

DEADLOCK RECOVERY IN ON-CHIP INTERCONNECTION NETWORKS

A Dissertation
Presented to
The Academic Faculty

By

Aniruddh Ramrakhyani

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

May 2017

Copyright © Aniruddh Ramrakhyani 2017

DEADLOCK RECOVERY IN ON-CHIP INTERCONNECTION NETWORKS

Approved by:

Dr. Tushar Krishna, Advisor
School of Electrical and Computer
Engineering
Georgia Institute of Technology

Dr. Ada Gavrilovska
School of Computer Science
Georgia Institute of Technology

Dr. Sudhakar Yalamanchili
School of Electrical and Computer
Engineering
Georgia Institute of Technology

Date Approved: April 24, 2017

Do the difficult things while they are easy and do the great things while they are small. A journey of a thousand miles must begin with a single step.

Lao Tzu

This work is dedicated to my parents, my grandparents and my sister.

ACKNOWLEDGEMENTS

I have been fortunate to receive the guidance and mentorship from some of the brightest people in my field who have helped shape the ideas contained in this work. Without their help and guidance this work wouldn't have the same intellectual richness as it does now. I would like to start by thanking my advisor, Prof. Tushar Krishna for mentoring and guiding me and being a constant source of support throughout my graduate school life. He was always available to help me out when I was stuck with the problems and always inspired me to strive harder. His role has been pivotal in the development of my intellectual abilities for which I thank him from the bottom of my heart. I also extend my sincere gratitude to Prof. Ada Gavrilovska and Prof. Sudhakar Yalamanchili for serving on my Masters Thesis committee and providing feedback on my work. I am thankful to Prof. Paul Gratz from Texas A&M University for his guidance in architecting the deadlock recovery scheme and suggesting ingenious ways to solve the corner cases of the design. I would also like to thank Swati Gupta from MIT for her help in deriving a closed form proof for the static bubble placement algorithm. I am grateful to Chia Hsin and Suvinay Subramanian from MIT for their feedback on this work.

TABLE OF CONTENTS

Acknowledgments	v
List of Tables	viii
List of Figures	ix
Chapter 1: Introduction	1
1.1 Definition of key NoC Terms:	3
1.2 The Origin of Irregular on-chip Topologies	4
1.2.1 Power-gating of network elements	4
1.2.2 Waning Silicon Reliability	5
1.2.3 Heterogeneous SoCs	6
1.3 The Problem	6
Chapter 2: Background and Related Work	11
2.1 Assumptions	11
2.2 Definitions	12
2.3 Deadlock Avoidance	12
2.3.1 Turn Model	13
2.3.2 Spanning Trees	14

2.3.3	Alternate approaches in Resilient NoCs	15
2.3.4	Alternate approaches in Power-gated NoCs	16
2.4	Deadlock Detection and Recovery	16
2.4.1	DISHA	17
2.4.2	Escape-VC	17
2.5	Bubble Flow Control (BFC) in Rings	19
2.6	Routing over Irregular topologies	19
Chapter 3:	Static Bubble	20
3.1	Static Bubble Placement Algorithm	20
3.2	The Recovery Algorithm	23
3.2.1	Walk-through Example	24
3.3	Special Cases and Design Details	30
3.4	Router Microarchitecture	35
Chapter 4:	Evaluations	40
4.1	Simulation Methodology	40
4.2	Configuration and Baselines	41
4.3	Network Performance and Energy Analysis	42
4.4	Deadlock Detection Threshold Sweep	46
4.5	Real Applications	47
Chapter 5:	Conclusion	50
References		53

LIST OF TABLES

3.1	Special Message Types	38
3.2	Static Bubble vs. Escape VC	39
4.1	System Configuration	41

LIST OF FIGURES

1.1	Figure showing packet path from Source(S) to Destination(D) in green and the turn restrictions in red for XY routing.	2
1.2	Irregular topologies on-chip due to (a) Heterogeneous SoCs (b) Router or link failures/gating.	7
1.3	Dynamically changing irregular topology due to Power-gating of network elements.	8
1.4	Percentage of deadlock-prone irregular topologies for a given number of faulty/absent/off routers and links in a 8x8 mesh.	9
1.5	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.	10
2.1	Figure showing the limitations of XY routing in irregular topologies. Here the source S cannot route to destination D as the only path that connects them involves turns that are not allowed by the XY routing algorithm. . . .	14
3.1	Placement of static bubbles on a 8x8 mesh at design-time to guarantee a bubble in any possible cycle.	22
3.2	Finite State Machine of the static bubble counter.	23
3.3	Walkthrough: Probe Traversal	24
3.4	Walkthrough : Disable Traversal	26
3.5	Walkthrough : Check_Probe Traversal	28
3.6	Walkthrough : Enable Traversal	29

3.7	Router Microarchitecture : Additional Probes/Disable/Enable Circuitry in grey.	35
4.1	Average 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.	43
4.2	Average and max network latency improvements demonstrated by Static Bubble, normalized to Spanning-tree, across the irregular topology space with bit-complement traffic at low-loads.	43
4.3	Average network throughput, normalized to Spanning-tree, of all designs across the irregular topology space with uniform random traffic.	44
4.4	Average Network Energy.	45
4.5	Deadlock Detection threshold t_{DD} sweep.	46
4.6	Scatter plot of application throughput with escape-vc and static bubble, normalized to spanning tree scheme, for Rodinia workloads, with increasing link and router faults.	48
4.7	(a) Application Runtime and (b) Network EDP for PARSEC with four link faults.	48

SUMMARY

The demise of Dennard Scaling and the continuance of Moore's law has provided us with shrinking chip dimensions and higher on-chip transistor density at the cost of increasing power density. Chips today are highly power-constrained and often operate close to their melt-down energy thresholds. To avert the thermal meltdown of chip, designers use intelligent power-gating techniques. Here, the mode of operation is to power-up only a subset of IP blocks at a time. In addition to the power-density problem, decreasing transistor size has led to decreasing silicon reliability which has led to increasing instances of on-chip faults. Both these effects lead to irregular on-chip topologies that change at runtime. Chip designers and architects today face the problem of routing packets over a dynamically changing irregular topology without sacrificing performance and more importantly without running into routing deadlocks.

Another trend in the semi-conductor industry that has contributed to the significance of this problem is the increasing use of heterogeneous System-on-Chip (SoC). SoCs in most instances are tailored to the application needs. To maximise performance, these SoCs employ custom-built irregular topologies to connect IP blocks. SoC designers have to run a large number of simulations to understand the network traffic flows of the application it is being designed for. These simulation studies are carried out to ensure the absence of routing deadlocks. This leads to increase in design time and consequently the time to market, leading to increase in costs and decrease in profits.

Prior works in power-gating, resiliency and SoC design domains have addressed the routing deadlock problem by constructing a spanning-tree over the irregular topology and using it either as a deadlock avoidance mechanism (spanning-tree based routing) or as a deadlock-recovery mechanism (escape-vc) to route packets. However, this spanning-tree

based solutions leads to significant loss in throughput and performance as shown in this work. In addition, a new spanning-tree construction is required every time the topology changes due to a fault in or power-gating of a network element.

In this work, a new deadlock recovery framework called *Static Bubble* is proposed to achieve deadlock freedom in a static or dynamically changing irregular on-chip topology that doesn't require any tree construction and thus is able to eliminate any overhead or limitations associated with the spanning-tree based solutions. Compared to the other state of the art works, *static bubble* provides upto 30% less latency, 4x more throughput and 50% less network EDP at less than 1% hardware overhead.

CHAPTER 1

INTRODUCTION

Shrinking of transistor dimensions as a result of Moore's law [1] enabled us to pack an increasingly large number of transistors on chip. As architectures moved from a single core to multi-core era, it drove the need for a fast and efficient communication fabric that could be used to connect together the communicating cores and the memory system. The communication fabric also known as Network-on-chip (NoC) has today become a key determinant of chip performance. Consequently, there has been a significant amount of research literature published in recent years that deals with the design and synthesis of low-latency and high throughput interconnects. The key problems faced in the design of interconnects are that of topology design, routing algorithm and router micro-architecture.

In the early days of multi-core architectures when the core count on-chip was less than four, *bus* interconnect was the most dominant interconnect topology that was used to connect the cores and memory system in both academic and commercial designs. However, as the core count increased and the demand for bandwidth went up due to the emergence of better memory technologies like DDR3 [2], the bus interconnect proved to be insufficient to meet the challenge. Architects then moved away from the bus based interconnect to a more scalable topology : the *mesh* interconnect. The mesh topology arranges the cores and the directories in a $m \times n$ grid. This today has emerged as the most dominant interconnect topology in both academic (MIT RAW [3], Princeton Piton [4], MIT Scorpio [5]) and commercial (Intel Xeon, Intel Xeon phi, Intel Teraflops, Tilera Tile64, Intel SCC) designs. Besides being more scalable, mesh provides abundant path diversity (important for load balancing and resiliency) and is easy to layout.

The move from a bus based interconnect to a mesh based interconnect also drove the need for efficient routing algorithms to route packets. In a bus based interconnect, every

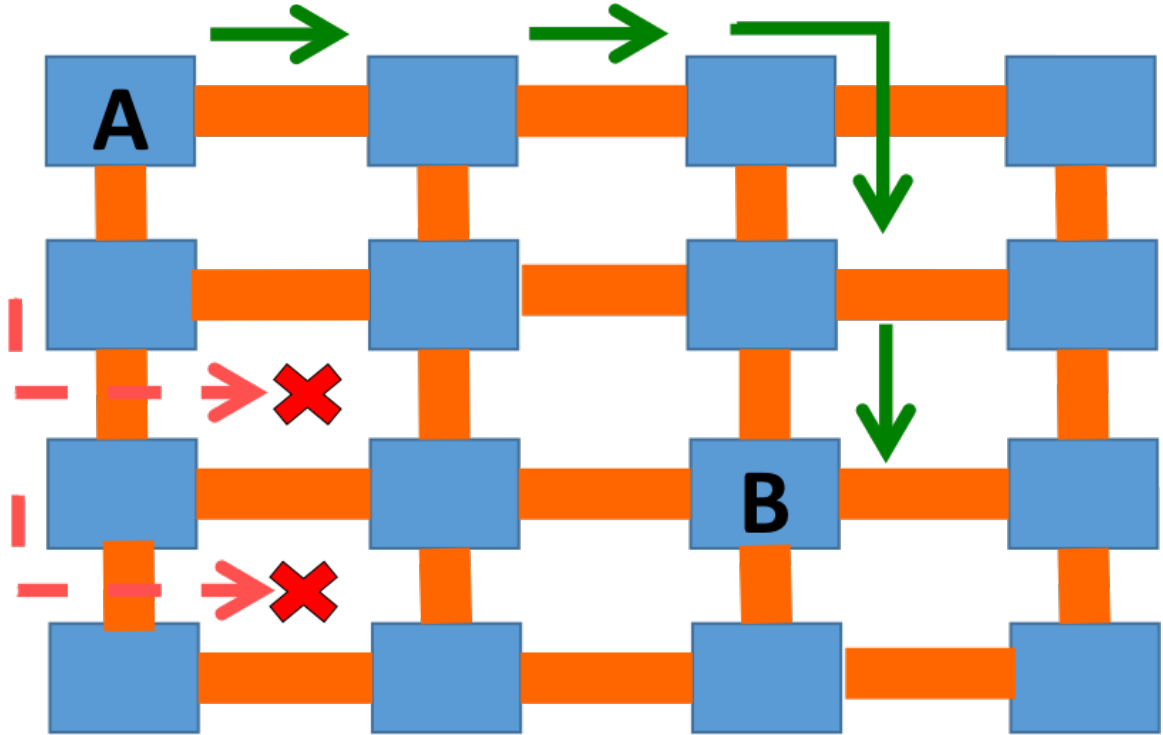


Figure 1.1: Figure showing packet path from Source(S) to Destination(D) in green and the turn restrictions in red for XY routing.

message was broadcasted on the bus. This was inefficient as a broadcast required too much energy and it allowed only one set of nodes to communicate at a given time. The key metrics for evaluating a routing algorithm are : packet latency, peak throughput, ease of implementation and deadlock freedom. Due to the strict energy constraints on-chip, architects preferred simpler routing algorithms over fancy ones. Turn-restriction based routing algorithms that relied on deadlock avoidance became popular with chip architects as they were easy to implement and provided deadlock freedom at the same time. One such popular turn-restriction based algorithm *XY routing* restricts all *Y* to *X* turns and only allows *X* to *Y* turns. The path taken by a packet using the XY routing algorithm for a Source (S) to a destination (D) is shown in fig 1.1 on a (4 * 4) mesh . The allowed turns are shown in green while the disallowed turns are shown in red.

1.1 Definition of key NoC Terms:

Before we dive deep into the details of the proposed solution for the problem of deadlocks in irregular on-chip topologies, it is imperative to define key NoC terms to facilitate discussion in the following sections.

- **Topology:** An arrangement of cores, directory and memory system and their connections.
- **Virtual Channel (VC):** Buffers added to the router to avoid head-of-line blocking. This allows sharing of router links on a per-flit basis.
- **Virtual Cut Through (VCT):** A routing style where the VCs are as large as the largest packet size.
- **Deadlock:** A cyclic buffer-dependence such that no forward progress is possible.
- **Routing Algorithm:** An algorithm used to derive a path between any source and destination pair in the network.
- **Deadlock Avoidance:** A class of routing algorithms that rely on placing routing restrictions to prevent a network deadlock.
- **Deadlock Recovery:** A class of routing algorithms that allow network deadlocks to happen and then recover out of them rather than avoiding deadlocks from forming in the first place.
- **Injection Rate:** Number of packets injected into the network by a node per cycle.
- **Throughput:** Number of packets delivered per cycle.
- **Wormhole Routing:** A class of Routing algorithms where the VC size is less than the size of largest packet.

1.2 The Origin of Irregular on-chip Topologies

In this work I address the problem of routing deadlocks in both dynamically changing and static irregular topologies. Before diving into the causes for deadlocks in irregular topologies, we need to understand how irregular topologies are created/occur on-chip. In this section, I provide a background on how irregular topologies occur in on-chip networks.

1.2.1 Power-gating of network elements

Moore's law, given by Gordon Moore the co-founder of Intel, states that the transistor density on a chip would double every two years. This law has been the reason for exponential increase in compute capability of machines over the years. A key contributor to the success of Moore's law has been *Dennard Scaling*. Dennard Scaling [6] given by Robert H. Dennard in 1974 states that to keep the electric field constant in a transistor, the channel length and operating voltage needs to scale down by α and the doping density needs to scale up by α . Both the Moore's law and Dennard scaling provided architects with faster and more number of transistors on-chip that improved the processing power of the chip. Around 2006 however, Dennard Scaling broke-down as the operating voltage of the chip came too close to the threshold voltage of the transistor and thus it was no longer possible to reduce the operating voltage. Decreasing feature size of transistors with almost constant operating voltage led to a large increase in the leakage energy of transistors. As leakage became a large fraction of the total power consumption of chip, it no longer was possible to keep the whole of the chip "on" or powered-up at the same time due to the fear of thermal runaways. Thus as technology scaled, a large part of the chip was off or *dark* and it came to be called *dark silicon* [7].

