SUBACTIVE TECHNIQUES FOR GUARANTEEING ROUTING AND PROTOCOL DEADLOCK FREEDOM IN INTERCONNECTION NETWORKS

A Dissertation
Presented to
The Academic Faculty

By

Mayank Parasar

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Georgia Institute of Technology

Georgia Institute of Technology

August 2020

SUBACTIVE TECHNIQUES FOR GUARANTEEING ROUTING AND PROTOCOL DEADLOCK FREEDOM IN INTERCONNECTION NETWORKS

Approved by:

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

Dr. Hyesoon Kim
School of Computer Science
Georgia Institute of Technology

Dr. Alexandros Daglis
School of Computer Science
Georgia Institute of Technology

Dr. Paul Gratz
Department of Electrical and Computer Engineering
Texas A&M University

Dr. Moinuddin Qureshi
School of Computer Science
Georgia Institute of Technology

Date Approved: July 6, 2020

Remember to look up at the stars and not down at your feet. Try to make sense of what you see and wonder about what makes the Universe exist. Be curious. And however difficult life may seem, there is always something you can do and succeed at. It matters that you don't just give up.

Stephen Hawking

To my family.

ACKNOWLEDGEMENTS

Back in IIT Kharagpur, India, when I was an undergraduate student, I used to dread Ph.D. as I viewed it challenging and extraordinary endeavor to accomplish. Whenever I would see Ph.D. scholars, respectful feelings used to come from within. Fast forward to today, I am finishing my Ph.D. and this is the last chapter I am adding to my thesis. Looking back, I think fascination for Ph.D. and exceptional hard-work associated with it, encouraged me to quit my job at Nvidia and take up this remarkable endeavor. I can tell you this is by far the best thing that happened to me till date and all of this is because of wonderful people that I met and got a chance to work with during my Ph.D. journey. It is because of them that I could finish this journey, and all I can do is to acknowledge how grateful I am that our path crossed.

I am forever grateful to Dr. Tushar Krishna, my Ph.D. advisor, he has been a guiding light throughout my Ph.D. journey. It all started with my first class in Advance Computer Architecture (ECE6100) at Georgia Tech and it was his first-class too that he taught at Georgia Tech. He shaped me carefully throughout my Ph.D. and meticulously guided me. He always made himself available whenever I needed his guidance. I will always relish our discussions on various ideas, some of which culminated into papers and many could not see the light of the day but there were always learnings. I vividly remember, back when I was not his Ph.D. student, how at the end of every ICN/NoC (Interconnection Network/ Network-On-Chip) class I would go to him and ask, *what if we do . . . ?* and he would say you missed this concept or sometimes he would mention this work is already done. I adore his ever-encouraging attitude towards work. As I grew senior in my Ph.D. our discussion became more vibrant and intense. I look forward to working with him in the future!

I would like to extend my heartfelt gratitude to Dr. Paul V. Gratz. He is my collaborator for all my network deadlock papers and projects; and served on my Ph.D. committee. He always encourages and challenges me to further push the limits of the idea. His insights on

various NoC ideas, we discussed over the years, have been invaluable. He taught me many valuable insights about NoC varying from fundamental concepts to simulation strategies. I admire the jovial nature of Dr. Paul and learning from him over the years has been a privilege.

Through my Ph.D. journey, I got to work with wonderful collaborators. Dr. Natalie Enright Jerger taught me on improving my presentation talk skills and provided her unbiased views on many of the research ideas. I relished working with Dr. Joshua San Miguel over many NoC projects. I admire his insights in poking and finding holes in the idea. Over the years, discussions with him have helped in building strong fundamentals in Network-On-Chip.

SEESAW was my first paper on virtual memory in collaboration with Dr. Abhishek Bhattacharjee. As a fresh Ph.D. student, I learned a lot through this paper, about various aspects of virtual memory and computer architecture. Many thanks to Dr. Abhishek and Dr. Tushar for being so patient with me throughout this project. This paper by far has seen most rejects in my Ph.D. career, this paper taught me the power of perseverance and the value of the peer-reviewed system.

This paper then became the stepping stone of my internship at AMD research where I worked on another project on virtual memory. I got to interact and work with industry veterans; it was truly a remarkable learning experience throughout my Ph.D. career. Learnings I received from my internship at AMD research later shaped the rest of my Ph.D. path. I like to thank Dr. Arkaparva Basu, Eric Van Tassell, Dr. Michael Wayne LeBeane, and Sooraj Puthoor for their help and encouragement during the internship.

I would like to extend my sincere thanks to Dr. Alexandros Daglis, interacting with him on my research, and later on, the thesis draft has improved the quality of many of my techniques. His insights and suggestions during my Ph.D. defense have been very valuable. I thank Dr. Hyesoon Kim and Dr. Moin Qureshi for helping me improve presentations of several of my techniques during the Arch-whisky practice talk at Georgia Tech. They also

served on my Ph.D. committee.

Early in-person feedback on my work by Dr. Timothy M. Pinkston helped me to critically think about many of the techniques proposed in this thesis. He also suggested to position some of the proposed techniques for reconfigurability deadlocks; where two routing algorithms can be independently deadlock-free, while dynamically switching between them can result in deadlock. I thank Dr. Tim for the feedback and time.

I interacted with late Dr. Sudhakar Yalamanchili on my first day at Georgia Tech in the capacity of Teaching Assistant (TA) for one of his undergraduate computer architecture courses (ECE-3056). I worked as a TA for him for two semesters before switching to Research Assistant (RA) with Dr. Tushar Krishna. Back then Georgia Tech used to have written Ph.D. qualifier examinations in ECE, and we have to pass it before we could formally start the Ph.D. It used to be a daunting challenge, as we were given limited chances. Dr. Sudhakar made sure that I understand his course well, to do good in the qualifier exam. He mentored me during those two semesters on various aspects of being a TA. I will miss him.

I find myself to be fortunate that during my Ph.D. I got to be part of Asha society's running team. Back in IIT Kharagpur, I started running 2.2 (road circuit of 2.2 kilometers in length, popularly known as 2.2) in my senior year, but I was not regular. Over at Georgia Tech, after clearing my qualifiers and securing a position in Dr. Tushar Krishna's lab, I resumed running on treadmills (on and off) until finally, someone recommended me to join Asha's running team, which raises funds to help underprivileged women and children, by taking part in marathons. Through Asha, I came in contact with many friends: Prasoon Suchandra, Arvind Krishna, Dr. Samarth Brahmbhatt, Diego Vaca, Dr. Chirag Jain, Dr. Dhwanil Shukla. Learning from them about various running techniques and practice runs on Saturday mornings proved to be very helpful in improving my overall health. I look forward to doing practice runs with you all once things get to normal.

Thank you, Dr. Steffen Maass, for your helpful advice. Best wishes to you! I miss

conversing with Dr. Mohan Kumar and Dr. Sanidhya Kashyap in Klaus hallway. Learning from you about your work is always enriching. Dr. Anosh Daruwalla, thank you for your help in Ph.D. qualifier preparation. Interacting with you, last summer and fall were very uplifting. Thank you, Chetan Kale, for your support and encouragement.

I should admit ECE Ph.D. course work can take its toll (+ pressure of maintaining an excellent GPA). I give it to my friends for the reason for maintaining a 4.0 GPA. Thank you, Shu-han Hsu, for working together on ECE-6601 (Random Processes) assignments, going to TA office hours; you kept me motivated to work harder and improve.

Giving presentations and making slides is a skill that gets better with suggestions. Thank you, Poulami Das, for feedback on many of my talks including Ph.D. proposal practice talks and valuable suggestions.

Being the part of Synergy group from the beginning, I have got the chance to interact with every synergy member. I remember the time Hyoukjun and I were the only members working in Klaus-3305 lab. It used to be mostly empty back then. I am happy to see our lab growing in size. Congratulations to both of us for graduating in summer-2020!

Thank you, Anand, for all the suggestions on paper drafts and poster drafts over the years. Your careful suggestions have helped improve the quality of several of my paper drafts. I first met Eric at a graduate student hosting event, when he came to Georgia Tech after undergrad from ASU. Thanks Eric, for all the great discussions. I love interacting with Felix, Saeed, and Yehowshua about best practices in coding. Best wishes to Matthew, as a new synergy lab member. Thank you, Ankit for all the discussions and words of wisdom. Best wishes to Geonhwa, and William (Jonghoon); To future synergy lab members I look forward to interacting with you.

My family has been an enormous source of support throughout my Ph.D. journey. Thank you very much *Jijaji* (brother-in-law, Madhur Mayank Sharma) and *Didi* (elder sister, Anubha Sharma) for all the valuable advice whenever I needed. Conversing with you has kept me motivated. I am looking forward to spending more time with my Nephew

(Ikshaan Sharma) and my Niece (Miraya Sharma), your jovial nature has kept us all delighted.

Finally, and most importantly I am deeply, deeply grateful to my parents. You have been a constant source of guidance throughout my life, all the great qualities you instilled in me, has made this extraordinary endeavor, Ph.D., possible. Words cannot do justice for all that you have done for me. Without your unwavering support, encouragement, and love I would not be where I am today. This dissertation is dedicated to my parents. Here is to the first Dr. Parasar (Dr. Parashar) in our Family! Cheers!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Interconnection networks are the communication backbone for any system. They occur at various scales: from on-chip networks between processing cores, to supercomputers between compute nodes, to data centers between high-end servers. One of the most fundamental challenges in an interconnection network is that of deadlocks. Deadlocks can be of two types: routing level deadlocks and protocol level deadlocks. Routing level deadlocks occur because of cyclic dependency between packets trying to acquire buffers, whereas protocol level deadlock occurs because the response message is stuck indefinitely behind the queue of request messages. Both kinds of deadlock render the forward movement of packets impossible leading to complete system failure.

Prior work either restricts the path that packets take in the network or provisions an extra set of buffers to resolve routing level deadlocks. For protocol level deadlocks, separate sets of buffers are reserved at every router for each message class. Naturally, proposed solutions either restrict the packet movement resulting in lower performance or require higher area and power.

In this thesis, we propose a new set of efficient techniques for providing both routing and protocol level deadlock freedom. Our techniques provide periodic forced movement to the packets in the network, which breaks any cyclic dependency of packets. Breaking this cyclic dependency results in resolving routing level deadlocks. Moreover, because of periodic forced movement, the response message is never stuck indefinitely behind the queue of request messages; therefore, our techniques also resolve protocol level deadlocks. We use the term ‘subactive’ for these new class of techniques.

CHAPTER 1
INTRODUCTION

The complexity for minimum component costs has increased at a rate of roughly a factor of two per year... Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years.
- Gordon E. Moore, Cramming more components onto integrated circuits, Electronics Magazine, 19 April 1965.

Gordon Moore's observation on the economically viable number of components per integrated circuit is popularly called Moores Law, and continues till today, well beyond the 10 years he initially believed it would last. In the semiconductor industry, this law has become the de facto driver for technological innovation and has led to a sustained doubling of the number of transistors on a die approximately every 2 years.

1.1 Multi-Core Era

Moores Law coupled with Dennard's scaling[1], MOSFET dimensions and operating voltages should be scaled by the same factor to keep electric field constant, allowed each technology generation to produce twice the number of transistors, with each transistor 1.4 $\times$ faster than previous generation at the same power density within the same area. However, in early 2000s, Voltage scaling slowed down because chips were already operating close to threshold voltage physical limit at which transistors turn ON and OFF. The end of voltage scaling also led to the end of frequency scaling to ensure that chips do not cross the power wall (100W) and overheat, as power equals capacitance $\times$ frequency $\times$ voltage-squared. Because of this, and due to ILP (Instruction Level Parallelism) limitations, it was no longer possible to get similar performance gains per unit power as before. Instead, computer

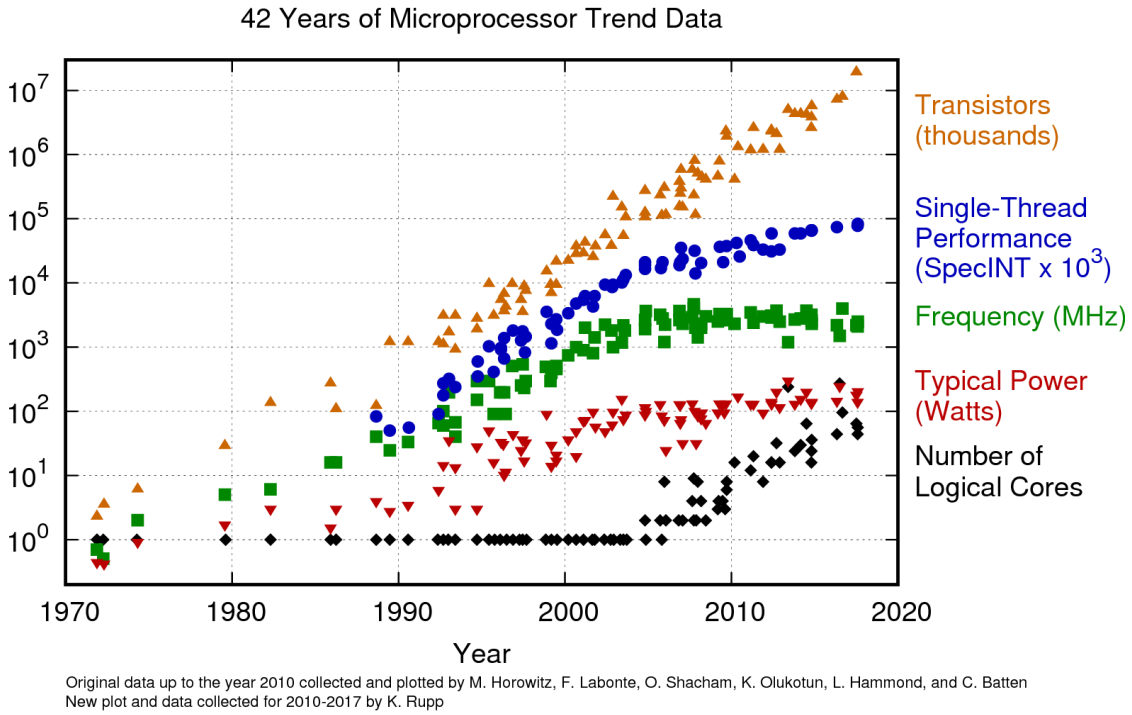


Figure 1.1: The y-axis shows the number of transistors, and x-axis shows the time in years. To keep churning more performance after Dennard scaling[1], we see resurgence of parallel applications and number of cores starting from 2005 onwards.

architects decided to extract performance by multiplying the number of processing cores on-chip (using the exponentially growing number of transistors from Moores law) and running them in parallel. This has led to the current wave of Chip Multiprocessors (CMPs) or Multicores. Figure 1.1 and Figure 1.2 shows how increasing transistor count resulted in a greater number of cores post Dennard multi-core era.

1.2 Network-on-Chip

As the number of on-chip cores increases, a scalable and high-bandwidth communication fabric to connect them becomes critically important. As a result, packet- switched on-chip networks are fast replacing buses and crossbars to emerge as the pervasive communication fabric in many-core chips. Such on-chip networks have routers at every node, connected to neighbors via short local on-chip wiring, while multiplexing multiple communication flows over these interconnects to provide scalability and high bandwidth. This evolution

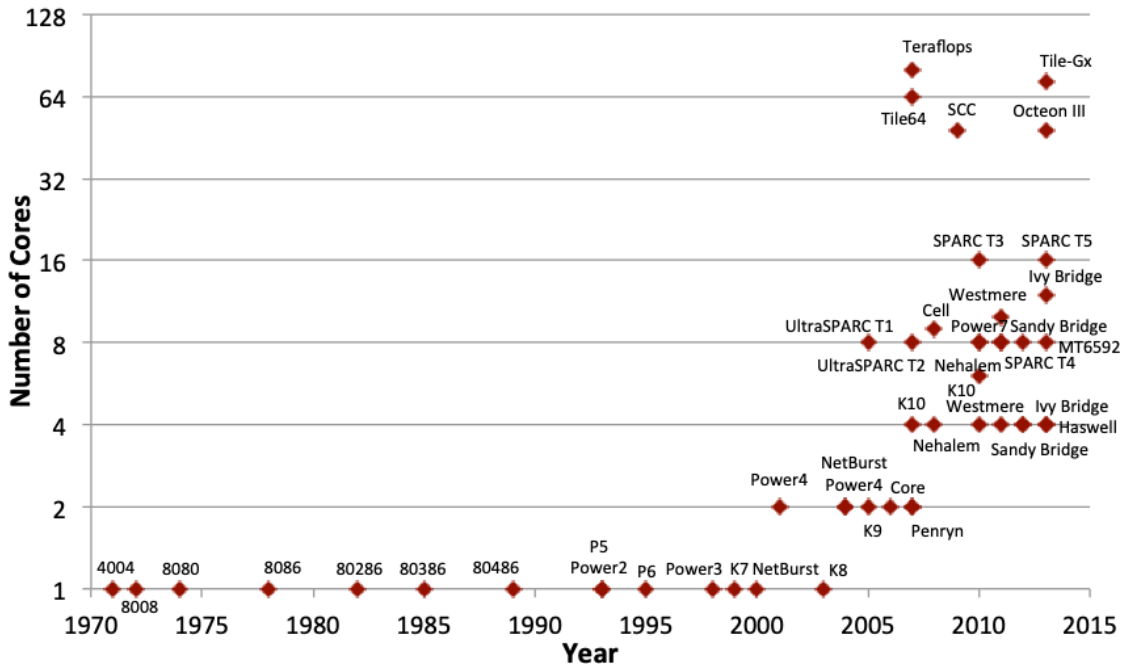


Figure 1.2: The y-axis shows the number of cores, and x-axis shows the time in years. Commensurate with Figure 1.1 we observe an increase in the number of cores, to keep up increasing performance.[5]

of interconnection networks as core count increases are clearly illustrated in the choice of a flat crossbar interconnect connecting all eight cores in the Sun Niagara (2005)[2], four packet-switched rings in the 9-core IBM Cell (2005)[3], and five packet- switched meshes in the 64-core Tilera TILE64 (2007)[4].

Figure 1.3 shows a tiled CMP (Chip Multi-Processor), here each tile is shown in a gray rectangle(tile). Each tile contains a processor core, its private instruction and data cache, a slice of shared L2 cache and an on-chip *router*. On chip router is shown in green color rectangle and its zoomed-out figure is shown on the right. The router shown here is a *virtual channel* router. This means there is a single physical channel (or link) over which packets interleave to next hop router and finally reach their destinations. More about virtual channel routers and physical channel routers are explained in chapter 2.

On-chip network uses input buffered routers, that is, buffers are present at the input port of the router to store the incoming packets coming from neighboring routers/core. Buffers serve following purpose:

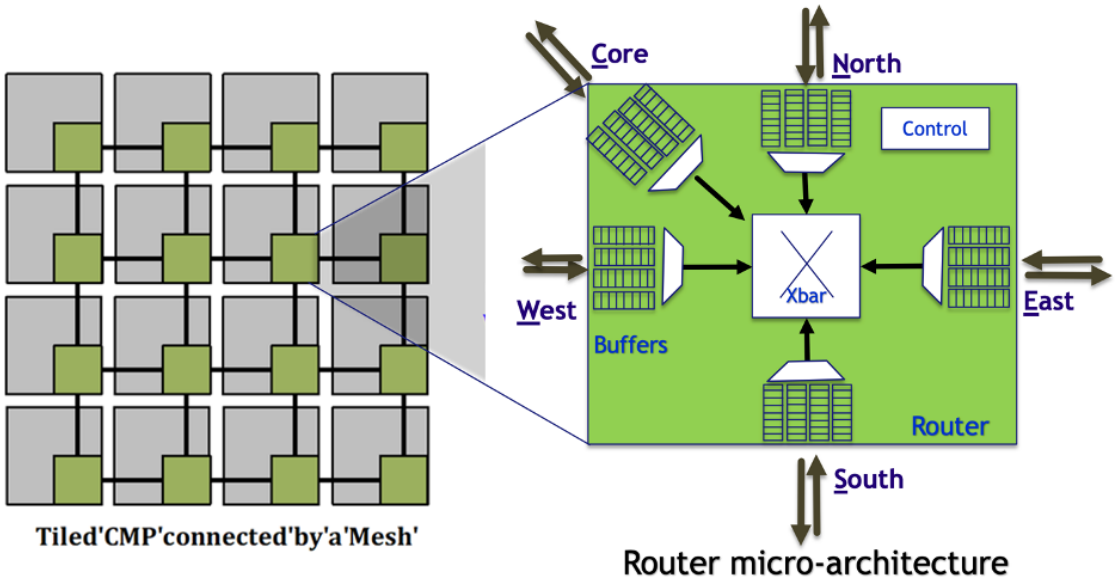


Figure 1.3: Figure shows the 4x4 Mesh topology and zoomed out router micro-architecture of an on-chip network.

- They allow packets to be routed minimally to their destination
- If two packets present at different input port (say North and East) want to travel to south, then only one packet can travel at a given time other packet needs to be buffered to try again next time
- In the face of congestion at the downstream router, these buffers hold the packet until congestion clears

Buffers in the network are also important to provide *deadlock freedom* as explained in later chapters.

1.3 Deadlocks

A deadlock is a situation in which a set of *agents* wait indefinitely trying to acquire a set of *resources*. From an interconnection network perspective, agents are the network packets, and resources are the buffers in the network which temporarily store the network packets as move in the network to reach their destination. The cyclic dependency where network

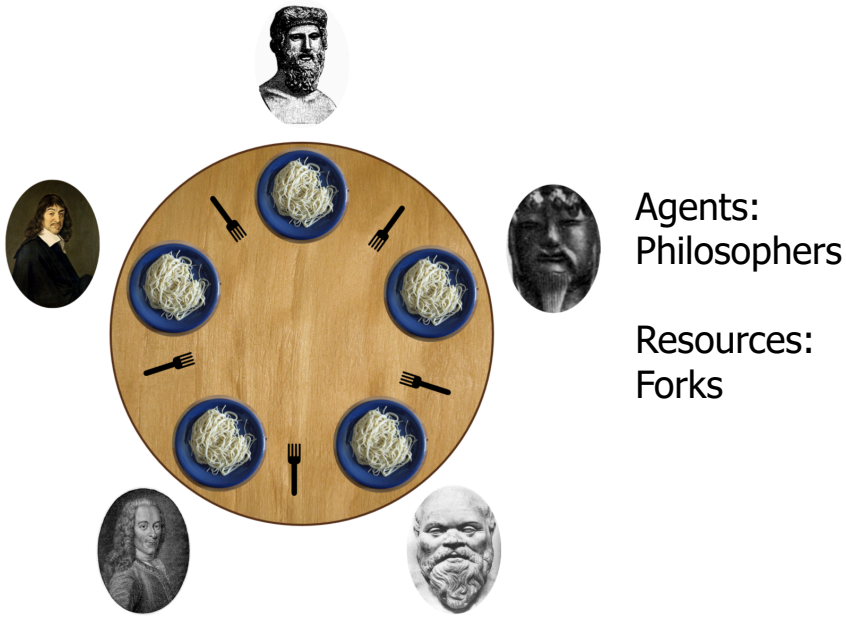


Figure 1.4: Dining philosopher problem: Each philosopher can only eat if he has two forks for him. Here we have five philosopher and five forks. All philosophers take their left-hand side fork together and wait on next philosopher to free their fork in a cyclic manner while holding their own fork. This results in a deadlock, and no one could eat the meal. This is the philosophy behind deadlocks!

packets(agents) are trying to acquire network buffers (resources) results in a deadlock. The classic pedagogical example of deadlock is the dining philosopher's problem [6]. Formally, four conditions must simultaneously hold for agents to be involved in a resource deadlock in a computer system:

- **Mutual Exclusion:** Resources can be used only in a mutually exclusive manner.
- **Hold and wait:** An agent is allowed to hold a resource while waiting for other resources
- **Circular wait:** There is a cyclic dependence among the agents waiting for resources (i.e., A is waiting for resource held by B; B is waiting for a resource held by C; ... ; Z is waiting for a resource held by A).
- **No preemption:** An agent holding a resource has to give it up voluntarily.

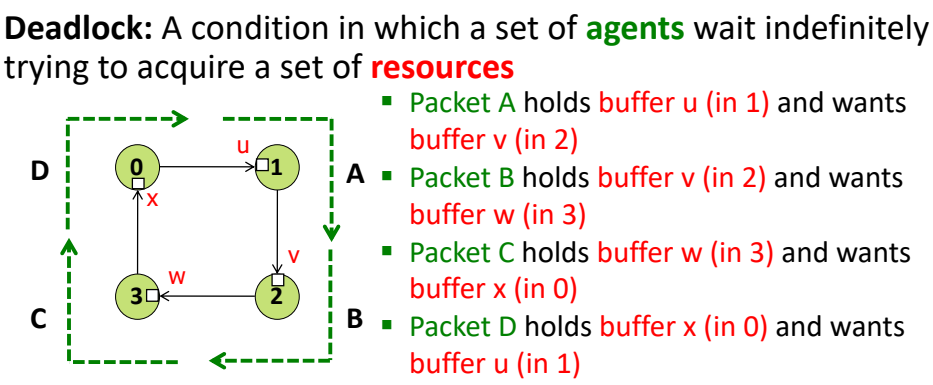


Figure 1.5: Theoretical condition for deadlock

1.3.1 Why Deadlocks Matter?

It is important at the outset to point out that deadlocks are a *correctness* issue, not a performance issue. Any multiprocessor communications and interactions in a system (e.g., OS processes in software, coherence messages in the memory system, packets in the interconnection network) need to be deadlock-free by design.

1.3.2 Deadlocks under Consideration

In this proposal, we consider two classes of deadlocks that plague multiprocessor systems. These are: Routing Level Deadlock and Protocol Level Deadlock. These deadlocks are explained in detail with figures in chapter 2

Our primary objective in this thesis is to rethink the way we provide deadlock-freedom in interconnects and communication protocols to decrease the expensive buffering that traditional deadlock-freedom mechanisms require.

1.4 Dissertation Contribution and Outline

In this thesis ***we show periodic coordinated packet movement is sufficient to resolve both routing-level and protocol-level deadlocks in any interconnection network***

The rest of the thesis is organized as follows:

- Chapter-2 presents relevant background on NoCs and cache coherence protocols. It explains the deadlocks that we have addressed in this thesis. It also presents our evaluation methodology and performance metrics

- Chapter-3 talks about prior work done in the field of deadlocks and classify the prior work under new taxonomy: Proactive, Reactive and subactive. *Subactive class of techniques to resolve deadlocks is the contribution of this thesis*

- Chapter-4 talks about tools developed for evaluation of the subactive class of techniques over regular and irregular topology. It also talks about the simulation tool, integrated with gem5[7], called as DRAGON which finds deadlock cycles in a given network topology

- Chapter-5 introduces the first subactive technique called as Brownian Bubble Router (BBR)[8], it provides deadlock freedom moving the packet involved in a deadlock to other input port within the router. Chapter discusses the changes to baseline router micro-architecture and performance evaluation

- Chapter-6 discusses next subactive technique: BINDU[9], which is an acronym for Bubble in Irregular Network for Deadlock pUrging. It proposes to provide deadlock freedom using one *bubble* or empty VC which keeps circulating through all the input ports of the routers in the network. Work further generalize the concept to k-BINDUs and show the performance impact.

- Chapter-7 discusses next new subactive technique: DRAIN[10]: Deadlock Removal for Arbitrary Irregular Networks. This work proposes to periodically and temporarily convert a given topology into a virtual ring and move packets few hops on that virtual ring. The normal network routing ensues afterwards. *Oblivious* movement of packets over virtual ring removes any deadlock if present. Chapter-8 discusses router micro-architecture and performance of this scheme

- Chapter-8 talks about a novel subactive technique: SWAP[11] which stands for Synchronized Weaving of Adjacent Packets for Network Deadlock Resolution. This works proposes to *swap* packets among neighboring routers in the network to provide deadlock freedom. The operation of *packet-swap* occurs periodically and distributed throughout the network to provide deadlock freedom. Unlike DRAIN[10], which globally moves all packets of the network periodically, SWAP[11] proposes localized *oblivious swapping* of packets across neighboring routers

- Chapter-9 talks about final subactive technique called as SEEC: Stochastic Escape Express Channel. It proposes to create direct connection between packets in the network to their destination routers. SEEC then routes those packets minimally till destination without buffering them over intermediate routers. Chapter shows a comprehensive evaluation of SEEC against prior subactive techniques

- Chapter-10 concludes this thesis and discusses the future research directions

CHAPTER 2
BACKGROUND

2.1 Network-on-Chip basics

An on-chip network, as a subset of a broader class of interconnection networks, can be viewed as a programmable system that facilitates the transporting of data between nodes. An on-chip network can be viewed as a system because it integrates many components including channels, buffers, switches and control.

With a small number of nodes, dedicated ad hoc wiring can be used to interconnect them. However, the use of dedicated wires is problematic as we increase the number of components on-chip: The amount of wiring required to directly connect every component will become prohibitive.

This chapter introduces readers to a Network-on-Chip and discusses its components in sufficient detail for understanding the thesis. Necessary background about both synthetic traffic and cache coherence protocol traffic used to drive the NoC is also presented.

2.2 Topology

Topologies define the connection and physical layout between nodes in the network. Topology has profound impact on the performance of the network, it decides the number of hops a message has to take to reach its destination router in the network. Number of hops directly correlate with the latency and energy expended due to data movement in the network.

As topology connects different cores/IPs which have different latency and bandwidth requirement for example, in an SoC, there are different types of topologies used in the on-chip network as shown in the Figure 2.2

There are following performance metrics associated with the topology, which helps us

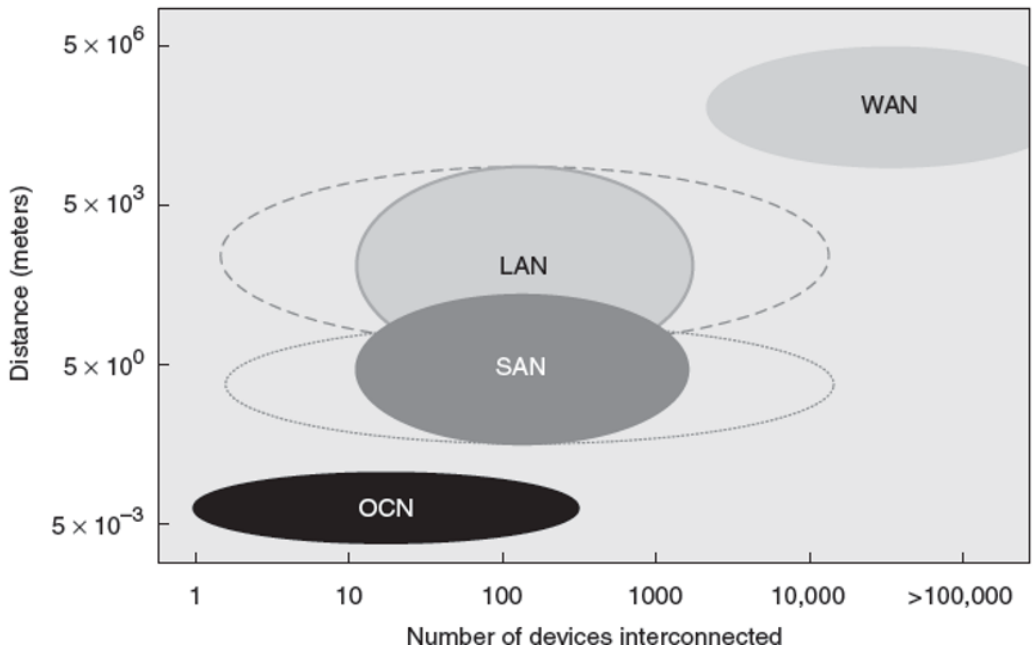


Figure 2.1: This Figure is taken from Hennessy and Patterson, 5th Edition, Appendix F.

to compare different topologies and reason about their trade-offs. These metrics are briefly described below:

- **Degree**. The degree of a topology refers to the number of links at each node. For example, a *ring* topology has degree 2, while torus topology has degree 4 for each node.

- **Bisection bandwidth**. The bisection bandwidth is the bandwidth across a cut that partitions the network into two equal parts

- **Diameter**. The diameter of the network is the maximum distance between any two nodes in the topology, *where distance is the number of links in the shortest route.*

- **Hop count**. The number of hops a message takes from source to destination, or the number of links it traverses, defines hop count.

A direct network is one where each terminal node (e.g. a processor core or cache in a chip multiprocessor) is associated with a router; all routers act as both sources/sinks of traffic and as switches for traffic from other nodes.

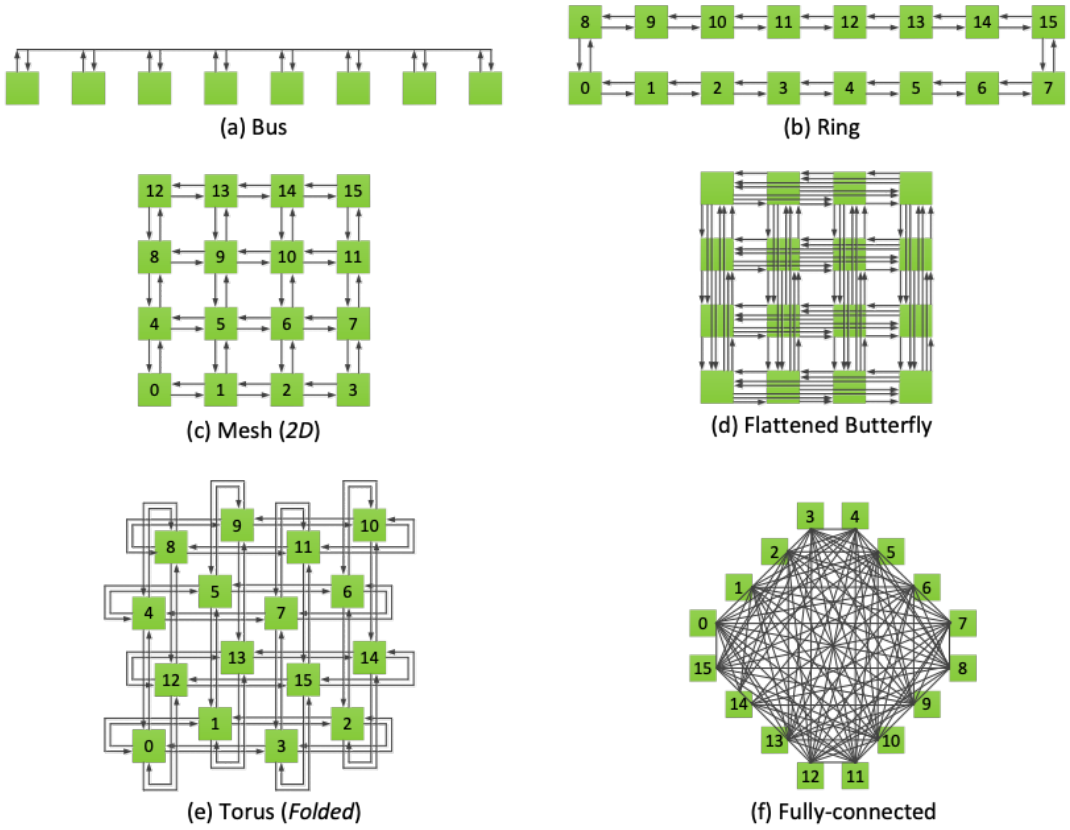


Figure 2.2: Common Network-on-Chip Topologies.[5]

Indirect networks connect terminal nodes via one or more intermediate stages of switch nodes. Only terminal nodes are sources and destinations of traffic, intermediate nodes simply switch traffic to and from terminal nodes. In this thesis we have assumed direct network in our evaluations, however, techniques can be extended to indirect network as well.

2.2.1 Faulty Topologies

As process technologies continue to shrink into the deep sub-micron domain, the breakdown of Dennard Scaling has meant that on-chip current densities are increasing as device density increases. Thus individual devices and wires are exposed to higher operating temperatures and currents, both of which are known to accelerate their eventual breakdown by one of a number of wear-out mechanisms, including Bias Temperature Instability [12],

Time-Dependent Dielectric Breakdown [13], Hot-Carrier Injection [14] and Electromigration [15]. In each case, heat and/or current accelerate wear, increasing the odds of individual device failure. When put together with increasing device and wire density, the odds of component failures on-chip are rising dramatically with each process generation. Traditional techniques, such as adding extra timing guard bands and wire thickening in vulnerable locations are no longer sufficient to address this growing problem; thus architectural techniques to deal with wear-out failures during product lifetimes must be developed [16].

These eventual component failures imply the expectation that individual cores and other components may fail during the lifetime of the product. An individual core or other redundant component failure can be dealt with via detection hardware and associated fail-over software, allowing continued operation at lower capacity [17]. However, failures of the interconnect components (e.g. links, routers) can be more challenging. In networks-on-chip (NoCs), applying routing restrictions is the most common deadlock-freedom mechanism; yet it requires static and regular network topologies. Thus, as links and routers fail, these routing restrictions may be violated, leading to potential deadlocks. Similarly, for topologies that are irregular [18] or random [19] by design, traditional deadlock avoidance techniques that rely on network regularity do not work.

Existing mechanisms to handle router and link wear-out failure [20, 21, 22, 23] require significant extra hardware to support runtime routing reconfiguration and often create strong network hot spots due to the need to ensure deadlock avoidance in the newly irregular network.

2.2.2 Irregular Topologies

MPSoC design may leverage a wide variety of heterogeneous IP blocks; as a result of the heterogeneity, regular topologies such as a mesh or a torus described above may not be appropriate. With these heterogeneous cores, a customized topology will often be more power efficient and deliver better performance than a standard topology.

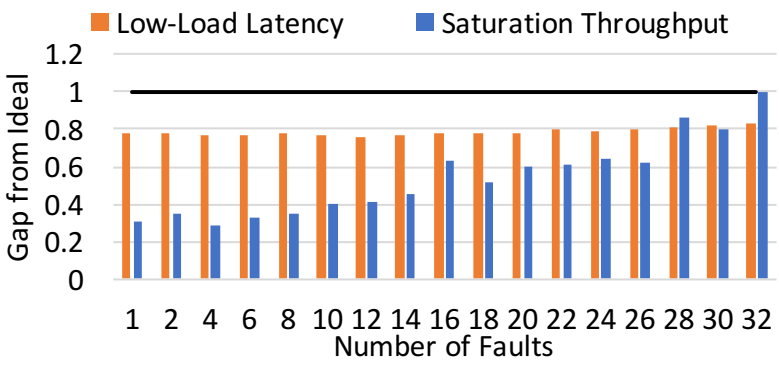


Figure 2.3: Gap between low-load latency and saturation throughput for up*/down* routing and ideal (shown as a black line at 1).

Often, communication requirements of MPSoCs are known a priori. Based on these structured communication patterns, an application characterization graph can be con- structed to capture the point-to-point communication requirements of the IP blocks. To begin constructing the required topology, the number of components, their size and their required connectivity as dictated by the communication patterns must be determined.

An example of a customized topology for a video object plane decoder is shown in Figure 2.4. The MPSoC is composed of 12 heterogeneous IP blocks. In Figure 2.4, the design is mapped to a 3×4 mesh topology requiring 12 routers (R). When specific application characteristics are taken into account (e.g. not every block needs to communicate directly with every other block), a custom topology is created (Figure 2.4b). This irregular topology reduces the number of switches from 12 to 5; by reducing the number of switches and the links in the topology, significant power and area savings are achieved. Some blocks can be directly connected without the need for a switch, such as the VLD and run length decoder units. Finally, the degree of the switches has changed; the mesh in Figure 2.4a requires a switch with 5 input/output ports (although ports can be trimmed on edge nodes). The 5 input/output ports represent the four cardinal directions: north, south, east and west plus an Injection/Ejection port. All of these ports require both input and output connections leading to 5×5 crossbars. With a customized topology, not all blocks need both input and output ports; the largest switch in Figure 2.4b is a 3×3 switch. Not every connection between

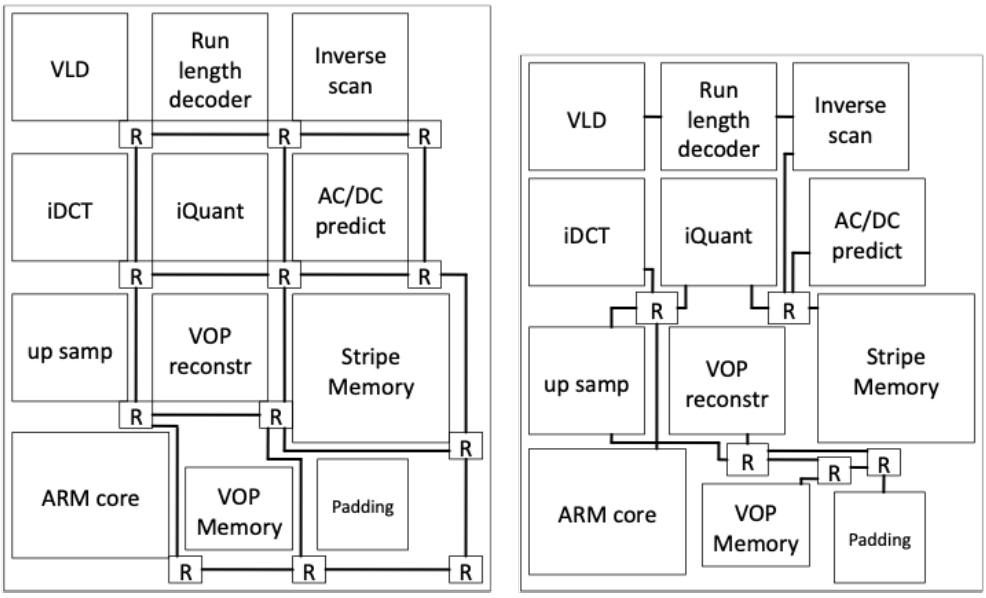


Figure 2.4: A regular (mesh) topology and a custom topology for a video object plane decoder (VOPD)[24]

links coming into and out of a router is necessary in the customized topology resulting in smaller switches; connectivity has been limited because full connectivity is not needed by this specific application.

2.3 Physical Channel Router and Virtual Channel Router

Channel here refers to a *conduit* or a *link* which connects two routers/nodes. Physical channel routers have dedicated physical links connecting individual queue of buffers present at the input ports of the neighboring routers. Here the crossbar, which connects input port and output port within the router, is also separate of each buffer queue for a given set of input ports and output ports. Physical channel router is bigger in area and power because of the wiring involved in laying out physical channel routers.

Virtual channel routers use a single link connecting two neighboring routers. These links are time multiplexed by the queues of buffers present at the input port of the router and input ports and output ports share the common crossbar to transport packets from the input

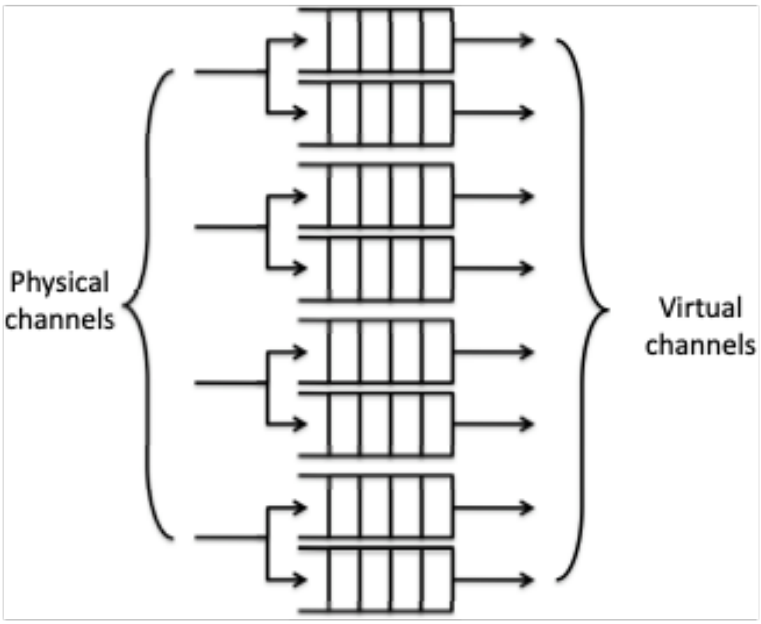


Figure 2.5: Showing difference between *Physical Channel* and *Virtual channel*[24]

queues, out of the router via output port. Here virtual channel refers to the individual queue of the buffers present at the input port of the router. Virtual channel routers are smaller in size and allow better usage of input port buffers by allowing packets present at one of queue (VC) of upstream router to occupy different queue at the next hop downstream router. This step is called *VC-allocation*. In the router microarchitecture designs proposed in this thesis, we have assumed baseline to be Virtual Channel routers. Although the techniques introduced here can be extended to Physical Channel routers.

2.4 Input buffered router and output buffered router

Input buffered router refers to the router micro architecture which buffers the packet at its input port. These routers may suffer the head-of-line blocking where the packet at the head of the queue (or VC) is not able to move in the network because of the congestion at its requested output port, blocking the movement of other enqueued packets. Output buffered router refers to the router micro architecture where the packets are buffered at the output port of the router. This router micro architecture is free from head of line blocking,

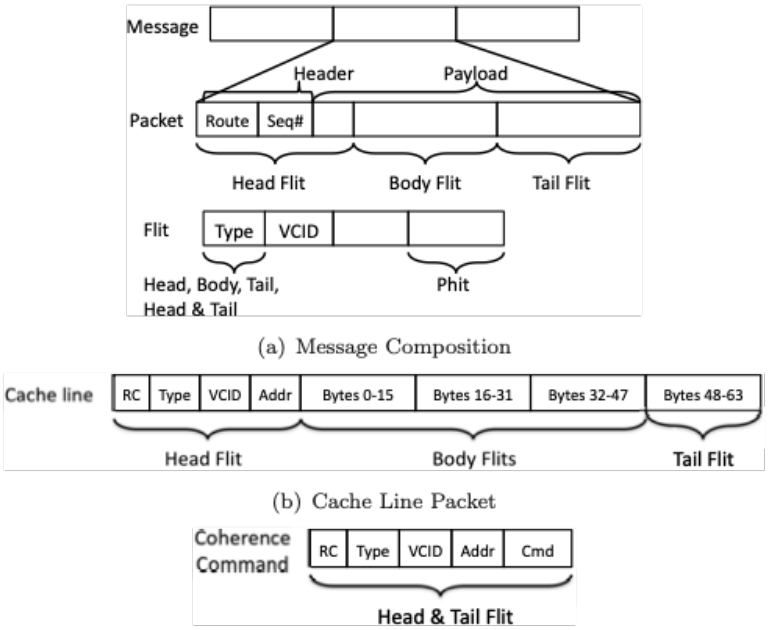


Figure 2.6: Composition of a message, packet, flit in an on-chip network[24]

however, the output put needs to perform enqueue of packets at higher frequency without over-writing incoming packets within the queue. This is one of the challenges in designing output buffered routers. Most on-chip router uses input port buffering.

2.5 Message, Packet, Flit and Phit

Different nodes connected in the network talk to each other by sending *messages*. When a message is injected into the network, it is first segmented into packets, which are then divided into fixed length Flits (Flow Control Unit). The packet will consist of a head flit that contains the destination address, body flits and a tail flit that indicates the end of a packet. Flits can be further broken down into phits, which are physical units and correspond to the physical channel width. The breakdown of messages to packets and packets to flits is depicted in Figure 2.6.

Link bandwidth decides the size of Phit. Usually in on-chip networks there is ample bandwidth available compared to off-chip network. Therefore, Flit and Phits are same in on-chip network, link bandwidth is then of the same size as that of Flit size. However, in

off-chip networks, channel widths are limited by pin bandwidth; this limitation causes flits to be broken down into smaller chunks called Phits

2.6 Routing Algorithm

After determining the network topology, the routing algorithm is used to decide what path a message will take through the network to reach its destination. The goal of the routing algorithm is to distribute traffic evenly among the paths supplied by the network topology, so as to avoid hotspots and minimize contention, thus improving network latency and throughput. All of these performance goals must be achieved while adhering to tight constraints on implementation complexity: routing circuitry can stretch critical path delay and add to a routers area footprint. While energy overhead of routing circuitry is typically low, the specific route chosen affects hop count directly, and thus substantially affects energy consumption.

2.6.1 Types of Routing Algorithm

In this section, we briefly discuss various classes of routing algorithms. Routing algorithms are generally divided into three classes: deterministic, oblivious and adaptive.

2.6.2 Deterministic Dimension Ordered Routing

While numerous routing algorithms have been proposed, the most commonly used routing algorithm in on-chip networks is dimension-ordered routing (DOR) due to its simplicity. Dimension-ordered routing is an example of a deterministic routing algorithm, in which all messages from node A to B will always traverse the same path. With DOR, a message traverses the network dimension-by-dimension, reaching the ordinate matching its destination before switching to the next dimension. In a 2-dimensional topology such as the mesh in Figure 2.7, X-Y dimension-ordered routing sends packets along the X-dimension first, followed by the Y-dimension. A packet travelling from (0,0) to (2,3) will first traverse 2 hops

along the X-dimension, arriving at (2,0), before traversing 3 hops along the Y-dimension to its destination.

2.6.3 Oblivious Routing

Another class of routing algorithm are oblivious ones, where messages traverse different paths from A to B, but the path is selected without regard to network congestion. For instance, a router could randomly choose among alternative paths prior to sending a message. Figure 2.8 shows an example where messages from (0,0) to (2,3) can be randomly sent along either the Y-X route or the X-Y route. Deterministic routing is a subset of oblivious routing.

2.6.4 Adaptive Routing

A more sophisticated routing algorithm can be adaptive, in which the path a message takes from A to B depends on network traffic situation. For instance, a message can be initially following the X-Y route and see congestion at (1,0)s east outgoing link. Due to this congestion, the message will instead choose to take the north outgoing link towards the destination (see Figure 2.7).

Routing algorithms can also be classified as minimal and non-minimal. Minimal routing algorithms select only paths that require the smallest number of hops between the source and the destination. Non-minimal routing algorithms allow paths to be selected that may in- crease the number of hops between the source and destination. In the absence of congestion, non-minimal routing increases latency and also power consumption as additional routers and links are traversed by a message. With congestion, the selection of a non-minimal route that avoids congested links, may result in lower latency for packets.

Before we get into details on specific deterministic, oblivious and adaptive routing algorithms, we will discuss the potential for deadlock that can occur with a routing algorithm.

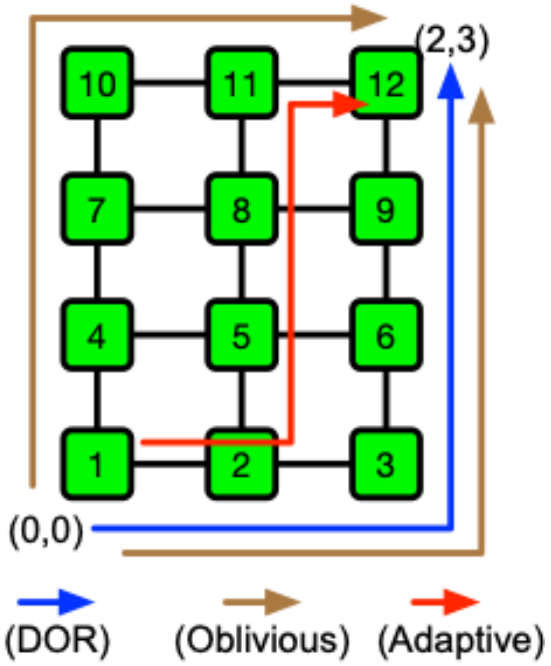


Figure 2.7: DOR illustrates an X-Y route from (0,0) to (2,3) in a mesh, while Oblivious shows two alternative routes (X-Y and Y-X) between the same source-destination pair that can be chosen obliviously prior to message transmission. Adaptive shows a possible adaptive route that branches away from the X-Y route if congestion is encountered at (1,0)

2.6.5 Routing on Irregular Topologies

The discussion of routing algorithms in this chapter has assumed a regular topology such as a torus or a mesh. In the previous chapter, the potential for power and performance benefits of using irregular topologies for MPSoCs composed of heterogeneous nodes was explored. Irregular topologies can require special considerations in the development of a routing algorithm. Common routing implementations for irregular networks rely on source table routing or node-table routing. Care must be taken when specifying routes so that deadlock is not induced. Turn model routing may not be feasible if certain connectivity is removed by the presence of oversized cores in a mesh network, for example.

Up*/Down*[25] routing is a popular deadlock-free routing algorithm for irregular topologies, that marks each link as either Up or Down, starting from a root node. All flits can only transition from a Down link to an Up link, but never the opposite, which guarantees

deadlock freedom. More details of Up*/Down* routing is present in next chapter subsection 3.1.3 where we talk about prior work done in the field of deadlock-freedom in networks.

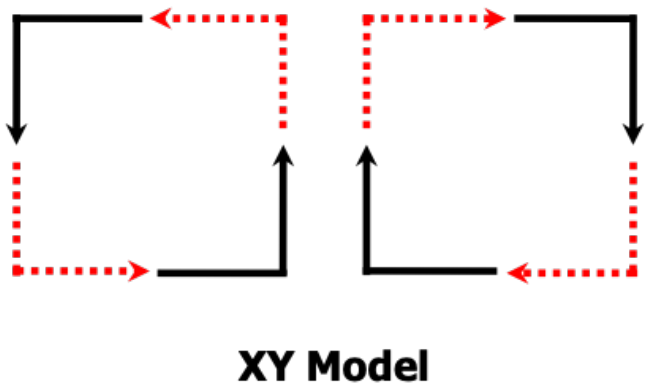


Figure 2.8: XY DoR routing

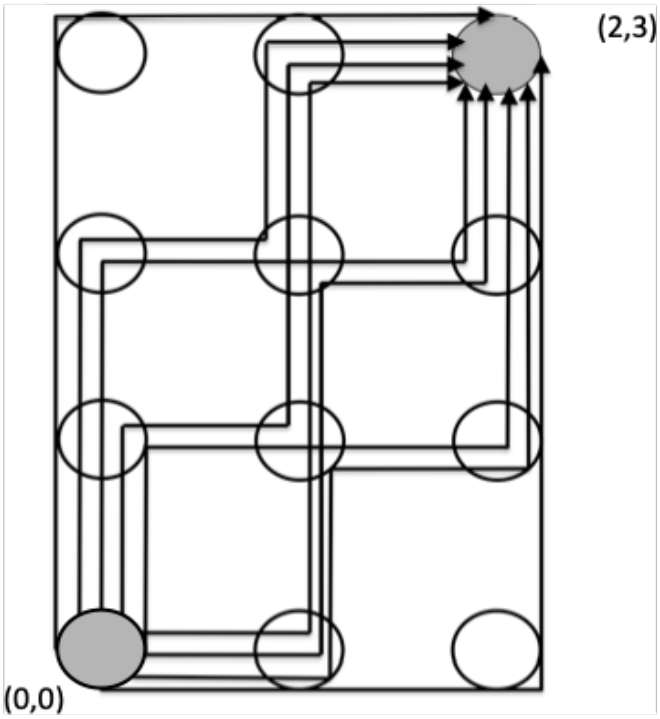


Figure 2.9: Adaptive routing example

Figure 2.9 shows all possible (minimal) routes that a message can take from Node (0,0) to Node (2,3). There are nine possible paths. An adaptive routing algorithm that leverages

only minimal paths could exploit a large degree of path diversity to provide load balancing and fault tolerance.

Adaptive routing can be restricted to taking minimal routes between the source and the destination. An alternative option is to employ misrouting, which allows a packet to be routed in a non-productive direction resulting in non-minimal paths. When misrouting is permitted, livelock becomes a concern. Without mechanisms to guarantee forward progress, livelock can occur as a packet is continuously misrouted so as to never reach its destination. We can combat this problem by allowing a maximum number of misroutes per packet and giving higher priority to packets than have been misrouted a large number of times. Misrouting increases the hop count but may reduce end-to-end packet latency by avoiding congestion (queueing delay).

Routing Algorithm	Source Routing	Combinational	Node Table
Deterministic			
DOR	Yes	Yes	Yes
Oblivious			
Valiant's	Yes	Yes	Yes
Minimal	Yes	Yes	Yes
Adaptive	No	Yes	Yes

Figure 2.10: Different implementation of routing algorithms[24]

2.7 Routing Algorithm: Implementation

In this section, we discuss various implementation options for routing algorithms. Routing algorithms can be implemented using look-up tables at either the source nodes or within each router. Combinational circuitry can be used as an alternative to table-based routing. Implementations have various trade-offs, and not all routing algorithms can be achieved with each implementation. Figure 2.10 shows examples for how routing algorithms in each of the three different classes can be implemented

2.7.1 Source Routing

Routing algorithms can be implemented in several ways. First, the route can be embedded in the packet header at the source, known as source routing. For instance, the X-Y route in Figure 2.7 can be encoded as $<$ E, E, N, N, N, Eject $>$, while the Y-X route can be encoded as $<$ N, N, N, E, E, Eject $>$. At each hop, the router will read the leftmost direction off the route header, send the packet towards the specified outgoing link, and strip off the portion of the header corresponding to the current hop.

2.7.2 Node Table Based Routing

More sophisticated algorithms are realized using routing tables at each hop which store the outgoing link a packet should take to reach a particular destination. By accessing routing information at each hop (rather than all at the source), adaptive algorithms can be implemented, and per-hop network congestion information can be leveraged in making decisions.

2.7.3 Combinational Circuit

Alternatively, the message can encode the ordinates of the destination and use comparators at each router to determine whether to accept (eject) or forward the message. Simple routing algorithms are typically implemented as combinational circuits within the router due to the low overhead.

With source routing, the packet must contain space to carry all the bits needed to specify the entire path. Routing using combinational circuits requires only that the packet carry the destination identifier. The circuits required to implement the routing algorithm can be quite simple and executed with very low latency.

By implementing the routing decision in combinational circuits, the algorithm is specific to one topology and one routing algorithm. The generality and configurability of table-based strategies are sacrificed. Despite the speed and simplicity of using a circuit to compute the next hop in the routing path, this computation adds latency to the packet

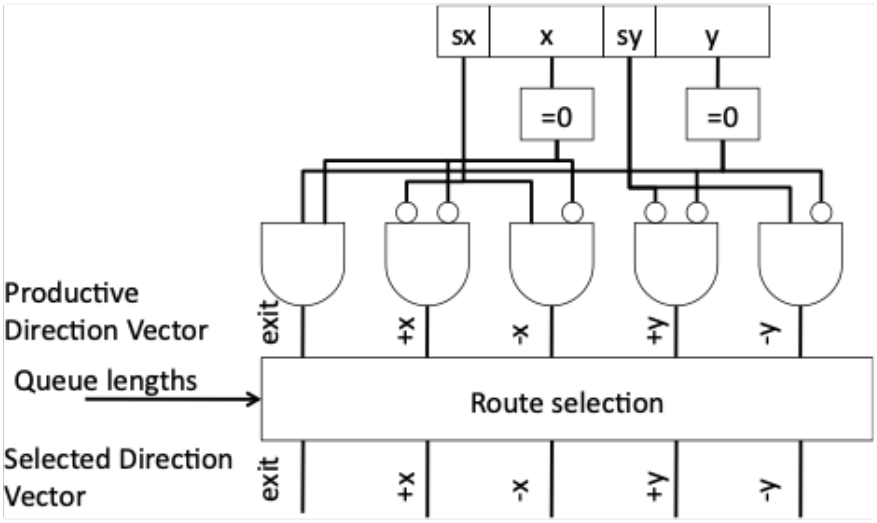


Figure 2.11: Implementation of XY routing using combination circuit[24]

traversal when compared to source-based routing. As with node-routing, the next output must be determined at each hop in the network.

2.7.4 Adaptive Routing

Adaptive routing algorithms need mechanisms to track network congestion levels and update the route. Route adjustments can be implemented by modifying the header, by employing combinational circuitry that accepts as input these congestion signals, or by updating entries in a routing table. Many congestion sensitive mechanisms have been proposed, with the simplest being tapping into information that is already captured and used by the flow control protocol, such as buffer occupancy or credits.

The primary benefit of increasing the information available to the routing circuitry is adaptivity. By improving the routing decision based on network conditions, the network can achieve higher bandwidth and reduce the congestion latency experienced by packets.

The disadvantage of such an approach is complexity. Additional circuitry is required for congestion-based routing decisions; this circuitry can increase the latency of a routing decision and the area of the router. Although the leveraging of information already available at the router is often done to make routing decisions, increasing the sophistication of the

routing decision may require that additional information be communicated from adjacent routers. This additional communication could increase the network area and energy.

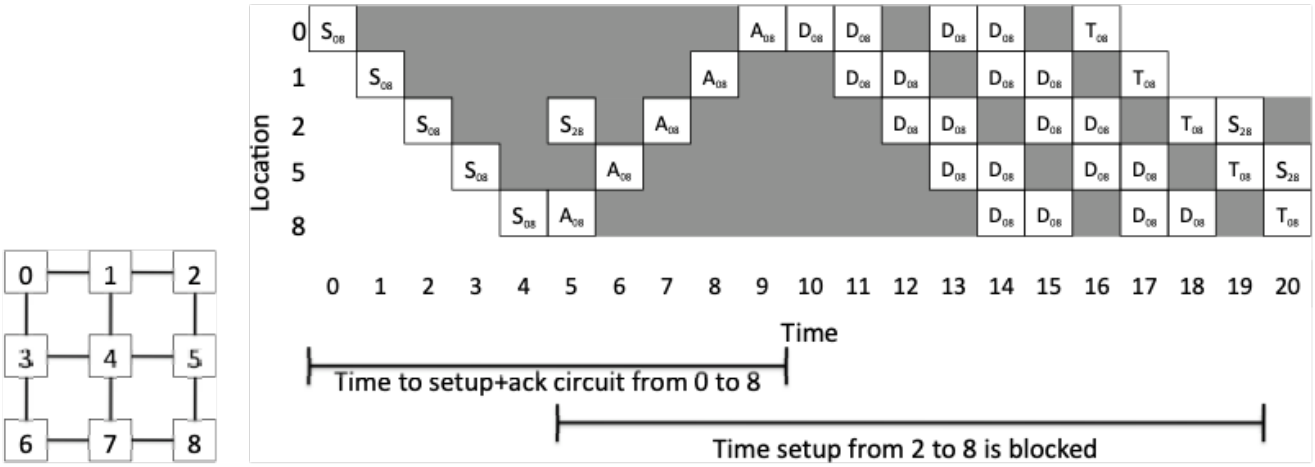
2.8 Flow Control

Flow control governs the allocation of network buffers and links. It determines when buffers and links are assigned to messages, the granularity at which they are allocated, and how these resources are shared among the many messages using the network. It is equivalent to the operation of traffic signals at a junction. The role of an efficient flow control mechanism is to minimize the latencies at low-loads and maximize the throughput at high-loads. Both these goals can be achieved by ensuring that network resources (buffers and links) are not idle when there are flits waiting to use them. While the topology and routing algorithm fix the theoretical latency and throughput characteristics for a particular traffic pattern, it is the flow control that determines how close to this theoretical capacity can the network operate.

Flow control techniques are classified by the granularity at which resource allocation occurs. We will discuss techniques that operate on message, packet and flit granularities next.

2.8.1 Message-based flow control

Circuit switching is the technique that operates at message granularity, while communicating between the pair of nodes. A dedicated path is reserved between the communicating nodes before sending the message. Once path is reserved the packets are sent in the buffer less manner over the reserved path. The overhead of setting-up the path is ameliorated with larger message size (comprising of many packets and flits). However, if messages constitute fewer packets and flits then *circuit switching* is not preferred.



(a) Network (b) Time-Space Diagram

Figure 2.12: Circuit-switching example from Core 0 to Core 8, with Core 2 being stalled. S: Setup flit, A: Acknowledgement flit, D: Data message, T: Tail (deallocation) flit. Each D represents a message; multiple messages can be sent on a single circuit before it is deallocated. In cycles 12 and 16, the source node has no data to send.[24]

2.8.2 Packet-based flow control

Circuit switch flow control allocate resources at message granularity. It is inefficient when the message size is small. Moreover, the path reserved by circuit switch flow control is not available to be used by any other flow until its teared down. The next set of flow control scheme reserves buffer at packet granularity. These are:

Store and Forward

In store-and-forward flow control, each router waits until an entire packet has been received, before forwarding it to the next router. The buffer size present at the input port of each router must be greater than the maximum packet size in the network. Buffer space and link bandwidth are thus allocated at a packet granularity.

Virtual cut-through

Virtual cut-through improves the per-hop delay experienced by the packet in store-and-forward flow control by starting the transmission of packet as soon as first flit arrives.

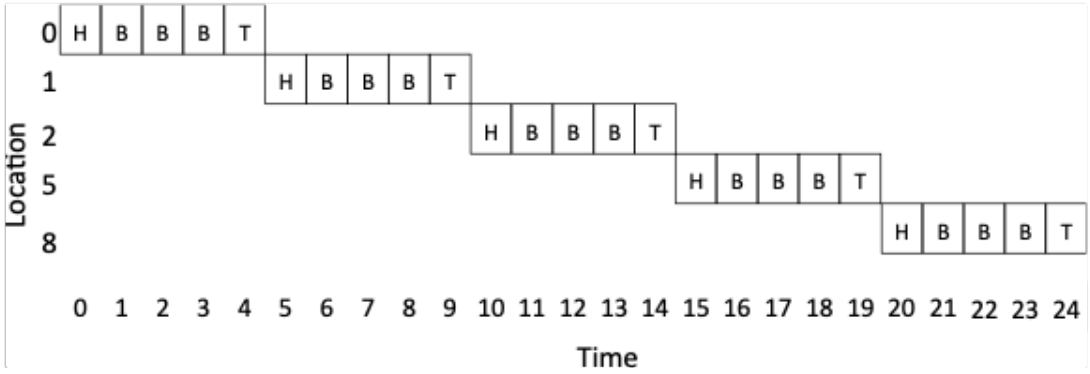
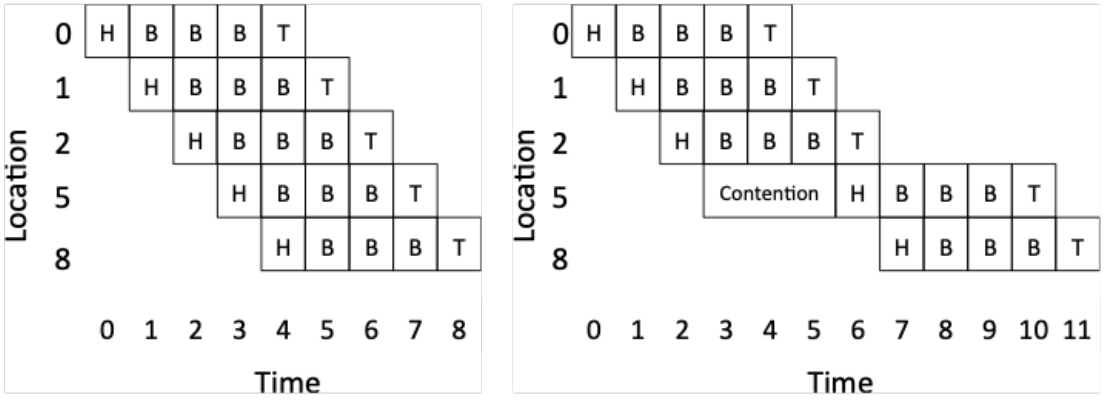


Figure 2.13: Progress of packet in the network with time in store and forward flow control[24]

Virtual cut through does not wait for the whole packet to arrive to begin the transmission, it *cut-through* the packet transmission to next router as soon flits of packet arrive.



(a) VCT with no delay (b) VCT with delay

Figure 2.14: Progress of packet in the network with time in virtual cut through flow control.[24]

In this thesis we have extensively assumed virtual cut through as the baseline flow control, in the evaluation of techniques.

2.8.3 Flit-based flow control

Packet based flow control mechanism requires the buffering at each input port to be equal to or greater than the size of the packet. To reduce this buffering requirement flit-based flow control mechanisms are introduced.

Wormhole

Wormhole flow control is similar to virtual cut through, in that it allows the incoming flit arrived to cut through to the next router if buffer space available. However, unlike Virtual Cut through, in wormhole flow control buffers and link bandwidth are allocated at flit level rather than packet level. This allows relatively small flit buffer to be used at the input port of each router. A key issue with wormhole flow control is that packets go out serially from a routers input port, in the order in which they came in, since there is only one FIFO queue for the physical input channel. If the flit of a packet at the head of the queue gets blocked due to insufficient buffer space due to congestion at its next router, the packet behind it also gets blocked even though it might want to use a separate non-congested route. This is known as head-of-line blocking.

Virtual channel

Virtual Channel (VC) flow control removes the head-of-line blocking problem by associating separate queues for different flows at a router, rather than queuing them one behind the other like wormhole routers, even though there is only one physical input/output channel.

A head flit allocates a VC and arbitrates for the output physical channel bandwidth before it can proceed to the next router. The body and tail flits use the same VC, but still need to compete for the channel bandwidth with flits in other VCs. A VC is freed once the tail flit leaves. When a packet in some VC gets blocked, packets behind it can still traverse the physical channel using other virtual channels, thus solving the head-of-line blocking problem and enhancing throughput.

Virtual Channel, as we will see in next chapter, can serve dual purpose they provide higher performance and can provide deadlock freedom. However, virtual channel come at the cost of higher area/power overhead of the router.

2.9 Buffer Management

Network-On-Chip does not allow packets to over-write each other, hence a packet from one router can only move to the next router when there is guaranteed buffer present for it at the downstream router. There are two common ways of communicating buffer availability - on-off and credits.

2.9.1 ON-OFF Signaling

In on-off signaling, the downstream router sets a bit high (low) if the number of free buffers is above (below) some threshold value. The upstream router sends a flit only if the on-off bit is high. The threshold value is set by the buffer turnaround time. This is the round-trip delay (in cycles) for the on-off signal to go to the upstream router, be processed and be visible to the arbitrating flits. The threshold value guarantees that all flits received in between the time that the bit is turned low and the upstream router stops sending flits have a free buffer available. For VC flow control, an on-off bit is required for every VC. On-off signaling can end up lowering throughput compared to credit-based signaling since the on-off signal could be low and yet there could be idle buffers at the next router.

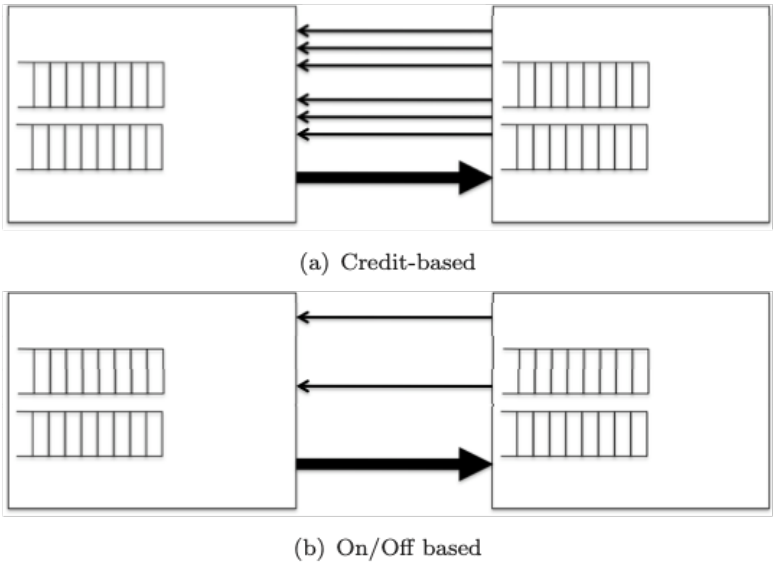


Figure 2.15: ON-OFF vs Credit based signaling[24]

2.9.2 Credit-based signaling

In credit signaling, each upstream router maintains a count of the number of free buffers at its adjacent downstream router. It decrements the count each time a flit is sent out. When a flit leaves the downstream router, it sends a credit bit back to the upstream router which increments its credit count. For VC flow control, the credit count is maintained on a per-downstream-VC basis, and the credit signal carries the credit bit, the VCid, and an additional bit to indicate if the VC is now free or not.

