Static Bubble: A Framework for Deadlock-free Irregular On-chip Topologies

Aniruddh Ramrakhyani
School of ECE
Georgia Institute of Technology
aniruddh@gatech.edu

Tushar Krishna
School of ECE
Georgia Institute of Technology
tushar@ece.gatech.edu

***Abstract*—Future SoCs are expected to have irregular on-chip topologies, either at design time due to heterogeneity in the size of core/accelerator tiles, or at runtime due to link/node failures or power-gating of network elements such as routers/router datapaths. A key challenge with irregular topologies is that of routing deadlocks (cyclic dependence between buffers), since conventional XY or turn-model based approaches are no longer applicable.**

Most prior works in heterogeneous SoC design, resiliency, and power-gating, have addressed the deadlock problem by constructing spanning trees over the physical topology; messages are routed via the root removing cyclic dependencies. However, this comes at a cost of tree construction at runtime, and increased latency and energy for certain flows as they are forced to use non-minimal routes. In this work, we sweep the design space of possible topologies as the number of disconnected components (links/routers) increase, and demonstrate that while most of the resulting topologies are deadlock *prone* (i.e., have cycles), the injection rates at which they deadlock are often much higher than the injection rates of real applications, making the current solutions highly conservative.

We propose a novel framework for deadlock-freedom called Static Bubble, that can be applied at *design time* to the underlying mesh topology, and guarantees deadlock-freedom for any *runtime* topology derived from this mesh due to power-gating or failure of router/link. We present an algorithm to augment a subset of routers in any n×m mesh (21 routers in a 64-core mesh) with an additional buffer called static bubble, such that any dependence chain has at least one static bubble. We also present the microarchitecture of a low-cost (less than 1% overhead) FSM at every router to activate one static bubble for deadlock recovery. Static Bubble enhances existing solutions for NoC resiliency and power-gating by providing up to 30% less network latency, 4x more throughput and 50% less EDP.

***Keywords*-Networks on chip; Deadlocks; NoC Power Gating; NoC Fault tolerance; Irregular topologies;**

I. Introduction

With increasing core count in chips [1], [2], Network-on-chip (NoC) has today emerged as the de-facto on-chip communication fabric because of its proven scalability compared to the bus based interconnect [3]. Regular topologies such as meshes are preferred on-chip due to their simplicity and ease of layout. However, in future, we should expect to see increasing instances of the on-chip topology becoming irregular. As Fig. 1 shows, this could occur at design time - due to heterogeneous sized big cores, little cores, GPUs, and other accelerators interconnected together-, or at runtime during the lifetime of a chip - due to link/node failures [4], [5], [6], [7], [8], or even due to power-gating of network elements such as routers/datapaths/link-drivers [9], [10], [11], [12]. A key problem in irregular network topologies is that of deadlocks. A deadlock occurs when there is a cyclic buffer dependency chain in the network such that no forward progress can be made (Fig. 1(b)).

*We thank Chia-Hsin Chen and Suvinay Subramanian from MIT for feedback on the motivation for this work, and Swati Gupta from MIT for helping us create a closed form for the static bubble placement algorithm.

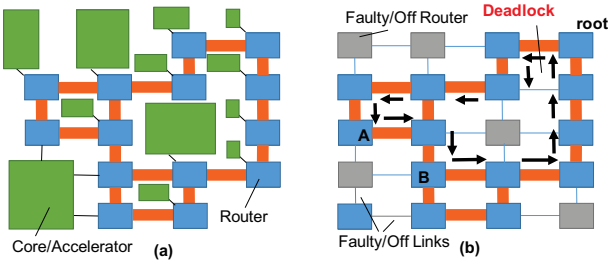


Figure 1: **Irregular On-Chip Topologies due to (a) Heterogeneous SoCs (b) Router or Link failures/gating.**

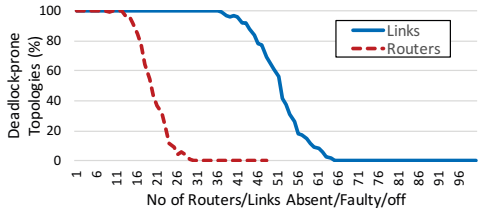


Figure 2: **Percentage of deadlock-prone irregular topologies for a given number of faulty/absent/off routers and links in a 8×8 Mesh. (See Section V-A for simulation methodology).**

The problem of deadlocks becomes more severe in irregular topologies as these topologies offer much less path-diversity compared to a regular topology like a Mesh and thus are more prone to deadlocks. In Fig. 2 we sweep the design space of all resulting topologies as the number of links and routers in an underlying 8×8 mesh substrate are removed, and count the percentage of topologies which are *deadlock-prone*[1], i.e., have cycles in their topology graph. Even with very few random faults, we see that all the probable topologies are deadlock-prone. This evinces the need for providing a solution to this problem for functional correctness of the chip. Beyond 65 link and 30 router faults, the resulting topologies are heavily partitioned and no longer have any cycles. However at this point, the chip itself may be unusable if certain key components such as the memory controller become unreachable.

Deadlocks in irregular topologies is one of the key themes

[1]We obtain this plot by injecting a flit every cycle from every node for a random destination in every topology for a million cycles, and observing if the network deadlocks. Each flit randomly chooses from one of its possible minimal routes without any routing restrictions. A network with zero-faults is also deadlock-*prone* by definition, unless a deadlock-free routing algorithm like XY is chosen.

2378-203X/17 $31.00 © 2017 IEEE
DOI 10.1109/HPCA.2017.44

IEEE computer society

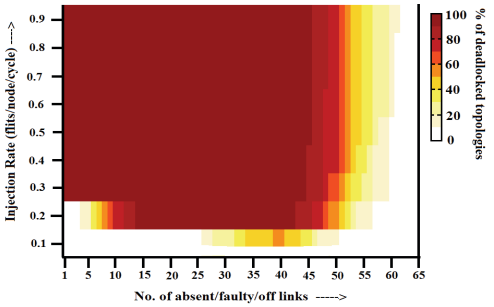


Figure 3: **Heat-map of the cumulative frequency distribution of irregular topologies that deadlock at a particular injection rate for a given number of faulty links with uniform random traffic. (See Section V-A for simulation methodology).**

in works across 3 domains: heterogeneous SoC synthesis [13], NoC resiliency [4], [7], [5], [6], [14], [8], and NoC power-gating [9], [10], [12]. It gets exacerbated in the resiliency and power-gating domains as the irregular topology changes dynamically. The most common solution to address this, is to construct spanning trees over the irregular topology, and route packets in all Virtual Channels (VCs) (or within an escape VC) via the root to *avoid* deadlocks. This approach has **two challenges**:

(1) Routing via the root makes certain routes non-minimal (for instance A's packet being routed via the root to B (10 hops) instead of minimal (2-hops) in Fig. 1, to avoid a cyclic dependency) and reduces path diversity. This adds latency, throughput, and energy penalties at the network-level, which in turn affect full-system runtime, as we show in our evaluations. There is also a huge variance in the potential performance impact, depending on the topology instance and spanning tree.

(2) Constructing an optimized spanning tree across all possible root nodes, while maintaining a high-connectivity, reducing average hops, and providing sufficient bandwidth over the irregular topology is an exponential state-space search, and often requires optimization solvers [15] running in software and 1000s of cycles for reconfiguration [5]. A significant body of work in the NoC resiliency domain is solely focused on coming up with better heuristics for hardware/software co-solutions for this unavoidable problem [4], [5], [8]. Unfortunately, this state space exploration is required every time a new link/router turns off/on or fails, since the optimal/heuristic solution may be very different, which adds to design and verification complexity. And the resulting performance can still be up to 2-4× worse than that with all minimal routes (for Rodinia workloads (Section V)).

