or under test. Fig.3 illustrates the mechanism at work. The sequence of events is displayed under the assumption that token propagation inside switches takes 3 cycles as from the RTL characterization in [25]. In contrast, tunnel opening requests are processed in one cycle.

After detecting the fault event on the link (Fig.3a), the two switches, connected to the link, trigger the reconfiguration process by sending token/tunnel activation requests to neighbor switches to open tunnels at some of their output ports (those with dependencies with the link under test, see Fig.3b). Once tunnels are opened (Fig.3c), and after an emptying transient of in-transit traffic of at least 3 cycles, switches 8 and 9 guarantee tokens are triggered through the faulty link (Fig.3e). Indeed, all input dependencies of the output port connected to the faulty link are either inexistent or there is a tunnel at the input port. Thus, it is guaranteed that no old traffic in any direction will arrive needing to cross the failed link.

Figure 3c shows the initial location of tunnels (at N output ports of routers 12 and 13, at S output port of router 5 and at W output port of router 10) and their equivalence with tokens at upstream switches of tunneled ports. Once tunnels are opened, they trigger new tunnels (Figure 3c). In particular, switch 12 and 13 request tunnel propagation to the east, since without a tunnel from there it is not possible to close their tunnels on the north ports (see token propagation requirements in Fig.1). However, switch 13 is on the boundary of the fault-region, therefore its request is dropped (equivalent to an incoming token in Figure 3d). Similarly, tunnel opening requests from switch 10 (affecting both scroll-up and scroll-down links) will be dropped. Finally, only one request from 5 is served (Figure 3d). Always in Figure 3d, switch 4 is showed to be further propagating tunnels at its inputs.



(a) Detection of a link fault (b) Token requests triggered (c) Tunnel Creation

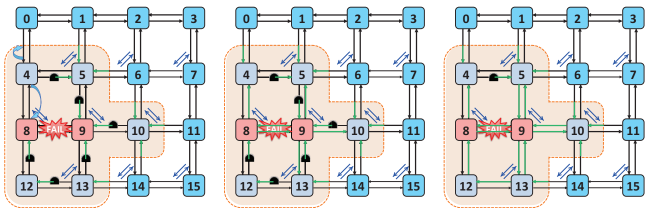

(d) Token request at boundaries (e) Token propagation on the link. (f) Tunnel propagation

Figure 3. Tunneling Mechanism at Work.

In Figure 3e the normal OSRLite token propagation occurs. In particular, switches 8 and 9 have completed the processing latency of their input tokens and can fire this token on the output ports. Interestingly, we can see that the tunnel request from switch 4 is not served because a token is concurrently fired by switch 8 on the connecting link. After 3 cycles, the tunnel between 4 and 5 will be closed (it would be logically in Figure 3g, not shown). It is exactly at the point in time illustrated by Figure 3e that the link under test is actually put offline.

Finally, in Figure 3f tokens coming out from switch output ports have closed the associated tunnels. Here, token propagation becomes apparent, since the initial tunnels have disappeared by crossing the fault region boundary, and a derived tunnel is still open awaiting to disappear in the same way.

In order for the reconfiguration process to complete, a few scroll-down links are left (in black in figure), whose reconfiguration will be triggered by switch 5. This latter in fact has all conditions to fire a token to the west, which will in turn cause the token propagation across the remaining links.

IV. Experimental Evaluation

We evaluate our mechanism using an RTL-equivalent SystemC model of the xpipesLite NoC architecture [23].

A. *Reconfiguration Latency*

We evaluate the reconfiguration latency in an unloaded 4×4 mesh network. We perform the analysis for every 1-link failure and for three different mechanisms. The first one represents the native OSRLite mechanism. The second and third ones represent our mechanism (Tunneled OSR, TOSR) with and without its limitation to the fault-region.

As Fig.4) shows, the baseline OSRLite mechanism (in red) provides a uniform reconfiguration delay, as OSR involves a global reconfiguration process, involving the whole network and starting from the root node. We did not consider possible signaling needs with the global controller, nor the new routing function computation of this latter, but only the pure reconfiguration latency. In blue, we show TOSR without the fault-region optimization. As we see, reconfiguration latency

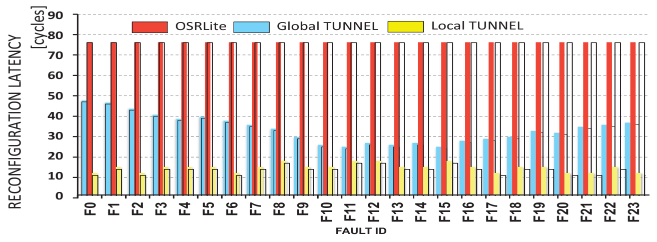


Figure 4. Reconfiguration Latency in a 4×4 mesh: baseline OSRLite(red), Global TOSR (Blue) and optimized Local TOSR (yellow).