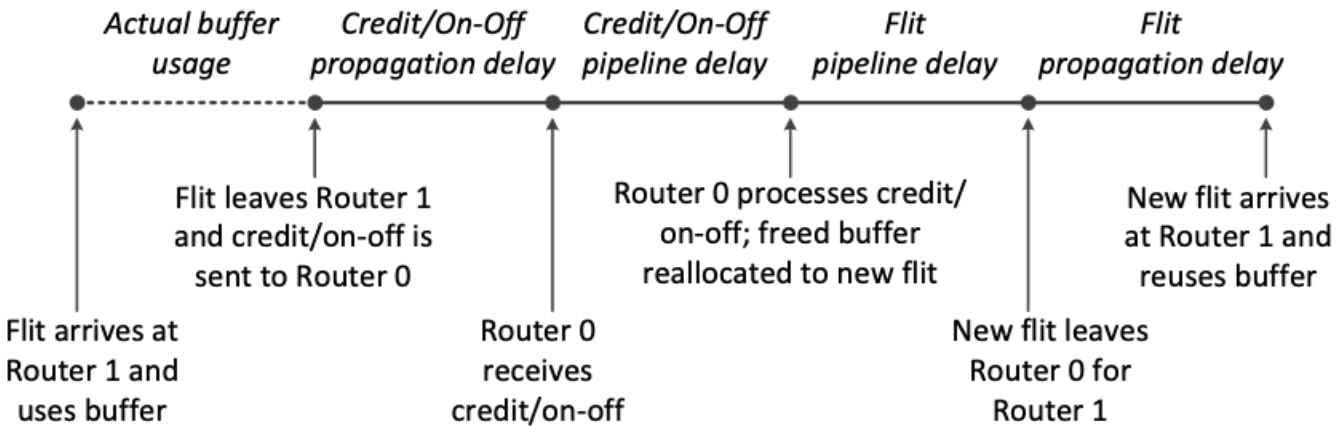


Figure 2.16: Buffer turnaround time[24]

The buffer turnaround time determines the minimum number of VCs and/or buffers-per-VC to avoid self-throttling of the system. It depends on the wire propagation delay and the router pipeline depth, as shown in Figure 2.16. The longer the delay, the longer is the idle time for a free buffer, the lower is the buffer utilization and poorer is the throughput.

2.10 Virtual Channel Router Microarchitecture

Router microarchitecture. A generic router microarchitecture is comprised of the following components: input buffers, router state, routing logic, allocators, and a crossbar (or switch). Router functionality is often pipelined to improve throughput. Delay through each router in the on-chip network is the primary contributor to communication latency. As a result,

significant research effort has been spent reducing router pipeline stages and improving throughput.

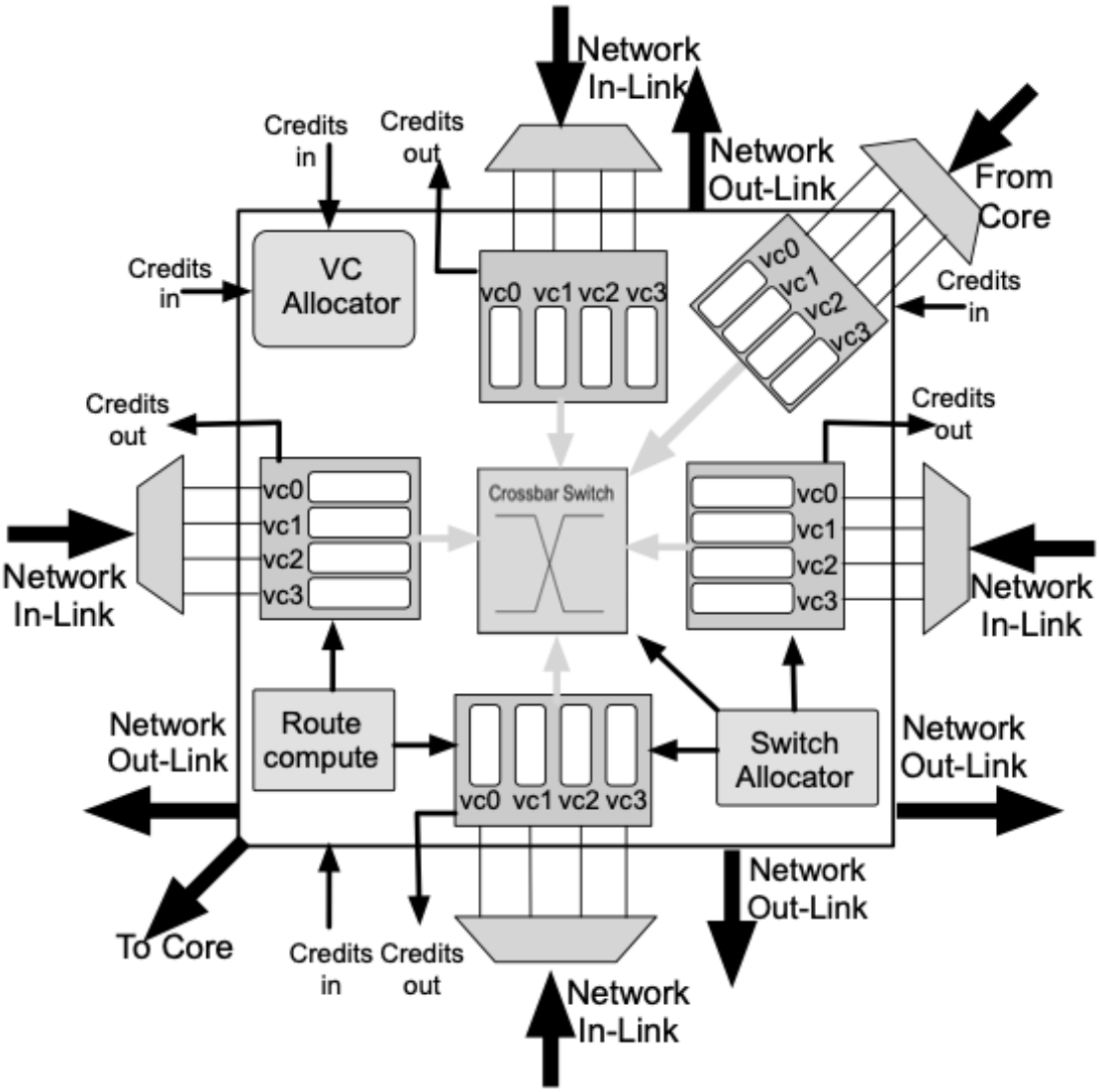


Figure 2.17: Microarchitecture of a 5-port Mesh Router

Figure 2.17 shows the microarchitecture of a state-of-the-art NoC router. We show a 5-ported router (for a mesh). Each input port has buffers that are organized into separate VCs. Buffers are FIFO queues that can be implemented using Flip Flops or register files or SRAM. Each input port connects to a crossbar switch which provides cycle by cycle non-blocking connectivity from any input port to any output port. A crossbar is fundamentally a mux at every output port. Mux-based crossbars are actually implemented by synthesizing

muxes at every output port, while matrix crossbars layout the cross- bar as a grid with switching elements at cross-points. Each input port also houses a route compute unit, an arbiter for the crossbars input port, and a table tracking the state of each VC. Each output port has an arbiter for the crossbars output port, and also tracks the free VCs and credits at the neighboring routers input port. A n : 1 arbiter allows up to n requests for a resource, and grants it to one of them. Matrix arbiters [26] maintain fairness across cycles and are used in this thesis. Each flit that goes through a router needs to perform the following actions on its *control-path*:

- **Router Compute (RC)** All head and head tail flits need to compute their output ports, before they can arbitrate for the crossbar. RC can be performed either by a table lookup, or simply by combinational logic. The former is used for complex routing algorithms, while the latter is used for simpler routing schemes like XY which we assume in most of this thesis. To remove RC from the critical path, we use lookahead routing[27] where each flit computes the output port at the next router, instead of the current one so that its output port request is ready as soon as it arrives.

- **Switch Allocation (SA)** All flits arbitrate for access to the crossbars input and output ports. For a n$\times$n router with v VCs per input port, Switch Allocation is fundamentally a matching problem between n resources (output links) and n$\times$v contenders (total VCs in the router). To simplify the allocator design in order for it to be realizable at a reasonable clock frequency, we often use a separable allocator [28].The idea is to first arbitrate among the input VCs at each input port using a v : 1 arbiter at every input port, and then arbitrate 4 among the input ports using a n : 1 arbiter at every output port.

- **VC Allocation (VA)** All flits need a guaranteed VC at the next router before proceeding. VC Allocation is only performed by head tail and head flits, while body and tail flits use the same VC as their head. VC Allocation can also be performed in a

separable manner [28] like SA. In this thesis, we use a simpler VA scheme proposed by Kumar et al. [29] which we refer to as VC Select (VS). Each output port maintains a queue of VC ids corresponding to the free VCs at the neighbors input port. The SA winner for that output port gets assigned the VCid at the head of the queue, and the VCid is dequeued. When a VC becomes free at the next router and it sends back a credit, the VCid is enqueued into the queue. If the free VC queue is empty, then flits are not allowed to perform SA.

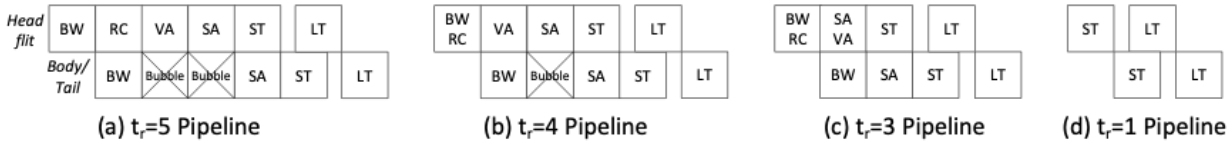


Figure 2.18: Evolution of Router pipeline[24]

Once a flit completes RC, SA and VA, it can proceed to its data-path:

- **Switch Traversal (ST)** Winners of SA traverse the crossbar in this stage. The select lines of the crossbar are set by the grant signals of SA.
- **Link Traversal (LT)** Flits coming out of the crossbar traverse the link to the next router.
- **Buffer Write (BW)** Incoming flits are buffered in their VC. While the flit remains buffered, its control-path (RC, SA and VA) is active
- **Buffer Read (BR)** Winners of SA are read out of their buffers and sent to the crossbar.

2.11 Router Pipeline

Early on-chip router prototypes were modeled similar to off-chip routers. Their pipeline is shown in Figure 2.18(a). This design has a 5-stage router, i.e., tr = 5. Lookahead routing,

which computes the route one hop in advance, shortens the router pipeline by one stage, as shown in Figure 2.18(b), allowing VA and SA to commence as soon as the route is read out in the first stage. Speculative VC allocation or VC Select allow VA to occur in parallel to SA, reducing the pipeline even further to 3-cycles, as shown in Figure 2.18(c). To this 3-stage baseline router, which is similar to Intels recent 48-core SCC router, recent research has proposed speculative pre-arbitration of the crossbar to reduce the pipeline to 1-cycle within the router, as shown in Figure 2.18(d). If the pre-arbitration (i.e., VA and SA) succeeds, the crossbar is setup for the incoming flit to directly traverse it, bypassing the conventional BW stage. If the pre-arbitration fails, the incoming flit is buffered as before and continues to arbitrate for the switch and VC.

This design was fabricated as part of this thesis work. It will be referred to as BASELINE (tr=1) throughout the thesis.

2.12 NoC Traffic

In shared memory systems, the on-chip network interconnects the memory subsystem (L1, L2, directory, memory controller etc). The traffic through the network is thus cache coherence traffic. In addition, we stress test our network with myriad synthetic traffic patterns to characterize the latency/throughput characteristics. Both these kinds of traffic domains are described in this section.

2.12.1 Cache Coherence traffic

The role of the cache coherence protocol is to maintain the semantics of one writer or many readers in parallel programs. While processors perform reads and writes with various sizes, ranging from 1 to 64 bytes, in practice, coherence is commonly maintained at the granularity of cache blocks

2.12.2 Coherence Protocols

There are two families of cache coherence protocols: snooping and directory-based protocol. Snooping and directory protocols Two main classes of coherence protocols are commonly in use

- **Snooping protocols**: In these protocols, cache controllers initiate a block request by broadcasting it to all other coherence controllers which will process it and reply, if needed, with data for instance. They rely on the interconnection network to deliver the messages in a consistent order, and most of them typically assume a total order obtained via a shared bus. Relaxed buses however exist.

- **Directory protocols**: In these protocols, cache controllers initiate a block request by unicasting it to the block home memory controller which looks into its directory which caches are the current owner or sharers for that block. If the LLC/memory is the owner, it terminates the transaction by sending data to the requester. Otherwise, it forwards the request to the owner cache, which completes the transaction.

Snooping protocols are simple, but they do not scale to large numbers of processors as broadcasting does not scale. Directory protocols are scalable because they unicast, but many transactions take more time because they require an extra message to be sent when the home controller is not the owner. In addition, the choice of protocol affects the interconnection network as, for instance, classical snooping protocols require total order

In 1986, Sweazey and Smith[30] introduced a five-state MOESI model on which many coherence protocols nowadays still rely on. These states are formed from combinations of the previously defined characteristics. The three first main states are:

- **M(odified)**: The block is valid, exclusive, owned, may be dirty, and may be written or read. The cache has the only valid copy of the block and it is potentially stale at the memory. The cache is responsible for requests for the block.

- **S(hared)**: The block is valid but not exclusive, not dirty, not owned and is read-only. The other caches may hold valid, read-only copies of the block.

- **I(nvalid)**: The block is invalid. Either the cache does not hold the block, or it holds a stale copy that it may not read or write.

In addition to this three first states, the MOESI set specifies an O state and a E state, which are used to optimize certain situations:

- **O(wned)**: The block is valid, owned, and potentially dirty, but not exclusive, and readonly. It is potentially stale at the memory. The cache is responsible for requests for the block. The other caches may have a read-only copy but are not owners.

- **E(xclusive)**: The block is valid, owned, exclusive, clean, and read-only. It is up-to-date at the memory. No other cache has a valid copy of the block.

A Venn diagram of the MOESI states is illustrated in Figure 2.19. All states except I are valid states. M, O and E are owned states. M and E are exclusive states. M and O are potentially dirty states. This diagram also shows that the example IV protocol condensed the MOES states into the V state. The MOESI states are quite common and may be called differently. However, they are not an exhaustive set. For instance, Intel is known to use a MESIF set, in which the F(orward) state is similar to the O state except that it is clean. Therefore, the memory has an up-to-date copy of the block.

In this thesis, we used both snoopy and directory-based cache coherence with gem5 full system simulation for evaluation purpose. Each cache coherence protocol implements its own set of transactions, which are realized by messages from different message class. Each message class is allocated its own set of virtual channels at every input port of the router, called as virtual network. In any cache coherence transaction, there are two types of messages, *non-terminating* and *terminating*. Non-terminating messages are those which are needed to realize a cache coherence transaction but does not end the transaction for ex-

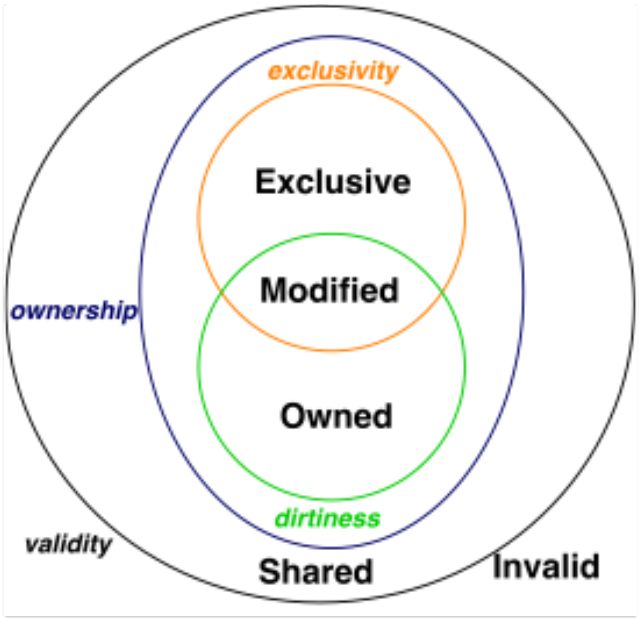


Figure 2.19: Venn diagram of the M-O-E-S-I states [31]

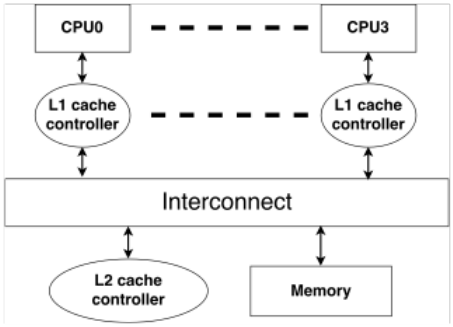
ample request-messages. Terminating messages are those which ends the coherence transaction for example response messages.

2.12.3 Protocols Considered in this Thesis

Two main protocols used in this thesis are:

- Two-Level MESI (Figure 2.20)
- AMD MOESI Hammer (Figure 2.21)

Two-Level MESI requires three message classes to implement its cache coherence transactions, these are request, response and forward. Therefore, there are three virtual networks for Two-Level MESI cache coherence protocol. AMD MOESI Hammer requires six message class to implement its cache coherence transaction, therefore it has six virtual networks. Next we present the memory hierarchy configuration for each cache coherence protocol used.

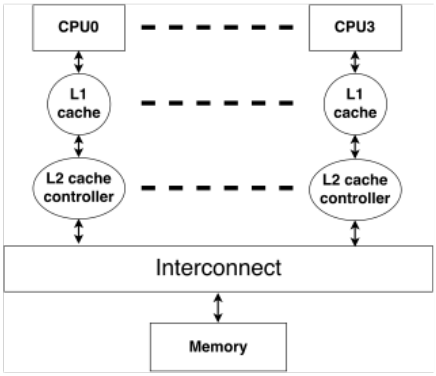


(a) Memory hierarchy

Parameter	Value
Number of directories	1
L1 instruction cache size	64kB
L1 instruction cache associativity	2
L1 data cache size	64kB
L1 data cache associativity	2
L2 data cache size	2MB
L2 data cache associativity	8
Cache line size	64B

(b) Default parameters

Figure 2.20: Configuration of memory system used for Two Level MESI protocol for full system simulations on gem5 [31]



(a) Memory hierarchy

Parameter	Value
Number of directories	1
L1 instruction cache size	64kB
L1 instruction cache associativity	2
L1 data cache size	64kB
L1 data cache associativity	2
L2 data cache size	2MB
L2 data cache associativity	8
Cache line size	64B

(b) Default parameters

Figure 2.21: Configuration of memory system used for MOESI hammer protocol for full system simulations on gem5 [31]

We assume a CMP design, with a private L1 per tile, and a shared L2 distributed across all tiles. Each L2 acts as a home node for part of the address space and holds the directory state for each cache line.

2.12.4 Virtual Network (VNets)

The series of messages sent by a coherence protocol as part of a coherence transaction fall within different message classes. For instance, all directory protocols (full-state, partial-state and no-state) used in this thesis use 4 message classes: request, forward, response and unblock. Token Coherence, a snoopy protocol, uses 3 message classes: request, response,

persistent request.

A potential deadlock can occur in the protocol if a request for a line from a L2 is unable to enter the network because the L2 it is waiting for a response for a previous request, while the response is unable to reach the L2 since all queues in the network are full of such waiting requests. To avoid such deadlocks, protocols require messages from different message classes to use different set of queues within the network. This is implemented by using virtual networks (vnets) within the physical network. Virtual Networks are identical to VCs in terms of their implementation: all vnets have separate buffers but multiplex over the same physical links. In fact, many works on coherence protocols use the term virtual channels to refer to virtual networks. However, in this thesis we will strictly adhere to using the term virtual networks or vnets to refer to protocol level message classes. The number of vnets is thus fixed by the protocol. Each vnet, on the other hand, can have one or more VCs within each router, to avoid head-of-line blocking or avoid routing deadlocks. In all NoC designs considered in this thesis, we will use the same number of VCs within each vnet.

2.12.5 Point-to-Point Ordering

Certain message classes (and thus their vnets) require point-to-point ordering in the network for functional correctness. This means that two messages injected from the same source, for the same destination, should be delivered in the order of their issue. We implement point-to-point ordering for flits within ordered vnets by (i) using deterministic routing, and (ii) using FIFO/queuing arbiters for SA-i at each router. The first condition guarantees that two messages from the same source do not use alternate paths to the same destination as that could result in the older message getting delivered after the newer one if the formers path has more congestion. The second condition guarantees that flits at a routers input port leave in the order in which they came in.

In the protocols used in this thesis none of the vnets in protocols have this requirement

as there are extra states and logic in the protocols to handle out-of-order messages.

2.13 Synthetic traffic

In all our experiments, we assume all sources inject with a uniform random injection rate (without bursts), while the destination coordinates depend on the traffic pattern. Table-2.22 lists some common synthetic traffic patterns used for studying a mesh network, along with their average hop-counts and theoretical throughput with XY routing. The theoretical throughput or capacity is the injection rate at which some link(s) in the mesh is (are) sending 1-flit every cycle. This is the best a topology can do, with perfect routing, flow control and microarchitecture.

Source (binary coordinates): $(y_{k-1}, y_{k-2}, \ldots, y_1, y_0, x_{k-1}, x_{k-2}, \ldots, x_1, x_0)$

Traffic Pattern	**Destination** (binary coordinates)	**Avg Hops** (for $k = 8$)	**Throughput** (for $k = 8$) (flits/nodes/cycle)
Bit-Complement	$(\bar{y}_{k-1}, \bar{y}_{k-2}, \ldots, \bar{y}_1, \bar{y}_0, \bar{x}_{k-1}, \bar{x}_{k-2}, \ldots, \bar{x}_1, \bar{x}_0)$	8	0.25
Bit-Reverse	$(x_0, x_1, \ldots, x_{k-2}, x_{k-1}, y_0, y_1, \ldots, y_{k-2}, y_{k-1})$	5.25	0.14
Shuffle	$(y_{k-2}, y_{k-3}, \ldots, y_0, x_{k-1}, x_{k-2}, x_{k-3}, \ldots, x_0, y_{k-1})$	4	0.25
Tornado	$(y_{k-1}, y_{k-2}, \ldots, y_1, y_0, x_{k-1+\lceil \frac{k}{2} \rceil - 1}, \ldots, x_{\lceil \frac{k}{2} \rceil - 1})$	3.75	0.33
Transpose	$(x_{k-1}, x_{k-2}, \ldots, x_1, x_0, y_{k-1}, y_{k-2}, \ldots, y_1, y_0)$	5.25	0.14
Uniform Random	$random()$	5.25	0.5

Figure 2.22: Synthetic traffic pattern[24]

2.14 Message sizes

We size our network parameters such that control messages (requests/forwards/unblocks) fit within one single flit, while data responses span multiple flits. For 128-bit flits which we assume in most of this thesis, unless specified, 64B cache line responses fit in 5 flits.

Thus VCs within the request, forward or unblock vnets are 1-flit deep, while VCs within the response vnet are often more than 1-flit deep

2.15 Types of Deadlocks

Deadlock is a condition which renders the forward movement of packet impossible in the network, resulting in complete system failure. This thesis addresses two types of deadlocks that plagues the on-chip network:

- Routing Level Deadlock (Figure 2.23)
- Protocol Level Deadlock (Figure 2.24)

2.15.1 Routing Level Deadlock

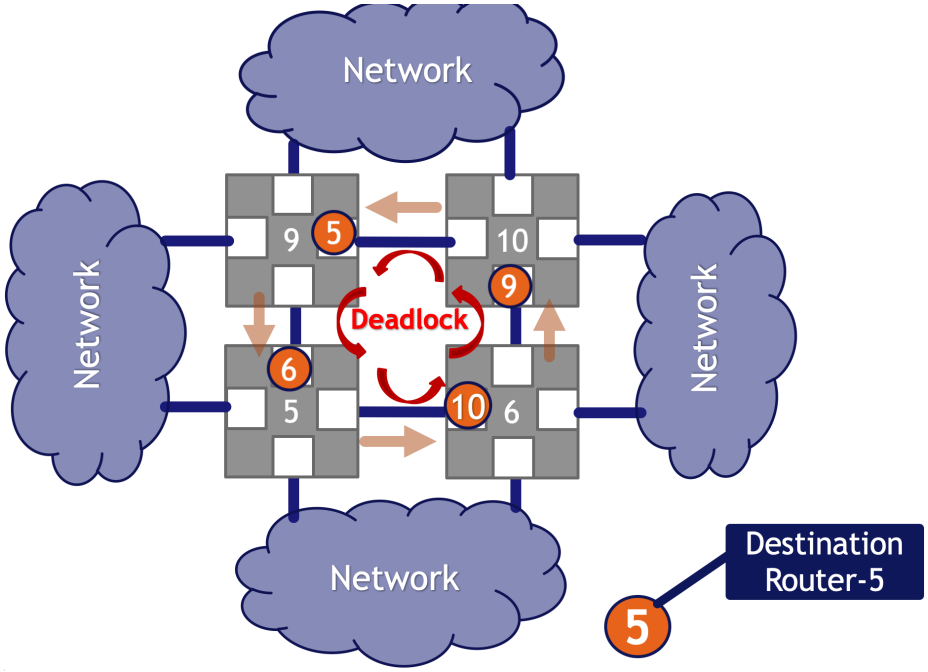


Figure 2.23: Routing-level deadlock.

Routing-level deadlocks occur within an interconnection network, when packet flows form a cyclic dependence. In such a system, the agents are net- work packets (or flits), while the resources are buffers for channels (i.e. physical links). Figure 2.23 shows an

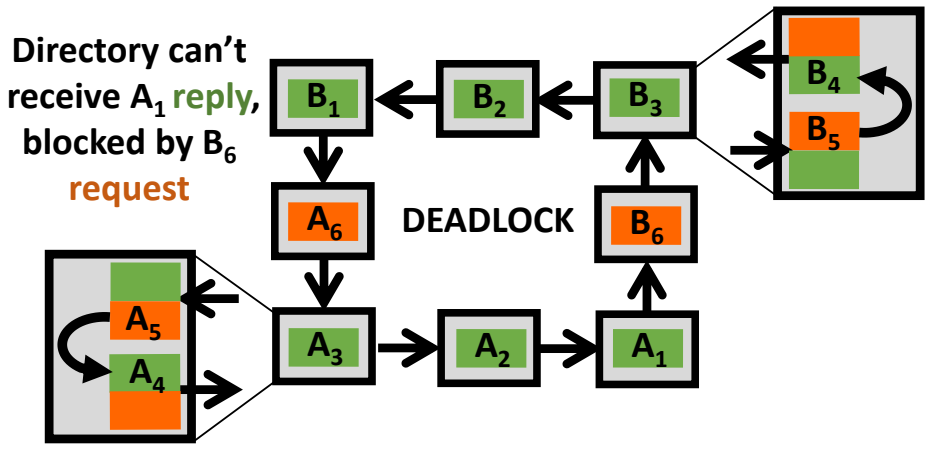


Figure 2.24: Protocol-level deadlock.

example of a deadlock in an interconnection network. All four conditions hold: buffers are mutually exclusive among the packets as a buffer can only hold one packet at a time; each packet holds on to a buffer while waiting for a downstream buffer; there is a cyclic dependence between buffers; and a packet holding an upstream buffer can only give up its buffer once it receives a credit from the downstream router (i.e., the downstream buffer become free).

Reconfigurability Deadlocks

Reconfigurability deadlocks are a kind of routing level deadlocks, *reconfigurability-deadlocks* arises because of dynamically changing routing algorithm to achieve higher performance. Although two individual routing algorithms can themselves be deadlock free, dynamic switching of routing algorithm can result in a deadlock. This refers to as reconfigurability deadlock. In reconfigurability deadlock same circular buffer dependency of packets occur as shown in the figure.

2.15.2 Protocol Deadlocks

Interconnected multiprocessor systems must use some form of communication protocol (e.g., message passing, cache coherence) to coordinate action among processors and stor-

age nodes. Generally, these protocols consist of transactions that must appear to occur atomically. When implemented, these transactions are broken into multiple messages that are non-atomic and may interleave with those of other transactions. Since they all interact on top of the same physical substrate, deadlocks may occur among them. The agents are messages, and the resources are buffers. Depending on the system, messages can consist of one or more physical packets and are often broken down into different classes. For example, cache coherence protocols in shared memory multiprocessors have a mix of message classes (e.g., requests, forward, responses, unblocks) based on the communicating entities (e.g., cache, directory, memory controller) and the state of the requested data (e.g, modified, shared, invalid).

Protocol deadlock (Figure 2.24) arise due to the amount of physical resources (i.e., buffers) being finite. Since all transactions constitute a back-and-forth of messages, there is an implicit dependence between head of a nodes inbound queue and the tail of its outbound to continue the transaction. If the outbound buffer is full, the node must wait, stalling other incoming messages and potentially creating a cyclic dependence between message, resulting in protocol level deadlock.

2.16 Performance Metrics

We characterize the performance of NoC designs on three metrics.

- **Network Latency** The target metric that this thesis is aimed at is network latency. the network latency has a fixed component (router + link delay), a variable component (contention delay) and a serialization component. The thesis presents microarchitectural optimizations to reduce the fixed component from 1-cycle at every hop to 1-cycle through- out the network.

- **Network Throughput** We define the saturation throughput of the network as the injection rate at which the network latency becomes $3\times$ the low- load latency (Fig-

ure 2.25). Throughput is a function of link utilization. Inefficient arbitration, buffer management and routing can lead to links going idle while there is waiting traffic and/or some links getting over-provisioned, leading to throughput loss. While the primary goal of this thesis is latency, most flow control techniques presented also try to push the saturation throughput closer to the theoretical capacity for synthetic traffic. For full-system traffic across all protocols and designs, we observe pretty low injection rates so the NoC is not throughput constrained as much as it is latency constrained. This is in part because our cores are in-order and non-speculative, and in part because most applications in SPLASH-2[32] and PARSEC[33] have well behaved working sets that do not stress the cache subsystem a lot.

- **Full-system Runtime** The full-system runtime is the runtime of the parallel section of our benchmarks, and our most important performance metric. A faster network by itself may not provide any returns if the messages whose delivery was speeded up are not on the critical path of the computation. In fact, in some cases we observe faster NoCs result in higher cache miss rates since remote lines get invalidated faster before they can be used by their local cores. But by the same argument, in some cases a minor speedup in the network can provide enormous speedups at the full-system level if certain threads were able to get access to locks faster, or certain requests hit in remote caches before that line was evicted off-chip, and so on. The thesis thus uses full-system simulations instead of trace-driven ones

2.17 Chapter Summary

In this chapter we covered a large breadth of topics related to designing and evaluating Network-on-Chip (NoC). Terms and background introduced here will be useful throughout this thesis to understand and appreciate different deadlock freedom schemes proposed thus far in the literature. This chapter lays the foundation of whole of the thesis and based on

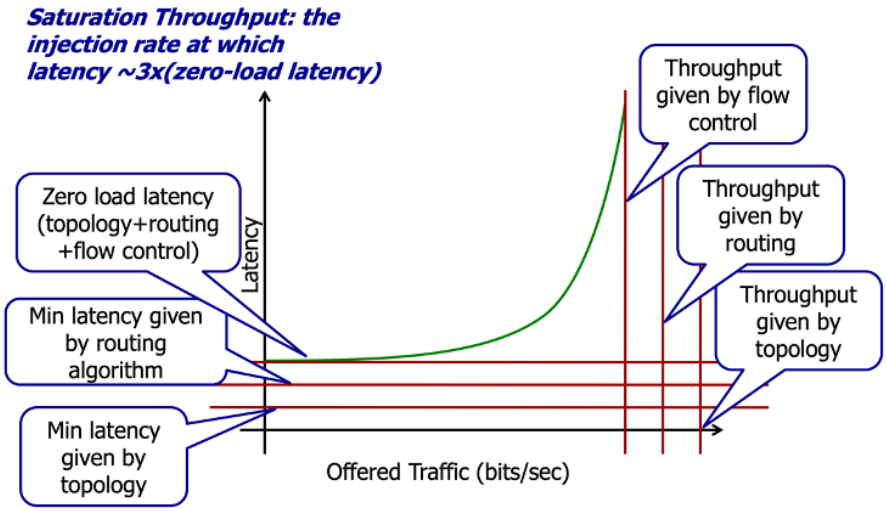


Figure 2.25: Typical Latency injection rate curve of the network. Different traffic patterns/applications will have different saturation throughput based on the *topology*, *routing algorithm*, and *flow control*, but they all will observe the same curve pattern.

our knowledge thus far we next we discuss the prior work done in networks to provide deadlock freedom.

CHAPTER 3
PRIOR WORK IN DEADLOCK FREEDOM

Deadlock in network is an age-old problem. This chapter briefly presents some of the prominent works done in this field, their insights and benefits. This chapter then classify the earlier work under a new taxonomy of *Proactive* and *Reactive*. Later, this chapter introduces a new category of deadlock freedom technique, called as *subactive*.

We classify techniques providing routing level deadlock freedom and protocol level deadlock freedom separately as follows:

3.1 Routing Deadlock Freedom Techniques

3.1.1 Routing Restrictions/Turn Restrictions

The most common technique to avoid deadlocks is to make the Channel Dependency Graph (CDG) acyclic[34]. *Channel Dependency Graph* or CDG of the given network is the function of both the topology and routing algorithm used in the network. Each link in the topology represents a vertex in the CDG and each edge joining two vertex represents the turn as allowed by the routing algorithm. If there are cycles present in the CDG of the network then it is *deadlock prone*. In essence, an acyclic CDG establishes a total order in the acquisition of buffer resources by network packets with deadlock freedom guarantee.

For Mesh topology we can make the CDG acyclic using standard routing algorithms presented below without delving into the complicated CDG analysis.

Dimension Ordered Routing

Dimension order routing such as XY or YX routing for Mesh topology provides deadlock freedom. It allows packet from source S to destination D to traverse in one dimension, say

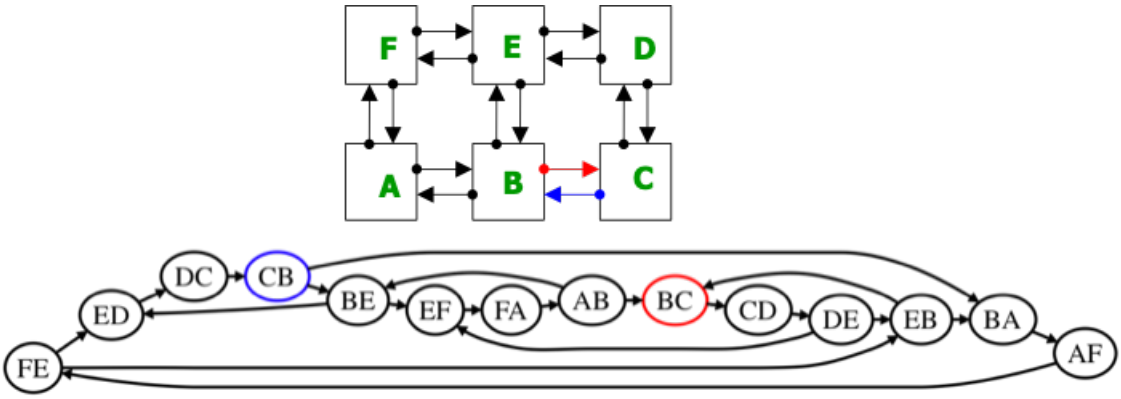


Figure 3.1: Figure shows the CDG of a 2x3 Mesh. Cycles presents in the CDG shows that network is *deadlock prone* with the routing algorithm used to build this CDG.

X, completely before switching to next dimension, say Y, for XY routing. It is applicable for n-ary k-cube topology and can be extended to 3D NoCs.

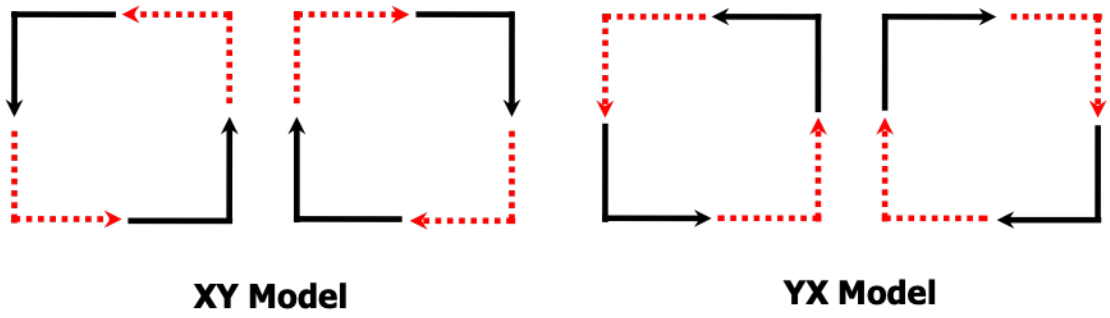


Figure 3.2: Deadlock Free DoR routing for Mesh

Turn Model

Turn model routing algorithms are deadlock free routing algorithms that are specific to Mesh topology. They provide more path diversity than DoR routing such as XY or YX. Moreover, these algorithms can be made adaptive in order to avoid the region of local congestion in the network. In Credit based signaling, number of free buffers present at the downstream router can be used as a measure of congestion in a given direction. Therefore, whenever there is a choice (as allowed by routing algorithm) of direction, then packet can choose the direction which has lowest congestion (or maximum number of free buffers

at the downstream router). For example, in West-First turn model, if the packet is going East-North, then it can decide at each hop if it wants to go North, or East based on local congestion information available.

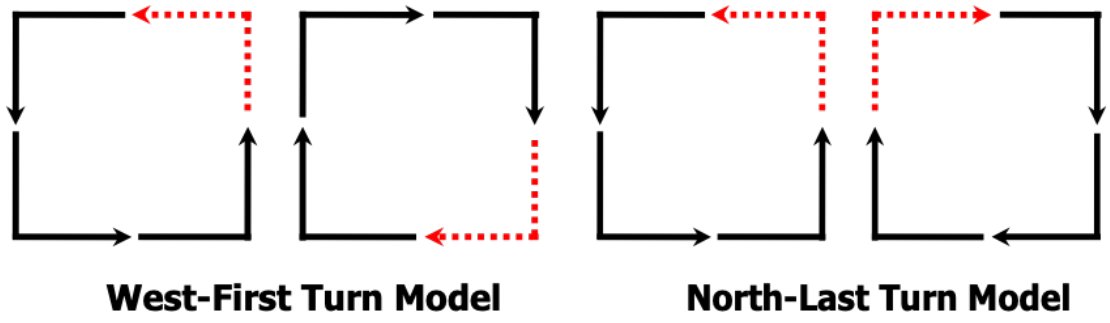


Figure 3.3: Deadlock Free Turn Models routing for Mesh

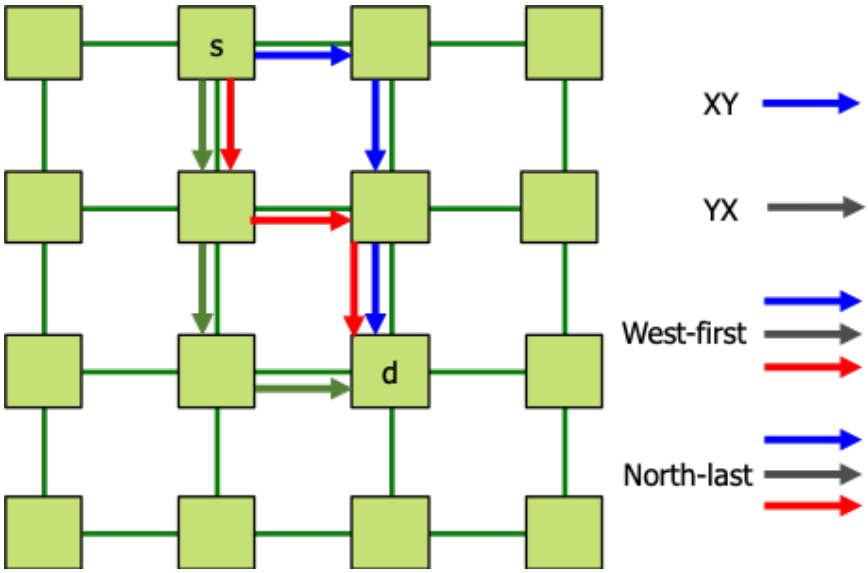


Figure 3.4: Bring it all together, the figure shows different choices of path that a packet can take for a given Mesh topology with different routing algorithms

3.1.2 Resource ordering

To avoid deadlocks to occur *Resource ordering* technique can be used. Here Links are resources which can be assigned weights. A packet is allowed to go from its source S, to its destination D by only acquiring links which have weight greater than or equal to current

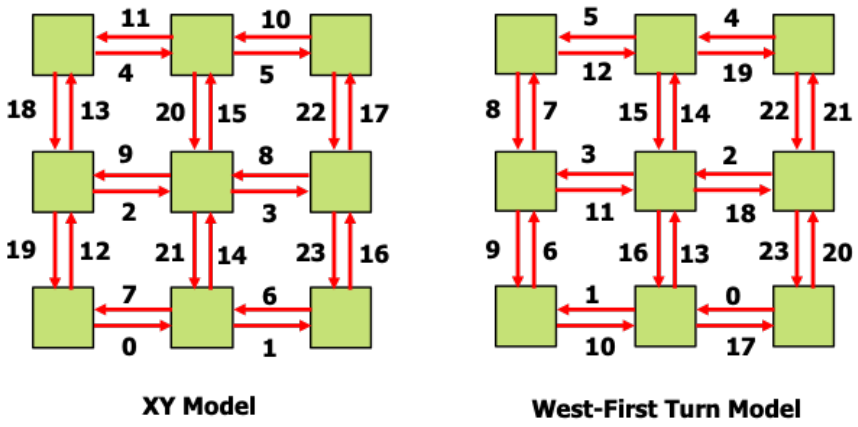


Figure 3.5: Resources (links) can be assigned weights to realize DoR or Turn Model routing in Mesh as shown

link acquired by the packet. Resource ordering concept can be used to apply different routing algorithms as shown in the figure.

In summary, the most common technique to avoid deadlocks is to make the Channel Dependency Graph (CDG) acyclic. In one variant of this technique, certain turns in a given topology are not allowed, to ensure that a deadlock is never created. The turn model for a mesh is the most prevalent implementation. These algorithms allow selective adaptivity in the routing algorithmfor example, west-first routing only allows adaptivity if the destination happens to be in the North-East or South-East quadrant of the mesh. An alternate implementation is to change the virtual channel (VC) at which the packet would sit at downstream router whenever certain turns are made, to ensure that the VCs themselves do not form a cyclic dependence. This is used in off-chip networks for algorithms such as UGAL[35] in dragon-fly networks and require at least three VCs. Fully adaptive routing can be implemented to allow full path diversity across all VCs, while each VC itself has turn restrictions[36]. In irregular topologies, arising due to network faults or power-gated nodes, spanning trees are often used to guarantee the same acyclic CDG behavior.

3.1.3 Up*/Down* Routing

The baseline **deadlock-free** routing solution in an irregular topology is up*/down*[25] routing. It is based on the turn-restriction model and tags each link in the topology as either up* or down*. This approach however is expensive not only in implementation because of extra memory needed in the form of routing table to implement up*/down* routing but also leads to non-minimal packet traversal in the network.

Theory

The theory behind the up*/down* routing is that in a connected graph, we choose one node as the *root* node. Any link that goes towards the root is tagged as up*. That is suppose a link connects node-A to node-B, if the minimum distance from node-B to the root in terms of number of hops is less than the distance from node-A to the root, then that link is tagged as up* link. Similarly, a link that takes packet away from the root is tagged as down* link. The links (connecting node-A to node-B) which do not change the minimum distance from either node-A or node-B to the root node can either be tagged as either up* or down*. If they are in opposite direction then one link will be tagged as up* and another as *down.

up*/down* routing provides deadlock freedom by restricting the order a packet can traverse the link to reach to its destination. As we have tagged all the links as either up* or down*, a packet can take following pairs of combinations of one hop link traversals to reach its destination node:

- up*-up*
- up*-down*
- down*-down*
- down*-up*

A valid (deadlock-free) path in up*/down* routing is the one in which one of the up*-down* or down*-up* hop traversal is prohibited. The tool prohibits down*-up* order of

link traversal. For example, the following order of link traversals are allowed in the tool: [up*-up*-down*-down*] for a packet to reach its destination node in three hops. In practice up*/down* routing is implemented using routing table which requires expensive buffer storage. Each path in the routing table for any pair of src-dest node follows the up*/down* routing as mentioned above with turn restriction.

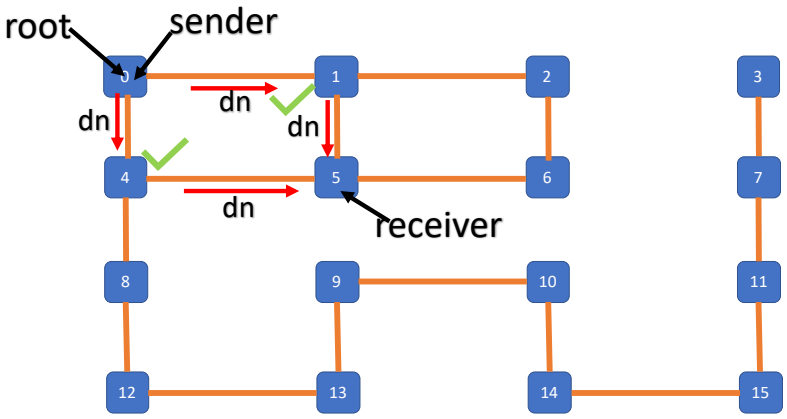


Figure 3.6: Limited path-diversity provided by the up*/down* routing

This makes the network deadlock free as the resulting channel dependency graph is acyclic. However, up*/down* routing is restrictive in terms of path-diversity as we will see in the next section.

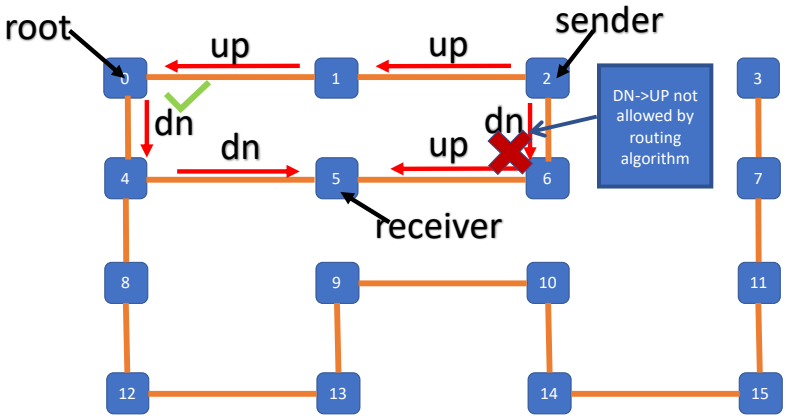


Figure 3.7: up*/down* routing can lead to non-minimal path traversal because of its turn restriction as shown in this topology with given sender(src) and receiver(dest).

Path diversity

up*/down* routing provides limited path diversity as we see in Figure 3.7 and Figure 3.6. Here root and sender are the same node and both paths are possible for sender to send packet to destination. However, in certain cases as shown in Figure 3.7 the senders packet may have to take the non-minimal path to reach to its destination and path diversity are also reduced.

3.1.4 Escape Virtual Channel (Escape VC)

Escape Virtual Channel (Figure 3.8) guarantees deadlock freedom for networks with or without cycles in their CDG as long as there exists a connected sub-graph in the extended CDG that is acyclic. It splits each physical channel into a set of additional VCs (i.e., buffers) to form an escape network. Packets in the regular VCs are routed adaptively while those in escape VCs get routed using a turn restriction-based deadlock free routing algorithm as explained above. Escape VC can be used for both deadlock- avoidance and deadlock-recovery.

However, it suffers from (a) energy and area overheads of escape network buffers, and (b) additional routing tables/logic to support deadlock free routing within the escape VCs.

3.1.5 Flow control

Another technique to provide routing deadlock freedom is to make sure that cyclic dependency of packet does not occur at run time even though CDG could be acyclic. One of the first work in this direction is Bubble Flow Control[37], the underlying theory is that as long as there is one *bubble* or free-VC present in a *ring topology*, the network is deadlock-free. Bubble flow control is applicable to ring topologies, there are several implementations of BFC which apply it to regular topologies such Mesh and Torus and irregular topologies as well. This technique has been extensively used to provide deadlock freedom for Torus networks. VCs are divided into two sets: regular/normal and escape, similar to the division

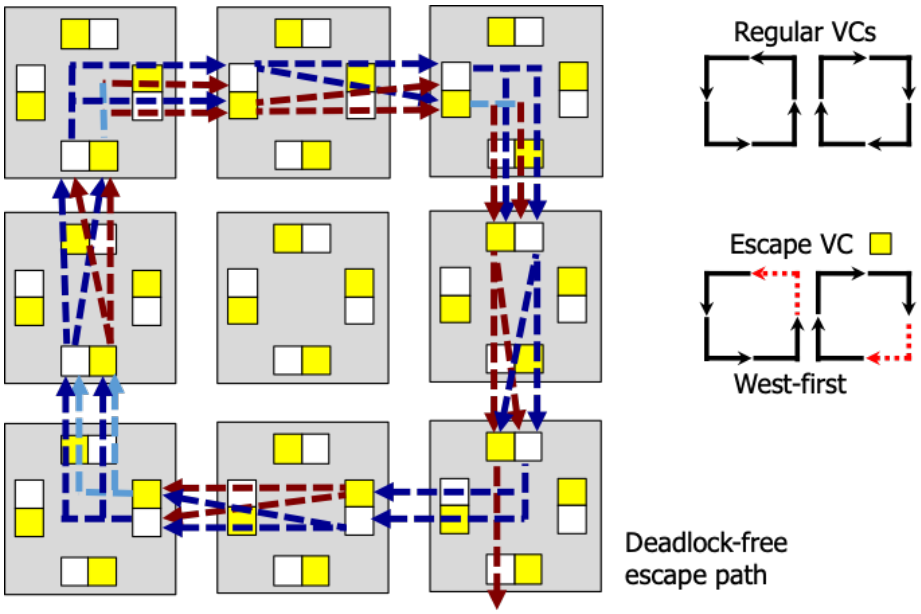


Figure 3.8: Figure showing escape VC in a 3x3 Mesh. Here Escape VC follows deadlock free Turn Model routing (West First), while Normal VC follows random (minimal) routing

used in Escape VC. Fully adaptive routing is used in regular/normal VCs while packets are routed using Dimension Ordered Routing (DOR) with BFC within the escape VC. Static Bubble[38] leverages BFC within dynamically changing irregular topology derived from the mesh topology. Critical Bubble Scheme (CBS) [39] implements global BFC by marking one packet sized VC in each ring as Critical Bubble. The critical bubble flows backwards as the packets move forward in the ring. Worm-Bubble Flow Control (Worm-BFC)[40] extends CBS to wormhole routing. Buffers are colored as grey, white and black and serve as token for routers ensuring the presence of critical bubble in the ring at all times thereby preventing starvation and deadlock. Flow-control based schemes suffer from limitations like buffer coloring/token capturing complexity, expensive solutions required for preventing packet injection starvation particularly for wormhole routing, energy overhead of circulating the token/buffer color information and lack of guarantees for packet latency that have prevented their adoption in commercial and academic designs.

3.1.6 Deadlock Recovery

Another school of thought for resolving deadlocks stems from the observation that deadlocks are a rare phenomenon, therefore instead of adding turn restrictions or extra VCs in the routers to avoid deadlocks, one should detect, locate, and resolve deadlocks. Deadlock recovery algorithms allow packets to take all paths provided by the topology to reach to their destination. However, in doing so, the packets can get deadlocked due to cyclic CDGs. To recover from it, these techniques require detection via timeouts, location via probes, and resolution via synchronization messages. Implementing deadlock recovery is therefore quite complex. Moreover, this technique is not scalable; as network sizes increase, the probability, shape, and length of deadlock-ring also increases, making it even more difficult to locate them dynamically. There have also been proposals to drop flits in the network, if they fail to win the switch for a threshold number of tries, thereby breaking any deadlocks. However, this approach comes with the overhead of tracking and re-transmitting the dropped flit as NoCs do not tolerate packet/flit losses.

SPIN: Synchronized Progress in Interconnection Networks

SPIN[41] stands for Synchronized Progress in Interconnection Networks. It also proposes to dynamically detect deadlocks as they occur in the network. SPIN is applicable to both regular (Mesh, Torus, etc) and irregular topologies. One salient insight of SPIN is that it views routing deadlocks as a lack of communication/coordination problem rather than as lack of network buffers problem. It proposes that we do not need extra buffers in the form of Escape VC or extra free VC in the form of *bubble* as in *Bubble Flow Control*, if we can coordinate between the packets to move synchronously along the deadlock-ring then after finite *spins* or packet movement we can resolve the deadlock.

SPIN is a *new* theory for routing deadlock freedom as it proposes to resolve deadlocks by *coordination* of moving packets along the deadlock ring unlike other works in the domain of deadlock recovery which use mechanisms to eject any one packet out of the

deadlock ring via extra buffers to resolve deadlocks.

SPIN uses global coordination to achieve deadlock freedom. This requires extensive deadlock detection circuitry in hardware to dynamically map and detect deadlock ring via *probes*. Probes are sent out after expiration of time-out counters, which tracks the time elapsed since the packet at the head of the router queue has not moved. Time-out counter resets as the tracked packet moves out of the router. Each router has a empty buffer which records the dynamic deadlock cycle detected by probes and instruct the movement of packets according to the length of deadlock cycle. In the original SPIN paper network topology was considered to be homogeneous with equal link bandwidth and latency. SPIN idea can be extended to heterogeneous topology with variable link and/or router delays and link bandwidth with the help of extra buffering.

The idea of SPIN is explained with the help of a figure below:

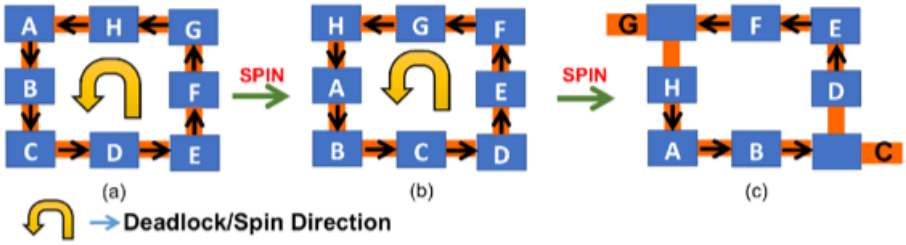


Figure 3.9: Deadlock Freedom with SPIN[41]

One of the challenges in SPIN (or other Deadlock Detection) is when link latency becomes much more than packet switching latency in a router. This is typically the case in off-chip networks, then the cost of deadlock detection and/or coordination can become prohibitively expensive. For example, consider link-latency to be 100 cycles and router latency to be 1 cycle for packet switching. Here we are not considering queuing delay experienced by packet while buffered at the router. Also considering the deadlock is spanning eight routers, then it would take num_routers_involved x router_latency + num_links_involved x link_latency number of cycles. This comes out to be 808 cycles in above scenario just to detect deadlock, then there would be equal number of additional cycles to coordinate the

movement of packets involved in the deadlock.

Moreover, from energy perspective, if link energy is more, then deadlock detection mechanism tends to be more dynamic energy expensive compared to peer deadlock freedom techniques.

3.1.7 Deflection Routing

Deflection routing[42] or Hot-potato[43] routing eliminates routing buffers by requiring routers to assign every input flit to some output port every cycle. When more than one flit requests the same output port, only one (chosen according to a priority scheme) is allotted the output port and the rest are deflected to some other available output port. Each flit however is assigned a unique output port and is not buffered. This idea, and its variants has been leveraged by BLESS[42] , MinBD[44] , CHIPPER [45], and others.

Deflection routing is a not a deadlock-freedom theory in itself; its inherent deadlock-free nature is a result of the observation that for any given router with n output ports, there will always be some matching of up to n input packets to the n output ports such that packet movement can be ensured without causing deadlock (though not always forward progress). Deflection routing, however, suffers from major limitations, including requirement of live-lock freedom solutions, large reassembly buffers for out of order packet delivery and lack of guarantees on packet latency. In addition, it offers lower saturation throughput compared to buffered routing algorithms and higher packet latency and network energy consumption at high loads due to misrouting [11].

From the discussion above we understand there are different techniques for providing deadlock freedom. Each technique has its own merits, among many factors which decide using one technique over other, network *topologies* play a vital role.

Many of the techniques proposed would work on a given set of topology, for example *XY* routing only works in Mesh to provide deadlock freedom. In Torus topology the XY routing is *routing deadlock prone*

3.2 Protocol Deadlock Freedom Techniques

In this section we discuss briefly the solutions proposed in the literature to provide Protocol Level deadlock Freedom. Surprisingly protocol level deadlock has not observed same attention as that of routing level deadlock from research community. Some of the prominent solutions are explained as follows:

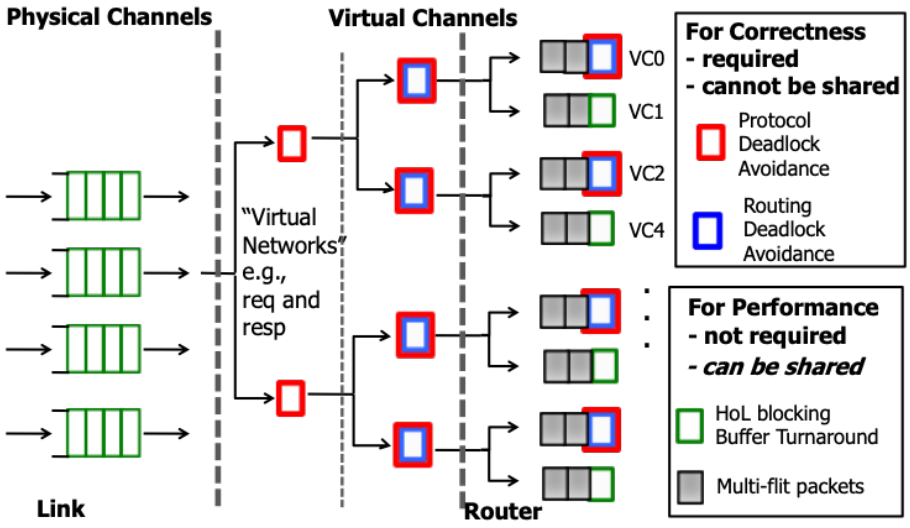


Figure 3.10: Figure shows minimum number of buffers typically present in a modern NoC to Routing and Protocol Level Deadlock freedom. Buffers for performance are optional, but buffers for deadlock freedom are essential.

3.2.1 Virtual Network

Virtual Network[46] is a straightforward way to provide protocol level deadlock freedom. Here each message class is allocated its own set of buffers at each input port (including ejection port) of all the routers of the network. This ensures that *response* packet is not stuck indefinitely because all the buffers of the network are occupied by *request* packets. However virtual network comes at the higher area and power cost because of buffers. Moreover, Virtual Network adds extra complexity to the switch arbitration stage of the router pipeline, which is in the critical path and determines the frequency at which a router can operate. Therefore, Virtual Networks also impact the performance negatively.

Figure 3.10 shows the number of Virtual Channels needed per input port of the router, both for functional correctness and then providing higher performance.

3.2.2 Protocol Deadlock Detection

Disha[47] is one of the first work in the direction of providing deadlock freedom by detecting deadlocks. However, it was originally proposed for routing level deadlocks. Since then many variants of Disha have been proposed to provide deadlock freedom for example with wormhole from control[48].

One such variant of Disha called as mDisha[49] provides message dependent deadlock freedom (also referred to as Protocol Level Deadlock). mDisha just like its variant is a deadlock detection and recovery scheme. It dynamically detects the protocol level message dependence using tokens and then provide deadlock recovery mechanism.

3.2.3 Bubble coloring

Bubble coloring[50] is a protocol level deadlock freedom technique, just like mDisha, it is applicable to both regular and irregular topologies. It is based on the principle of BFC[37]. It uses bubble for each message class, and converts the topology in a *knotted virtual ring* running through all the input port of the network. Since BFC claims that the network is deadlock free as long as there is one bubble present in the ring, hence packets are routed non-minimally over the knotted virtual ring created from the topology.

There are escape VC implementation available for bubble coloring at a cost of higher area and power overhead. This scheme and its variants pay the overhead of tracking the bubble for each message class, and results in lower performance because of non-minimal packet traversal.

3.3 Taxonomy

We provide a new taxonomy to classify the solutions proposed in the field of network deadlock freedom as follows

3.3.1 Proactive solutions

Proactive deadlock freedom (Figure 3.11(b), and Figure 3.15(b)) solutions are those which make sure deadlock does not occur in the network to begin with. In essence proactive solutions *proactively* prevent deadlock to occur in the network.

For routing level deadlocks prior techniques that come under this category include turn-restriction/routing restrictions for example: DoR or Turn model for Mesh, Up*/Down* routing for irregular topologies, Escape Virtual Channel, Bubble Flow Control and its variants and Deflection Routing. For protocol level deadlocks freedom proactive solutions include Virtual Networks, and Bubble Coloring.

3.3.2 Reactive solutions

Reactive solutions (Figure 3.11(c)) argue that *network deadlock* is a rare phenomenon as it requires certain ensemble of packets requesting to move in a certain direction so as to form cyclic packet dependency. Therefore, instead of paying higher area, power and performance overhead of avoiding deadlocks by restricting turns or allocating extra buffers in the form of escape VC, one to detect and recover from them if they ever happen in the network. In essence reactive solutions *react* to a deadlock situation when it arrives.

For routing level deadlocks solutions include Disha[47] and its variants, Static-Bubble[38] and recently proposed SPIN[41]. For protocol level deadlock freedom solution include mDisha[49] and its variants.

Table 3.1: **Comparison of solutions for routing-level and protocol-level deadlock freedom.**

	Types of solutions	High Performance	Low Area and Power	Low Hardware Complexity	Resolves Routing-Level Deadlock	Resolves Protocol-Level Deadlock
Turn Restrictions [51]	Proactive	✗	✓	✓	✓	✗
Escape VCs [52]	Proactive	✓	✗	✓	✓	✗
Virtual Networks [53]	Proactive	✓	✗	✗	✓	✓
SPIN [41]	Reactive	✓	✓	✗	✓	✗

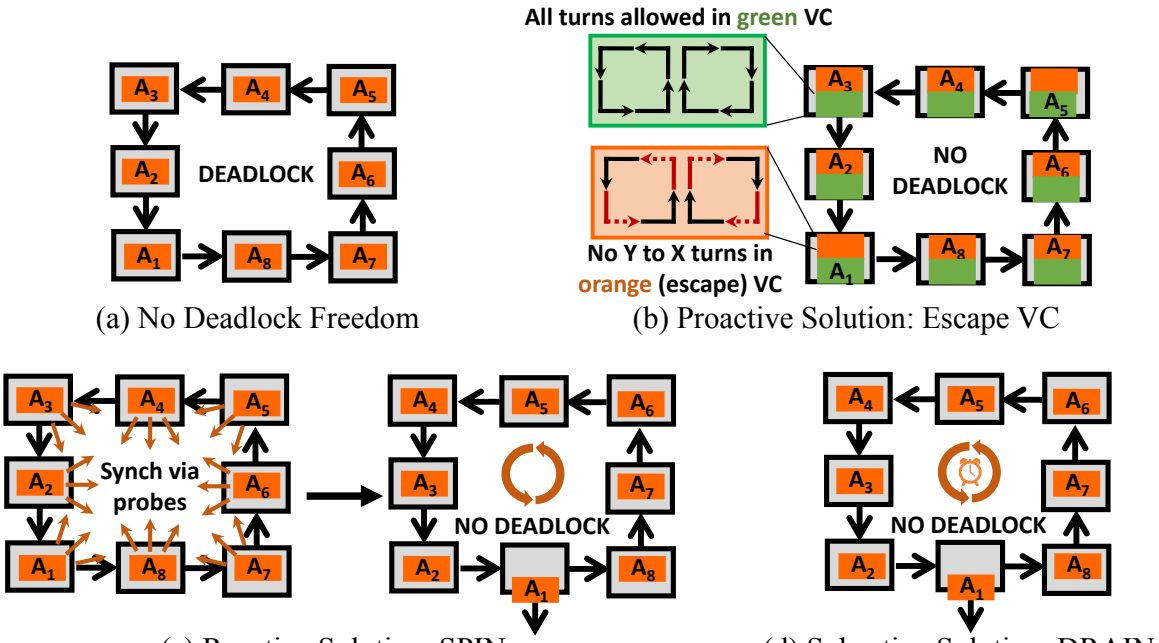
3.3.3 Subactive solutions

Subactive class of solutions (Figure 3.11(d), and Figure 3.15(c)) shows example of new type of solutions proposed in this thesis. These solutions propose neither to allocate network resources to avoid deadlocks like proactive solutions nor to implement complex circuitry to detect and recover from the deadlocks. Instead these solutions provide periodic coordinated packet movement to make sure if there is any deadlock present in the network then it gets cleared away. Next few chapters are dedicated to discussing each of the subactive approach proposed.

Primary objective of this thesis is to *rethink* the way we provide deadlock-freedom in interconnects and communication protocols to decrease the expensive buffering that traditional deadlock-freedom mechanisms require.

3.4 Motivation for subactive deadlock freedom

Designing routing and protocol deadlock freedom in the face of increasing irregularity (as failures increase over time) is very challenging. Both proactive and reactive solutions have fundamental limitations, as listed in Table 3.1. This thesis exploits the insight that dead-



(a) No Deadlock Freedom

(b) Proactive Solution: Escape VC

(c) Reactive Solution: SPIN

(d) Subactive Solution: DRAIN

Figure 3.11: Routing-level deadlock and solutions.

locks *very rarely* occur in practice. Deadlocks require a specific confluence of packet routes and timings to actually emerge in a given network. Given the rarity of actual deadlocks, should designers spend precious runtime power to mitigate this *remote possibility*? We believe the answer is *no*.

3.4.1 Observation: Deadlocks are Rare

To demonstrate the rare and unlikely occurrence of deadlock, we look at both application workloads and synthetic traffic (Figure 3.12, Figure 3.13). In Figure 3.12, links are removed randomly from an 8×8 mesh to simulate faulty, irregular topologies. All nodes remain connected to the network when links are removed. Here the routing algorithm is fully adaptive and not deadlock-free by design. Each PARSEC [33] workload is run five times with 1 VC and 4 VCs per virtual network. The grayscale corresponds to the percentage of runs that result in a deadlock. No deadlocks are observed for the fully functional case (i.e., 0 links removed). Note that because the routing algorithm is not deadlock-free, even with no links removed, deadlocks are possible in this network. Only upon removing four

links do we begin to encounter deadlocks for `canneal`, which has the highest injection rate of these five workloads. A higher injection rate implies that there may be enough packets in the network at any given moment for a deadlock to emerge. As more links are removed, deadlocks become more common across several of the workloads; removing more links increases the likelihood that packets can coincidentally form a cycle on the remaining links. Note that the presence of additional VCs may delay the onset of deadlock but is not sufficient to provide deadlock freedom.

Furthermore, previous work [38] has shown that faulty topologies are more deadlock-prone than fault-free topologies. This is because faulty topologies limit the path diversity, resulting in higher hop counts in the network. Thus, packets stay longer in the network and have a higher chance to be involved in a deadlock cycle. This is why deadlocks can occur at a lower injection rate in faulty topologies.

Takeaway. Even in the absence of any explicit deadlock avoidance or prevention mechanism, the occurrence of deadlocks is quite rare. Thus, we aim to follow the adage of "make the common case fast": deadlocks are uncommon but still need to be handled correctly. We do not want to devote significant hardware resources to such an uncommon case nor do we want to cripple the performance of the common case of deadlock-free operation by imposing routing restrictions on all network packets. We use this insight to guide us towards a new design: we eventually resolve deadlocks, should they occur, but we achieve this at very low cost and design complexity while maintaining flexibility.

3.4.2 Observation: Virtual Networks are Costly

Figure 3.14 shows the total power consumed by virtual networks, the de facto solution to protocol-level deadlock freedom. The number of virtual networks depends on the cache coherence protocol of the system. In the figure, active power refers to power expended to transfer a packet through the virtual network, while wasted power refers to power expended even though no packet was in flight. We observe the vast majority of power consumption

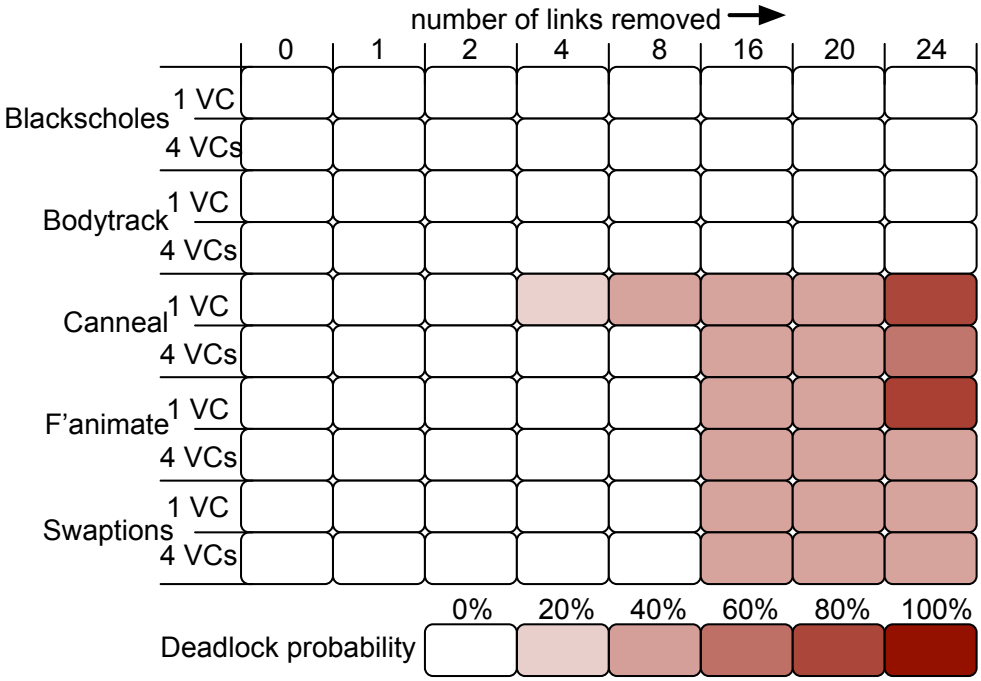


Figure 3.12: Likelihood of routing deadlocks for PARSEC workloads as links are removed from an 8x8 Mesh topology.

		MOESI_Hammer		
Benchmarks	**fault-0**	**fault-1**	**fault-4**	**fault-8**
blackscholes				
bodytrack				
canneal				
fft				
lu_ncb				

Figure 3.13: Protocol deadlocks incurred for PARSEC workloads as links are removed from an 4x4 Mesh topology. 'Red' indicates a protocol deadlock while 'Green' corresponds to the successful completion of the application.

in virtual networks is wasted. Virtual Networks are composed of virtual channels. Here explain the overhead because of virtual channels. **Virtual Channel Overhead.** VCs (Virtual Channel) require significant buffering in the router. VCs not only takes up higher area but also contibute to higher power consumption in the network. There is also *state* overhead associated with each virtual channel. Having higher number of VCs complicates switch arbitration and increases the cretical path delay. This effectively lowers the frequency at which interconnect can operate. In fact buffers in the router are the major contributor for area and power. For example, supporting 8 VCs incurs a ~3× increase in area and ~2× increase in power of the interconnection network. Implementing coherence between private chaces, in commercial shared memory multiprocessors, requires 5 to 16 virtual (or physical) channels per router port to ensure deadlock-freedom. This is indicated in the specs frrom commercial chips such as Tilera iMesh[54], Intel QPI[55], and AMD HyperTransport[56] respectively. These extra VCs are required over already essential exta VCs to avoid routing level deadlock. Furthermore as the protocols become more complex due to heterogeneousl processing elements, higher number of VCs will be essential to make the interconnection network deadlock free, this will excerbate the of deadlock freedom[57, 58].