In this work, we make the following key observation - despite most irregular topologies being deadlock-prone as Fig. 2 showed, the chances of deadlocks actually occurring at runtime are fairly low. Fig. 3 plots a heat map of the percentage of topologies that deadlock at increasing injection rates with uniform traffic, as a function of increasing number of disconnected links in a 8×8 mesh. Most topologies only start to deadlock at injection rates around 0.1-0.3 flits/node/cycle, which are fairly high since most real workloads on multicores have an order of magnitude lower network injection rates due to high L1 hit rates, as we observed via full-system simulations on a 64-core system with PARSEC 2.0 [16] and Rodinia [17] benchmarks.

Based on this insight, we make a case for deadlock-recovery, rather than avoidance, for heterogeneous/resilient NoCs going forward. The only known techniques for deadlock recovery rely on escape VCs providing a deadlock-free route to drain deadlocked flows [18], [19], [20]. While this can address the first challenge of non-minimal routing prior to deadlocks, it does not address the second challenge of constructing a deadlock-free route over the irregular topology for the escape VCs. Moreover, the "deadlock-free route" may disallow certain links and make parts of the original NoC inaccessible as soon as a deadlock occurs, making this approach infeasible.

In this work, we address both the listed challenges. We present a novel plug-and-play framework for deadlock recovery that can be applied to any mesh topology at design time and guarantees deadlock-freedom across any topology (regular/irregular) derived from this mesh, either statically (to build a heterogeneous SoC) or at runtime (due to faults/gated components). There are no escape channels with routing restrictions. All flows can use minimal routes all the time, removing the spanning-tree construction challenge completely and associated performance/energy penalties. Our framework consists of two components:

(1) A novel algorithm to augment a subset of routers in a mesh (21 in 64 core, 89 in 256 core) with an additional buffer called **Static Bubble** at design time, such that any dependency chain passes through at least one node with a static bubble.

(2) A low-cost FSM microarchitecture embedded in every router that intelligently activates (and deactivates) static bubbles upon detection of a deadlock and performs recovery.

We demonstrate that Static Bubble is a significantly more performance, energy and area efficient solution than conservative spanning-tree or escape-channel based approaches. Across the irregular topology design space sweeping both router and link faults, we demonstrate a 20% latency, up to 4× throughput, and 53% network EDP improvement with synthetic and real (PARSEC and Rodinia) apps. Since our framework is general-purpose and plug-and-play, it is easy to design and verify, and can augment MPSoC topology generators [13] and current state-of-the-art NoC resiliency and NoC power-gating solutions.

The rest of the paper is organized as follows. Section II discusses background and related work. Section III introduces our algorithm for static bubble placement and Section IV presents the microarchitecture and implementation of our deadlock recovery scheme. Section V presents evaluation results and Section VI concludes.

II. Background and Related Work

We present mechanisms for NoC deadlock-freedom employed by recent solutions across resiliency and power gating domains, providing adequate background as necessary.

A. Deadlock Avoidance

Turn Models. Traditionally, regular topologies like Mesh have relied on using deadlock avoidance schemes like dimension-ordered XY routing [21] to achieve deadlock freedom. These schemes, based on the turn model given by Glass and Ni [22], place turn restrictions to avoid cyclic dependencies. For instance, in XY routing, flits are restricted from making a Y (North/South) to X (West/East) turn. However, turn model based schemes rely on having at least two paths, and do not work in a scenario where the topology is irregular and changes dynamically. This is because the turn restrictions may lead to certain core/set of cores not being able to communicate with others leading to a deadlock or a disconnected topology even though healthy and fully functional links are present in the otherwise irregular topology that connect them [23].

Spanning Trees. To overcome this limitation of traditional deadlock avoidance schemes, state-of-the-art NoC designs in the resiliency and power-gating domains construct a spanning tree over the surviving nodes and links and use it to route packets in the irregular topology and avoid deadlocks [4], [5], [8], [6], [12], [10]. Ariadne [4] adapts the topology-agnostic off-chip *up-down* [24] routing algorithm to find deadlock-free paths in the irregular topology. *Up-down* routing enforces strict ordering of nodes in the network by marking the links towards the root node as *up*, those away as *down*, and arbitrarily tagging the equidistant ones [8]. All cyclic dependencies in the network are broken by disallowing turns from a *down-link* to an *up-link*. uDIREC [5] extends this work to cover unidirectional link failures by modifying the methodology of spanning tree construction. Panthre [12], a recent work in the NoC power-gating domain, also leverages up-down routing. Spanning tree based routing, however, makes certain paths non-minimal. **We model this as our first baseline in the evaluations.**

Alternate approaches in Resilient NoCs. Vicis [7] uses a heuristic to determine routing turn restrictions for deadlock avoidance. This heuristic however fails to *guarantee* deadlock freedom as prior works point out [4]. Immunet [6] uses local Bubble Flow Control (BFC) [25] in a ring constructed using the spanning tree of remaining nodes in the network. This work however uses three routing tables and offers poor performance compared to our first baseline [4]. BLINC [8] and Balboni *et al.* [26] use segment routing [23], where the network is divided into segments, each with a different turn restriction. They, however, place a restriction on the number and/or the location of faults and thus cannot handle arbitrary irregular topologies. Wachter *et al.* [27] partition the VCs into 2 classes where each class uses a different deadlock-free turn-restriction based routing (for example west-first for Class I and east-first for Class II). A packet requesting an illegal turn in Class I is put into Class II. However a packet in Class II cannot go back to Class I. This again has the limitation of not being able to guarantee connectivity in any arbitrary irregular topology. Fattah *et al.* [14] use deflection routing [28] on encountering faulty links, but this adds high complexity for achieving deadlock and livelock freedom.

Alternate approaches in Power-gated NoCs. Two recent techniques, Power Punch [11] and CatNap [29], maintain network regularity for routing purposes by switching-on routers that fall in the path of the flits which are routed using deadlock-free XY routing. This is orthogonal to our work as we target irregular topologies (both static and dynamic).

B. Deadlock Detection and Recovery

Traditionally, deadlock detection and recovery has not been a very popular approach for achieving deadlock freedom in regular topologies like a Mesh because deadlock avoidance schemes like XY are easier to implement, while providing adequate path diversity to traffic despite turn restrictions. In irregular topologies, however, as discussed earlier, traditional deadlock avoidance schemes do not work, while spanning-tree based schemes provide non-minimal paths to traffic. Based on our analysis in Fig. 3 that shows deadlocks to be rare in irregular topologies at low loads, we look at deadlock recovery schemes as a possible solution to provide minimal paths to the network traffic and guarantee deadlock freedom at the same time.

DISHA [18] was an early work in the deadlock recovery domain that detected deadlocks using counters present in each buffer queue in the network. Upon the detection of a deadlock, a router would wait to capture a token that circulated through the network all the time using dedicated links. After capturing the token, the packet would be put in a reserved network of buffers (one buffer per router) and routed minimally. After the packet reached its destination, the token would be released, breaking the dependency chain.

DISHA and its recent variants [19], [20] will not work in a dynamically changing irregular topology as they need a path connecting all nodes to circulate the token. Computing this path in a dynamically changing irregular topology is a non-trivial task. In addition, the scheme uses XY routing for its reserved network which as pointed out earlier, cannot guarantee packet delivery between any source and destination pair in an arbitrary irregular topology. If DISHA were to use spanning-tree based routing for the reserved network it would still be inefficient due to the latency and energy overhead of the circulating token.

Ping and Bubble [30] was a subsequent idea to DISHA for off-chip networks that sends a ping from an output port upon detection of a possible deadlock. The ping traverses a control network to trace the deadlocked dependency chain and reserve the output ports at all routers along the route. If

the ping returns, it is a deadlock and an extra deadlock buffer is turned on to drain the deadlock (and turned off when the bubble returns). False positives may occur, i.e., paths may be reserved even if there was no real deadlock, but the design does not handle them. This scheme was proposed for off-chip networks where area and energy are not as precious as in on-chip router implementations. We employ a similar ping for the dependence chain detection phase.