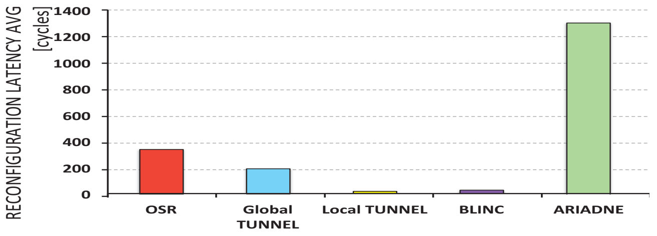


Figure 5. Average reconfiguration Latency in an 8×8 2D mesh.

is always better than the baseline solution with OSRLite. This reduction is achieved as the tunnel approach speeds up the reconfiguration process during both the scroll-up or scroll-down phase. As we can observe, TOSR triggered at failures in the center of the mesh achieve lower reconfiguration times as the scroll-up and scroll-down phases are balanced and take the same time. Contrary to this, at the corner of the mesh (link F0) TOSR highly depends on the scroll-down phase which is slower as it needs to wait for the token given by the scroll-up phase to be created with tunnel mechanism.

Finally in yellow, the figure shows the TOSR reconfiguration when the fault-region optimization is included. In our case, the defined region includes from 5 up to 8 routers. As we can see, Local TOSR achieves faster reconfiguration times as the reconfiguration domain is much smaller. The reconfiguration dynamics are also changed, since faults in the middle of the NoC cause the largest fault-regions to be reconfigured.

Figure5) shows the average reconfiguration latency for an 8×8 mesh network. We add available results for BLINC and ARIADNE for this network size (extracted from their publications). Only for (Local) TOSR we consider the worst case latency. Clearly, TSOR achieves about 35% of speedup with respect to the closest competitor, that is BLINC, under the same operating conditions. While BLINC's latency grows weakly with the network size (15% from 8x8 to 10x10), TSOR worst case latency stays constant because the maximum fault bounding region (8 switches) has already showed up in an 8x8 network. Therefore, our gap with BLINC widens.

B. Area Overhead

MECHANISM	8x8 2D mesh	16x16 2D mesh
TOSR	464	464
BLINC	648	2368

Table I. AREA OVERHEAD IN TERMS OF REGISTER BITS.

We express area overhead in terms of number of additional register bits that each mechanism under test requires with respect to the baseline architecture not capable of reconfiguration. Considering the worst case of TOSR, as shown in Table I), it scales better then other solutions proposed because of the fixed maximum dimension the region to be reconfigured can reach. We have to consider that, as shown in [21], in the worst case

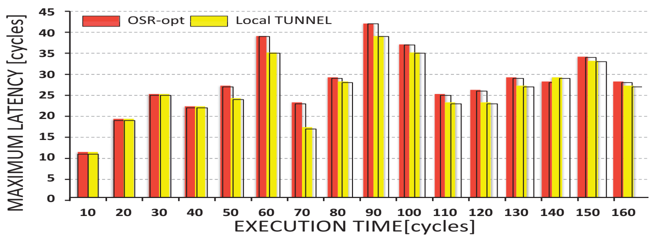


Figure 6. Impact on upper-network considering a medium injection rate.

a router can be involved in 16 faults, so it needs 26 LBDR bits for each fault. This creates an overhead of 416 bits per switch. Additionally, to support the TOSR optimization with the token propagation localized in a region around the failure, we need 4 extra Connectivity bits to inform a router that it's on the boundary of a reconfiguration region, so it has to absorb the token request during the propagation. Also in this case we have to consider that a switch can be part of different region boundaries, depending on the position of the fault that is occurring. In the worst case, we calculate that a switch can be part of 12 boundaries, giving 48 additional bits. Already in an 8x8 mesh, this result outperforms BLINC's one, essentially due to its preference list to provide good-enough emergency paths.

As the table shows with a 16×16 mesh, while TOSR stays constant, BLINC's requirements skyrocket, due to the longer children sets and preference list, clearly denoting a lack of scalability of this latter scheme.

C. Impact on Packets Latency

BLINC is too conservative and does not exploit the routing capabilities of SR since multiple valid paths are possible from any pair of source-destination nodes. This negative impact can be alleviated in BLINC by using the expensive preference list table. In contrast, with our mechanism, we manage to use minimal paths for every source-destination pair even when the failure is present. This is achieved since we rely on already computed entries for all the one-link failure cases. Non-minimal paths are taken only when bypassing the failed link, thus being minimal for the topology with the failure. As a result, our mechanism yields a steady state with better-performing routes. Fig. 6 in [4] quantifies this penalizing gap for BLINC as an increase by 3% of the average hop count with respect to optimal routes, the same optimal routes that this work provides.