Takeaway. Despite the wasted power, virtual networks are still needed; otherwise correct execution is not guaranteed due to protocol-level deadlock. We strive for a solution that is capable of resolving protocol-level deadlocks simultaneously to resolving routing-level deadlocks.

This thesis proposes a set of solutions that periodically *reconfigures* network resources to recover from *potential deadlocks* in the network. This thesis introduces the concept of *subactive* deadlock freedom: neither avoids nor reacts to deadlocks but rather lets them happen and eventually cleans them up, as shown in Fig. 3.11d and 3.15c. Solutions proposed guarantee deadlock-free operation without adding any performance restrictions nor expensive buffers, unlike in proactive solutions. These solutions are the first to employ

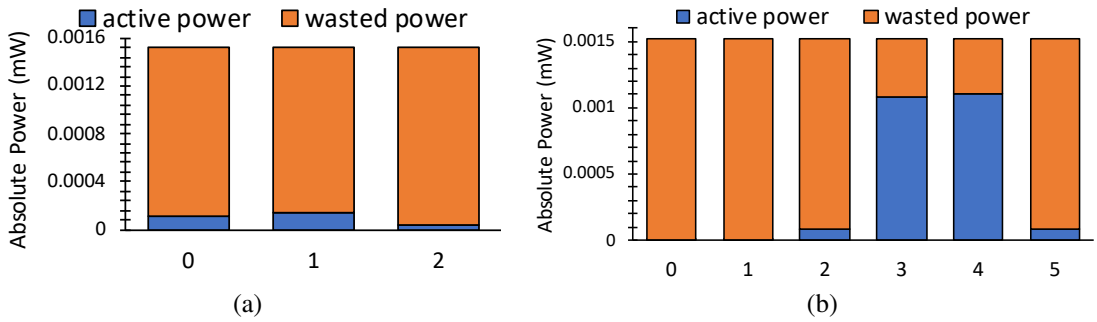


Figure 3.14: X-axis presents the Virtual Network id (VNet). Wasted power in virtual networks for (a) MESI cache coherence protocol and (b) MOESI hammer cache coherence protocol

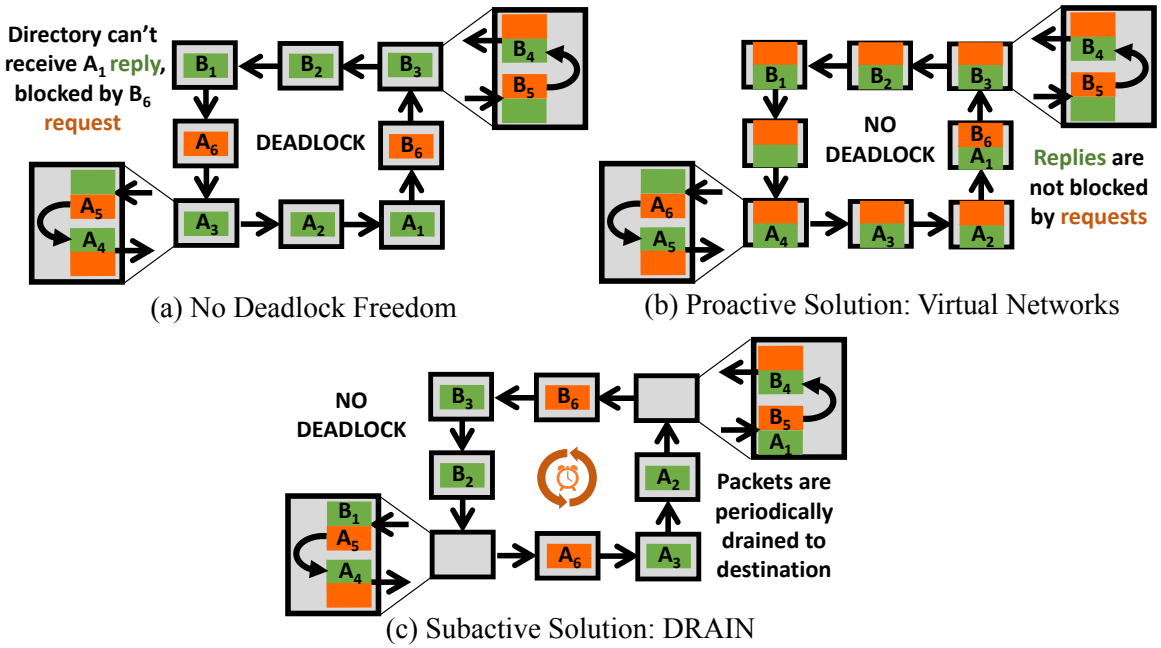


Figure 3.15: Protocol-level deadlock and solutions.

oblivious deadlock removal; they require no complex detection nor recovery mechanisms, unlike in reactive solutions. This property makes these solutions unique in its ability to resolve both routing and protocol-level deadlocks simultaneously without the need for virtual networks.

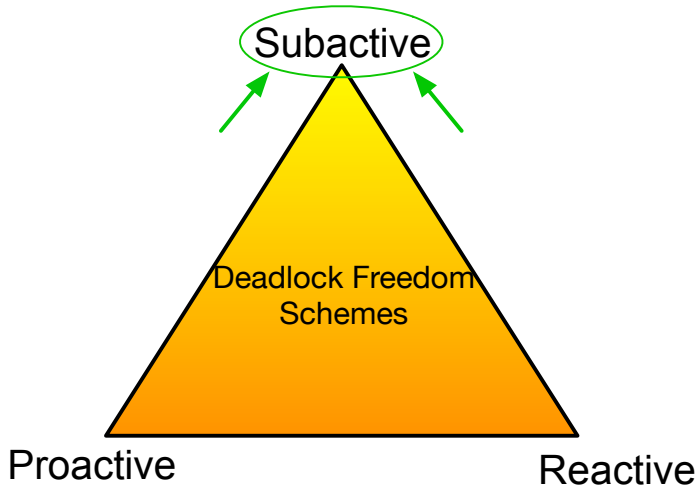


Figure 3.16: Pictorial representation of a new Taxonomy of deadlock freedom schemes. *subactive* approach introduced in this thesis has favorable traits of both proactive and reactive solutions, therefore it is shown on the apex of the *deadlock freedom scheme* triangle.

3.5 Chapter Summary

In this chapter we provided the state-of-the-art solutions proposed to provide routing and protocol level deadlock freedom. We discuss merit, as well as challenges of the solutions and conditions under which they are applicable (for example regular and/or irregular topology).

We provided the motivation to renew the interest to study deadlocks in the interconnection networks.

We then classify the proposed solutions under a new taxonomy of Proactive, Reactive and Subactive solutions. The philosophy behind subactive technique can be seen in Figure 3.16, It shows the three vertices of *deadlock freedom scheme-triangle*, this represents three ways to achieve deadlock freedom in network. It further underlines the contribution

of this thesis, as it adds a apex vertex in *deadlock freedom scheme-triangle* to show that it is an important class of techniques to provide deadlock-freedom.

We briefly explained Subactive solutions which we will discuss in-depth in next few chapters of this thesis. Next chapter is dedicated to the tools developed to study different properties of deadlocks in irregular topologies, in conjunction with cycle accurate system simulator gem5[7].

CHAPTER 4

EVALUATION TOOLS

Studying different aspects of on-chip network using simulations, requires knowing the sensitivity of the network under different condition. This helps gain greater insight about the phenomenon of deadlock and further helped to develop novel solutions presented in this thesis. This chapter, talks about, various tools developed, their working and how to use them. The tools are hosted on GitHub page for public use[1]. There are three tools introduced in this thesis:

- **Irregular Topology Generator (ITG):** Irregular topology generator takes input from users and generates random irregular topology of given size. This tool is explained in section 4.1

- **Routing Table Generator (RTG):** It generates deadlock free *up*/down** routing table for any valid irregular topology provided. This tool is explained in section 4.2. All the files generated by *ITG* and *RTG* are read by gem5[7] to simulate the network.

- **DRAGON:** It stands for Deadlock Recognizing Topology Agnostic Network Tool. *DRAGON* is a framework integrated within *garnet*[59] to find the number, shape and size of routing deadlocks that can form within the network at runtime. The frequency of determining deadlocks during network simulation is provided by the user as gem5 command-line option. This tool is explained in section 4.3

Let us study these tools in detail as below:

[1]Irregular topology generator and populates up*/down* routing table: https://github.com/Mayank-Parasar/topology_generator

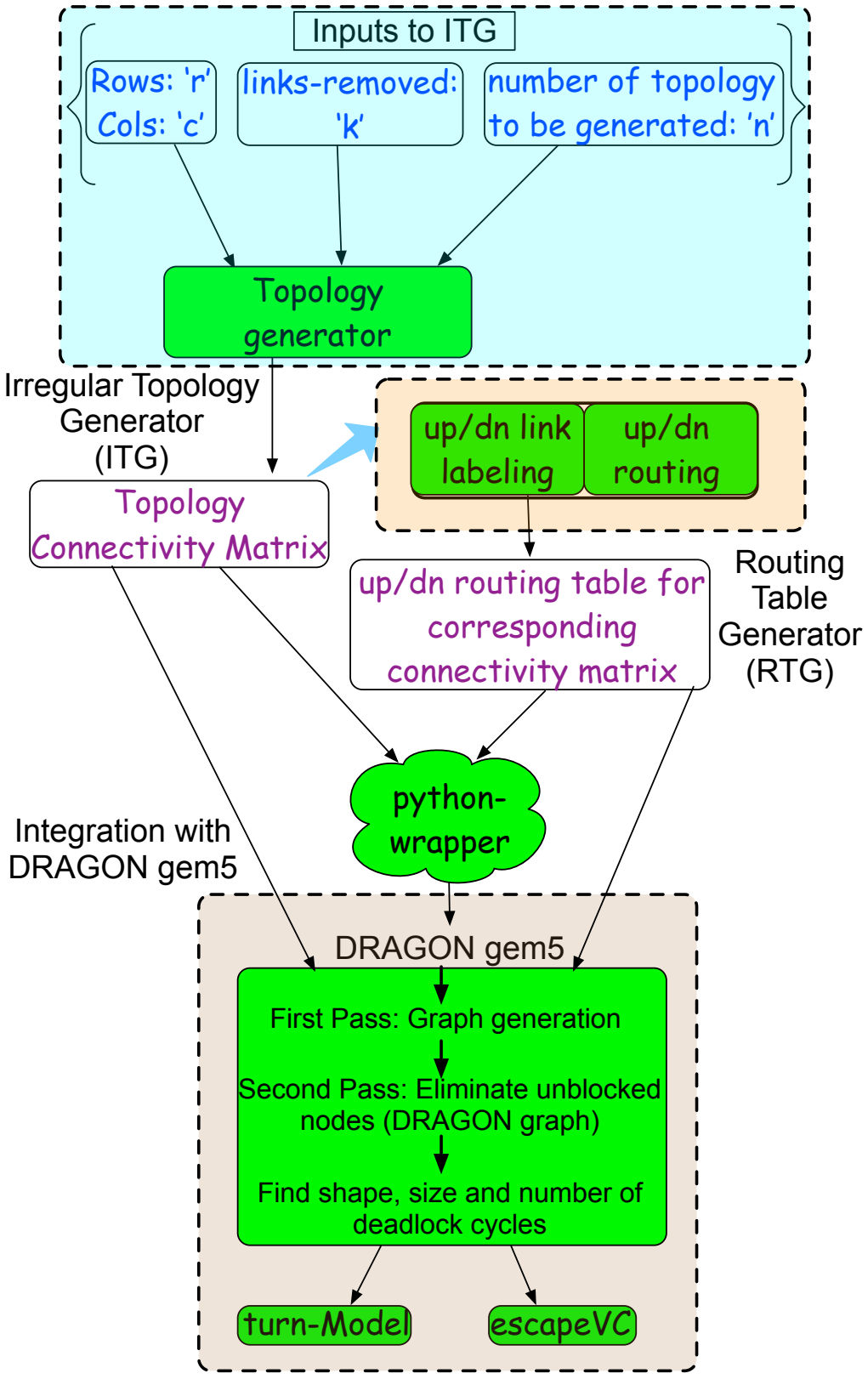


Figure 4.1: End-to-end integration flow diagram of *irregular topology generator*(section 4.1), *routing table generator*(section 4.2) with DRAGON-gem5(section 4.3)

4.1 Tool-1: Irregular Topology Generator (ITG)

Irregular topologies can arise in the network because of faults and/or power gating of nodes. In other cases, network is designed with irregular topology because of constrains from heterogeneous IPs. Few IPs could need low network latency compared to other IPs; this translates to number of hops a packet needs to reach its destination. Network designed with such considerations results in an irregular topology. This is frequently observed in designing network for MPSoC.

Therefore, a tool which can generate irregular topologies becomes important to study the effect of irregularities of the topology on performance and on other sensitivity analysis. Since the proposed techniques in this thesis use packet coordinated movement, independent of the underlying topology, to provide deadlock freedom, the proposed tool is used to compare the performance and other sensitivity metrics against prior work in irregular topologies.

4.1.1 Introduction: ITG

The tool to generate irregular topologies derived from original regular topology was developed. Although the tool can be augmented to generate *adhoc* irregular topologies, in current state it requires a baseline topology to generate irregular topology from. Irregular topology is generated by removing links randomly from the baseline topology, while still making sure topology is *strongly connected* as shown in the Figure 4.3 and Figure 4.2.

Strongly connected means there exists at least one path from any node (source-node) in the network to any other node (destination-node) after removing the link. Given, of course, that the baseline topology is strongly connected to begin with. The same path in opposite direction can be used to reach the original source to original destination. In graph theory parlance, the term *strongly connected* is referred to as *connected* and topology is said to be a *connected graph* (not to be confused by *complete graph*).

The irregular topology generated in current state assumes *Mesh* to be the baseline topology. Although the tool can be augmented to consider other types of regular topologies. For the baseline Mesh, tool assumes that there are two uni-directional links (in opposite directions) between the connected nodes as per Mesh topology.

This information is readily available in the form of *connectvty_mtrx* data structure. This data structure needs to be changed to extend the tool to other topologies (example torus, flattened butterfly, etc). The tool takes the *row* and *column* to generate the Mesh and its associated connectivity matrix.

4.1.2 Connectivity matrix

Connectivity matrix is a matrix in which column-ids represent the source nodes and row-ids represent the destination-nodes. Therefore, for a $N \times M$ Mesh, connectivity matrix is of size $(M \times N) \times (M \times N)$. Value 0 at given [row][column] index in connectivity matrix represents that there is no connection between a given source node id - column and destination node id - row, value 1 at the given [row][column] represents a connection between the nodes, while value -1 represents both row and column index represents the same node.

Connectivity matrix is a blueprint for a topology. Because of its generality is can be used to represent any type of network topology and not necessarily a Mesh. Tool works on this connectivity matrix to generate different topology and outputs the result in the form of a new connectivity matrix.

To generate an irregular topology from this given row $\times$ column Mesh, we also need to provide how many links need to be removed from the given N x M mesh. Tool after taking this input works on the connectivity matrix thus generated to remove the links. By default, it is assumed that the number provided by the user to remove the links are the pair unidirectional links, in opposite direction, connecting the two nodes. Therefore, if user input 2 as number of links to be removed then 2 pairs of unidirectional links will be removed

between two randomly chosen src-dst pair nodes. Since tool works at the granularity of removing pair of unidirectional links between two nodes, from now on the term *link* will refer to pair of unidirectional links connecting two nodes in opposite direction. This is of course in the context of the topology generator tool.

After removing any link, the tool checks if the topology is still *strongly connected*, this is done by depth-first-traversal (*DFT*) of the topology. Starting with any node (node-0 in the tool) if all nodes are visited as per the new connectivity matrix of the topology then topology is said to *strongly connected*, otherwise the removed link is re-inserted (this essentially undo the changes done to the connectivity matrix and new src-dest pair is randomly selected to remove the link between them.

The tool keeps count of how many links have been removed randomly in the topology. As the number of links becomes same as mentioned by the user, the topology is written into a file in the form of a new connectivity matrix.

4.1.3 Upper limit on the number of links that can be removed

Given a baseline topology (Mesh (k-ary 2-cube) in our case) there is a limit on number of links that can be removed while still keeping the topology *strongly connected*. We derive this condition for N x M mesh (N rows and M columns) as follows:

The maximum possible number of links that can be removed from a N x M mesh is obtained when the topology converts into a ***snake-topology*** *or simply a bus as shown in the figure.*

Maximum number of links that can be removed from the topology is evaluated as follows:

Number of links present in the snake-topology(Figure 4.2) derived from $N \times M$ $mesh$:

$$= (N \times M - 1) \tag{4.1}$$

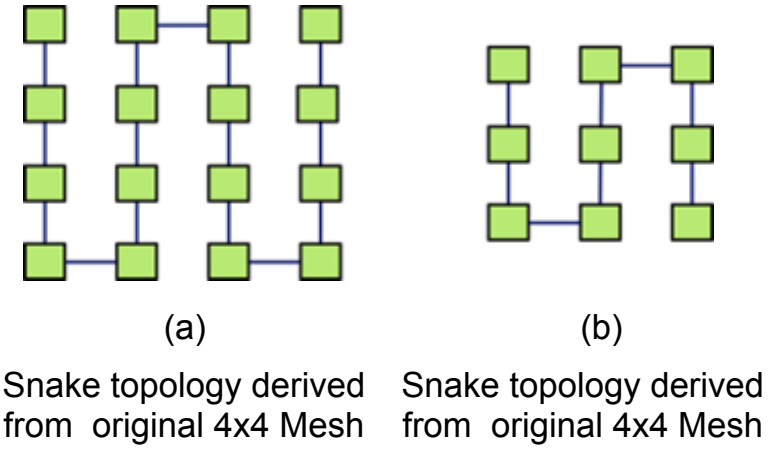


Figure 4.2: Shows the maximum link removal that can be allowed form the original Mesh topology while keeping it still connected.

Total links in the original $N \times M$ $mesh$:

$$[2 \times N \times M - (N + M)] \tag{4.2}$$

Maximum number of links that can be removed from N x M mesh such that the resultant topology is strongly connected:

$$(total\ links\ in\ N \times M\ mesh) - (total\ links\ in\ snake\ topology\ derived\ from\ N \times M\ mesh)$$
$$= N \times M - (N + M) + 1 \tag{4.3}$$

Implicit check is present in the tool, that outputs only strongly connected topology, after removing the specified number of links. In case the user input is more than maximum number of links that can be removed, then tool will not output any new topology.

Since the tool removes links from the topology randomly, a user can also specify the number of topologies he/she would like the tool to generate with a fixed number of links removed (as faults). The tool will generate that many connectivity matrices.

4.1.4 Condition of converting a N x M mesh into a ring

It is possible to convert a N x M mesh into a single ring, by continuously removing links.

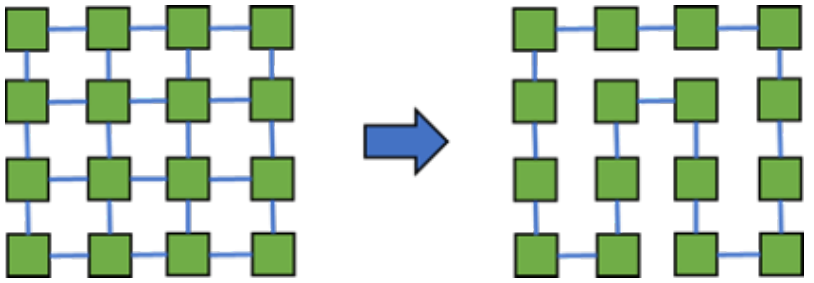


Figure 4.3: Shows the maximum link removal that can be allowed form the original Mesh topology while keeping it still connected.

However not all $N \times M$ mesh can be converted to a ring by link removal. If both N and M are odd then there is no single ring possible that connects all the input nodes, for example 3×5 Mesh.

If either N or M is even then there's a ring possible that connects all the node.

Currently this feature of deriving a ring topology from NxM mesh is not implemented in the tool, however the tool can be extended to generate one, by utilizing characteristic of 'connectivity matrix' associated with ring topology derived from $N \times M$ mesh.

4.1.5 Helper functions

To help debugging any changes and to understand the working of the code there are helper functions provided in the code, namely:

- $print\_matrix()$: For printing topologys connectivity matrix
- $print\_char\_matrix()$: For printing up-down routing matrix
- $prnt\_state()$: For printing topology information (rows, columns, links removed, connectivity matrix)

The summary of functions are present in $generate\_topology.cc$ file above the definition of the functions.

4.2 Tool-2: Routing Table Generator (RTG)

This tool generates deadlock free *UP*/DOWN** routing table for a given topology provided by *ITG*. UP*/DOWN* routing has been explained in subsection 3.1.3. The up*/down* routing information is augmented in the same file containing the connectivity matrix of newly generated irregular topology using *ITG*. Next subsection describes the implementation of up*/down* routing in the tool.

4.2.1 Implementation in tool

In the *RTG* tool, node-0 (row-id: 0, column-id:0) of the topology is assumed to be the 'root' node. With this assumption tool first populates the *UP/DOWN* matrix. It is similar to *connectivity matrix*, but now tags each link present in the new topology (as per connectivity matrix) as either u (stands for UP) or d (stands for DOWN). The corollary of using node-0 as the root node in the topology is that the UP/DOWN matrix thus generated would have 'd'/down* links in the upper right triangle and 'u'/up* links in the lower left triangle of the *UP/DOWN* matrix it generates. *legal* path is calculated as per the turn restriction mentioned in subsection 3.1.3 between every possible src-destination node pair. The routing algorithm used restricts down* to up* turn. This forms the up/down routing table.

All of this information is populated in a text file generated by this tool, which is later fed into gem5[7] to configure the garnet2.0[59] routing table to be used to implement up*/down* routing and Escape VC up*/down* routing in network simulation. gem5's python scripts are modified to read the topology file and configure the network's topology accordingly at the starting of simulation. Garnet2.0's C++ code is changed to allow routers to populate their routing table after reading from the same file. Correctness of the algorithm is checked by injecting fixed number of packets and making sure all packets are received

by the end of the simulation at different injection rate, traffic patterns[60] and topologies respectively.

4.3 Tool-3: DRAGON (Deadlock Detection Infrastructure)

We quantified the *probability* of deadlock occurrence as a measure of lowest injection rate at which first deadlock occurs in the network. Lower the injection rate of first deadlock occurrence implies that given topology and traffic patterns are more susceptible to deadlock. We quantify our observation in subsection 4.3.1.

4.3.1 Observation: Routing Deadlock Likelihood with Synthetic Traffic Pattern

We evaluated the likelihood of deadlocks for synthetic traffic patterns, under different network and router configurations and plotted the lowest injection rate at which the deadlock in the topology manifests. We used Mesh topology of different sizes (from 16 nodes (4x4 Mesh) to 256 nodes (16x16 Mesh)) and used *random routing algorithm* which routes packets minimally to their destination without any turn restrictions.

As shown in Figure 4.4, Figure 4.5, and Figure 4.6, *we observe that as topology size ion increases deadlock manifests at comparably low injection rate*. This is an important observation in the light of *Moore's law*, which says transistors per unit area of chip would keep doubling in every 18 months. One affect of increase in transistors is incorporation of more and more IPs/Cores within the chip. This would translate to higher nodes in the network. This observation calls for renewal of interest in the field of deadlocks in interconnection domain.

Figure 4.4, Figure 4.5, and Figure 4.6 shows deadlock for a given topology across different traffic patterns. We observe that *transpose* traffic pattern never deadlocks at any injection rate and with any topology size. This is attributed to the packets in *tranpose* traffic patterns never acquire the input port that results in a *deadlock*. This is attributed to *src-dst* pairs and *no u-turns/non-minimal paths* allowed in the network.

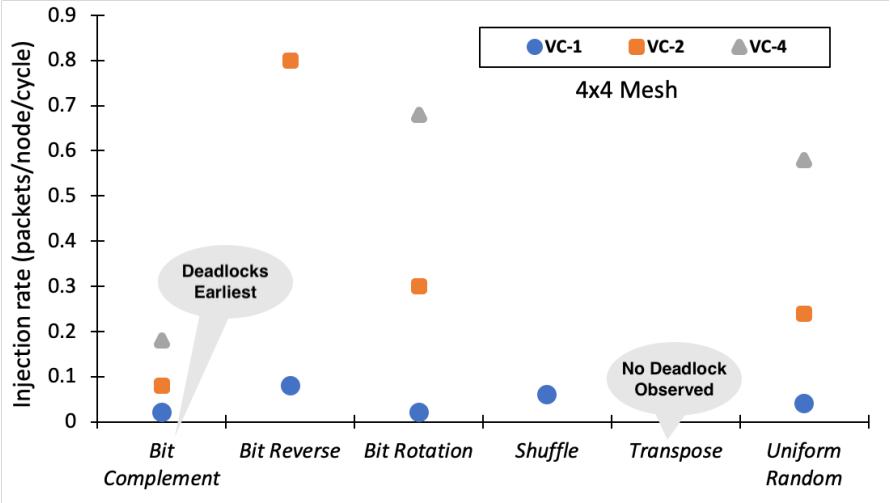


Figure 4.4: Figure shows the first occurrence of deadlock at lowest injection rate for different synthetic traffic pattern in 4x4 Mesh with routers configured as: VC-1, 2, and 4

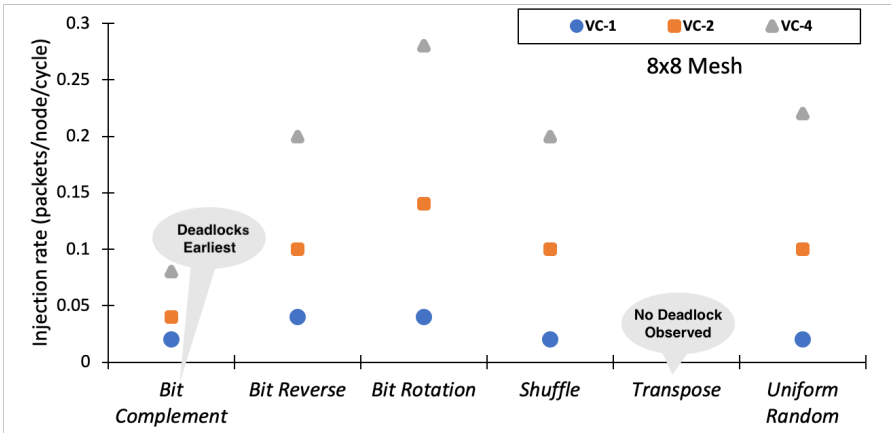


Figure 4.5: Figure shows the first occurrence of deadlock at lowest injection rate for different synthetic traffic pattern in 8x8 Mesh with routers configured as: VC-1, 2, and 4

In Figure 4.7, Figure 4.8, and Figure 4.9 we characterized the sensitivity of each traffic pattern for deadlock with respect to different router configurations and topology size.

We observed *the routers with more number of VCs per input port deadlocks later compared to routers configured with less number of VCs per input port*. This is attributed to the *buffer turn-around time*. We use credit signaling (subsection 2.9.2) to communicate the buffer occupancy among the neighboring routers.

Lesser number of VCs per input port implies packet would be buffered longer at the router before making forward progress. This increases overall time a packet spend in

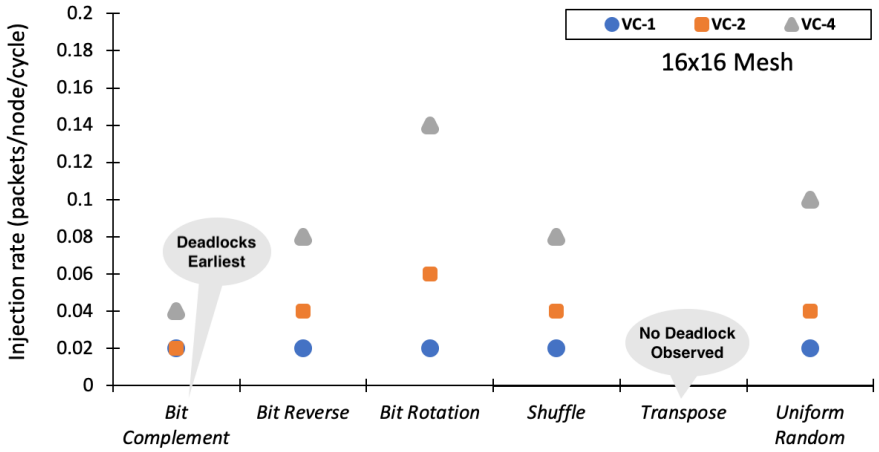


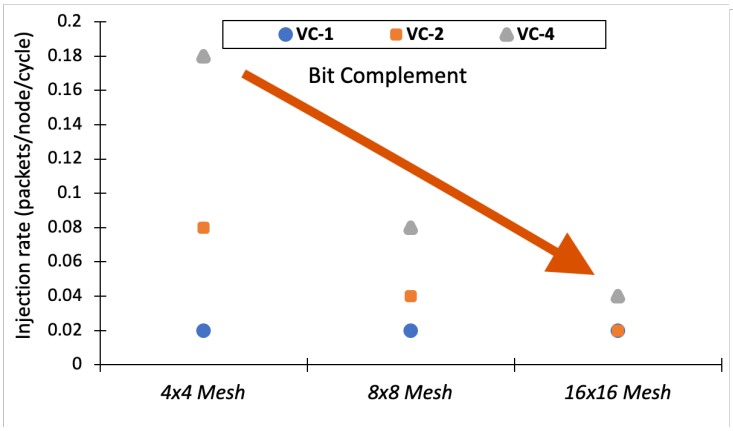
Figure 4.6: Figure shows the first occurrence of deadlock at lowest injection rate for different synthetic traffic pattern in 16x16 Mesh with routers configured as: VC-1, 2, and 4

traversing the network and hence the packets, in general, are more susceptible to deadlock under *random routing*. With more number of VCs the packet at one VC at *upstream router* can jump to different VC at *downstream router* if the VC is free, resulting in reduction of overall time packet spend in the network. Moreover, for deadlock to occur all VCs of the deadlocked input ports of concerned routers must be occupied in a cyclic fashion. This probability reduces as number of VCs per input port increases.

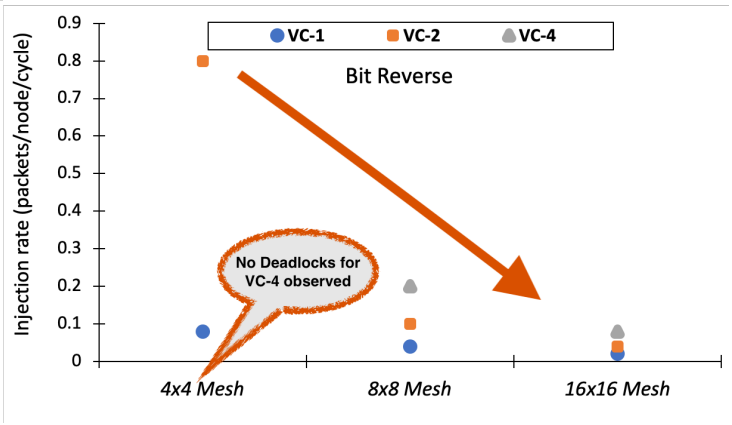
We also observed that *bit-complement* traffic deadlocks at lowest injection rate among all traffic patterns. This can be attributed to the *src-dest* node pairs used in this traffic pattern. Figure 4.10 further shows the *source destination* pairs for *bit-complement* traffic pattern when it is mapped on a 4x4 Mesh topology. The cycles in its *source destination* pairs results in a cyclic dependency (when ran with *random routing*), this results in deadlock. Since *transpose* traffic never deadlocks we did not perform its sensitivity study.

4.3.2 Introduction: DRAGON

To understand the nature of routing deadlocks, their frequency and likelihood in real applications and under different network configurations a deadlock detection infrastructure inside Garnet2.0[59] was designed and integrated it to the gem[7] simulator.

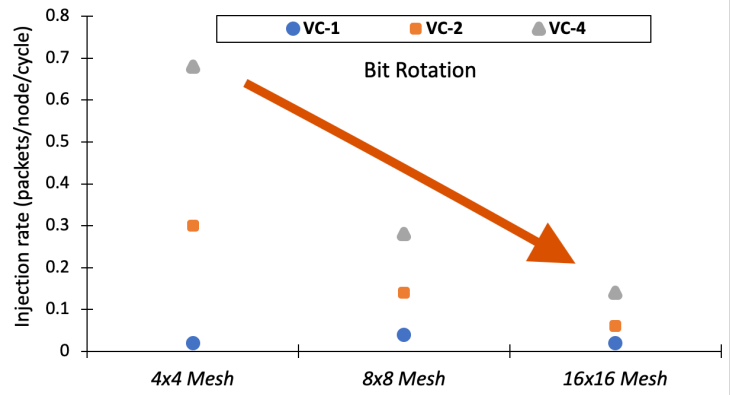


(a) Bit Complement

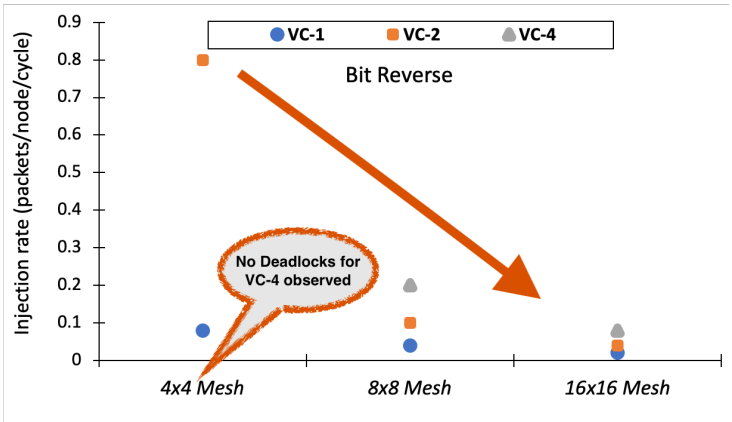


(b) Bit Reverse

Figure 4.7: First occurrence of routing deadlock as a function of injection rate and different topology sizes



(a) Bit Rotation



(b) Shuffle

Figure 4.8: First occurrence of routing deadlock as a function of injection rate and different topology sizes

This tool gives us a deeper understanding of cyclic buffer dependency of packets in a given topology that results in a deadlock. We have named the tool as: DRAGON: Deadlock Recognizing Topology Agnostic Network Tool.

DRAGON tool when combined with *random irregular topology generator* (section 4.1) explained above helped in visualizing the effect of path diversity on the likelihood of the deadlocks both for real application and synthetic traffic patterns. The random topology generator helps in generating different topologies with fixed number of link failures. With

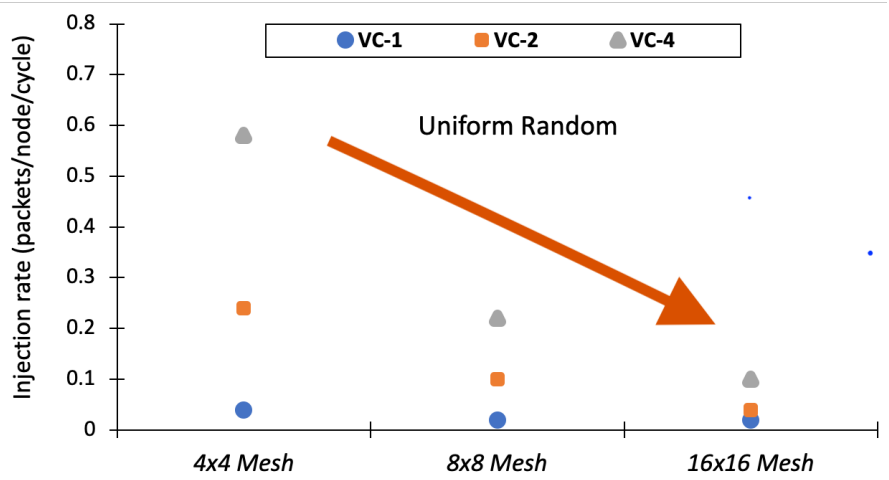


Figure 4.9: First occurrence of routing deadlock as a function of injection rate and different topology sizes

DRAGON tool we can find out how the link-failure at particular location in the topology affects the performance and likelihood of deadlocks with respect of given synthetic traffic pattern such as Transpose, Shuffle, Tornado etc. Some synthetic traffic patterns[60] are more likely to deadlock early compared to others due to the nature of flows (*src-dst* pairs), for example, *bit-complement* is more likely to experience deadlock early in the simulation compared to *tornado* traffic, because of its src-dst pairs.

4.3.3 Overview

The tool works in tandem with the gem5 network simulation. Routing algorithm in garnet is aware of paths available in the topology, packets are assigned direction by the routing algorithm towards their destination router for the next hop by the routing algorithm if there is a valid path available in the topology. This information is used by the DRAGON tool to generate the graph from the current state of the network.

We can set the time period for DRAGON tool to scan the network from command-line option and generate the graph. DRAGON generate a packet dependency graph as follows.

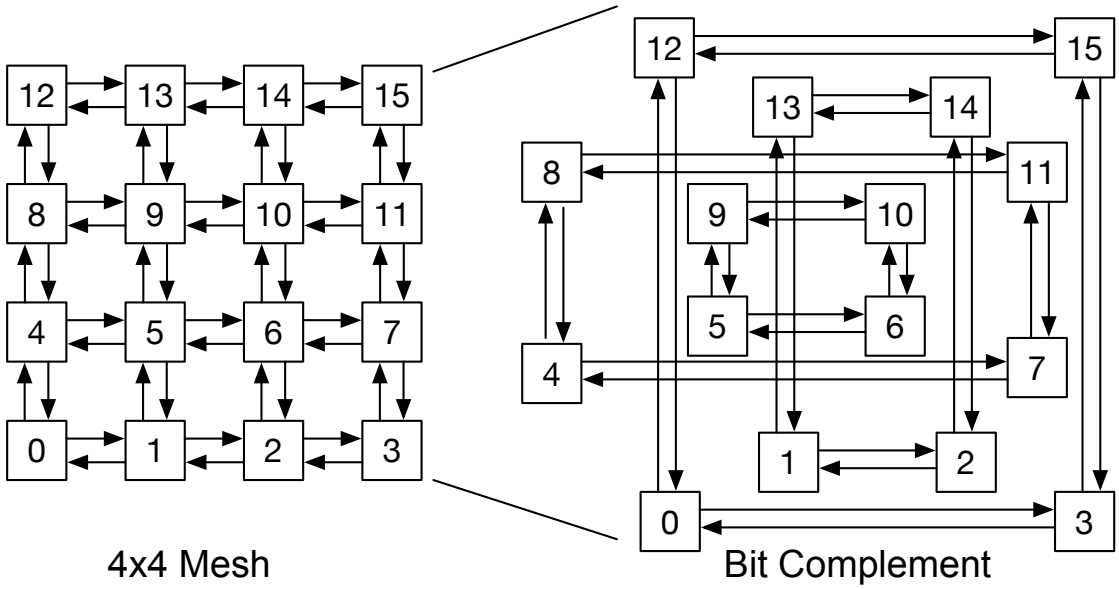


Figure 4.10: Figure shows the *source-destination* pairs in bit complement traffic pattern in a 4x4 Mesh with links arrangement considering *XY routing*. For example, node-id: 5, 6, 9, and 10 are forming a cycle where each node is one hop away and is in the center of topology.

4.3.4 Graph generation

Each node in the graph represents the input-port of the router. These nodes do not include the injection input ports of the router. Network nodes in the graph are numbered with a unique integer node-ids starting from '0' for the North input port of router-id '0'. Edges from one node to another node represents the direction in which the packet needs to move in-order to make forward progress towards its destination.

The maximum number of edges from any node would be equal to number of VCs present in the input port represented by that node in the graph. The edge would connect this node to the node-id of corresponding input port at the downstream router where packet would move next if the input port has free buffers as guided by the routing algorithm.

As chosen using command line option, we can set the frequency for the DRAGON tool to kick-in and scan the network to generate this *initial graph*.

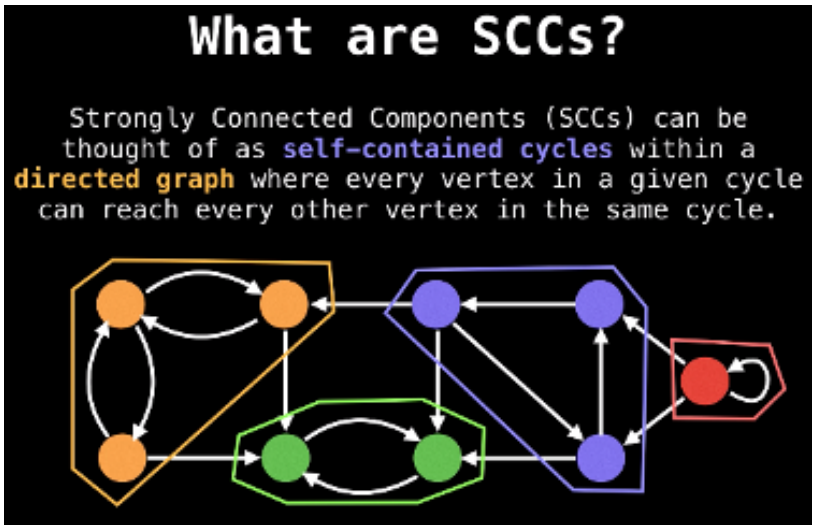


Figure 4.11: *Strongly connected component (SCC)* analysis done by standard graph algorithms to find the deadlock cycle can give approximate number of deadlock rings [61].

4.3.5 Analysis

Once this graph is generated, we can do the analysis on it to find out shape, size and number of cycles present in this graph (a representation of network current state).

Standard graph algorithms such as Kosaraju's algorithm[62] or Tarjan's algorithm[63] are popular to find *strongly connected component* (SCC) in a directed graph, as shown in the Figure 4.11. However, this analysis may not serve our purpose of finding accurate number of deadlock cycles, as there could be one or more deadlock cycles present within one SCC.

Moreover, what if there are more than one VC at one or many input ports of the network? In this case, these algorithms can give false positive, because the essential condition for a deadlock to occur is not just the cyclic dependency, but also all input ports are completely occupied and are also involved in cyclic dependency. Finding cycles in a multi-VC per input port configuration of the network makes the analysis non-trivial.

To that end DRAGON proposed to find cycles by generating another graph derived from the *initial graph* called as *DRAGON graph*. Here each node of the graph has another property called as *blocking*. A node is considered *blocking*, if all the VCs of corresponding

input ports are occupied. Consequently, *unblocking node* is the one which has one or more VCs free to occupy by the packets at upstream routers input port. A set is generated from the initial graph for all such blocking nodes. After this first pass, we remove those blocking nodes in this set which are dependent on any of the unblocking nodes.

After this second pass we are left with the set of nodes which are blocking and the corresponding nodes they are dependent on are also blocking. Now we do the reverse traversal of this graph to find the nodes from the initial graph which are dependent on any of the blocking node-set obtained after the first pass and are also blocking.

The process repeats over all the blocking nodes remaining after first pass to find the total set of blocking nodes which are dependent upon each other, in this way a graph is generated. We call this derived *directed* graph as *DRAGON graph* as shown in Figure 4.12. Once the process of generating DRAGON graph is finished number of individual cycles are counted each of which corresponds to a deadlock.

With the command line option, we can set how frequently we want to analyze the network to find deadlocks. Once the tool finds the deadlock, it can either end the simulation or allow the packets to magically spin over deadlock ring to resolve the deadlock. Latter helps in finding the frequency and likelihood of deadlocks for a given topology and traffic pattern for entire simulation, while former helps to understand how susceptible the given topology is for the routing deadlock. For example, if deadlocks occur much early during the network simulation then the network is more susceptible to deadlocks.

4.4 Putting it together: ITG, RTG and DRAGON

Figure 4.1 shows the complete flow diagram of working of ITG, RTG and DRAGON. *DRAGON-gem5* refers to the modified version of gem5 which supports DRAGON functionality. This tool is generic to find deadlocks in any irregular topology generated from ITG tool. Integration with gem5 allows to find number of deadlocks in the real application in gem5's full-system mode simulation. The results are shown in Figure 4.13.

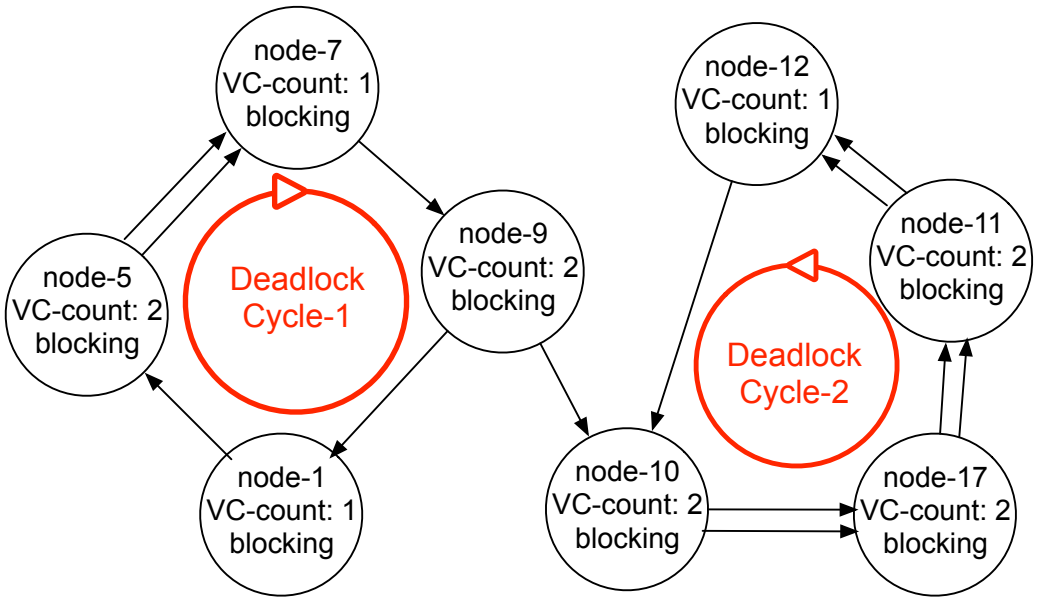


Figure 4.12: *DRAGON graph*: Here input port of the routers involved in deadlocks have different number of VCs. This figure intends to show the working of tool's concept with different network configuration. This graph is unique to each virtual network. Number of arrows coming out from each node represents the VC count of each input node.

Table 4.1: **Key Simulation Parameters.**

Real Application Simulation Parameters	
Core count	16 cores and x86 ISA (PARSEC), 1 GHz No Prefetcher
L1 Cache	Private, 32KB Instruction + 64KB Data 4-way set associative
Last Level Cache (LLC)	Shared, distributed, 2MB 8-way set associative
Network Parameters	
Topology	4x4 Mesh (PARSEC)
Routing Algorithm	Fully adaptive random
Router Latency	1-cycle
Virtual Network	6-VNets (MOESI-Hammer) 1 VC/VNet; 2 VCs/VNet
Link Bandwidth	128 bits/cycle

4.4.1 Results

The results obtained from real shared memory applications: PARSEC3.0[33] using full system simulation on gem5 on regular 4x4 Mesh topology are presented in Figure 4.13. The applications were run using MOESI_hammer [64] cache coherence protocol.

We configured the network two configurations; first configuration has one VC in each input port of the router for each message class, in second configuration there are two VCs per input port of the router. Here in order to complete the simulation of the application, we configured the tool to spin the packets over the deadlock cycle found, to resolve deadlock and allow forward progress of the simulation. This way no packets were misrouted during the simulation. Table 4.1 summarize the configuration used to obtain the results with DRAGON tool as shown in the Figure 4.13.

Here we used the metric of Deadlocks per Million Cycles (DMC) to show how many deadlocks are encountered during the simulation of the application within a million network cycles. We believe that this is a good indicator of likelihood of deadlocks for a given application and network configuration. Higher the DMC number more likely it is for the application to get involve in a routing deadlock.

We can observe number of VCs per input port has a strong effect on the number of deadlocks. Having a greater number of VCs increases the static area and power of the network by many times. In this case I have used MOESI-hammer cache coherence protocol which has 6 Virtual Networks (VNets). Therefore, the area and static power increases by *six-times* for vc-per- vnet-2 configuration compared to vc-per-vnet-1 network configuration. The tool helped in quantifying the sensitivity of deadlocks with respect to number of VCs per input port.

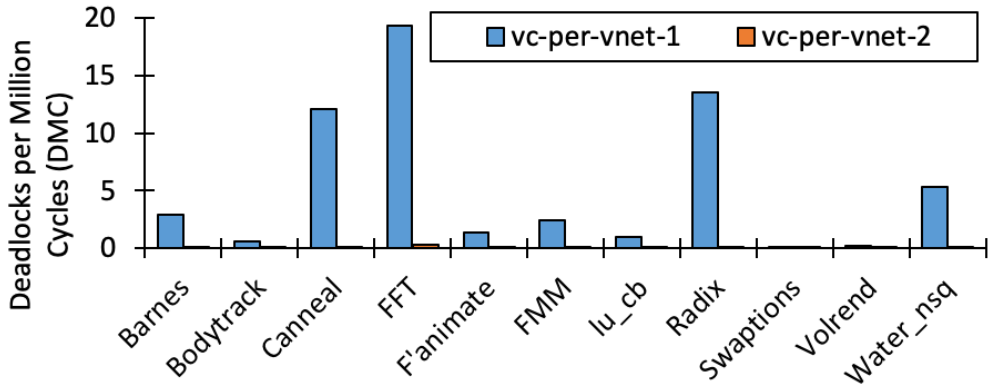


Figure 4.13: Graph shows the sensitivity of number deadlocks in real application with respect to number of VCs available per input port. 'vc-per-vnet-2' has six times buffer overhead.

4.5 Chapter Summary

In this chapter, we introduced three tools developed to facilitate the study of deadlocks in the network. In particular, tools were developed to:

- Generate random irregular topology of given size (section 4.1)
- Outputs Up*/Down* routing table for the generated irregular topology (section 4.2)
- Count number of deadlocks, its shape and size, at user provided sampling period (section 4.3)

The tool has been integrated to run with gem5[7], this allowed us to characterize deadlocks in real application traffic running with different cache coherence protocols in a full system configuration mode with OS traffic. Next we will talk about first *subactive* technique called as *Brownian Bubble Router*.

CHAPTER 5

BROWNIAN BUBBLE ROUTER (BBR)

Deadlocks are a bane for network designers, be it a Network on Chip (NoC) in a multi-core or a large scale HPC/datacenter network. A routing deadlock occurs when there is a cyclic dependence between the buffers of network routers. Most modern systems avoid deadlocks by placing routing restrictions or adding extra virtual channels, in turn hurting performance and adding overhead respectively.

In this work, we demonstrate that instead of placing such restrictions, we can, in fact, design routers to themselves guarantee deadlock-freedom, by (i) ensuring that every router always has at least one bubble (i.e., free buffer slot) at any input port, and (ii) this bubble pro-actively moves between input ports. We call this a Brownian Bubble Router (BBR). A BBR guarantees forward progress in any network topology, without requiring any routing restrictions or additional virtual channels.

BBR moves bubble within the router at a periodic rate set at design time. It is the first subactive technique to resolve routing level deadlocks. We qualitatively compare BBR with prior techniques in Table 5.1, on the metric of *full path diversity*, *Deadlock Detection needed*, *scheme misroute packets*, *does it require extra-buffers*, *provides routing deadlock freedom*, and *provides protocol deadlock freedom*. Let us study the BBR scheme in more detail.

5.1 Brownian Bubble Router

Brownian Bubble Router proposes to instrument routers with the ability of moving a *bubble* across the input ports of the router. **A bubble refers to an empty packet-sized input VC in a virtual cut through router.** This provides an opportunity to a packet that is currently part of a deadlock cycle, to move into an empty VC in the router, potentially breaking the

Table 5.1: **Summary Table of Qualitative Comparison of Deadlock Freedom Mechanisms. P: Proactive, R: Reactive. S: Subactive.**

Techniques	Full Path Diversity	No Detect deadlock	No Misroute	No Extra Buffers	Routing deadlock freedom	Protocol deadlock freedom
Dally's theory/Acyclic CDG (P) [26]	✗	✓	✓	✓	✓	✗
Duato's theory/Escape VC (P) [52]	✗*	✓	✓	✗	✓	✗
Bubble [37, 50] (P)	✓	✓	✗	✗	✓	✓$^{+}$
Deflection (P) [42]	✗**	✓	✗	✓	✓	✓
Deadlock Buffers (R) [49, 47, 65, 38]	✓	✗	✓	✗***	✓	✓***
Coordination (R) [41]	✓	✗	✓	✗****	✓	✗
BBR (S) [8]	✓	✓	✓$^{++}$	✓	✓	✗

* Within escape VCs: limited path diversity + requires topology information for escape path.
**At low-loads, full path diversity is available. But at medium-high loads, packets cannot control the directions or paths along with they are deflected.
***DISHA [47] uses timeout counters present at each input port to choose a packet to eject from the network. It requires a set of extra buffers to route the packet involved in deadlock. Some variations of DISHA, such as mDISHA [49] provide protocol-level deadlock freedom
****SPIN[41] requires a buffer in each router to hold the dynamic deadlock path over which packets involved in deadlock would move synchronously.
$^{+}$ Bubble Coloring [50] provides protocol-level deadlock freedom but involves non-minimal path traversal.
$^{++}$ BBR provides limited misrouting of packet because of *Bubble Exchange* subsection 5.3.1

deadlock cycle. We have assumed each VC to be as deep as 1-packet in this work. See section 5.4 for extensions.

At the beginning of the network run, one of the VCs (VC-0 for simplicity) at one of the input ports (excluding the injection port) of each router is tagged as the "brownian bubble

(BB)". The *invariant* of BBR is that, there will always be one BB present per router. To maintain this invariant, no packet from any other router is allowed to enter the BB. Otherwise the bubble might get consumed while the deadlock continues. At a certain user-specified frequency, the BB is moved across input ports. Moving the bubble essentially means tagging some target VC at some other input port as the BB. If the target VC is non-empty, all flits of its packet are explicitly moved into the original bubble - for e.g., for a 5-flit packet, this step takes 5 cycles. If the target VC is empty, some additional credit signals are required, as we discuss later in subsection 5.4.2.

The frequency of bubble movement (BM) is based on an epoch counter. We use *BBR-k* to refer to a BM every k cycles, where k is user-specified.

The target VC that the bubble is moved into is also a design choice. For the purpose of simplicity, in our implementation the target VC is *randomly* selected by an arbiter with higher priority being given to empty over non-empty VCs, to avoid explicit packet moves.

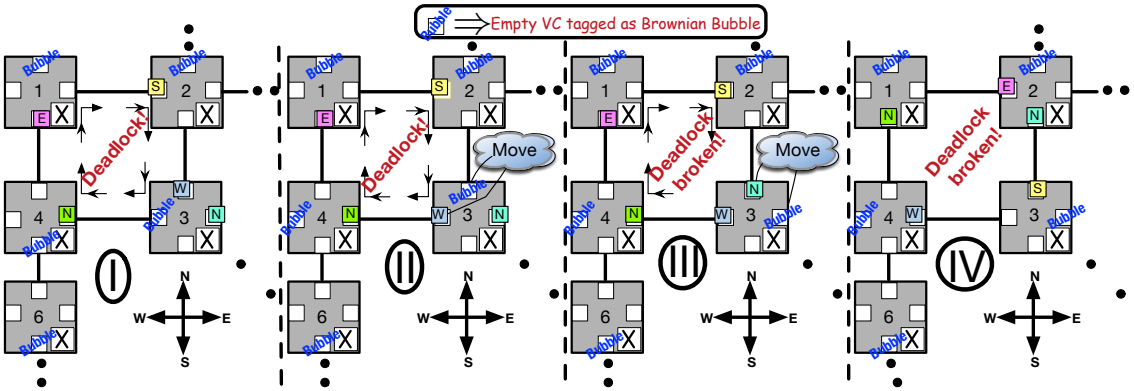


Figure 5.1: Walkthrough [Left to Right] shows how Brownian bubble movement helps in breaking deadlock cycles. It allows a deadlocked packet to move to some other port in its router, and other packets, not part of the deadlock ring, to acquire its place and eventually leave the router, thus breaking the deadlock ring. In this example, it takes two bubble movements to break the deadlock.

5.2 Key Concept

5.2.1 Walk-through Example of Bubble Movement

Figure 5.1 illustrates functioning of of BBR in detail. As shown in Figure 5.1(I), packets present in the *South, West, North* and *East* input ports of router 1, 2, 3 and 4 respectively are currently in a deadlock. The direction (N/S/E/W) written on the packets is the direction in which the packet is destined to go, in order to reach to its destination. The empty VC at the West input port is tagged as a "bubble". For the purposes of this example, consider only Router 3 (R3). In Figure 5.1(II), the packet at the North input port of R3 is moved into the bubble - thus the packet now sits in the West input port, while the VC at the North input port is now free (and tagged as the BB). As explained earlier, to maintain the invariant of keeping one BB per router, no external packet (from another router) is allowed to enter the BB. Thus, even though there is a free VC in the deadlocked loop, the deadlock persists. Next, in Figure 5.1(III), the bubble moves to the East input port, and the packet sitting there comes to the North input port.' This packet wanted to go North (to router 2) which is *unblocked* (since Router 2's South input port is free) and will thus leave the router. This is shown in Figure 5.1(IV). At this point, it will free the VC at the North input port, into which the packet sitting in Router 2's West input port can move, leading to forward progress. Figure 5.1(IV) shows the final state of the network, where the four packets originally part of the deadlock ring have made forward progress and now there is no deadlock. Note that the direction initial written on the packets still represents the direction in which the packet needs to move in order to reach to its destination.

5.3 Proof of Deadlock Freedom

Suppose there are one or more deadlocked rings going through a router.

Definition: Unblocked Packet. A packet that will eventually leave the router because of a free credit at its downstream router.

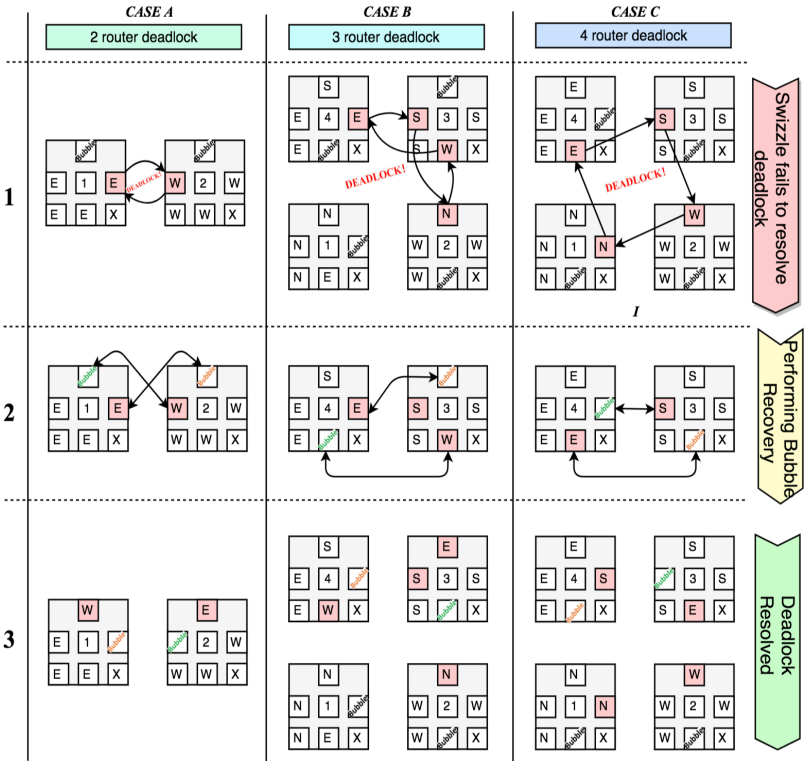


Figure 5.2: Bubble-Exchange: Deadlock corner cases can still occur with simple bubble movement technique (subsection 5.2.1). In each column, the first row shows the deadlock ring with involves 2, 3 and 4 routers respectively; the second row shows the bubble-exchange state in action, and third row finally shows the routers state after deadlock is broken.

Theorem: As long as a router has (a) at least one empty input VC (not the BB) or (b) at least one *unblocked packet*, BBR breaks any cyclic

Proof: Bubble movement can move one of the deadlocked packets into the empty VC (in case (a)), or into the VC originally occupied by the unblocked packet (in case (b)). In case (b), the unblocked packet will eventually leave the router, leaving an empty VC equivalent to case (a). The introduction of an empty VC into the deadlocked ring can guarantee forward progress, breaking the deadlock.

The walk-through example in Figure 5.1 had an unblocked packet in Router 3. However, this might not always be true. Next, we discuss how an unblocked packet can be introduced into the router, in case it does not exist, via a *Bubble Exchange*.

5.3.1 Bubble Exchange

With the current BM technique discussed so far, it is still not guaranteed that deadlock will be broken, depending on the output directions of the buffered packets. Consider the three cases present in Figure 5.2. Each column in the figure shows a deadlock scenario, bubble exchange and final state of the router after exchange. Like before, the initial of the direction present on the packet is the direction in which the packet is intended to go. Colored packets in the router are the ones which are involved in the deadlock ring, there state is show before deadlock, during bubble exchange and after deadlock in row-I, II and III respectively.

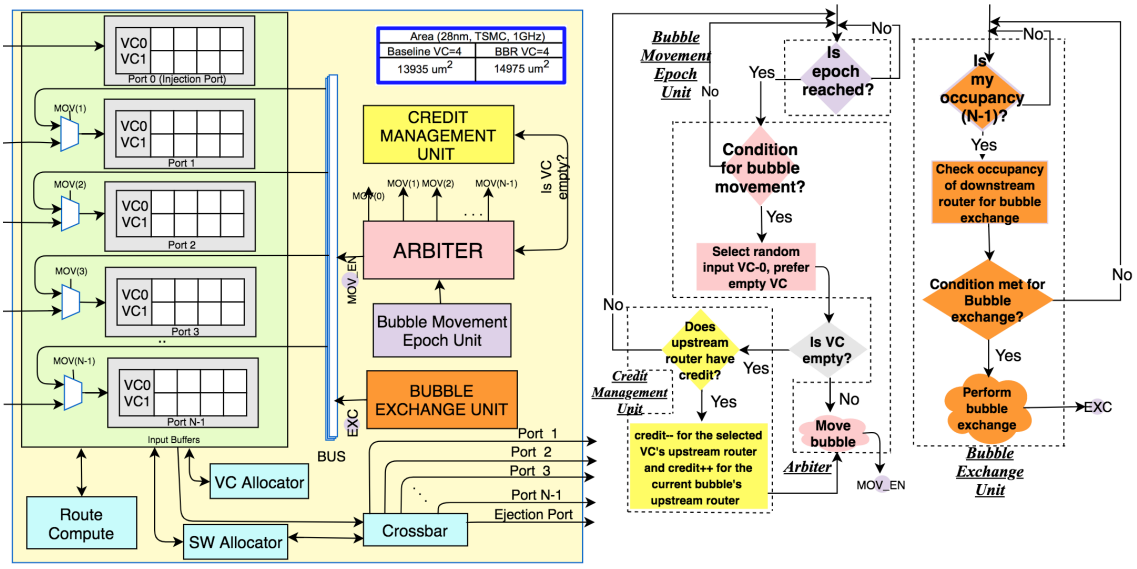


Figure 5.3: Figure showing router micro architecture on the left for Brownian Bubble Router and flow diagram illustrating the order in which Brownian Bubble Router specific actions are performed on right. Note that Brownian Bubble Router concept is generic to any underlying topology, hence number of ports are kept as N for generality of the idea. Here VC stands for *virtual channel*. Specific details about each module are discussed in section 5.4. The area consumed by the router at 28nm is also shown.

Consider Row-I. Case A shows a 2-router deadlock. Packets intending to go East are sitting in the east input port at the first router, and those going west are sitting at the West input port of the neighboring router, leading to a deadlock dependence. This would not occur in a baseline design if u-turns are not allowed. However, with bubble movement, this scenario is possible - recall that in Figure 5.1(III), the North input port of Router-3 houses

a packet that wants to go North. In Row-1, Case A, no amount of bubble movement in the two routers can resolve the deadlock, since all packets in the respective routers point to the same direction. From the necessary condition described in section 5.3, this is because of the lack of an unblocked packet in either of the routers. Row-1, Case B shows a 3-router deadlock. Here bubble movement is possible with the packets requesting different output ports. We do not enumerate all possible scenarios here in the interest of space, but all bubble movements will still end up with a 2-router or a 4-router deadlock. Finally, Row-2 Case C shows a 4-router deadlock, where no amount of bubble movement can resolve it, again due to the absence of an unblocked packet in any of the routers.

To resolve deadlocks in such scenarios, we introduce the concept of *bubble exchange (BE)*. The idea is to force forward progress of one of the packets by moving it into the bubble at its downstream router, and then recover the bubble by moving one of the packets at the downstream router into this router. BE is initiated by the upstream router when all the neighboring *downstream* routers that the packets of **this** upstream router want to get to have an *occupancy* of $N-1$, where occupancy is defined as the number of non-empty VCs in the router across all ports (except local), and

$$N = num\_inport \times num\_vcs\_per\_inport\_except\_local \tag{5.1}$$

In other words, BE is initiated when the downstream routers are completely full, except their BBs. BE takes two steps:

① Upstream router routes one of the packets to the downstream router to sit at the BB of the downstream router. This is equivalent to the upstream router consuming the BB of the downstream router. This leads to a situation where there are 2 bubbles present at upstream router and none at the downstream router, breaking the BB invariant temporarily.

② The downstream router mis-routes one of its packet to upstream router's original bubble, to recover its bubble back. The input VC of the packet chosen to mis-route is

selected at random, and becomes the BB at the downstream router.

Note that steps ① and ② described above, are performed in tandem on bi-directional link connected between the upstream and downstream routers involved in the bubble exchange.

Row-2 in Figure 5.2 shows bubble exchange in action for all three cases. The deadlocks in all cases are broken in Row 3.

Why does Bubble exchange guarantee deadlock freedom? As discussed in section 5.3, BBR works only if a router has an unblocked packet that is guaranteed to *eventually* leave. Bubble exchange forces one of the packets in the router to make forward progress towards its destination, essentially making it an unblocked packet for the purposes of the proof. In the worst case, a packet might move all the way to its destination via bubble exchange, where it will eventually get consumed[1].

Does Bubble exchange require deadlock detection? No. It is important to note that the bubble recovery algorithm is a heuristic for exchanging bubbles between neighboring routers. We do not actually detect the deadlock, thereby do not pay its associated overheads in terms of timeout counters and probes [38]. This implies that there can be false positives.

Why is Bubble exchange performed at an occupancy of N-1? In BBR, the necessary condition for a deadlock is an occupancy of (N-1), since the brownian bubble is always empty. Since explicit deadlock-detection is not performed, an occupancy of N-1 triggers a guaranteed forward movement of one of the blocked packets via bubble exchange. This guarantees that there will never be any false negatives, though there may be false positives (i.e., an occupancy of N-1 due to congestion and not a true deadlock).

Does Bubble exchange lead to mis-routing? Sometimes, but not always. Bubble exchange always leads to forward progress of at least one packet. In certain cases, the other packet might be moved to a neighbor that is not its actual preferred output port. However, in other cases, such as Figure 5.2-Case A, both packets might end up making

[1]We assume that the protocol is deadlock-free, and any packet in a router may stall but will eventually get consumed by its destination.

forward progress.

Does Bubble exchange lead to livelocks? It is theoretically possible, though extremely unlikely for the same packet to keep getting misrouted as part of the bubble exchange condition at every router it enters, never reaching its destination, leading to a livelock. Livelocks can be avoided by disallowing more than a certain number of misroutes for any packet, like prior works on mis-routing have explored [45]. We do not implement it in BBR for simplicity. Our evaluations show that the number of misroutes is actually quite low.

5.4 Implementation

Figure 5.3 shows the BBR microarchitecture. In addition to modules such as VC Allocator, Route Compute, Switch Allocator and Crossbar which have their usual function as in a baseline router, we introduce a few additional ones to implement BM and BE. We implemented the BBR modules in RTL, and observed around 7.4% area overhead (Figure 5.3) and 4.3% power overhead over a 4-VC baseline router [66] post-layout at 28nm. BBR introduces an additional mux in front of each VC, but meets timing at 1GHz like the baseline. Thus, BBR's additions are extremely light-weight.

We also show a flow-chart of the BBR operation in Figure 5.3. Each functional unit specific to BBR is color-coded with the same color as its microarchitecture counterpart. We describe key components next.

5.4.1 Bubble Movement Epoch Unit

Based on the configurable epoch parameter k, the Bubble Movement Epoch unit triggers a BM every k cycles. BM may be aborted in a special case discussed later in the credit management unit.

Bus. We add a small bus inside the router connecting the outputs of all the VCs excluding the port VCs. This is to facilitate bubble movement between two ports. We chose as bus based on the insight that at any point in time there is only one packet which would

be moved to the BB to perform BM. This implies that there will never be contention on this interconnect media, which suits the bus. Also, since we randomly move bubble across input ports by giving preference to empty input ports over non-empty input ports, a bus which connects all input ports fits our purpose. The input to the VCs can be multiplexed between the input link and the bus. This is determined by the arbiter which handles all the multiplexers using control signals (MOV(1), MOV(2), ...). There will never be any contention for the input port of a VC between the link and the bus, since the VC into which a packet is being written from the bus was the BB and will never receive a packet from the input link.

Arbiter. If a BM is triggered, the arbiter chooses the input port for BM by choosing an input VC at an input port in a round-robin manner. Priority is always given to a empty VC, if available. The bus-arbiter unit sends out two signals, the MOV_EN and the MOV signals. The MOV_EN signal is sent to the bus to indicate that a BM is impending in the next cycle while the MOV signal is used to select the port to where this movement will happen (in other words, the port from where a packet will be read out and put on the bus to be inserted into the current BB). This signal remains active till all flits of the packet have been moved.

5.4.2 Credit Management Unit

Credit management is an integral part of the BBR. As mentioned earlier, no packet is allowed to come and occupy the VC tagged as the brownian bubble. This is ensured by *not* sending a credit for this VC to the upstream router so that it believes that this VC is actually occupied. Thus, a BB simply looks like a full VC to the upstream router. During BM, two cases arise.

Case I: The bubble is moved to a full VC (i.e., the packet from the full VC is moved into the bubble). In this case, no credits need to be sent to the respective upstream routers. This is because both upstream routers connected to the downstream router believe that the VCs are full - it is agnostic to the fact that one has an actual packet, one is the BB, and the

bubble moved between them.

Case II: The bubble is moved to an empty VC. The VC that becomes the BB needs to send a *decrement credit* signal to the upstream router to inform it that this VC is actually not empty. The original VC which was the BB needs to send an *increment credit* signal to the upstream router signaling that this VC is now free. This is done after one cycle of delay to manage a corner case where the upstream router may have already started sending flits for a new packet into this VC, which is currently on the link. This is handled by *aborting the bubble movement* as follows: (a) if the upstream router receives a decrement credit for a VC that it has already started sending flits to, it ignores the decrement credit, (b) if the downstream router receives flits into a VC that became the BB in the previous cycle, the original (empty) VC is tagged as the BB again. (c) the original VC sends its increment credit signal after waiting for a cycle only if the above scenario does not occur.