Escape VCs. An alternate approach to spanning trees for achieving deadlock freedom in an irregular topology is to use an *escape-VC* [31]. In this approach, all VCs use deadlock-prone minimal paths to route traffic except one (the *escape-VC*) which uses a deadlock-free routing path. Deadlocked packets drain out using the *escape-VC*. The concept of escape VCs can be used as an avoidance scheme, if packets can actively go into it, or as a recovery scheme if they are enabled upon detection of a deadlock. Escape VCs require an additional VC (buffer) per message class per input port at *every router* in the network. A separate routing table is also required for identifying the deadlock-free escape path in the irregular topology. In addition to the energy and area overhead of the extra buffers and routing tables, prior works [4] have shown that escape VCs cause throughput loss since one VC per message class per input port always needs to be kept reserved. NoRD [9], recently proposed for NoC power-gating, uses a high-latency deadlock-free ring snaking around the network as the escape VC path. Packets are made to enter the *escape-VC* after their misrouted hop count increases by a certain threshold. Router Parking [10] replaces the high-latency ring of NoRD with a spanning tree constructed using up-down routing. Deadlocks are detected using a timer and packets in a deadlock get routed using the escape path. Since escape paths based on spanning trees offer better performance (in terms of lower hop count in the escape path) compared to the ring connecting all routers, **we model this as our second baseline in the evaluations**.

C. Bubble Flow Control in Rings

Bubble Flow Control [25] is a popular flow control technique for ring topologies (or each dimension in a Torus) that avoids deadlocks by ensuring that there is at least one bubble (one empty buffer) in the ring all the time via intelligent injection. In this work we leverage the underlying theory behind this technique: as long as there is one bubble within a dependence chain, there will be no deadlock and forward progress can be made by flits. The *Static Bubble* scheme places a buffer (called static bubble) in a subset of routers in a mesh via a novel placement algorithm at design time that **guarantees the presence of at least one static bubble in any dependency chain in the mesh network**. Upon detection of a deadlock, a static bubble is introduced in the deadlocked ring at runtime and a novel flow-control strategy is run to recover from the deadlock and break the dependence chain. Since Static Bubble can handle any possible dependence chain in the mesh network, any irregular topology based on the mesh topology can be made deadlock-free.

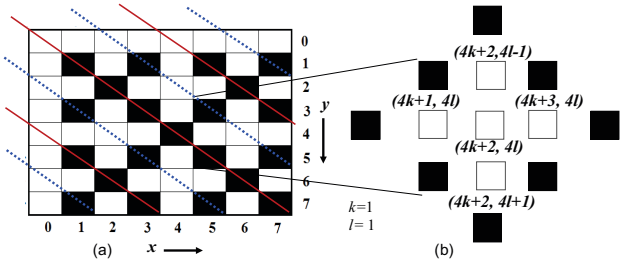


Figure 4: **Placement of static bubbles on a 8x8 mesh at design-time to guarantee a bubble in any possible cycle.**

D. Routing over Irregular Topologies.

Prior works across resiliency and power-gating use a mix of hardware [4], [5] and software [8], [10], [12] techniques to identify connectivity among the currently active routers and links upon detection of a fault or upon turning on/off certain nodes. Disconnected components are discarded, and routing tables are populated at the source NI or at every router. We leverage this rich body of work, and add a routing table at every source NI that populates every packet with a route to its destination. In our spanning tree baseline, this route is spanning-tree based and may be non-minimal, while for escape VC and Static Bubble, this route is minimal (but deadlock-prone). For the escape VC baseline, a spanning tree routing table is used within the escape VCs.

III. Static Bubble Placement

We present an algorithm for the placement of static bubbles in an arbitrary $n \times m$ mesh topology that guarantees that *there will be at least one static bubble in every possible cycle within every possible irregular topology on the underlying mesh*, without having to add a bubble to every router. The algorithm describes a systematic way to decide the placement, but alternate hand-optimized placements, some with fewer static bubbles, are also possible.

For node (x, y) in any $n \times m$ mesh, we add a static bubble if $x > 0$ and $y > 0$ (i.e., no bubbles on the first row and column), and any one of the following conditions hold:
(1) $x \mod 4 \equiv y \mod 4$
(2) $x \mod 4 \equiv 1$ and $y \mod 4 \equiv 3$
(3) $x \mod 4 \equiv 3$ and $y \mod 4 \equiv 1$

Fig. 4(a) shows the placement of 21 static bubbles in a 8×8 Mesh. Visually, the nodes on the solid diagonals satisfy condition (1), while the ones on the dotted diagonals satisfy conditions (2) or (3).

Lemma: *There is at least one static bubble in every possible cycle within the mesh.*

Proof: Starting at node (x,y) in a mesh, any cyclic buffer dependency chain needs to return to the same node (x, y).
Case I. Node (x, y) itself contains a static bubble. The proof is trivial in this case since any cycle going through it will have at least one static bubble.

Case II. Node (x, y) does not contain a static bubble. The coordinates of any such node (except on the first row and column) will be of one of the following 5 forms: $(4k{+}2, 4l)$, $(4k{+}1, 4l)$, $(4k{+}3, 4l)$, $(4k{+}2, 4l{-}1)$, $(4k{+}2, 4l{+}1)$, or the mirror images of this (swap k and l). Fig. 4(b) demonstrates this. All five nodes are bounded by static bubbles. Every hop or turn is an increment or decrement of k or l. It is not possible to make 4 turns (the requirement to get a cycle[2]), without encountering a node that satisfies one of the 3 conditions of the placement algorithm[3].

As a corollary, any irregular topology derived from such an underlying mesh will also have at least one static bubble in any dependence cycle. *Even if the nodes with static bubbles are themselves faulty/turned-off, the dependence chain gets broken and the network will still be deadlock free*. The same static bubble node could be part of multiple dependency chains and resolve deadlocks in all of them (Section IV-B).

The number of static bubbles in a $n{\times}m$ mesh used by our algorithm is:

$$\sum_{k=0}^{[\frac{m}{4}-1]} (min(m-4k, n)-1) + \sum_{l=1, l\epsilon odd}^{[\frac{m}{2}]} \frac{(min(m-2l, n)}{2} + \sum_{p=1}^{[\frac{n}{4}-1]} (min(m, n-4p)-1) + \sum_{r=1, r\epsilon odd}^{[\frac{n}{2}]} \frac{(min(m, n-2r)}{2} \quad (1)$$

where [] represents the Greatest Integer Function (GIF). The bubble count scales linearly with the min. of (m,n) which keeps the complexity of the scheme low.

IV. Deadlock Recovery with Static Bubbles

We define *Static Bubble (SB) routers* as the nodes that the algorithm in Section III picks. In each SB router, we assign *one* extra packet-sized buffer[4] called *static bubble* and *one* special counter with a finite-state machine at design time. When the system starts, all the static bubbles are off; they are turned on by the counter FSM upon detection of a deadlock. The FSM has 6 states, as shown in Figure 5, and manages deadlock detection and recovery. The counter can count[5] from 0 up to two possible thresholds (depending on the FSM state): t_{DD} (**DD = Deadlock Detection**), which is a configurable parameter and t_{DR} (**DR = Deadlock Recovery**), which is set dynamically based on the length of the deadlocked cycle. There are four *special* messages that aid in deadlock detection and recovery: *probe, disable, check_probe* and *enable*.

[2]We assume packets cannot take 180 degree, i.e., u-turns in our design.

[3]The first row and column do not have static bubbles since turns in all directions are not possible and thus fewer bubbles are required.

[4]For simplicity, we size the static bubble to be as deep as data packets (5-flits); though sometimes it may be occupied by 1-flit control packets.

[5]The counters can be off if the entire mesh is ON with no faults/gated components and using say XY routing.