To decrease the leakage power consumption of the chip, circuit designers started to *power gate* chip components when not in use. In power-gating [8], the power supply of a chip component is cut by inserting a high threshold voltage transistor between the supply

and the Pull-up Network (PUN) or ground and Pull Down Network (PDN). The PUN or PDN themselves can contain low threshold voltage transistors and often do, as low threshold transistors have faster switching speeds. However, the penalty for using power-gating to reduce static leakage energy is paid in terms of circuit switch-on time every time a power-gated element is powered-up. This latency can be in hundreds of cycles depending on where in circuit the high threshold voltage transistor has been inserted. An interesting consequence of power-gating is that it leads to irregular topology on-chip at run-time. As there is no guarantee on which network element will or will-not be power-gated, the irregular topology can be any arbitrary configuration derived from the underlying mesh topology. In addition, the problem gets exacerbated as the irregular topology will be changing dynamically depending on the power-gating decisions made by the controller.

1.2.2 Waning Silicon Reliability

Silicon devices today are becoming less and less reliable as technology scales to smaller feature size. The unreliability of silicon has been attributed to manufacturing defects and variability, aging, chip design complexity and limitation of validation techniques. According to a commentator from Intel in [9], "within a decade we will see 100 billion transistor chips. However, 20 billion of them will fail in manufacture and a further 10 billion will fail in the first year of operation". Thus, a 20-30% failure rate underscores the fact that component failure over the lifetime of the chip is going to be a common occurrence in future.

This reduced silicon reliability and increased likelihood of faults has important consequences for the NoC. Random faults in this communication fabric can lead to highly irregular and disconnected topology even though the initial topology was a regular mesh. As these faults are random and can occur anytime during the lifetime of the chip, the topology is going to change dynamically providing no guarantees about the structure of the resulting topology. Routing over such a highly irregular topology is highly prone to deadlocks as it

would often involve packets taking convoluted cyclic routes.

1.2.3 Heterogeneous SoCs

Heterogeneous System-on-chip pack together dissimilar co-processors in a single chip to provide enhanced performance compared to homogeneous chips that use the same type of cores. These systems are designed and optimized for specific applications and take advantage of the fact that application requirements are known in advance. The interconnects designed to connect different IP blocks in such system are often custom-made to take advantage of the known traffic flows. An example of a custom-made interconnect would be *Spidergon* interconnect by ST Microelectronics. The topology used in these systems is often irregular and only connects IP blocks that need to communicate with each other. This topology due to its irregular nature would require convoluted routing paths to send messages and thus is prone to network deadlocks. SoCs designers spend considerable time in studying network traffic flows of applications to make sure no deadlocks ever occur. This, as mentioned before, increases design effort, cost and the time to market. It thus necessitates the need for a plug and play solution to achieve deadlock freedom in any irregular topology. An interesting point to note about such an irregular topology is that it can be mapped to a regular mesh with faulty or power-gated elements as shown in figure 1.2 thereby showing the similarity of this problem with those discussed above.

1.3 The Problem

Having looked at how irregular topologies occur on-chip, its time to now look at why deadlocks are a key problem in such topologies. This can be attributed to three reasons. First, irregular topologies offer very little path diversity as the number of paths between a given source and destination may be severely reduced due to faults or power-gating decisions. Second, often the remaining paths between nodes are highly convoluted and involve long and twisted cycles. Third, due to the reduced path diversity, it may now be very easy to sat-

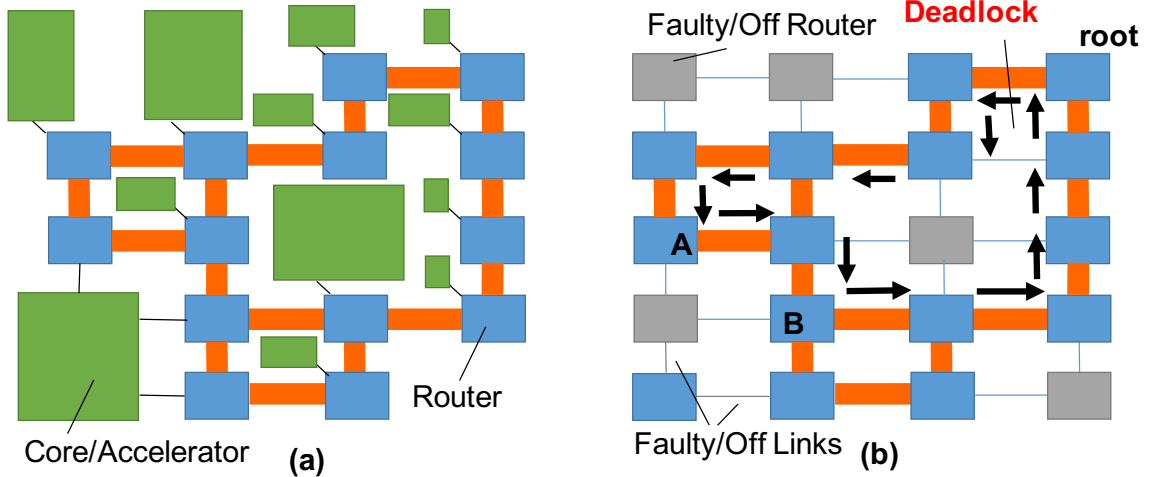


Figure 1.2: Irregular topologies on-chip due to (a) Heterogeneous SoCs (b) Router or link failures/gating.

urate the network even with regular traffic flows. All these make it easier to create a cyclic buffer dependence between routers leading to a deadlock. Figure 1.2 shows an example of a deadlock in (a) Heterogeneous SoC with irregular NoC topology and (b) an irregular NoC topology created due to faulty/power-gated routers.

Figure 1.3 shows an example mode of operation of a chip which was designed using 8x8 mesh as the interconnection network. Due to power-gating decisions the number of cores/routers that are "on" changes from 64 routers to 32 routers and later to 16 routers. The fig also shows the case where different cores/routers are chosen to be power-gated in different epochs. An interesting thing to note here is that not all the resultant irregular topologies shown contain cycles in their topology graph. An absence of cycle in topology graph also necessarily means the absence of any deadlock since to create a deadlock you need at least one cyclic path in the topology. Figure 1.3(a) and 1.3(c) don't have any cycle in the topology graph and hence have been labeled as "Deadlock free" whereas figure 1.3(b) and 1.3(d) contain cycle/s in their topology graph and are labeled as "Not Deadlock free".

The need for the presence of at least one cyclic path in the topology to be able to create a deadlock presents an interesting scenario in case of irregular topologies that are derived from a regular mesh. As the faults or the power-gating decisions are random, the

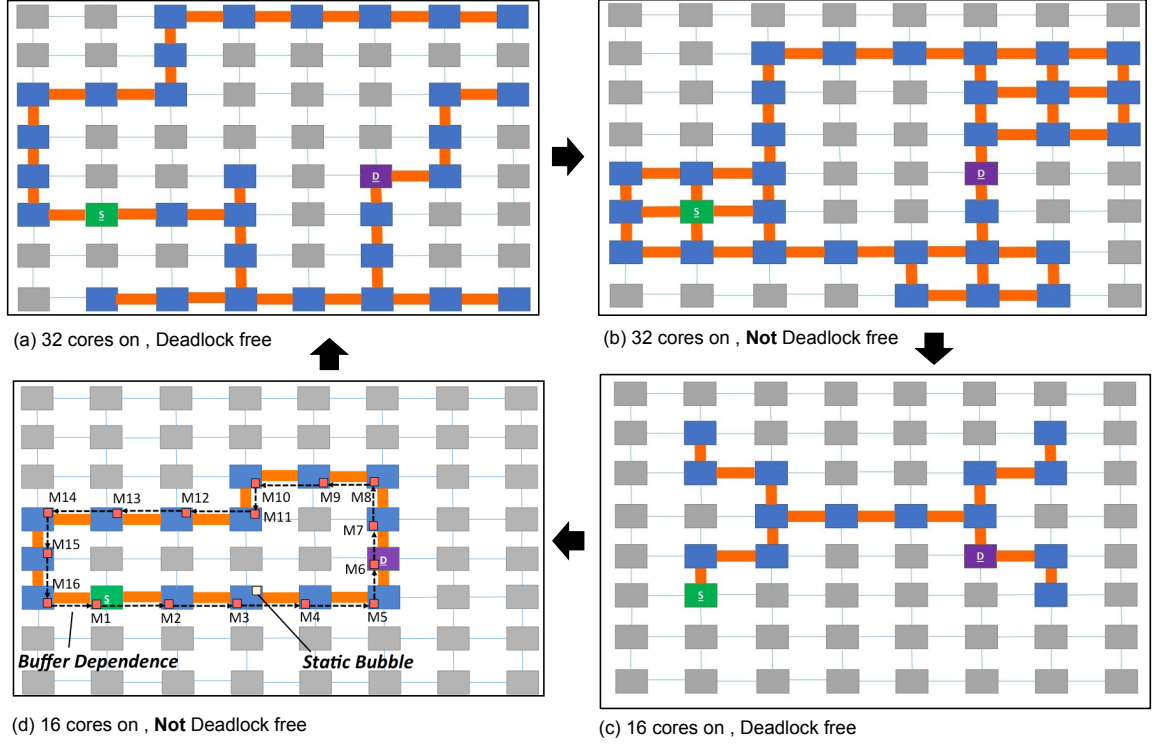


Figure 1.3: Dynamically changing irregular topology due to Power-gating of network elements.

resulting irregular topologies may not contain any cycles and thus are inherently deadlock free. It would thus be interesting to find out how many topologies at a given fault number are deadlock-prone to ascertain if deadlocks in irregular topologies are a real problem. Figure 1.4 shows the results of such an experiment where faults were randomly injected in a 8x8 mesh. For each fault number, 10K topologies were generated and each router was allowed to inject one packet per cycle (injection rate of 1.0). Each packet was routed minimally without any routing restrictions. The percent of topologies that deadlocked have been plotted against the number of chip faults.

At low number of router and link faults, almost all the possible topologies are found to be deadlock-prone i.e. they deadlocked at injection rate of one packet per node per cycle. This necessitates the need for a solution to the problem of deadlocks in irregular topologies for functional correctness of the chip. At high-fault number, the topology becomes highly-fragmented with very few short cycles that don't manifest as deadlocks even

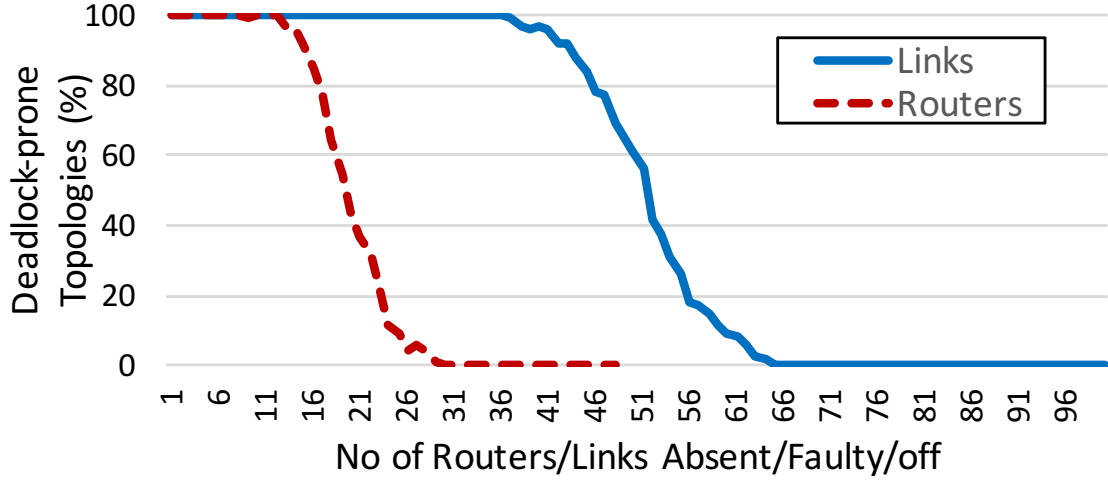


Figure 1.4: Percentage of deadlock-prone irregular topologies for a given number of faulty/absent/off routers and links in a 8x8 mesh.

at high injection rate. An important point to note is that at high fault number, the chip itself might be unusable as certain key components like memory controllers would be disconnected/unreachable from the nodes.