5.4.3 Bubble Exchange Unit

Each router keeps track of its *occupancy*, which was defined earlier in subsection 5.3.1. If the occupancy of the router reaches (N-1), where N is the total number of input VCs at all ports of the router (except local), it collects the occupancy of its neighbors. If the neighbors have an occupancy greater than a certain threshold (max threshold is N-1), BE is triggered by setting the EXC flag. This reads one of the packets from the router and sends it out of the crossbar and output link to the neighboring router. The neighbor in turn sends a packet to this router which is added into this router's BB. A subtle point to note is that BE does not steal any useful link bandwidth since the occupancy of (N-1) at both routers means that they were unable to send packets to each other via regular switch allocation due to the lack of credits, and so the links between them were anyway idle.

BE is a measure of how reactive the router is towards recovering from the deadlock by exchanging the bubble. The occupancy metric we use to trigger BE is just a heuristic. From a correctness point of view, BE can be triggered more pro-actively or at a fixed time epoch

Table 5.2: **Network Configuration.**

Network	
Topology	8x8 Mesh
Router latency	1-cycle
Num VCs	1, 2, 3, 4
Buffer Organization	Virtual Cut Through Single packet per virtual channel
Target Networks	
Deadlock Avoidance	West-first and Escape VC
Deadlock Recovery	Static Bubble [38] and SPIN [41]
Brownian Bubble Router	BBR-k (k= BM frequency)

in alternate implementations.

5.5 Adding BBR over Alternate Router Microarchitectures

BBR's underlying mechanism of periodic bubble movement across input ports *within* a router, followed by occupancy-driven bubble exchanges between *neighboring* routers, can be applied to any input buffered VC router to guarantee deadlock freedom in the network, as we showed in section 5.3. This makes it agnostic to the underlying topology (mesh/high-radix/irregular/reconfigurable [67]), routing algorithm (XY/adaptive [68]) and router bypass optimizations [66]. BBR can also work with wormhole routers, but will require additional complexity (such as packet truncation [42]) to manage ordering since parts of the same packet might end up at different input ports of the same router due to BM.

5.6 Evaluation

5.6.1 Methodology

We model Brownian Bubble Router in the Garnet [59] cycle-accurate NoC simulator. For BBR, we implement a minimal *fully adaptive random routing* algorithm. We use credits at the downstream router to decide the direction if more than one choice exists.

Table 5.2 lists the system configurations we evaluated. We contrast BBR against both

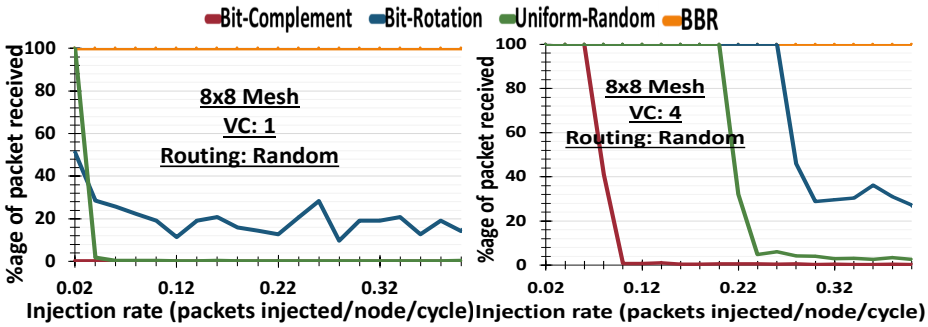


Figure 5.4: Correctness of Brownian Bubble Router. For a fixed number of packets for the simulation, x-axis shows total packets injected in network per node per cycle and y axis shows %age of total packets received at the end of simulation.

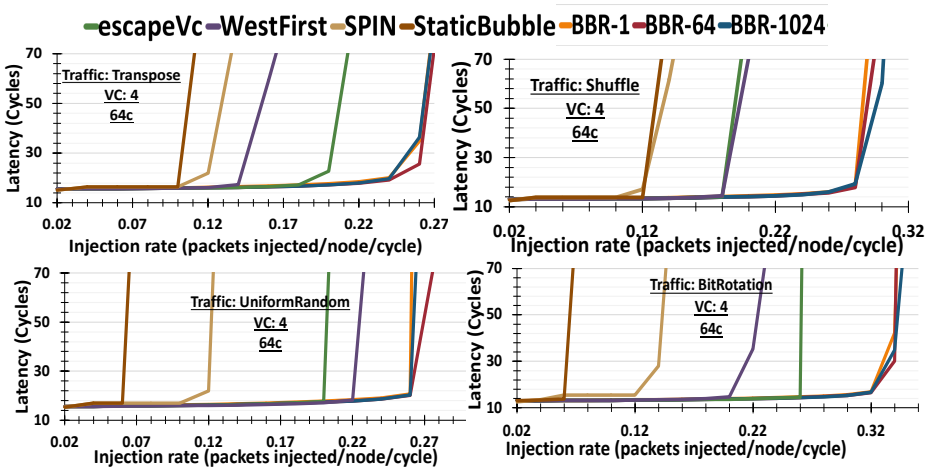
classic deadlock-avoidance (West-first and escape VC) and state-of-the-art deadlock-recovery (Static Bubble [38] and SPIN [41]) schemes. All networks use a single-cycle router.

Recall that the rate of bubble movement (BM) is a knob given to the network designer to tune the frequency of bubble movement within the router.

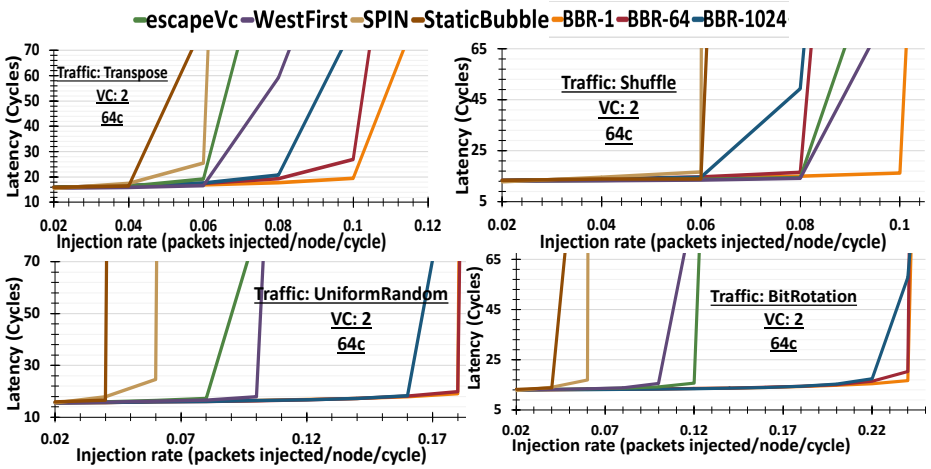
We evaluate BBR with multiple BM epoch values and report results with 1 (i.e., every cycle), 64 (every 64 cycles) and 1024 (every 1024 cycles). For BE, we empirically set the occupancy threshold at downstream routers to 4.

5.6.2 Correctness

The primary claim of Brownian Bubble Router is to make sure there is no deadlock that persists in the network. We show this in Figure 5.4 for a 8×8 mesh topology. On y-axis we plot the percentage of packets received over fixed packets injected in the network. X-axis shows the injection rate at which these packets are injected in the network. All traffic patterns use fully random routing. Figure 5.4 also shows how sensitive the network is towards the number of VCs present per input port in the router. We see that network deadlocks at much lower injection rate when number of VCs is 1 compared to when it is 4. This is especially stark in bit complement traffic which deadlocks almost immediately in a 1 VC design. BBR performs consistently by delivering 100% packets that are injected in



(a) VC per VNet = 4; 8x8 Mesh



(b) VC per VNet = 2; 8x8 Mesh

Figure 5.5: Performance of Brownian Bubble Router technique compared against recently proposed deadlock recovery schemes and well known deadlock avoidance schemes such as escapeVC and WestFirst Routing, proving its superiority. Here x-axis shows total packets injected in network per node per cycle and y-axis shows the average latency incurred by packets in cycles.

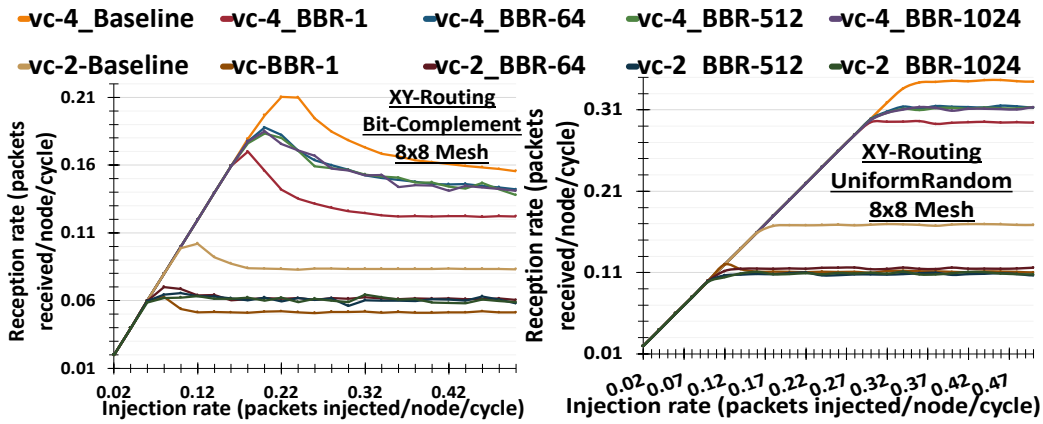


Figure 5.6: Overhead introduced when adding BBR over a baseline deadlock-free XY routing algorithm.

the network at all injection rate for all traffic patterns.

5.6.3 Performance

Next, in Figure 5.5, we evaluate BBR against state-of-the-art deadlock freedom techniques for a 8×8 mesh at different VC counts.

In the interest of space, we only present results for transpose, shuffle, uniform-random and bit rotation traffic pattern for 2 and 4 VCs respectively. With 4 VCs, we observe 37% throughput improvement over WestFirst and escapeVC (deadlock avoidance) on average and $3\times$ improvement over Static Bubble and SPIN (deadlock recovery). With 2 VCs, we observe 44% improvement over WestFirst and escape VC, and $.2.5\times$ improvement over Static Bubble and SPIN. For many of the patterns, the performance improvements are higher at 2 VCs as opposed to 4 because of the path diversity provided by fully adaptive routing enabled by BBR in all VCs. In contrast, West-first lacks path diversity in the west direction, while escape VC only provides full path diversity within one of its VCs (the other one restricted to west-first).

With 1 VC, however, we found the latency with BBR to be erratic at a few injection rates. This is because the input port where the bubble resides essentially gets blocked for the upstream router for a period of time, leading to uneven and unpredictable delays until

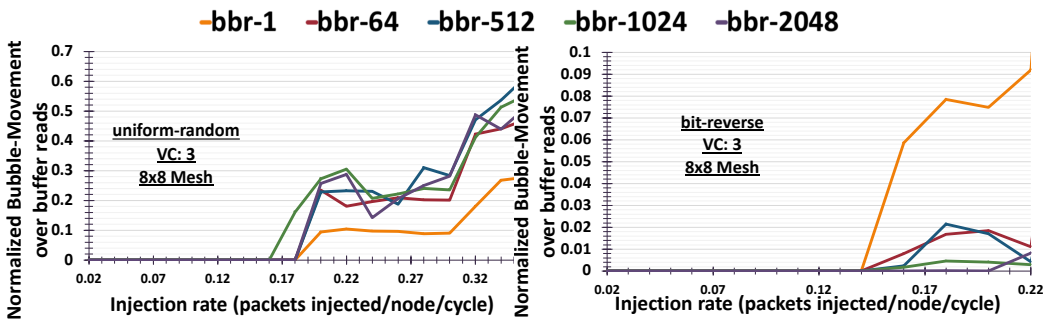


Figure 5.7: Bubble Movement Frequency: y-axis shows ratio of buffer reads (or writes) due to BM over the baseline buffer reads (or writes). BBR-1 shows the highest BM for bit-reverse compared to other BBR-k; this behavior is opposite in uniform random traffic. This shows distribution of BM across BBR-k is highly traffic dependent.

packet reaches its destination. However, it is important to note that a 1 VC BBR design is deadlock-free, as we showed earlier in Figure 5.4.

We also performed an experiment to understand the performance overhead BBR adds on top of an already deadlock-free routing algorithm, such as XY. In Figure 5.6, we plot the reception rate for a baseline XY scheme and various BBR schemes with 2 and 4 VCs. We notice that an aggressive BBR-1 (that tries to perform bubble movement every cycle) leads to a 25% drop in throughput for uniform random and 35% drop in throughput for bit complement averaged over all VC count. But with higher values of the epoch, the drop is only 9%. Recall that at high loads, BE kicks in once the upstream and downstream routers start becoming full, which leads to a performance differential at high loads, even if the BM epoch is set very high. If we restrict BE to occur on a very high fixed threshold, rather than based on occupancy, BBR would have essentially no overhead if the underlying algorithm is inherently deadlock free.

5.6.4 Bubble Movement and Bubble Exchange Frequency

Having shown the correctness and performance benefits of BBR, next we study the potential overhead. As discussed earlier in section 5.4, the area overhead for implementing BBR is negligible. However, each bubble movement involves reading a packet out of its current VC and writing it into an empty VC, increasing the total buffer activity. The same occurs

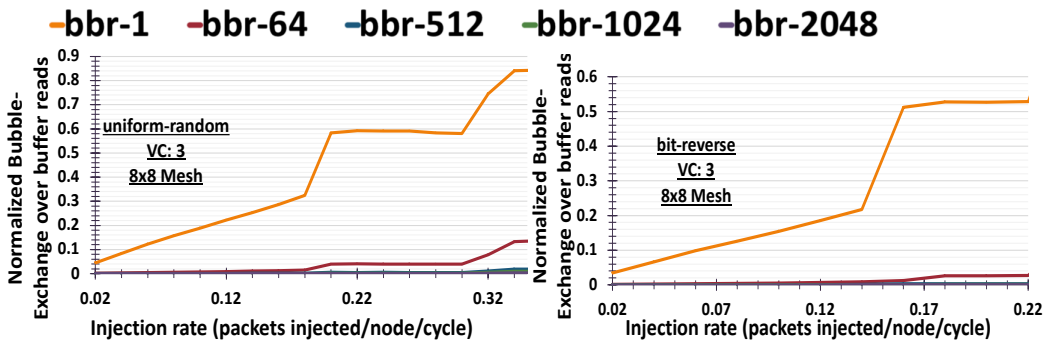


Figure 5.8: Bubble-Exchange Frequency: here y-axis shows ratio of buffer reads (or writes) due to BEs over the baseline buffer reads (or writes) and x-axis shows the packets injected in the network per node per cycle. We see that BBR-1 has highest BE over any other BBR-k.

in during a bubble exchange as well, across neighboring routers. This can naturally have energy implications.

We quantify this overhead in the next set of experiments. In Figure 5.7 and Figure 5.8 we plot the ratio of additional buffer reads (or writes) due to BM and BE respectively over the baseline buffer reads (or writes) for various values of BBR-k, where k is the frequency of BM. In the interest of space, we plot the behavior for two patterns - uniform-random and bit-reverse which show contrasting characteristics. Other traffic patterns showed similar behavior to one of these patterns. Note that a BM between empty VCs does not count as a buffer read/write.

In Figure 5.7, we highlight that there is *no bubble movement* up to a certain injection rate; this is because at low loads, more than two VCs are empty across all ports of the router in most cases, so a BM does not need an explicit packet read and write. This shows that BM actually adds no energy overhead, especially at low to medium loads which is the common operating point for most NoCs. At high injection rates, the network is more susceptible to deadlock, and naturally the number of BMs go up as well. One might expect low values of k (i.e., high frequency of BM) to lead to higher number of buffer reads/writes. This can be seen as true in bit rotation, where BBR-1 shows up to 4x more buffer reads/writes than BBR-64 post saturation. Counter intuitively, though, the opposite is seen in uniform ran-

dom traffic, where a higher value of k actually end up leading to more bubble movements overall. This is because with a low frequency of BM, deadlocks persist for longer and end up requiring more BMs to provide forward progress. This shows a subtle yet important feature of BBR that it is adaptive; hence the energy consumption will be more only if the traffic is susceptible to deadlock.

In Figure 5.8, we see that BBR-64 and beyond, the number of BEs is negligible. A notable exception is BBR-1 (moving bubble every cycle) which shows the highest number of BEs over any other BBR-k (lower frequency of bubble movement). This can be understood as follows - high BM ends up moving packets to other free VCs in the router very frequently, leading to more occupancy within the router. This in turn triggers BE more frequently. In contrast, at low BM frequency, the router tends to remain emptier (especially at low loads), leading to fewer forward movements due to BE.

In summary, the impact of the BM frequency depends heavily the traffic pattern and injection rate. For uniform random traffic, we observe that BE, not BM actually tends to dominate. Moreover, the energy overhead can be controlled via various knobs exposed to the designers such as the BM frequency and BE occupancy threshold.

5.6.5 BBR for Irregular Topologies

Next, we study BBR performance with an irregular topology as shown in left most subfigure in Figure 5.9. Here the link between routers 5 and 6 in a mesh is broken, which could be due to various reasons such as power gating or dynamic faults in the network [69]. A challenge with irregular topologies is that traditional turn-restrictions (such as XY) will not longer work - for e.g., any packet from 5 to $\{10, 11, 14, 15\}$ will have to make a Y to X turn at 9. Similarly, a Y to X turn at 2 will have to be allowed. Such turns could lead to a $5 \rightarrow 9 \rightarrow 10 \rightarrow 6 \rightarrow 2 \rightarrow 1 \rightarrow 5$ deadlock. In such scenarios, the deadlock free routing option is to construct a *spanning tree* [69, 70, 71, 72], which will not allow the use of the link between router 1 and 2 (highlighted in grey) to avoid cycles. BBR does not need any

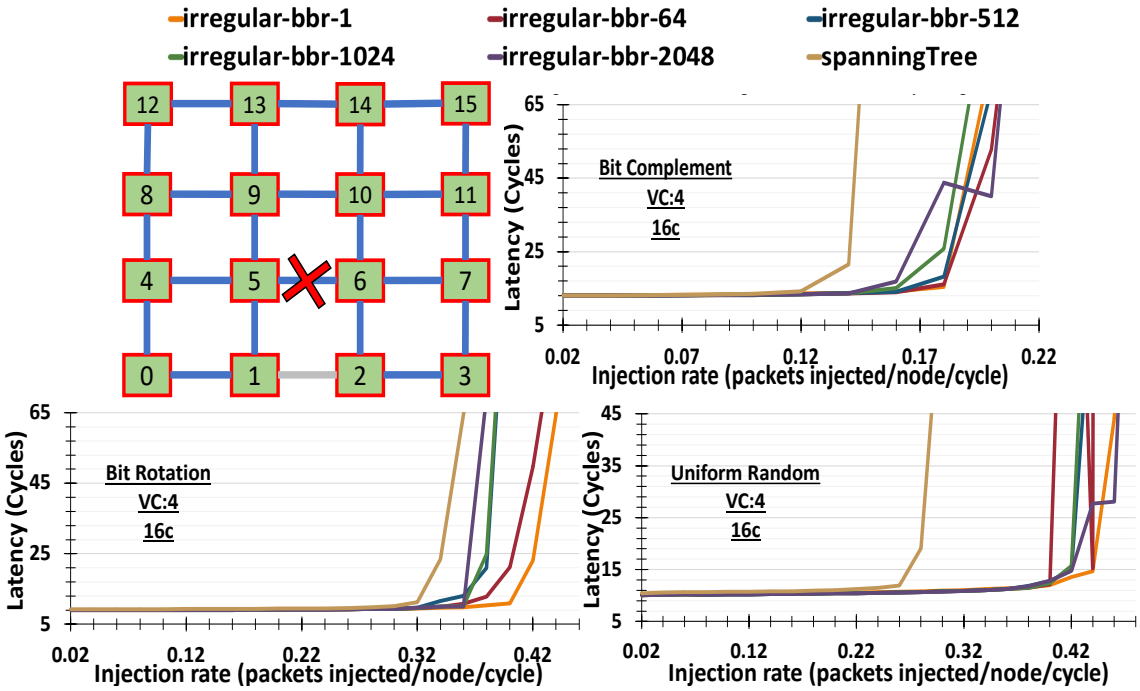


Figure 5.9: A 4x4 Mesh with a faulty link (shown with X). XY routing can no longer work. Traditional deadlock avoidance (Spanning Tree) will disable the use of the grey link to avoid cycles, leading to non-minimal routes. Thus BBR provides higher saturation throughput.

such restrictions and enjoys the full path diversity. This translates to around 40% higher throughput on average over the spanning tree routing algorithm as shown in Figure 5.9.

5.7 Discussion

BBR is a subactive technique because unlike proactive techniques it does not reserve hardware resources to make sure deadlock does not occur to begin with, nor it detects deadlock in the network at runtime and resolve it, instead it periodically moves packets within the router to make sure if there is any deadlock present in the network it gets resolved. BBR uses *bubble* to shuffle packets within the router. This allows the blocked packet to free up the input port which was earlier involved in the deadlock. Having bubble however puts restrictions on thenumber of packets that can use buffers in the network, consequently it lowers the throughput of the router. One extension to BBR is to shuffle the packets within the routers input port, which is occupying the head of the queue. Instead of using *bub-*

ble to shuffle packets within the network. Moreover, instead of doing oblivious shuffling, which could increase the energy consumption, we can shuffle packets based on a threshold number of cycles. If the packet is stuck for a threshold number of cycles, then it becomes a candidate to be replaced from its current inport and occupy inport which has either free buffers available or there is *unblocked* packet present in that input port.

5.7.1 Improving BBR using CDG information of the topology

BBR proposal moves packet obliviously within router. Channel Dependency Graph (CDG) gives an important indication of the routing deadlock cycle that can form within the topology. In CDG each link of the original topology is a node and edge in CDG represents the turn/direction in which a packet can move as allowed by the routing algorithm. CDG is a static information and it can be used tochoose the input port and unblock packet, present in the router to replace the current blocked packet to resolve the deadlock.

5.7.2 Extending BBR for protocol deadlock freedom

BBR in its present form provides routing deadlock freedom. One natural extension of BBR could be to provide protocol-deadlock freedom. Routing deadlock freedom ensures that there is no cyclic dependency of packets in the network. With BBR, if we can prioritize the movement of response packet/terminal message class packet over request (or Ack)/non-terminal class packet, then we can ensure protocol level freedom. It will be an interesting case-study to ensure how packet prioritization logic works in tandem with packet-shuffling algorithm of BBR.

5.8 Chapter Summary

BBR [8] was the first work to introduce the routing deadlock freedom by freeing-up the input port involved in the routing-deadlock. This was achieved by obliviously shuffling packets among the routers' input-ports such that the deadlocked packet occupying the dead-

locked/blocked input port moves to other input port. This packet-shuffling, in turn, would free the original deadlocked input port, allowing packet involved in the deadlock at the upstream router to make forward progress. BBR is the first scheme to provide subactive routing deadlock freedom. We qualitatively compare BBR with prior work in Table 5.1.

In the next chapter, we will see how we can take the idea of BBR and apply it to the whole network to provide not only routing deadlock freedom but protocol deadlock freedom. This leads us to our next work in the domain of subactive deadlock freedom called BINDU[9].

CHAPTER 6

BUBBLE IN IRREGULAR NETWORK FOR DEADLOCK PURGING (BINDU)

Every interconnection network must ensure, for its functional correctness, that it is deadlock free. A routing deadlock occurs when there is a cyclic dependency of packets when acquiring the buffers of the routers.

Prior solutions have provisioned an extra set of *escape* buffers to resolve deadlocks or restrict the path that a packet can take in the network by disallowing certain turns. This either pays higher power/area overhead or impacts performance. In this work, we demonstrate that (i) keeping one virtual-channel in the entire network (called 'Bindu') empty, and (ii) forcing it to move through all input ports of every router in the network via a pre-defined path, can guarantee deadlock-freedom. We show that our scheme (a) is topology agnostic (we evaluate it on multiple topologies, both regular and irregular), (b) does not impose any turn restrictions on packets, (c) does not require an extra set of escape buffers, and (d) is free from the complex circuitry for detecting and recovering from deadlocks.

We delineate BINDU and BBR while comparing with prior *Proactive* and *Reactive* deadlock freedom techniques in Table 6.1. Unlike BBR, BINDU does more frequent misrouting. However, it does not require bubble in each router of the network. Let us study this subactive scheme in more details.

6.1 BINDU

In this work, called BINDU (Bubble in Irregular Network for Deadlock pUrging), we demonstrate, for the first time, that it is possible to provide deadlock freedom with just a *single* bubble (referred to as a 'Bindu'[1] and defined formally in subsection 6.1.1) in the

[1]Bindu is a Hindi word, meaning 'point'. With a collection of points, we can draw any geometrical figure; similarly with BINDU we can make any topology deadlock free.

Table 6.1: **Summary Table of Qualitative Comparison of Deadlock Freedom Mechanisms. P: Proactive, R: Reactive. S: Subactive.**

Techniques	Full Path Diversity	No Detect deadlock	No Misroute	No Extra Buffers	Routing deadlock freedom	Protocol deadlock freedom
Dally's theory/Acyclic CDG (P) [26]	✗	✓	✓	✓	✓	✗
Duato's theory/Escape VC (P) [52]	✗*	✓	✓	✗	✓	✗
Bubble [37, 50] (P)	✓	✓	✗	✗	✓	✓$^{+}$
Deflection (P) [42]	✗**	✓	✗	✓	✓	✓
Deadlock Buffers (R) [49, 47, 65, 38]	✓	✗	✓	✗***	✓	✓***
Coordination (R) [41]	✓	✗	✓	✗****	✓	✗
BBR (S) [8]	✓	✓	✓$^{++}$	✓	✓	✗
BINDU (S) [9]	✓	✓	✗	✓	✓	✗

* Within escape VCs: limited path diversity + requires topology information for escape path.

**At low-loads, full path diversity is available. But at medium-high loads, packets cannot control the directions or paths along with they are deflected.

***DISHA [47] uses timeout counters present at each input port to choose a packet to eject from the network. It requires a set of extra buffers to route the packet involved in deadlock. Some variations of DISHA, such as mDISHA [49] provide protocol-level deadlock freedom

****SPIN[41] requires a buffer in each router to hold the dynamic deadlock path over which packets involved in deadlock would move synchronously.

$^{+}$ Bubble Coloring [50] provides protocol-level deadlock freedom but involves non-minimal path traversal.

$^{++}$ BBR provides limited misrouting of packet because of *Bubble Exchange* subsection 5.3.1

entire network. The Bindu moves through the network in a fixed path, covering all the routers and their input ports in the network. During its course the Bindu shuffles the packets present in the network, naturally resolving any deadlock that comes in its path.

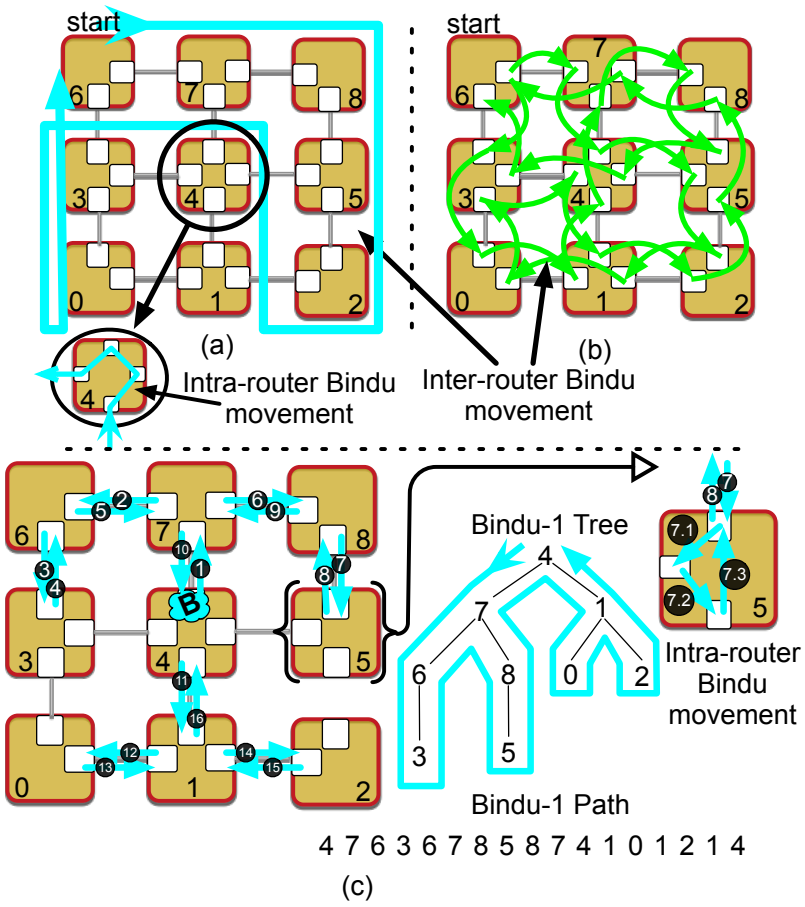


Figure 6.1: Examples of Bindu-paths. Each Bindu must go through all input ports of all routers of the network, at least once. (a) Bindu moving through all ports of a router before jumping to the next router, (b) Bindu jumping between input ports of different routers throughout its path, (c) A tree-based Bindu-path for an irregular topology

The following are the primary contributions of this work:

1. A novel technique to guarantee deadlock freedom in arbitrary irregular topologies by having *one* or more Bindus (empty VC) in the entire network, and to force these Bindus to move through-out the network at a periodic rate.

2. BINDU frees the designer from any consideration of deadlock when designing their routing algorithm, allowing high performance with minimal overhead.

3. And evaluation of how BINDU performance compares with previously proposed deadlock freedom techniques with both synthetic traffic and real applications on both

regular and irregular topologies[2].

6.1.1 Definitions

We formally define terms here that we will use throughout the paper.

Bindu: In BINDU, *Bindu* refers to a reserved packet-sized *empty VC*, that is instantiated at the starting of the network run. It makes pro-active movement throughout the network covering all the input ports of every router in the network, as shown in Figure 6.1. Unlike 'Bubble' used in previous works [37, 39], no packet can sit inside the Bindu. As a Bindu proactively moves through the network, packets get displaced, as shown in the walk-through Figure 6.2.

k-Bindu: BINDU networks can incorporate multiple Bindus within the network, each following its own path as shown in Figure 6.1(c) and Figure 6.2. We refer to this configuration as 'k-Bindu', where 'k' is the number of Bindus instantiated at the starting of network run.

Bindu movement: Movement of Bindu from one VC to another within the router or across routers.

Moving a Bindu from <Router_i, Port_m, VC_k> to <Router_i+1 Port_n, VC_o>, effectively means reading the packet from <..., VC_o> and writing it to <..., VC_k>[3].

Blocked Packet: A packet which is *indefinitely stuck* because it is part of a deadlock ring.

Unblocked Packet: A packet which might be *temporarily stalled*, because of congestion in the network or unavailability of credits at the downstream router. However, it is *guaranteed to eventually leave the router*. **Empty slot:** An empty VC in a router.

[2]We would like to note that the focus of this work is not on the important problem of dynamic fault-tolerance; rather it guarantees deadlock freedom in static irregular topologies, which may be created due to faults or at design time.

[3]We assume Virtual Cut-Through, i.e., all VCs that the Bindu traverses are sized to hold the largest packet. Wormhole designs will be discussed in section 6.4

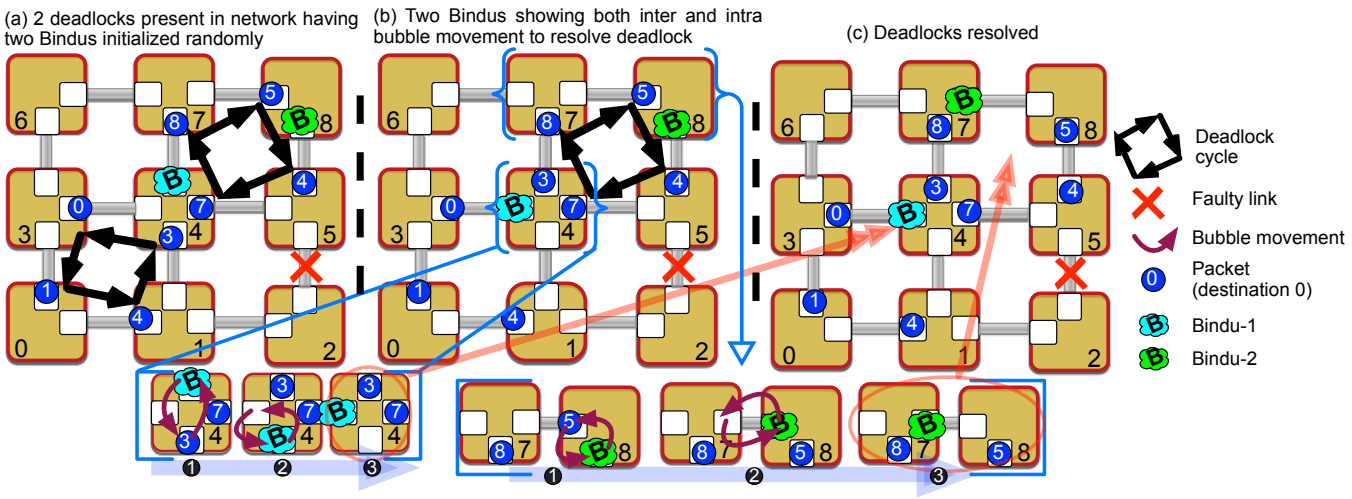


Figure 6.2: Walkthrough figure showing the BINDU in action. Here deadlock involving router-0,1,3 and 4 is resolved by intra-router Bindu movement of Bindu-1 and deadlock involving router-4, 5, 7 and 8 is resolved by inter-router Bindu movement of Bindu-2. Network state corresponding to each type of Bindu movement is shown in sub-figure (b) and (c) respectively.

6.1.2 Basic Idea and Walk-through Example

Deadlocks are characterized by cyclic dependency of packets, which renders the forward movement impossible. BINDU tries to resolve this cyclic dependency using one or more Bindus in the network. A Bindu can be randomly initialized at one of input VCs of any router, barring the injection input VCs. Multiple Bindus can co-exist within a router - but cannot reserve the same VC at the same time.

6.2 BINDU Network

Each Bindu pro-actively moves in a predefined path which cycles back from where it started (Figure 6.1). Bindu's movement results in the partial shuffling of packets in the network. This results in naturally resolving any deadlock cycle which comes in Bindu's path. To understand the BINDU technique in more detail, let us walk-through the scheme with an example. Figure 6.2(a) shows two deadlocks in a 3x3 Mesh; the link between router-2 and router-5 is faulty, resulting in an irregular topology. The number on the packet refers to its destination router-id.

Even though one Bindu is enough to resolve the deadlock, we show two Bindus in the walk-through figure to underline the generality of the scheme.

In Figure 6.2(b), we focus on the deadlock between packets in Routers-0, 1, 3 and 4. Bindu-1 moves within Router-4 from the North to the South to the West port. This results in an empty slot in the South port, which can now be used by the packet stuck in Router-1 to make forward progress, resolving the deadlock.

In Figure 6.2(c), we focus on the deadlock between packets in Routers-4, 5, 7 and 8. Bindu-2 jumps from Router-8 to Router-7. An important point to observe is that Bindu-2 first traverses within Router-8 from South to West (similar to Bindu-1), and then jumps to the connected Router-7 to its East port. The deadlock is resolved as Bindu replaces an earlier blocked (i.e., deadlocked) packet at West input port of Router-8 with an empty VC.

All Bindus need to traverse through all ports of all routers, as we discuss in subsection 6.2.1. The implications of Bindu's intra and inter-router movements on the router micro-architecture are discussed in section 6.4.

6.2.1 Bindu Path

All Bindus move through *all the input ports of every router* in the topology (both regular and irregular) at a periodic rate indefinitely. There can be multiple possible paths. Figure 6.1 shows three possible paths for Bindus in a 3×3 Mesh. In Figure 6.1(a), each Bindu snakes through the network routers, moving through all input ports at each router; in Figure 6.1(b), Bindus jump between input ports of different routers in each step; Figure 6.1(c), the Bindu uses a tree to loop through the entire network as it is an irregular topology where the original snake does not work. As mentioned earlier, there can be more than one Bindu in the network, and each Bindu can choose different paths. The Bindu-path is encoded within each router; i.e., each router has a preset order of input ports through which the Bindu should move, and a preset neighbor to which the Bindu should jump. The next inport-id (intra-router) or router-id (inter-router) where Bindu needs to move is provided

by the current router. A router can have multiple paths encoded, indexed by the Bindu id. We discuss static vs. dynamic configurability of Bindu paths in section 6.4.

6.2.2 Bindu Movement

Moving a Bindu from VC-A to VC-B requires explicitly moving the packet from VC-B to VC-A. VC-B could be a VC within the router (during intra-router Bindu movement), or at a neighboring router (during inter-router Bindu movement). Moving a Bindu across routers can lead to a temporary misroute of the packet.

In our design, we constrain Bindu to move only across VC-0 within the input ports of all routers in the network during both intra- and inter-router Bindu movement. This decision simplifies the router micro-architecture for BINDU (section 6.4). VC-0 is Virtual-Cut Through. A Bindu movement takes f cycles, where f is the number of flits in the packet. Naturally, the Bindu movement period p needs to be greater than the size of the largest packet that can sit in VC-0.

6.3 Proof of Deadlock freedom

In this section, we explain the proof of BINDU technique for deadlock freedom using Figure 6.3 as reference. We refer to the terms defined in subsection 6.1.1 and provide the following arguments.

1. By the virtue of the Bindu's looped path, it is guaranteed to visit every deadlock ring in the network.

2. As Bindu moves into the deadlock ring and then moves out of deadlock ring it either brings a fresh packet or an empty slot into the deadlock ring.

3. If Bindu brings in the empty slot at its place (Figure 6.3-(a)), then, by default, it resolves the deadlock as the earlier deadlocked packet can take up that empty slot breaking the deadlock dependency.

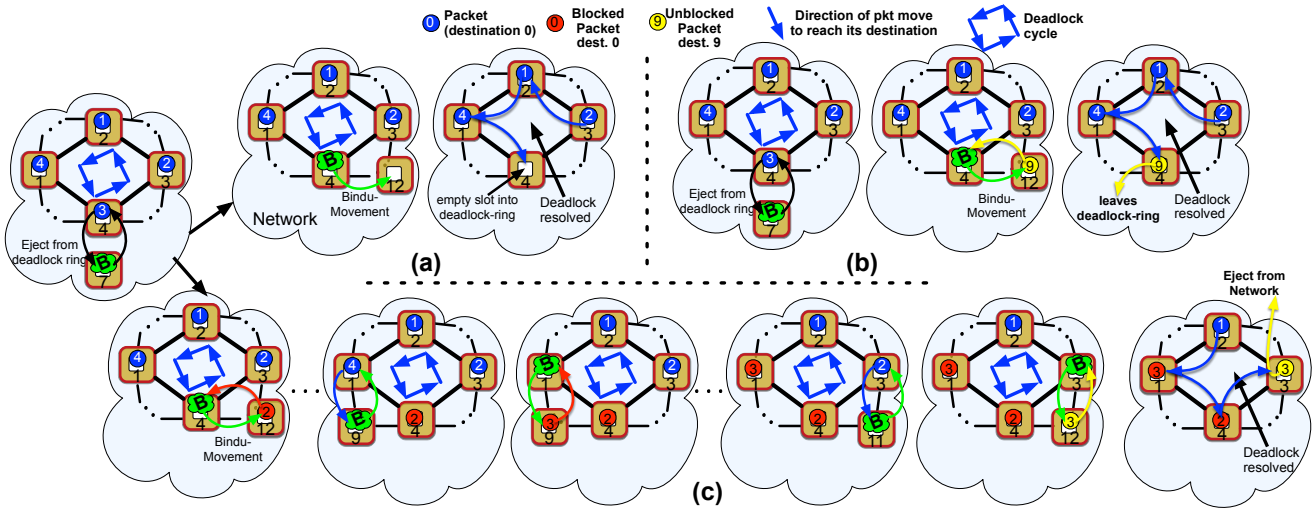


Figure 6.3: The figure shows: (a) The way Bindu resolves the deadlock when it brings an empty slot to the deadlock ring. (b) How Bindu resolves the deadlock when it brings a unblocked packet to the deadlock ring. (c) Bindu resolves the deadlock by shuffling the packets present within the deadlock ring. The number inside the packet refers to its destination.

4. If Bindu brings in a fresh packet to the deadlock ring, then there could be two possibilities. This fresh packet can either be 'unblocked' or 'blocked', as defined in subsection 6.1.1.

5. If the fresh packet is unblocked (Figure 6.3-(b)), it will naturally leave the deadlock ring; this will create an empty slot in the deadlock ring, and deadlock will naturally get resolved as described in point 3.

6. However, if the fresh packet is blocked, then deadlock would persist until the next Bindu movement into the deadlock ring. If the next Bindu movement creates the empty slot or brings in unblocked fresh packet to deadlock ring, then deadlock will get resolved as mentioned in point 3 and point 5.

7. There could be a very rare, corner case in which Bindu movement will keep bringing a blocked packet to the deadlock ring (Figure 6.3-(c)). This, in fact, means that the packets being brought into the deadlock ring by Bindu are part of the same deadlock ring. Here, Bindu movement effectively shuffles the packets within the deadlock ring.

This shuffling of packets within the deadlock ring ensures that eventually, at least one packet would reach to its destination because of shuffling and eject-out of the network as shown in Figure 6.3-(c). This would finally lead to deadlock resolution.

8. A final pathological corner case could be when all packets of the network are in one big deadlock loop (e.g., this could occur in a ring topology). In such a scenario, the movement of packets due to Bindu will continue to remain in a cyclic loop. However, once a Bindu completes a full looped path, all packets would have effectively been spun around, as described in [41]. Eventually after 'k' spins one of the packets would reach its destination and eject out from the network. This would again lead to deadlock resolution.

It is worth noting that we did not actually observe either (7) or (8) in our extensive experiments.

Livelocks: Livelock is the condition where a packet keeps moving indefinitely but *never* reaches its destination. Recall that packets get misrouted by one-hop due to a Bindu movement. As long as the packet makes two forward hops before a Bindu misroutes it again, it will not livelock. Since Bindu paths are fixed, for the Bindu to arrive at the same router again, it will take $N \times r \times p$ cycles, where 'N' is the number of nodes, 'r' is the router radix and 'p' is the Bindu movement period. During this time, if the network is not congested, a packet will definitely move forward two hops (even if N and r are small, p can be set to a large enough value to ensure two hops). If the end points are congested, it is theoretically possible, though extremely unlikely, for the same packet to be stuck at a router till the Bindu traverses the entire network and returns, and get misrouted again. However, as long as the network is deadlock-free (proven above), it cannot be congested indefinitely, thereby ensuring that eventually the packet will move forward two hops, and thus not livelock.

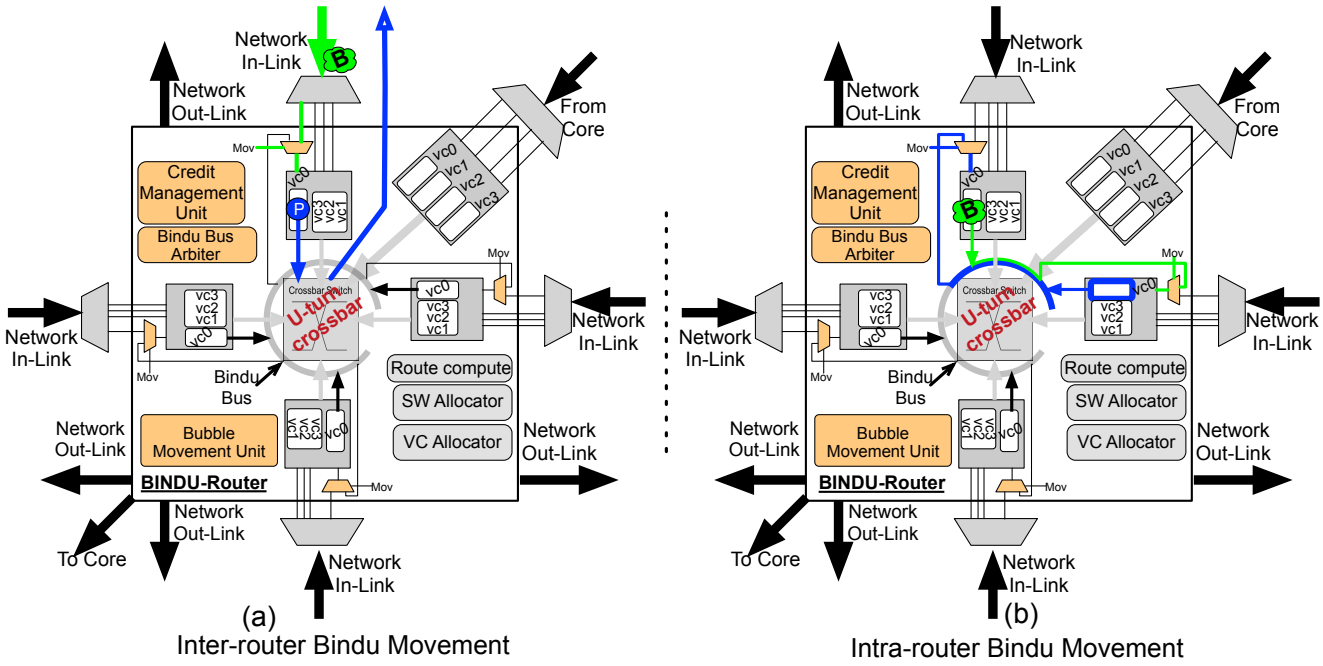


Figure 6.4: Router micro-architecture of BINDU. Additional components over baseline router are highlighted

6.4 Router micro-architecture

Bindu-bus: This is a bus connecting all the input ports, and is used to facilitate movement of the Bindu between VC-0 of the input ports (by moving the packet into the VC occupied by Bindu previously). A Bus suffices since only one Bindu can move per cycle; if multiple Bindus are concurrently present at a router, their period is skewed such that they do not contend for the bus in the same cycle.

Credit management unit: The upstream router is agnostic to the fact that there is a Bindu or an actual packet sitting at the downstream router. This implies that whenever Bindu replaces a packet in the router, there is no need to send updated credit signals to upstream routers involved. However, when Bindu replaces an empty slot, within the router, then both upstream routers, the one connected to the input port where Bindu was originally present and the other connected to the input port which originally had an empty slot, needs to be updated with credits accordingly.

Bindu movement unit: It is responsible for the overall movement of Bindu within

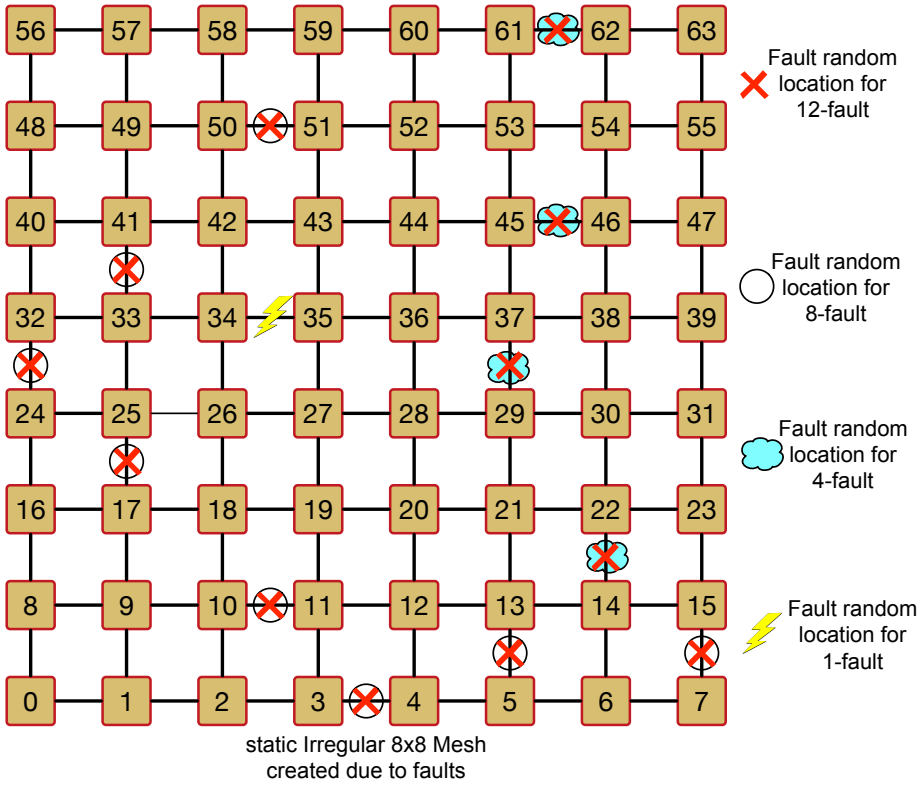


Figure 6.5: The figure shows irregular topologies, created out of a regular mesh. Faults in the network are shown as link failures at a random location, distributed randomly throughout the topology

the router until it leaves the router at a specific period. It comprises of two registers - one holding the Bindu movement period, and the other encoding its path within the router, including the output port connecting to the next router to route the Bindu to.

Multi-flit packets with Virtual Cut-Through (VCT) Routers. If the head-flit is present in VC-0 and it is the turn of this VC to turn into a Bindu, the VC is locked till the entire packet arrives and is not allowed to take part in switch arbitration. Once the entire packet arrives, transfer of this packet into the VC previously occupied by the Bindu is performed. The intra router Bindu period is chosen appropriately at design time such that entire packet can arrive and move before the next movement. If the head flit has already left, then the packet is allowed to naturally drain into its downstream VC without moving into the Bindu VC.

Multi-flit packets with Wormhole Routers: BINDU, as defined so far, works if VC-0 in each router is VCT, i.e., sized to hold complete packets (while other VCs can be smaller). To implement BINDU in wormhole routers, we would need to support packet truncation within VC-0, like prior works in deflection routing [42].

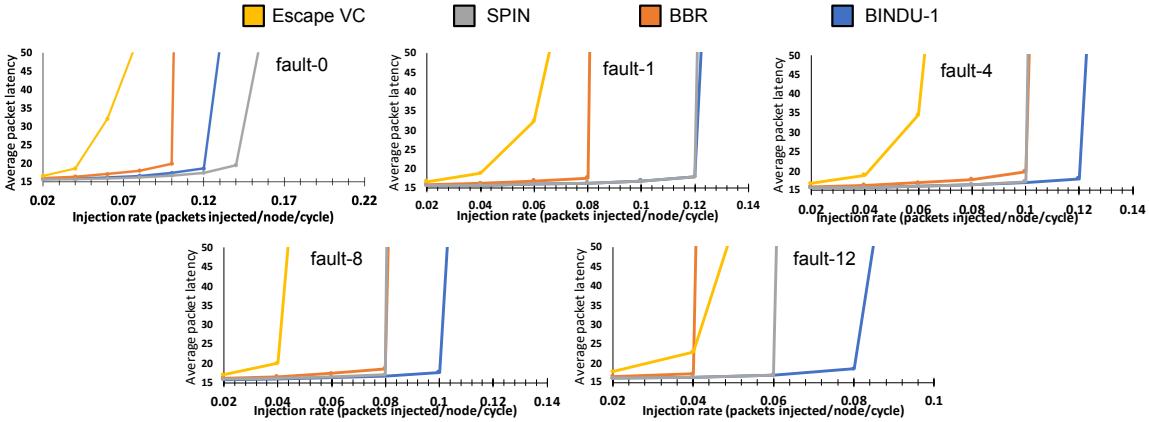
The router micro-architecture is shown in Figure 6.4. We discuss the key modules incorporated to facilitate Bindu movement:

Bindu Movement Example. We explain the Bindu movement within and across the router, with the example shown in Figure 6.4.

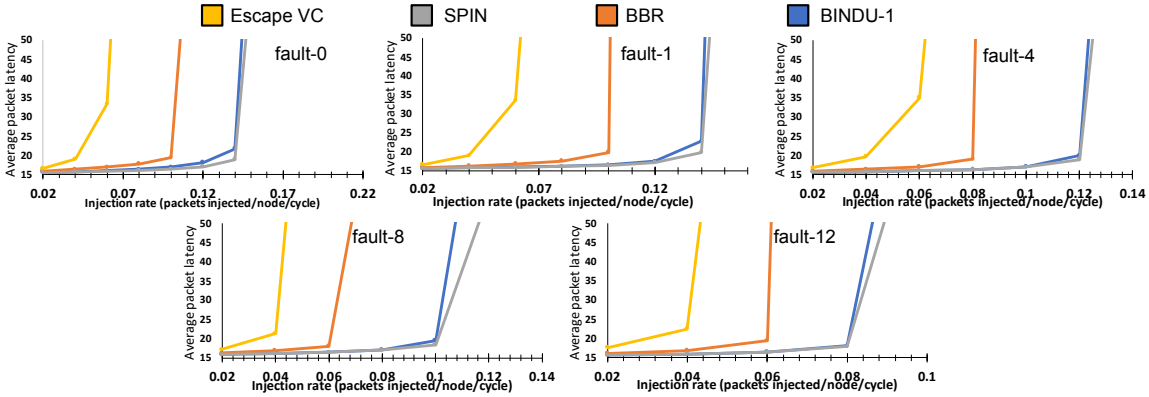
1. A Bindu enters the router from the North input port - this effectively means that a packet sitting at the North input port leaves the router to go sit in the South input port of the upstream router, in place of the Bindu, as shown in Figure 6.4(a).

2. The Bubble Movement Unit encodes the period and the route of the Bindu within this router. In the example in Figure 6.4(b), it moves the Bindu from the North to the East input port, via the Bindu bus. This step is similar to BBR [8].

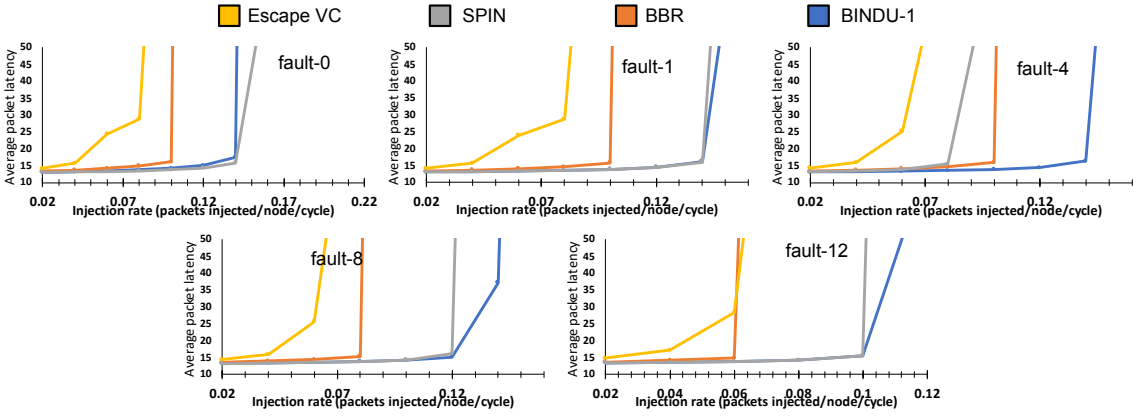
3. The Bindu will traverse all input ports sequentially. The last stop of the Bindu will



(a) Uniform Random



(b) Transpose



(c) Shuffle

Figure 6.6: Performance of BINDU compared against Deadlock avoidance, Deadlock recovery and BBR for synthetic traffic: Uniform-Random, Transpose and Shuffle. Evaluated for vc=2, 64 node irregular topology derived from 8x8 Mesh.

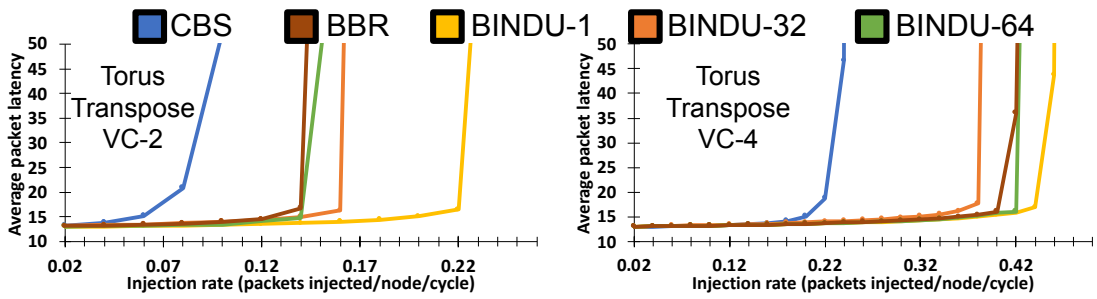


Figure 6.7: The graph compares the performance of BINDU with num Bindu=1, 32, 64 respectively with Critical Bubble Scheme and BBR. Graphs are for regular 8x8 Torus topology.

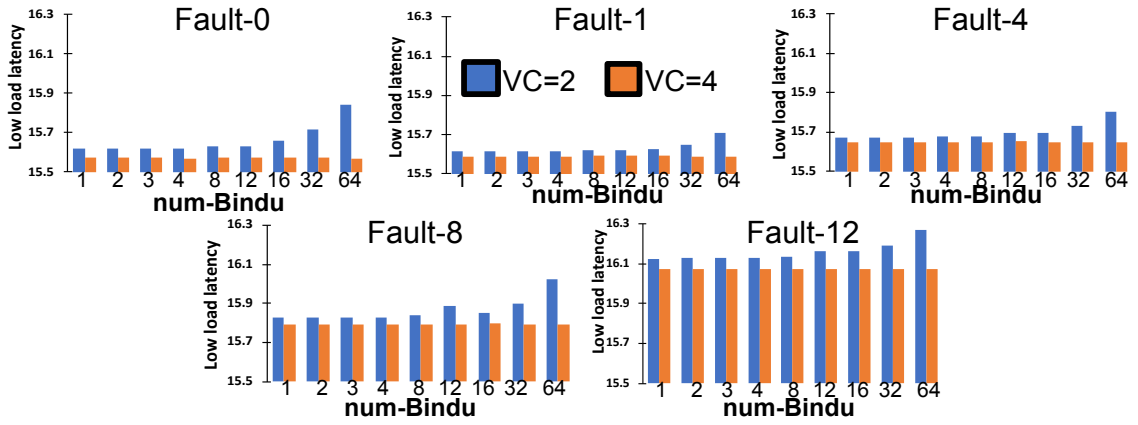
be the input port, whose corresponding output port is connected to the next router in Bindu's path.

4. The Bindu will use the crossbar to traverse to the neighbor by moving a packet (or an empty slot) at VC-0 of the corresponding input port from the neighbor to the current location occupied by Bindu, as shown in Figure 6.4(a).

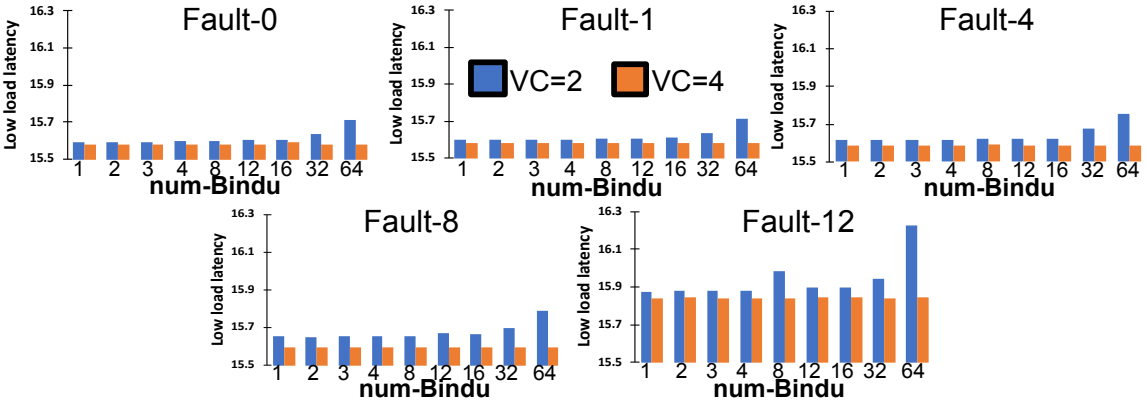
5. This whole process is now repeated at the neighboring router.

Implementation Cost. The hardware overhead of BINDU comes because of addition of *Bindu-bus*, and *Bindu Movement Unit*. We used DSENT [73] to estimate the area and power overhead. Area overhead comes around 6% and static power overhead is 5% over baseline router with VC=4.

Implementation choice for Bindu: In this work, we proposed to embed Bindu-path inside the router, and Bindu moves through the network along that specified path. Another implementation choice could be to think of Bindu as a dummy packet which moves throughout the network in a cyclic manner and never gets consumed. We then could have the path for Bindu embedded inside the Bindu itself. This would further simplify the router micro-architecture of BINDU, as each router would then only need to read the content of Bindu to know where to route it next. It would also be easier to reconfigure Bindu's path dynamically by updating the route within Bindu based on some metric. However, the flexibility and reconfigurability comes at the cost of scalability; encoding the Bindu-path will



(a) Uniform Random



(b) Transpose

Figure 6.8: Graphs are for Uniform Random and Transpose traffic pattern as number of Bindus increase from 1 to 64 in 8x8 irregular Mesh topologies with given fault. Graph shows the effect of low-load latency. We observe that the effect of number of bubbles on performance, is more for the router with fewer VCs compared to the router with more VCs per input port. All Bindus in BINDU are confined to VC-0 of each input port.

need $log(N) \times log(r) \times r$-bits (where N is the number of nodes, and r is the router radix), which can exceed typical flit sizes for large networks.

Implementation of Bindu-Path: There could also be multiple ways in which Bindu path can be implemented as shown in Figure 6.1. These paths would have different network-performance sensitivity for different irregular topologies. All these design choices are interesting to explore in future work, we, however, do not present these studies in this paper in the interest of space. We assume a snake-like structure for regular topologies, and a tree for irregular topologies.

Table 6.2: **Qualitative Comparisons of CBS, BBR and BINDU**

	CBS [39]	**BBR** [8]	**BINDU**[9]
Bubble Implementation	Bubble is an empty VC	Bubble is empty VC	Bindu is reserved VC or dummy packet
Movement	Bubble moves naturally as the packet moves	Random proactive bubble movement within router	Proactive Bindu movement as per Bindu-path.
Minimum Empty Buffers	one bubble per ring dimension. 8x8 torus network requires 32 bubbles	one bubble per router. 8x8 torus requires 64 bubbles	one Bindu in the entire network. 8x8 torus requires one Bindu
Topologies	closed loop/ring topologies, for example Torus	Works for any arbitrary topology	Works for any arbitrary topology
Routing restriction	uses dimensional order routing in the VC that contains the bubble	uses minimal random adaptive routing	BINDU uses minimal random adaptive routing
Misrouting	No packet gets mis-routed	in-frequent misroute during bubble exchange	At most one packet per Bindu movement
Flexibility	not flexible	not flexible	flexible in terms of number of Bindus and their path
Reconfigurable	not reconfigurable	not reconfigurable	dynamically reconfigurable

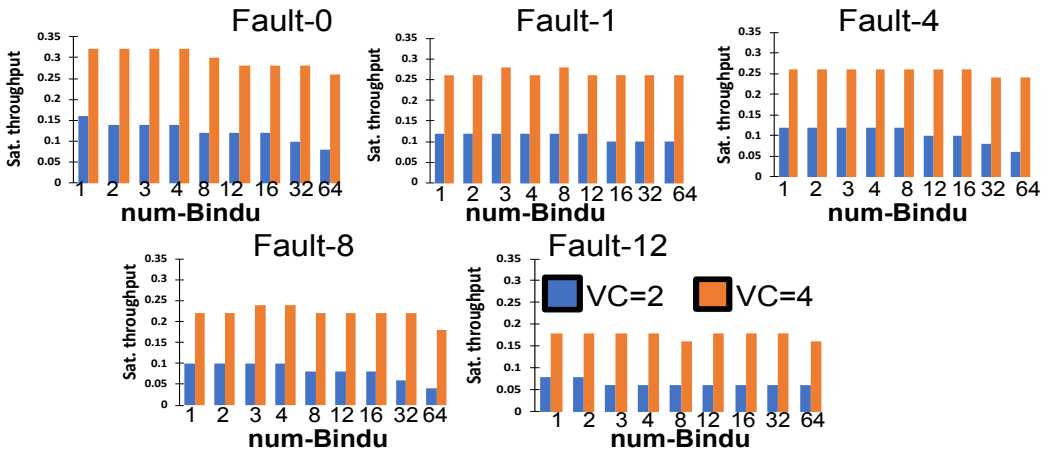
6.4.1 Comparison with CBS and BBR

Here we delineate BINDU from two notable works which use bubble to provide deadlock-freedom to the network. We paraphrase the main points in Table 6.2. BBR can be viewed as Bindu-64.

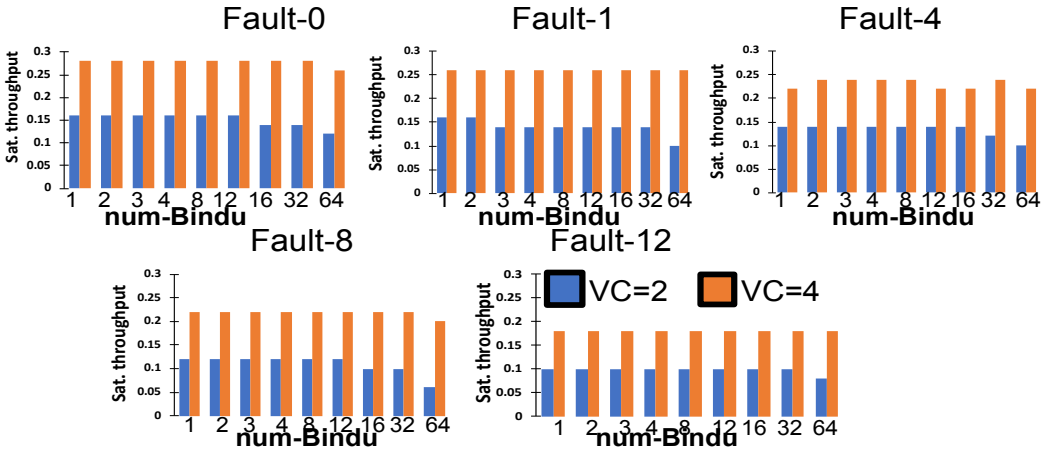
6.5 Evaluations

6.5.1 Methodology

BINDU is evaluated using *gem5* [7] with the *Garnet2.0* [59] network model and the *Ruby* memory model. We use *DSENT* [73] to model power and area for a 11 nm process. Table 6.3 lists all key configuration parameters for our evaluation.



(a) Uniform Random



(b) Transpose

Figure 6.9: Graphs are for Uniform Random and Transpose traffic pattern as number of Bindus increase from 1 to 64 in 8x8 irregular Mesh topologies with given fault. Graph shows the effect of saturation throughput. We observe that with increase in number of Bindus, saturation throughput decreases. Bindu-64 is similar to BBR

Table 6.3: **Key Simulation Parameters.**

Real application simulation parameters	
Core	64 cores and RISCV ISA (Ligra), 1GHz 16 cores and x86 ISA (Parsec3.0), 1GHz
L1 Cache	Private, 16KB Ins. + 16KB Data, 4-way set assoc.
Last Level Cache	Shared, distributed, 64KB, 8-way set assoc.
Cache Block Size	64B
Cache Coherence	MESI Directory (Ligra) Vnets=5 MOESI hammer (Parsec3.0) Vnets=6
Target Networks	
Topology	irregular 8x8 Mesh (Ligra and Synthetic workloads) irregular 4x4 Mesh (Parsec3.0)
Router latency	1-cycle
Num VCs	1, 2 and 4
Buffer Organization	Virtual Cut Through. Single packet per VC
Link Bandwidth	128 bits/cycle
Deadlock Avoidance **Deadlock Recovery** **Bubble based**	Escape VC [74] with Up-Down [25] within Esc-VC SPIN [41] CBS [39]; BBR [8]; BINDU-k (k: num of Bindus)

Baseline Networks. We select state-of-the-art baseline deadlock-free networks to compare against BINDU. From deadlock-avoidance, we use escape VCs (which are known to perform better than turn-restriction schemes [38, 41]). From deadlock-resolution, we choose SPIN [41], which has been shown to performs better than Static Bubble [38]. From bubble-based, we choose BBR [8] and CBS [39]. BBR works for arbitrary topologies while CBS only for Torii.

Benchmarks. Both real applications and synthetic traffic are used to evaluate BINDU. Applications are drawn from the Ligra benchmark suites [75] and from Parsec3.0 [33] . For synthetic traffic, we focus on *uniform random*, *transpose* and *shuffle* traffic with the mix of 1-flit and 5-flit packet size; results for other traffic patterns are qualitatively similar. The simulator is warmed-up for 1000 cycles, thereafter network statistics are collected by

injected fixed number of tagged packets by each node in the system. Simulation completes when all the tagged packets are received. We use an 8×8 irregular network for Ligra and synthetic traffic. Ligra applications have been simulated in syscall-emulation (SE) mode of gem5 while Parsec3.0 applications are simulated on irregular 4x4 network using full system simulation mode in gem5.

Topologies. BINDU performance is evaluated on fault-free and faulty 2D mesh networks as well as 2D Torus network topology.To create irregular topologies from 2D mesh, faults are injected randomly into the network while network connectivity is maintained as shown in Figure 6.5. For the 8×8 network, we consider a range of faulty links up to 12 in 2D mesh.

6.5.2 Performance

Irregular topologies. Figure 6.6 shows the performance comparison of BINDU with other state of the art deadlock-freedom schemes.

for irregular 2D 8x8 Mesh. Key-takeaway from this performance graph is that except regular 2D Mesh, BINDU performs comparably to the state of the art solutions. In fact, under certain traffic pattern for example uniform random and shuffle, BINDU performs better than state-of-the-art. On average, we see 15% improvement over saturation throughput in synthetic traffic pattern using BINDU.

Bubble-based Schemes. Figure 6.7 compares the performance of state-of-the-art bubble based deadlock-freedom techniques with BINDU for regular 8x8 Torus topology. Since CBS has 32, and BBR has 64 bubbles, we also contrast against iso-Bindu configurations. BINDU provides up to 2.2× higher throughput. Also, the performance of BINDU decreases as the number of Bindus increases in the topology. BBR can be approximately considered as Bindu-64 as now each router has a Bindu/bubble. Therefore, BBR's performance can be approximated with Bindu-64, and Bindu-32 lies between Bindu-1 and BBR. We observe 35% throughput improvement for VC=2 and 15% higher throughput for VC=4

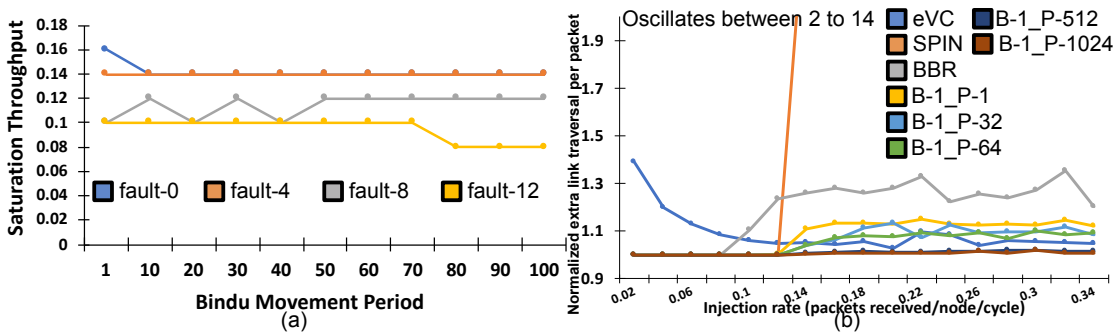


Figure 6.10: (a)Sensitivity of saturation throughput with increase in inter-router Bindu movement period of one Bindu for uniform random traffic. These results are for irregular 8x8 Mesh with VC=2.
(b)Uniform-random traffic, VC=2, with Fault-1. The graph shows the extra link traversal over the baseline using minimal deadlock-free routing. Here *B-1_P-X* means Bindu-1 with 'X' as Bindu Movement Period

with BINDU-1 compared to BBR.