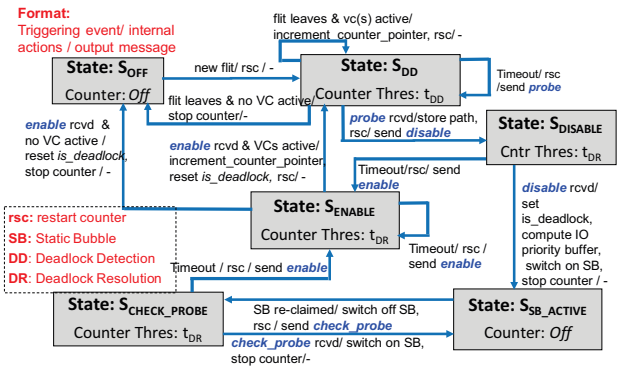


Figure 5: **Finite State Machine of Counter**

A. Walk-through Example

The FSM starts in the S_{OFF} state. When a new flit arrives at the router (at any port other than the local), the state is changed to S_{DD} and points to the VC it occupies with the threshold set to t_{DD}. This deadlock detection threshold is a configurable parameter in every static bubble router. If the flit leaves within the threshold time, the FSM points to the next non-empty VC (VC in active state) in the router in a round-robin manner and the counter is reset and restarted. If all VCs at the router are idle, the FSM goes back to S_{OFF}.

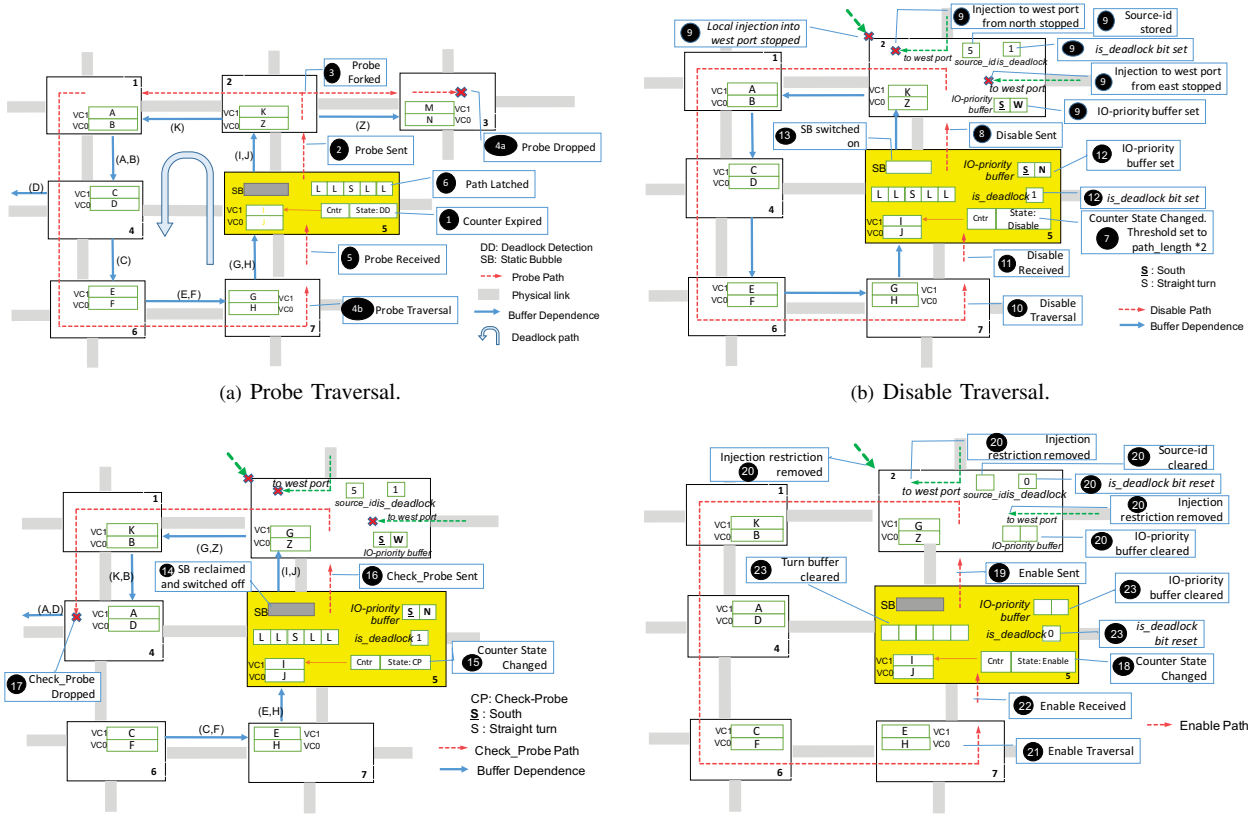
We explain the operation of the FSM and all its states using the walk-through example in Fig. 6. Each buffer dependence in the figure is marked with the packet(s) that want to use it to go to the next hop. As can be seen, there exists a deadlock due to the following cyclic buffer dependency chain:

(A,B)→(C)→(E,F)→(G,H)→(I,J)→(K)→(A,B)

Each VC can hold one packet. We assume Virtual Cut-Through (i.e., packet-sized VCs) and describe all dependencies at the packet level for simplicity, though a flit-level design would also work.

1) Probe Traversal (Fig. 6(a)): The counter at node *5* reaches its threshold in S_{DD} state (**Step 1**) as packet *I* does not leave within the threshold time. Node 5 sends out a *probe* message (**Step 2**) from the North output port (output port for packet *I*) to detect if there is actually a deadlock, and not a false positive due to congestion. The counter is reset and restarts counting with the same threshold t_{DD}.

At each router, if *all* VCs at the input port of the probe are active, the probe is forked out of all the output ports that any of the VCs at that input port are waiting on (except ejection). Otherwise it is dropped. The forking operation creates identical copies of the probe message, so all the information already present in the probe is retained. The router appends the input to output turn (Left Turn (L) or Right Turn (R) or Straight (S)) in each output probe. For instance, packet *K* wants to go West while packet *Z* wants to go East, hence the probe is forked out of these output ports (**Step 3**). Node 2 appends a *L* to the probe that goes West, while it appends a *R* to the probe that went East.



(a) Probe Traversal.

(b) Disable Traversal.

(c) Check_Probe Traversal.

(d) Enable Traversal.

Figure 6: **Walk-through Example**

At node 3, the probe is dropped (**Step 4a**) as packets *M* and *N* are waiting to get ejected, and thus are not part of any deadlock dependence chain.

At nodes 1, 4, 6, and 7, the probe is forwarded out of the south, south, east, and north output ports (**Step 4b**), and the turns L, S, L, and L are appended respectively.

When node 5 receives the probe back (**Step 5**), *the dependence chain is confirmed*, and the path acquired by the probe (L, L, S, L, L) is latched in a special buffer called *Turn Buffer* (**Step 6**).

What if the counter expires before the probe returns, or all copies of the probe got dropped? The counter restarts and resets, and the FSM sends out a new probe. This however cannot continue infinitely. If there is deadlock, the probe would return. Else things may be moving slow due to congestion. Eventually, the congestion will clear-up and the flit would leave.

What if the flit leaves by the time the probe returns? This is just a false positive and does not affect correctness. The next set of actions still occur.

2) Disable Traversal (Fig. 6(b)): Node 5 changes the FSM state to $S_{Disable}$ and the counter threshold to t_{DR} (**Step 7**). t_{DR} is set to 2 times the length of the path brought back by the probe, as the disable message is guaranteed to return within this time, unless it is dropped, as will be explained later in Section IV-C. The same value of t_{DR} is also used by the check_probe and enable messages, as shown later.

A *disable* message is sent (**Step 8**), embedded with the path of the probe and the *node-id* of the sender (node 5).

Upon receiving the disable, each router disables injection of traffic from any other port into the turn specified by the disable. For instance, node 2 (**Step 9**) extracts the first turn field, *Left (L)* from the disable entering at the south input port, and identifies that this corresponds to a south to west turn. It stores this in a *IO_priority buffer* and the *node-id* of the sender in a *source-id* buffer. It also sets an *is_deadlock* bit to *1*. The *is_deadlock* bit, if set, instructs the switch allocator at node 2 to disable injection into the West output port from every input port *except* the South input port. In other words, no other flit is allowed to enter the detected dependence chain. Node 2 then removes the first turn from the disable message and sends it out of the West port.