Next, we then explore the impact of tunnel opening on the two coupled networks during the reconfiguration transient. We consider two 4×4 mesh NoCs. The injectors are synthetic testbenches set to generate uniform random traffic.

The first experiment is set considering a medium injecting rate on the network under reconfiguration, with 0% injection rate on the escape network. A reconfiguration is triggered after 45 cycles of the simulation, when the link between switches 8 and 9 needs to be put offline. Fig.6 shows instantaneous maximum packet latency over time. Our approach gives improvements from 1% to 11% with respect to OSRLite, and an average improvement of 6%, which means the escape network is providing the expected improvements in performance predictability during reconfiguration. It is worth recalling that the OSRLite variant used here is the one which yields quasi-transparent reconfiguration from [24], hence it is not the native OSRLite, which would have been trivially outperformed by 40%.

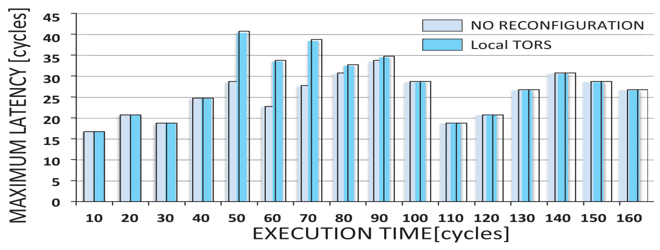


Figure 7. Impact on escape network traffic considering a 40% injection rate.

The counterpart of opening tunnels is a perturbation on the traffic in the escape network. We test this effect considering two different setups: first considering 5% of the traffic from the masters injected into the escape network, then 40% . In the former case, the effect of having tunnels opened causes less than 9% worst-case increase of the maximum latency, while in the latter case, due to a bigger amount of traffic in the network, the latency is worsened by about 45% in the worst case (Fig.7). But being the reconfiguration mechanism very fast, this perturbation dies out quickly (roughly 30 cycles).

D. Coverage

BLINC states that is able to support any failure combination whenever every link failure is localized on a different segment. This is a property of the SR algorithm since it keeps connectivity and deadlock-free conditions by constructions. Indeed, a failed link represents a turn restriction located inside the segment. Therefore, BLINC achieves 100% coverage for 1-link failure cases. For link failure combinations with two failed links in the same segment BLINC relies on an external and conservative solution that will recompute the algorithm again. Therefore, BLINC does not achieve (by its own) 100% coverage for 2-link failures and beyond. Our mechanism achieves the same coverage of BLINC since it also relies on the segment-based approach. That is, we achieve 100% 1-link failure coverage. For 2-link failures the coverage is 98.8% for an 8x8 mesh, which grows to 99.3 for a 10x10 mesh. The proof is in [21], since it relies on a similar table of encoded 1-link faults, but implemented in software. As stated in that work, failure combinations are compatible in the sense that the correct actions to perform in the network is the addition of each individual action to handle each link failure.

V. Conclusions

In this paper, we show that the synergistic exploitation of multiple physical networks can lead to a fast, low-impact and scalable dynamic reconfiguration of the routing function at runtime. We bound the area affected by a fault, and devise a mechanism for the fast yet controlled switching of the routing function to the new epoch in it. We rely on concurrent token and tunnel propagation, thus quickly moving the boundaries between new-old traffic and old-new traffic respectively. We show minimum perturbation of the escape NoC, and only for an overly short amount of time with respect the reconfiguration latencies of competing approaches. The mechanism can finally scale to a large number of cores, since the bounding area of faults stays the same.

Acknowledgement

This research has been supported by the 7th Framework Program of the EU through the **vIrtical** Project, GA 288574.

References

[1] Young-Jin Yoon, Nicola Concer, Michele Petracca, Luca P. Carloni: ”Virtual Channels and Multiple Physical Networks: Two Alternatives to Improve NoC Performance.” IEEE Trans. on CAD of Integrated Circuits and Systems 32(12): 1906-1919 (2013)

[2] D.Fick, A.DeOrio, G.Chen, V.Bertacco, D.Sylvester, D.Blaauw: ”A Highly Resilient Routing Algorithm for Fault-Tolerant NoCs.” Design Automation and Test in Europe Conference, pp.21-26, 2009.

[3] S.Rodrigo et al.: ”Yield-oriented evaluation methodology of network-on-chip routing implementations.” Int. Symp. on Systems-on-Chip, 2009.

[4] D.Lee, R.Parikh, V.Bertacco: ”Brisk and Limited-Impact NoC Routing Reconfiguration.” DATE 2014.

[5] Alberto Ghiribaldi et al.: ”A complete self-testing and self-configuring NoC infrastructure for cost-effective MPSoCs.” ACM Trans. Embedded Comput. Syst. 12(4): 106 (2013).