Deadlocks in irregular topologies are a key problem that have been addressed in all the three domains of work discussed previously: Power-gating, resiliency and heterogeneous SoC synthesis. However, the solutions proposed are highly conservative and perform poorly both in terms of latency and throughput as I show later in the evaluations. To highlight the highly conservative nature of the solutions that state of the art works propose, I performed an experiment to find out the minimum injection rate at which the irregular topologies start to deadlock with minimal routing and with no routing restrictions. Figure 1.5 shows the heat map of injection rate at which the irregular topologies for a given fault number start to deadlock. Most of topologies at low link fault number start to deadlock at injection rates of around 0.1-0.3 flits per node per cycle. This is fairly high since the injection rate of real applications (PARSEC 2.0 [10] and Rodinia [11]) is around 0.01-0.02 flits per node per cycle due to the high L1 cache hit rates. Thus, we need a very light weight solution that has a low overhead and doesn't cause significant performance loss as the problem of deadlocks in irregular topologies is going to occur very rarely. In this work,

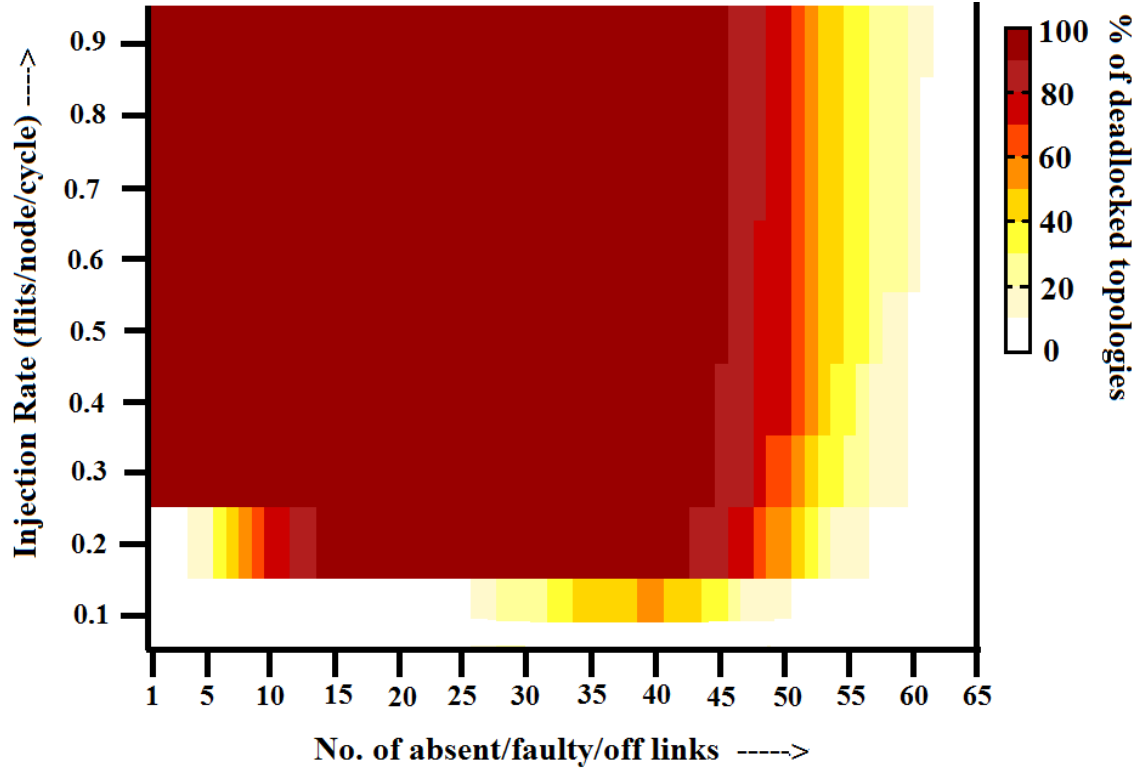


Figure 1.5: Heat-map of the cumulative frequency distribution of irregular topologies that deadlock at a particular injection rate for a given number of faulty links with uniform random traffic.

I propose a solution called *Static Bubble* that provides up-to 30% less network latency, 50% less Energy-Delay-Product (EDP) and 4x more throughput at less than 1% area overhead.

CHAPTER 2

BACKGROUND AND RELATED WORK

Deadlocks as a problem date back to the advent of Inter-connection networks in chips and supercomputers. Consequently, there have been several works [12, 13, 14, 15, 16] that have addressed it. These works can broadly be classified into two domains : *Deadlock Recovery* and *Deadlock Avoidance*. In this section, I describe these theories and provide formal definition of deadlock and related concepts and also detail the assumptions. Later on, I discuss the work related to the area of deadlocks in irregular topologies.

2.1 Assumptions

- Virtual Cut Through (VCT) routing has been assumed. The design can however be extended to wormhole routing as well.
- Packets can have arbitrary lengths.
- Although the scheme can work on any arbitrary topology, the bubble placement algorithm has been specifically designed for a mesh topology. However, the size of the mesh can be arbitrary.
- There is no minimum or maximum requirement on the number of Virtual Channels.
- A node can generate traffic destined for other any node at any rate.
- A message arriving at the destination is eventually consumed.
- Minimal Adaptive routing is used to route packets.
- The routing algorithm provides packets with live-lock-free routes.

2.2 Definitions

Interconnection Network: An Interconnection Network I is a strongly connected directed graph represented by $G(N, C)$ where N is number of vertices and C is the number of edges. Vertices in G are the processing nodes or routers. Edges in G are physical channels connecting the nodes. Each channel c_i has a queue with some fixed capacity $\text{cap}(c_i)$.

Routing Algorithm: A Routing Algorithm provides the next channel (c_{next}) for a message given the last channel used by packet (c_{prev}) and the destination node (n_{dest}).

Deadlocked Configuration: In a deadlocked configuration, there is no flit one hop from the destination. Header flits cannot move forward as the queues for all alternative output channels are full. More formally,

1. $\text{head}(c_i) \neq d_i$ and $\text{size}(c_j) > 0 \forall c_j \in R(d_i, \text{head}(c_i))$
2. $\text{head}(c_i) \neq d_i$ and $\text{size}(\text{next}(c_i)) = \text{cap}(\text{next}(c_i))$

2.3 Deadlock Avoidance

Deadlock Avoidance based routing algorithms rely on placing routing restrictions to prevent deadlocks from occurring. These techniques prevent messages from taking certain turns which prevents packets from creating a cyclic buffer dependence. The concept of turn-restriction based routing algorithm was first given by Dally and Seitz in [12]. In this paper, the authors defined the concept of *Channel Dependency Graph* and used it to prove the necessary condition for deadlock freedom : the existence of an acyclic channel dependency graph. More formally, a Channel Dependency graph is defined as

Channel Dependency Graph: A Channel Dependency Graph, D , for a given interconnection network I , and Routing function R , is a directed graph $D = G(C, E)$. The vertices of D are channels of I . The edges of D are the pairs of channels connected by R :

$$E = \{ (C_i, C_j) \mid R(c_i, n) = c_j \text{ for some } n \in N \}$$

By creating a channel dependency graph for a given interconnection network with a defined routing algorithm it can be ascertained if the routing algorithm is deadlock free. Alternatively, the routing algorithm can be made deadlock-free by removing certain edges from the cyclic channel dependency graph to make it acyclic. As edges in a Channel Dependency Graph correspond to turns, removing edges corresponds to preventing packets from taking those turns or in other words placing routing restrictions.

2.3.1 Turn Model

The concept of Channel Dependency graph was extended by Glass and Ni in their paper on deadlock-free routing for partially adaptive algorithms [15]. They introduced the concept of Turn Model to define the necessary condition for a deadlock. The turn model is constructed by analyzing all the turns that packets in network can take. By prohibiting just enough turns to break all the cycles in the turn model, a routing algorithm can be made deadlock free. For a regular mesh, this model proposed four different deadlock-free routing algorithms which are explained below:

West-first Turn Model: In this algorithm you route a packet west-first if necessary and then adaptively south, east and north. This routing algorithm is partially adaptive.

North-last Turn Model: In this algorithm a packet travels North only if that is the last direction it needs to travel. In other words, a packet is first routed adaptively west, south and east and then north. This routing algorithm is partially adaptive too.

Negative-first Turn Model: In this algorithm, a packet is routed first adaptively west and south and then adaptively east and north. This algorithm is partially adaptive.

Dimension-ordered Routing: or XY routing in case of a 2-D mesh is a completely non-adaptive deterministic routing that routes packets in X direction first and then in Y direction. Another version of this routing YX routing does exactly the opposite.

While these routing algorithms guarantee connectivity and deadlock freedom in a regular mesh topology, they fail to guarantee connectivity in case of an irregular topology. This

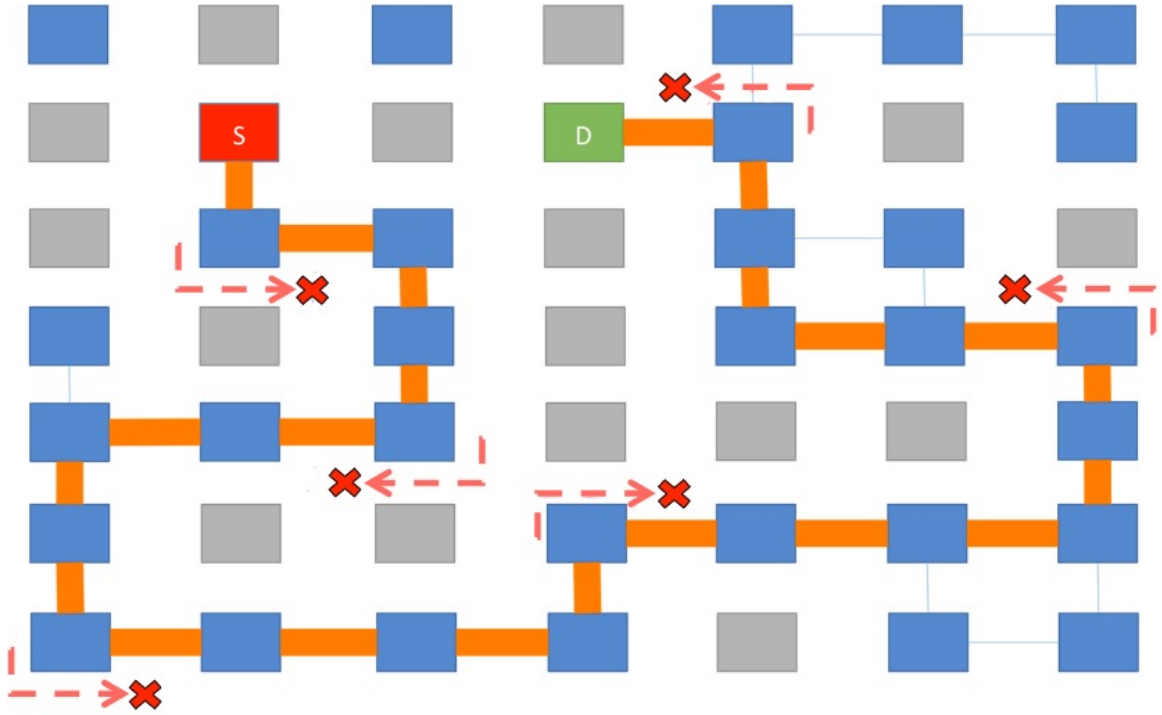


Figure 2.1: Figure showing the limitations of XY routing in irregular topologies. Here the source **S** cannot route to destination **D** as the only path that connects them involves turns that are not allowed by the XY routing algorithm.

is because certain turns might be unavoidable to guarantee connectivity between every source and destination pair. If these turns are allowed, the routing algorithm will no longer be deadlock-free. An example of such a scenario is shown in figure 2.1. Here, the source **S** is routing a packet to destination **D** in an irregular topology. The only path between **S** and **D** is highlighted in orange. However, if we were to use XY routing algorithm to route the packet, it would involve taking restricted turns (shown in red) and consequently the packet will not be able to make it to **D** from **S** despite the presence of healthy and fully-functional links connecting them. Thus, the use of turn model based routing algorithms exacerbates the problem of connectivity in an irregular topology.

2.3.2 Spanning Trees

To overcome this limitation of turn-based routing algorithms, state of the art NoC works that deal with irregular topologies construct a spanning tree over the remaining nodes and

use it to route packets over them [17, 18, 19, 20, 21]. A tree by definition cannot have a cycle and thus the routing cannot lead to a deadlock. A very common topology-agnostic off-chip routing algorithm used for the construction of spanning trees is the *up-down* routing algorithm. *Up-down* starts by selecting a root node. This step is crucial as the connectivity and throughput of the topology are highly dependent on the optimal selection of the root node. Several works have been proposed for the optimal selection of the root node to reduce the reconfiguration time or increase network connectivity [17, 18]. After selection of the root node, all links towards the root node are marked as *up*, those away are marked as *down* and equidistant ones are tagged arbitrarily. All cyclic dependencies are broken by disallowing packets from using *down* links immediately after using *up* links.

Spanning-tree based routing however, provides non-minimal paths for some traffic flows which leads to increase in latency, reduction in network throughput and increase in network energy. This is because packets have to go via the root node when being routed from one sub-tree to the other to avoid a cyclic dependency even when healthy and fully-functional links may be present connecting the nodes. This is shown in Figure 1.2 where a packet from A has to be routed via the root and takes 10-hops instead of the minimal 2-hops. In addition, construction of an optimized spanning tree across all possible root nodes, while maintaining high-connectivity and providing sufficient bandwidth is an exponential state space problem that often requires optimization solvers running in software for 1000s of cycles. Spanning-tree routing has been used as the first baseline in evaluations.

2.3.3 Alternate approaches in Resilient NoCs

Apart from spanning-tree based routing, several other techniques have been proposed to deal with deadlock-free routing over irregular topologies. However, these techniques suffer from either not being able to guarantee connectivity and deadlock-freedom in any arbitrary topology or perform poorly compared to the first baseline. Vicis [22] uses a heuristic to determine the routing restrictions. This heuristic however fails to guarantee deadlock

freedom on any arbitrary irregular topology as prior work points out [17]. ImmuneNet [23] uses local Bubble Flow Control (BFC) [24] in a ring constructed using the spanning tree of the remaining nodes. This work however uses three routing tables and offers poor performance compared to the first baseline [17]. BLINC [21] and Balboni et al. [25] use segment routing [26], where the network is divided into segments each with a different turn restrictions. They however, place a restriction on the location and/or the number of faults and thus cannot handle arbitrary irregular topology.

2.3.4 Alternate approaches in Power-gated NoCs

Two recent techniques, Power Punch [27] and CatNap [28], maintain network regularity for routing purposes by waking up the routers that fall in the path of the flits that are routed using deadlock-free XY routing algorithm. These schemes are orthogonal to this work as I address the problem of deadlocks in irregular topologies (both static and dynamic).

2.4 Deadlock Detection and Recovery

In contrast to the Deadlock Avoidance routing algorithms, routing algorithms based on deadlock detection and recovery allow packets to get into a deadlock and then recover out of it. The Channel Dependency Graph (CDG) of such an algorithm can contain cycles which will lead to deadlocks. Traditionally, such recovery based schemes have not been very popular as turn based schemes like *West-first* routing are simpler to implement while providing adequate path diversity to traffic. However, such turn-based schemes will not work in an irregular topology as pointed out earlier. Also, spanning-tree routing provides network traffic with non-minimal paths. Based on the experiments discussed in the introduction section that show deadlocks to be rare occurrences, it would be better to use deadlock recovery based routing schemes for solving deadlocks in the irregular topology domain instead of avoidance based schemes that penalize the common case to ensure correct behaviour in rare scenarios.

2.4.1 DISHA

DISHA [16] was an early work in the deadlock recovery domain that detected deadlocks using counters that are present at every node in the network. Upon the expiry of the counter, the router would wait to capture a token that circulated around the network in a loop. After successfully capturing the token, the router would put the deadlocked packet in a separate network that connected all the nodes in a loop. When the complete packet had reached its destination, the token would be released by the destination node. At a given time, only one packet could exist in the separate deadlock recovery network. This is ensured by the use of tokens.

DISHA and its recent variants [29, 30] will not work in an environment with dynamically changing irregular topology as it requires 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, DISHA would have high energy and latency penalties due to the need for constant token circulation and the need for token capture before starting the recovery process.

Ping and Bubble [31] was a subsequent idea proposed for off-chip networks that sent out a ping upon detection of a possible deadlock. The ping would traverse a control network tracing the dependency chain and reserving the output ports along the way. If it is a deadlock, an extra-buffer is turned on when the ping returns to drain the deadlock. The extra-buffer is switched off when the bubble returns. However, the design doesn't handle false positives where output ports might have been reserved even though it was not a deadlock. In addition, this scheme was proposed for off-chip networks where energy and area are not as precious of a resource as in the on-chip world.