All nodes along the dependence chain (1, 4, 6 and 7) do the same thing (**Step 10**). At each node the first turn is stripped away from the disable message and it is forwarded out. This ensures that the turn corresponding to the node is always the first turn in the disable message when it is received, speeding up the forwarding circuitry.

Once the disable is received back at node 5 (**Step 11**), it begins deadlock recovery (**Step 12**) by setting its *is_deadlock* bit and the ports for its *IO_priority buffer* to South and North respectively.

The static bubble is now switched ON (**Step 13**), and the FSM moves to S_{SB_Active}. In this state, there is no threshold, and the counter does not increment its count.

We have now introduced a bubble into the deadlocked ring, and disabled any other packets from entering it except the ones already there. This will allow packets in the deadlocked ring to move forward one step. This can be seen by looking at the buffer occupancy change between Figure 6(b) and (c). Once the SB is switched on, this is conveyed to node *7* via standard credit flow control messages (not shown). Packet G from node 7 comes and occupies it. This allows packets *E, C, A* and *K* to each move by 1 hop. Packet G sitting in the static bubble moves to VC1 at node *2* and the static bubble becomes empty again[6].

Why is the disable signal necessary? If we turn on the static bubble without the disable, a new packet may come and occupy it. In this case, we will be back to *a deadlocked ring* without any recovery mechanism.

3) Check_Probe Traversal (Fig. 6(c)): Once the deadlocked ring moves forward one step, the static bubble is re-claimed and switched off (**Step 14**).

At this point, the FSM moves to S_{Check_Probe} (**Step 15**), t_{DR} remains the same as before, and a check_probe message is now sent out along the same path as the disable (**Step 16**). Unlike a regular probe, the check_probe is not forked, but simply forwarded along the same dependency cycle as long as at least one VC is still a part of that dependence chain (indicated by the *IO_priority buffer*).

If the check_probe returns, the static bubble is again switched on and the dependence ring moves forward one more step[7]. In the example, the check_probe is dropped at node 4 (**Step 17**) as both the packets (A,D) in north input port VCs do not want to use the south output port. If the check_probe does not come back, it indicates that the deadlock due to the previously detected dependency chain has been resolved.

4) Enable Traversal (Fig. 6(d)): If the counter reaches its threshold, and the check_probe has not returned, the FSM moves to S_{Enable}, t_{DR} is again retained to be 2 times the length of the path in the *Turn_Buffer* and the counter is re-started (**Step 18**).

An *enable* message is now sent out along the same path (**Step 19**), embedded with the turns and the node-id, just like the disable.

Each router along the path checks if the *node-id* field in the enable matches with its *source-id* buffer, and if it does, clears the *is_deadlock* bit and *IO-priority* buffers (**Step 20 and 21**). This resumes normal traffic flow across all ports since the deadlock has been resolved.

Once the enable returns to the originating node (**Step 22**), it resets the *is_deadlock* bit and clears the *Turn Buffer* and *IO-priority buffer* (**Step 23**). The deadlock has been resolved. The FSM points to another non-empty VC (in a round-robin manner), its state is updated to (S_{DD}), and the counter starts counting up to t_{DD}. If all input ports, other than the local injection port, are empty, the FSM moves to S_{OFF}.

Why do routers need to check if the *node-id* field in the enable matches its *source-id* buffer? It is possible for a router to receive an enable from a different router than the one that sent it the disable, as discussed in Section IV-B.

Next, we discuss how the design guarantees deadlock recovery in the midst of multiple special messages from multiple static bubbles.

B. The Devil is in the Details

A strict priority order (Sec. IV-C) in processing of messages at each node prevents races and ensures all routers in a deadlocked chain maintain a consistent micro-architectural state. At the static bubble node, in addition to the priority order, the FSM provides additional control in processing of messages which ensures that the FSM state cannot be changed by other nodes once it has started the recovery operation. Together these guarantee functional correctness. Here we discuss some of the interesting corner cases.

What happens if there are two or more static bubble nodes in a deadlocked cycle and both send out probes? The static bubble node with the higher id is responsible for resolving the deadlock. If a static bubble node receives a probe from another static bubble node with a lower id, it drops that probe, ensuring that only its own probe, sent earlier or later, will complete the full loop and later send out the disable/enable messages.

What if there are deadlocks in two cycles that are both sharing only one static bubble? The static bubble will successfully resolve the deadlocks one after the other, depending on which direction it sent the probe out first.

What happens if a static bubble node sends a probe, followed by a disable, and then receives a copy of its probe back? This means that there are two dependence cycles that this static bubble is part of, in the same direction. Since the disable for the first one has been sent, the second probe will be dropped. Once the first deadlock resolves, the timeout counter will send out a new probe and resolve the second deadlock if it still exists. Multiple deadlocks can be resolved in parallel by multiple static bubbles, but if the

[6]If packet G does not want to use the north output port of the router after moving to node 5, and is stuck waiting for some other output port, we still have packet I that wants to go north. Packet I would then move north vacating VC1 at node 5. Packet G would move from the static bubble to VC1 and thus the SB would be freed and re-claimed/switched off.

[7]This is an optimization to speed up deadlock recovery. Even if the check_probe did not exist, eventually the counter will again expire, and send out a regular probe, repeating the same process.

same static bubble is part of multiple deadlocked rings, it resolves them serially.

Why do we need to fork the probe? Can we not drop the probe if all VCs at the input port do not want to use the same output port? There may be buffer dependency scenarios where one buffer dependency cycle may depend on another. In the walk-through example, if the probe message was dropped at node *4* and there was such a dependency cycle, the deadlock would never get resolved.

Can a probe loop around infinitely due to buffer dependency? No. Each turn takes 2-bits to encode. Since special messages like probes, disables, etc. are all one flit messages, in a 64 core mesh assuming 128-bit wide links, the probe can only carry a maximum of 59 turns (3-bits for message type + 6 bits for sender node-id). After the turn capacity of the probe is exhausted, it is dropped.

Can false positives (i.e., no real deadlock) lead to enabling of the static bubble? If there is no dependence cycle, the probe will get dropped without returning. There may however be dependence cycles due to congestion which make the probe return. In this scenario, the disable is sent out. If any of the intermediate nodes, including the sender, no longer have the same buffer dependence as earlier, the disable is dropped. In some cases congestion may lead to the probe and disable successfully returning, and the static bubble turning ON. It will let the dependence chain move forward by one and turn OFF again, and so there is no correctness problem.

What if a node receives two probes or disables or enables in the same cycle? Send the one with the higher node-id and drop the other. The FSM at the sender of the dropped probe/disable/enable will handle retransmissions.

Can a non static bubble node receive more than one disable, one after the other? A node will not receive more than one disable signal from the same sender. But it can be part of two dependence cycles and receive disables from static bubble nodes in each cycle. If the is_deadlock bit is already set, the second disable would be dropped.

What happens if a disable gets dropped midway and does not return to the sender node? The static bubble router FSM in $S_{Disable}$ state will timeout and send an enable (Figure 5) since the nodes that the disable went to before getting dropped would have processed it and placed injection restrictions, which now need to be removed.

Can a static bubble node receive a disable/enable when it has itself sent a disable/enable? The same static bubble may be a part of two dependence cycles. In the first cycle it may be the highest id and send a probe and then a disable/enable. In the second cycle it may have the lower id and let the probe from the higher id pass through earlier. Now it receives an enable/disable from that cycle. Since it is in S_{DR} state, it will drop that disable/enable. Effectively the first chain clears its deadlock followed by the second one.