[6] D.Wentzlaff et al.: ”On-Chip Interconnection Network Architecture of the Tile Processor.” IEEE Micro, Vol.27, Issue 5, pp.15-31, 2007.

[7] R.Pang, T.M.Pinkston, J.Duato: ”The Double Scheme: Deadlock-Free Dynamic Reconfiguration of Cut-Through Networks.” ICPP 2000.

[8] O.Lysne, J.M.Montanana, J.Flich, J.Duato, T.M.Pinkston, T.Skeie: ”An Efficient and Deadlock-Free Network Routing Reconfiguration Protocol.” IEEE Trans. on Computers, vol.57, issue 6, pp.762-779, 2008.

[9] D. Teodosiu, J. Baxter, K. Govil, J. Chapin, M. Rosenblum, and M. Horowitz: ”Hardware Fault Containment in Scalable Shared- Memory Multiprocessors.” Proc. 24th Ann. Int. Symp. Computer Architecture, Computer Architecture News, vol. 25, pp. 73-84, 1997

[10] R. Casado, et al.: ”A Protocol for Deadlock-Free Dynamic Reconfiguration in High-Speed Local Area Networks.” IEEE Trans. Parallel and Distributed Systems, 2001

[11] D. Avresky and N. Natchev, ”Dynamic Reconfiguration in Computer Clusters with Irregular Topologies in the Presence of Multiple Node and Link Failures.” IEEE Trans. Computers, vol. 54, no. 5, May 2005.

[12] D. Avresky and N. Natchev, ”Intelligent Dynamic Network Reconfiguration.” Proc. 21st Intl. Parallel and Distributed Processing Symp., pp. 1-9, 2007.

[13] E. Wachter, A. Erichsen, A. Amory, and F. Moraes, ”Topology-agnostic fault-tolerant NoC routing method.” Proc. 21st Intl. Parallel and Distributed Processing in Proc. DATE, 2013.

[14] K. Aisopos, A. DeOrio, L.-S. Peh, and V. Bertacc0, ”ARIADNE: agnostic reconfiguration in a disconnected network environment.” in Proc. PACT, 2011.

[15] A. DeOrio et al., ”A reliable routing architecture and algorithm for NoCs,” IEEE Trans. CAD, vol. 31, no. 5, 2012.

[16] V. Puente, J. Gregorio, F. Vallejo, and R. Beivide, ”Immunet: a cheap and robust fault-tolerant packet routing mechanism,” in Proc. ISCA, 2004.

[17] S. Rodrigo et al., ”Addressing manufacturing challenges with costefficient fault tolerant routing,” in Proc. NOCS, 2010.

[18] M. Ebrahimi, et al., ”MD: minimal path-based fault-tolerant routing in on-chip networks,” in Proc. ASPDAC, 2013.

[19] Z. Zhang, A. Greiner, and S. Taktak, ”A reconfigurable routing algorithm for a fault-tolerant 2D-mesh network-on-chip,” in Proc. DAC, 2008.

[20] F. Trivino, D. Bertozzi, and J. Flich, ”A fast algorithm for runtime reconfiguration to maximize the lifetime of nanoscale NoCs,” in Proc. INA-OCMC, 2013

[21] F. Trivino, et al., ”A complete self-testing and self-configuring NoC infrastructure for cost-effective MPSoCs,”, ACM Transactions on Embedded Computing Systems (TECS), Volume 12 Issue 4, June 2013.

[22] S. Rodrigo, J. Flich, A. Roca, S. Medardoni, D. Bertozzi, J. Camacho, F. Silla, J. Duato, ”Addressing Manufacturing Challenges with Cost-Efficient Fault Tolerant Routing”, Proceedings of the 2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip.

[23] S. Stergiou, F. Angiolini, S. Carta, L.Raffo, D. Bertozzi, G. De Micheli. ”xpipes Lite: A Synthesis Oriented Design Library For Networks on Chips”, DATE 2005: 1188-1193.

[24] Marco Balboni, Francisco Triviño, José Flich, Davide Bertozzi. ”Optimizing the Overhead for Network-on-Chip Routing Reconfiguration in Massively Parallel Multi-Core Platforms”, Int. SoC Symposium, 2013.

[25] A. Strano, D. Bertozzi, et al., ”OSR-Lite: Fast and deadlock-free NoC reconfiguration framework”, *International Conference on Embedded Computer Systems* (SAMOS), July 2012.

[26] F. Gilabert, M.E. Gomez, S. Medardoni, D. Bertozzi. ”Improved utilization of NoC channel bandwidth by switch replication for cost-effective multi-processor systems-on-chip”, pp.165-172, NOCS 2010.