2.4.2 Escape-VC

Escape-VC solution for deadlock freedom was proposed by Duato [14] in 1993. He showed that the requirement of an acyclic Channel Dependency Graph (CDG) given by Dally et.

al. [12] was a sufficient condition for a deadlock-free routing algorithm but not a necessary condition. According to Duato's theory, a routing algorithm can be deadlock-free with cycles in its Channel Dependency graph if and only if there exists a subset of channels $C_1 \subseteq C$ that defines a routing sub-function which is connected and has no cycles in its extended channel dependency graph. In other words, if there exists an acyclic sub-part of the Channel Dependency graph that connects all the nodes, the routing algorithm is deadlock-free.

This acyclic connected sub-part of the Channel Dependency Graph can be used to define a deadlock-free routing algorithm that can be used to route packets within a subset of Virtual Channels (VCs). This subset of virtual channel is called *escape vc*. Routing in other VCs can be done in a fully adaptive manner. This is an improvement over the turn-based models which only allowed at best partial adaptive routing algorithms and also killed the path-diversity provided by the topology. This improvement however, comes at the cost of hardware overhead of counters and routing tables. Escape-VCs can be either used as a deadlock avoidance scheme if the packets actively use it or as a deadlock recovery scheme if the packets are pushed into the escape-vc network upon detection of deadlock.

The use of escape-vc's has been proposed as an alternative to spanning trees for providing deadlock-free routing in irregular topologies. It is interesting to note that spanning-tree construction is still required as it will be used by the escape-vc network. In addition, escape-vc requires that at least one VC per input port per message class be reserved at every node for the escape-vc network. Also, an extra-routing table is required to store the spanning tree path for the escape-vc network. VC reservation causes throughput loss while the extra-routing table has associated energy and area overheads. NoRD [32], recently proposed for NoC power-gating uses a high-latency deadlock-free ring snaking around the network as the escape path. Packets are forced to enter the escape-Vc network after their mis-routed hop count increases beyond a certain threshold. Router Parking [20], another work in the NoC power-gating domain, replaces the high-latency ring of NoRD with a

spanning-tree constructed using up-down routing. Escape-vc's using spanning-tree based routing for only the escape network and minimal adaptive routing for other VCs has been used as the second baseline for comparison in this work.

2.5 Bubble Flow Control (BFC) in Rings

Bubble Flow Control [24] is a flow control algorithm that provides the necessary and sufficient condition for deadlock avoidance in a ring : as long as there is one bubble or one empty buffer in a ring, there cannot be any deadlock and forward progress can be made. The presence of one empty slot is ensured by intelligent injection of packets. Nodes are required to capture tokens before injecting packets in the ring. In this work the underlying theory of one empty slot being enough to make forward progress in a ring has been leveraged.

2.6 Routing over Irregular topologies

Prior works across resiliency and power-gating use a mix of hardware [17, 18] and software [21, 20, 19] techniques to identify connectivity among currently active routers upon detection of faults in or power-gating of network elements. Disconnected components are discarded and routing tables are populated at every source Network Interface (NI). In this work I leverage this rich body of work and provide a routing table at every node in the network. For spanning-tree baseline, this routing table has routes derived from the spanning tree. For escape-vc and static bubble, these routes are minimal. The escape network in the escape-vc baseline also uses routes derived from spanning tree.

CHAPTER 3

STATIC BUBBLE

Static Bubble is a deadlock recovery algorithm that guarantees deadlock-freedom for any regular or irregular (both static and dynamically changing) topology that can be derived from a mesh topology. Upon detection of a deadlock, the recovery algorithm introduces an extra-buffer (called **static bubble**) in the deadlocked chain. According to the principle of Bubble Flow Control (BFC), one empty buffer in a deadlocked ring is enough to guarantee forward progress. At the core of the deadlock recovery algorithm lies a novel placement algorithm that guarantees the presence of at least one static bubble in every possible dependency chain in the mesh. Since the placement algorithm covers all possible dependency chains, any irregular topology derived from this mesh topology is guaranteed deadlock freedom by the recovery algorithm discussed in this work. In this chapter I discuss the placement algorithm, the recovery algorithm, the micro-architecture and the special cases.

3.1 Static Bubble Placement Algorithm

The deadlock-freedom guarantee for any irregular topology derived from a mesh topology is based upon the guarantee provided by the static bubble placement algorithm : every possible dependency chain in any NxM mesh topology will have at least one static bubble. The algorithm picks up nodes where the extra-buffer or the static bubble will be placed. The nodes that the algorithm picks have been referred to as *static bubble nodes* in the work. The algorithm is as follows: For any node (x,y) in a NxM mesh, a static bubble will be added if ($x > 0$) and ($y > 0$) (no static bubbles in the first row and the first column) and one of the following conditions hold:

- (1) $x \bmod 4 \equiv y \bmod 4$
- (2) $x \bmod 4 \equiv 1$ and $y \bmod 4 \equiv 3$

(3) $x \bmod 4 \equiv 3$ and $y \bmod 4 \equiv 1$

Figure 3.1(a) shows the placement of 21 static bubbles in a 8x8 mesh. Visually, the nodes on the solid diagonal satisfy condition (1), while the ones on the dotted diagonals satisfy conditions (2) or (3).

I now provide the proof for the claim made earlier in the section : every possible cycle in the mesh topology has at least one static bubble.

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) doesn't contain a static bubble.

The coordinates of any such node (except on the first row and column) will be of one of the following 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). This is shown in Figure 3.1(b). All these five types of nodes (shown in fig) are bound by static bubbles. Every hop or turn is an increment or decrement of k or l . It is not possible to make 4 (min number required to complete a cycle) turns that would complete a cycle without encountering a node that satisfies one of the three conditions of the placement algorithm.

It is worthwhile to note that to reduce the number of static bubbles, no node from the first row and column is selected as not all turns are possible from these nodes. In addition, the design doesn't allow 180 degree turns or u-turns. A corollary of the proof presented above is that any irregular topology derived from this mesh topology will also have at-least one static bubble in each of its dependency chain. An interesting scenario arises if the static bubble nodes themselves become faulty or are selected for power-gating. If a

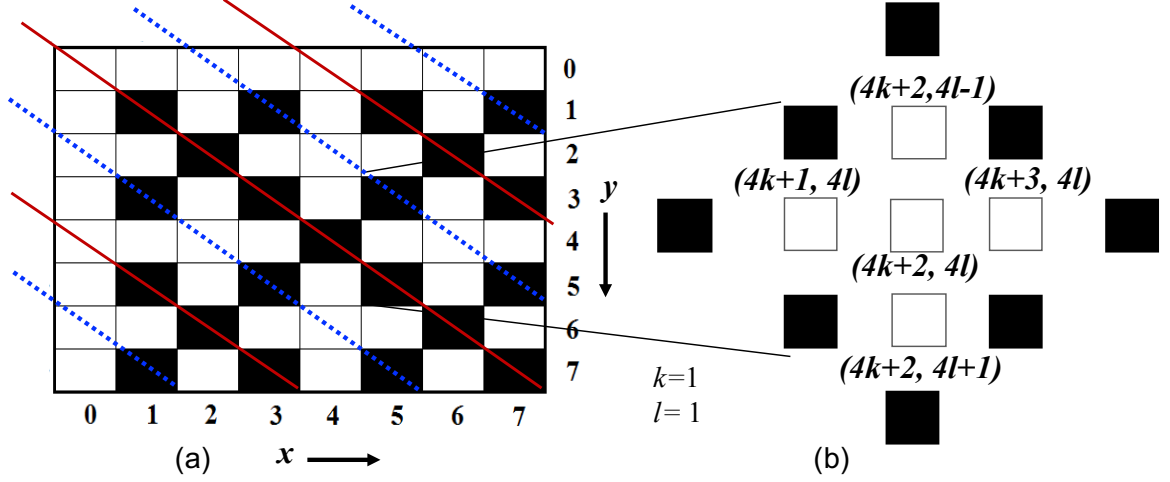


Figure 3.1: Placement of static bubbles on a 8x8 mesh at design-time to guarantee a bubble in any possible cycle.

static bubble stops functioning then all the dependency chains/cycles that are a part of this deadlock get broken. In the extreme scenario where all the static bubble nodes go offline, all possible network dependency chains are broken and the network is deadlock-free. In addition there can be cases where one static bubble is part of more than one dependency chain. In this case, the algorithm can resolve deadlocks in all of them serially. The static bubble count provided by the algorithm scales linearly with the minimum of (M,N) which keeps the hardware overhead low. The number of static bubbles in a NxM mesh given by the algorithm is:

$$\begin{aligned}
 & \sum_{k=0}^{\lceil \frac{m}{4} \rceil - 1} (\min(m - 4k, n) - 1) + \sum_{l=1, l \text{ odd}}^{\lceil \frac{m}{2} \rceil} \frac{(\min(m - 2l, n))}{2} + \\
 & \sum_{p=1}^{\lceil \frac{n}{4} \rceil - 1} (\min(m, n - 4p) - 1) + \sum_{r=1, r \text{ odd}}^{\lceil \frac{n}{2} \rceil} \frac{(\min(m, n - 2r))}{2}
 \end{aligned} \tag{3.1}$$

where $\lceil \cdot \rceil$ represents the Greatest Integer Function (GIF).

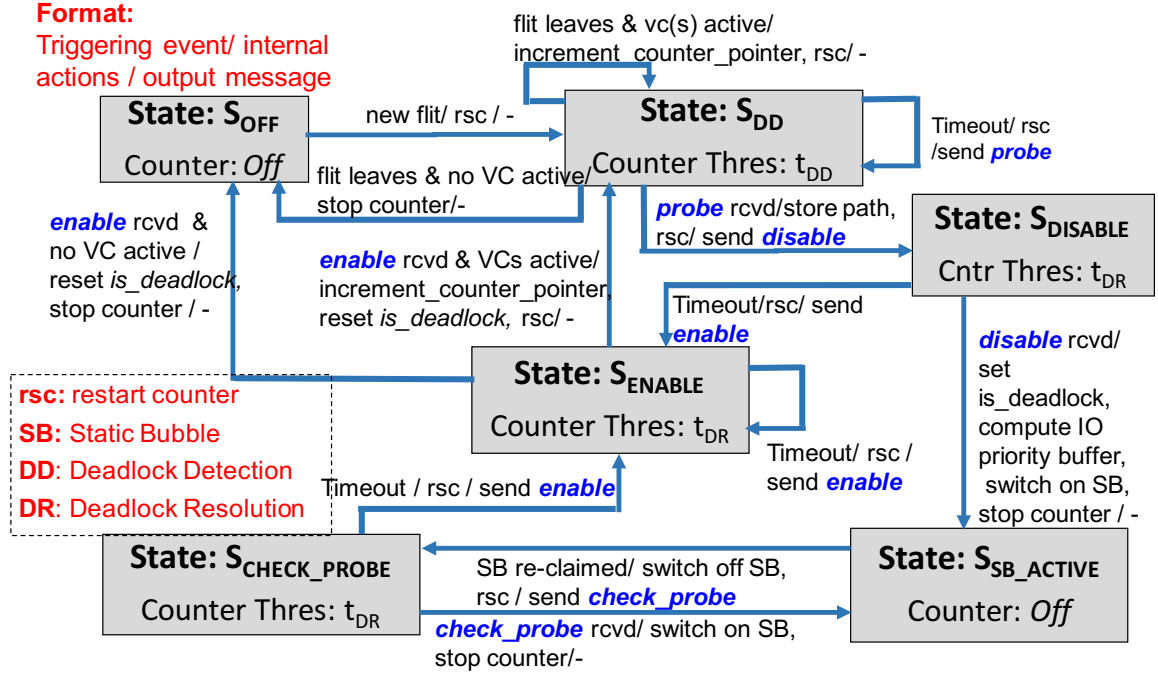


Figure 3.2: Finite State Machine of the static bubble counter.

3.2 The Recovery Algorithm

Each static bubble router contains one extra-packet sized buffer called static bubble and one counter associated with a finite state machine (fsm). When the system starts, all static bubbles are in off state. The FSM associated with the counter at a static bubble node has six different states shown in Figure 3.2. The counter has two possible thresholds : t_{DD} (Deadlock Detection), which is a configurable parameter and t_{DR} (Deadlock Recovery), which depends on the length of the deadlock path. To aid in flow-control when doing deadlock recovery, four special messages have been introduced namely, *probe*, *disable*, *enable* and *check_probe*. The probe message is used to obtain the deadlocked path by following the buffer dependency chain. Disable message places injection and routing restrictions at nodes that are a part of the deadlock while the enable message lifts those restrictions. Check_probe messages are used to confirm the presence of the dependency chain after the static bubble has been reclaimed.

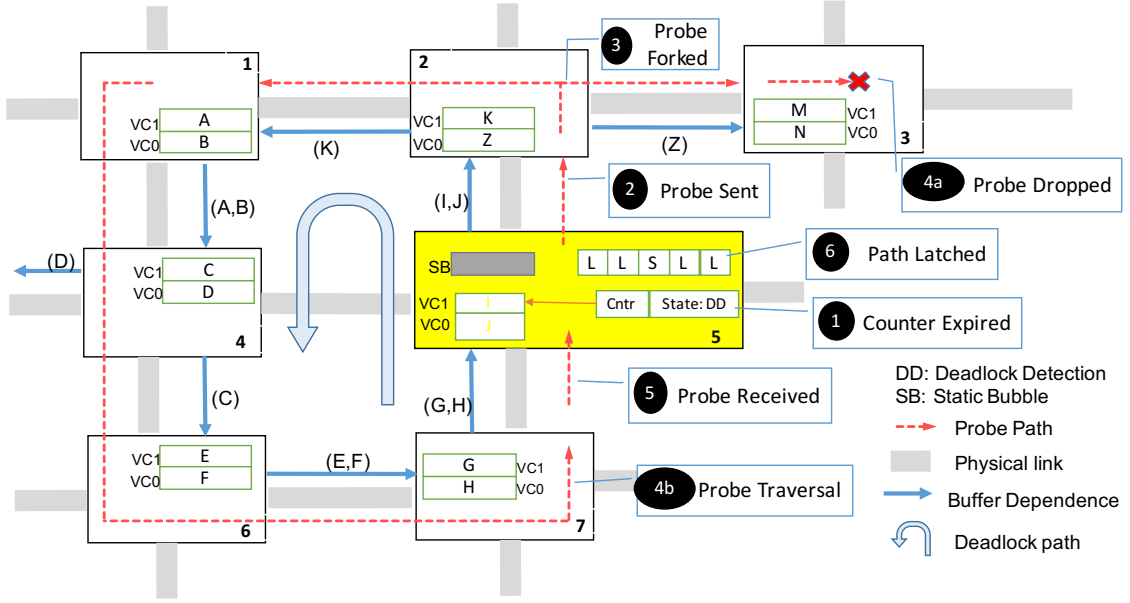


Figure 3.3: Walkthrough: Probe Traversal

3.2.1 Walk-through Example

The FSM at a static bubble router starts in the OFF state. When a new flit arrives at the router, (other than from the Local port or the Network Interface) the counter is made to point to the VC it occupies and starts counting to the Deadlock Detection threshold. If the flit leaves within the threshold time, the counter pointer is incremented to point to a non-empty VC (VC in active state) in the router in a round robin manner. If there is no VC in active state, the FSM transitions to the OFF state and the counter stops counting.