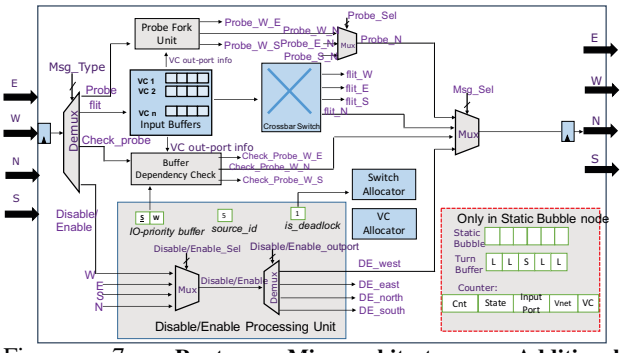


Figure 7: **Router Microarchitecture: Additional Probes/Disable/Enable Circuitry (in gray).**

Which state the does the FSM of a static bubble node go to, if it receives a disable from a higher-id static bubble node? The counter would go to the S_{Off} state. When the enable messages arrives, the counter pointer will be incremented to point to a non-empty VC and its state changed to S_{DD}.

What if a node receives an enable from a node that is different from the node that sent it the disable? This can happen since different VCs at the various ports of a node can be part of multiple dependence chains. If the *node-id* carried by the enable does not match the *source-id* stored at the node, the enable message is not processed and is simply sent out of the port calculated from the turn, not dropped.

C. Router Microarchitecture

We use the following features to make our scheme low-cost and plug-and-play:

- All the special messages: *probe, enable, disable and check_probe* are **not** buffered anywhere in the network. When these messages arrive at a node, they are either sent out of their intended output port or dropped at the node. Thus, the transmission of these special messages is **completely bufferless** which saves area and energy.
- All the special messages use the same links as the regular flits and get higher priority[8]. Thus there are **no additional wires**.
- The processing of the special messages is carried out in parallel units off the critical path of the router pipeline for flits. The only component we add is a mux at the output port that selects between different messages (including flit) during the link traversal stage. Implementation of the design in 32nm DSENT [32] shows that this does not increase our critical path, which is dominated by the switch allocator, and we can still use a state-of-the-art single cycle router [33].

Fixed Delay of Messages. A highly unique and useful feature of our design is that once a probe returns with

[8]During a deadlock the output links are idle since the flits are stuck, and hence leveraging these for the special messages does not have any major performance implications.

the deadlocked path, the delay of disables, check_probes, enable is *fixed*: 2×*path_length*[9]. This is due to the traversals being bufferless, and having higher priority over flits. If the message does not return in this time, it means it was dropped. The FSM (Fig. 5) transitions to an appropriate state and transmits an enable in all these cases without having to worry about race conditions.

If there is only one static bubble in a dependence cycle, things are simple: a router will receive one of the special messages, as the walk-through example showed. However, if there are multiple static bubbles within a dependency cycle, or a router is a part of more than one dependency chain, multiple static bubbles can start sending probes and other signals which arrive at a router in the *same* cycle. In any cycle, a router can receive up to 4 special messages (probe/disable/enable/check_probe) from its 4 input ports leading to 43 different combinations. Since there are no buffers, the router will forward upto *one* message from the output port(s) and drop the rest/all. A strict priority order is enforced to guarantee correct deadlock recovery.

Figure 7 shows the microarchitecture of each router. The router contains standard units like VCs at each input port, Switch Allocator, Virtual Channel Allocator and the Crossbar Switch. The units colored grey show the components that we have added in a standard NoC router. Each incoming special message is demuxed into an appropriate unit.

- **Probe Fork Unit** (per input port): This unit sends one (or none) probes to every output port. It checks the output ports that the VCs at the input port want to use and creates a copy of the probe for each. If multiple probes want to use the same output, *Probe_Sel* selects the one from the highest *node-id* and drops the rest.
- **Enable/Disable Processing Unit** (centralized): This unit is responsible for sending one (or none) disable *or* enable to an output port. It sets (clears) the *is_deadlock* bit, the *IO_priority buffer* and *source-id* buffer from the disable (enable). If both an enable and disable are received for the same output port, then if the *is_deadlock* bit is set, the enable is sent and the disable dropped, else the opposite happens.
- **Buffer Dependency Check:** (centralized): This unit checks if there exists at least one VC at the input port that wants to use the output port stored in the *IO_Priority Buffer*. If yes, the check_probe is forwarded out of that output port, else it is dropped.

After going through these 3 units, at the output mux the following priority is enforced by the *Msg_Sel* signal: *check_probe* > *disable or enable* > *probe* > *flit*[10].

Static Bubble Routers. At a static bubble router, in addition to the above units we also add the FSM, counter, a static bubble and a *Turn_Buffer*, as shown in Fig. 7. A static bubble is like any other VC.

[9]It takes one-cycle to process/forward each message in the router, and one-cycle to traverse the link.

[10]The switch allocator disables the arbitration for this output port if any special message is received for this output port.

Table I: Static Bubble vs. Escape VC

	Static Bubble	**Escape VC**
Operating Mode	Deadlock Recovery	Deadlock Avoidance or Recovery
Pre-Deadlock	Minimal	Minimal
Post-Deadlock	Minimal	Non-Minimal
Control	FSM (Sec IV-C)	Spanning Tree/Ring Routing Table
Additonal Buffers in n×m Mesh	Equation 1 21 in 64 core 89 in 256 core	n×m×5 320 in 64 core 1280 in 256 core
Area Overhead	∼0%	18%

Static Bubble vs. Escape VC. Escape VCs are a powerful framework for deadlock-freedom, and Table I compares Static Bubble against them qualitatively and quantitatively in terms of cost. Performance comparisons are done later in Section V. The deadlock resolution time for escape VCs depends on the misrouting penalty through the tree root; for SB it depends on the length of the deadlocked path, as the disable and enable need to traverse it to resolve the deadlock.

Implementation. We implemented the SB microarchitecture in DSENT [32] at 32nm and observed less than 0.5% area overhead compared to a conventional 1-cycle mesh router (where the buffers and crossbar dominate area), and 18% lower area than escape VCs. Moreover, since SB does not require a deadlock-free spanning tree for its operation unlike the deadlock avoidance schemes or escape VC (which needs it for its escape path), we can reduce reconfiguration cost significantly compared to prior works [4], [8].

V. Evaluations

A. Simulation Methodology

State-space Exploration with Fault Model. For all our simulations, we assume an underlying 8×8 mesh. We develop two models, where we randomly inject faults in the network and map them to link failures in one, and router failures in the other, and remove these components from the topology graph. Our fault model is in line with previous works in the resiliency domain [14], [5], [7]. For simplicity, we call these faults throughout this section, though they can can also be viewed as power-gated link-drivers or routers.

For each fault number, since the state-space of possible topologies is exponential, a full exploration is infeasible. Instead, we increase the number of topologies till the average value of the trend we wish to study (rate at which they deadlock, average network latency, throughput, application runtime, etc) stabilizes[11].

A key observation that Fig. 2 and Fig. 3 show is that at high number of faults/power-gated links or routers, the

[11]Because of the high symmetry of the mesh topology, many of the generated irregular topologies gave similar results and the trend stabilized within 100 topologies in most cases.

Table II: System Configuration

Network Configuration	
Topology	8x8 Mesh
Routing	Source Routing (Sec II-D)
Num VCs	3 Vnets, 4VCs per VNet per port
Latency	1-cycle router + 1-cycle link
Flit Size	128b
SB t_{DD}	34
Fault Model	Random [14], [5], [7] (links & routers)
Traffic (using gem5 [34] + Garnet [35])	
Synthetic	Uniform Random and Bit-Complement with mix of 1-flit and 5-flit packets
Multi-threaded	PARSEC [16] running on HyperTransport [36] protocol
Heterogeneous	Rodinia [17] traces

topologies become highly fragmented lacking cycles and thus do not deadlock. At low number of faults/power-gating, though, which is expected to be the common case, a significant number of topologies can deadlock.

Routing Algorithms. As described in Section II-D, we model a routing table at each NIC that populates the route in each packet, across our baselines and Static Bubble, leveraging prior work on routing over irregular faulty NoCs [4], [8], [5]. With uniform random traffic, if the destination is not reachable (due to disconnected topologies), the packet is simply dropped. With real application traffic (PARSEC2.0 and Rodinia), the application is mapped on cores that are part of a connected sub-network, and only those topologies that do not disconnect the Memory Controllers are considered.