6.5.3 Sensitivity studies

Number of bubbles

Figure 6.8 shows the sweep of low load latency as the number of Bindus increases in the topology from one to 64.

In general, we see low load latency increases with an increase in the number of Bindus, both for regular as well as irregular 8x8 Mesh topology. Effect on low load latency is more prominent for VC=2 than VC=4. Also, sensitivity reduces for higher fragmented topology (for example fault-12) compared to lower fragmented topology (for example fault-1).

Similar trends are shown by saturation throughput as the number of Bindus increases, saturation throughput decreases, for both regular and irregular Mesh topology in Figure 6.9. This happens mainly because of two reasons. Firstly, as the number of Bindus increases, more packets will be misrouted in the network. Secondly, Bindus cannot be consumed therefore they put indirect restrictions on the number of packets that can be injected into the network. Since Bindu only stays in VC-0, we see less sensitivity in saturation throughput when there are many VCs (for example VC=4) compared to when there are fewer VCs (for

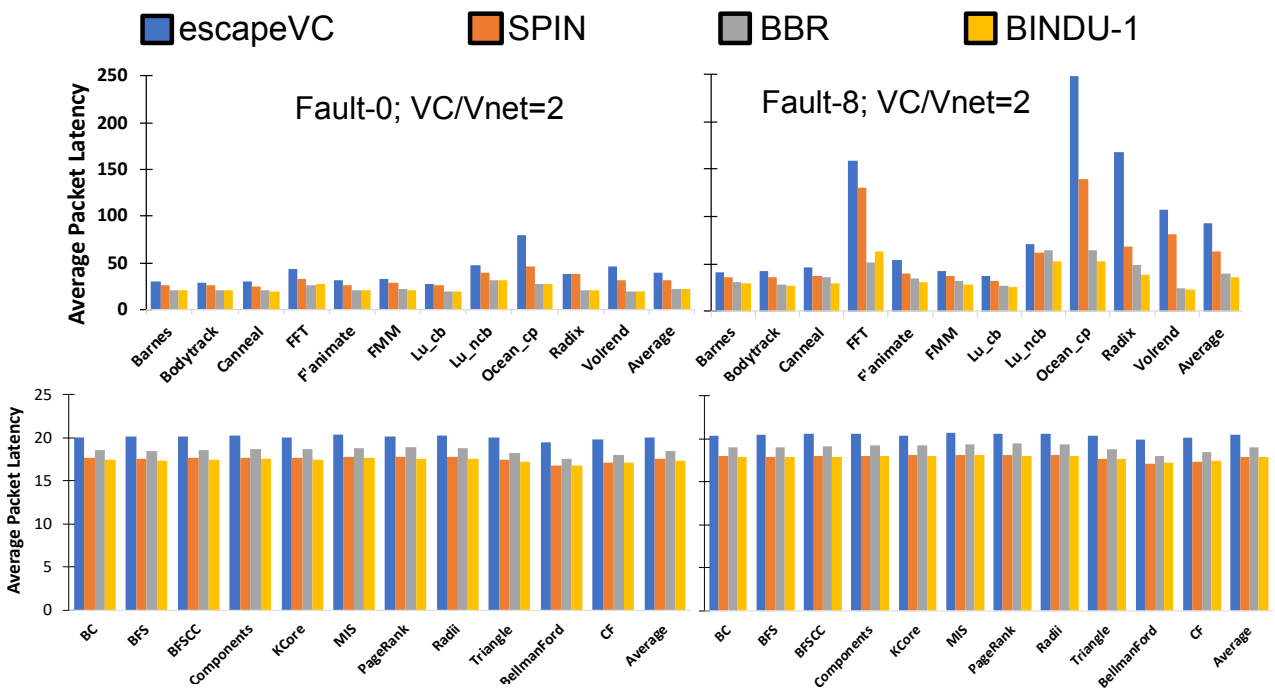


Figure 6.11: Packet latency from real workloads BINDU when compared to other state of the art schemes. Upper row is for Parsec3.0[33] workloads and lower row shows result for Ligra[75] workloads

example VC=2).

Inter-router Bindu movement period sweep

Figure 6.10-(a) shows the sensitivity of saturation throughput with increase in inter-router Bindu period. The experiment is performed for uniform random traffic pattern with VC=2, with one Bindu in the whole network. We observe decrease in saturation throughput with increase in inter-router Bindu-period. This happens because it takes longer for the Bindu to reach to the packets stuck in deadlock and free them from deadlock.

Energy Overhead

Figure 6.10-(b) shows the energy overhead for various schemes, in the form of extra link-traversal per packet, over the baseline assuming *ideal minimal routing* without any overhead for an irregular mesh with one fault. For escape VC the extra link traversal comes be-

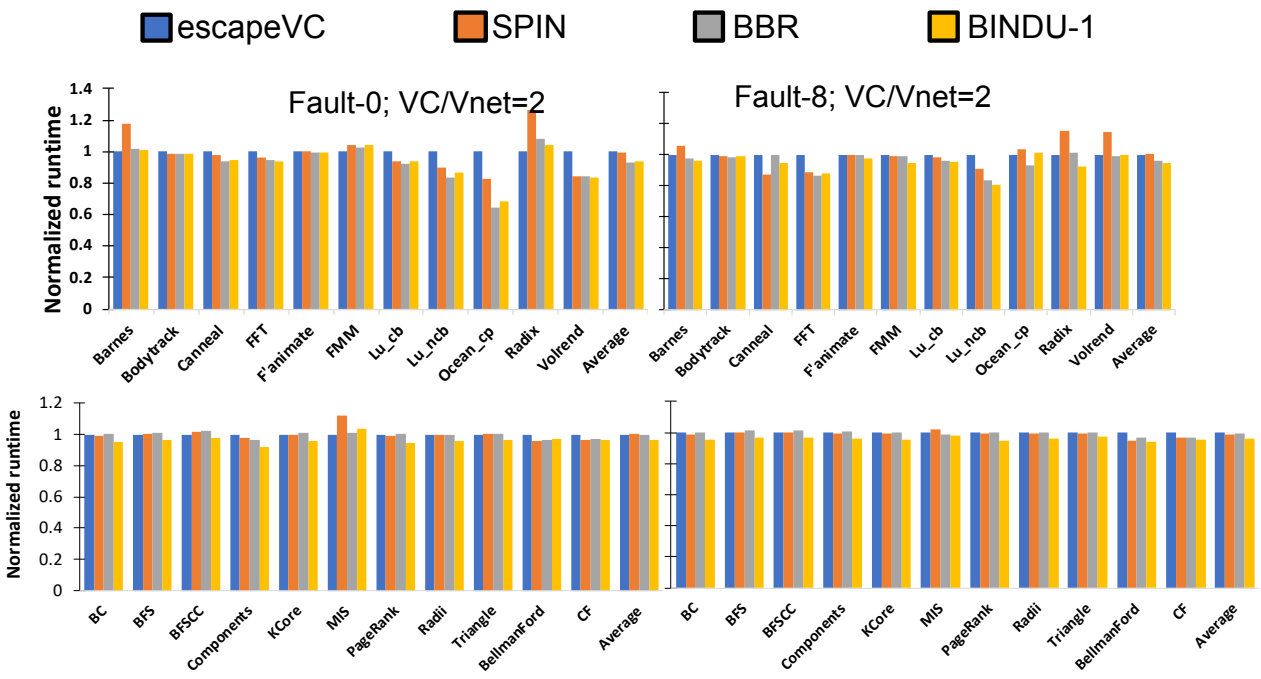


Figure 6.12: Normalized runtime improvement from real workloads with BINDU when compared to other state of the art schemes. Upper row is for Parsec3.0[33] workloads and lower row shows result for Ligra[75] workloads

cause of the non-minimal Up-Down [25] path a packet takes to reach its destination within the Escape VC. We observe higher overhead at lower injection rate because the fewer the packets in the network, the more sensitive they are to non-minimal path of escape VC. SPIN's [41] overhead over ideal minimal routing is because of probes used for detecting deadlock, especially at higher loads due to increased forking at intermediate routers. BBR's [8] link traversal overhead is because of increased *bubble-exchange* at high injection rate. BINDU's extra link traversal over ideal minimal routing is because at higher injection rate, there are more packets in the network, hence the likelihood of Bindu replacing a packet, as it moves along its path, increases. In summary, we can see that the additional energy expended by the misroutes due to Bindu movement is negligible at low-loads, and less than 10% post saturation (even with an aggressive Bindu movement period of 1), which is either equal to or much lower than other state-of-the-art solutions.

6.5.4 Real application results

Figure 6.11 shows the packet latency improvement for Parsec3.0[33] and LIGRA[75] benchmarks respectively. This culminates into 6% average improvement for fault-0 and 7% average improvement in packet latency for fault-8 in 4x4 mesh respectively.

Figure 6.12 shows the normalized runtime improvement for Parsec3.0[33] and LIGRA[75] applications respectively. There is overall around 5% improvement of BINDU over other schemes.

6.6 Discussion

BINDU shows that unlike BBR where *bubble* in each router is used to unblock the input port involved in deadlock, we can have one global *bubble* which would visit each input port of the router cyclically. There were two components to the motion of Bindu one is *inter-router bindu period* another is *intra-router bindu period*, both periods determine the frequency of movement of packets across routers and within the router across input ports.

Considering the routing deadlocks as rare event we see performance improvements of Bindu over BBR which conservative put bubble within each input port of the router.

6.6.1 Using CDG for *bindu-path*

In BINDU not much attention is given to bindu-path, except that it visits all input ports of the router and cyclic in nature. We believe there is room for improvement if we are selective in choosing the bindu-path. We believe CDG (subsection 3.1.1) is a good candidate for bindu path for two reasons:

- Cycles in CDG suggests that potential deadlock cycles in the network. In Bindu we showed that movement of bindu gives a *spin-effect*[41] to the packets involved in deadlock. If we move bindu along CDG of the network, then we can achieve higher performance.

- CDG naturally has the features of Bindu-path, as it visits all the input port of the router. Therefore, evaluating BINDU with CDG as Bindu-path could give us more insights of this scheme.

6.6.2 BINDU to resolve Protocol level deadlocks

Finally, we believe that BINDU can be extended beyond its current form of resolving routing level deadlock to resolving both routing and protocol level deadlocks. Protocol level deadlocks (subsection 2.15.2) arise because all the finite buffers of Network on Chip are occupied by packets from non-terminating message classes example: request packets, Ack packets. This results in packets from terminating message class, example response packet, not able to make forward progress to reach their destination. With Bindu, if bindu-path is chosen dynamically in such a way that it forces the response packets to reach its destination then protocol level deadlock can be resolved. We believe it would be interesting to study the performance of *new* BINDU technique which resolves both routing and protocol deadlocks.

In next set of chapters, we will see that we can resolve routing level deadlocks without *bubbles*. Particularly, we will talk about another subactive deadlock freedom mechanism called as *DRAIN: Deadlock Removal for Arbitrary Irregular Networks*. The idea is similar to BINDU as DRAIN also has a *virtual-ring* embedded in the network, called as *drain-path* (similar to *bindu-path*) instead of moving a bubble (*bindu*) all packets present on the drain-path move simultaneously. We will discuss it in more detail in next chapter. One Salient feature of DRAIN is that it resolves routing as well as protocol level deadlock.

6.7 Chapter Summary

BINDU is the first work, to the best of our knowledge, to demonstrate deadlock freedom by reserving a single bubble (empty VC) in the entire network, and pro-actively moving it through all routers and all input ports. BINDU requires no deadlock-detection (unlike reac-

tive schemes) and requires no turn-restrictions or escape VCs (unlike proactive schemes). Table 6.1 compares BINDU with prior proactive, reactive and subactive schemes. BINDU is topology-agnostic and provides around 15% average throughput improvement over state-of-the-art techniques in synthetic traffic, and around 7% improvement on an average runtime of real applications. This makes BINDU an effective solution to implement in irregular network topologies to guarantee deadlock-freedom.

CHAPTER 7

DEADLOCK REMOVAL FOR ARBITRARY IRREGULAR NETWORKS (DRAIN)

Correctness is a first-order concern in the design of computer systems. For multiprocessors, a primary correctness concern is the deadlock-free operation of the network and its coherence protocol; furthermore, we must guarantee the continued correctness of the network in the face of increasing faults. Designing for deadlock freedom is expensive. Prior solutions either sacrifice performance or power efficiency to proactively avoid deadlocks or impose high hardware complexity to reactively resolve deadlocks as they occur. However, the precise confluence of events that lead to deadlocks is so rare that minimal resources and time should be spent to ensure deadlock freedom. To that end, we propose DRAIN, a *subactive* approach to remove *potential* deadlocks without needing to explicitly detect or avoid them. We simply let deadlocks happen and periodically *drain* (i.e., force the movement of) packets in the network that *may* be involved in a cyclic dependency. As deadlocks are a rare occurrence, draining can be performed infrequently and at low cost. Unlike prior solutions, DRAIN eliminates not only routing-level but also protocol-level deadlocks without the need for expensive virtual networks. DRAIN dramatically simplifies deadlock freedom for irregular topologies and networks that are prone to wear-related faults.

DRAIN is next subactive technique after BINDU, and provides deadlock freedom by moving all packets in the network obliviously over pre-defined path. Unlike BINDU, it does not require bubble to move one packet at a time, instead it moves the packets on the pre-defined path all at once. Table 7.1 contrasts DRAIN against the prior work and subactive schemes presented so far.

Table 7.1: **Summary Table of Qualitative Comparison of Deadlock Freedom Mechanisms. P: Proactive, R: Reactive. S: Subactive.**

Techniques	Full Path Diversity	No Detect deadlock	No Misroute	No Extra Buffers	Routing deadlock freedom	Protocol deadlock freedom
Dally's theory/Acyclic CDG (P) [26]	✗	✓	✓	✓	✓	✗
Duato's theory/Escape VC (P) [52]	✗*	✓	✓	✗	✓	✗
Bubble [37, 50] (P)	✓	✓	✗	✗	✓	✓$^{+}$
Deflection (P) [42]	✗**	✓	✗	✓	✓	✓
Deadlock Buffers (R) [49, 47, 65, 38]	✓	✗	✓	✗***	✓	✓***
Coordination (R) [41]	✓	✗	✓	✗****	✓	✗
BBR (S) [8]	✓	✓	✓$^{++}$	✓	✓	✗
BINDU (S) [9]	✓	✓	✗	✓	✓	✗
DRAIN (S) [10]	✓	✓	✗	✓	✓	✓

* Within escape VCs: limited path diversity + requires topology information for escape path.
**At low-loads, full path diversity is available. But at medium-high loads, packets cannot control the directions or paths along with they are deflected.
***DISHA [47] uses timeout counters present at each input port to choose a packet to eject from the network. It requires a set of extra buffers to route the packet involved in deadlock. Some variations of DISHA, such as mDISHA [49] provide protocol-level deadlock freedom
****SPIN[41] requires a buffer in each router to hold the dynamic deadlock path over which packets involved in deadlock would move synchronously.
$^{+}$ Bubble Coloring [50] provides protocol-level deadlock freedom but involves non-minimal path traversal.
$^{++}$ BBR provides limited misrouting of packet because of *Bubble Exchange* subsection 5.3.1

7.1 DRAIN

This section describes the design of our DRAIN architecture, from theory to implementation.

Theory. The premise behind DRAIN is that routing-level and protocol-level deadlocks fundamentally need neither be detected nor strictly avoided; they just need to be *subactively* removed. Deadlocks are so rare that the cheapest solution is to simply let them happen (if ever) and periodically and obliviously *drain* resources in the network that *may* be in a deadlock. Even if no deadlock exists, correctness is maintained; draining would merely incur an infrequent and minimal performance overhead.

Analogy. Consider an analogy to street sweepers: periodically, sweepers will traverse the city streets along a pre-defined route to clear out leaves and other debris. This sweep is executed regardless of whether it is needed; fall may come late and the sweepers do their cleaning prior to the majority of leaves falling or fall may come early and the sweepers effectively clear away accumulated debris. DRAIN operates similarly, periodic draining of packets will occur regardless of need (ie., deadlock); ideally, these drains will coincide with the occurrence of an actual deadlock. Should the draining occur just prior to a deadlock, that deadlock will persist until the next scheduled draining.

Implementation Overview. Figures 3.11d and 3.15c show a high-level overview of DRAIN. Draining can be performed by any operation that 1) can eliminate a deadlock among packets if one exists, and 2) does not hurt correctness if no deadlock exists. In DRAIN, we employ a very low-cost draining mechanism that is inspired by the forced movement introduced in SPIN [41] yet avoids its complexity. Conventionally packets in a deadlock cannot move forward until they have observed that the packets in front of them have moved forward. In DRAIN, we periodically *drain* the network: we force all packets to move in a predetermined cyclic path (*drain path*). This unhinges any deadlocked packets and gives them the opportunity to eventually exit the deadlock cycle—by either making a turn or ejecting from the network—thus eliminating the deadlock. DRAIN does not need to globally coordinate the deadlocked packets via complex probe messages at runtime (as SPIN does, Figure 3.11c). Instead, DRAIN determines, offline, a cyclic path through the entire network that covers all links. Then routers *locally* drain the packets in their VC

buffers along this path, at preset periods at runtime (*drain windows*), even if no deadlock exists. Since both when to drain (i.e., drain window) and where to drain (i.e., drain path) are statically predetermined, DRAIN incurs very low hardware complexity. Though draining may misroute any packets currently in the VCs, misrouting does not hurt correctness. As our results demonstrate, misroutes are sufficiently infrequent that they have no significant impact on performance.

Routing-Level and Protocol-Level Deadlocks. DRAIN guarantees both routing-level and protocol-level deadlock freedom *simultaneously*. They share the same hardware implementation. If a cyclic dependence exists in the network, regardless of whether the packets belong to the same message class or different classes, the dependence is guaranteed to break eventually via periodically forcing packets to move.

Before describing the details of the architecture, we first state our baseline assumptions (Section 7.1.1). We then describe the two key components of DRAIN:

- An offline algorithm that finds a drain path composed of all links in the network (Section 7.1.2).
- A low-cost router architecture that periodically drains packets along this path (Section 7.1.3).

We conclude this section with the necessary proofs (Section 7.2) and a walk-through example (Section 7.2.5).

7.1.1 Assumptions and Definitions

We make three assumptions about the topology, which are commonly found in networks:

1. All routers are reachable by all other routers, even in the presence of faults. In other words, the network is connected, and all source-destination pairs are possible. This is a typical assumption since disconnected topologies serve little value in real-world multiprocessors.

2. All routers are connected via bidirectional links (i.e., two opposing unidirectional links). We find that this is true for most topologies. If a single unidirectional link becomes faulty, we assume that both opposing links (and their VC buffers) are disabled.

3. All turns (including U-turns) are possible in every router (i.e., every input port can route to every output port). Networks that employ adaptive routing often already provide this capability. Allowing for U-turns only requires modest changes to the allocators and crossbars.

Any topology that holds all the above assumptions is guaranteed to have at least one cycle (i.e., drain path) that spans all links. Since the network is connected, it is always possible to construct a spanning tree that covers all bidirectional links in the topology. Since each bidirectional link allows for an implicit turn to itself via a U-turn, the spanning tree is equivalent to a unidirectional cycle that covers all links and all routers. This cycle is equivalent to the path taken by a depth-first traversal through the spanning tree.

Definitions. We list key terminology:

- *Drain:* Force all packets currently in the network to take a specific turn, regardless of whether or not they are in a deadlock.
- *Drain Path:* A cycle that covers all links in the network, specifying where each drained packet must turn.
- *Pre-Drain:* Before draining, let any packets currently traversing a link complete.
- *Drain Window:* The predetermined time period reserved for all routers to perform draining.
- *Pre-Drain Window:* The predetermined time period reserved for all routers to perform pre-draining. This immediately precedes the drain window.

- *Epoch:* The time between drain windows.
- *Full Drain:* Allows all packets in the network to traverse the whole topology and eject out when they visit their destination router during traversal.

Protocol-Level Deadlocks. For protocol-level deadlocks, we make two additional assumptions. First, each router's injection and ejection ports use separate queues per message class. This is typical in modern shared-memory multiprocessors, which have dedicated queues for different coherence messages outside the network. Second, we assume that it is not possible for packets of a single message class to occupy all buffers in the network, leaving no space for other message classes. This is typical, since miss status handling registers (MSHRs) are often few enough relative to the number of VCs in the network, bounding the number of packets per message class. With these assumptions, if the network eliminates protocol-level deadlocks within it, then the multiprocessor system is guaranteed to be free of protocol-level deadlocks.

Draining Only Escape VCs. In networks with multiple VCs per port, we perform draining only on one VC and designate it as an escape VC. Every packet has an opportunity to enter the escape VC, and if it does, it is no longer allowed to move to any non-escape VCs. In contrast to typical escape VCs, our escape VC has no turn restrictions placed on it. Other VCs do not need to be deadlock-free since DRAIN ensures that the escape VC is deadlock-free.

7.1.2 Offline Algorithm

DRAIN ensures deadlock freedom by conservatively, periodically draining all escape VC buffers in the network. Given any arbitrary network topology, the goal of our offline algorithm is to find the *drain path*: a single cycle composed of all unidirectional links in the network. This can be done offline and rerun whenever a link becomes faulty. At runtime, during each drain window, all packets in the escape VCs are circulated along this path in

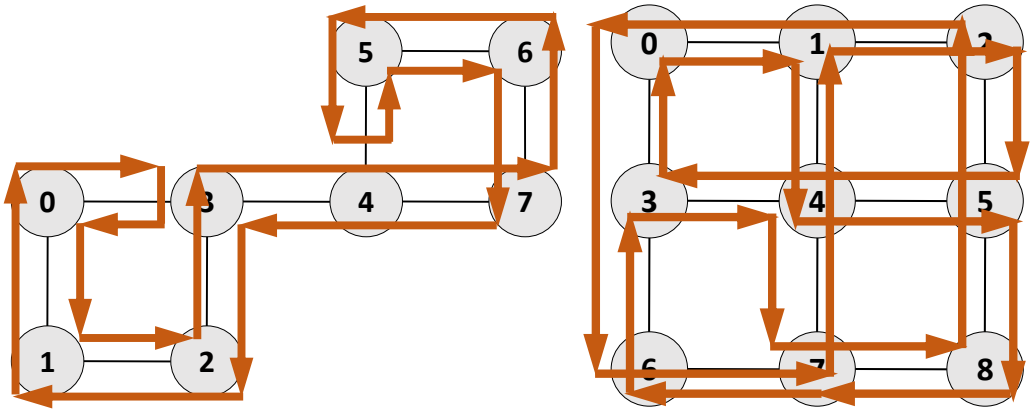


Figure 7.1: Sample outputs of our offline algorithm for (a) an irregular topology and (b) a regular topology. Each arrow represents a unidirectional link in the drain path.

unison, for some number of hops. Fig. 7.1 shows example outputs of our algorithm for an irregular topology and a regular topology. In the figures, each edge represents two opposing unidirectional links. All unidirectional links in these topologies are covered by the drain path found by our algorithm.

During each drain window, any packet currently in an escape VC buffer is forced to make a turn following the path.

DRAIN supports irregular network topologies in the presence of faulty links. The input topology is represented as a dependency graph G where each node is a unidirectional link in the topology, and each directed edge is a turn between two unidirectional links. We denote the set of all unidirectional links as L. A cycle in G is defined as a sequence of links $l = \{l_1, ..., l_n\}$ where l_i is connected to l_{i+1} via a turn, and l_n is connected to l_1 via a turn. Only *elementary* cycles need to be considered: an elementary cycle visits each link at most once. Non-elementary cycles are impossible since a link can only transfer at most one packet at a time. Our algorithm's goal is to find a cycle C where $l = L$. We are guaranteed to find at least one such cycle given our baseline assumptions in Section 7.1.1. Our implementation builds upon the cycle-finding method proposed by Hawick and James [76], with complexity $O((V + E) \times (C + 1))$, where V, E and C are the number of vertices, edges and cycles

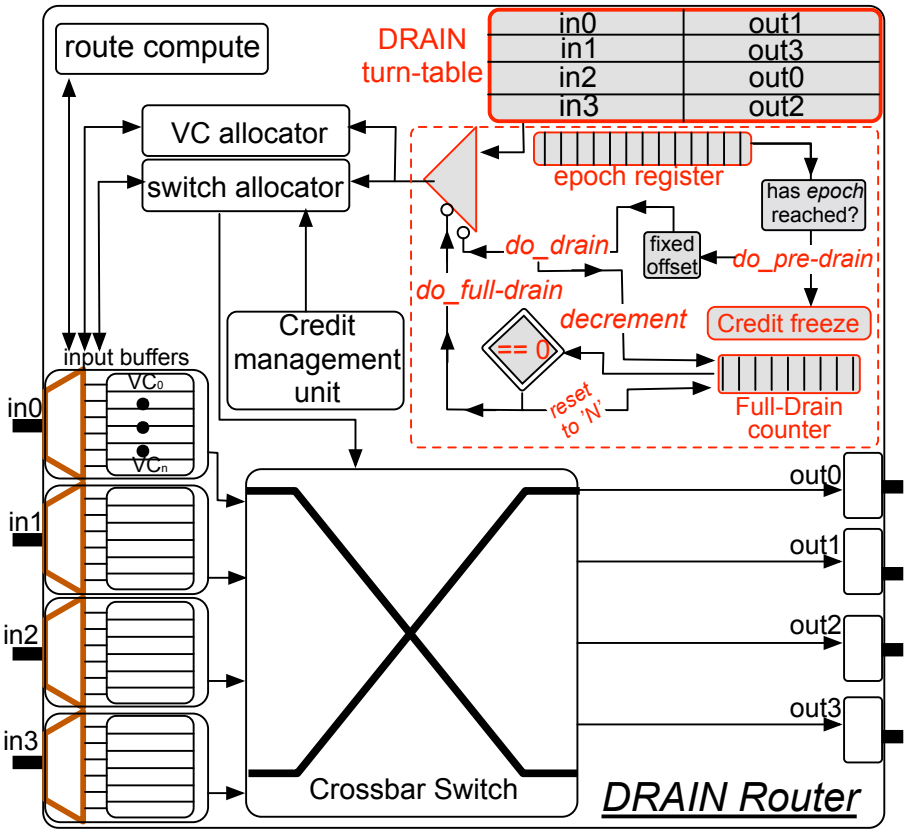


Figure 7.2: DRAIN router microarchitecture. The red modules are unique to DRAIN.

in G, respectively. This method employs a recursive tree search with efficient structures for tracking vertex adjacency lists. We augment this to terminate early as soon as a single cycle is found that covers all links in L. Since the algorithm only needs to be computed whenever the topology changes (i.e., when a fault occurs, upon a system reboot), we expect its runtime to be negligible to overall system performance.

7.1.3 Router Microarchitecture

Three changes are necessary to the router microarchitecture, highlighted in grey in Figure 7.2, each described in the following sections:

1. The *epoch register* for determining when it is time to pre-drain and drain.

2. The *credit freeze* for preventing new packets from allocating a VC during each pre-drain.

3. The *turn-table* for determining where each input port turns during each drain.

When to Drain and Pre-Drain

We denote the *pre-drain window* and *drain window* as the time periods (i.e., clock cycles) reserved for pre-draining and draining, respectively. Every drain window is preceded by a short pre-drain window. When to pre-drain and drain is established ahead of time and known by all routers in the network; these values are loaded at boot time and require no subsequent global coordination. We add an epoch register per router that counts down until the next pre-drain and drain window. As is common in chip multiprocessors, all routers operate on the same clock; thus, all epoch registers are always in sync. This means that routers *do not* need to communicate and coordinate their draining with each other. This is an important advantage of DRAIN over reactive deadlock resolution mechanisms (e.g., SPIN [41]) that must perform additional synchronization between routers to detect and eliminate deadlocks. Such synchronization limits scalability and incurs significant hardware complexity despite the fact that deadlocks very rarely arise.

Epoch. The time between drain windows (the *epoch*) is parameterizable and statically chosen at design time. On one hand, draining more frequently incurs higher energy and performance overhead since it drains (and potentially misroutes) packets more often even if no deadlocks exist. On the other hand, draining too infrequently runs the risk of allowing deadlocks to persist for longer periods and impeding system performance. We explore these trade-offs in our evaluation.

How to Drain and Pre-Drain

We describe how the network operates and what architectural changes are necessary for draining.

Pre-Drain Window. When the epoch counter reaches zero, each router initiates a pre-drain. To implement pre-draining, credit allocation is frozen for all packets that are cur-

rently not in-flight (i.e., not traversing a link). This ensures that when the drain window begins, no packets are in motion. The length of the pre-drain window is statically determined by the maximum packet size supported in the network (in our evaluation, this is 5 cycles).

Drain Window. At every drain window, the drain path specified by our offline algorithm (Section 7.1.2) is drained. As shown in Figure 7.2, draining employs a turn-table per router that overrides the VC and switch allocators. The turn-table stores the output port for which each input port is bound, corresponding to the drain path. Since only one entry is needed per input port, the turn-table is small and scales with the number of ports per router; the table size does not increase as more routers are added to the network.

Turn-tables can be configured at boot time, which will permit a new drain path to be computed by our offline algorithm in the event of a link fault.

When draining, the turn-table overrides the allocators and grants the packet exclusive priority to turn onto the specified output port. Every packet currently occupying an escape VC buffer is forced to move; they must follow the drain path by turning onto the next output port in the cycle, specified by the turn-table. Draining moves each packet by one hop.[1] If packet arrives at its destination router during draining, it may immediately eject if there is a free slot in the ejection queue.

Full Drain. To address livelock, DRAIN performs a *full drain* once every N drain windows, for very large N (full drain counter in Figure 7.2). A full drain involves draining the entire path such that each packet in an escape VC can visit all routers and may eject upon arriving at its destination. This incurs a high performance overhead but only needs to be done very rarely, since the likelihood of livelock is extremely low under typical operating environments.

[1]While it is possible to perform more than one hop, we find this to always perform worse than the single hop case

Discussion

Here we discuss how DRAIN eliminates the need for virtual networks and supports flit-based flow control.

Virtual Networks. As discussed previously, the convention is to use virtual networks to avoid protocol-level deadlocks, regardless of what mechanism is used to resolve routing-level deadlocks. The number of virtual networks is dependent on the number of messages classes in the system's communication protocol. Each virtual network requires a distinct set of VCs across all routers, so that the packets of one message class never block the packets of any other message classes. This imposes a significant area and power overhead. Unlike prior solutions for routing-level deadlocks, DRAIN provides protocol-level deadlock freedom implicitly and does not require any virtual networks. This is due to two properties of DRAIN: 1) the act of draining guarantees the movement of all packets along the drain path, and 2) the drain path passes through all routers in the network by design. As a result, DRAIN ensures that any packet of any message class will eventually have the opportunity to reach its destination router, regardless of what other packets are in its way. A detailed proof is provided in Section 7.2.3.

Flit-Based Flow Control. DRAIN is straightforward to implement on networks that use packet-based flow control; we opt for virtual cut-through in our implementation. To support flit-based flow control (e.g., wormhole), DRAIN leverages packet truncation mechanisms from prior work [42, 45]. Since DRAIN forces flits to turn obliviously, packets may be truncated by the draining; i.e., some flits are forced to turn in one direction while others turn elsewhere. Routers are augmented with additional logic for generating a new packet whenever a packet is truncated. Specifically, the router dynamically 1) encodes the downstream flit as a tail flit and 2) embeds header information to the upstream flit. Upon ejection, all flits are buffered at the MSHRs of the cache controllers. When all flits have been ejected, the full packet is reassembled and processed as usual. Unlike in prior work on deflection

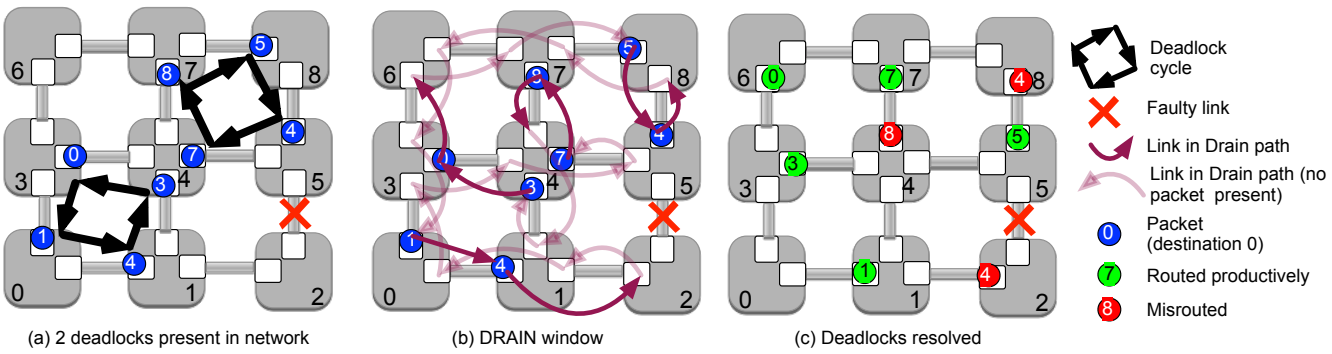


Figure 7.3: Step-by-step process of how DRAIN resolves deadlocks. (a) Packets have routed into two deadlock cycles in a faulty network. (b) During the drain window, all packets follow the predefined drain path in unison. (c) After draining for one hop, both deadlocks are broken.

routing [42, 45], packets are rarely truncated, only once every drain window.

7.2 Correctness Proof of Deadlock Freedom

7.2.1 Assumption

In DRAIN we assume that the *local* port of the router has separate VNet for each message class, this assumption makes sure that the response packet gets to enter the network irrespective of current occupancy of network buffers. However since *Drain path* only connects the network to network input ports of the topology, if all the network buffers are filled with *request packets* then *response packet* may not traverse the network as it will not get a chance to hop on a network input port either using usual routing algorithm or DRAIN. *This is extremely unlikely situation.* However, in this condition protocol deadlock will persist.

To resolve protocol level deadlock under this condition, we would *extend* drain path to cover the local input port (or injection input port) of the routers. This would allow the response packet at the local port of the router to occupy network input port of the router. *This is similar to the bindu movement as mentioned in subsection 6.2.2*. To facilitate this movement we would require *a bus connecting the local input port with network input ports of the router*. We would augment DRAIN router microarchitecture similar to Figure 6.4.

Once the response packet hops on the network input port of the router from its local input port during drain window, it is guaranteed to make forward progress either by following usual routing algorithm if unblocked or via DRAIN if blocked.

7.2.2 Proof of Routing-Level Deadlock Freedom

If a routing-level deadlock is present in a network, there must exist some cyclic dependence among packets in escape VC buffers along a set of links l. Deadlocks cannot exist in non-escape VCs since these packets will always have an opportunity to move to an escape VC. At every drain window, each escape VC in every link in l is drained, forcing any deadlocked packet in it to move one hop in some direction. After moving all packets by one hop, the deadlock will now be in one of two states: 1) the deadlock will either be broken, or 2) the deadlock will remain. The deadlock is broken if at least one of the deadlocked packets now has the opportunity to either eject from the network or turn away from the deadlock cycle onto a minimal path to its destination. If the deadlock remains, then all of the deadlocked packets in l must have moved one hop to a different link in l. Since the network is fully reachable (an assumption we make in Section 7.1.1), for each deadlocked packet, there is always at least one link in l that would arrive at the packet's destination router and offer the opportunity to eject. All deadlocked packets are guaranteed to eventually arrive at such a link, since at each subsequent drain window, each packet will continue to move one hop to the next link in l. Though each individual drain window is not guaranteed to break a deadlock, all deadlocks are guaranteed to be broken eventually in a future drain window or full drain (Section 7.1.3).

7.2.3 Proof of Protocol-Level Deadlock Freedom

If a protocol-level deadlock is present in a network, there must exist some packet p_A of some message class A that is blocking a packet p_B of another message class B, and message class A is dependent on message class B in the coherence protocol. If p_B were to reach

its destination router, the deadlock would be broken since all ejection queues are separated by message class, as stated in our assumptions (Section 7.1.1). The problem is that p_A is occupying a VC buffer that p_B needs to reach its destination router. By design, at every drain window, every packet must leave its current VC buffer and move in some direction, as designated by the drain path. After moving all packets by one hop, p_A and p_B will now be in one of two states: 1) p_A and p_B have moved in different directions, breaking the deadlock, or 2) both p_A and p_B have moved in the same direction. Even though the deadlock may still exist in the latter case, p_B is now waiting in the next router of the drain path. Since the drain path is guaranteed to pass through each router in the network at least once, p_B is guaranteed to eventually reach its destination router at some future drain window.

What if ejection queues are full? Though ejection queues may be full sometimes, they will never be full due to deadlock since we assume separate ejection queues per message class (Section 7.1.1). Thus, ejection queues are guaranteed to eventually free up. There will always be at least one *sink* message class (e.g., response messages) that corresponds to the end of a coherence transaction; the ejection queue of a sink message class can always be consumed.

What if there are a burst of deadlocks? Though deadlocks occur with low probabilities, there could be scenarios where packet injection leads to a burst of deadlocks one after the other. Each drain will resolve one/more deadlocks as the packets get re-distributed. In some cases, multiple drains may be required for the deadlocks to get resolved. The periodic full-drain (Section 7.1.3) will guarantee that no deadlock will be persistent.

7.2.4 Livelock and Starvation Avoidance

If ejection ports are busy, blocked packets may require multiple drains before they eventually exit the network. Though highly unlikely, this has the risk of continuously misrouting packets to the point where they never reach their destination. Both livelock and starvation are avoided by full draining, as discussed in Section 7.1.3. Though a full drain incurs

a performance overhead, it is very infrequent; for the vast majority of applications, a full drain is never needed. To further reduce the likelihood of livelock and starvation, we set the epoch (i.e., time between drain windows) to be no less than the expected worst-case latency of a packet in the network, which is proportional to the network diameter (i.e., the largest number of hops between any pair of routers). This ensures that if a packet is misrouted, it will have sufficient time to reach its final destination before the next drain window where it may be misrouted again. For most cache-coherent multiprocessors, the worst-case packet latency can be statically estimated because 1) the routers use well-known VC and switch arbiters that are proven to be fair, and 2) the caches and directories have finite queues and MSHRs, bounding the total number of in-flight transactions in the system.

7.2.5 Walk-Through Example

Figure 7.3 presents a walk-through example showing how DRAIN eliminates deadlocks. The `X` indicates a faulty link in the network between routers 2 and 5. Figure 7.3a shows two deadlock cycles. Packets are indicated by blue circles with their destinations specified; e.g., Packet `0` at Router `3` needs to travel south to Router `0` but is stalled waiting for Packet `1`. During the drain window (Figure 7.3b), all packets follow the drain path for one hop, as highlighted by the magenta arrows. The complete drain path, as computed by our offline algorithm (Section 7.1.2), is shown; the bolded arrows indicate the turns taken by the deadlocked packets. Figure 7.3c shows the resulting location of the packets after one hop along the drain path. For example, Packet `4` follows the drain path to Router `2`; this is a misroute. When the drain window ends, Packet `4` will need to travel back towards its destination. Similarly, Packet `0` is also misrouted away from its destination to Router `6`. Packets `1`, `3`, `5` and `7` are routed closer to their destination. Draining for one hop successfully breaks both deadlocks. In some cases, more than one drain window may be required to clear all deadlocks.

7.3 Methodology

DRAIN is evaluated using *gem5* [7] with the *Garnet2.0* [**Garnet**] network model and the *Ruby* memory model. We use *DSENT* [73] to model power and area for a 11 nm process. We simulate 16 and 64-core processors with a 2-level cache hierarchy. The cache architecture uses 32KB and 64KB L1 instruction and data caches respectively and 2MB last level cache (LLC) with MESI directory coherence protocol.

DRAIN is compared against escape VCs (with routing restrictions) [52] and SPIN [41]. While DRAIN only needs one virtual network, the baseline designs need multiple virtual networks to prevent protocol-level deadlocks. In our evaluations, we provision each virtual network with two VCs. Unless otherwise specified, our default implementation of DRAIN uses 64K-cycle epochs and a single virtual network with two VCs; we refer to this configuration as *VN-1, VC-2*. For completeness, we also evaluate DRAIN with 1) the same number of virtual networks as the baselines (*VN-3, VC-2*) and 2) one virtual network with the same number of total VCs as the baselines across all virtual networks (*VN-1, VC-6*). Note that only the escape-VC baseline requires two VCs per virtual network; SPIN can operate with a single VC per virtual network. However, for a fair performance comparison, we evaluate our baselines with two VCs per virtual network.

DRAIN's performance is evaluated on a fault-free 2D mesh and faulty irregular networks. Faults are injected randomly as link failures in the network topology while ensuring connectivity is maintained. For the 4×4 network, we model 0 and 8 faulty links; for the 8×8 network, we consider a range of faulty links up to 12. For each fault case, 10 different simulations with 10 randomly selected fault patterns of the given link failures are chosen. These different patterns result in a wide range of irregular topologies. The results presented are averaged across all 10 cases. In our results graphs, latency is shown in cycles and saturation throughput is shown in packets received/node/cycle.

Table 7.2: **Key Simulation Parameters.**

Real application simulation parameters	
Core	64 cores and RISCV ISA (LIGRA), 1GHz 16 cores and x86 ISA (PARSEC, SPLASH-2), 1 GHz
L1 Cache	Private, 32KB Instruction + 64KB Data 4-way set associative
Last Level Cache (LLC)	Shared, distributed, 2MB 8-way set associative
Cache Coherence	MESI (LIGRA, PARSEC, SPLASH-2); VNet=3
Network parameters	
Topology	Irregular 8x8 Mesh (LIGRA and synthetic workloads) Irregular 4x4 Mesh (PARSEC and SPLASH-2)
Routing Algorithm	DoR (Regular Mesh, Escape VC) Up*/Down* (Irregular topologies, Escape VC) Fully adaptive random (SPIN) Fully adaptive random (DRAIN)
Router Latency	1-cycle
Virtual Network	3-VNet (Escape VC, SPIN) 1-VNet (DRAIN) 2 VCs/VNet
Buffer Organization	Virtual Cut Through. Single packet per VC
Link Bandwidth	128 bits/cycle
Number of faults	0, 8 (LIGRA, PARSEC, SPLASH-2) 0, 1, 4, 8, 12 (Synthetic traffic)

7.3.1 Workloads

DRAIN is evaluated on both real-world applications and synthetic traffic. Applications are drawn from the PARSEC, SPLASH-2 and Ligra benchmark suites [33, 32, 75]. For synthetic traffic, we focus on *uniform random* and *transpose* traffic with a mix of 1-flit and 5-flit packet sizes; results for other traffic patterns are qualitatively similar. The simulator is warmed up for 1000 cycles; thereafter network statistics are collected by injecting a fixed number of tagged packets at each node in the system. Simulation completes when all the tagged packets are ejected from the network. We use an 8×8 network for Ligra and synthetic traffic and a 4×4 network for PARSEC and SPLASH-2.

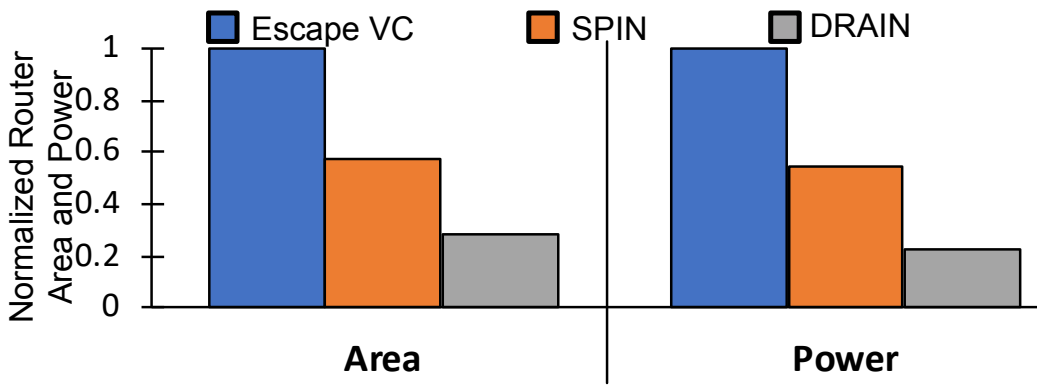


Figure 7.4: Router area and static power comparison.

7.4 Evaluation

We first compare DRAIN against prior proactive (escape VCs with up*/down* routing and virtual networks) and reactive (SPIN) solutions in terms of area and power. We then evaluate performance, under both synthetic traffic and real application execution. Finally we sweep the key design-space parameters of DRAIN and quantify the impact on packet tail latency, compared against the baselines.

7.4.1 Area and Power

In this section, we highlight the advantage of DRAIN in terms of area and power consumption. Figure 7.4 shows both the router area and static power normalized to the baseline escape VCs. The total router power includes that of the baseline hardware resources (i.e., buffers and allocators) and the power consumption of the additional resources that are required to handle deadlock. Escape VCs require an extra VC to proactively avoid deadlocks, which leads to significant overhead. Both baseline systems (escape VCs and SPIN) require multiple virtual networks to ensure protocol-level deadlock freedom. DRAIN, on the other hand, inherently eliminates protocol-level deadlocks and thus improves router power by about 77% compared to the baselines.

As shown in Figure 7.4, the simplified design of DRAIN yields almost 72% reduction in area compared to escape VCs. The majority of the area reduction comes from the elim-

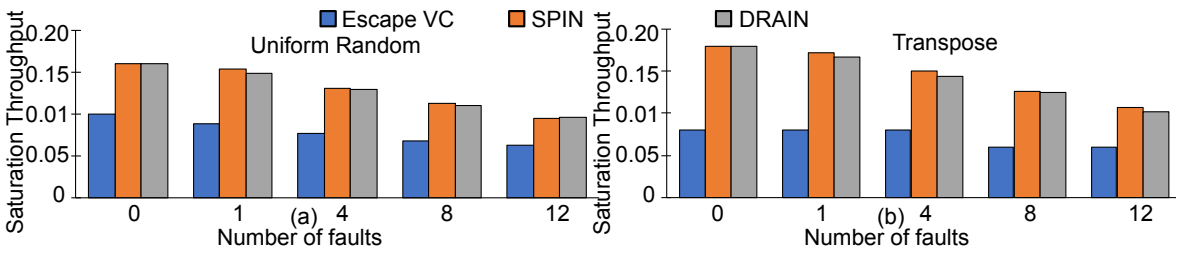


Figure 7.5: Saturation throughput for synthetic traffic patterns with increasing number of faults in an irregular 8×8 mesh.

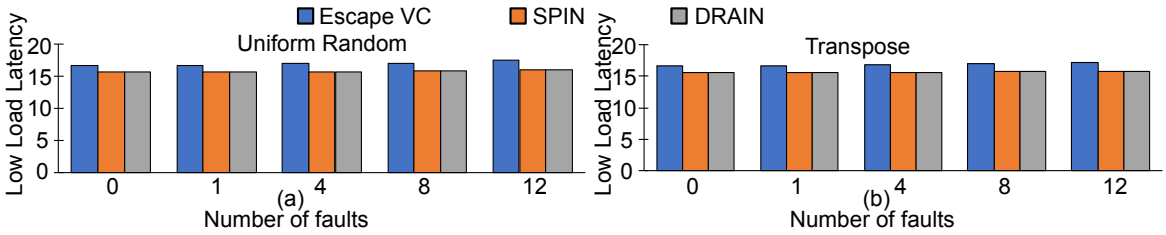


Figure 7.6: Low-load latency for synthetic traffic patterns with increasing number of faults in an irregular 8×8 mesh.

ination of extra virtual networks; however, it is worth noting that SPIN imposes a ∼15% overhead compared to a basic router design with dimension-order routing (DoR) to handle the extra control complexity for global coordination. In our comparison, both escape VC and SPIN have an equal number of virtual networks, but escape requires at least two virtual channels per virtual network, while SPIN can work with a single virtual channel per virtual network. DRAIN works with a single virtual network (as it is protocol level deadlock-free) and a single virtual channel within the single virtual network. This yields significant savings in router area and power with DRAIN.

Though our MESI protocol requires only three virtual networks, other coherence protocols may require even more; e.g., MOESI requires six virtual networks. In these cases, the area and power savings of DRAIN would be even greater.

7.4.2 Performance

This section evaluates DRAIN's performance compared to prior deadlock-freedom solutions.

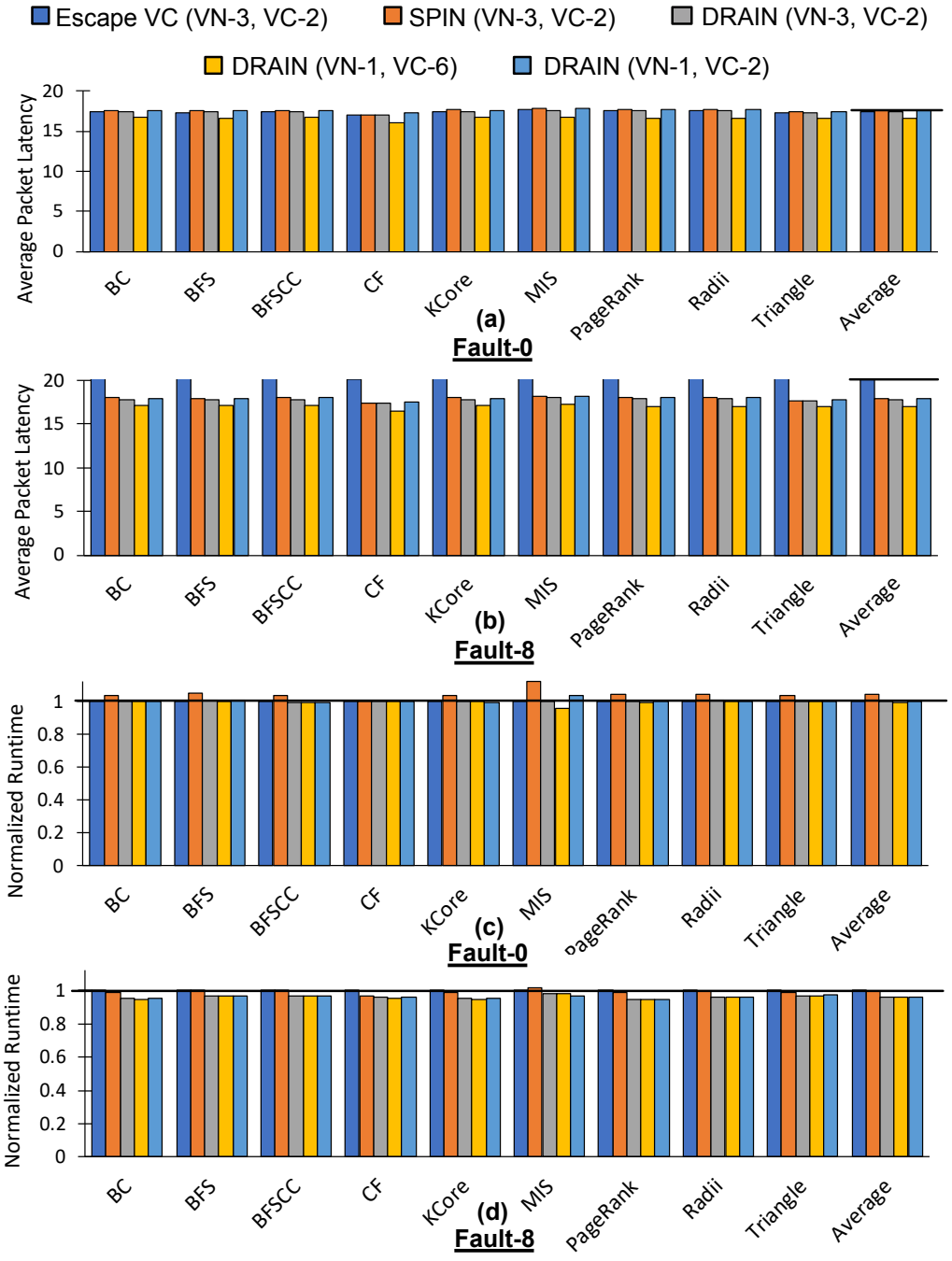


Figure 7.7: Packet latency and runtime of LIGRA applications on an 8×8 mesh with 0 and 8 faults.

Synthetic Traffic.

Figure 7.5 shows the saturation throughput for DRAIN and the baseline designs with increasing faults. Escape VCs yield the lowest throughput of the three techniques. The routing restrictions on the escape VC significantly reduce performance for all packets regardless of the low probability of deadlock. SPIN increases throughput by reacting to the rare case of deadlock. DRAIN achieves the same throughput as SPIN for uniform random traffic and slightly lower throughput for transpose traffic. In the event of a deadlock, DRAIN may wait longer than SPIN to resolve the deadlock, since DRAIN is an oblivious approach (i.e., it has no detection mechanism). SPIN uses a time-out of 1024 cycles, while we evaluate DRAIN with an epoch of 64K cycles. With substantially lower complexity than SPIN (Section 7.4.1), we achieve nearly equivalent performance.

Figure 7.6 compares the low-load packet latency for the three designs. DRAIN achieves the same latency as SPIN and both techniques achieve better latency than escape VCs. The escape VC with up*/down* routing forces non-minimal paths on the majority of packets leading to higher hop counts. At low loads, we expect deadlock to be extremely rare; thus DRAIN achieves equivalent low-load latency to SPIN. For all techniques, low-load latency increases with increase in number of faults; faults reduce path diversity and some packets must route non-minimally around faults for all three techniques.

Application Results. Figures 7.7 and 7.8 show the performance results of SPIN and DRAIN, normalized against escape VCs on the Ligra and PARSEC workloads, respectively. We evaluate on mesh networks with 0 and 8 faults. For the topology with 0 faults, we configured the escape VCs to use minimal dimension-order routing while all other non-escape VCs use fully adaptive routing. For 8 faults, we configured the escape VCs to use non-minimal up*/down* routing, since DoR is not possible in the presence of failed links.

We evaluate three configurations of DRAIN: same number of virtual networks as the baselines (VN-3, VC-2), one virtual network with the same number of total VCs as the baselines (VN-1, VC-6) and the default configuration (VN-1, VC-2). In general, DRAIN

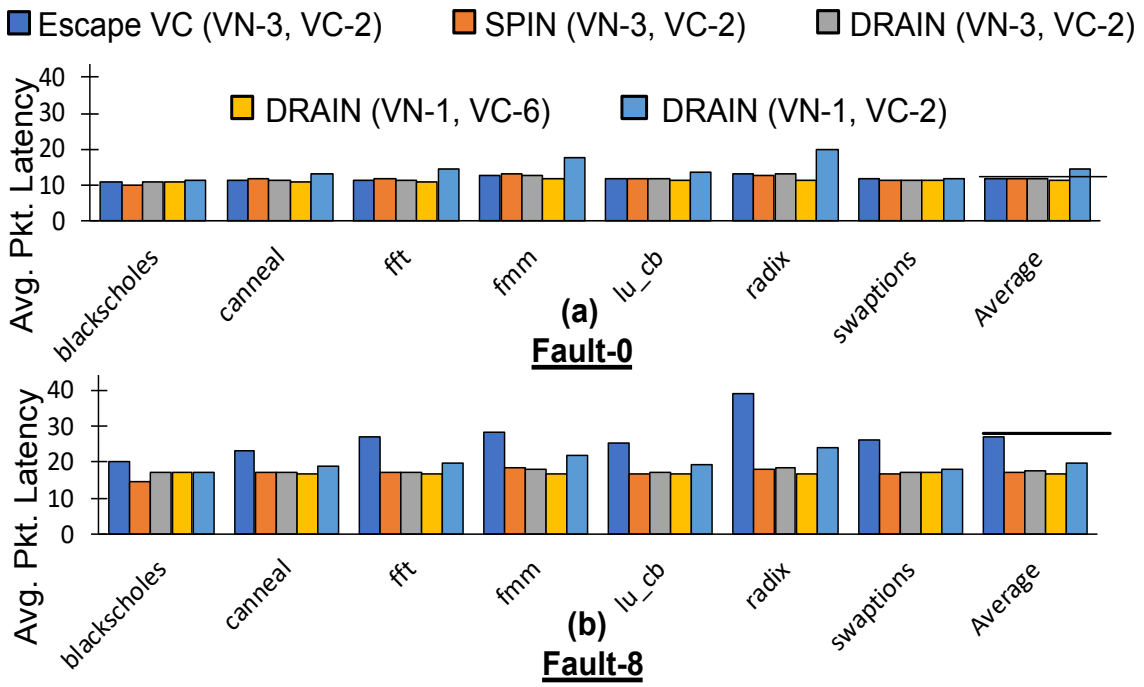


Figure 7.8: Packet latency of PARSEC and SPLASH-2 applications on a 4×4 mesh with 0 and 8 faults.

and SPIN show similar average performance on our applications. The average packet latencies across these workloads, as shown in Figures 7.7a and 7.7b for Ligra and Figures 7.8a and 7.8b for PARSEC, are fairly close between DRAIN and SPIN. In our default DRAIN configuration (VN-1, VC-2), packet latency is higher since there are $1/3$ less total VCs than the baselines. Despite this, the application runtimes are not harmed, as shown in Figures 7.7c and 7.7d. Thus, DRAIN achieves the same performance as SPIN at $\sim 1/3$ the hardware cost (Figure 7.4).

7.4.3 Sensitivity Studies

This section explores DRAIN in greater detail.

Epoch. Figures 7.9a and 7.9b show the impact of varying the drain epoch from 16 to 64K cycles on low-load packet latency and saturation throughput, respectively. These experiments are performed on uniform random traffic. In the extreme case of 16 cycles, the network is continuously flushing the drain path, leading to poor throughput and latency

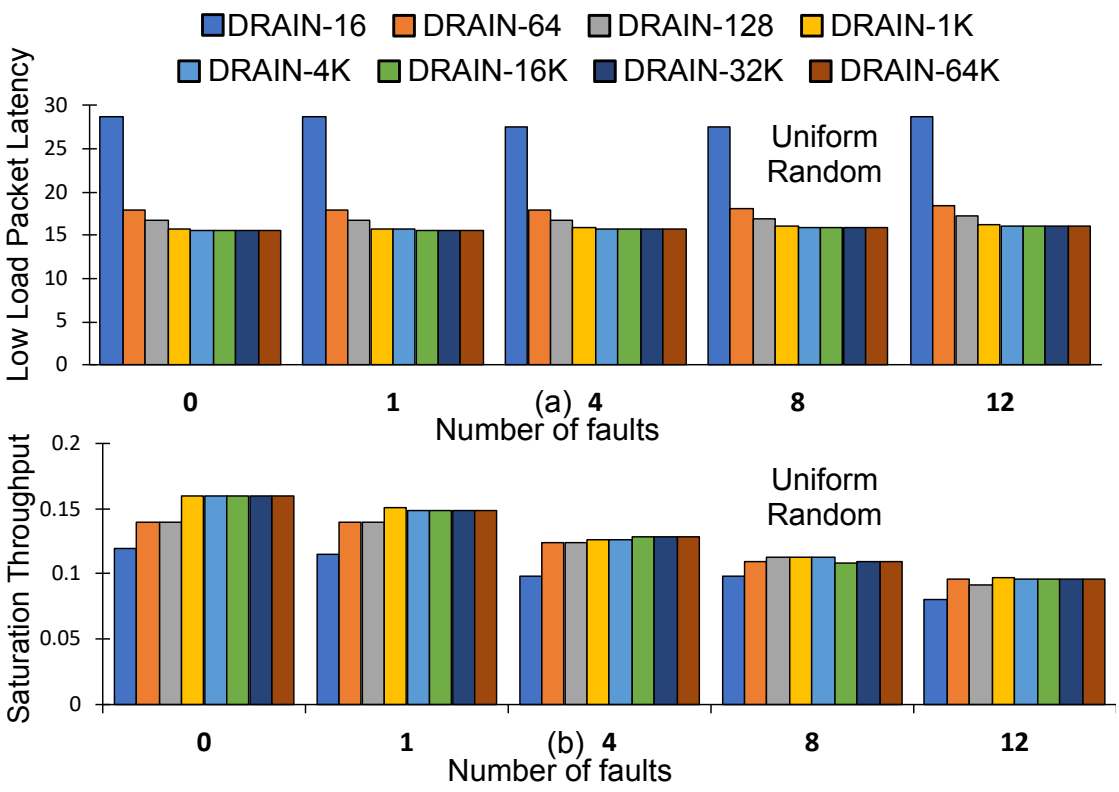


Figure 7.9: Low-load latency and saturation throughput of DRAIN as a function of the epoch, with increasing number of faults.

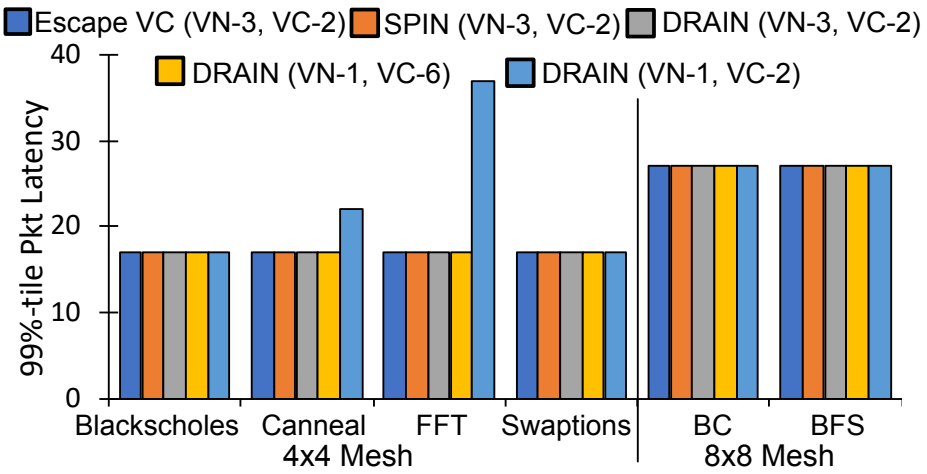


Figure 7.10: 99th-percentile latency comparison.

due to frequent misrouting. As the epoch is increased, latency is reduced and saturation throughput is increased. Draining is best done very infrequently due to the low likelihood of deadlocks.

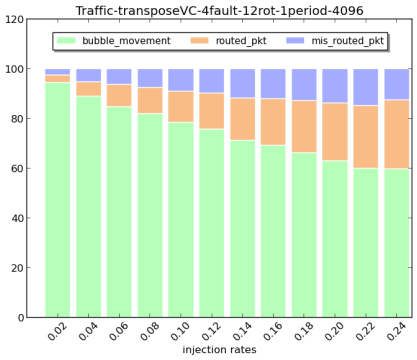
Tail Latency. Figure 7.10 shows the effect of DRAIN on the 99th-percentile network packet latency compared to escape VCs and SPIN. Since DRAIN is oblivious to deadlocks, there is a risk of allowing deadlocks to clog the network and degrade performance for a long period of time. In our experiments, we find that despite infrequent draining (i.e., large 64K epochs), the impact on tail latency is small. We observe a modest increase in 99th-percentile latency only when DRAIN is configured with less total VCs than the baselines (VN-1, VC-2), running the most memory-intensive applications.

7.5 Discussion

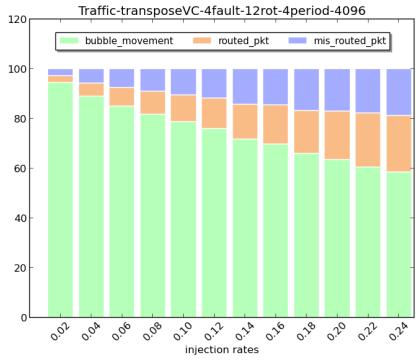
DRAIN is a new way to propose both routing and protocol level deadlock freedom. A question that is important to answer is:

How much the oblivious packet movement on pre-defined drain path cause packet *mis-routing*?

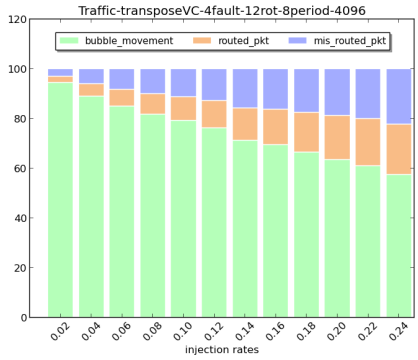
Figure 7.5 and Figure 7.6 does qualitatively shows that the impact is not significant for the



(a) One hop on DRAIN-path



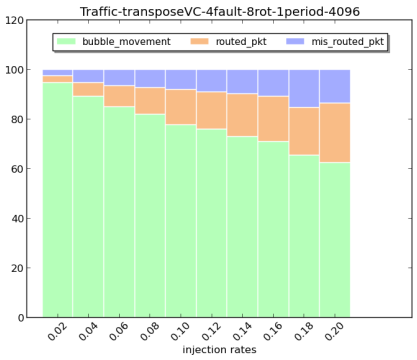
(b) Four hops on DRAIN-path



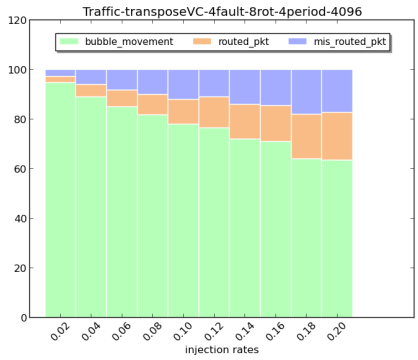
(c) Eight hops on DRAIN-path

Figure 7.11: Transpose traffic; 8x8 Mesh with 12 link failures, each input port has 4 VCs. Percentage of routed, mis-routed and free buffers present in the topology until saturation.

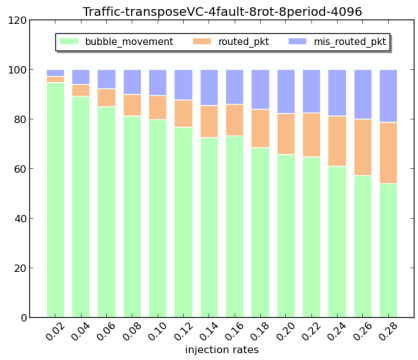
given traffic pattern. We further investigated and quantified what percentage of packets get misrouted, routed due to the oblivious packet movement over pre-defined drain paths as shown in Figure 7.11, Figure 7.12, and Figure 7.13.



(a) One hop on DRAIN-path



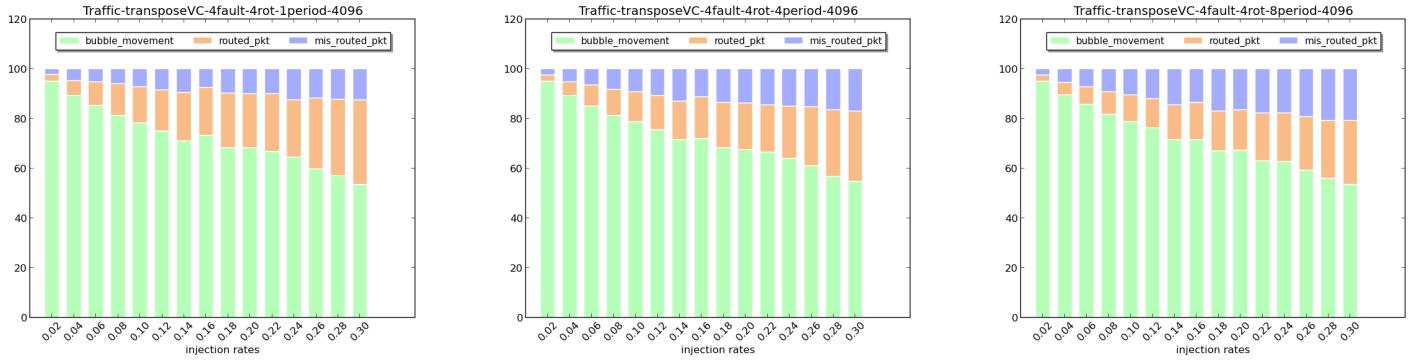
(b) Four hops on DRAIN-path



(c) Eight hops on DRAIN-path

Figure 7.12: Transpose traffic; 8x8 Mesh with 8 link failures, each input port has 4 VCs. Percentage of routed, mis-routed and free buffers present in the topology until saturation.

The consistent trend that we observe here is that misrouted packets is less than routed packets. Although the %age of packets getting misrouted increases as the number of hops of packets traveling on the oblivious DRAIN path increases. In the original paper we however showed the results with packets traveling one hop on the predefined DRAIN path during each epoch.



(a) One hop on DRAIN-path (b) Four hops on DRAIN-path (c) Eight hops on DRAIN-path

Figure 7.13: Transpose traffic; 8x8 Mesh with 4 link failures, each input port has 4 VCs. Percentage of routed, mis-routed and free buffers present in the topology until saturation.

In certain cases, however, the saturation throughput increases when the hops travelled per DRAIN increases. We think that this happens because of better load balancing in the face of regional congestion in an irregular topology for certain traffic patterns.

In this thesis, we started with BBR as first subactive technique to provide deadlock freedom, BBR arguments each router of the network with empty VC or bubble this limits the number of buffers available for packet to use, which results in low saturation throughput. BINDU, improves upon BBR by using one bubble or *bindu* which traverses the network in a predefined-path called *bindu-path*. DRAIN does not involve any bubble, instead moves the packets on a predefined cyclic path, called *DRAIN-path*, all at once periodically.

DRAIN proposes to obliviously spun the packets in the network and cause misrouting. In next chapter we will talk about another subactive technique to resolve routing and protocol deadlocks, called *SWAP: Synchronized Weaving of Adjacent Packets for Network Deadlock Resolution*. It proposes to *swap/exchange* packets from adjacent routers periodically, throughout the network to resolve any deadlock that might form. This work does not require global coordination among the packets in the network, SWAP is light weight, and provides higher performance. Moreover, it limits the misrouting of packets compared to DRAIN as we will see in next chapter, SWAP request by upstream router can be declined by the downstream router, if the packet at downstream router is at its destination,

or if there is free VC present at the downstream router. Therefore, SWAP provides higher performance compared to DRAIN.