In this section, I describe the various steps involved in the deadlock recovery process using a walk-through example in Figure 3.3. Each buffer dependence in the figure is marked by 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 dependency chain :

$$(A,B) \rightarrow (C) \rightarrow (E,F) \rightarrow (G,H) \rightarrow (I,J) \rightarrow (K) \rightarrow (A,B)$$

In this dependency chain, Node 5 is the static bubble router. Each VC can hold one packet. I explain this design with Virtual Cut Through traffic (VCT) (packet sized buffers) for simplicity though a wormhole design would work too with minor alterations.

Probe Traversal

In Figure 3.3 the counter expires in S_{DD} state (**Step 1**) as packet I doesn't leave within the threshold time (Deadlock Detection threshold). Node 5 then sends out a *probe* message (**Step 2**) from the North output port (output port for packet I) to check if it is a real deadlock (and not a false positive due to congestion) and get the deadlock path. After sending out the probe, the counter is reset and restarted and the counter pointer is incremented in a round-robin manner to point to a non-empty VC. The threshold is set to t_{DD} .

The probe message will track the buffer dependency and if it is cyclic (requirement for deadlock), the probe message will come back to the starting node. When a probe reaches a router from a given Input port, it gets forked out of all the outports that the VCs at the input port are waiting on. If however, any VC is waiting for the local port to get ejected or there is an empty VC at the input port, the probe is dropped at the router. This is because of the assumption of packet consumption made earlier. Consuming a packet would empty a VC in the dependence chain and consequently forward progress can be made with one empty buffer in a dependency according to the principle of Bubble Flow Control (BFC). Thus a deadlock is not possible and the probe should be dropped.

The forking operation creates identical copies of the probe message, so all the information already present is retained. The router appends input to output turn (Left turn (L) or Right turn (R) or Straight turn (S)) in each output probe. For instance, at node 2 the probe gets forked out of the west and east output ports and the router appends turns (L) and (R) respectively to the probes (**Step 3**). At node 3 (**Step 4(a)**), the probe is dropped as packets M and N are waiting to get ejected and thus cannot be a part of a cyclic buffer dependence chain. At nodes 1, 4, 6 and 7 the probe is forwarded out of the south, south, east and north output ports respectively (**Step 4(b)**) and the turns L, S, L and L are appended respectively. When the probe returns to node 5 (**Step 5**), the deadlock is confirmed and the path acquired by the probe (L, L, S, L, L) is latched in a special buffer called *the turn buffer* (**Step 6**).

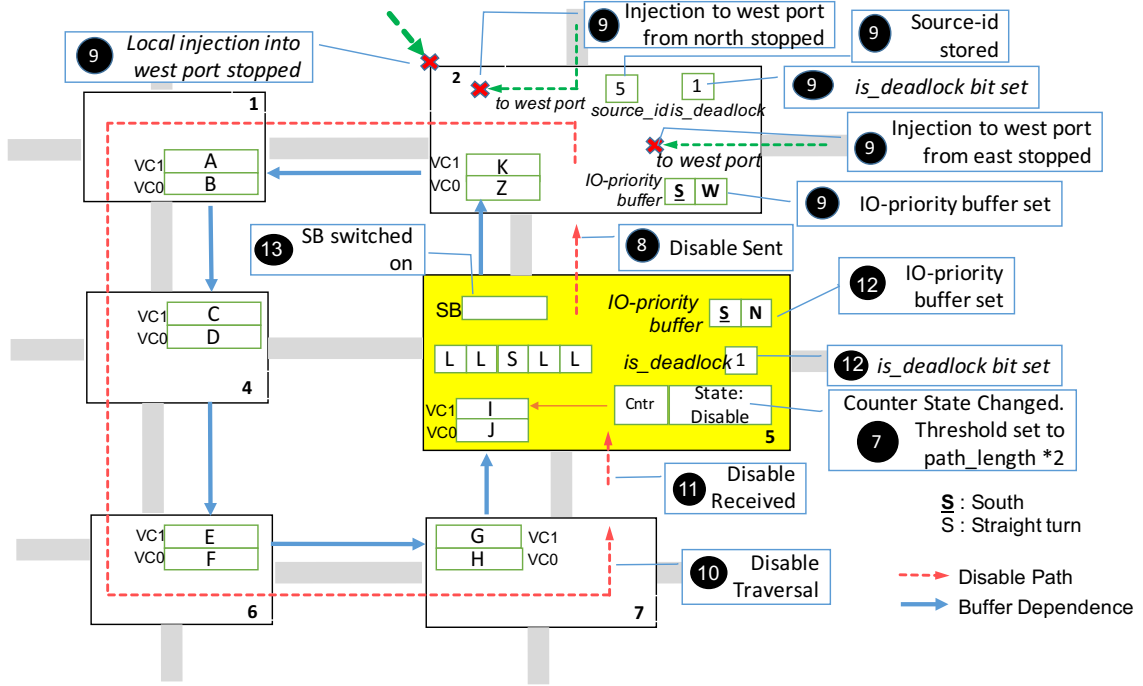


Figure 3.4: Walkthrough : Disable Traversal

Disable Traversal

After the probe returns with the path of the deadlocked chain, router 5 sends out a disable message (**Step 8**) along the deadlocked path (Figure 3.4). The counter state changes to $S_{Disable}$ and the counter threshold to t_{DR} (**Step 7**). t_{DR} is set to twice the deadlock path length as the disable is guaranteed to return within this time unless it is dropped as will be explained later. The disable message carries the deadlock path and the node-id of the sender (node 5).

Upon receiving the disable, each router disables injection of traffic into the output port corresponding to the turn specified by the disable. Only the input port through which the disable entered the router is still allowed to inject flits into this output port. For instance, node 2 (**Step 9**) extracts the first turn field, Left (L) from the disable entering at the south 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 *source-id* buffer. It also sets the *is_deadlock* bit to '1'. The *is_deadlock* bit if set, instructs the switch allocator at node 2 to disable injection

into West output port from every input port except the South input port. In other words, no other flit is allowed to enter the detected dependency chain. Node 2 then removes the first turn from the disable message and sends it out of the West output port.

All nodes along the dependence chain (1, 4, 6 and 7) process the disable message in a similar manner (**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 the deadlock recovery process (**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 transitions to state S_{SB_Active} . In this state, there is no threshold associated with the counter and the counter doesn't increment its count.

A bubble has now been introduced in the deadlocked ring and any new packet has been prevented from entering the deadlocked loop. This allows the packets that are a part of the deadlock to move forward by one hop. This can be seen by looking at the buffer occupancy change between Figure 3.4 and Figure 3.5. The information about switching-on of the static bubble is conveyed to node 7 through standard flow control credit messages. Packet G from node 7 comes and occupies the static bubble node. This allows packets E, C, A and K to each move forward by one hop. Packet G occupying the static bubble moves to VC1 at node 2 and the static bubble is empty again. Even 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 then move to the VC vacated by packet I. The Static Bubble would become free and thus can be reclaimed/switched-off.

It is interesting to note here that the static bubble cannot be switched on without first placing the restriction on injection into the deadlocked loop. This is because a new packet can come and occupy the static bubble. If this happens, we will be stuck in a deadlock

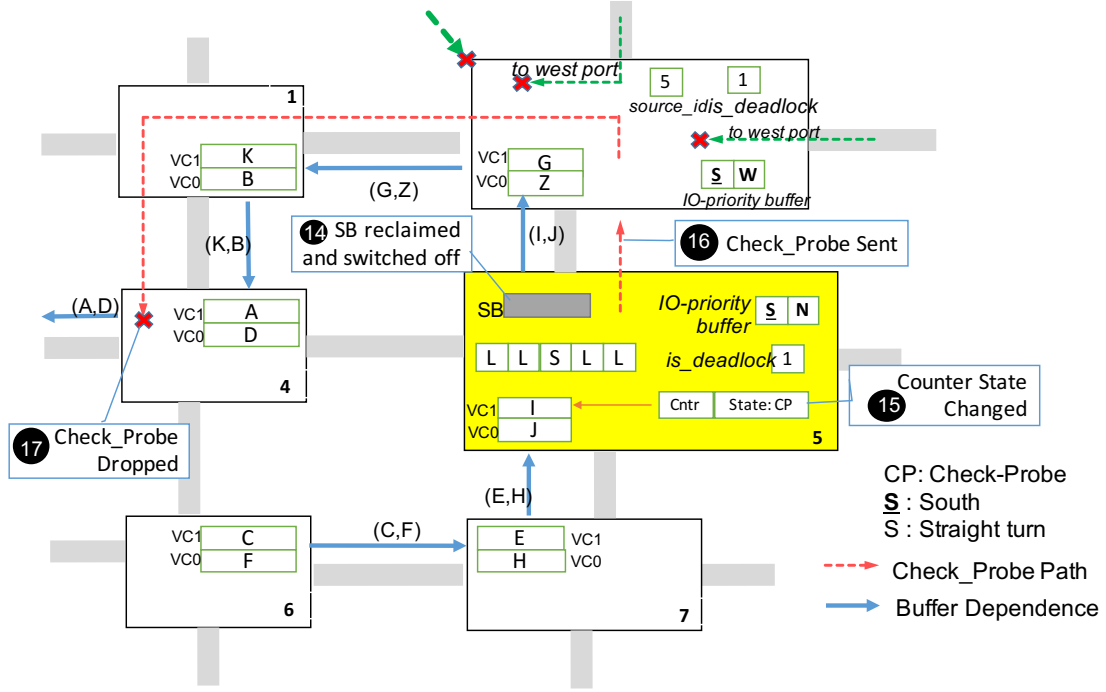


Figure 3.5: Walkthrough : Check_Probe Traversal

without any recovery mechanism.

Check_Probe Traversal

Once the deadlocked ring moves forward by one step Figure 3.5(c), the static bubble is reclaimed and switched-off (**Step 14**). At this point the FSM moves to the S_{Check_Probe} (**Step 15**) and the counter is reset and restarted with threshold t_{DR} . A *check_probe* message is sent out along the same deadlocked path (**Step 16**). Unlike a regular probe, the check_probe is not forked but simply forwarded out along the same dependency chain as long as at least one VC at the router is still a part of the same dependency chain detected earlier by the probe (and later traversed by the disable).

If the check_probe returns, the static bubble is switched on again and the dependence ring moves forward by one more hop. In this walk-through example the check_probe is dropped at node 4 (**Step 17**) as both packets (A,D) at the north input port don't want to use the south output port. If the check_probe doesn't come back within the threshold time

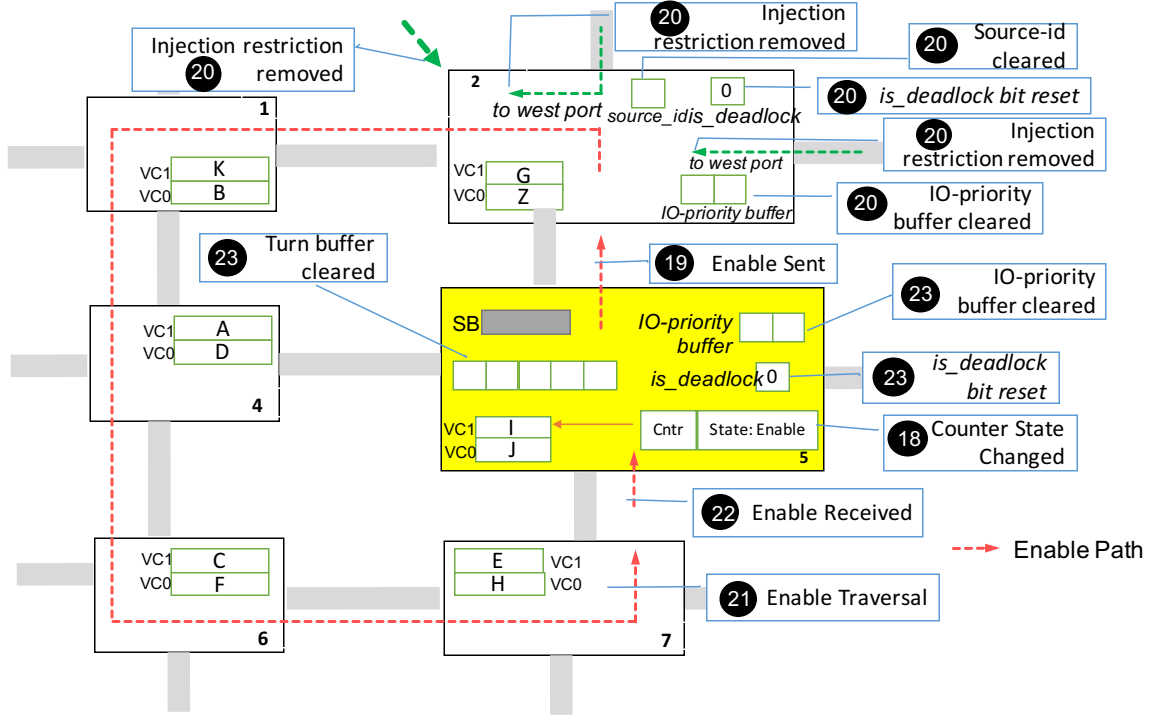


Figure 3.6: Walkthrough : Enable Traversal

(t_{DR}), it indicates that the deadlock due to the previously detected dependency chain has been resolved/no longer exists.

Enable Traversal

If the counter expires in state S_{Check_Probe} due to the check_probe not making it back to the source static bubble node, the FSM transitions to S_{Enable} and the counter is reset and restarted (Figure 3.6). The threshold is again set to t_{DR} (equal to 2 times the length of deadlock path in the Turn Buffer) (**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 in the enable path checks if the sender-id of the enable matches the id stored in the source-id buffer. If the ids match, the router clears the *is_deadlock* bit and the *IO-priority* buffers (**Steps 20 and 21**). This lifts the injection restrictions and 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 its *is_deadlock* bit and clears the turn-buffer and

the IO-priority buffer (**Step 23**). The recovery process is now complete and the deadlock has been resolved. Next, the counter pointer is incremented in a round-robin manner to point to a non-empty VC. The FSM state is updated to S_{DD} and the counter threshold is set to t_{DD} . If all the VCs at a router are empty the counter goes back to S_{OFF} . It is essential to note here that it is necessary that routers check the source-id buffer and the node-id of the sender and make sure that they match. This is because a router may receive enable from a different static bubble router than the one whose disabled was processed by the router. This will happen when the disable of some router gets dropped and the FSM transitions straight to S_{Enable} and sends out an enable.