B. Configuration and Baselines

We use the gem5 [34] full-system simulator with the Garnet [35] network model for our cycle-accurate simulation studies. Network energy and area is estimated using DSENT [32]. We model 32nm and 2GHz. Table II lists the system configuration.

We use the following two baselines across all our simulations that reflect the state-of-the-art.

Deadlock Avoidance with Spanning Tree. We model a deadlock avoidance scheme using an *up-down* routing similar to state-of-the-art works in NoC resiliency [4], [5], [8], [6] and power-gating [12]. We assume zero cycles to reconfigure for spanning tree construction, though this cost is in 1000s of cycles [4], [8]. All packets come embedded with a deadlock-free route (Section II-D).

Deadlock Recovery with escape VCs. We model a deadlock recovery scheme, where upon detection of a deadlock, packets move to an escape VC and use a deadlock-free route within that [10], [9]. Packets inside regular VCs use minimal routes set by the source (Section II-D) with 1-cycle routers, while escape VCs use a per-router routing table configured with a spanning tree. Again we assume zero cycles to reconfigure for spanning tree reconfiguration.

C. Network Performance and Energy Sweep

We start with a performance sweep of the entire design-space of irregular topologies with synthetic traffic.

Low-load latency. Fig. 8 plots the average latency benefit that Static Bubble (SB) provides over the baselines at low-loads for (a) uniform-random and (b) bit-complement traffic patterns. Since deadlocks do not occur at this point, both escape VC and SB show the same performance, providing around 22% latency savings with uniform random and 15% with bit-complement traffic on average across all the topologies, with low number of link and router faults. This reiterates our motivation that restricting path diversity in an already irregular topology that the baselines do is not very robust. Beyond 53 link faults, the topologies become highly disconnected with no cycles and very little path diversity in the topology itself, so minimal routes show similar performance to the spanning tree.

Throughput. Fig. 9 plots the average network saturation throughput as a function of link and router faults. SB provides up to 3.5 to 4X throughput benefit over a spanning tree. This is due to higher path diversity that the tree limits. We also observe a 1.2 - 1.3X higher throughput than escape VCs. This is due to escape VCs having to reserve one VC solely for deadlock recovery at all nodes, limiting throughput at high loads, while SB reserves an extra VC only at a few nodes. At around 21 router faults, the performance of all 3 designs is very similar. This is because there are very few deadlock-prone topologies at this fault number. Beyond this, the network becomes fragmented, and there are cycles within each fragment which leads to performance improvements.

Energy. Fig. 10 plots the network energy using DSENT [32] as routers are turned off. Across the design space, we observe about 10% energy reduction compared to spanning tree, and 20% compared to escape VC which are modeling state-of-the-art power-gating NoC designs [12], [10], [9]. In addition, we see 22% leakage energy improvement at low router faults. SB can thus be used to increase the static energy savings of existing works in NoC power-gating domain. At high router faults, leakage becomes a larger part of the overall energy as the topology becomes fragmented and so the average hop count reduces, leading to a dip in dynamic energy, but SB still provides 20% savings.

D. Deadlock Detection Threshold Sweep

Next, we study the impact of the Deadlock Detection threshold (t_{DD}), the only configurable parameter in our design. Intuitively, a very low value of t_{DD} will result in a lot of probes being sent out while a very high value may delay deadlock resolution. In practice, however, we observed that at low (0.01 flits/node/cycle) and medium (0.1 flits/node/cycle) loads, even with a t_{DD} of just 5-cycles, no probes were sent out since a flit would leave within this time. Fig. 11 sweeps t_{DD} at high loads, when the network is deadlock-prone (Fig. 3), for 10K cycles. The NoC has 20 router faults and the average values across all topologies are plotted. A very low value of t_{DD} results in over 4000 (0.4 per cycle) probes across the NoC. As t_{DD} increases, there

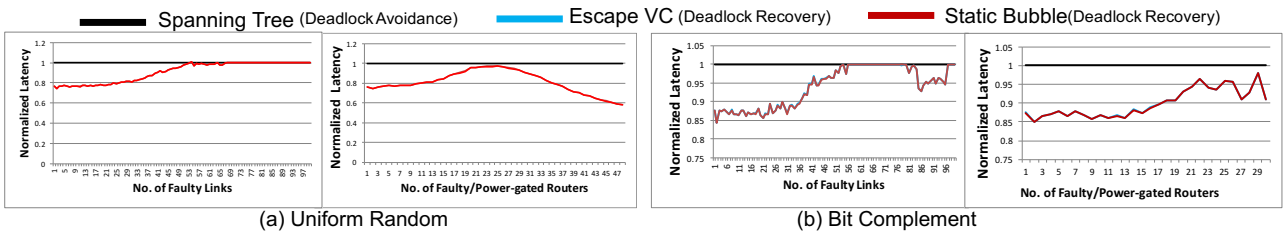


Figure 8: **Avg and Max network latency improvements demonstrated by Static Bubble, normalized to Spanning Tree, across the irregular topology space with uniform random traffic at low-loads.**

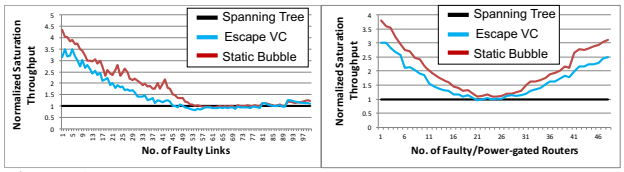


Figure 9: **Average network throughput, normalized to Spanning Tree, of all designs across the irregular topology space with uniform random traffic.**

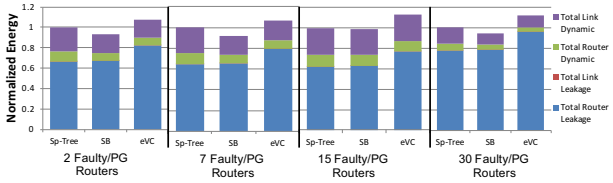


Figure 10: **Average Network Energy.**

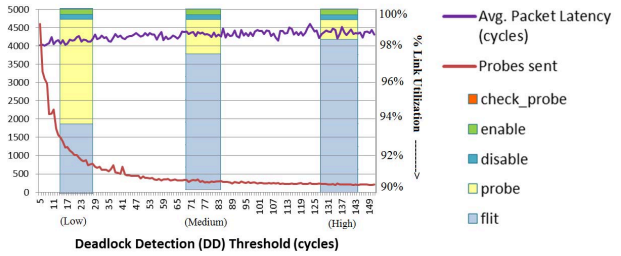


Figure 11: **Deadlock Detection Threshold Sweep.**

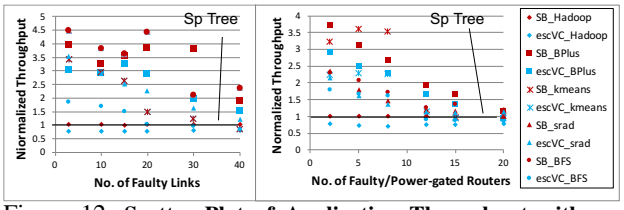


Figure 12: **Scatter Plot of Application Throughput with escape VC and SB, normalized to Spanning Tree, for Rodinia workloads, with increasing link and router faults.**

is an exponential decline in the number of probes being sent out and saturates to about 200 (0.02 per cycle). While more probes steal bandwidth from flits, it turns out that probes are only being sent when the network is truly deadlocked in which case the links are idle. At this point the network is saturated which is reflected by the extremely high packet latency. Recall that the number of probes does not affect the

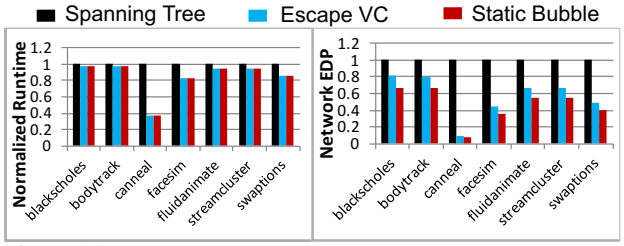


Figure 13: **(a) Application Runtime and (b) Network EDP for PARSEC with 4 faults.**

functional correctness of the design; it just affects the time to detect deadlocks and link energy consumption. While we did not see any noticeable difference in the average latency of a flit as the threshold is swept since link traversal by probes is orders of magnitude lower than by flits, a slight improvement in packet latency is seen at low t_{DD} as the deadlocks get detected faster.