7.5.1 Packet Latency Histogram

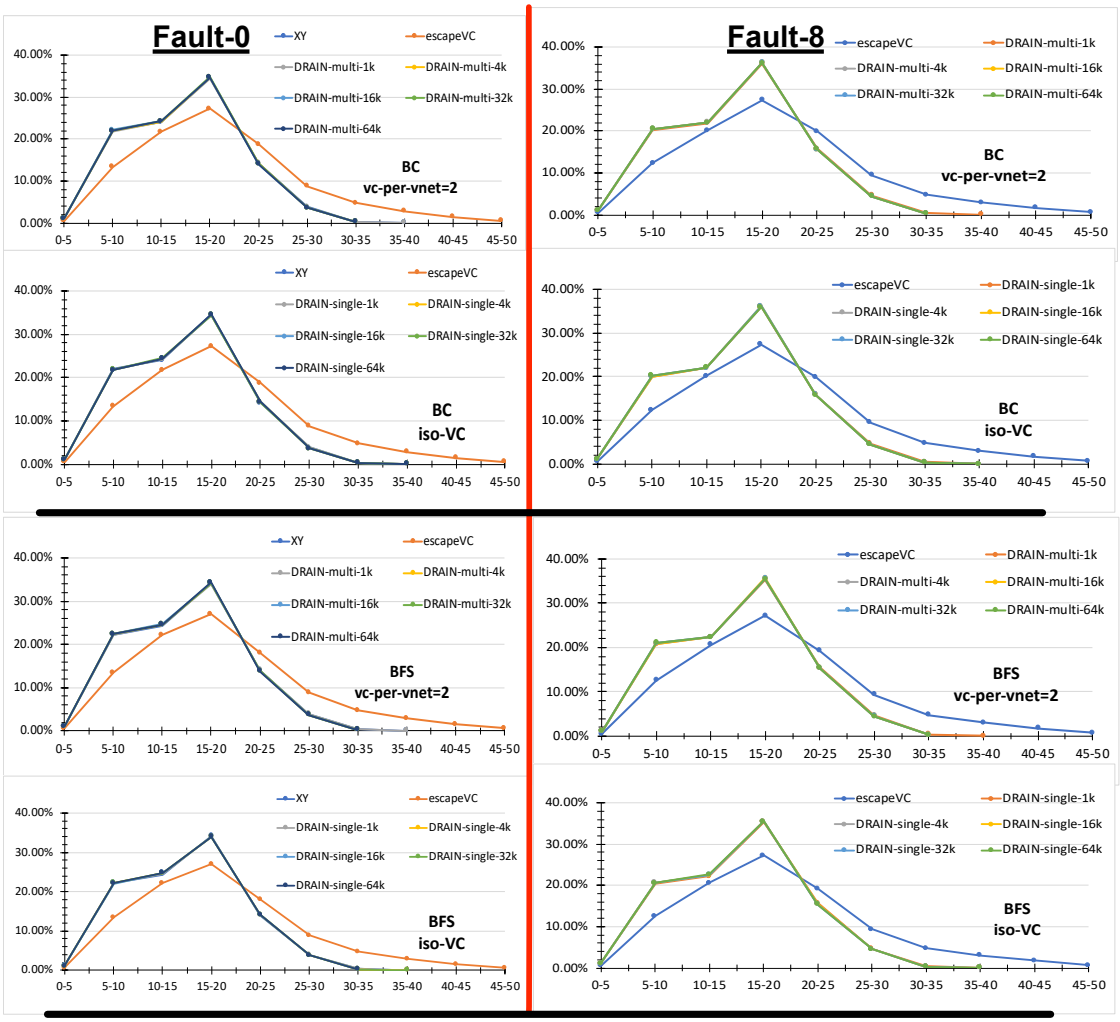


Figure 7.14: Network Packet latency distribution of LIGRA[75] application benchmarks with regular 4x4 and irregular 16 core topology

Figure 7.14 shows the network packet latency distribution of LIGRA[75] applications when run with MESI Two Level cache coherence. The evaluation is done for both regular 4x4 Mesh and irregular topology. The figure, shows that DRAIN does not mis-route packets drastically to change the latency distribution graph towards right. Here *iso-VC* is the configuration which uses *single VNet* in DRAIN with same number of VCs as other sister

schemes. *VC-per-vnet* configuration uses *same number of VNet (3)* in DRAIN as in other sister schemes.

Its peak coincides with minimal XY routing in a regular 4x4 Mesh and for irregular topology it provides higher performance compared to non-minimal UP*/DOWN* routing used in the escape VC.

7.6 Chapter Summary

In this chapter, we propose a new technique of deadlock-freedom technique for providing *subactive* deadlock freedom in interconnection networks. Traditional, well-studied approaches are either proactive, i.e., deadlock is avoided by design through turn restrictions, or reactive, i.e., deadlock is detected and recovered from. In contrast, a subactive approach periodically sweeps away potential deadlocks. we propose DRAIN, a low-cost mechanism to flush potentially deadlocked packets from their current location, which is unique in its ability to eliminate both routing-level and protocol-level deadlocks simultaneously. This new approach leverages the critical observation that deadlocks are rare in practice. While it is imperative that we handle deadlocks, we need not devote extra resources (VCs and virtual networks), extra complexity (detection and recovery mechanisms) nor reduce nominal operating performance (turn restrictions). DRAIN solves this problem with lower complexity and can be implemented with minimal VCs and virtual networks. Finally, DRAIN can be reconfigured to handle random hard faults, thus increasing the usable lifetime of interconnected many-core architectures.

CHAPTER 8

SYNCHRONIZED WEAVING OF ADJACENT PACKETS FOR NETWORK DEADLOCK RESOLUTION(SWAP)

An interconnection network forms the communication backbone in both on-chip and off-chip systems. In networks, congestion causes packets to be blocked. Indefinite blocking can occur if cyclic dependencies exist, leading to deadlock. All modern networks devote resources to either avoid deadlock by eliminating cyclic dependences or to detect and recover from it.

Conventional buffered flow control does not allow a blocked packet to move forward unless the buffer at the next hop is guaranteed to be free. We introduce SWAP, a novel mechanism for enabling a blocked packet to perform an in-place swap with a buffered packet at the next hop. We prove that in-place swaps are sufficient to break any deadlock and are agnostic to the underlying topology or routing algorithm. This makes SWAP applicable across homogeneous or heterogeneous on-chip and off-chip topologies. We present a light-weight implementation of SWAP that reuses conventional router resources with minor additions to enable these swaps. The additional path diversity provided by SWAP provides 20-80% higher throughput with synthetic traffic patterns across regular and irregular topologies compared to baseline escape VC based solutions, and consumes 2-8$\times$ lower network energy compared to deflection and global-synchronization based solutions.

SWAP is another nugget in the gamut of subactive schemes proposed thus far, and an important one. As we will see the swap operation is fundamentally local operation compared to DRAIN which moves all the packets in the network. Unlike DRAIN, SWAP does not need embedded ring covering the whole topology. Table 8.1 qualitatively compares SWAP against the prior work and subactive techniques proposed so far.

Let's study this subactive technique in more detail.

We focus on schemes that provide full path diversity, and *resolve* deadlocks that have occurred. We characterize prior work on deadlock resolution via the following taxonomy:

- Deadlock resolution via **escaping** (e.g., escape buffers [38, 47]); the key drawback is the need for extra buffers;
- Deadlock resolution via **misrouting** (e.g., deflection routing [42, 45, 44]); the key drawback is increased energy consumption and loss in throughput;
- Deadlock resolution via **coordinating or synchronizing** (e.g., SPIN [41]); the key drawback is expensive global coordination for detection and spinning.

None of the state-of-the-art deadlock-freedom solutions (avoidance/recovery) provide full path diversity and high-throughput without requiring deadlock detection or global coordination for arbitrary topologies. This motivates our work. Going back to first principles, we argue that a network deadlocks because a packet is indefinitely blocked; the reason is a fundamental network design rule that says that a packet should be forwarded from an upstream router to a downstream router *if and only if* the downstream buffer is free (which is known via credits/on-off signaling), or is guaranteed to become free by the time the packet arrives [77, 41]. If this rule is violated, packets may be dropped.

We question this fundamental rule and propose *Synchronized Weaving of Adjacent Packets (SWAP)*. SWAP introduces a new deadlock resolution paradigm: **backtracking**[1] , where deadlocked packets yield their position and allow other packets to move forward. Backtracking is performed via *in-place packet swaps* across buffers in neighboring routers. Figure 8.1 shows the conceptual idea. Unlike software, where a swap requires additional temporary storage, in hardware, an in-place swap is conceptually the same as a cyclic shift register. Intuitively, SWAP allows any blocked packet to make *guaranteed* forward progress to its destination, regardless of the congestion in the network, via the mechanism of swaps. More formally, we prove that performing swaps at periodic intervals ensure that

[1] We acknowledge Dr. Joshua San Miguel for coining the term 'backtracking'

Table 8.1: **Summary Table of Qualitative Comparison of Deadlock Freedom Mechanisms. P: Proactive, R: Reactive. S: Subactive.**

Techniques	Full Path Diversity	No Detect deadlock	No Misroute	No Extra Buffers	Routing deadlock freedom	Protocol deadlock freedom
Dally's theory/Acyclic CDG (P) [26]	✗	✓	✓	✓	✓	✗
Duato's theory/Escape VC (P) [52]	✗*	✓	✓	✗	✓	✗
Bubble [37, 50] (P)	✓	✓	✗	✗	✓	✓$^{+}$
Deflection (P) [42]	✗**	✓	✗	✓	✓	✓
Deadlock Buffers (R) [49, 47, 65, 38]	✓	✗	✓	✗***	✓	✓***
Coordination (R) [41]	✓	✗	✓	✗****	✓	✗
BBR (S) [8]	✓	✓	✓$^{++}$	✓	✓	✗
BINDU (S) [9]	✓	✓	✗	✓	✓	✗
DRAIN (S) [10]	✓	✓	✗	✓	✓	✓
SWAP (S) [11]	✓	✓	✗	✓	✓	✓

* Within escape VCs: limited path diversity + requires topology information for escape path.

**At low-loads, full path diversity is available. But at medium-high loads, packets cannot control the directions or paths along with they are deflected.

***DISHA [47] uses timeout counters present at each input port to choose a packet to eject from the network. It requires a set of extra buffers to route the packet involved in deadlock. Some variations of DISHA, such as mDISHA [49] provide protocol-level deadlock freedom

****SPIN[41] requires a buffer in each router to hold the dynamic deadlock path over which packets involved in deadlock would move synchronously.

$^{+}$ Bubble Coloring [50] provides protocol-level deadlock freedom but involves non-minimal path traversal.

$^{++}$ BBR provides limited misrouting of packet because of *Bubble Exchange* subsection 5.3.1

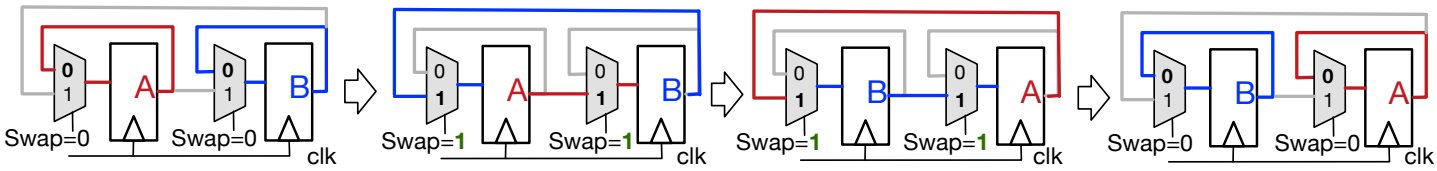


Figure 8.1: The basic hardware implementation for swapping the content of two Flip Flops / FIFOs.

any cycles, if they form in the network, are broken dynamically via swaps, guaranteeing deadlock freedom. Further we show that despite infrequent misrouting of packets, SWAP is free from livelock.

This chapter makes the following contributions:

- We propose SWAP, a novel mechanism for in-place packet swaps across routers in a network.
- SWAP provides deadlock-freedom via packet backtracking, while providing desired metrics of full path diversity, high throughput, no additional VCs, no deadlock detection, and topology agnosticism. Table 8.1 qualitatively contrasts SWAP with current deadlock-freedom solutions.
- We present a light-weight implementation of SWAP that adds 4% area overhead over a state-of-the-art VC router.
- SWAP increases throughput by 20-80% with synthetic benchmarks for a full and faulty mesh, compared to escape VCs, and reduces network link activity by 2-8$\times$ compared to deflection and synchronization-based schemes.
- We show that using SWAP with conventional deadlock-free routing enhances network throughput by 10% since swaps allow packets to move away from congested parts of the network. Thus SWAP *emulates the behavior of VCs* without adding any additional buffers, making it applicable beyond deadlock resolution.

8.1 SWAP Theory

In this section, we present the theoretical underpinnings of our SWAP scheme. First, we provide necessary definitions for the reader, overview the basic operation and provide a concrete walk-through example. Then we provide a proof of deadlock and livelock freedom. section 8.6 then presents one possible realization of our SWAP theory.

8.2 Definitions

Deadlock: Deadlock occurs when packets remain inside the network indefinitely and never reach their destination. Packets wait forever to acquire the buffer at next router due to a *cyclic dependency* between the buffers.

Forward Progress: We use forward progress for a packet to refer to a scenario where it moves towards its destination.

Backtrack: We refer to backtracking for the packet that yields its position and moves back to the upstream router during the swap.

Swap: A swap refers to the act of interchanging two packets from two adjacent routers. The high-level idea is shown in Figure 8.1. *A Swap requires no additional buffers*. It leverages the bi-directional links between adjacent routers to *simultaneously* send two packets (serializing the flits over the link) in either direction.

swapFwd packet: During a swap, the initiator (a.k.a. upstream router) chooses and routes the swapFwd packet towards the productive direction to the downstream router. This packet makes *forward progress.*

swapBack packet: The packet backtracked from the downstream router to the initiator to allow *swapFwd packet* to sit in its place. The *swapBack packet* acquires the buffer of upstream router which was held by the *swapFwd packet*.

swapCycle: The cycle during which a specific upstream router performs a swap of one of its buffered packets.

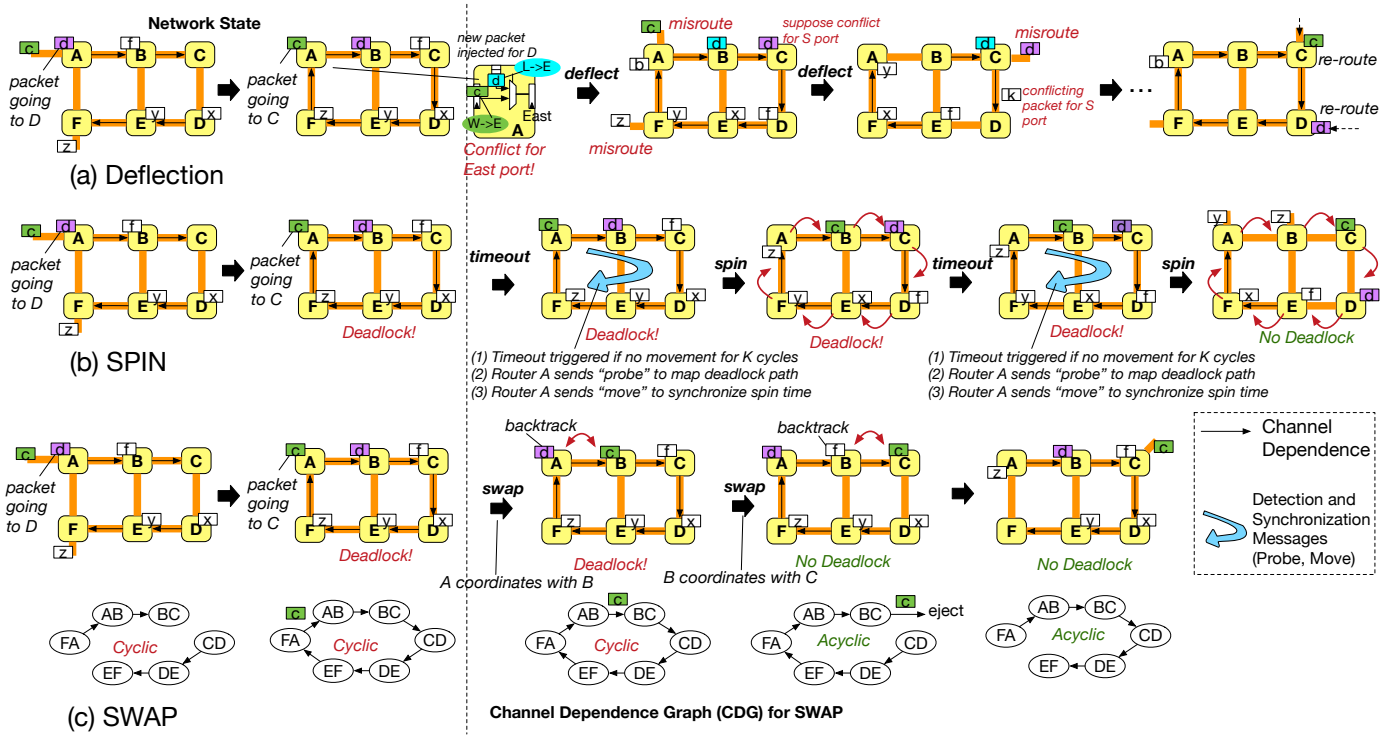


Figure 8.2: Example comparing (a) Deflection, (b) SPIN and (c) SWAP using a 3×2 mesh. The left side of the figure (before dotted line) sets up the same initial condition of deadlock in the three designs, and the right side demonstrates how they operate. In deflection routing, the deadlock does not persist as packets move every cycle. However, the green packet (going to Router C) and the purple packet (going to Router D) are both misrouted due to conflicts, and take multiple cycles to be re-routed to their destinations. In SPIN, if a packet in a specific VC (e.g., at Router A) does not move for a specified number of cycles, a timeout occurs, and a probe is sent to map the possible deadlock path. The probe returns after 12 cycles. A move message synchronizes all routers on the deadlock path to perform a spin. Once the move returns, the spin is performed, and every packet moves forward one hop. The deadlock still persists, so the timeout, probe, move, and spin process repeats. In the last step, packet c reaches its destination and the deadlock is resolved. In SWAP, Router A (at a fixed period), coordinates locally with its neighbor (Router B) and performs a swap: packet d is backtracked, and packet c moves forward. The deadlock still persists. Packet c performs another swap, reaches its destination, and the deadlock is resolved. The corresponding CDG at every step in SWAP is also shown.

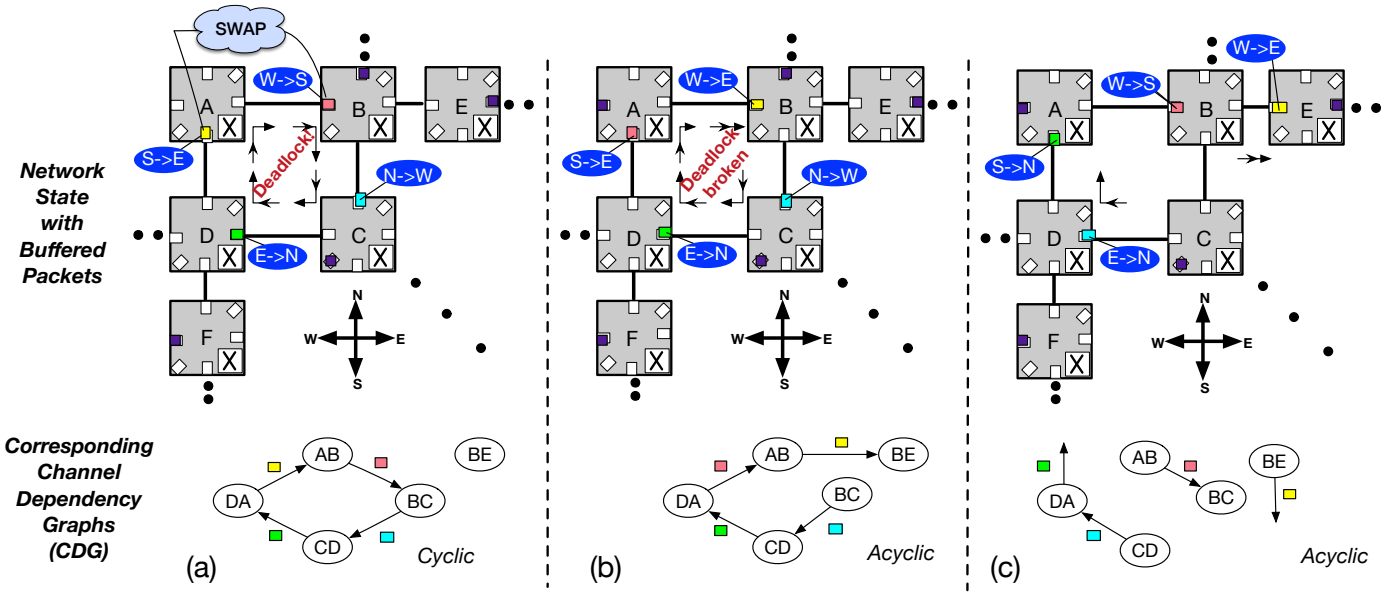


Figure 8.3: Walk through example of SWAP with corresponding CDG. Each node in the CDG represents a link (e.g., node '*AB*' is the link from *router-A* to *router-B*) and each edge represents a packet that wants to turn from the source link to target link (e.g., '*AB*' to '*BC*' represents the pink packet currently buffered at router-B making a West to South turn). (a) there is a deadlock between the four packets as seen by the cyclic CDG. A swap is initiated by router-A between the yellow packet at A with the pink packet at B. (b) The swap completes. Now the yellow packet (swapFwd) moves to B and wants to go East, while the pink packet (swapBack) is *backtracked* to A. The CDG is acyclic: the deadlock is broken. (c) All packets move forward via normal operation.

swapPeriod: $swapPeriod$ refers to the number of cycles it takes for the same router in the network to try and initiate a swap for one of its buffered packets.

8.3 Assumption

In SWAP we assume that the *local* port of the router (injection input port) has separate VNet for each message class, this assumption makes sure that response packet gets to enter the network irrespective of current occupancy of network buffers.

8.4 Proof of Routing Deadlock Freedom

Theorem 1: *In a deadlock cycle of length n, at most $n-1$ swaps by a specific packet are sufficient to break the deadlock.*

Proof: A swap operation removes one edge from and adds one new edge to the runtime CDG. The edge that is removed is the original direction the swapBack packet intended to go towards, while the edge that is added is the new direction the swapFwd packet intends to go towards. $n-1$ swaps allows the packet to reach the node one-hop behind it. One of these $n-1$ intermediate routers will either be its final destination, or an exit point out of this ring. In either scenario, the swapFwd packet will no longer have an edge in the CDG along the original dependence cycle, thereby breaking the deadlock.

Example: Figure 8.3 shows the network state along with the channel dependence graph (CDG) at each step. For the sake of simplicity, we show one VC per port, and single-flit packets. In Step (a), four packets are in a deadlock, as is also evident by the cyclic CDG. Suppose Router A chooses the yellow buffered packet as its swapFwd packet. The yellow packet wishes to go East making the pink packet at the downstream router the swapBack packet. A sends a swap request to B, receives an ACK, and the swap is executed. In Step (b), the yellow packet has made forward progress; the deadlock is broken as it wants to go East. This is also seen from the CDG. The packets now move forward via conventional means, as seen in Step (c). Here, the first swap led to the CDG becoming acyclic, since the new edge was no longer along the original cycle. In a more general case, it may be possible that even after a swap, the CDG remains cyclic (for example, if there is a longer dependence cycle, as shown in Figure 8.2(c)).

Theorem 2: *For a given system which implements SWAP, as long as every packet gets a chance to perform a swap, the network is deadlock-free.*

Proof: A deadlock, by definition, is an *indefinite* blocking of a packet. As long as the implementation of SWAP can guarantee that every packet will have a chance to perform a swap, it will make forward progress, making the network inherently deadlock free. However, it is possible for the packet that made forward progress to later be backtracked by another packet. Indefinite backtracking could lead to a livelock (i.e., the packet never reaches its destination). Next, we discuss why SWAP is livelock free.

8.5 Proof of Livelock Freedom

Theorem: *For a given system that implements SWAP, as long as any backtracked packet eventually has the opportunity to move forward by two hops before being backtracked again, the network is livelock-free.*

Proof: Backtracking can be viewed as incrementing a packet's number of remaining hops h (to its destination) by at most 1. Assume that the system allows a packet to move two hops before being backtracked again (i.e., before being selected as a swapBack packet). For every increment by 1 in h due to backtracking, there is a decrement by 2. This implies that in this system, for any given packet, h eventually goes to 0, guaranteeing forward progress to the destination.

Implementation: There are two important design considerations to implement such a system. First, the swapPeriod must be greater than the time it would take for a packet to move two hops forward in the absence of contention. This avoids any pathological cases of packets continuously being backtracked. Second, the selection scheme that decides which packet should be swapped next must be fair. Ideally, this selection is random; though for practical purposes, round robin is sufficient. A fair selection scheme ensures that no packet is starved in the presence of contention. Even if a packet is not able to move two hops forward now, it will eventually be able to, as guaranteed by the fair selection. Since the network topology is finite, h is upper-bounded by the network diameter (i.e., the largest number of hops between any two routers). Thus, no amount of contention in the system can cause a packet to be backtracked indefinitely.

8.5.1 SWAP in Arbitrary Topologies

Arbitrary topologies are challenging for popular deadlock-avoidance solutions such as XY/West-first routing algorithms; routing algorithms now require CDG analysis to determine topology-specific turn restrictions (for all paths or within escape VCs). In contrast,

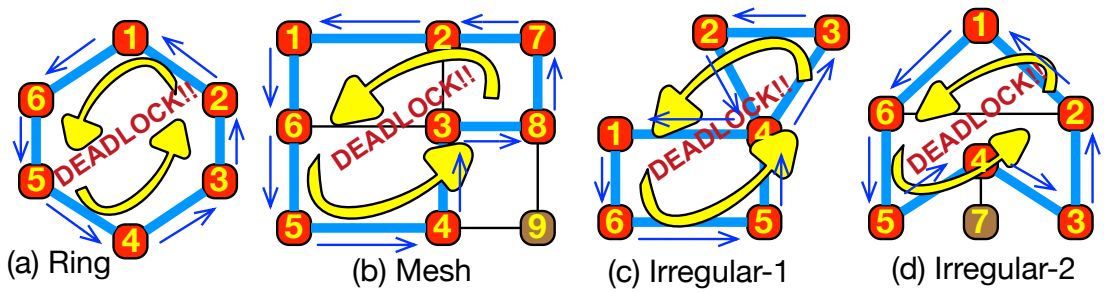


Figure 8.4: Examples of Deadlocks in Arbitrary Topologies

SWAP is agnostic to the topology as any swap just involves neighboring routers. For example, the deadlock ring in Figure 8.2 could lie within any topology in Figure 8.4 and use the same mechanism of swaps for deadlock freedom. SWAP is also agnostic to the underlying routing algorithm. The routing algorithm decides the output port (i.e., neighboring router) of the swapFwd packet, and the swapBack packet is chosen from the corresponding input port at the neighbor.

8.6 Swap Implementation

Multiple implementations of SWAP are possible. We favor an implementation with low complexity. We describe the possible design space and the intuition behind our given design choices, acknowledging that alternate implementations are possible.

8.6.1 Initiating a Swap

Although it is possible to map out the full deadlock loop at runtime via timeout and probes [38, 41], and then perform controlled swaps to recover from the deadlock (Proof 1 in section 8.4), we prefer a less expensive approach. Recall that any SWAP implementation ensures deadlock and livelock freedom if it ensures that (a) every packet gets the chance to make forward progress via a swap, and (b) the system allows a packet to move two hops in an uncongested scenario, before being backtracked. To ensure (a), we enforce periodic swaps by every router at a configurable time period (swapPeriod) and we add a

pointer in every router to cycle through all VCs at all ports that decides which VC will try and initiate a swap. To ensure (b), we need to account for the worst case delay for a packet in two adjacent routers without any stalls due to insufficient credits. This would be a packet in a VC contending with all other VCs at that router for a specific output port, followed by traversing the router and link, and repeating the same at the next router. Thus,

$$swapPeriod \geq 2 \times (\textit{\# ports} \times \textit{\# vcs/port} + (\textit{router\_pipeline\_delay} + \textit{link\_delay})) + \textit{serialization\_delay} \tag{8.1}$$

This works out to be 54 cycles for a 5-ported mesh router with 4 VCs per port, 5-flit packets, 4-cycle routers and 1-cycle links, and 18 cycles for a 1-cycle, 1 VC per port mesh router.

In our implementation, each router performs a swap during its **swapCycle**. The swapCycle is defined as

$$(cycle/m)\%(K \times N) == router\_id \tag{8.2}$$

where K is a configurable *swapDutyCycle*, N is the number of routers in the network, m is the maximum number of flits of any packet in the system and $K \times N$ is the *swapPeriod*. K determines how often each router initiates swaps; the lower the value of K (minimum could be 1), the more *swaps* performed in the network. When K = 1 and m = 1, each router initiates a swap every N cycles in a TDM manner. In a 64-core system, this means that even with K = 1, each router attempts a swap once every 64 cycles, which is greater than the minimum swapPeriod calculated above for livelock avoidance.

The router initiating the swap, as dictated by its ID and current cycle, is the *upstream router* and router with which it will swap its packet, is the *downstream router*. At any given cycle, by design, there can only be one upstream router and several possible downstream routers depending on the topology.

During the swapCycle, the upstream router selects a swap-Fwd packet from one of its internally buffered packets. It sends a swap request signal via a 1-bit wire to the downstream router at the output port for this packet (which is determined by the routing algo-

Table 8.2: **SWAP Operation Details.**

Updating swapPointer	Conditions for Failed Swaps
* When the packet pointed by *swapPointer* leaves the router naturally by winning switch arbitration, the swapPointer moves in round-robin fashion to the next non-empty VC. * When a swapFwd packet arrives at this router from an upstream router via a swap. This packet now becomes the swapFwd packet to give it the highest priority at the next swapCycle in case it does not leave naturally.	* At least one of the VCs within the virtual network of the *swap_req* is empty. In this case, the packet could arrive by normal means, and a swap is not required. * In virtual cut-through routers, if the candidate swapFwd and swapBack packet is distributed across two routers, a swap is not performed. In wormhole, this condition leads to packet truncation [42, 45].

rithm). The downstream router selects a swapBack packet and sends an ACK. section 8.7 details how the swapFwd and swapBack packets are selected. Upon receiving the ACK, a swap is executed over the next m cycles (for m-flit packets at maximum) with both routers sending their respective packets out at the same time to each other over the respective unidirectional links connecting them. These links are reserved for the swap by the ACK and are not allocated to any other packets by the switch allocators at the two routers.

A successful swap can only be initiated when all the input buffers of both upstream and downstream routers are occupied. If this condition is false, either the swap request is not sent, or it is NACK'd. Other conditions for NACK'd requests are discussed in subsection 8.7.2.

Deadlock Resolution Time Trade-off. Since deadlocks are rare [38, 41], our implementation allows only one packet swap in the system at any time. This reduces the number of bactracked packets, reduces complexity and eliminates any race conditions that may arise if the same router is both trying to initiate a swap as an upstream router, and acknowledge a swap request as a downstream router. We can tune the rate of deadlock resolution by tuning the swapPeriod. It is possible to have implementations that allow multiple routers in disjoint parts of the network to perform swaps concurrently, or have implementations that detect deadlocks and perform controlled swaps to resolve it, at the cost of more overhead.

8.7 Selecting the packets to swap

8.7.1 Selecting the *swapFwd* packet

Every router has a *swapPointer*. *swapPointer* is valid when there is packet present in any of its input VCs; it points to that VC. At the onset when there are no packets present in the router, *swapPointer* is invalid. If multiple packets arrive at different input ports of the router in the same cycle, the *swapPointer* becomes valid and randomly points to any of the input VCs containing the recently arrived packets. Conditions for further updating *swapPointer* are detailed in Table 8.2.

The packet sitting in the *swapPointer* VC is the *swapFwd* packet[2] to be sent to downstream router at the swapCycle of *this* router. The downstream router is chosen by the next hop router in the minimal path to this packet's destination based on the routing algorithm.[3] However, if the packet is due to be ejected, it cannot be the *swapFwd* packet, as that would violate the basic requirement of SWAP to have swapFwd packets always make forward progress towards their destination. Thus, the *swapPointer* will move in a round-robin manner to the next non-empty VC with a packet wishing to use a non-ejection port. If no such VC exists, it becomes invalid. If a valid swapFwd packet exists, the router initiates a swap by sending a *swap_req* to the downstream router. The *swap_req* carries the VC ID as explained in subsection 8.7.2. If it is ACK'd, the swap operation is setup for the next cycle. A full swap operation thus takes 4 cycles: (i) req from upstream, (ii) conditions check at downstream, (iii) ACK, and (iv) swap. These are pipelined; each cycle there can only be one router performing a swap, as mentioned in subsection 8.6.1.

[2]More complex solutions to select swapFwd packets can be devised if QoS is needed.

[3]For example, a fully random routing algorithm might pick the next hop based on some congestion metric such as available credits.

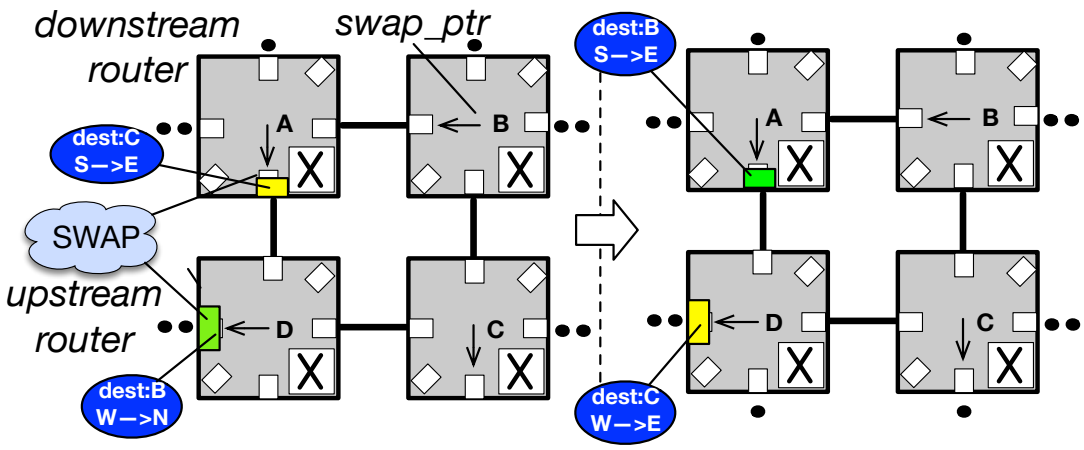


Figure 8.5: Example showing that it is possible for both the swapFwd (green) and swapBack (yellow) packets to make forward progress towards their destinations (B and C respectively) after a swap, due to path diversity in the underlying topology

8.7.2 Selecting the *swapBack* packet

Upon receiving a *swap_req* from an upstream router, the downstream router selects a *swapBack* packet at the input port connected to the upstream router and respond with a *swap_ack*. The swapBack packet is selected from any VC within the protocol message class (inferred from the VC ID sent by the upstream as part of *swap_req*). For simplicity, we select the packet with the same VC ID as the swapFwd packet. Table 8.2 details the conditions under which the swap is NACK'd (i.e., swap_ack is sent back as 0). **Backtracking.** It may appear that the swapBack packet always moves away from its destination. This is not always true. Both the swapFwd and the swapBack packet could move closer to their destinations (i.e., make forward progress), as shown in Figure 8.5, due to path diversity in the system.

U-turns. A swapBack packet effectively makes a u-turn, and will request to go back to the router it was swapped from (unless the routing algorithm finds an alternate minimal path for it). After the swap, it will move again either via regular switch allocation or via a swap (once it becomes the swapFwd packet). It is possible for it to backtrack again in the next *swapPeriod* without moving forward due to either of these conditions. But this will never happen indefinitely, as proven in section 8.5.

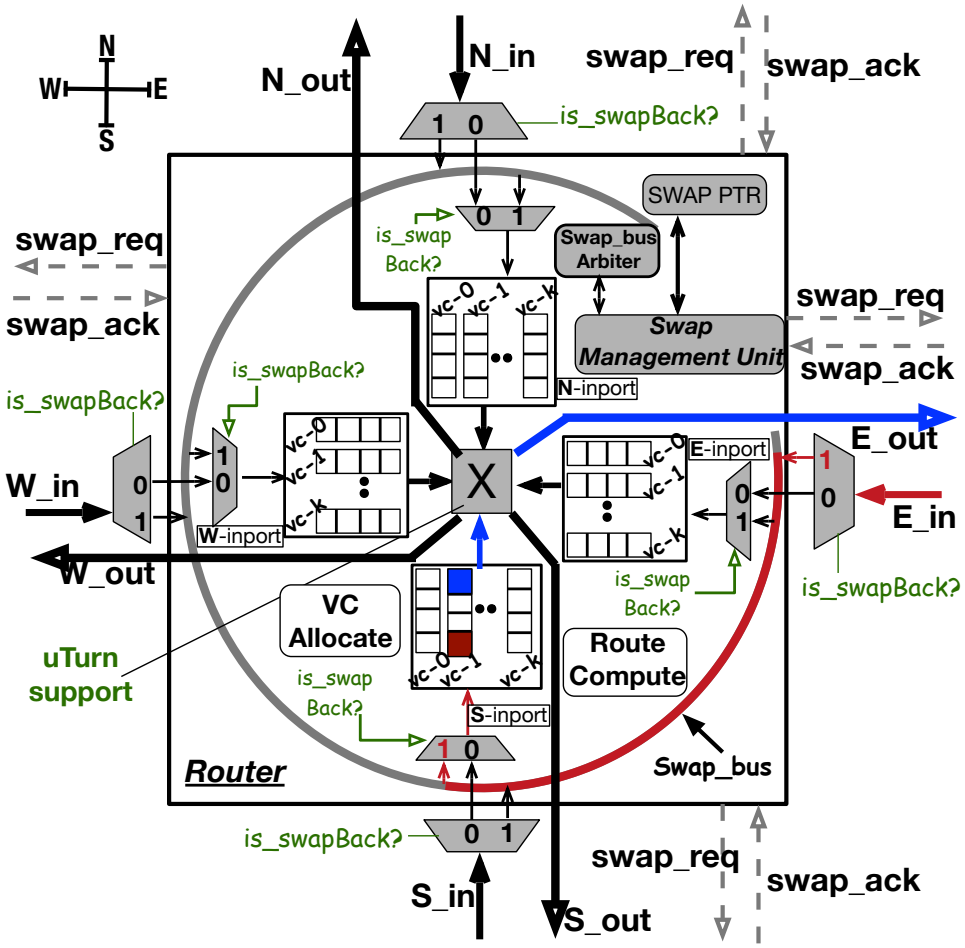


Figure 8.6: SWAP Router Microarchitecture. Features added by SWAP are shaded in grey. Datapath: bus connecting all input ports to allow a swapBack packet from the downstream router to get buffered at any input VC, and u-turn support in the crossbar. Control path: Swap Management Unit controlling when and what to swap. The blue and red paths show a swapFwd packet going from South in port to East out port, and a corresponding swapBack packet entering from East out port and getting buffered in the South in port.

8.8 Router microarchitecture

Figure 8.6 shows the microarchitecture of the SWAP router. We show a mesh router for simplicity, though the same idea works for a router with any number of ports.

Datapath. We assume bi-directional links. A swap operation requires both a forward path and a backward path to be setup between the upstream and downstream routers (red and blue paths in the Figure 8.1) to swap the swapFwd and swapBack packets between their respective VCs. The additions to the conventional router are quite minimal: one 2:1 mux and one 2:1 demux in front of every input port, u-turn support in the crossbar, and a bus connecting all input ports.

As an example, suppose the swapFwd packet is at the South input port Router A, and the swapBack packet is at the West input port at Router B (Figure 8.3(a)). For generality, suppose that their current VC IDs are #1 and #3, respectively (even though our implementation restricts the swaps to occur within the same VC ID).

- **forward path (A_South_VC$_1$ to B_West_VC$_3$):** the swapFwd packet reuses A's crossbar to traverse to B's West input port (blue path in Figure 8.6) and gets buffered into VC$_3$. This is exactly like a regular traversal. As mentioned earlier, during the swap, the output link from A to B is not allocated to any other packet.
- **backward path (B_West_VC$_3$ to A_South_VC$_1$):** the swapBack packet reuses B's crossbar to make a u-turn towards A. The swapBack packet arrives at the *East* input port at A, but needs to be buffered at the South input port. The pre-set *swap_bus* transports the packet from East to South, and buffers it in VC$_1$ (red path in Figure 8.6).

Why is a simple bus sufficient? SWAP does not support multiple swaps in the same cycle. Thus, we do not need a crossbar at the input of the router to support multiple swaps from multiple downstream routers simultaneously. The *swap_bus* is pre-configured by the *swap_ack*. This makes the SWAP implementation extremely light weight.

Virtual Cut-Through (VCT) and Wormhole Implementations. VCT routers have buffers

Table 8.3: **SWAP vs. Deflection Routing**

	Deflection Routing	SWAP
Mis-routing	Forces packet deflections upon buffers overflow [44] (every cycle in case of bufferless designs [42]) without support for stalls. This leads to high misrouting and congestion.	Provides localized misrouting, which we call *backtracking*, the rate of which can be controlled using the *swapPeriod* parameter.
Spread	Deflections in one part of the network can trigger deflections in another part, leading to high latencies and dropped throughput for all packets.	Backtracking is controlled by swapPeriod and swapCycle parameters.
Router Ports	Indirect restriction on the router micro-architecture: number of input ports must equal the number of output ports of the router.	Places no restrictions on the router's radix, making it more amenable to arbitrary irregular topologies.
Router Critical Path	High hardware overhead for switch-arbiter to perform the best matching upon packet conflict. This also lies in the critical path of the router.	Adds minimal changes to the baseline router micro-architecture (Figure 8.6), with the SMU operating off the critical path.
Routing	Deflection routing algorithm is a de-facto routing algorithm, controlled purely by current network congestion	Any routing algorithm (minimal/non-minimal/adaptive) which by itself may or may not be deadlock-prone.

deep enough to hold an entire packet. This design naturally works well for SWAP. Swaps are only performed once the entire packet is received, as shown in Table 8.2. To support SWAP with wormhole routers, we would add packet truncation support, similar to prior works in deflection routers [42, 45, 44]. Packet truncation occurs on swapFwd and/or swapBack packets if the former initiates a swap. .

Multi-flit Packets For m-flit packets, the swap operation takes m cycles, as the flits are serially swapped.

Control Path. The control path of SWAP adds a Swap Management Unit (SMU) that handles if, when and what to swap, as described in section 8.7.

Table 8.4: **SWAP vs. SPIN**

	SPIN	SWAP
Detection Approach	Maps entire deadlock path upon a timeout using probes that take multiple cycles.	No detection. Performs swaps periodically based on a VC occupancy threshold.
Detection Time	Longer deadlock cycles take longer to map and resolve	Independent of deadlock cycle length
Synchronization	Global: all routers in deadlock must spin at same time	Local: with neighbor who will be performing swap
Resolution Approach	All packets in deadlocked ring move forward simultaneously	Only two packets move simultaneously.
Resolution Time	(N-1) spins in worst case for deadlock of length N	(N-1) swaps of specific packet in worst case for deadlock of length N
Misrouting	None	Backtracks packet one hop

8.8.1 SWAP vs. Deflection Routing and SPIN

Deflection routing [42], SPIN [41] and SWAP all rely on moving packets in the absence of credits; thus, they are similar in their underlying mechanism for deadlock freedom. Figure 8.2 shows an example comparing the three schemes. The key qualitative differences of SWAP versus these are highlighted in Table 8.3 and Table 8.4. Quantitative comparisons are present in section 8.9.

8.9 Evaluation

8.9.1 Methodology

We use gem5 [7]; to model networks with different configurations we use Garnet [59]. Table 8.5 provides the detailed configuration parameters. Our baselines include a mix of state-of-the-art deadlock avoidance (deterministic XY, congestion-aware adaptive west-first, and escape VC), recovery (StaticBubble [38], SPIN [41]), and deflection (CHIPPER [45] and MinBD [44]) techniques. Static Bubble relies on extra buffers added in a subset of routers at design time, which are turned on upon deadlock detection (via probes) to drain deadlocked

packets. SPIN sends probes (upon timeouts) to detect the deadlock dependence ring, and then performs a coordinated forward movement of the entire ring. For full-system simulations, we run PARSEC [33] and LIGRA [75] (a graph processing suite) over gem5's x86 and RISC-V models which have support for running these respectively.

Irregular Topologies. For our evaluations, we derive some irregular topologies by removing links from a mesh which emulates an SoC with heterogeneous-sized cores or accelerators, or a many core where some links are faulty [70, 78], or have been power-gated [72]. In these scenarios, the resulting topology will not longer be able to use a simple turn-model (e.g., XY/west-first) since certain turns are inevitable to reach some of the destinations. Using these restricted turns can lead to routing deadlocks. We use a spanning tree based Up-Down routing algorithm [25, 70, 78] across all VCs, or within an escape VC, as our baseline deadlock avoidance schemes, and SPIN as the baseline deadlock resolution scheme. For irregular topologies, we assume that information about the missing links and the exact routing path (spanning tree vs minimal) is computed offline and embedded into routing tables [70, 78].

8.9.2 Correctness

We start by demonstrating why a deadlock-freedom solution is imperative in any network. Figure 8.7 runs a set of synthetic traffic patterns with fully-random minimal adaptive routing with no turn restrictions. The occurrence of deadlock causes the percentage of delivered packets to drop sharply; the onset of deadlock depends on the traffic pattern, injection rate, and number of VCs. This shows that deadlocks are highly dependent on the runtime network state. SWAP delivers all packets successfully. To the best of our knowledge, SWAP is the first *non deadlock-recovery-based scheme*[4] to provide fully-adaptive random routing with only 1 VC.

[4]SPIN [41] is the first to provide fully random routing with only 1 VC but relies on deadlock detection and recovery.

Table 8.5: **Network Configuration.**

Network	
Topology	8x8 Mesh, Irregular
Routing	Fully-Adapt Random (except when specified)
Latency	Router: 1-cycle, Link: 1-cycle
Num VCs	1, 2, 3, 4
Buffer Organization	Virtual Cut Through Single packet per virtual channel
Deadlock Freedom Mechanism	
Deadlock Avoidance	Mesh: XY, West-first, EscapeVC. Irregular: Up-Down [25]
Deadlock Recovery	Static Bubble [38], SPIN [41] with deadlock-detection threshold=128 cycles
Deflection	CHIPPER [45], MinBD [44]
SWAP	SWAP-K, where K = swapDuty-Cycle
Traffic Pattern	
Synthetic	Bit-Rotation, Bit-Reverse, Uniform-Random, Transpose, Shuffle. Mix of 1 and 5-flit packets
Real Applications	PARSEC [33], LIGRA [75]
System Configuration (for Real Apps)	
Core	64 cores, x86/RISC-V In-Order, Private L1D=32kB, L1I=32kB, Shared L2 (LLC) Slice=128kB
Memory	MOESI Directory Coherence, 4 DRAM Ctrlrs

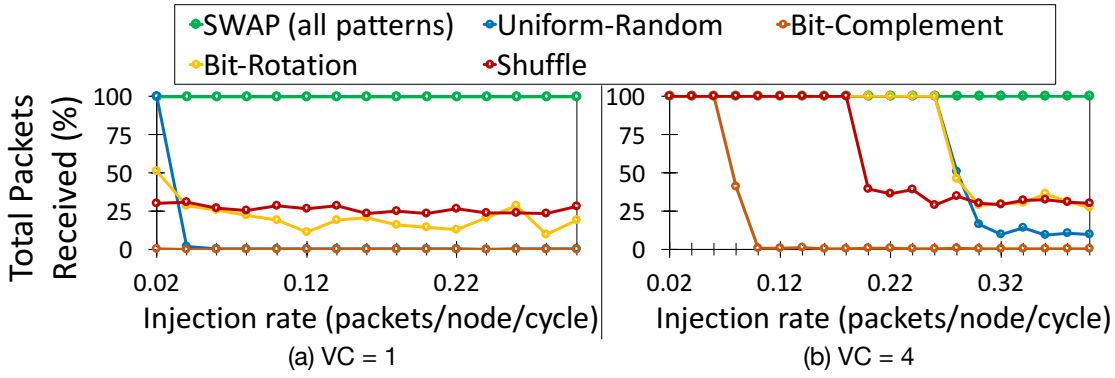


Figure 8.7: Percentage of received packets when running a *fully random* routing algorithm. SWAP delivers all packets, irrespective of the traffic pattern. Without SWAP all traffic patterns see a sharp drop in delivered packets, due to deadlocks. The injection rate when deadlocks start depends on the traffic pattern and number of VCs.

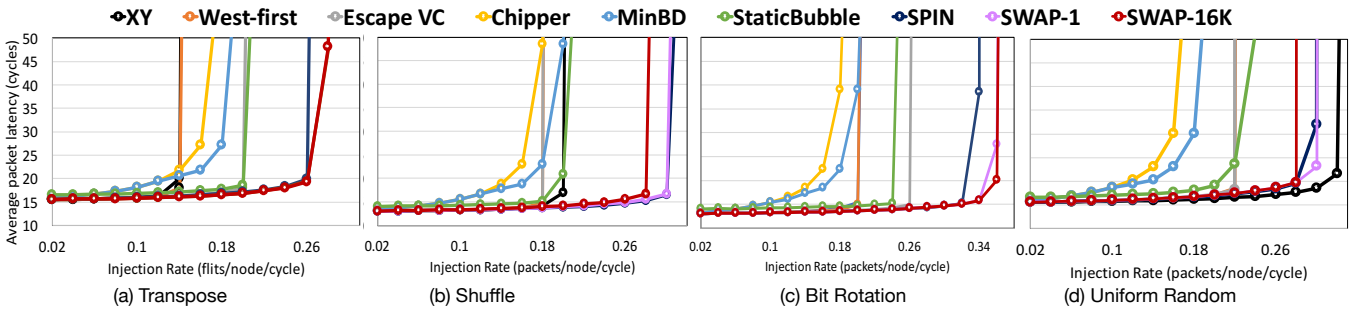


Figure 8.8: Performance of SWAP-K (K = swapDutyCycle) with different traffic synthetic patterns, across deadlock-freedom techniques in a 8×8 Mesh. Num VCs=4. Packet Size = Mix of 1 and 4 flits.

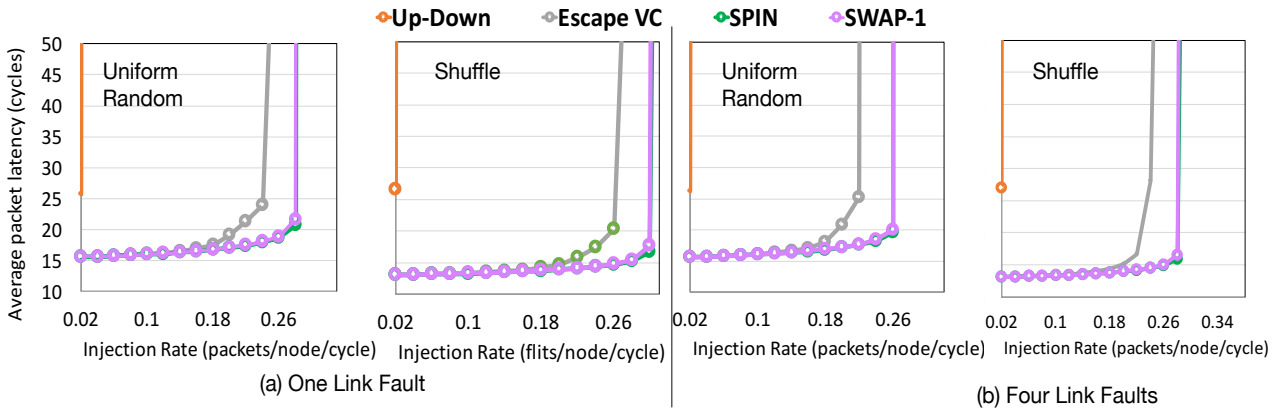


Figure 8.9: Performance of deadlock-free networks over Irregular Topologies.

8.9.3 Performance

Synthetic Benchmarks on a Mesh. Figure 8.8 shows the performance improvement of SWAP over state-of-the-art deadlock avoidance and recovery schemes on a 8×8 mesh. All designs except XY use adaptive routing. SWAP consistently matches or beats SPIN.

Here are some key observations:

- The swapDutyCycle K does not affect the achieved throughput; backtracking does not adversely affect throughput.
- SWAP has a large throughput advantages over CHIPPER, which is known to have low throughput due to deflections. Compared to MinBD, SWAP still provides better throughput because even in the extreme design point of K=1, there is only one poten-

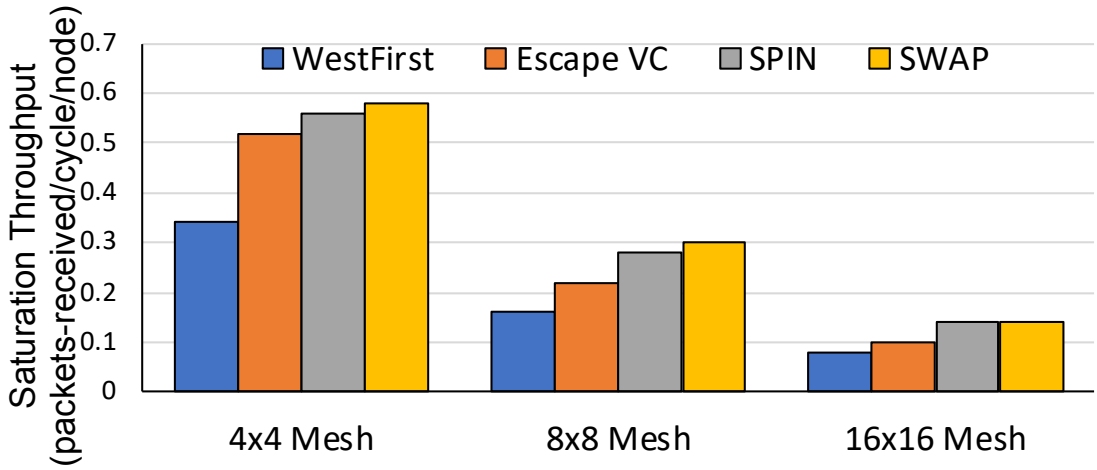


Figure 8.10: Effect on throughput as network size increases for Transpose traffic.

tial backtracking packet every cycle (at high-loads), while MinBD at high loads (and high congestion) will result in heavy deflections once its extra buffer becomes full. We quantify this in Figure 8.14.

- Compared to avoidance schemes, the performance benefits of SWAP come because it can use fully adaptive random routing, with no turn restrictions. In addition, with SWAP packets can swap and leave a congested region, without relying in credit flow. Both these features push throughput.
- Compared to the recovery schemes (SPIN and Static Bubble), SWAP has no inherent advantages due to path diversity – all designs use fully adaptive random routing. However, the reason SWAP ends up beating SPIN for a few patterns is because once deadlocks kick-in (see Figure 8.7), it takes multiple cycles to map out the deadlock path (scales with length of deadlock) and synchronize [38, 41], during which time the network essentially saturates (due to deadlock-driven congestion), leading to loss in throughput. SWAP on the other hand, with periodic swaps with a neighbor, ensures that even if a cycle were to form, it very quickly gets removed.

In summary, SWAP provides robust throughput improvements over both avoidance and resolution-based deadlock-freedom techniques.

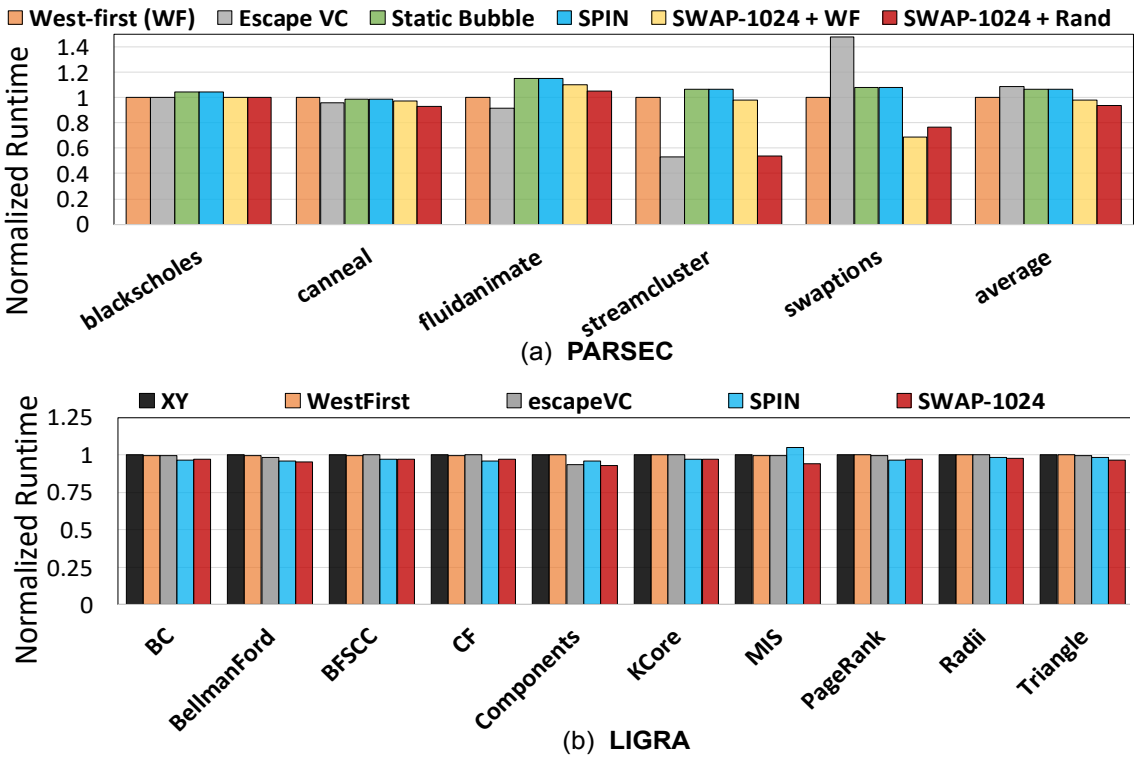


Figure 8.11: Normalized Runtime with Multi-threaded Workloads.

Synthetic Benchmarks with Irregular Topologies. Next, we evaluate SWAP with two irregular topologies and compare it against Up-Down routing (i.e., deadlock avoidance), and SPIN (i.e., deadlock resolution).

These topologies have one and four links removed in an underlying 8×8 mesh. Figure 8.9 plots the latency vs. injection rate for uniform random and shuffle. For irregular topologies, non-minimal routing in Up-Down completely kills throughput. Escape VCs help get some of the throughput back, but still use Up-Down within the escape VC which limits throughput. SWAP gets the same performance as SPIN.

In summary, the performance benefits of SWAP compared to deadlock-free routing algorithms such as Up-Down are magnified when path diversity is at a premium, such as in irregular topologies. Moreover, SWAP matches SPIN without requiring any of the expensive circuitry and signaling overheads for mapping the deadlock cycle and performing global synchronization.

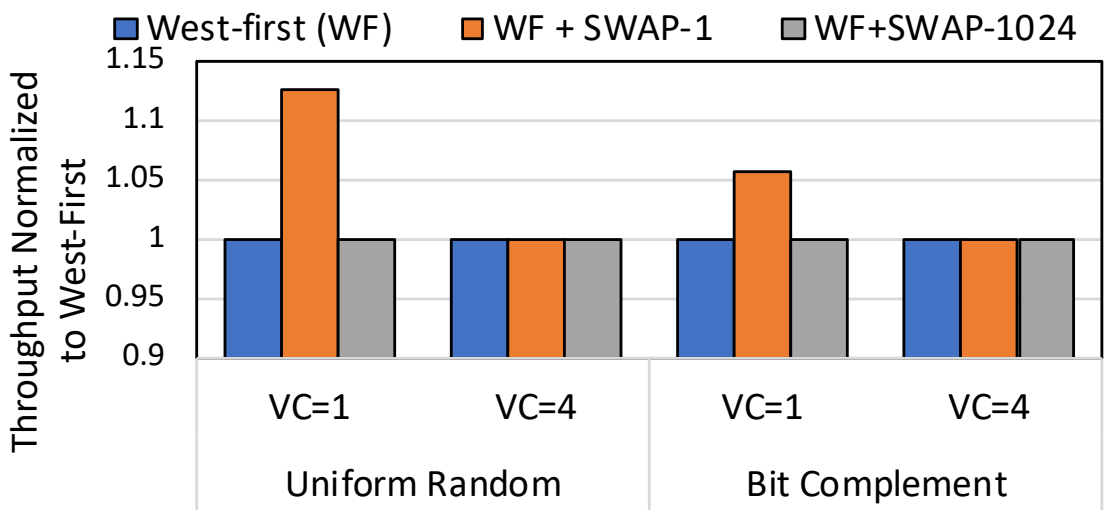


Figure 8.12: SWAP throughput with Uniform Random and Bit Complement traffic running with a deadlock free routing algorithm. SWAP provides throughput benefits, especially at low VC counts, by providing extra path diversity. With high VC counts, it is no worse than the underlying algorithm.

Scalability study on saturation throughput. We compared the effect on saturation throughput as network size increases across state-of-the-art deadlock avoidance and deadlock recovery schemes in Figure 8.10. The analysis is done on the regular mesh, each input port has four VCs in the router (same as Figure 8.8).

In summary, we observe the trends in performance remain consistent - SWAP continues to provide higher throughput. However, the performance difference between schemes decreases as network size increases.

Real Benchmarks. Figure 8.11 compare the normalized runtime of PARSEC and LIGRA across deadlock-freedom schemes.

Real applications do not significantly stress the NoC due to low injection rates; for most benchmarks, all deadlock-freedom schemes fared similarly as the injection rates were quite low. In PARSEC, SWAP shows 30% runtime reduction for swaptions where we saw significant network traffic, and for LIGRA, SWAP provides 2-4% runtime reduction. *This study re-iterates the motivation of deadlocks being rare and probabilistic events, where-in a deadlock-freedom solution is necessary for correctness. SWAP provides deadlock freedom*

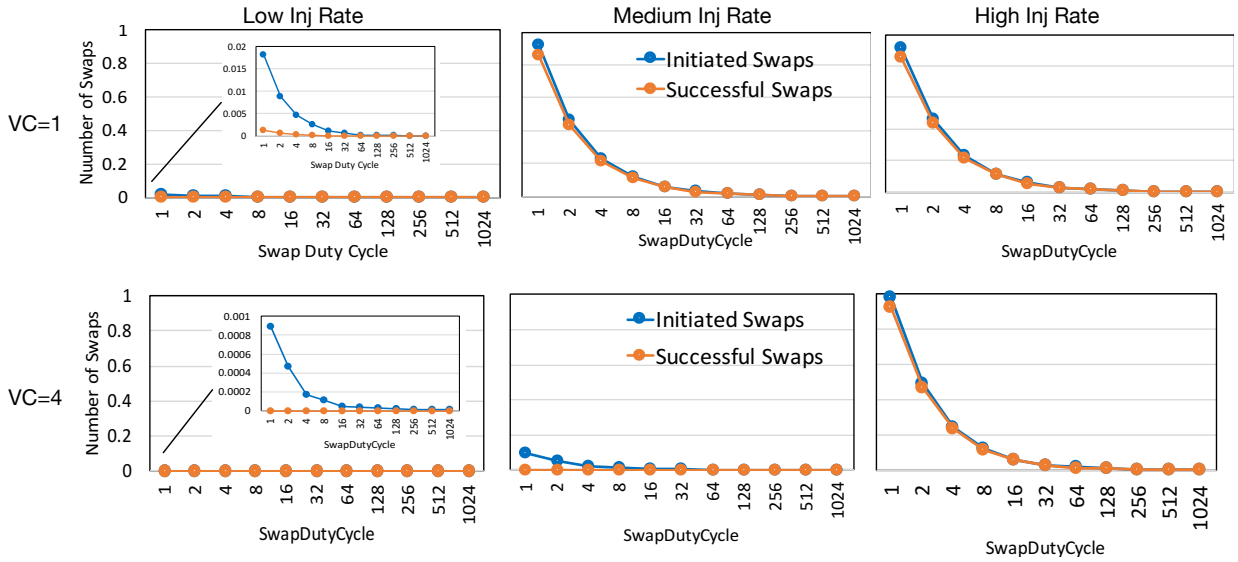


Figure 8.13: Relation between number of initiated and successful swaps per cycle, as a function of SwapDutyCycle for low, medium and high injection rates with uniform random traffic. The top row is for VC=1 and the bottom for VC=4. The conditions for unsuccessful (failed) swaps are discussed in Table 8.2.

without conservative restrictions for a rare event, or expensive circuitry to detect this event.

SWAP as an overlay over deadlock-free routing. SWAP can be overlaid on any network since its basic functionality is to enable packet swaps between neighbors. So far we have focused on the deadlock-freedom capabilities of SWAP, but it can also help reduce congestion due to forced forward progress. We ran SWAP with an underlying west-first deadlock-free algorithm. Figure 8.12 plots the peak throughput for two synthetic benchmarks, with one and four VCs, normalized to the throughput of a west-first system without SWAP. With one VC, SWAP-1 provides a 12% throughput boost for uniform random, and 6% for bit complement. This result can be interpreted as follows: packet swaps emulate the behavior of additional VCs, as they can force packet movement even if downstream packets are head-of-line blocked. With larger values of K and with higher number of VCs, west-first provides similar performance with and without SWAP. Thus, SWAP's backtracking does not adversely affect the underlying system performance. *This study demonstrates that deadlock-freedom benefits aside, adding SWAP to existing routers does not hurt, and can potentially enhance throughput as it emulates the behavior of VCs.*

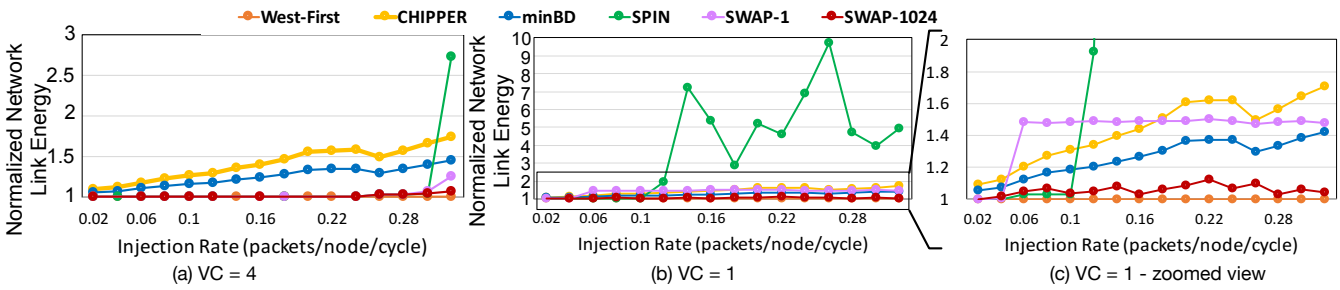


Figure 8.14: Energy (i.e., activity) of links in Deflection, SPIN and SWAP networks with VC=4 and VC=1, normalized to a west-first routing algorithm which as purely minimal routing. SWAP's duty cycle parameter can help limit the amount of backtracking.

8.9.4 Sensitivity Studies

Recall that the swapDutyCycle K controls the rate of swaps in the network. In Figure 8.13, we study the impact of K on the number of initiated and successfully executed swaps. Let us start with a pathological worst case: very high post saturation injection rate (last column), and K=1. K=1 leads to a swap getting initiated every cycle by each router in the network in a round-robin manner. In this case, the number of initiated swaps is close to 1, irrespective of VC count since all VCs are active post saturation. Moreover, nearly all swap requests are successfully executed. For the same K=1, when network the injection rate is lower, or if the number of VCs is high, the number of swaps initiated drops close to zero, since the likelihood of the VC pointed to by the swapPointer being empty becomes very high. At very low injection rates (first column), the average number of swaps initiated per cycle is less than 0.02 for VC=1, and less than 0.001 for VC=4. Moreover, the number of successful swaps is zero, since the input port from where a swapBack packet would have been selected has empty VCs. When K becomes greater than 32, the number of initiated swaps drops close to zero at all injection rates. This study shows that the number of swaps is actually quite low; SWAP is a low-complexity solution for deadlock freedom with low overhead. We quantify this overhead next.

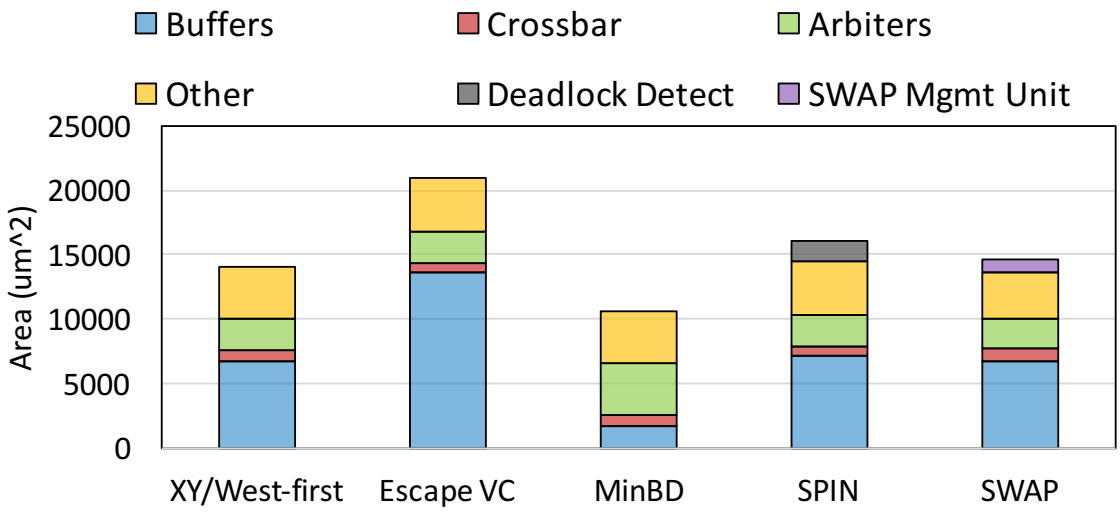


Figure 8.15: Post Place-and-Route Router Area (28nm TSMC, 1GHz).

8.9.5 Overheads

Energy Overhead due to Swaps vs. Spins vs. Deflections. Recall that a swap operation involved two packets: a swapFwd and a swapBack packet. The swapFwd packet makes *forward progress*: it gets read out of its upstream buffer, traverses the link, and gets written into the downstream buffer. These are actions it would have had to take anyway and thus do not contribute any energy overhead. The swapBack packet, however, makes a u-turn and goes back to the router it came from. Its corresponding buffer read, link traversal and buffer write are direct energy overheads. Figure 8.14 quantifies the extra link activity due to swaps for uniform random traffic. With VC=4, the overhead is close to zero with K=1024, since swap requests fail due to free VCs. With an aggressive K=1, the additional link activity starts rising and goes up to 30% post saturation.

In contrast, CHIPPER and minBD start showing higher link activity at the onset of contention even at low loads, and have 40-80% higher link utilization post saturation. Although SPIN does not inherently misroute like deflection routing or backtrack like SWAP, it adds link activity overhead due to its global synchronization messages (probe, move). This leads to 2.8$\times$ higher link activity in the network post saturation due to the incessant number probes that are sent and forked along the way to detect deadlocks.

The link activity behavior at the extreme design point of one VC is interesting. Here, SPIN's probes increase link activity by 8-10×. Deflection routing NoCs (which do not have VCs) have the same 40-80% higher link activity discussed above. SWAP-1024 sees increased activity, but it remains within 10%. The aggressive K=1 configuration sees a jump in link activity once the network starts to saturate, and eventually adds similar energy overhead as MinBD. *This analysis shows that SWAP is a better design choice than both deflection routing and SPIN. Compared to Deflection routing, it provides steady movement that resolves deadlocks, without adding significant energy overhead due to backtracking as its rate can be controlled. It also provides much better energy efficiency than SPIN as it does not require probe broadcasts which consume significant energy.*

Area Overhead. Figure 8.15 plots the area breakdown of the SWAP router compared to XY/West-first, Escape VC, MinBD and SPIN. All routers were implemented using open-source RTL [66] and synthesized and placed-and-routed using TSMC 28nm, targeting 1GHz with 1-cycle pipelines. MinBD, which enhances a bufferless NoC with some buffers, naturally has the lowest area, but comes with misrouting overheads discussed above. All buffered NoCs have 1 VC per port. The escape VC router is assumed to have 2 VCs per port. SPIN's overhead comes due to the synchronization, storage and management of the detected deadlocked loop, that adds about 15% overhead. SWAP has ∼30% lower area overhead than an escape VC router, and ∼4% higher area than a XY/West-first router. The area overhead comes due to the swap management components that were presented earlier in Figure 8.6.