3.3 Special Cases and Design Details

A number of interesting cases arise due to the interaction of multiple special messages which are flowing through the network at the same time. A lot of the design decisions have been taken to ensure correct behavior in these special cases i.e. guarantee deadlock freedom in the midst of multiple special messages from multiple static bubbles. In this section, I discuss these special cases and show how the design manages them. This will also illustrate the motivation behind a particular implementation of the architectural feature of the scheme when there were more than one possible ways to implement it. The special cases have been discussed in a question-answer format.

A strict priority order is maintained in the processing of special messages at each node :

$$Check_Probe > Disable\ or\ Enable > Probe > Flit$$

This priority order is required for switch arbitration in case more than one special messages want to use the same output port in the same cycle at the same node. It also ensures that all the nodes maintain a consistent architectural state which helps avoid race or deadlocked conditions. At the static bubble node, in addition to the priority order rules, the FSM provides additional control in the processing of messages which ensures that the FSM state

cannot be changed by other nodes once it has started the recovery process.

1. What if the counter expires before the probe returns or all copies of the probe get dropped ?

The counter resets and restarts and the FSM sends out a new probe. This however cannot continue indefinitely. If there is a real deadlock the probe would return. Else things might just be moving slowly due to congestion. Eventually, the congestion will clear-up and the flit would leave.

2. What if the flit leaves by the time probe returns ?

When the probe returns, the source router checks the VCs at the input port to find if there still exists any VC that wants to use the output port from which the probe was sent out. If that is the case, the next steps which include sending a disable happen else the probe is dropped and the disable is not sent.

3. What happens if there are two or more static bubbles in a dependency cycle and both send out probes ?

The static bubble node with the highest-id in the dependency chain is responsible for resolving the deadlock. If a static bubble node receives a probe from another static bubble node with lower id, it drops that probe. However, if the probe came from a static bubble node with higher-id, the probe is allowed to pass through. This ensures that the probe that came from the static bubble node with the highest-id in the loop(sent earlier or later) completes the full loop and returns to its source router. Once the probe returns, the router starts the recovery process by sending out a disable.

4. What if there are two different deadlocked dependency chains that are both sharing only one static bubble ?

The static bubble will successfully resolve the deadlocks serially i.e. one after the other depending on which direction it sent out the probe first.

5. What happens if a static bubble node sends a probe, followed by a disable and then receives a copy of its own probe back ?

This means there are two different cyclic dependency chains starting from the same output port that this static bubble is a part of. Since the router has already sent out the disable for the first deadlock chain, this copy of the probe will be dropped. Once the first deadlock resolves, timeout counter will expire again and 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 (each trying to resolve a different deadlock). If however, the same static bubble is a part of multiple deadlocked rings, the deadlocks are resolved serially by the static bubble node.

6. 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 dependence scenarios where one buffer dependency chain might depend on another. In the walk-through example discussed in the previous section, if the probe message was dropped at node 4, and there was such a dependency cycle, the deadlock would never get resolved.

7. Can a probe loop around infinitely due to buffer dependency ?

No. Each turn takes 2-bits to encode. Since all special messages in our design are single flit messages that use the same links as the regular flits, their turn carrying capacity is finite. In a 64-core mesh, assuming 128-bit wide links, the probe can carry a maximum of 59 turns (3-bits for message type and 6 bits for sender node-id). After the turn carrying capacity of the probe gets exhausted it is dropped.

8. Can false positives (i.e. no real deadlock) lead to the enabling of the static bubble ?

If there is no cyclic dependency chain, the probe will get dropped without returning back to its sender. There may however be dependency cycles due to congestion

which may make the probe return to its sender. In this case the disable will be sent out. If any of the nodes in the cyclic buffer dependence chain including the sender, no longer have the same buffer dependence as detected earlier by the probe, the disable is dropped at that node. Further in some cases, congestion may lead to probe and disable both returning successfully leading to the tuning ON of the static bubble. This will let the dependence chain move forward by one hop and the static bubble will be turned OFF again. Thus, there is no correctness problem.

9. What if a node receives more than one probes or disables or enables in the same cycle ?

Send the one with higher node-id and drop the rest. The FSM at the sender of the dropped probe/enable/disable will handle re-transmissions.

10. Can a non-static bubble node receive more than one disable, one after the other ?

A node will not receive more than one disable from the same static bubble node in the same cycle. But it can be a part of more than one dependency chain and receive disable from both of them. If the *is_deadlock* bit is already set, other disables will be dropped. If more than one disable messages are received in the same cycle, it will process the one that came from the highest node-id router and drop the rest.

11. What happens if a Disable gets dropped midway and doesn't return to the sender node ?

The static bubble router FSM which is in state $S_{Disable}$ will expire waiting for the disable to return. It will then transition to S_{Enable} and send out an enable message. This is required as the nodes that the disable went to before it was dropped would have processed it and placed injection restrictions that now need to be removed.

12. Can a static bubble node receive a disable/enable when it has itself sent an enable/disable ?

The same static bubble can be a part of two dependency cycles. In the first cycle, it may be the highest-id node and may have sent a disable/enable after receiving the probe back. In the second cycle, it may not be the highest-id node and thus would let the probe from a higher-id node pass through. Now since it allowed the probe from a different static bubble node to pass through, it may receive a disable/enable from that router. Since it is in state S_{DR} , it will drop that disable/enable. Effectively this means that we are allowing the first deadlock chain to clear. Later the recovery process can be initiated for the second deadlock cycle (if it still exists) by the other static bubble node.

13. **Which state does the FSM of a static bubble node transition 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 message arrives, the counter pointer will be incremented to point to a non-empty VC and its state will be changed to S_{DD} .

14. **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 the different VCs at various ports of a node can be part of multiple dependency chains. If the *node_id* carried by the enable doesn't 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 and not dropped.

15. **Can static bubble scheme solve a deadlock chain that is in the shape of a knot or the number "8" ?**

Yes. In a mesh with four input and four output ports per router and with 180 degree turns restricted, there can be upto two input-output port pairs that can be a part of a deadlock chain. Thus having a two entry *IO priority* buffer would suffice to solve such a deadlock with other things remaining the same. If the static bubble node is the

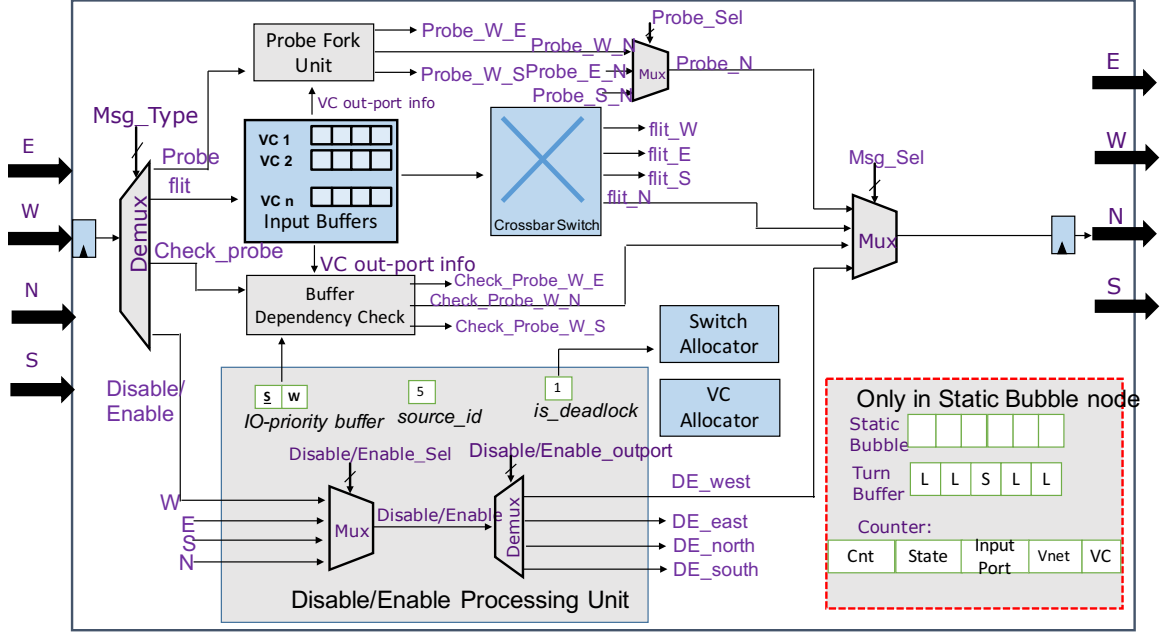


Figure 3.7: Router Microarchitecture : Additional Probes/Disable/Enable Circuitry in grey.

intersection router, the credit signal for the switching on of static bubble can be sent to either entries of the *IO priority* buffer (but not to both).

3.4 Router Microarchitecture

This section discusses the microarchitecture details of the static bubble scheme. We use the following features to make our scheme low-cost and plug-and-play:

- All the special messages : *probe*, *disable*, *enable*, *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 messages is completely buffer-less which saves area and energy.
- All the special messages use the same links as the regular flits and get higher priority. Thus, no additional wires have been added to the topology. During a deadlock condition, the output links are idle in any case since the flits are stuck waiting to get a free VC at the next router. Hence, leveraging these idle links for the transmission of special messages will not incur a significant performance penalty.

- The processing of special messages is carried out in parallel units off the critical path of the router pipeline for flits. The only component added to the critical path is a mux at the output port that is used to select between different messages (including the regular flit) during the link traversal stage. Implementation of the design in 32nm DSENT [33] shows that this doesn't increase the critical path delay which is dominated by the switch allocator and we can still use the state of the art single cycle router [34].

Fixed Delay of Messages: A highly unique and useful feature of the static bubble scheme is that once a probe returns with the deadlock path, the time in which other special messages *disable*, *enable*, *check_probe* would return back to the source node is known : twice the deadlock path length assuming 1-cycle latency for the link and a single cycle router. If the special message doesn't return within this time, it means it was dropped somewhere in the path. The FSM (figure 3.2) 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 the cyclic dependency chain, things are simple : a router will receive special messages from only one node as the walk-through example illustrated. However, if there are multiple static bubbles within a dependency chain, or a router is a part of multiple deadlocked dependency chains, multiple static bubbles can start sending probes and other special messages which arrive at the router in the same cycle. In any cycle, a router can receive upto four special messages (probe/disable/enable/check_probe) from its four 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 in processing of the special messages which guarantees correct behavior and deadlock freedom.

Figure 3.7 shows the micro-architecture of each router. The router contains standard units like Virtual Channel Buffers at each input port, Switch Allocator unit, Virtual Channel Allocator unit and the Crossbar switch. The units colored grey show the components that

have been added to the micro-architecture of a standard NoC router to support the static bubble scheme. Each incoming special message is demuxed into appropriate unit.

- **Probe Fork Unit:** is present at each input port. This unit sends one (or none) probes to every output port. It scans through all the VCs at the input port to find all unique output ports the packets are waiting on. It then creates copies of the probe and dispatches to the switch to be sent out of the appropriate output port. If multiple probes want to use the same output port, *Probe_sel* selects the one from the highest node-id and drops the rest.
- **Enable/Disable Processing Unit:** This unit is centralized and there is only one in every router. It 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 the source-id buffer using the information present in the disable/enable. If the router receives both an enable and a disable in the same cycle for the same output port, then if the *is_deadlock* bit is set at the router, the enable is sent out and the disable is dropped, else the opposite happens.
- **Buffer Dependency Check:** This is a centralized unit and only one is present in every router. This unit checks if there exists at least one VC at the input port of the router that wants to use the output port stored in the *IO_priority Buffer*. If that is the case, the check_probe is forwarded out of that output port, else it is dropped.

After being processed at the input units described above, the following priority order is used to select between special messages that want to use the same output port in the same cycle :

$$check_probe > disable\ or\ enable > probe > flit$$

This order is used to generate the *Msg_sel* signal at run-time that is used by the Mux at the output port. All the operations on special messages and the associated conditions have been summarised in table 3.1.

Table 3.1: Special Message Types

Message	Operation	Condition
Probe	<i>Fork:</i> Create copies of the <i>probe</i> to be sent along different directions	When VCs at the probe's input port at a router want to use different output ports (see walk-through example Step 3 for reference).
	<i>Drop:</i> Kill the <i>probe</i>	(i) If there exists at least one VC at probe's input port that is empty. (ii) If there exists at least one VC at probe's input port that contains a packet waiting for ejection at the router. (iii) If the <i>is_deadlock</i> bit is set at the router. (iv) If there is another probe from a node with a higher <i>id</i> (v) If it has exhausted its turn carrying capacity. Additional conditions at Static Bubble (SB) Routers: (vi) If the counter is in state $S_{Disable_Enable}$ or S_{Check_Probe} . (vii) If the static bubble router's own <i>id</i> is greater than the probe's node-id.
Disable	<i>Drop:</i> Kill the <i>disable</i> .	(i) If the <i>is_deadlock</i> bit is set at the router. (ii) If there is another disable from a node with a higher <i>id</i> (iii) [At SB Router]: If the counter is in state $S_{Disable_Enable}$ or S_{Check_Probe} .
Enable	<i>Drop:</i> Kill the <i>enable</i> .	(i) If there is another enable from a node with a higher <i>id</i> , except if (ii) <i>is_deadlock</i> is set and this enable's <i>node-id</i> is same as the <i>source-id</i> buffer.
Check_Probe	<i>Drop:</i> Kill the <i>check_probe</i> .	(i) When the buffer dependency indicated by the IO_priority buffer at a router does not exist anymore.

Static Bubble Routers: A static bubble router, in addition to the units described above, also contains a FSM, a counter, a static bubble and a turn buffer as shown in Figure 3.7. A static bubble is like any other VC and is deep enough to hold the largest packet (Virtual Cut Through).

Static Bubble vs Escape-VC: Escape-VCs are a powerful framework for deadlock-freedom and table 3.2 compares the two schemes both qualitatively and quantitatively in terms of area overhead. The deadlock resolution time for escape-vc solution depends on the spanning-tree based escape path while for Static Bubble this depends on the length of the deadlocked path as the disable and enable need to traverse the deadlocked loop to resolve the deadlock.