Fig. 11 also shows the link utilization of the message classes as t_{DD} is varied. Link utilization by probes falls from 5% when t_{DD} is low to 2% at medium t_{DD} to 1.5% at high t_{DD}. The other special messages have constant link utilization (enable(0.45%), disable(0.45%), check_probe(0.6%)) at all values of t_{DD} thus showing that these messages use the links only in case of a deadlock and thus do not steal bandwidth from flits which remain the dominant users of the NoC (>93% utilization across all values of t_{DD}).

E. Real Applications.

Fig. 12 plots the application throughput of Rodinia [17] benchmarks as a function of link and router faults across the topology space, normalized to the Spanning Tree. At low faults, the system with SB is consistently higher performing than both escapeVC, and Spanning Tree, by up to 2-4X. The only exception is *Hadoop* which shows similar performance with all systems due to high collective traffic which saturates all the NoCs very early. At high link fault rates, *BPlus* and *SRAD* show throughput improvements with SB, while the performance of others drops. At 20 router faults, all designs perform almost identically as very few connected topologies to run Rodinia exist at this fault rate, and there is hardly any path diversity that minimal routes can exploit. We observed deadlock occurrence (and resolution) in *Hadoop*, *BFS*, and *SRAD* for some instances of the topologies at low faults.

Fig. 13 runs a full-system 64-core simulation of PARSEC with 4 link faults. Both escape VC and SB provide ∼15% reduction in application runtime, on average, validating our case for deadlock recovery solutions over Spanning Tree solutions. Results with 32 router faults were very similar. PARSEC workloads have very low network injection rates, and hence no deadlocks were observed which is why SB and escape VC have identical performance. The benefit of SB over escape VC is visible in Fig. 13(b) where the network EDP is plotted. SB has a 53% lower EDP than Spanning Tree, and a 17% lower EDP than escape VC.

VI. Conclusion

Current solutions for deadlock freedom for irregular topologies (that may occur due to NoC faults or power gating) require expensive spanning tree constructions and non-minimal routing over them. The alternative of using escape VCs still requires such a tree for providing a deadlock-free escape path. We perform a state space exploration and conclude that while most irregular topologies are deadlock-prone, the actual occurrence of deadlocks at runtime is rare. We present a plug-and-play solution for deadlock recovery, known as Static Bubble, that augments a set of routers in a mesh with an extra buffer via a novel algorithm that guarantees the existence of at least one static bubble in any dependency cycle. We provide a low-cost mechanism for deadlock recovery and demonstrate performance gains and energy reduction over state-of-the-art solutions. Static Bubble does not require any tree construction and can augment current solutions in the space of heterogeneous design, NoC resiliency and power-gating.

References

[1] Y. Hoskote *et al.*, "A 5-GHz mesh interconnect for a teraflops processor," *IEEE Micro*, vol. 27, no. 5, pp. 51–61, Sept. 2007.

[2] D. Wentzlaff *et al.*, "On-chip interconnection architecture of the tile processor," *IEEE Micro*, vol. 27, no. 5, Sep. 2007.

[3] W. J. Dally and B. Towles, "Route packets, not wires: On-chip inteconnection networks," in *DAC*, 2001.

[4] K. Aisopos *et al.*, "ARIADNE: agnostic reconfiguration in a disconnected network environment," in *PACT*, 2011.

[5] R. Parikh and V. Bertacco, "udirec: Unified diagnosis and reconfiguration for frugal bypass of noc faults," in *MICRO*, 2013.

[6] V. Puente *et al.*, "Immunet: A cheap and robust fault-tolerant packet routing mechanism," in *ISCA*, 2004.

[7] D. Fick *et al.*, "A highly resilient routing algorithm for fault-tolerant nocs," in *DATE*, 2009.

[8] D. Lee *et al.*, "Brisk and limited-impact noc routing reconfiguration," in *DATE*, 2014.

[9] L. Chen and T. M. Pinkston, "Nord: Node-router decoupling for effective power-gating of on-chip routers," in *MICRO*, 2012.

[10] A. Samih *et al.*, "Energy-efficient interconnect via router parking," in *HPCA*, 2013.

[11] L. C. *et al.*, "Power punch: Towards non-blocking power-gating of noc routers," in *HPCA*, 2015.

[12] R. Parikh *et al.*, "Power-aware nocs through routing and topology reconfiguration," in *DAC*, 2014.

[13] S. Murali and G. De Micheli, "Sunmap: a tool for automatic topology selection and generation for nocs," in *DAC*, 2004.

[14] M. Fattah *et al.*, "A low-overhead, fully-distributed, guaranteed-delivery routing algorithm for faulty network-on-chips," in *NOCS*, 2015.

[15] M. Kinsy *et al.*, "Application-aware deadlock-free oblivious routing," in *ISCA*, 2009.

[16] C. Bienia *et al.*, "The PARSEC benchmark suite: Characterization and architectural implications," in *PACT*, 2008.

[17] S. Che *et al.*, "Rodinia: A benchmark suite for heterogeneous computing," in *IISWC*, 2009.

[18] K. V. Anjan and T. M. Pinkston, "An efficient, fully adaptive deadlock recovery scheme: DISHA," in *ISCA*, 1995.

[19] Y. Ho Song and T. M. Pinkston, "A progressive approach to handling message-dependent deadlock in parallel computer systems," *IEEE TPDS*, vol. 14, no. 3, Mar. 2003.

[20] A. Lankes *et al.*, "Comparison of deadlock recovery and avoidance mechanisms to approach message dependent deadlocks in on-chip networks," in *NOCS*, 2010.

[21] L. M. Ni and P. K. McKinley, "A survey of wormhole routing techniques in direct networks," *IEEE Computer*, Feb. 1993.

[22] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," *J. ACM*, vol. 41, no. 5, Sep. 1994.

[23] A. Mejia *et al.*, "Segment-based routing: An efficient fault-tolerant routing algorithm for meshes and tori," in *IPDPS*, 2006.

[24] M. D. Schroeder *et al.*, "Autonet: A high-speed, self-configuring local area network using point-to-point links," *J-SAC*, vol. 9, no. 8, 1991.

[25] V. Puente *et al.*, "The adaptive bubble router," *J. Parallel Distrib. Comput.*, vol. 61, no. 9, Sep. 2001.

[26] M. Balboni, J. Flich, and D. Bertozzi, "Synergistic use of multiple on-chip networks for ultra-low latency and scalable distributed routing reconfiguration," in *DATE '15*, 2015.

[27] E. Wachter *et al.*, "Topology-agnostic fault-tolerant noc routing method," in *DATE*, 2013.

[28] C. Fallin *et al.*, "Minbd: Minimally-buffered deflection routing for energy-efficient interconnect," in *NoCS*, 2012.

[29] R. Das *et al.*, "Catnap: Energy proportional multiple network-on-chip," in *ISCA*, 2013.

[30] Y. H. Song and T. M. Pinkston, "A new mechanism for congestion and deadlock resolution," in *ICPP*, 2002.

[31] J. Duato, "A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 6, no. 10, Oct. 1995.

[32] C. Sun *et al.*, "Dsent - a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling," in *NOCS*, 2012.

[33] S. Park *et al.*, "Approaching the theoretical limits of a mesh noc with a 16-node chip prototype in 45nm SOI," in *DAC*, 2012.

[34] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, Aug. 2011.

[35] N. Agarwal *et al.*, "GARNET: A detailed on-chip network model inside a full-system simulator," in *ISPASS*, 2009.

[36] A. Ahmed *et al.*, "AMD Opteron shared memory MP systems," in *14th Hot Chips Symposium*, August 2002.