8.10 Discussion

8.10.1 Providing Routing and Protocol Deadlock Freedom using Directed SWAPs

Thus far we discussed one of the ways to resolve routing deadlock by interleaving packet swap operations throughout the network periodically. Here we discuss another approach to guarantee deadlock freedom in irregular topology using swaps. This technique involves the

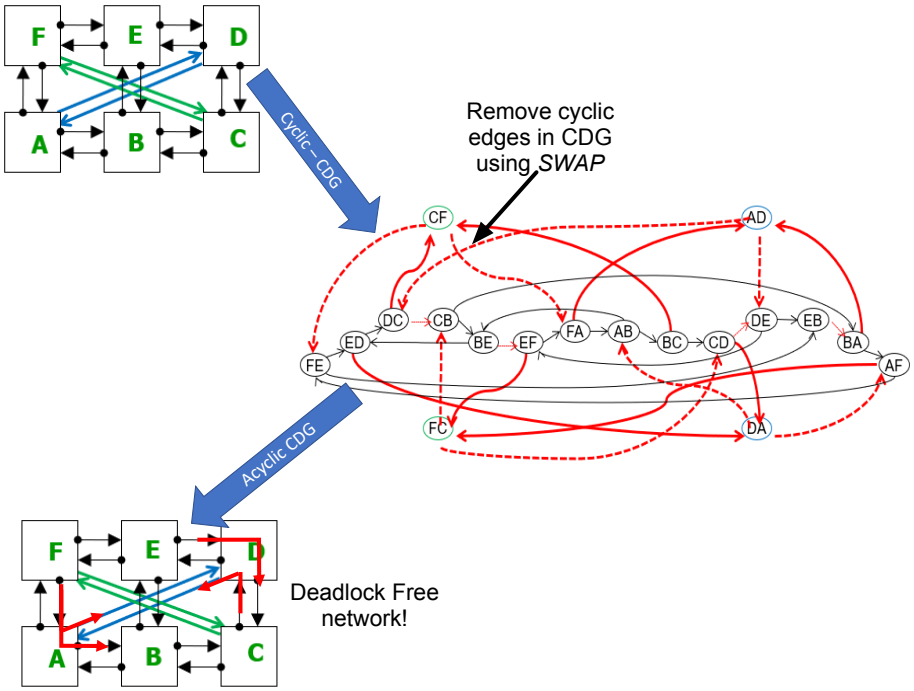


Figure 8.16: *swap* operation essentially removes one edge and adds another in the runtime CDG of the network. SWAP can guarantee deadlock freedom by making sure cyclic edges in the runtime CDG of the network does not persist.

static analysis of the topology and creates the channel dependency graph (CDG) as shown in the Figure 8.16. We can find all the cycles present in the CDG. We can then mark-out those edges in the whole CDG which when removed can make the CDG acyclic. These edges can effectively makes the network to deadlock. Remember that edges in the CDG represent the turn that packet can take as allowed by the routing algorithm.

Now these marked edges can become the candidate over which swap can take place. Swap between two packets from adjacent routers essentially removes one edge and adds another edge in the runtime CDG of the network. If packets are stuck along that particular marked edge in the static CDG, then using SWAP we can allow this packet to make forward progress, until the original marked edge is no longer present in the runtime CDG of the network. Discussion on the specifics of on how many swaps are needed to make sure the original marked edge is no longer present is beyond the scope of this thesis. However, Static CDG analysis combined with swap provides a promising way to make sure certain

edges of CDG responsible for creating deadlock never persists in the runtime CDG of the network. The backtracking property of SWAP essentially allows any blocked packet to be unblocked.

We can further augment *directed SWAP* to prioritize the response packets (*packets from terminating message class of the protocol*) over other packets. This, along with the assumption of separate VNets for each message class at the local port of the router (section 8.3) makes SWAP simultaneously Routing and Protocol level deadlock free.

8.11 Long Term Impact

The growth of multi- and many-core processors places the interconnection network front and center in architectural designs. The interconnection network will dictate overall system performance by efficiently orchestrating data movement. Deadlock is a cornerstone in interconnection networks and will continue to be the focus for its functional correctness. SWAP represents a fundamental contribution to resolve deadlocks. It proposes a new elemental operation, '*swap*', distributed throughout the network.

SWAP views deadlocks as an ordering problem. If any packet can replay back the hop that lead to deadlock, then deadlock can be resolved. We presented one implementation of the concept; there can be many others. For example, multiple swaps can occur concurrently in disjoint portions of the network to further improve scalability.

8.11.1 Salient features of SWAP

- In-place swap of packets, therefore no extra buffers needed.
- Packets are swapped locally between adjacent routers, therefore no global coordination is needed.
- SWAP is free from deadlock detection scheme's design and validation overheads. Instead it has a configurable knob to control the frequency of swapping packets.

- If there is empty VC present at the downstream router, then a swap will not occur. This naturally reduces the extra link traversal energy due to backtracking at low loads.
- SWAP enables adaptive routing algorithms providing full path diversity without worrying about deadlocks.

Even in OS, deadlocks are viewed as resource dependency problem, however, we believe if we could view deadlocks as an ordering problem then novel insights can be extracted and learning from SWAP can be applied directly to these problems.

8.11.2 Going beyond Routing Deadlocks

Learning from SWAP goes beyond resolving routing deadlocks in the interconnection networks. The following are the immediate benefits with SWAP:

Protocol level deadlocks

Protocol level deadlock occur when message of class A gets indefinitely blocked by the message of class B.

To avoid protocol level deadlocks, dedicated sets of VCs are allocated to each message class which cannot be occupied by packets from other message classes. These dedicated set of VCs, known as Virtual Network (VNets), are not only present at each input port of every router but also present at the network interface, in the form of multiple injection and ejection queues. The extra VNet VC imply a dramatic overhead in terms of power, area and complexity, that is typically under-utilized, particularly for rarely used packet classes. As protocols increase in complexity especially due to the rise in heterogeneity in current systems, the number of VNets will also increase leading to very high area network implementations.

SWAP provides a way to re-order the packets in the network. This inherently implies that a packet of one class will never be stuck indefinitely behind the packet of other class. By carefully orchestrating swap operations, SWAP can not only remove routing deadlocks,

but also protocol level deadlock. Thus, SWAP dramatically reduces the power, area and complexity of routers needed to ensure functional correctness of the NoC, irrespective of the cache coherence protocol. These benefits can be realized all the way up to network interface, as now with SWAP capabilities, separate injection and ejection queues need not be maintained for each message class–unified injection and ejection queues would suffice.

SWAP for modular network design

SWAP localizes the decision of swapping packets to adjacent routers and is agnostic to different link widths and disparate link/router latencies co-existing in the same network.[5] This aspect makes it suitable for plug-n-play designs where the topology is not known, has asymmetric properties and/or changes over time for example SWAP is readily implementable in multi-chip modules, chiplet interposer-based systems, off-chip interconnects such as LANs or datacenter clusters. For implementations such as Ethernet that drop packets in the face of congestion, SWAP might provide an alternative that allows lossless transmission of packets by guaranteeing forward progress through swapping. This would avoid overheads associated with retransmission. How scheduling swaps affect the performance across the range of systems is an important research question.

Internet could be a new avenue to implement SWAP, as currently during congestion packets are dropped in internet. This comes with an onerous overhead of keeping a copy of already transmitted message, retransmitting it in case it gets dropped and sending ACKs/-NACKs for keeping track of messages. With SWAP even during congestion the packet can make forward progress by backtracking packets ahead of it.

SWAP for reduced verification cost

SWAP considerably reduces the design and verification cost of the network. Under prior schemes, significant design complexity is required to ensure deadlock avoidance/resolu-

[5]Naturally swapping of packets will take longer if the link width is smaller and/or router latencies are higher (slow swap) compared to normal link-width and router latency (fast swap).

tion, either in provisioning of extra VCs, designing of bespoke routing algorithms for particular topologies, or hardware to detect and resolve deadlocks. With SWAP, one need only implement the same basic swapping mechanism regardless of network topology, routing algorithm, etc and the deadlock problem is solved without further difficulty.

SWAP for improved system efficiency

Quality of service: In SWAP, packets are swapped obliviously to resolve potential deadlock. With the SWAP hardware in place, packets could be swapped in a non-oblivious fashion, such as according to packet priority. Thus, a high priority packet could swap its way through congestion like an ambulance down a congested highway. This would allow designers to layer quality-of-service on top of their network design with negligible additional overhead. Swapping can further be extended into the memory controller queues present at the edge of network. This allows end-to-end QoS from processor pipeline all the way to memory.

SWAP for improved system security

Caches are well-known to exhibit timing side channels. In a NUCA cache architecture, SWAP can be leveraged to mitigate this side channel. Currently, the SWAP period and swap router selection is static; however, this could be easily modified to be stochastic. Such modifications of the SWAP mechanism could obfuscate information about network congestion, bandwidth constraints and even cache block placement often used in side channel attacks.

For example, the QoS used to provide latency guarantees can delude the adversary about cache-block placement and defuse his/her attack. We envision QoS combined with cache replacement policies to open door for secure-QoS research.

8.12 Summarizing SWAP with other subactive techniques

Like other subactive techniques, SWAP also does not pay hardware overhead of detecting deadlock instead it periodically swaps packet in the network, to flush any deadlock that may have formed. SWAP does mis-route packets, but it is not always true as shown in Figure 8.5. Because of this limited mis-routing of packets SWAP offers superior performance compared to other sub-active techniques.

We will quantify the performance of all subactive techniques in chapter 10, here we present the qualitative summary of subactive techniques proposed thus far.

Next chapter presents the final subactive technique in this thesis. It is called SEEC, which stands for *Stochastic Escape Express Channel*. The main idea of SEEC (and its variant mSEEC) is to choose packets from the network in a *round-robin* fashion and zoom them through the network until ejection. The chosen packets are not buffered at intermediate routers in the network. As we will observe, SEEC provides higher performance among all subactive techniques as it does not misroute packets.

8.13 Chapter Summary

We present *SWAP*, a novel technique that enables in-place packet swaps across neighboring routers. SWAP guarantees deadlock freedom by design as it can dynamically break any buffer dependence cycles that might form in the network. It enables the network to take full advantage of available path diversity without requiring turn restrictions, injection restrictions or escape VCs to avoid deadlocks. A steady rate of packet swaps also means that deadlocks do not need to be explicitly detected and recovered from. We present lightweight extensions to implement swaps in our baseline router microarchitecture using a simple bus connecting all input ports of the router, and a unit to initiate the swap at a fixed rate. SWAP is the first non-recovery-based deadlock-freedom technique that enables fully-random minimally adaptive routing with just 1 VC. Moreover, it works seamlessly in

systems with irregular network topologies, emanating from heterogeneity, faults or power gating. SWAP is a powerful idea that goes beyond just deadlock resolution. It allows packets to escape congested parts of the network and can be overlaid on any network for enhancing throughput, without adding additional buffers.

Next chapter: *SEEC: Stochastic Escape Express Channel*, of this thesis is another subactive technique which seeks to provide simultaneous routing and protocol deadlock freedom with no misrouting. This work offers superior performance compared to schemes proposed thus far. We also compare its performance to all the prior schemes for both synthetic traffic patterns and real applications. Let's study this scheme in detail in next chapter.

CHAPTER 9

STOCHASTIC ESCAPE EXPRESS CHANNEL (SEEC)

Allocating a free buffer before moving to the next router is a fundamental tenet for packet movement in NoCs. However, there are two challenges involved. First, if there is a cyclic dependency of the buffers then it can result in a deadlock. Second, if buffers are full of packets going towards congested regions, other packets get blocked as well.

In this work, we introduce the idea of stochastic escape express channels (SEEC). The network interfaces in SEEC send special tokens called seekers to find packets destined to them and upgrade them to use a novel flow control called Free-Flow (FF). FF-packets traverse the network minimally from link to link, bypassing routers (bufferlessly) to the destination. Thus, any deadlock that a FF-packet was originally involved in is guaranteed to break, without requiring turn restrictions or extra VCs.

FF-packets have an added advantage of bypassing regions of congestion in the network without needing more buffers. We also present an extension called multi-SEEC (mSEEC) that enables multiple simultaneous non-intersecting FF packet traversals in the NoC to enhance throughput further.

We propose SEEC, a novel unified approach for solving the two sets of challenges discussed above. Instead of explicitly adding VCs in every router, SEEC adds a *stochastic escape express channel* that blocked packets can use to make forward progress, without requiring a free buffer at the downstream router. SEEC does not introduce new VCs or buffers, but instead relies on the following observation: when packets are blocked at routers waiting for credits, the links are still idle, as can be seen in Figure 9.1(a) and Figure 9.3(a). SEEC introduces a new flow-control technique called Free Flow (FF). A FF-packet can traverse *bufferlessly* over the links of the network up to its destination network interface (NIC) via a minimal path, bypassing all the intermediate routers.

Table 9.1: **Summary Table of Qualitative Comparison of Deadlock Freedom Mechanisms. P: Proactive, R: Reactive. S: Subactive.**

Techniques	Full Path Diversity	No Detect deadlock	No Misroute	No Extra Buffers	Routing deadlock freedom	Protocol deadlock freedom
Dally's theory/Acyclic CDG (P) [26]	✗	✓	✓	✓	✓	✗
Duato's theory/Escape VC (P) [52]	✗*	✓	✓	✗	✓	✗
Bubble [37, 50] (P)	✓	✓	✗	✗	✓	✓+
Deflection (P) [42]	✗**	✓	✗	✓	✓	✓
Deadlock Buffers (R) [49, 47, 65, 38]	✓	✗	✓	✗***	✓	✓***
Coordination (R) [41]	✓	✗	✓	✗****	✓	✗
BBR (S) [8]	✓	✓	✓++	✓	✓	✗
BINDU (S) [9]	✓	✓	✗	✓	✓	✗
DRAIN (S) [10]	✓	✓	✗	✓	✓	✓
SWAP (S) [11]	✓	✓	✗	✓	✓	✓
SEEC (S)	✓	✓	✓	✓	✓	✓

* Within escape VCs: limited path diversity + requires topology information for escape path.

**At low-loads, full path diversity is available. But at medium-high loads, packets cannot control the directions or paths along with they are deflected.

***DISHA [47] uses timeout counters present at each input port to choose a packet to eject from the network. It requires a set of extra buffers to route the packet involved in deadlock. Some variations of DISHA, such as mDISHA [49] provide protocol-level deadlock freedom

****SPIN[41] requires a buffer in each router to hold the dynamic deadlock path over which packets involved in deadlock would move synchronously.

+ Bubble Coloring [50] provides protocol-level deadlock freedom but involves non-minimal path traversal.

++ BBR provides limited misrouting of packet because of *Bubble Exchange* subsection 5.3.1

A packet is selected to become a FF-packet by a token called a *seeker*, that is sent by a destination NIC to search for any packet in the NoC intended for it, after reserving a buffer slot at the ejection queue. This guarantees ejection for the FF-packet. We also present an enhancement called Multi-SEEC (mSEEC) that enables simultaneous non-minimal bufferless traversals of multiple FF-packets in the network with no collisions.

Figure 9.1(b) and Figure 9.2(b) illustrate how FF-packets can resolve both routing and protocol deadlocks. Similarly, in Figure 9.3, the *up* packet in the SEEC NoC leverages FF flow-control to bypass the congested region to reach its destination.

This work makes the following contributions:

- We introduce SEEC, a unified approach for deadlock-freedom (both routing and protocol) and higher network performance without requiring any routing restrictions, injection restrictions, escape VCs, virtual networks, or misrouting (like prior schemes).
- We introduce a novel flow-control called Free Flow for bufferless, minimal, non-blocking traversal of packets all the way to their destinations.
- We present two policies for upgrading packets to use FF. The baseline SEEC policy enables only one packet in the NoC to use FF at a time; the second, mSEEC, enables multiple simultaneous non-colliding bufferless FF traversals.
- We evaluate SEEC and mSEEC on a wide range of synthetic workloads and real applications and observe 34%-40% reduction in average packet latency for real application and 10%-50% average improvement in throughput for synthetic traffic pattern over the state-of-the-art solutions at $1/6^{th}$ area/power budget.

SEEC/mSEEC is the final subactive technique proposed in this thesis unlike BBR and BINDU, it does not require empty VC or bubble or *Bindu*. Moreover, unlike SWAP and DRAIN it does not misroute packets in the network. Table 9.1 contrasts SEEC/mSEEC against the prior work and subactive schemes presented so far.

Let's study SEEC in detail and in section 9.11, we would draw the contrast of this

technique against prior subactive techniques.

9.1 Background and Related Work

SEEC makes primary contributions in both flow control and deadlock freedom. In this section, we discuss relevant background and related work in flow control and deadlock freedom in NoCs.

9.1.1 Flow-Control Optimizations

In packet-switched NoCs, flow control determines the allocation of input buffers to flits of a packet, to make sure they do not overwrite each other. Credits are the predominant signaling mechanism for buffers. Credits in the upstream router keep track of the number of free buffers available at the downstream router; and are used to determine if packets can be forwarded downstream. To improve performance in NoCs, several proposals have explored flow control optimizations. Techniques like flit reservation flow control [77] reserve buffer resources ahead of flits over dedicated fast links, to reduce credit signaling delays. In circuit-switched coherence [79], circuits are established on a best-effort basis to allow packets to travel bufferlessly through the network, only getting latched at each hop. If link resources are not available for reservation, packets start getting buffered again. Multiple concurrent forms of flow control have also been explored over multiple distinct physical networks [80, 81]. Some techniques allow packets to reserve buffers multiple-hops away, and bypass intermediate router buffers. For e.g., Express Virtual Channels (EVCs) [82] creates bypass paths of short fixed lengths along the X or Y dimension, between statically assigned source and sink nodes. The packet between intermediate nodes gets prioritized using lookahead, while at source and sink nodes the packet gets buffered in a separate VC called: *express VC*. Its size depends on the length of express paths, due to buffer turnaround time considerations, and tends to be deeper than normal VCs. Token Flow Control (TFC) [83] dynamically creates hints for bypass paths of upto a fixed max-

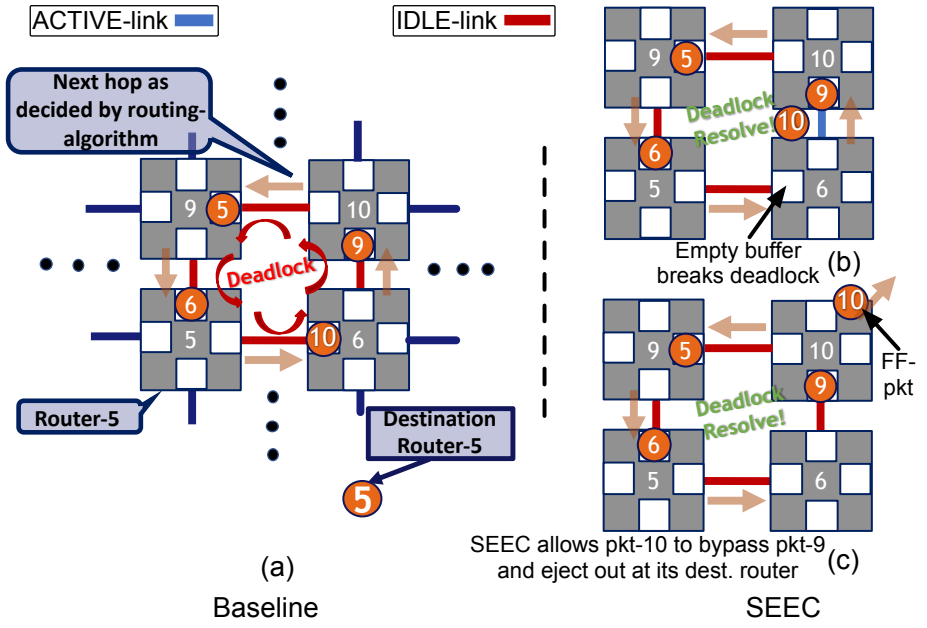


Figure 9.1: Routing Deadlock: (a) Packets' ability to make forward progress is blocked by other packets. Arrows represent desired movement direction. (b) SEEC resolves the routing deadlock by allowing FF pkt (pkt-10) to bypass the buffered pkt (pkt-9) until ejection, creating an empty buffer, breaking the deadlock. (c) Shows FF-pkt ejecting out of the network.

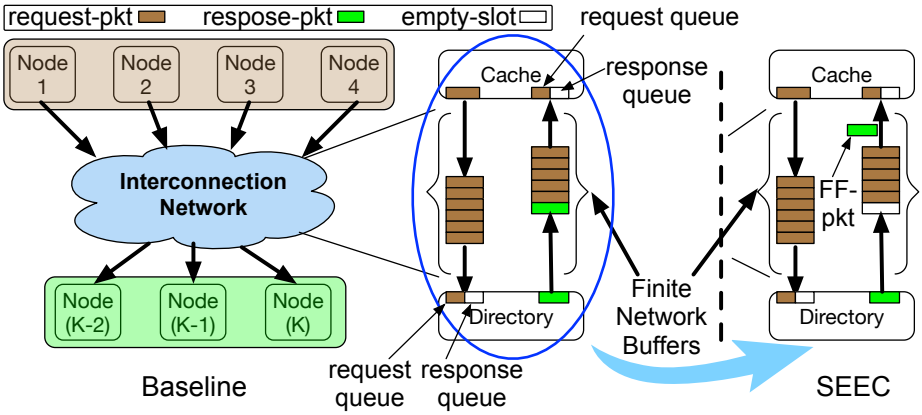


Figure 9.2: Protocol Deadlock: All buffers occupied with request packets. Thus, the response packet is stuck indefinitely. Forward progress is only possible by consuming the response packet. SEEC allows the response packet to become FF and reach its destination by bypassing all request packets.

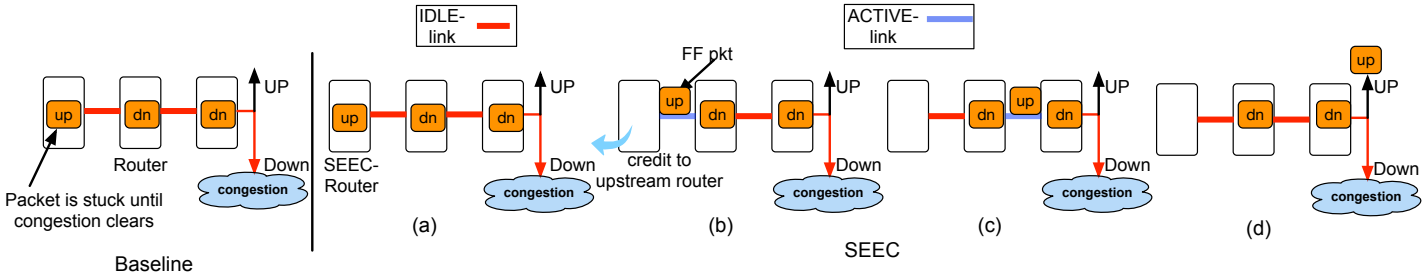


Figure 9.3: Head-of-line Blocking due to congestion. (a) A packet going "up" is blocked by packets going "dn" in the Baseline. (b)-(d) *SEEC*'s FF flow control allows the "up" packet to bypass the congested region to reach its destination.

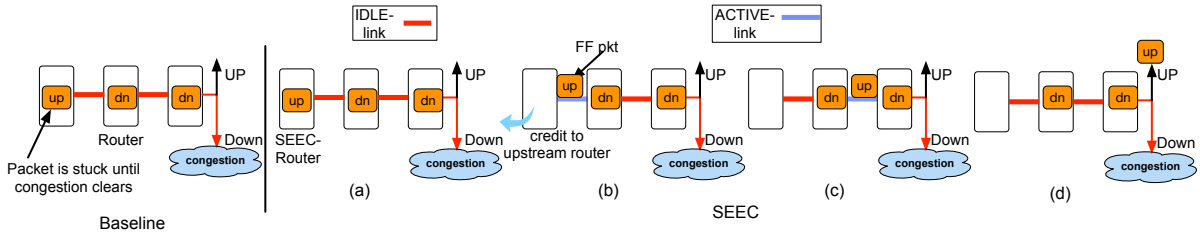


Figure 9.4: With SEEC, packets are not stuck indefinitely. FF flow control allows packets to bypass the congested region.

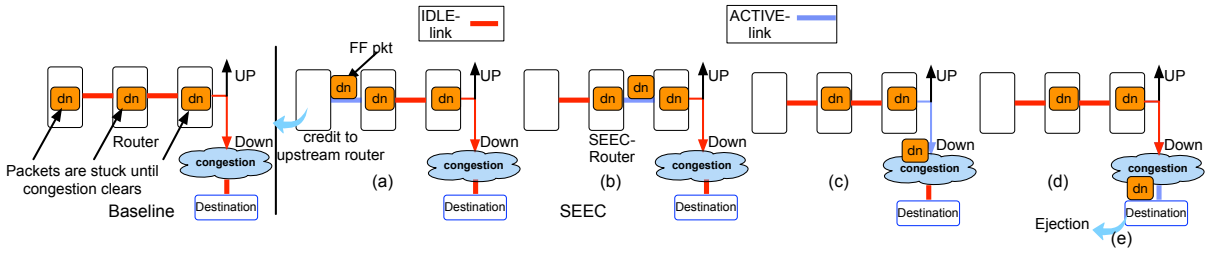


Figure 9.5: SEEC's FF control allows packets to bisect through the congested region to reach its destination.

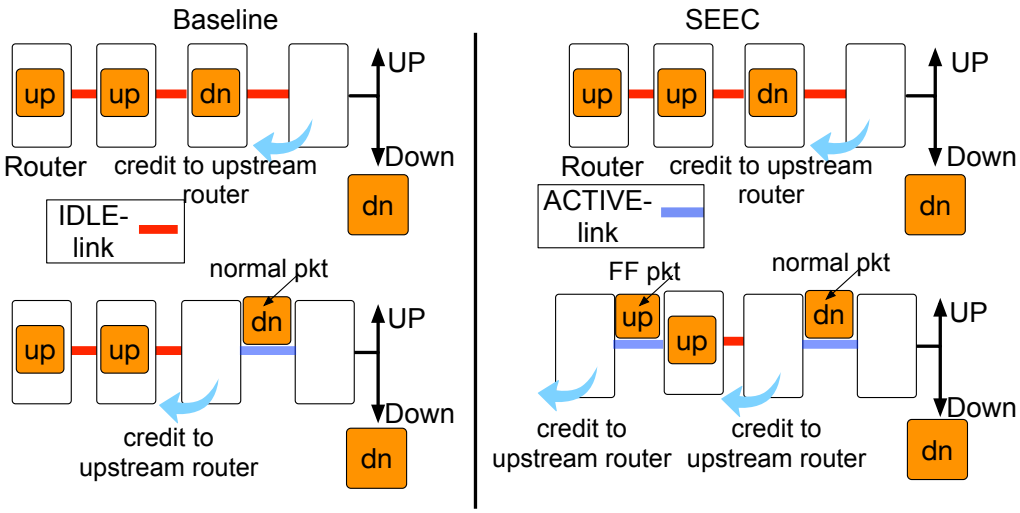


Figure 9.6: SEEC improves throughput by ameliorating the effect of credit round trip delay and utilizing the otherwise *idle-links* in the baseline network.

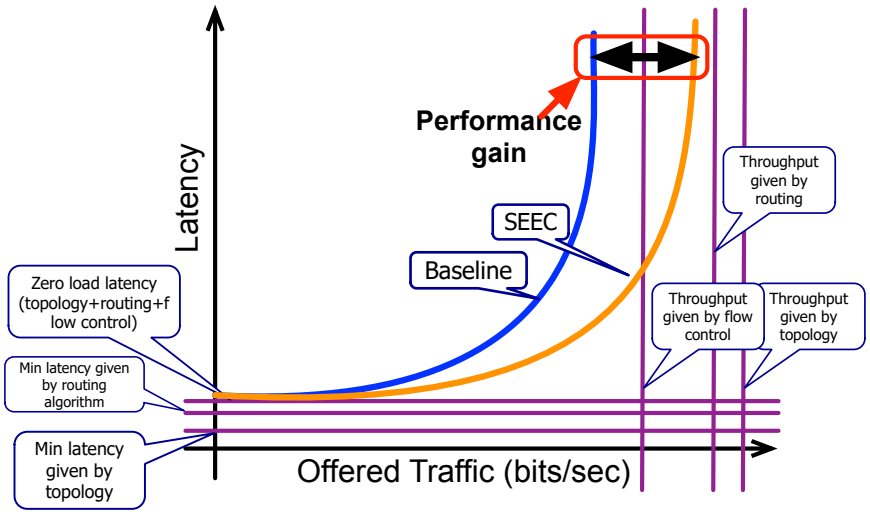


Figure 9.7: Traditional latency throughput curve. SEEC improves performance by reducing the effect of credit turnaround time due to its novel flow control

imum length using the deadlock free adaptive turn-model routing, by grabbing "tokens", which are hints about buffer availability multiple hops away. TFC provides *adaptability* to turn model routing at a cost of complex routing algorithm and wiring overhead. Qualitative comparison of EVC and TFC vs SEEC is present in section 9.10.

Deflection flow control is an alternate approach where an upstream router does not wait for credits at the downstream router, and blindly forwards the flits of the packet; however, in the face of contention (where two packets request the same output port), one of the packets is deflected out a non-productive port, leading to misrouting. This flow control is used in hot potato routing [43] and in bufferless NoC implementations [42, 45, 44].

SEEC versus prior Flow Control techniques. Free Flow traversal in SEEC is similar in flavor to bypass in EVC and TFC. However, while EVC and TFC offer opportunistic bypasses based on hints of buffer availability, SEEC establishes a bypass path all the way to the destination since a buffer is reserved at the destination NIC. Moreover, it helps with both performance and deadlocks, while EVC and TFC focus only on performance. We further delineate SEEC from EVC and TFC in detail under section 9.10.

The traversal of FF packets within SEEC has some similarities to the traversal of golden packets in CHIPPER [45] and MinBD [44], though in these prior works the goal is to provide livelock freedom (vs. deadlock freedom and performance in SEEC) and their mechanism to upgrade packets is different. Table 9.2 compares SEEC and prior works qualitatively.

9.1.2 Deadlock Freedom

Here we examine deadlock freedom from both routing and protocol deadlocks. **Routing deadlocks** can occur if packets form a dependence cycle in the NoC, as shown in Figure 9.1(a), while **Protocol deadlocks** can occur if messages from terminating message classes (e.g., responses) in coherence protocols get blocked by messages from non-terminating message classes (e.g., requests) [84](See Figure 9.2(a)). Table 9.1 qualitatively

Table 9.2: **SEEC/mSEEC contrasted against bypass mechanisms in EVC/TFC and CHIPPER/MinBD flow control.**

Property	EVC [82]/TFC [83]	CHIPPER[45] / MinDB [44]	SEEC/mSEEC (This Work)
Bypass Mechanism Purpose	Higher Throughput (*bypass congestion*)	Livelock Freedom (*remove / reduce deflections for select packets*)	Higher Throughput (*bypass congestion*) & Deadlock Freedom (*remove blocked packets from network*)
Regular Packet Route	Always Minimal	Non-minimal (*misrouting via deflections upon contention*)	Always Minimal
Bypass Packet Route	Always Minimal	Always Minimal for Golden, Minimal or Non-Minimal for Silver	Always Minimal
Bypass Packet Upgrade Mechanism	Opportunistic based on buffer availability multiple hops down.	*Golden:* Fixed global priority based on unique packet id. *Silver:* Opportunistically within router	Explicit *seeker* messages sent by end-point NIC to find packets destined for the NIC.
Bypass Path Length and Buffer Availability at end-point	Fixed length (EVC); flexible length based on buffer availability (TFC). Bypassing packet guaranteed a buffer when it stops.	Bypass path up to destination router for Golden packet. If no empty slot at destination NIC, packet will be dropped and destination NIC sends a special message to sender to re-transmit once slot available (*"Retransmit-Once Flow control"*).	Guaranteed path up to destination NIC for FF packet. Slot at NIC reserved in advance by seeker; no retransmissions needed.
Bypass Path Length	Fixed length (EVC); flexible length based on buffer availability (TFC).	Bypass all the way till destination router for Golden packet.	Bypass all the way till destination NIC for FF packet.
Buffer Availability at End-point	Guaranteed via EVC/-token reservation before packet starts.	Not gauaranteed. If no empty slot at destination NIC, packet will be dropped and destination NIC sends a special message to sender to re-transmit once slot available (*"Retransmit-Once Flow control"*).	Guaranteed as slot at NIC reserved in advance by seeker; no retransmissions needed.
Packet Bypass Priority	Express > Regular	Golden > Silver > Regular	Free Flow > Regular
Simultaneous Bypass Paths	Multiple opportunistic bypass paths.	One guaranteed bypass path (golden packet) in NoC. Silver packets are local.	Multiple non-overlapping guaranteed bypass paths in NoC.
Deadlock Freedom	Turn-Model (proactive)	Deflection (proactive)	SEEC (subactive)
Livelock Freedom	No Livelocks with only minimal routes.	Livelocks possible due to deflections (removed via golden packets).	No Livelocks as there are no non-minimal routes.

contrasts the various previous approaches to deadlock freedom in the literature with SEEC. Generally, we categorize routing deadlock-freedom solutions into three categories: proactive (P), reactive (R) and subactive (S), using terminology from DRAIN [10]. Proactive schemes ensure a deadlock is never created in the first place, reactive schemes detect and recover, and subactive schemes allow deadlocks to form but guarantee that they get cleared eventually without requiring explicit detection. In the table, we categorize prior work based on their key idea, and specify if they are P, R or S, and contrast them across various properties. We discuss their details here.

Dally's theory/Acyclic CDG (P). Dally and others [85, 34, 36] propose that applying turn restrictions to packets can lead to an acyclic CDG. A key drawback is that turn restriction packets reduce path diversity and hurt performance.

Duato's theory/Escape VC (P): These solutions restrict the path that a packet can take in the network only for one Virtual Channel (VC) per input port, called an Escape VC. Packets enjoy full path diversity while in normal VCs [74, 52]. The key challenge is that Escape VCs require an extra VC at each input port for implementation.

Bubble (P): A bubble refers to an empty buffer or VC. The underlying theory [37, 39, 40, 86, 87] is that as long as there is an empty buffer present in a *ring*, that ring is deadlock free. The key drawback is global coordination for tracking the bubble. Moreover, when overlaid on an irregular topology, *bubble-based schemes* result in non-minimal routing. Even in regular topologies such as a mesh, overlaying a virtual *ring topology* to route packets also results in non-minimal packet traversal.

Deflection (P): Deflection routing argues that as long as packets keep moving every cycle, either towards or away from their destination, the network is deadlock free. It follows hot-potato routing [43] in which packets keep getting misrouted in the face of contention. The key drawbacks include non-minimal routes that packets take reach their destinations, leading to higher latency and dynamic energy, and expensive solutions to *guarantee* livelock freedom [42, 45, 44].

Deadlock Buffer (R): Schemes such as DISHA [47] and its extensions [49, 65, 38] embed extra *deadlock buffers* [49] at design-time in all routers, which remain off during nominal operation. Deadlocks are detected via time-outs and a blocked packets are progressively routed to their destinations via these deadlock buffers.

Coordination (R): Coordination-based schemes such as SPIN [41] detect deadlock and synchronously move packets to resolve the deadlock. The key drawback is the complex deadlock detection circuitry to detect and map the deadlock ring at runtime. Moreover, it requires global synchronization among the packets that must move simultaneously to

resolve the deadlock.

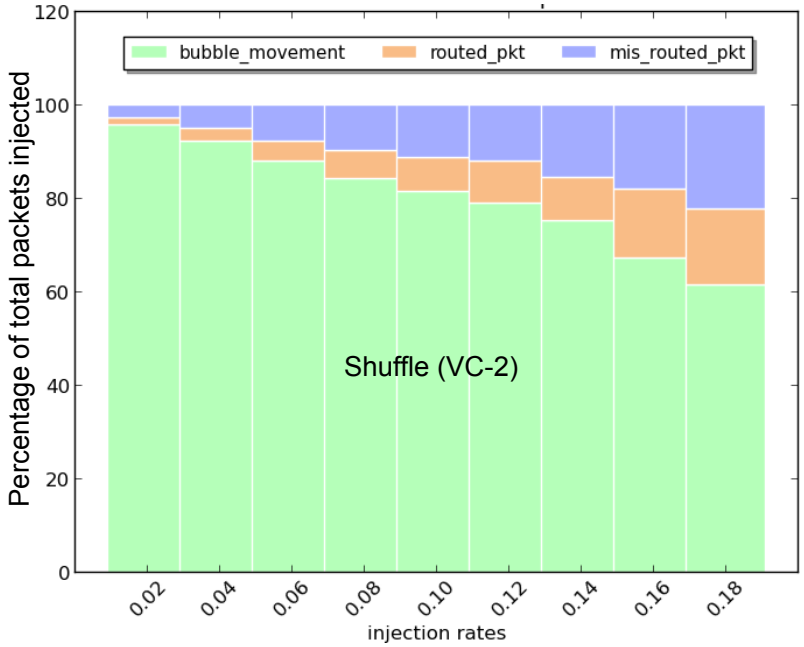
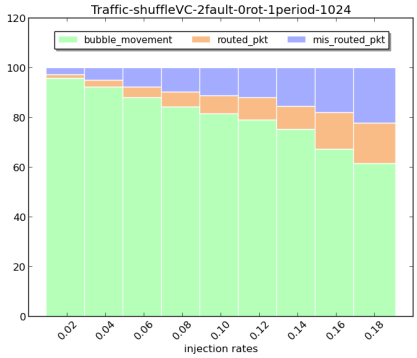


Figure 9.8: In DRAIN [10], each "drain" spins the contents of all buffers in the network. This graph plots the distribution of bubbles (empty slots), routed packets (moved in productive directions), and mis-routed packets in the network across all drains for the shuffle traffic pattern. It shows misrouting increasing as injection rates go up for 8x8 Mesh.

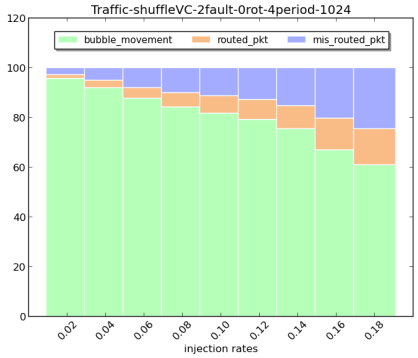
Figure 9.9, Figure 9.10, Figure 9.11, and Figure 9.12 shows the sensitivity analysis of DRAIN[10] in terms of relative %age of packets mis-routed and routed when allowed to *spin* obliviously on the DRAIN-path for different synthetic traffic patterns, DRAIN epoch period and number of hops traversed by packets on the predefined DRAIN path.

Oblivious Packet Shuffling (S): Recently a class of solutions has been proposed which leverage periodic packet shuffling in the network [8, 9, 11, 10] to remove deadlocks. The shuffling may be within a router [8], between neighbors [9, 11] or along an embedded ring across the entire network [10]. The underlying theory is that if packets are periodically forced to move obliviously in the network then any routing deadlock can be resolved. The periodic and oblivious nature of packet shuffling overlaid on a deadlock prone minimal routing algorithm in these solutions ensures deadlock freedom.

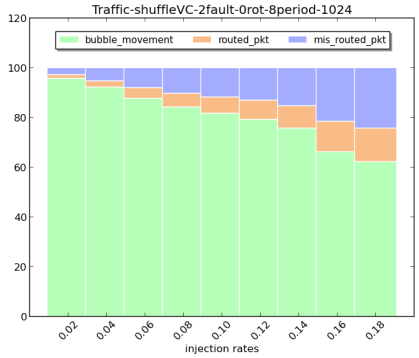
A key challenge is that oblivious packet movement may misroute a packet away from



(a) One hop on DRAIN-path

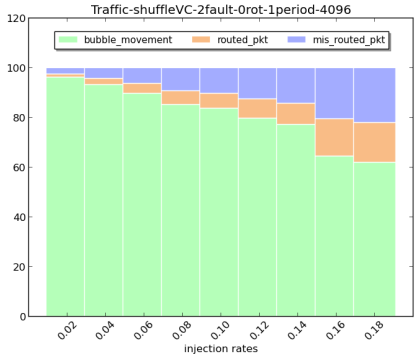


(b) Four hops on DRAIN-path

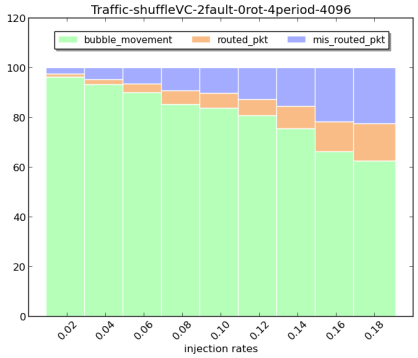


(c) Eight hops on DRAIN-path

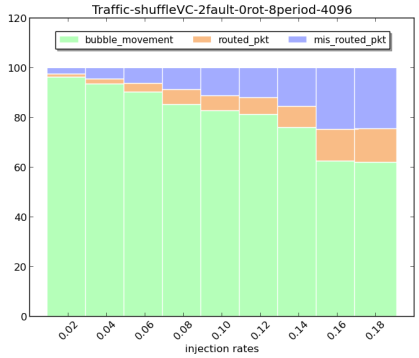
Figure 9.9: Shuffle traffic; 8x8 Mesh, each input port has 2 VCs. Percentage of routed, mis-routed and free buffers present in the topology until saturation with 1k cycle as DRIAN epoch. Percentage of misrouted packets is consistently higher than that of routed packets



(a) One hop on DRAIN-path

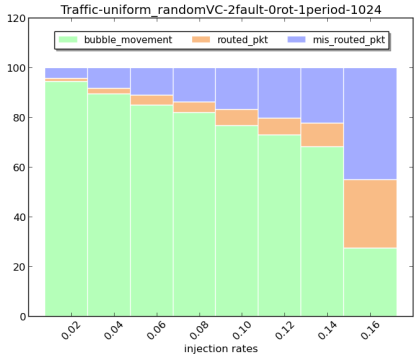


(b) Four hops on DRAIN-path

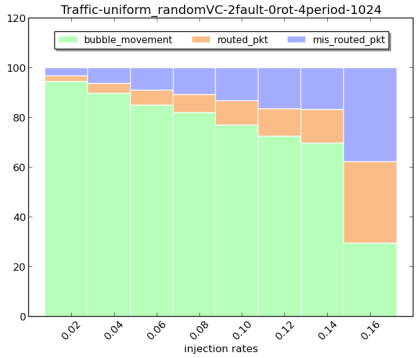


(c) Eight hops on DRAIN-path

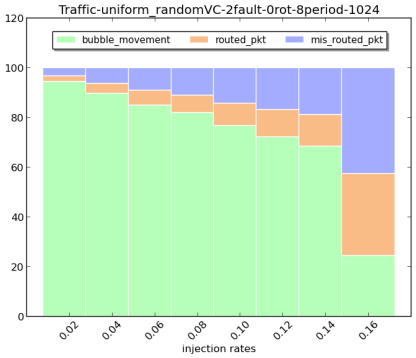
Figure 9.10: Shuffle traffic; 8x8 Mesh, each input port has 2 VCs. Percentage of routed, mis-routed and free buffers present in the topology until saturation with 4k cycle as DRIAN epoch. Percentage of misrouted packets is consistently higher than that of routed packets



(a) One hop on DRAIN-path

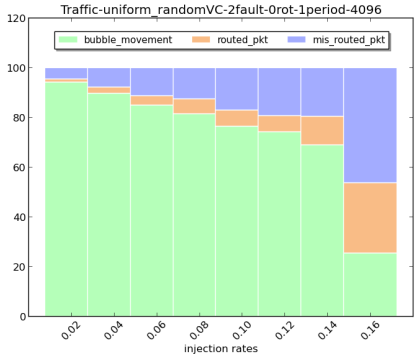


(b) Four hops on DRAIN-path

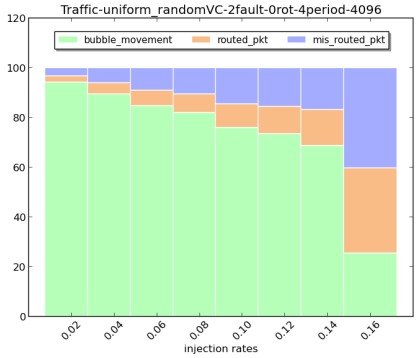


(c) Eight hops on DRAIN-path

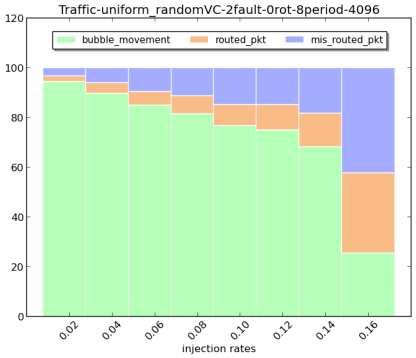
Figure 9.11: Uniform traffic; 8x8 Mesh, each input port has 2 VCs. Percentage of routed, mis-routed and free buffers present in the topology until saturation with 1k cycle as DRIAN epoch. Percentage of misrouted packets is consistently higher than that of routed packets



(a) One hop on DRAIN-path



(b) Four hops on DRAIN-path



(c) Eight hops on DRAIN-path

Figure 9.12: Uniform traffic; 8x8 Mesh, each input port has 2 VCs. Percentage of routed, mis-routed and free buffers present in the topology until saturation with 4k cycle as DRIAN epoch. Percentage of misrouted packets is consistently higher than that of routed packets

its destination; therefore, a packet may need to traverse the network non-minimally. Additionally, in pathological cases the deadlock may be slow to resolve.

As Table 9.1 shows, existing subactive solutions are promising alternatives to proactive and reactive schemes as they provide full-path diversity, no extra buffers, and no resources to detect and recover from a deadlock. However, their key downside is misrouting. Figure 9.8 quantifies the percentage of misroutes in DRAIN [10].[1] These misroutes can adversely affect application tail latencies, as we quantify in subsection 9.9.1.

The most common approach for protocol deadlock freedom today is to use separate VCs (called virtual networks (VNet) [10]) in the NoC for each message class. As shown in Table 9.1, Deflection [42], DRAIN [10], Bubble Coloring [50], and mDISHA [49] can also provide protocol deadlock-freedom without the need for virtual networks. However, they come at the cost of continuous misrouting, periodic misrouting, non-minimal routes, and extra buffers, respectively.

SEEC versus prior work. Prior work has shown that deadlocks are quite rare [84, 38, 10] and VCs within routers are highly under-utilized [10]. This means that over provisioning VCs and VNets for routing and protocol deadlock-freedom is highly inefficient from an area and power perspective; in proactive solutions, however, it becomes necessary to have them for correctness. Subactive solutions [8, 9, 11, 10] are promising alternatives to proactive and reactive schemes as they provide full-path diversity, no extra buffers, and no resources to detect and recover from a deadlock. SEEC is also subactive, i.e., it ensures that deadlocks, even if they were to form, will be broken as packets use FF to exit the NoC. SEEC guarantees both routing and protocol deadlock freedom with the theoretical minimum of one VNet with one VC per input port in the NoC. We present a formal proof later in section 9.3. Moreover, unlike prior subactive schemes that all cause misrouting (Table 9.1), SEEC is free of misrouting.

SEEC has similarities with DISHA [47] and its extensions [49, 65] as both use circulat-

[1] We used DRAIN's open-source code-base [88] to get this data. Simulation configuration details can be found in subsection 9.7.1.

ing out-of-band control messages over a predefined path covering all routers in the network, to select the packet to eject. However, there are key differences. SEEC does not use time-out counters to select the packet to eject, and instead sends a special message to *proactively search* for one. This can speed up deadlock recovery. Moreover, unlike DISHA that only triggers upon deadlock detection, SEEC continually allows packets to use FF, enhancing throughput.

9.2 SEEC

We describe SEEC from concept to implementation. Table 9.3 lists some key terms used throughout the paper.

As an analogy for SEEC, consider the flow of traffic; cars and trucks on the road are required to obey traffic signals. A car cannot advance to the next block until the light turns green (this is analogous to waiting for a credit). However, some vehicles can be designated to ignore such traffic rules. Emergency vehicles are an example; regular cars yield to emergency vehicles and let them pass. Similarly in SEEC, one packet (at a time) in the system gets elevated in priority to use an express path to reach the destination.

9.2.1 Free Flow

We propose a new flow control scheme called **Free Flow (FF)**. A packet using FF is given priority at each router over regular packets which use credit-based flow control, allowing its flits to traverse the network bufferlessly, over a minimal route (i.e., minimum possible hops) all the way to its destination NIC. Unlike deflection flow control (subsection 9.1.1), there is no misrouting of FF packets. This is ensured by guaranteeing that multiple packets concurrently traversing the NoC using FF use non-intersecting paths. For the base SEEC design, there can only be one FF packet in the network at any given time. section 9.4 discusses how multiple FF packets with non-intersecting paths can be guaranteed.

Table 9.3: **Key Terms in SEEC.**

Term	Definition
Free Flow (FF)	A flow control scheme where packets are not buffered, and move forward every cycle, without the need for credits.
Seeker	A token sent out by the destination NIC after reserving an ejection VC within a specific message class. It searches for a packet in the NoC intended for this NIC in that message class.
Seeker path	A side-band network connecting all routers in the network over which the seeker visits all routers, and comes back to its initiator in case it does not find a packet.
FF packet	This packet is chosen by the seeker and traverses to its destination over a *minimal path* using FF.
Lookahead	A signal sent one hop ahead of FF-packet so that next-hop router prioritizes the movement of incoming FF packet over the normal buffered packet.
mSEEC	An extension to SEEC where multiple seekers are sent out at the same time to different network partitions. The FF packets use non-overlapping, minimal routes.

9.2.2 Overview

In SEEC, all packets nominally use credit flow control. In a round-robin manner, each destination NIC selects a packet in the network destined for itself to use FF. The mechanism for selecting such a packet is discussed later in subsection 9.2.3. This packet zooms through the network without being buffered, by being prioritized to use the link at every hop along its route, without getting buffered within each intermediate router. This is implemented by sending a lookahead signal one cycle ahead on a dedicated link, similar to prior work [82, 83]. The lookahead carries the output port of the FF packet, and the intermediate router sets up its crossbar for the FF packet arriving next cycle, disallowing its own locally buffered flits from using that output link. As a result, the FF packet can bypass both contention and deadlock in the network. We describe the detailed operation of the design in the next section.

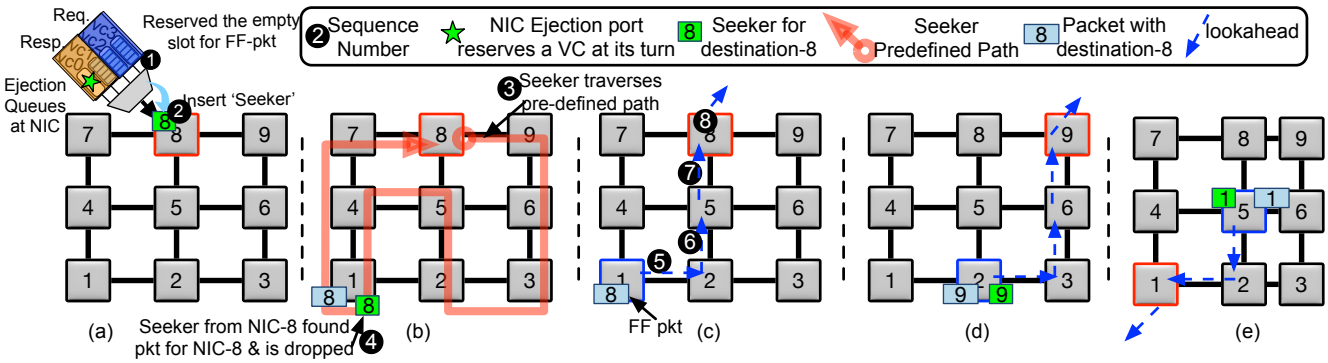


Figure 9.13: Step-by-step working of SEEC, (a) Router-8, at its turn, reserves a VC in its ejection port and inserts a 'Seeker' in the network (b) Router-8's *Seeker*, traverse the topology on a predefined path to look for packets to eject. (c) Seeker finds the packet at router-1, Seeker gets dropped and this buffered packet now becomes FF-packet (d) After router-8, router-9 repeats the process of reserving a VC in its ejection port (not shown to save space); sending seeker and ejecting packet *bufferlessly* (e) After router-9 (last router), router-1 (first router) repeats the process. Destination routers take part cyclically for ejecting packet.

9.2.3 Operation Details

System Assumptions. SEEC assumes that the NIC has per-message class ejection VCs, as shown in Figure 9.13. However, within the NoC, all messages share the same set of VCs. Separate VNets do not exist, as was shown in Figure 9.2.

Mechanism for choosing FF packet and guaranteeing ejection. SEEC is initiated by a NIC reserving an ejection VC within a message class, and circulating a *Seeker* through the network over a pre-defined path covering all routers to search for any packet intended for this ejection VC. If the seeker finds such a packet, the packet traverses the network via FF and the seeker is dropped. Since an ejection VC is reserved before the seeker starts searching, the FF packet is guaranteed to be consumed. Once the FF packet is ejected, or if the seeker returns, the control moves in a round-robin manner to ejection VCs of the next message class at this router, and then to the neighboring router, and so on, to send out their seekers.

What if the seeker does not find any packet? If the seeker circulates back to the original router, it indicates that there was no packet in the NoC currently for this message

class waiting to get routed to this NIC. The reserved ejection VC is freed up and the same process is repeated for the remaining ejection VCs. Once all ejection VCs are exhausted, the router notifies its neighboring router of its turn to send seekers in a round-robin fashion.

What if no ejection VC is free in a message class?

In this case, it passes its turn in a round-robin manner to the next message class, and eventually to the neighboring router to send its seeker signal instead. However, once a message class that missed its turn gets a free ejection VC, it is pro-actively reserved for the next round. No packets are allowed to use this ejection VC until this VC's next turn for sending a seeker.

How is the SEEC path implemented? In our implementation, the SEEC path is a ring through all routers in the NoC (as shown in Figure 9.13). Each router has local state to store the order in which the seeker should search through all input ports with that router, and where to forward the seeker next. For mSEEC, that we describe later in section 9.4, the seekers' paths no longer span the entire NoC, but instead only within partitions.

What is the policy for searching for packets within the SEEC path? There could be myriad policies used by the seeker for selecting packets to upgrade to FF. For fairness, we implement a *round-robin* policy, where each destination logs the location of the $< router - id\_inport - id >$ from where the last FF packet was selected and begins the search from there. It searches through all routers and ports and returns to its sender if it did not find a packet.

What if the ejection port is busy when the FF packet arrives? Though the FF packet is guaranteed to be ejected, it is possible for it to arrive at its destination router while another multi-flit packet is in the process of getting ejected. The FF packet in this case will take precedence over the ongoing ejection and the current ejection will wait until the FF-packet gets ejected from the network. Since the ejection VC for the FF packet was reserved prior to the seeker being sent, the stalled ejection will not be for the same VC.

9.2.4 Walk-through Example

Figure 9.13 shows a walk-through example of SEEC. As discussed in subsection 9.2.3, a packet is chosen to become FF by its destination NIC. This is done with the help of a seeker. In this example, router-8 first reserves an ejection VC in the response message class at NIC-8. It then sends the seeker (Figure 9.13(a)) on a predefined path covering all the routers in the network at least once. During its network traversal, if the seeker finds a response packet whose destination is router-8, it converts this normal buffered packet into the FF packet. In this example, this packet is at router-1 (Figure 9.13(b)). The seeker is dropped and the newly dubbed FF packet travels minimally to its destination via an express path created using lookaheads, as shown in Figure 9.13(b). Once this original FF packet is ejected, the same process is repeated for the next message class. After all message classes have tried (successfully or unsuccessfully) to receive a FF packet, the next router-id is notified to send its seeker. This process keeps repeating in a cyclic fashion over the topology (Figure 9.13(c) and Figure 9.13(d))).

9.2.5 Lookaheads

To allow the FF packet to make forward progress every cycle, and to suppress the movement of normal packets at the next downstream router through the same output port, a lookahead signal is sent by the router which accepted the seeker to make its buffered packet FF. This lookahead signal is sent on its dedicated link one cycle ahead of the data packet, similar to prior work [82, 83]. The lookahead reserves the output port for the FF packet at the next downstream router as dictated by the baseline routing algorithm. section 9.5 further describes the appropriate router micro-architecture modifications over the baseline router to realize a SEEC router.

The dedicated link width for *lookahead* is enough to encode the $< dest - id\_outport - id >$ for the current FF packet. We expect it to be 9 bit wide for an 8x8 Mesh (6 + 2 + 1): 6 ($log(num - routers)$ + 2 ($log(num - outport)$) bits for dest_id and outport_port_id + 1

valid bit. This is minimal overhead compared to the typically 128 bits wide network links (Table 9.4).

9.3 Proof of Correctness: Deadlock Freedom Proof

Assumption: In SEEC we assume that the *local* port of the router has separate VNet for each message class, this assumption makes sure that response packet gets to enter the network irrespective of current occupancy of network buffers.

We first prove that SEEC guarantees resolution of protocol deadlocks and then extend the proof for routing deadlocks.

Assumption: The ejection port at the NIC has separate VCs for each message class of the protocol.

Definition: Terminating Message Class. We define terminating message class to refer to a message class that ends the protocol transaction, e.g., responses.

Necessary condition for breaking protocol deadlocks. Messages belonging to terminating message classes should never be indefinitely blocked within the NoC.

Lemma 1: Messages within the ejection VC for a terminating message class in a coherence protocol are always guaranteed to be consumed by the cache controller (*aka consumption assumption*) and do not block waiting for other messages.

Proof: Whenever a request is issued to the NoC, an entry is allocated in an MSHR and injected into the outgoing request VC at the NIC. The response (data or ACK) arrives in the response VC at the NIC and will be consumed by the reserved entry at the MSHR for processing. If the rate of egress to the MSHR is slower than the rate of ingress into the ejection VC from the NoC, the ejection VC may temporarily be full, but will eventually become free to eject new packets from the NoC.

Similarly, invalidation requests issued by the coherence mechanism are replied with acknowledgment (ack) messages. While invalidation request processing may be blocked by the inability to insert acks, acks are immediately consumed upon receipt at the directory.

We cannot list every protocol dependence scenario, but terminating message classes such as responses/ACKs in protocols are built with the consumption assumption [84].

Lemma 2: A response packet that is causing a protocol deadlock is guaranteed to reach its destination via FF to break the deadlock.

Proof: Suppose a response packet is stuck in the NoC behind requests and is part of a protocol deadlock (Figure 9.2(a)). Thus it cannot make forward progress via regular means. The response message class at its destination will eventually have a free ejection VC for this specific packet as discussed by Lemma 1 (seen in Figure 9.2). Since each message class at each NIC gets a fair chance to send a Seeker in a round-robin manner, its destination will eventually send a seeker and enable this packet to zoom through the network via FF (as seen in Figure 9.2).

Lemma 1 + Lemma 2 prove that SEEC guarantees protocol-deadlock freedom.

Corollary 1: A request VC at the ejection port will never remain indefinitely blocked.

Proof: Since the network is protocol deadlock free, a request VC will not remain indefinitely blocked waiting for responses. Suppose a request message class does not have a free ejection VC during its turn to send a seeker, SEEC will send a seeker for the next message class. However, Lemma 2 and Corollary 1 prove that *all* message classes will eventually get a free ejection VC. Once the message class that missed its turn gets a free VC, this VC is *proactively reserved*. No packets are allowed to be ejected into this VC until this VC's next turn for sending a seeker. This bounds the number of times a message class might miss its turn to send a seeker.

Lemma 3: Every packet in a routing deadlock is guaranteed to reach its destination via FF control.

Proof: From Lemma 2 and Corollary 1, all message classes will eventually get a free ejection VC and get a chance to send a seeker. This will allow any packets in a cyclic dependence in the network to use FF to reach their destination, breaking the cycle (as seen in Figure 9.1).

Lemma 3 proves SEEC guarantees routing-deadlock freedom.

Livelock Freedom Proof. SEEC is livelock free because all the packets are routed minimally through the network. Thus SEEC does not need any additional livelock handling mechanism like tracking and prioritizing oldest packets [42, 45, 44] or performing network drains [10].

Point-to-Point ordering. Using FF, packets may get re-ordered from a given source to its destination. This effect is not unique to SEEC and is also present in adaptive routing algorithms [35]. Many commercial protocols such as HyperTransport support out-of-order delivery of messages [89]. For the protocols that do not, we assume re-order buffers for those message classes that need point-to-point ordering, and leave point-to-point ordering support for future work.

9.3.1 Applicability of SEEC

SEEC can be applied over any NoC topology and routing algorithm provided they satisfy the following conditions:

1. The topology can have a mix of bidirectional and unidirectional links, as long as every source router has a valid path to every destination router.

2. There are separate injection and ejection queues for each message class at the Network Interface. Therefore, there is no protocol-level deadlock at the end nodes.

9.4 Multi-SEEC (mSEEC)

In SEEC, we allow sending one seeker at a time on a pre-defined side-band path; as a result, there is a maximum of one FF packet at a time within the NoC. However, many popular network topologies such as meshes have lots of inherent path diversity available; sending one seeker at a time under utilizes the available path diversity of the topology for creating express paths. This is the motivation behind multi-SEEC (mSEEC). We first

discuss how mSEEC can be implemented in any abstract topology, and then describe our specific implementation.

To implement mSEEC, the topology needs to be divided into P independent *partitions*, and the NICs into N/P *groups* with P NICs each. Each partition receives a seeker from a different NIC. mSEEC operates over multiple phases. In the first phase, the P NICs from the first group *simultaneously* search for packets which wants to eject out from those partitions, using their respective seekers. Once the search, followed by FF-packet traversal is over (which is bounded due to the fixed time it takes for the seeker to traverse to the furthest router in its partition and the FF-packet to come back), the NICs permute these independent partitions among themselves. At the end of each phase, the P NICs would have searched through the entire topology. In the next phase, the next group of NICs follow the same approach.

To ensure no collisions, the partitions and groups need to be chosen in a way that the paths of the P seekers do not overlap during the seek portion, and the paths of the P FF-packets do not overlap during the FF portion of each phase.

In our implementation of mSEEC, we choose the partitions as the columns of the Mesh, and the groups as the rows of the Mesh. Figure 9.14 shows an example. In phase 0, during Step 1 (Figure 9.14(a) the three NICs (A, B, C) in row 0 of the Mesh topology simultaneously seek packets within Columns 0, 1 and 2. Next, they seek within Columns 1, 2 and 0, respectively (Figure 9.14(b)). Next, in Columns 2, 0, and 1, respectively (Figure 9.14(c)). The dotted line represents the path taken by the seekers, and the solid line is the path taken by the FF packets (which just follow the reverse path of the seekers). Unlike SEEC, where the SEEC path is an embedded ring (Figure 9.13), mSEEC uses minimal non-overlapping paths for the seekers.

After the first row finishes seeking packets from all the columns (i.e., the entire topology), in Phase 1, the next row now seeks packets in a round-robin manner over the columns of the topology (Figure 9.14(d)-(f)).

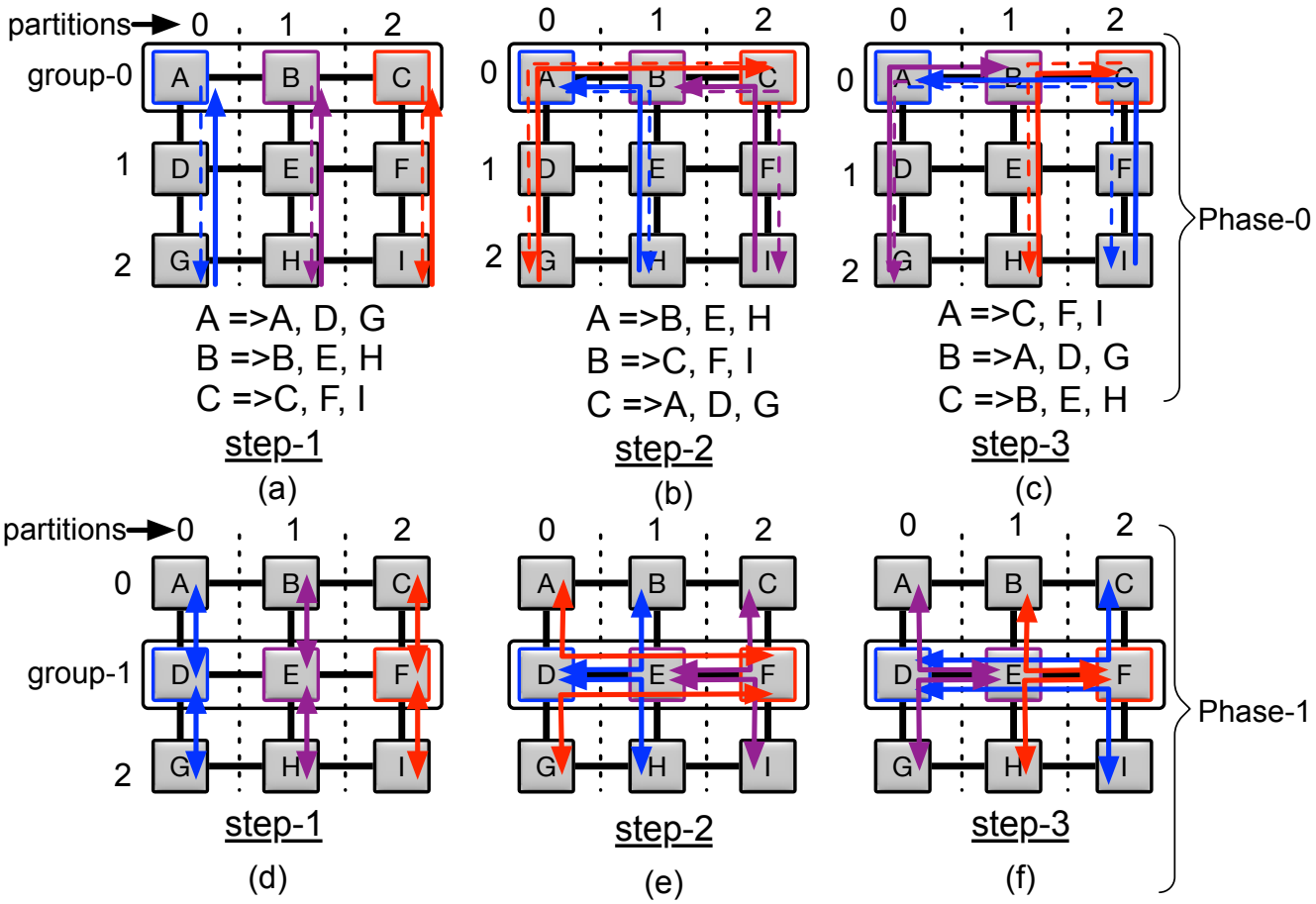


Figure 9.14: mSEEC implementation. The columns form "partitions" and the rows are "groups". In Phase-0, group-0 sends seekers to each partition. Phase-0's NICs send seekers to the routers listed after '=>'. Dotted lines represent the seeker path. FF-packet follows the same path in the opposite direction. No two paths overlap. Thus all FF-packets will simultaneously use minimal paths without collisions. In phase-1 double-ended arrows have been shown to convey seeker and FF-packet paths.

Figure 9.14 shows that each phase has a fixed number of steps, and in each step, there are fixed cycles involved for it to complete. For example, in Phase 0, Step 1, a *seeker* will take at most 2 hops to cover its own column(partition). However, in Phase 0 Step 2 the *seeker* from router-C's NIC will take 4 hops. This pre-computed static time-bound can be used to schedule each phase to realize mSEEC.

There exist alternate ways of partitioning the network to implement mSEEC, which we leave as future work.

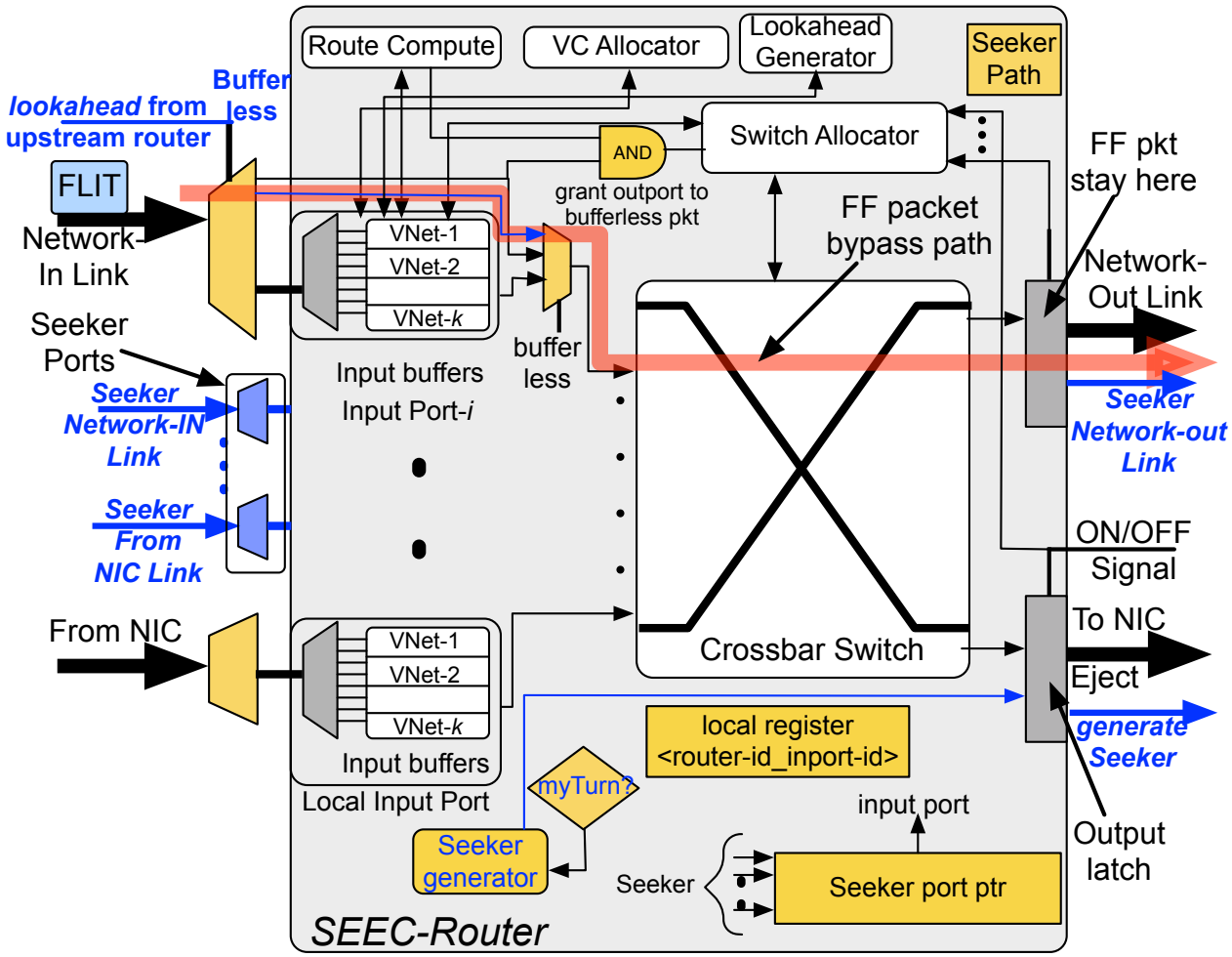


Figure 9.15: SEEC router microarchitecture, all mux signals are set up by the lookahead signal in advance, this allows seamless traversal of bufferless FF packet.

9.5 Router Microarchitecture

Figure 9.15 shows the microarchitecture of a SEEC router.

A FF packet entering the router bypasses the router input port and crossbar (setup in the previous cycle by the lookahead) and is latched at the output port, as shown in the figure.

The hardware required to implement SEEC is described next.

- Prioritization logic: This is added in the form of an extra mux, which allows the FF packet to take priority to acquire the output port over the current buffered packet in the network.

- Lookahead signal: A lookahead signal is sent one cycle in advance before the FF packet traverses the network, and sets the mux and prioritization logic appropriately. It is important to note that lookaheads are generated by input ports and not by the NICs. At the destination router, these lookaheads are used to temporarily stall the ongoing ejection (if any) to prioritize the ejection of FF-packet.
- Seeker logic: Seeker is inserted into the network in a round-robin fashion. Once the router has ejected a FF-packet, it signals the next router to insert the Seeker. Seeker also searches the appropriate router/input port in a round-robin fashion. Destination router keeps track of the last router/input-port from which it made the FF-packet. Seeker follows the predefined path which visits each router of the network at least once in a cyclic fashion.
- local-register: Each router has a local register which stores the $< router - id\_inport - id >$ from where the last FF packet was selected for ejection at this NIC. The seeker generated by this NIC would start looking for the packet starting from this $< router - id\_inport - id >$, to maintain a round-robin search policy.