Table 3.2: 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	Spanning Tree/Ring Routing Table
Additional Buffers in $n \times m$ Mesh	Equation 1 21 in 64 core 89 in 256 core	$n \times m \times 5$ 320 in 64 core 1280 in 256 core
Area Overhead	$\sim 0\%$	18%

Synthesis of Network Router: Implementation of a Static Bubble router in DSENT [33] at 32nm showed less than 0.5% area overhead compared to a conventional 1-cycle mesh router (where the buffers and crossbar dominate the area). In addition, compared to the escape-vc scheme 18% lower area overhead is observed. This is because one VC per vnet per input port at every router needs to be reserved for the escape-path which leads to an increase in router area. Moreover, since Static Bubble (SB) does not require a deadlock-free spanning-tree for its operation unlike the deadlock avoidance schemes or escape-vc (which needs it for the constructing the escape-path), the network reconfiguration cost can be reduced significantly compared to prior works [17, 21].

CHAPTER 4

EVALUATIONS

In this section, I quantitatively compare static bubble against the two baselines described earlier in the background and related works section namely deadlock avoidance using spanning-trees and deadlock recovery using escape-vcs. In addition, other evaluations in this section illustrate the correctness and low-cost nature of the scheme.

4.1 Simulation Methodology

State Space Exploration with Fault Model

All the simulations used a 8x8 mesh as the underlying base topology. All faulty/power-gated irregular topologies have been derived from a 8x8 mesh. Two fault models are used to generate the topologies. In first, faults are injected randomly in the network and mapped to link failures. These links are taken out of the topology graph. In the second model, faults are injected in randomly selected routers and they are then removed from the topology graph. Both of these fault models are in line with previous works in the resiliency domain [18, 22, 35]. For ease of writing, we call the network elements removed from the topology graph as faulty components though they may 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, I keep increasing the number of topologies till the average of the trend we wish to study (like throughput, latency, etc.) stabilizes. Due to the high symmetry of the mesh topology, many of the generated irregular topologies gave similar results and the trend stabilized within 100 topologies for most cases. A key observation from Fig 1.4 and Fig 1.5 is that at high number of faults /power-gated links or routers, the

Table 4.1: System Configuration

Network Configuration	
Topology	8x8 Mesh
Routing	Source Routing
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 [36, 18, 22] (links & routers)
Traffic (using gem5 [37] + Garnet [38])	
Synthetic	Uniform Random with mix of 1-flit and 5-flit packets
Multi-threaded	PARSEC [10] running on Hyper-Transport [39] protocol
Heterogeneous	Rodinia [11] traces

topologies are highly fragmented lacking cycles and thus don't deadlock. At low number of faults, which is expected to be the common case, almost all the topologies are deadlock-prone.

Routing Algorithms

As described in previous sections, each Network Interface Controller (NIC) has a routing table that is used to embed the path in each packet. The reconfiguration of network to populate the routing tables at the NIC for routing over faulty irregular topologies is based on the prior works in this domain [17, 18, 21]. With synthetic traffic patterns, if the destination is not reachable from the source, the packet is simply dropped. With real application traffic (PARSEC 2.0 and Rodinia), the application is mapped on cores that are a part of a connected sub-network. In addition to being connected, these topologies also have paths in the remaining topology connecting memory controllers.

4.2 Configuration and Baselines

gem5 [37] full system simulator with the *GARNET* [38] network model has been used for all the evaluations. Network energy and area is estimated using DSENT [33] using 32 nm technology node and 2 GHz clock frequency. Table 4.1 lists the system configuration. The

following two state of the art works have been used as baselines for comparison with the static bubble scheme.

Deadlock Avoidance with Spanning tree. Deadlock avoidance using spanning-tree based routing with the spanning tree constructed using the up-down routing algorithm similar to that used by state of the art in works NoC resiliency [17, 18, 23, 21] and power-gating [19] domains. To strengthen this baseline further zero cycle reconfiguration time has been assumed for the spanning-tree construction although this cost is usually in thousands of cycles [17, 21]. All the packets for this baseline simulation come embedded with a deadlock-free route.

Deadlock Recovery with escape VCs. The second baseline is a deadlock recovery scheme that uses spanning-tree based paths for the escape network. Upon detection of a deadlock, packets move to the escape VC. Once in escape VC they have to use the spanning-tree based route until they reach their destination [32, 20]. The path is given by a routing table present at every router, that is configured with spanning-tree based deadlock free paths. Packets in regular VCs use minimal routes set by the source with 1-cycle router. Again to strengthen the baseline, zero cycle reconfiguration cost has been assumed for spanning-tree construction.

4.3 Network Performance and Energy Analysis

I start by first presenting the performance sweep of entire design space of irregular topologies with synthetic traffic.

Low-Load Latency

Figures 4.1 and 4.2 plot the average latency of Static-bubble, Spanning-tree and escape-vc based deadlock freedom solutions at low injection rate (0.01 packet per node per cycle). The injection rate of real applications is very close to this. Results of all the three schemes have been normalized to the Spanning-tree scheme. The simulations have been

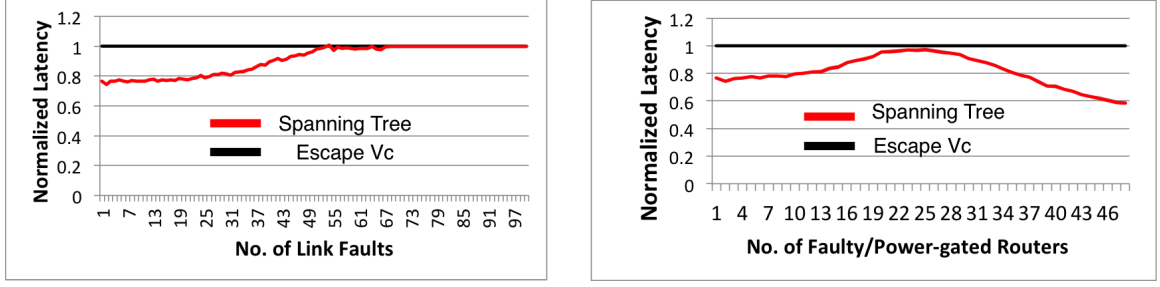


Figure 4.1: Average and max network latency improvements demonstrated by Static Bubble, normalized to Spanning-tree, across the irregular topology space with uniform random traffic at low-loads.

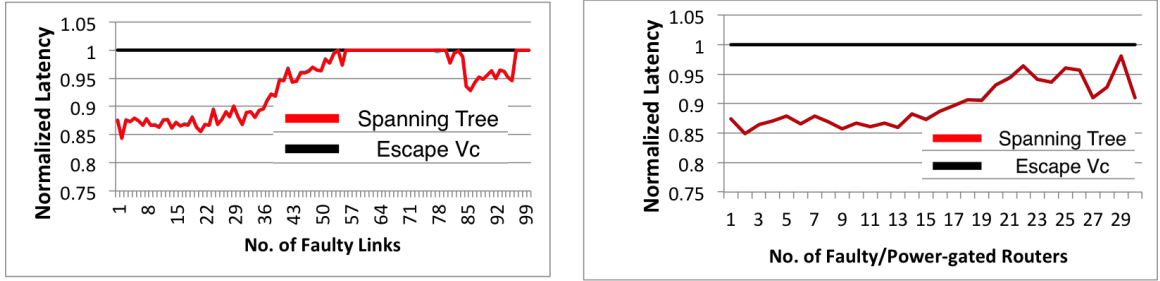


Figure 4.2: Average and max network latency improvements demonstrated by Static Bubble, normalized to Spanning-tree, across the irregular topology space with bit-complement traffic at low-loads.

carried out for both link and router faults using two different synthetic traffic patterns : (a) uniform-random and (b) bit-complement traffic patterns. As deadlocks don't occur at low injection rates, both static bubble and escape-vc show similar performance providing upto 22% improvement with uniform random traffic pattern and upto 15% improvement with bit-complement traffic pattern at low number of link and router faults. This reiterates the motivation story for this work that restricting path diversity in an already irregular topology that the spanning-tree based deadlock avoidance scheme does is very conservative and leads to performance loss.

Beyond 53 link faults, the topologies become fragmented and have very little path diversity that can be exploited, so minimal routes show similar performance compared to spanning-tree. The trend for router faults is similar to link faults except that at high router faults the network traffic is reduced due to the faulty routers being taken out of the topology

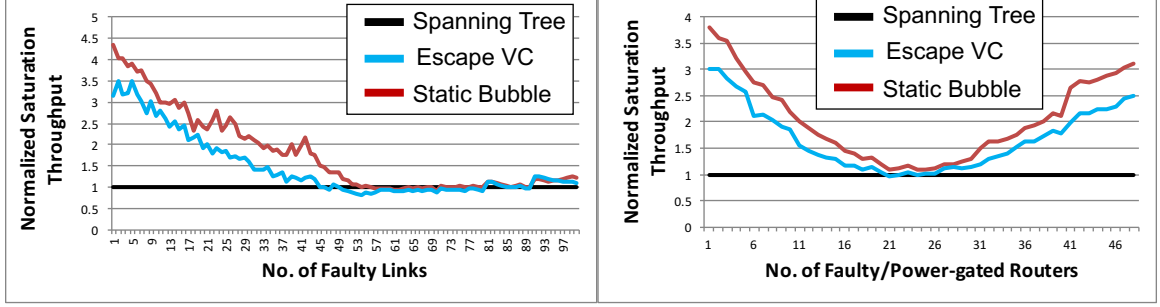


Figure 4.3: Average network throughput, normalized to Spanning-tree, of all designs across the irregular topology space with uniform random traffic.

graph. This leads to some latency improvement for minimal traffic compared to spanning-tree baseline. Again, recovery schemes perform better than avoidance scheme.

Network Throughput

Figure 4.3 plots the average network saturation throughput as a function of link and router faults with uniform random traffic. The throughput figures have been normalized to the spanning-tree baseline scheme. Static Bubble provides upto 3.5x to 4x more throughput compared to the spanning-tree based deadlock avoidance solution. This is due to the turn restrictions imposed by the spanning-tree scheme that disallows packets to use links even though they are otherwise healthy and fully-functional. Packets from one sub-tree have to go through the root if their destination lies in some other sub-tree. Thus, the packets are forced to take non-minimal paths which leads to early network saturation.

Compared to the escape-vc scheme, static bubble also provides a 1.2x to 1.3x higher throughput even though both the schemes are based on deadlock recovery. This is because the escape-vc reserves one VC per vnet per input port in all routers for the escape network. Consequently, this VC cannot be used by regular flits unless there is a deadlock. Thus, due to the reduction in number of buffers available for performance compared to the static bubble design where all VCs are available for use by regular flits, the escape-vc design shows poor performance in terms of throughput.

After 55 link faults, the topologies become highly fragmented offering little path di-

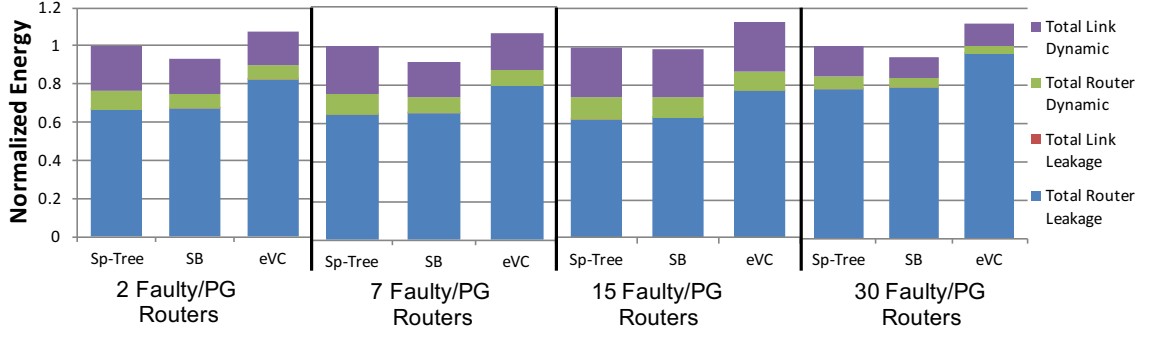


Figure 4.4: Average Network Energy.

versity and thus minimal routing saturates the network at the same point as spanning-tree based routing. Thus the performance of all three designs is similar. At around 21 router faults, the performance of all three designs is very similar. This is because there are very few deadlock-prone topologies at this fault number. Beyond 21 router faults, the network is partitioned and there are cycles within each fragment that lead to performance improvements for static bubble and escape vc compared to spanning-tree based solution. Static bubble still outperforms escape-vc solution in terms of network saturation throughput. It is interesting to note the similar trends shown by the latency and throughput graphs for link and router faults.

Energy

Figure 4.4 plots the average network energy for four different router fault number calculated using DSENT [33] with uniform random traffic at low injection rate (0.01). Across the design space, static bubble design shows a 20% energy reduction compared to escape-vc solution and 10% energy reduction compared to spanning-tree solution. In addition, static bubble shows about 22% reduction in leakage energy at low fault number. Static Bubble can thus be used to increase the static energy savings of existing power gating solutions. It is interesting to note the high leakage energy consumption of the escape-vc scheme in all fault scenarios. This happens because VCs are kept reserved for the escape network which continuously leak energy at low loads without contributing to performance. At high

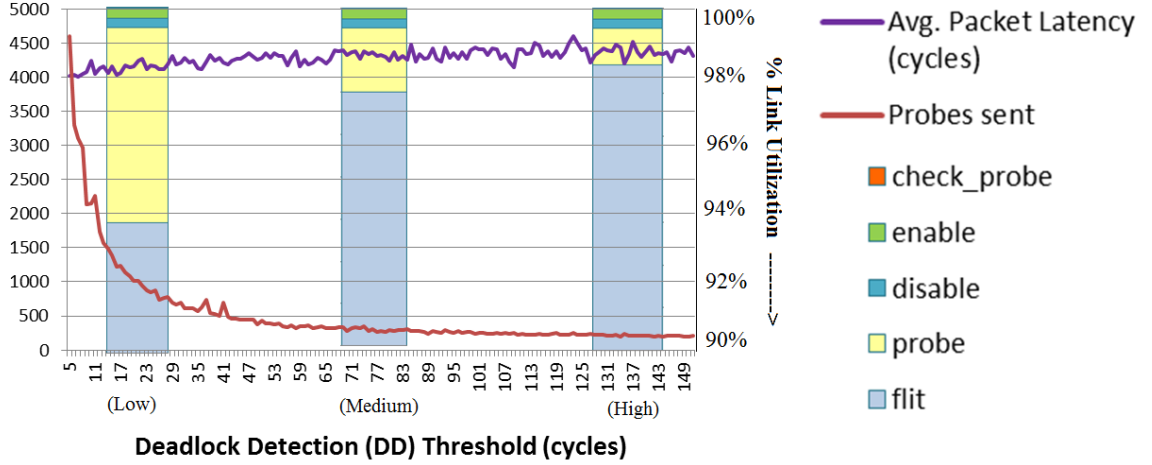


Figure 4.5: Deadlock Detection threshold t_{DD} sweep.

number of router faults, leakage becomes a larger part of the total energy consumption consumption. This happens due to the fragmentation of the topology which reduces the average hop count, leading to a dip in dynamic energy consumption. Even in this case, static bubble provides 20% more energy savings compared to spanning-tree scheme and about 40% compared to escape vc.

4.4 Deadlock Detection Threshold Sweep

Deadlock Detection Threshold (t_{DD}) is the only configurable parameter in the static bubble scheme. To recall, t_{DD} is the time allotted for a flit to leave the router failing which a probe would be sent out to get the deadlock path. In this section, I present the results as this configurable parameter is changed.

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, it is 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 as the flit would leave within this time. Figure 4.5 sweeps t_{DD} at high loads, when the network is deadlock-prone (Fig 1.5) for 10K cycles. The NoC has 20 router faults and the average value across all topologies are plotted. A

very low value of t_{DD} results in over 4000 (0.4 per cycle) probes being generated across the NoC. As t_{DD} increases, there is an exponential decline in the number of probes being sent out and it saturates to a total of about 200 (0.02 per cycle) probes generated across the NoC in 10K cycles.

While more probes steal bandwidth from flits, it turns out that probes are being sent only when there is a deadlock in which case the links are idle. At this point the network is saturated which is reflected by the extremely high packet latency. It is worthwhile to recall here that the number of probes sent out does not affect the functional correctness of the design; it just affects the time to detect deadlocks and impacts the total link energy consumption of the design. While we did not see any noticeable difference in the average latency of a flit as the threshold is varied since the link usage of probes is orders of magnitude less than flits, a slight improvement in packet latency is seen at low t_{DD} as the deadlocks get detected faster.

Figure 4.5 also shows the link utilization of different message classes as t_{DD} is varied. Link utilization of 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 don't steal bandwidth from regular flits that remain the dominant users of the NoC with greater than 93% link utilization at all values of t_{DD} .

4.5 Real Applications

Figure 4.6 plots the application throughput of Rodinia [11] benchmarks as a function of link and router faults across the topology space, normalized to the spanning-tree baseline scheme. At low link and router faults, the system with static bubble consistently outperforms both escape-vc and spanning-tree schemes by offering upto 2-4x more throughput than the spanning-tree baseline and 1.3-2x more throughput than escape-vc baseline. The

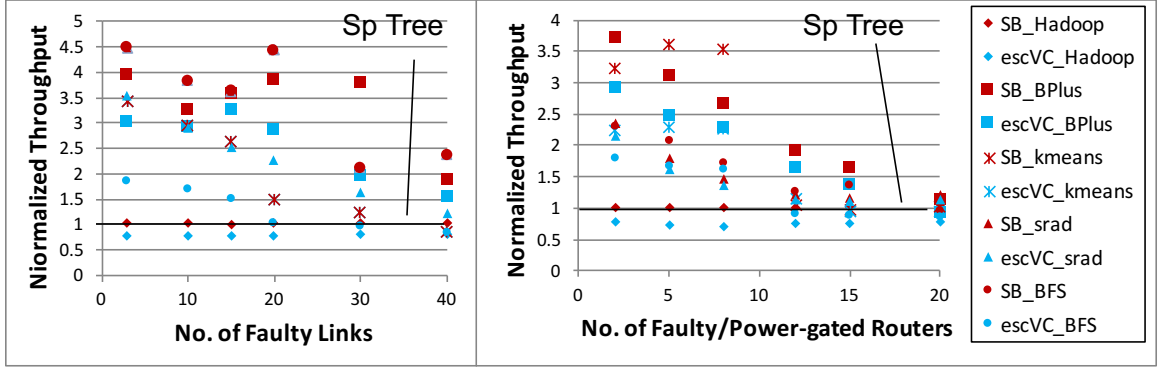


Figure 4.6: Scatter plot of application throughput with escape-vc and static bubble, normalized to spanning tree scheme, for Rodinia workloads, with increasing link and router faults.

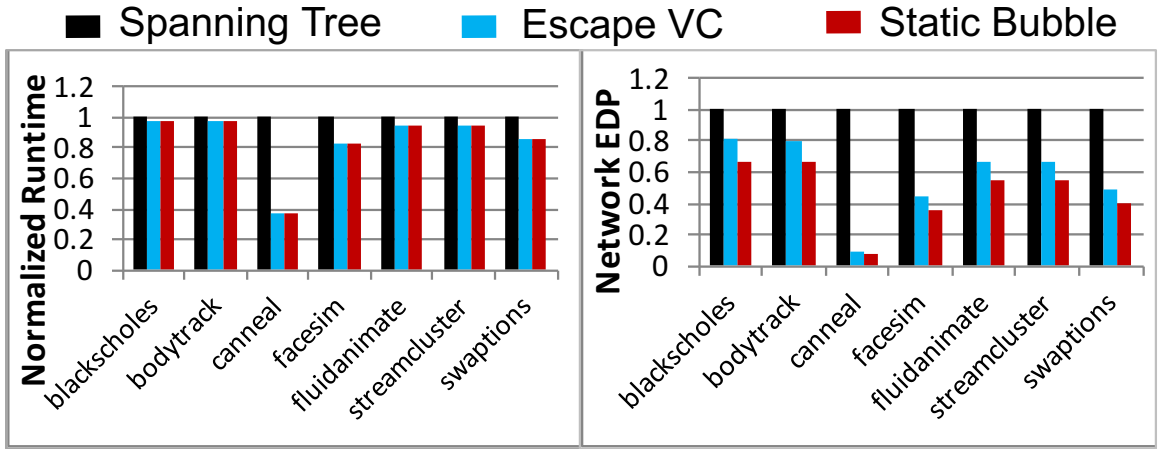


Figure 4.7: (a) Application Runtime and (b) Network EDP for PARSEC with four link faults.

only exception is *Hadoop* which shows similar performance with all designs due to the high collective traffic which saturates all the NoCs very early. At high link fault rates, *BPLUS* and *SRAD* show throughput improvements with static bubble while the performance of both the baselines drop. At 20 router faults, all designs perform almost identically on all rodinia benchmarks as very few connected topologies to run Rodinia exist at this fault rate and there is hardly any path diversity that the minimal routing schemes can exploit. Dead-lock occurrence (and resolution) was observed in some instances of the topologies at low number of faults with Hadoop, SRAD and BFS traffic.

Figure 4.7 plots the normalized runtime and Energy-Delay Product (EDP) of PARSEC

[10] benchmarks for all three schemes. PARSEC was run on a connected 8*8 mesh with four link faults for all the designs. As the injection rates of real applications are low, deadlock recovery schemes (static bubble and escape-vc) outbeat the deadlock avoidance schemes (spanning-tree) in terms of application runtime due to their highly conservative approach for achieving deadlock freedom which impacts performance. Results obtained with 32 router faults were very similar and hence haven't been shown. The benefit of static bubble over escape-vc for PARSEC benchmarks is seen in the EDP graphs plotted in Figure 4.7(b). Static bubble has 53% lower EDP compared to spanning-tree design and 17% lower EDP compared to escape-vc design. This is because non-minimal routing in spanning-tree design leads to increase in dynamic energy consumption while buffer reservation for the escape network leads to increase in leakage power consumption in the escape-vc design.

CHAPTER 5

CONCLUSION

Current solutions for achieving deadlock freedom in irregular topologies require expensive spanning tree construction and non-minimal routing over it. The alternative of using escape VCs still requires such a tree for providing a deadlock-free escape path. The network needs to be reconfigured to construct the spanning-tree every time there is a fault or a network element is power-gated. As faulty or power-gated irregular topologies are expected to be the common case in future, we need a low overhead scheme for deadlock freedom that doesn't cause loss in performance. In this work, I showed through state space exploration that while most irregular topologies are deadlock-prone, the actual occurrence of deadlocks at runtime is rare. I presented 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. Static Bubble provides a low-cost mechanism for deadlock recovery and performs better than state-of-the-art solutions in terms of latency, throughput and energy. In addition, it does not require any tree construction and can augment current solutions in the space of heterogeneous SoC design, NoC resiliency and NoC power-gating.

REFERENCES

- [1] R. R. Schaller, “Moore’s law: Past, present, and future,” *Ieee spectr.*, vol. 34, no. 6, pp. 52–59, Jun. 1997.
- [2] “DDR3 SDRAM Memory, en.wikipedia.org,”
- [3] J. Kim *et al.*, “Energy characterization of a tiled architecture processor with on-chip networks,” in *Islped*, 2003.
- [4] M. McKeown *et al.*, “Piton : A manycore processor for multi-tenant clouds,” *Ieee micro*, 2017.
- [5] B. K. Daya, C.-H. O. Chen, S. Subramanian, W.-C. Kwon, S. Park, T. Krishna, J. Holt, A. P. Chandrakasan, and L.-S. Peh, “Scorpio: A 36-core research chip demonstrating snoopy coherence on a scalable mesh noc with in-network ordering,” in *Proceeding of the 41st annual international symposium on computer architecture*, ser. ISCA ’14, Minneapolis, Minnesota, USA, 2014, pp. 25–36, ISBN: 978-1-4799-4394-4.
- [6] R. H. Dennard, F. H. Gaensslen, H.-n. Yu, V. L. Rideout, E. Bassous, Andre, and R. Leblanc, “Design of ion-implanted mosfets with very small physical dimensions,” *Ieee j. solid-state circuits*, p. 256, 1974.
- [7] H. Esmailzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, “Dark silicon and the end of multicore scaling,” in *Proceedings of the 38th annual international symposium on computer architecture*, ser. ISCA ’11, San Jose, California, USA: ACM, 2011, pp. 365–376, ISBN: 978-1-4503-0472-6.
- [8] Y. Shin, J. Seomun, K.-M. Choi, and T. Sakurai, “Power gating: Circuits, design methodologies, and best practice for standard-cell vlsi designs,” *Acm trans. des. autom. electron. syst.*, vol. 15, no. 4, 28:1–28:37, Oct. 2010.
- [9] Z. Zhang, A. Greiner, and S. Taktak, “A reconfigurable routing algorithm for a fault-tolerant 2d-mesh network-on-chip,” in *Proceedings of the 45th annual design automation conference*, ser. DAC ’08, Anaheim, California: ACM, 2008, pp. 441–446, ISBN: 978-1-60558-115-6.
- [10] C. Bienia *et al.*, “The PARSEC benchmark suite: Characterization and architectural implications,” in *PACT*, 2008.

- [11] S. Che *et al.*, “Rodinia: A benchmark suite for heterogeneous computing,” in *Iiswc*, 2009.
- [12] W. J. Dally and C. L. Seitz, “Deadlock-free message routing in multiprocessor inter-connection networks,” *Ieee trans. comput.*, vol. 36, no. 5, pp. 547–553, May 1987.
- [13] A. Ramrakhyani and T. Krishna, “Static bubble : A framework for deadlock-free irregular on-chip topologies,” in *Proceedings of the 23rd ieee symposium on high performance computer architecture, hpca*, 2017.
- [14] J. Duato, “A new theory of deadlock-free adaptive routing in wormhole networks,” *Ieee trans. parallel distrib. syst.*, vol. 4, no. 12, pp. 1320–1331, Dec. 1993.
- [15] C. J. Glass and L. M. Ni, “The turn model for adaptive routing,” *J. acm*, vol. 41, no. 5, Sep. 1994.
- [16] K. V. Anjan and T. M. Pinkston, “An efficient, fully adaptive deadlock recovery scheme: DISHA,” in *Isca*, 1995.
- [17] K. Aisopos *et al.*, “ARIADNE: agnostic reconfiguration in a disconnected network environment,” in *Pact*, 2011.
- [18] R. Parikh and V. Bertacco, “Udirec: Unified diagnosis and reconfiguration for frugal bypass of noc faults,” in *Micro*, 2013.
- [19] R. Parikh *et al.*, “Power-aware nocs through routing and topology reconfiguration,” in *Dac*, 2014.
- [20] A. Samih *et al.*, “Energy-efficient interconnect via router parking,” in *Hpca*, 2013.
- [21] D. Lee *et al.*, “Brisk and limited-impact noc routing reconfiguration,” in *Date*, 2014.
- [22] D. Fick *et al.*, “A highly resilient routing algorithm for fault-tolerant nocs,” in *Date*, 2009.
- [23] V. Puente *et al.*, “Immunet: A cheap and robust fault-tolerant packet routing mechanism,” in *Isca*, 2004.
- [24] —, “The adaptive bubble router,” *J. parallel distrib. comput.*, vol. 61, no. 9, Sep. 2001.
- [25] 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.

- [26] A. Mejia *et al.*, “Segment-based routing: An efficient fault-tolerant routing algorithm for meshes and tori,” in *Ipdp*s, 2006.
- [27] L. Chen *et al.*, “Power punch: Towards non-blocking power-gating of noc routers,” in *Hpca*, 2015.
- [28] R. Das *et al.*, “Catnap: Energy proportional multiple network-on-chip,” in *Isca*, 2013.
- [29] 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.
- [30] A. Lankes *et al.*, “Comparison of deadlock recovery and avoidance mechanisms to approach message dependent deadlocks in on-chip networks,” in *Nocs*, 2010.
- [31] Y. H. Song and T. M. Pinkston, “A new mechanism for congestion and deadlock resolution,” in *Icpp*, 2002.
- [32] L. Chen and T. M. Pinkston, “Nord: Node-router decoupling for effective power-gating of on-chip routers,” in *Micro*, 2012.
- [33] C. Sun *et al.*, “Dsnt - a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling,” in *Nocs*, 2012.
- [34] S. Park *et al.*, “Approaching the theoretical limits of a mesh noc with a 16-node chip prototype in 45nm SOI,” in *Dac*, 2012.
- [35] W. J. Dally and B. Towles, “Route packets, not wires: On-chip interconnection networks,” in *Dac*, 2001.
- [36] M. Fattah *et al.*, “A low-overhead, fully-distributed, guaranteed-delivery routing algorithm for faulty network-on-chips,” in *Nocs*, 2015.
- [37] N. Binkert *et al.*, “The gem5 simulator,” *Sigarch comput. archit. news*, vol. 39, no. 2, Aug. 2011.
- [38] N. Agarwal *et al.*, “GARNET: A detailed on-chip network model inside a full-system simulator,” in *Ispass*, 2009.
- [39] A. Ahmed *et al.*, “AMD Opteron shared memory MP systems,” in *14th hot chips symposium*, Aug. 2002.