Seeker port ptr is used to provide the round-robin priority to all the input ports of the router to get the chance of becoming FF, after consuming the seeker. The signals mentioned in the SEEC router (Figure 9.15) are setup by the lookahead signal prior to the arrival of FF packet. The highlighted components in the router are unique to SEEC. These components include a 16-bit local register, a small table holding the seeker path, and a few muxes. The total area overhead comes to be around 2% over a baseline dimension-ordered (DoR) router, as we quantify in subsection 9.7.2.

9.6 SEEC across Buffer Management Schemes

Store and Forward. SEEC naturally works in Store and Forward. As the complete packet is present in the network buffers for SEEC to make this packet traverse to its destination with the help of FF control.

Virtual Cut Through (VCT). In VCT, both buffers and links are allocated at packet granularity. It might happen that the complete packet is not present in the buffer, but remaining flits would be following the head-flit in close succession. The seeker makes the Head flit of a packet FF and records this information at the router. The remaining flits of the packet that subsequently arrive follow the head in the same FF manner.

Wormhole. In wormhole, unlike VCT, buffers may be smaller than the number of flits in the largest packet. Allowing adaptive routing with wormhole flow control adds the constraint that VCs must only contain one packet at a time to avoid deadlocks [26]. This requirement makes it easy to extend SEEC to wormhole networks. The seeker need only examine the flit at the front of a given VC queue, only upgrading it if it is a head flit. If the head is upgraded to FF, the remaining flits of the packet that subsequently arrive in this queue are marked FF as well.

This approach will work even if the wormhole queue has the minimum depth of 1-flit. SEEC thus has added advantages over other deadlock-freedom schemes such as SPIN [41], SWAP[11], and BLESS[42] as it does not require packet truncation to support wormhole, not does it re-order flits within the packet [42].

9.6.1 SEEC/mSEEC over irregular Topologies

SEEC and mSEEC are not tied to any particular topology. For arbitrary irregular topologies, seekers from NICs can be sent over a fixed virtual ring path through the entire network [10].

If no packet is found, the seeker would naturally come back to the initiator NIC and get dropped. Then the next NIC will send a seeker over the same path in a round-robin manner and so on. For mSEEC, *independent* and non-overlapping seeker paths must be established, as discussed in section 9.4. We leave this for future work.

Table 9.4: **Key Simulation Parameters.**

Real application simulation parameters	
Core	16 cores; x86 ISA (PARSEC, SPLASH-2), 1 GHz Out of Order cores, No prefetcher
L1 Cache	Private, 32kB Ins. + 64kB Data 4-way set associative
Last Level Cache (LLC)	Shared, distributed, 2MB 8-way set associative
Cache Coherence	MOESI, VNet=6
Network parameters	
Topology	4x4 Mesh, 8x8 Mesh, and 16x16 Mesh (synthetic traffic) 4x4 Mesh (PARSEC and SPLASH-2)
Routing Algorithm	DoR (XY) Turn-model (West-First) Bufferless (CHIPPER, MinBD) Escape VC (West First in Escape VC) Fully adaptive random (SPIN, SWAP, DRAIN, SEEC, mSEEC)
Router Latency	1-cycle
Virtual Network	6-VNet (Escape VC, SPIN, SWAP) 1-VNet (DRAIN,SEEC, mSEEC) 2 VCs/VNet
Buffer Organization	Virtual Cut Through. Single packet per VC Mixed single-flit (request, ack) and five flit (response) packets
Link Bandwidth	128 bits/cycle

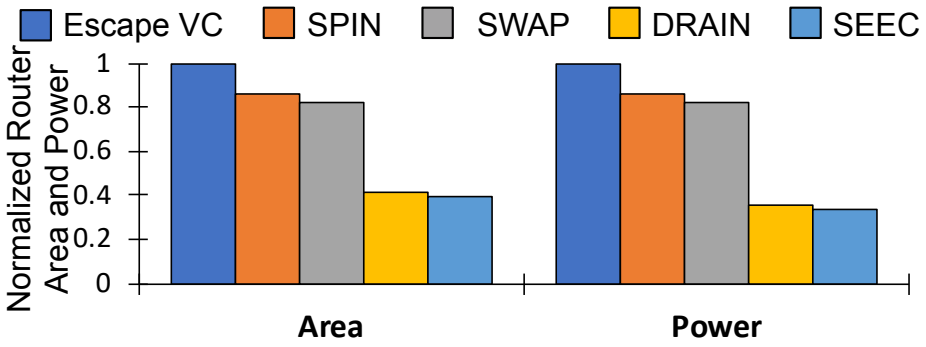


Figure 9.16: Router area and static power comparison.

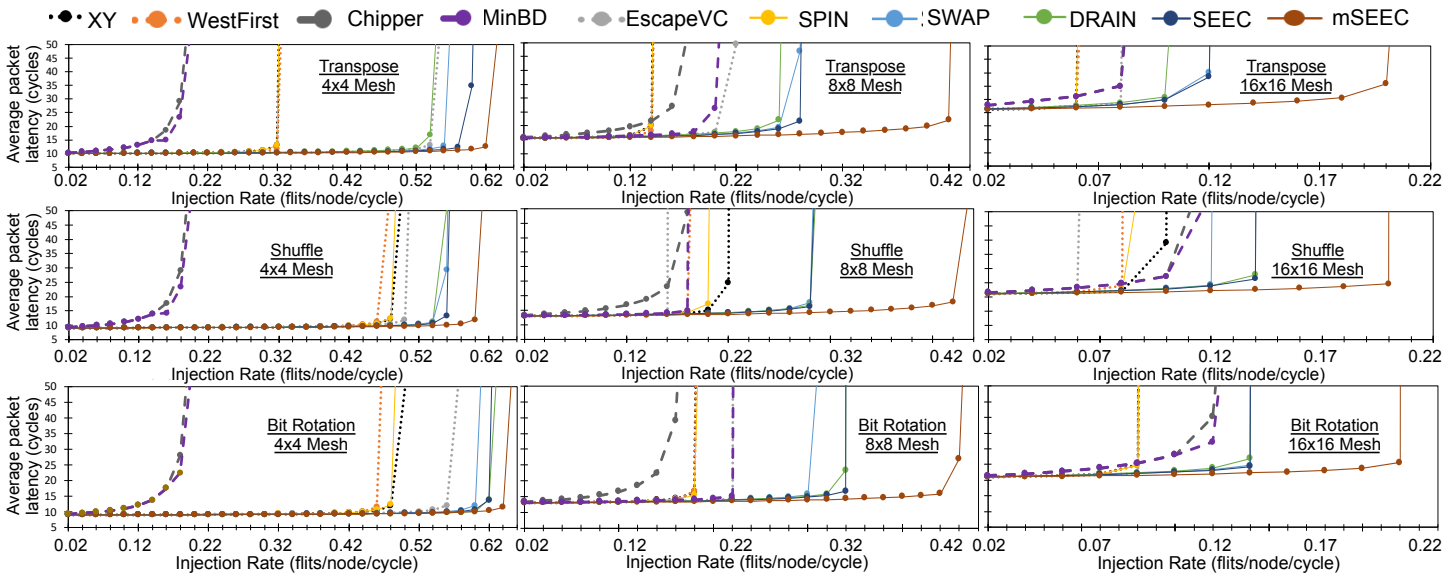


Figure 9.17: Latency curve for different traffic pattern across network sizes. SEEC and mSEEC out-performs the current state-of-art solutions

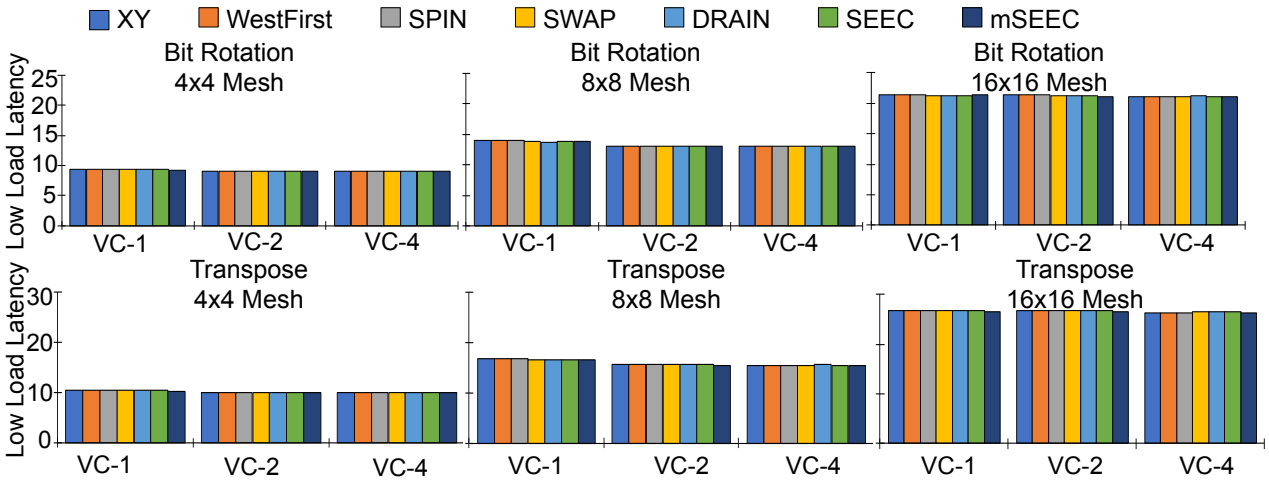


Figure 9.18: Low load latency for Bit Rotation and Transpose, traffic pattern. Topology of increased size 4×4/8×8/16×16 Mesh.

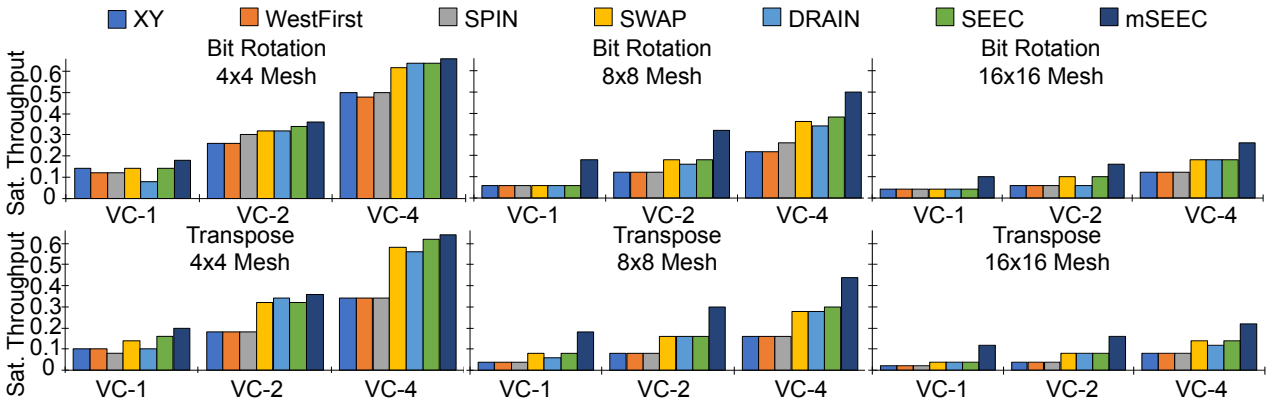


Figure 9.19: Saturation Throughput for Bit Rotation and Transpose traffic. Topology of increased size 4×4/8×8/16×16 Mesh.

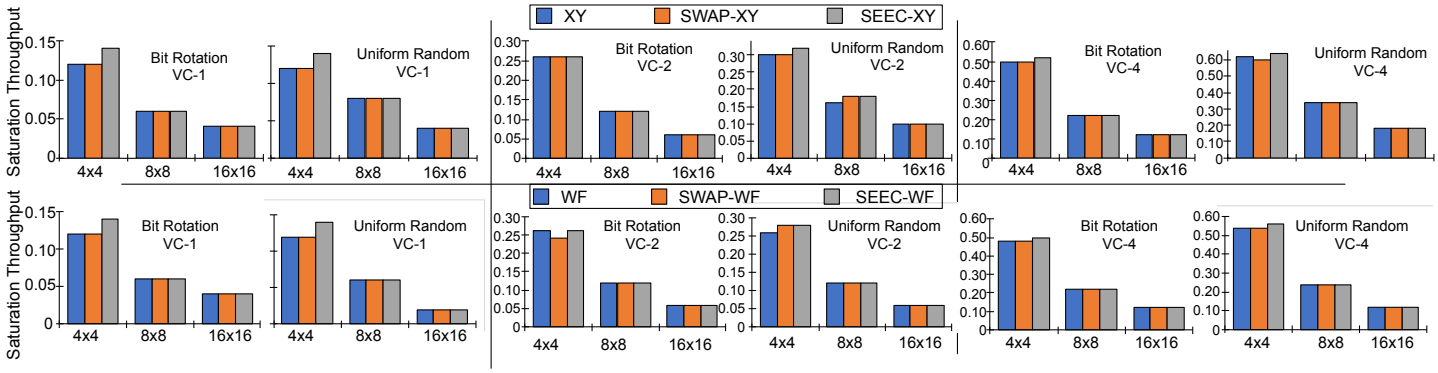


Figure 9.20: Baseline routing algorithm is deadlock free. SEEC provides higher performance (higher saturation throughput) when augmented with baseline routing algorithm. We evaluated it using XY and West First (WF) on a 4×4, 8×8 and 16×16 Mesh.

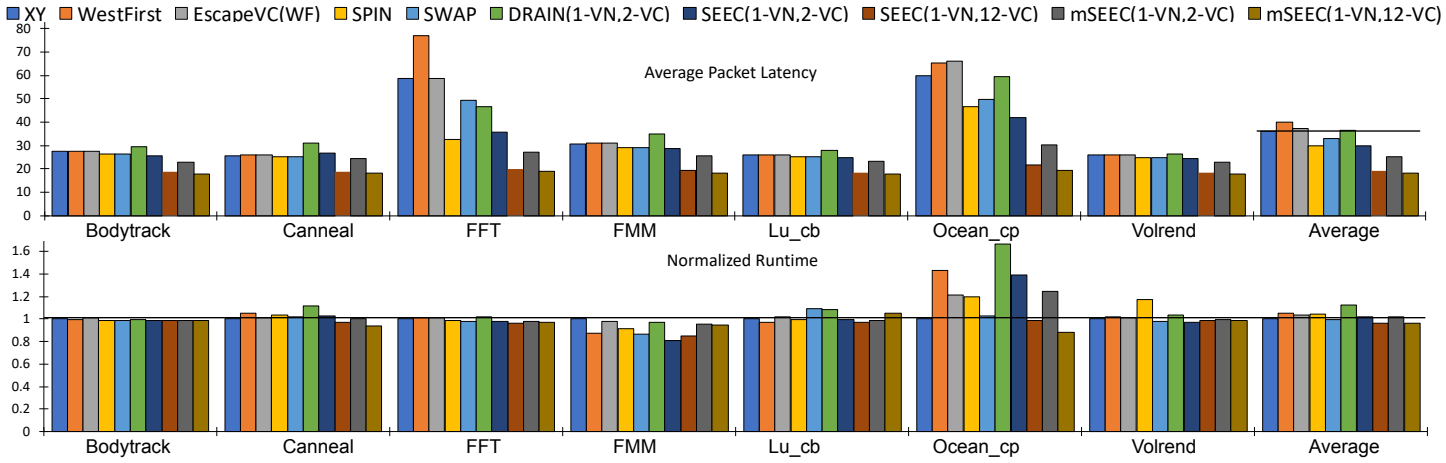


Figure 9.21: Average packet latency and normalized runtime (to XY routing) of applications in a 4×4 mesh using full system configuration with gem5[7] using MOESI_hammer[64] cache coherence protocol.

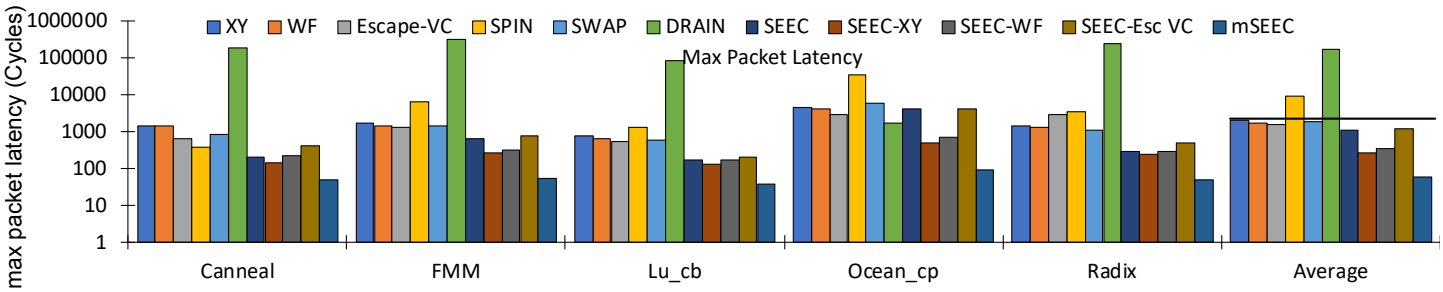


Figure 9.22: Experiment done on a regular 4×4 Mesh with different deadlock freedom schemes. The y-axis is a log scale latency in terms of maximum network cycles, that a packet has incurred.

9.7 Evaluation

9.7.1 Methodology

We evaluate SEEC using *gem5*[7] with the *Garnet2.0*[59] network model and *Ruby* memory model. We use *DSENT* to model power and area for an 11nm technology node. We compare SEEC and mSEEC against state-of-the-art techniques using both synthetic traffic and full-system applications. Table 9.4 lists our simulation parameters.

Baselines. We compare SEEC and mSEEC against state-of-the-art proactive, reactive and subactive solutions (subsection 9.1.2). For proactive, we pick XY, West-first, and Escape VC (using west first routing within the escape VCs and random routing in the regular VCs). We also run CHIPPER[45] and MinBD[44]. For reactive, we compare against SPIN [41] as it showed superior performance to other reactive solutions [38, 47]. For subactive, we compare against SWAP [11] and DRAIN [10]. SPIN, SWAP, DRAIN and SEEC/mSEEC all use fully adaptive random minimal routing to provide full path diversity to the network packets, unless explicitly stated otherwise. Adaptive routing uses the number of free VCs at the downstream routers to decide the direction, given a choice.

To avoid protocol-level deadlocks, XY, West-first, Escape VC, SPIN and SWAP all use 6 VNets as dictated by the MOESI hammer cache coherence protocol [7]. DRAIN and SEEC/mSEEC use 1 VNet. For performance simulations, we consider 2VC/VNet for

all systems. For SEEC/mSEEC, we evaluate two configurations: iso-hardware i.e., total number of VCs per input port is same (total of 12 VCs for 6-VN, 2-VC) and iso-VC-VNet i.e., number of VCs per VNet is constant irrespective of number of VNets (total of 2 VCs) We use the notation of (k-VN, p-VC) to further clarify the distinction. In our results, latency is shown in cycles and saturation throughput is shown in $packets\_received/node/cycle$.

Workloads. SEEC/mSEEC is evaluated on both real applications and synthetic traffic. Applications from the PARSEC-3.0 [33] and SPLASH-2 [32] benchmark suites are used.

We evaluate synthetic traffic with a mix of 1-flit and 5-flit packet sizes. The simulator was warmed for 1000 cycles to remove any effects due to empty queues in the packet latency statistics. Thereafter, a fixed number of tagged packets are injected by each node in the network. Once any core has injected its share of tagged packets, it keeps injecting normal packets. Simulation finishes when all the injected tagged packets are ejected from the network. Statistics are reported for tagged packets. This mechanism for collecting statistics ensures that any long tail latencies for tagged packets are captured in the final statistics.

9.7.2 Area and Power

The major benefit of SEEC is area and power savings due to its protocol-level deadlock freedom approach. Fig. 9.16 shows the normalized area and power breakdown, when compared to Escape VC, SPIN, SWAP and DRAIN.

We implement the configuration with the minimum number of buffers required by the router for each scheme for the NoC to function correctly. Thus, Escape VC uses 7 VC (1 VC per Vnet + 1 shared VC for adaptive routing), SPIN and SWAP use 6 VCs (1 VC per Vnet), DRAIN and SEEC use 1.

Escape VC requires an extra set of buffers at each input port, therefore it has the highest area and static power. SPIN and SWAP both can work with only one VC per virtual network. Since they are not free from protocol-level deadlocks, they have buffers per virtual

network. SPIN imposes ∼15% higher area and power compared to a standard VC router with XY routing, because of its deadlock detection and global coordination circuitry.

SWAP imposes ∼4% overhead over an XY router because of the swap_bus and swap management unit.

SEEC has limited overhead due to extra muxes and signaling. Overall, SEEC reduces the router area by 73% over escape VC and ∼50% over SPIN and SWAP. The SEEC Router has 77% lower static power compared to escape VC and ∼60% lower power compared to SPIN and SWAP. Recently proposed DRAIN [10] has similar area and power overhead as SEEC as it is also routing and protocol deadlock free with 1 VNet. DRAIN's overhead is the time-out counters and FSM for coordinating a periodic drain [10].

9.8 Analysis with Synthetic Traffic

Figure 9.17 shows latency-throughput curves for SEEC/mSEEC when compared against the baselines.

Results are collected with a 4-VC per input port router configuration. SEEC is as good as or better than all prior works across different traffic patterns and topology sizes.

On average across topology sizes, SEEC provides 65% higher throughput over the best proactive solutions (Escape VC[74]), 50% over reactive (SPIN[41]) and 10% over recently proposed subactive solutions (SWAP[11], DRAIN[10]). SEEC provides $2\times - 3\times$ throughput improvement compared to sister bufferless schemes (CHIPPER and MinBD). mSEEC pushes the envelope further, due to multiple FF packets creating express paths in the NoC. Its relative performance over SEEC increases as topology size increases from 4×4 Mesh to 16x16 Mesh because of higher path diversity. mSEEC performs 20% better in 4×4 Mesh, 25% better in 8×8 Mesh and 40% better in 16x16 Mesh on average than SEEC.

Figure 9.18 shows the effect of SEEC/mSEEC on low load latency with increased number if VCs and network size. Low load latency is independent of routing algorithm (given minimal routing) and the number of VCs. Low load latency does depend on the traffic pat-

tern and the size of topology. It monotonically increases with increase topology size. All networks have similar performance at low-loads. XY and West First both are minimal routing algorithms; at very low load there is no contention of packets. SPIN would not detect deadlock as its deadlock detection threshold will not be reached at low load. SWAP does not shuffle packets at low load because the downstream router will have empty ports. And latency hits due to misroutes in DRAIN are minimal. And express paths in SEEC/mSEEC do not help much as the network is uncongested anyway. mSEEC shows a slight latency improvement in a few cases.

Figure 9.19 compares the throughput of SEEC/mSEEC with other schemes, across topology sizes and number of VCs per input port. Here SPIN, SWAP, SEEC, and mSEEC use fully adaptive random routing. mSEEC provides the highest throughput in all of the cases, followed by SEEC an then SWAP, DRAIN. A generic trend shown by these graphs is the decrease in saturation throughput with increase in topology size as expected. In Figure 9.19, XY and WF routing suffer from lowest throughput because they restrict the path packets can take in the network. SPIN suffers lower saturation throughput than SWAP, DRAIN, and SEEC because at saturation the deadlock detection algorithm kicks in and its probes hinder the forward movement of packets. SWAP and DRAIN have lower saturation throughput than SEEC because of misrouting.

The difference between DRAIN and SWAP is that SWAP causes local pair-wise movement of packets while DRAIN proposes network-wide movement of packets. SEEC/mSEEC is free from such challenges and always routes packets minimally, while providing full path diversity.

9.8.1 FF vs Regular Packet Distribution

Figure 9.23 plots the percentage of FF packets (i.e., packets that got promoted to FF anytime during their traversal) received as a function of injection rate for uniform random traffic. It is not surprising to see that post saturation, heavy congestion and high likelihood

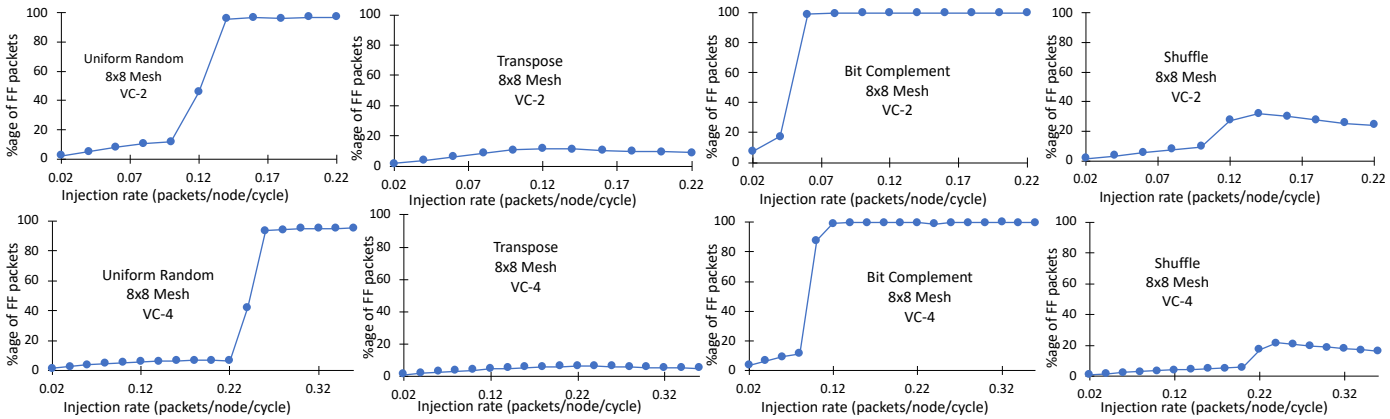


Figure 9.23: Percentage breakdown of FF versus Regular packets for synthetic traffic on a 8x8 Mesh.

of deadlocks in uniform random traffic [41] leads to almost every packet using SEEC to exit the network. For traffic patterns that have fewer average hops (e.g., shuffle) or less deadlock-prone (e.g., transpose [41]), the percentage was found to saturate at ∼10-30%.

Figure 9.24 breaks down the latency distribution of the received packets (both regular and FF) across the buffered traversal versus the bufferless (FF) traversal. A very interesting trend is observed here. While one may expect FF packets to be "faster" than regular packets, we observe the reverse,

both at low-injection rates and especially post saturation. This can be explained by the fact that the FF packets are those that were actually blocked at some router (and promoted via explicit seekers) and hence show a higher percentage of buffered time compared to unblocked packets that reached their destinations via a regular traversal. The bufferless component of the latency is quite small, as expected. These results points to potential future work on leveraging SEEC to enhance QoS.

9.8.2 SEEC over deadlock-free NoC

In the next experiment, we demonstrate the performance benefits of SEEC beyond deadlock-freedom. We run both SWAP and SEEC over NoCs using XY and West-first routing (inherently deadlock-free). Figure 9.20 shows the results for throughput. We found that SWAP

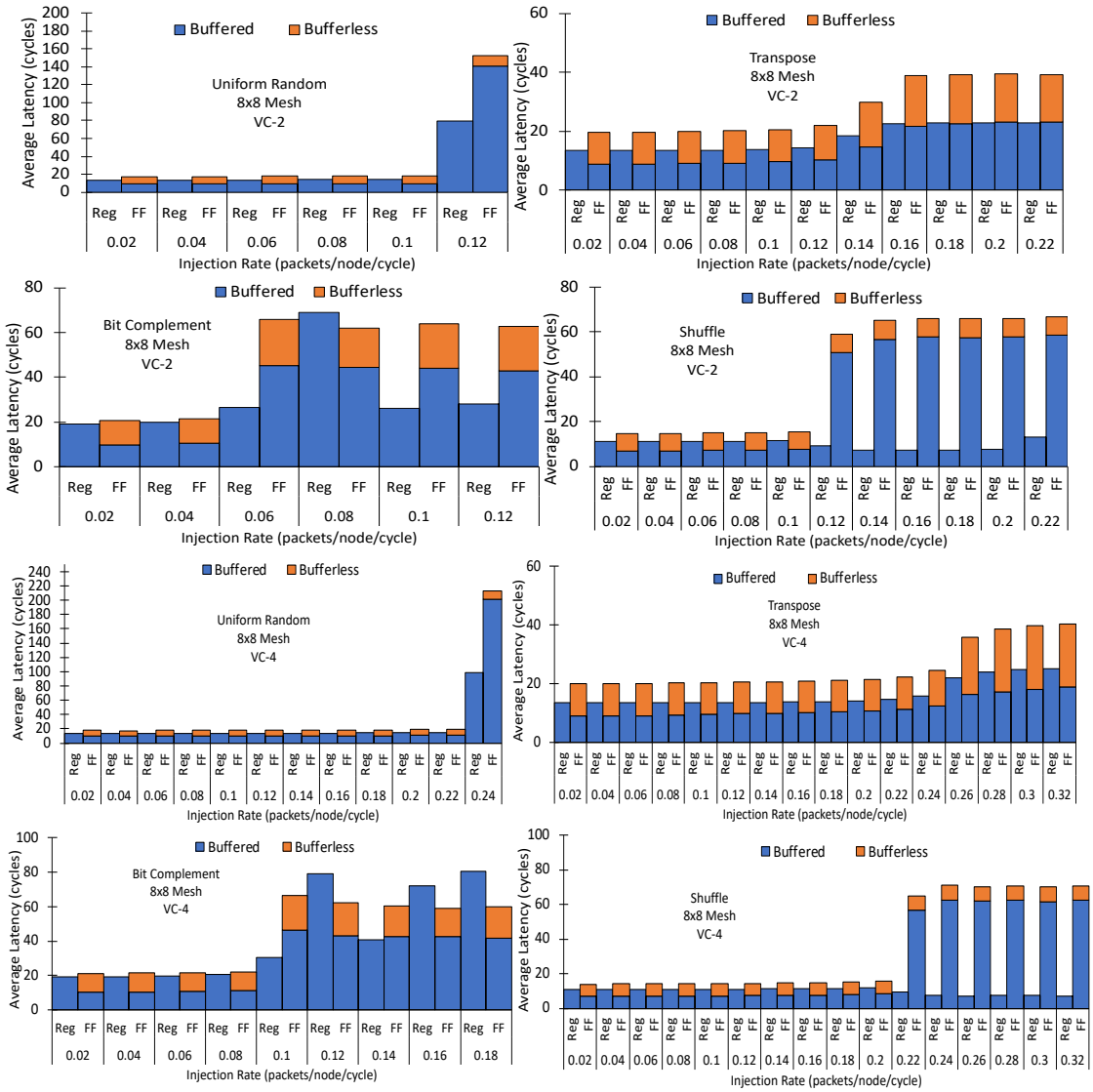


Figure 9.24: Latency breakdown of FF versus Regular packets for synthetic traffic on a 8x8 Mesh.

can either increase or decrease the saturation throughput of the baseline deadlock-free NoC, depending on whether the swaps led to a majority of packets getting routed towards productive directions or misrouted away from their destinations. This can be seen with the West First routing and bit-rotation traffic in Figure 9.20. SEEC, on the other hand, consistently either improves throughput, or keeps it the same. We observe higher throughput enhancements over baseline deadlock free routing on the 4×4 Mesh compared to 16×16 Mesh.

9.9 Application results

Figure 9.21 shows average packet latency and runtime improvement with SEEC/mSEEC when compared against XY, WF, Escape VC, SPIN, SWAP, and DRAIN. Other schemes use 6 VNets to provide deadlock freedom, while SEEC is configured with single VNet. We evaluated SEEC in two configurations: (1) only 2 VCs per input port (isoVC-per-VNet) and (2) total equal VCs shared across virtual networks. SEEC (1-VN, 2-VC) performs similar to SPIN and better than XY, WF, Escape VC and SWAP in average packet latency at $1/6^{th}$ buffer area. With equal number of VCs, SEEC shows 34% improvement in packet latency whereas mSEEC shows 35% improvement over subactive SWAP/DRAIN and around 38% improvement over proactive and reactive approaches at $1/6^{th}$ of hardware cost. At iso-hardware cost mSEEC provides 40% improvement on average. Both SEEC and mSEEC provide 5% average improvement in the total runtime of applications.

9.9.1 Impact on Application Tail latency

We measure tail latency on real applications running in gem5 full-system simulation. Figure 9.22 shows the tail latency incurred by the packets in the network. Note the Y-axis is a log scale. On average, XY, West-first and Escape VC have similar tail latencies; however, SPIN has an order of magnitude higher maximum packet latency. This is because the expensive deadlock detection of SPIN prioritizes the movement of deadlock detection probes over actual packets [41]. This can further slow down the movement of actual packets in the network. In the extreme case, this leads to slowdown for certain packets.

We also observe that DRAIN has the highest tail packet latency among all other schemes for MOESI_hammer cache coherence protocol; this is attributed due to frequent periodic mis-routing of packet as shown earlier in Figure 9.8.

SWAP performs similar to XY, WF and Escape VC. SEEC outperforms all baselines. Results on tail packet latency are further improved when SEEC is augmented with XY rout-

ing. We observed an order of magnitude lower latency with SEEC-XY compared against other schemes.

9.10 Discussion

SEEC has some similarities with previous works such as Token Flow Control[83] and Express Virtual Channel[82], as SEEC also allows packets to bypass the intermediate routers. However, *main* takeaway is that SEEC is a unified approach to provide routing and protocol level deadlock freedom, while previous works ([83], [82]) caters to provide higher performance in the network. We further delineate these work from SEEC in the following sub-sections.

9.10.1 SEEC compared to Express Virtual Channel (EVC)

Express Virtual Channel allows packets to zoom multiple intermediate routers from its source node to destination node using prioritization logic. There is a little overlap of EVC compared to SEEC as latter also uses prioritization logic to allow FF packets to zoom through the network till ejection. Apart from it EVC is an orthogonal approach to improve network performance by enabling higher network link utilization where SEEC is a unified approach to provide routing level and protocol level deadlock freedom in interconnection networks. EVC uses extra virtual channel per input port of the router in the form of express virtual channel (evc) and remaining virtual channels are tagged as normal virtual channel (nvc). Packets on evcs are prioritized over nvcs. This prioritization logic enables the evcs packets to bypass few stages in the router pipeline and hence can use more cycles in traversing the links compared to remain buffered in the routers. This results in higher link utilization. To allow evcs-packets to bypass the intermediate routers, credit signaling of evcs is used which bypasses intermediate routers. This results in higher buffer turnaround time, because of which evcs are expected to be deeper compared to nvcs. EVC uses Dimensional ordered Routing (DoR), XY routing, to provide routing level deadlock freedom.

Although not explicitly stated we assume EVC uses traditional method of separate virtual network for each message class to provide protocol level deadlock freedom. Therefore, EVC incurs many-times higher area and static power budget compared to unified approach such as SEEC.

SEEC on other hand is a unified approach to provide routing and protocol level deadlock freedom. It can work with minimal number of VCs (1 per input port) of the router. All packets in SEEC uses full-path diversity using minimal adaptive random routing. SEEC provides deadlock freedom by using round-robin policy both for choosing the destination routers for ejecting FF packet and for choosing the intermediate router for upgrading a buffered packet to FF packet.

9.10.2 SEEC compared to Token Flow Control (TFC)

TFC is a flow control to enable distributed adaptive turn model routing, such as West First routing in the network. It is flexible compared to EVC as routers can track the congestion at the downstream routers, in the form of buffer occupancy using *tokens*, up to a fixed maximum hop distance. Tokens are generated by the source router to communicate its buffer availability. Normal tokens used in TFC are a hint about the buffer occupancy at next few downstream routers.

TFC provides a way to implement adaptive turn model routing, whereas SEEC provides full path diversity (no turn restrictions) to the packets with adaptive random routing. SEEC only tracks the number of free buffers at the next hop routers, which is naturally available in the form of credits with credit flow control to the upstream router. SEEC uses this information to route the packets to the uncongested path. TFC on other hand uses complex token mechanism as a way to *predict* the uncongested path to adaptively route packet in North-East or South-East direction (with limited path diversity).

Moreover, with different types of tokens and their respective max-hop tracking, there is wiring overhead and route compute overhead in TFC which makes the router design

complicated compared to SEEC which is free from such complexity.

Finally, TFC is neither routing deadlock free nor protocol deadlock free. It leverages on already deadlock free turn model routing and improves performance of the network by providing distributive adaptive routing. The maximum hop count tracked by tokens puts indirect restriction on the buffer depth at the input port of the router, because of buffer turnaround time. Therefore, TFC has higher area and static power compared to SEEC, which is free from such complicates.

SEEC follows reservation of buffer-slot and therefore guarantees the availability by the time FF packet reaches its destination node using Free Flow control. Therefore, SEEC is free from buffer turnaround time considerations. mSEEC provides the maximum performance by allowing multiple FF packets to eject out from the network and hence is a scalable version of SEEC.

SEEC and mSEEC are the final techniques in the class of subactive techniques of providing deadlock freedom presented in this thesis. As shown in Figure 9.20, SEEC and mSEEC can be used to improve throughput of already deadlock free routing algorithm (XY, West-First, etc) as FF packet can bypass congestion and reduce buffer turn around time (Figure 9.4, Figure 9.4, and Figure 9.6). *In this case SEEC/mSEEC can be modified to create bypass paths from source-NI all the way to destination-NI.* We believe this would provide superior performance as the length of bypass path will always be maximum for a given FF packet. We believe this to be a good follow-on work to further evaluate the limiting performance of SEEC/mSEEC schemes. In next chapter we will conclude the work presented in this thesis, we will quantitatively compare the performance of each subactive technique on synthetic traffic patterns with different router configurations and finally discusses the future directions.

9.11 Chapter Summary

In this chapter, we proposed a new technique called SEEC to provide deadlock freedom. The key idea is to enable all packets to get on express paths that bypass buffering at intermediate routers and zoom to the destination via a novel flow-control called Free Flow. This provides both deadlock-freedom, as well as higher throughput (by bypassing congestion). We also present mSEEC, a mechanism to allow multiple simultaneous free-flow packets in the NoC with non-overlapping paths. SEEC is the first work that simultaneously provides routing and protocol level deadlock freedom without any turn restrictions, extra VCs, deadlock-detection, or misrouting required. SEEC opens up further research directions in deadlock-freedom, network performance and QoS.

CHAPTER 10
CONCLUSION

As we continue to scale more transistors within the chip, multicore designs and 2.5D/3D based designs, interconnection network will continue to play a vital role to determine the overall performance of the system and an area power envelope.

Deadlocks (both routing and protocol level) are a correctness issue and continue to pay a decisive role in designing interconnection network for such heterogenous systems. This thesis addresses this age-old problem of deadlocks in the interconnection network and does extensive literature review in this domain. This thesis classifies the prior solutions as either *Proactive*, which avoids the deadlocks or *Reactive*, which allows the deadlock to occur, but then detect and recover from deadlocks.

10.1 Dissertation Summary

This thesis introduces a *new* class of deadlock freedom solutions, which are called as *Subactive* solutions. These solutions neither avoids deadlocks nor detects deadlock, instead these solutions provide *periodic coordinated* packet movement in the network which flushes out any deadlock that may have occurred in the network. The thesis proposes five subactive techniques, namely: BBR, BINDU, SWAP, DRAIN, and SEEC.

Some of the solutions provide simultaneously routing and protocol level deadlock freedom while others are shown to provide only routing deadlock freedom. We believe that those solutions (BBR, and BINDU) can also be augmented to provide simultaneously routing and protocol deadlock freedom.

This thesis shows that these solutions are not only high performant, as they can bypass congestion (for example SEEC) or better load-balance traffic but also provide notable reduction in area/power budget of Network-on-Chip. In particular cache coherence protocols

known to have 3 to 6 virtual networks can now work with only one unified virtual network. This directly translates to 3 to 6 times static area/power reduction of the network.

Protocol deadlock freedom at lower area and power budget is particularly important during this era of heterogeneous computing. Myriad of accelerators are now getting integrated on chip and are becoming a part of coherence protocol. This has resulted in complex cache coherence protocols which require complex transactions even more message classes. This gets translated into a greater number of virtual networks in the on-chip network, so that packets from one message class does not block the packets from another message class or vice versa. A scheme that can provide protocol deadlock freedom with one unified virtual network becomes highly desirable. *Subactive* schemes naturally provide protocol deadlock freedom with one unified virtual network, highlighting its relevance in current time.

Contributions from this thesis go beyond On-Chip network and can be applied to off-chip system level networks where link latencies are comparatively higher and some of the prior work might not be suitable.

10.1.1 Thesis Statement

This thesis shows periodic coordinated packet movement is sufficient to resolve both routing-level and protocol-level deadlocks in any interconnection network

Property of sub-active approach. All subactive approaches follow minimal routing and occasionally provides coordinated forced movements to packets in the network. Since packets follow minimal routing most of the time, subactive solutions do not impose the high-performance overhead penalty. Moreover, these coordinated forced movement has a dual benefit. Firstly, it naturally resolves any deadlock that may occur due to minimal random routing at the same time enjoying full path diversity offered by the topology. Secondly, because of the nature of the forced movement of packets, a response message is never stuck indefinitely behind a request message. In general, no two types of message-packets are stuck indefinitely behind each other. Therefore, it provides protocol level deadlock

freedom as well.

10.1.2 Discussion

Here we briefly paraphrase the applicability of *subactive* class of techniques with respect to on going heterogeneity of the SoC (chiplet based designs) and different irregular SoC topologies because of this heterogeneity and dynamic/static faults as we go into sub-micron transistor size.

Heterogeneous Systems. Designing routing algorithms for chiplet-based architectures [18] is challenging. Multiple independently designed and verified networks must be connected through an interposer network while maintaining deadlock freedom. Just because each individual network is deadlock free does not guarantee that the composed network will be deadlock free. Recent work provides deadlock freedom through turn restrictions when entering and leaving chiplet-level networks. An alternative and potentially higher performance solution would be to enable deadlock freedom through subactive solutions. This would allow arbitrary vendor topologies to be connected together in a deadlock free manner without requiring any costly hardware overheads.

Random Topologies. Random topologies [90, 19] offer reduced diameter and lower average hop count making them attractive from a performance standpoint; however, high-performance, deadlock-free routing on random topologies is challenging. For example, Dodec [19] uses turn elimination to achieve deadlock freedom; however, given that its low radix routers lead to fewer turn options, some routing schemes result in non-minimal paths which can hurt performance. Maintaining all available turns would result in better performance for these networks and easier design flow; to achieve high performance, Dodec uses fully adaptive routing coupled with an escape virtual channel that uses a deadlock free routing scheme. Similarly, Koibuchi et al. [90] use an adaptive routing algorithm combined with up*/down* routing on the escape virtual channel for deadlock freedom. Additional virtual channels increase the cost of interconnection networks; lower-cost solutions that

mitigate deadlock, such as SWAP, BBR, BINDU, DRAIN, and SEEC could be a significant improvement.

10.2 Quantitative Comparison of Subactive Techniques

In this section we compare the subactive techniques proposed in this thesis. We used synthetic traffic pattern to observe the performance of each scheme individually irrespective of whether the scheme provides routing + protocol level deadlock freedom or only routing level deadlock freedom.

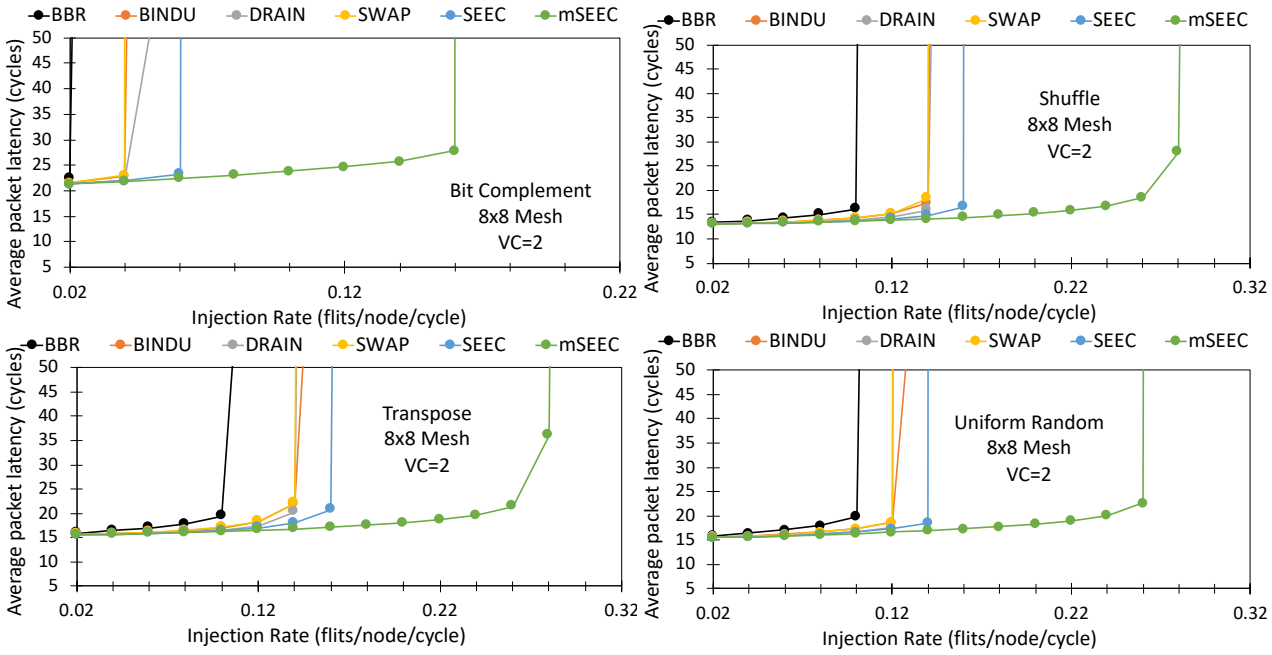


Figure 10.1: Quantitative Comparison of subactive techniques proposed in this thesis for VC=2 on 8x8 Mesh.

Using synthetic traffic pattern, we could determine with given hardware resources, how would each technique compare against each other. We also configured router with VC=2 and VC=4 and drew the latency injection rate curve as shown in Figure 10.1 and Figure 10.2 respectively. We observe the consistent behavior across different router configurations. The key observation on relative performance of subactive techniques is in agreement with the qualitative comparision table we drew in earlier chapter.

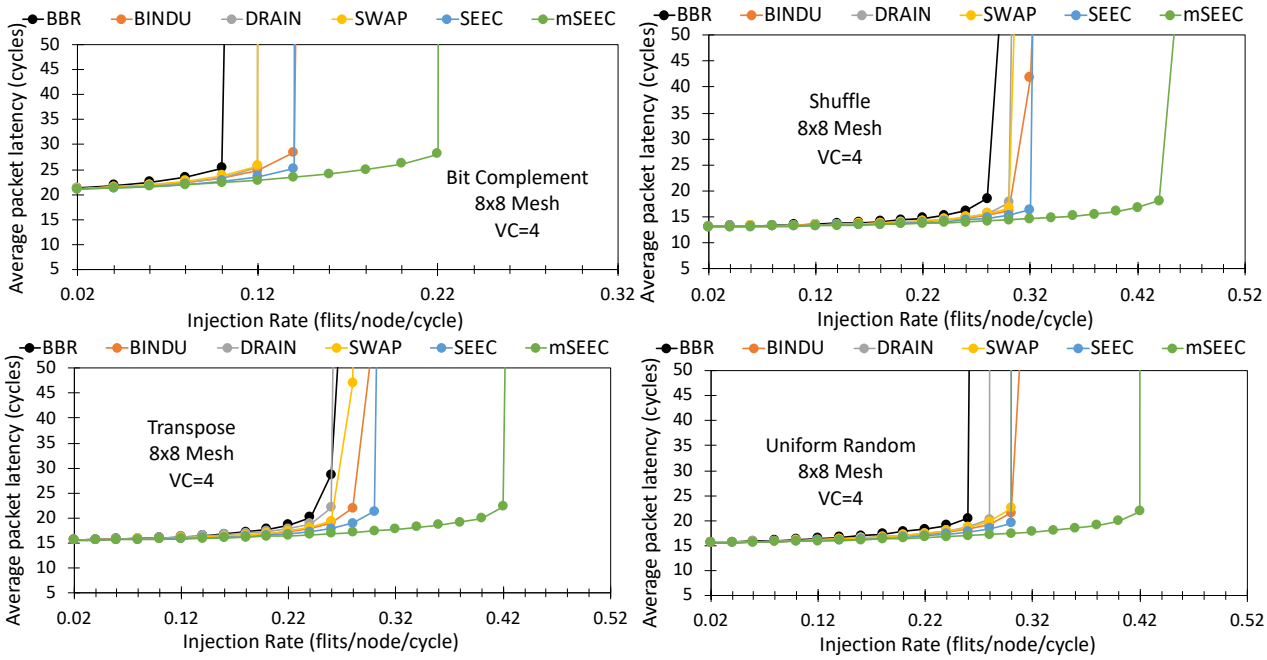


Figure 10.2: Quantitative Comparison of subactive techniques proposed in this thesis for VC=4 on 8x8 Mesh.

Particularly we observed, BBR to be consistently under performing among all the schemes, this is because of higher number of bubbles present in the network, which limit the throughput of network. Performance among BINDU, DRAIN, and SWAP is close to each other across traffic patterns, this is because all these subactive techniques can cause misrouting of packet. Under certain traffic patterns we observe SWAP to perform better than DRAIN, this is because SWAP takes local decision of initiating *swap*. In certain condition when packet at the downstream router is at its destination, or when the input port of downstream router has free buffer available, *swaping* of packet is not performed. This futer limits the mis-routing caused by SWAP. Finally, we observe SEEC performing better than the previously proposed subactive schemes, this is because it does not misroute packet. An important observation is that mSEEC provides significantly superior performance than all of its sister schemes, as like SEEC it does not misroute packets, moreover, it allows multiple FF packets (bigger the network size, higher the path diversity, more FF packets) to simultaneously zoom through the network to reach their destination.

10.3 Future Direction

This thesis proposed to resolve one of the fundamental challenge of deadlocks in interconnection networks. Coordinated packet movement techniques introduced in this thesis frees NoC designer from deadlock freedom considerations, and NoC designer can explore other ways of using NoC to in tandem with other subsystems. Here we discuss some of the future research direction where techniques proposed in this thesis can be helpful.

10.3.1 Unified Ejection Queues at End Nodes

Resolving deadlocks using some of the techniques, for example, SWAP, DRAIN and SEEC, reduced area/power budget of NoC routers considerably. These techniques, however, still assume dedicated injection/ejection queues for each message class at the end nodes. There is a rich research opportunity to apply these techniques at the injection/ejection queues to further reduce the complexity area/power budget.

10.3.2 Quality of Service

Some of the techniques proposed in this thesis can be extended to provide *Quality of Service* in the network without worrying about deadlocks. This is specifically important in SoC where heterogeneous IPs have different latency and bandwidth requirements.

10.3.3 Swap Channel

Swap Channel is a natural next step from the SWAP[11] work presented in this thesis. Swap Channel proposes a low-cost replacement for *Virtual Channel*. Virtual Channel has many favorable characteristics for example it avoids deadlocks, and ameliorate the affect of Head-of-Line blocking in NoC. It is for this reason, Virtual Channels are also called as *Swiss knife* of NoC. Appendix-A shows that *swap* operation can be used to ameliorate HoL blocking effect in wormhole routers at much lower area/power overhead and at comparable

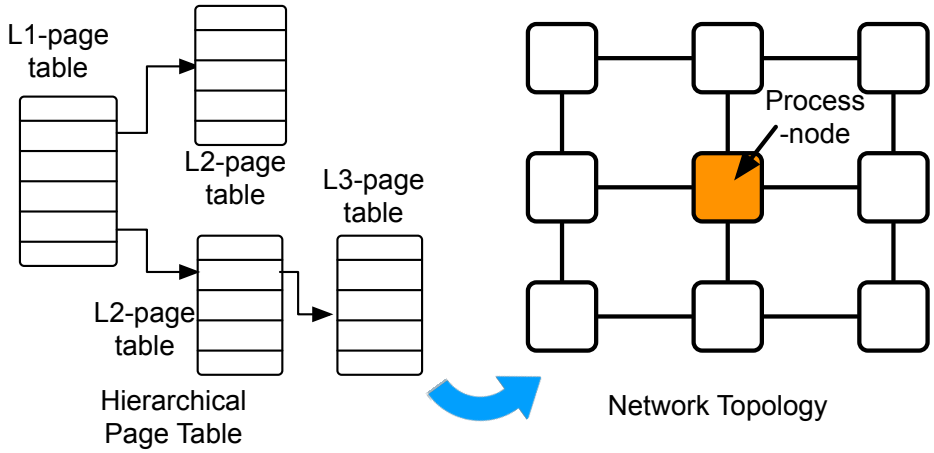


Figure 10.3: Co-locating Hierarchical page tables of the process closer to the node where the process is running can enable virtual address translation during network traversal.

performance as that of virtual channel routers.

Swap Channel aims to combine two works: SWAP[11] and SwapNoC[91] to provide light wight replacement of Virtual Channel.

10.3.4 Using NoC buffers as Victim Cache

On Chip network can experience varying degree of traffic loads. During high network-load all the network buffers could be filled with packets from different message classes, however during low network traffic these buffers would sit idle, contributing towards the static leakage power of the chip as shown in Figure 3.14 depending upon cache coherence protocol used. Instead we can use those buffers as variable size *victim cache* for L1 Data and L1 instruction cache. Therefore, during low network activity, network buffers can be used to cache useful data, which can lower the *Average Memory Access Time (AMAT)* [1]. Moreover, to further improve the memory system performance, network buffers can be used to store the prefetched data, to avoid the cache pollution.

10.3.5 NoC design to support Virtual Memory

Virtual Memory has now become integral part of most of the general-purpose computers. It is supported by the operating system. Dedicated hardware in the form of *TLBs* and *Page table walkers* are provided by the hardware to improve the performance of virtual memory subsystem. However, many features of hardware architecture are not exposed to operating system. For example, Operating system is oblivious to the NoC topology and micro architectural features a NoC can support.

If NoC latencies and topology are exposed to operating system, then performance can be further improve. For example page-walk can be accelerated by *caching* the *Page Table Pointers* of lower level page-tables closer to the router where process is running (Figure 10.3). This way by the time the original virtual address request reaches the memory controller it would need the last *Page Table Entry*, of the hierarchical radix page table, for from virtual to physical address translation.

10.4 Conclusion

The work in this thesis is properly motivated by showing that deadlocks are very rare in practice and yet a fundamental correctness problem. Hence it must be solved for functional correctness. This thesis also demonstrates that virtual networks are severely underutilized yet necessary to avoid protocol level deadlocks in the network. Therefore, clearly, there is a need to have an efficient class of solutions which resolves both protocol level and routing level deadlock simultaneously.

To conclude, this thesis proposed a new class of deadlock freedom using the oblivious coordinated movement of packets in the network. We called it the subactive class of solution and proposed different techniques using, Bubble, SWAP, DRAIN, and SEEC to achieve deadlock freedom. This thesis showed that many proposed solutions resolve both the routing level and protocol level deadlocks at the same time. They are amenable to

[1]Using NoC buffers as victim cache might require modifications to standard cache coherence protocol.

Table 10.1: **Comparison of prior solutions (proactive and reactive) for routing-level and protocol-level deadlock freedom with new *subactive* class of solutions. The new *subactive* class of solutions are contribution of the thesis.**

	Types of solutions	High Performance	Low Area and Power	Low Hardware Complexity	Resolves Routing-Level Deadlock	Resolves Protocol-Level Deadlock
Turn Restrictions [51]	Proactive	✗	✓	✓	✓	✗
Escape VCs [52]	Proactive	✓	✗	✓	✓	✗
Virtual Networks [53]	Proactive	✓	✗	✗	✓	✓
SPIN [41]	Reactive	✓	✓	✗	✓	✗
BBR [8], BINDU [9], SWAP [11], DRAIN[10], SEEC	Subactive	✓	✓	✓	✓	✓*

*BBR, and BINDU are proved to provide routing deadlock freedom. However, we believe these techniques can be extended to resolve both routing and protocol level deadlocks. Other subactive techniques: SWAP, DRAIN, and SEEC are proved to provide both routing and protocol deadlock freedom

static/dynamic irregular topologies which can arise either due to power gating and/or unreliable silicon. These characteristics of proposed set of solutions make them a lucrative choice in the present time.

Table 10.1 qualitatively delineates the subactive class of solutions from previously proposed approaches.

Appendices

APPENDIX A

LIGHTWEIGHT EMULATION OF VIRTUAL CHANNELS USING *SWAPS*

Virtual Channels (VCs) are a fundamental design feature across networks, both on-chip and off-chip. They provide two key benefits - deadlock avoidance and head-of-line (HoL) blocking mitigation. However, VCs increase the router critical path, and add significant area and power overheads compared to simple wormhole routers. This is especially challenging in the era of energy-constrained many-core chips.

The number of VCs required for mitigating HoL depend on runtime factors such as the distribution and size of single and multi-flit packets, and their intended destinations. In some cases, more VCs are beneficial, while in others they may actually harm performance, as we demonstrate. In this work, we provide a low-cost micro-architectural technique to emulate the HoL mitigation behavior of VCs inside routers, without requiring the expensive data path or control path (vc state and vc allocation) for VCs. We augment wormhole routers with the ability to do an in-place *swap* of blocked packets to the head of the queue. Our design (SwapNoC) can operate at low area and power specs like wormhole designs, without incurring their HoL challenges.

A.1 Introduction

Networks-on-Chip (NoCs) are prevalent across manycore CMPs and SoCs today. NoCs need to provide a delicate balance between meeting application's communication latency and throughput demands, while consuming as little real estate in terms of area and power as possible. NoC power continues to remain a concern [92] in the many-core era.

Wormhole routers are the simplest routers and use a simple queue at every input port. The challenge with wormhole routers, however, is *head-of-line (HOL) blocking*. Fig. A.1(a) shows an example. The brown packet which wants to go east is blocked by the yellow

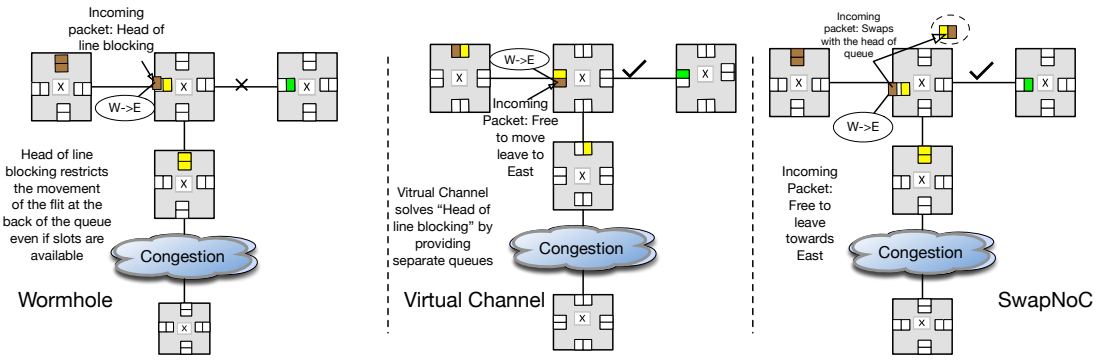


Figure A.1: Wormhole vs. Virtual Channels vs. SwapNoC

packet that wants to go south, due to congestion at the south output port.

To avoid HOL, the standard technique is to use Virtual Channels (VCs). VCs are like lanes on a highway, that can allow packets using different output ports to not get blocked by each other. Fig. A.1(b) shows that VCs allow the brown packet to traverse to its destination without getting blocked. VCs are prevalent across commercial and research NoCs [93, 94] today[1].

The challenge with VCs, however, is three-fold:

Latency. Flits need to know which VC to sit in when arriving at a router, and require a VC allocation step [93]. More the number of VCs, the larger is the critical path for this step. Most VC routers require at least 2-3 cycles even in the most state-of-the-art designs [93]. VCs also add significant area and power overheads.

Area and Power. Fig. A.2 plots the area and power requirements of a wormhole, SwapNoC (this work) and VC routers, as a function of increasing number of buffer slots. We implemented all three designs in RTL, and the numbers are from synthesis using Nangate 15nm FreePDK [95]. We can see that the VC area and power grows to more than 2$\times$ that of wormhole as the number of buffer slots go up. The reason is that VCs are often implemented as multiple independent FIFOs, each with its associated state, and muxes to write to and read from one of these FIFOs. Alternate organizations with trade-offs are discussed

[1]This work targets the performance enhancement (HoL mitigation) aspect of VCs. Some VCs would still be required for avoiding protocol or routing deadlocks.

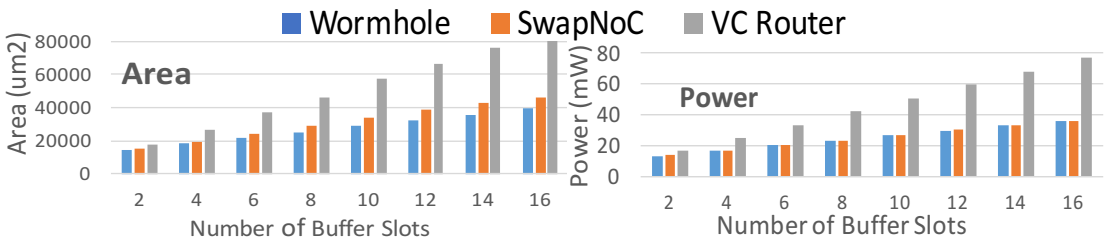


Figure A.2: Router Area and Power as a function of buffer slots

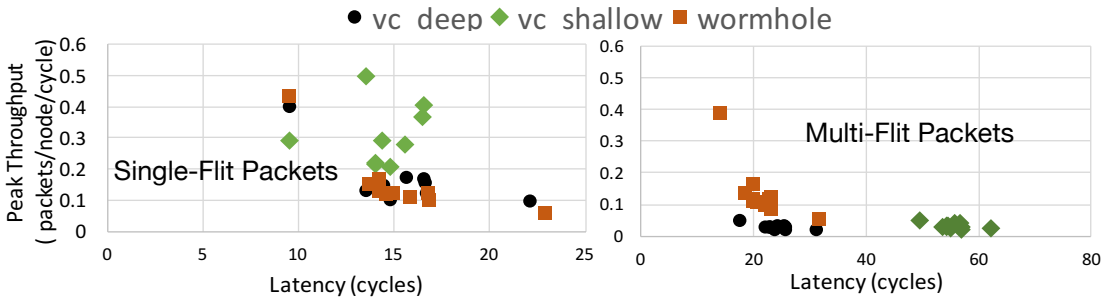


Figure A.3: Performance of Wormhole vs. VC-based Designs.

in Section A.2.

Traffic-dependent performance. We performed a design-space exploration by stressing a NoC with myriad synthetic traffic patterns, and observed the low-load latency and saturation throughput across wormhole and VC-based designs. For a fair comparison, we assumed N buffer slots at each input port, which could either all go into one N-deep wormhole FIFO, or shallow VCs (N VCs, each 1-flit deep) or deep VCs (N/2 2-flit deep VCs, N/4 4-flit deep VCs and so on). Fig. A.3 plots the distribution of latency and throughput across the designs and patterns, normalized to a wormhole NoC for each traffic pattern. We notice that single-flit packets favor shallow VC designs for throughput, while multi-flit packets favor wormhole NoCs for throughput. Moreover, wormhole NoCs always provide lower latency due to simpler routers.

These 3 observations should make us re-think the cost-benefit of VCs in many-core NoCs. In this work, we propose an alternate light-weight technique to reduce HoL, without requiring VCs. We identify that a blocked flit in a queue can in principle *swap* with the one at the head of the queue. This is possible in hardware because of the cyclic shift-register

behavior, as Fig. A.4(b) shows. A cyclic shift-register can allow the bits at the output of two latches to get swapped at the clock edge, without requiring another temporary latch. This is unlike the software notion of swap where a temporary buffer is required for a swap. Leveraging this principle, we allow HOL blocked flits that can leave the router to swap to the head of the queue and proceed. We enhance wormhole routers with this feature and call our design the *SwapNoC*. Fig. A.1(c) shows how the blocked packet can swap to the head and can traverse to its destination without getting blocked.

SwapNoC can provide the latency, power and area benefits of wormhole NoCs, and emulate the throughput benefits of VCs. The neat feature of our design is that it can adapt to both single and multi-flit packets, without requiring explicit VCs partitioned into a pool of shallow and deep queues, which can add performance loss when done at design time [96, 97] and add complexity when done dynamically at runtime.

Compared to wormhole and VC baselines, SwapNoC demonstrates up to a 3.7× reduction in latency, and 70%-95% improvement in throughput across synthetic and real workloads. It has 2× lower area and 2× lower power than traditional VC based routers, and adds less than 1% power and 8% area overhead over a wormhole router.

Sec. A.2 discusses related work. Sect. A.3 presents the microarchitecture. Sec. A.4 shows evaluations, and Sec. A.5 concludes.

A.2 Background and Related Work

A.2.1 Flow Control Techniques

To transmit messages within network, usually one of the following five *flow control* techniques are used, with increasing order of complexity:

Circuit Switching. The whole path for the transmission is blocked until the packet is transmitted. Since this leads to poor link bandwidth utilization, it is not preferred in NoCs.

Packet-switching with Store-and-Forward. Packet are stored completely in each router and only allowed to leave once the entire packet has arrived.

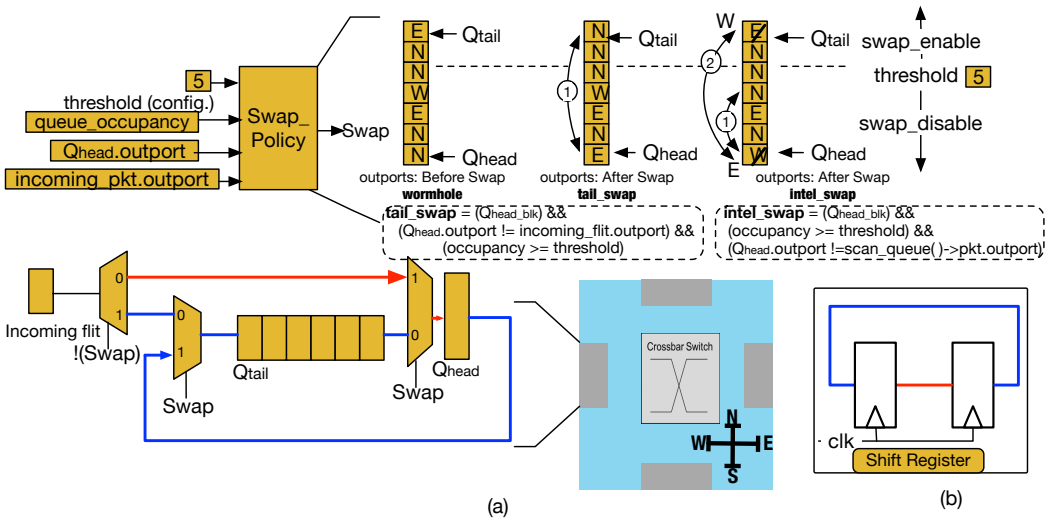


Figure A.4: Swap NoC Microarchitecture. For illustration purposes, we show the swap for a single flit packet. Presented above is an example of tail_swap and intel_swap policies. Suppose the North output port is blocked. tail_swap enables the packet going East at Q_{tail} to swap with the one at Q_{head} going North (Step 1). With intel_swap, a scan of the queue results first in the flit at location Q_{head}+3 (i.e., outport West) getting swapped with Q_{head} (Step 1), and subsequently, if West is also blocked, this gets swapped with the flit at Q_{tail} going East. intel_swap enables more number of swaps.

Packet-switching with Virtual cut-through (VCT). Packets are allowed to go to the downstream router as soon as an output channel is free. The buffer and link allocation is still done on a packet basis.

Packet-switching with Wormhole. This is similar to VCT, as the links is still allocated on a per packet basis. However, buffers at routers can be smaller than the size of the packet. Thus, buffering is done on a flit basis. Therefore, this design can work on routers with fewer buffers.

Head-of-Line Blocking. Despite obvious benefits of lesser buffering compared to other packet-switched techniques, wormhole routing suffers from Head-of-Line (HoL) blocking. Consider the case where a tail flit of certain packet is blocked at the head of the queue because the requested output channel for that flit is not free due to congestion. Even if the desired outports of the flits of the packets waiting behind the blocked flit are free, they cannot leave. This is called Head-of-Line (HoL) blocking. As this phenomenon is

unpredictable, it can lead to performance degradation.

Packet-switching with Virtual Channels (VCs). VCs reduces HOL, as Fig. A.1(b) demonstrates, by assigning separate queues for different packets. Lots of *shallow VCs* are better at heavy traffic; multiple *deep VCs* are better when there is low traffic and each packet has multiple flits. The VC queues are typically partitioned in a static manner during design time. VC-based flow control is one of the most prevalent flow control techniques in use today, and has been used across NoC prototypes [93, 94].

A.2.2 Buffer Management

In an on-chip scenario, wires are often abundant while real-estate for buffers is expensive. There has thus been a lot of prior work that tries to optimize for efficient usage of buffers - either for performance or for energy-efficiency.

Low-Cost Buffers. One class of solutions has focused on reducing the cost of VC buffers. These span from using bufferless routers [42] to intelligent bypassing of router pipeline [82, 98] to using multiple physical networks [54].

Elastic buffers [99] can also help reduce buffer area and power. These techniques are complementary to our approach and can augment it to reduce area and power further.

Dynamic VC Partitioning. The works most related to ours fall within the category of dynamic VC partitioning [96, 97, 100]. DAMQ [96] was one of the early works in this space that advocated for organizing the available buffers in the form of a linked list, allowing flits of a packet to dynamically allocate one to all of the slots at that input port. Follow-up research has tried to efficiently utilize the buffers for higher throughput [100] by emulating output buffered routers while operating the router at considerable frequencies. ViChaR [97] implements a Unified Buffer Structure (UBS) at each input port, controlled by a Unified Control Logic (UCL) structure, that dynamically maps flits to a given virtual channel to maximize buffer utilization.

In these designs, the router controls each buffer slot individually to dynamically create

shallow VCs or deep VCs. The challenge is that full flexibility requires the router to pay the *control overhead of N VCs* if there are N buffer slots at each input port to allow each buffer to act as its own VC and get direct access to the switch if required. This incurs overheads both in terms of complexity, and in terms of area and power. Our work, in contrast does the exact opposite. Rather than using individual buffer slots to dynamically construct VCs, we use a single FIFO and enable it to "act" like multiple VCs by swapping blocked packets dynamically to the head of the queue.

A.3 The SwapNoC

Wormhole routers have two key bottlenecks: (1) HoL blocking at the input port, and (2) losing arbitration for the switch output port due to flits at other input ports contending for the same output port. VCs directly address (1) since each input port has multiple candidates to choose from, not just one. VCs indirectly address (2) as well since VCs provide multiple choices to the switch allocator (SA) which can then try to find the best possible match between input requests and available output ports. However, VCs add tremendous area and power overheads as we have motivated so far. Moreover, designing a switch allocator that does a good matching is a NP-hard problem [24] and most hardware implementations use a simpler separable allocator [24] which independently arbitrates for input and output ports and cannot address (2).

We make a case for solving HoL by a much simpler solution - allow the blocked flit/-packet to perform an in-place swap with the head of the queue. We also present policies that help mitigate the output port arbitration challenge.

A.3.1 Microarchitecture

The fundamental rule in digital hardware design that is used extensively for building state machines is as follows - if multiple flip flops are connected to each other in series, at the rising edge of the clock all inputs move forward by one in parallel without clobbering the

old values[2]. For instance, in the circular shift register in Fig. A.4(b), at each clock edge, the blue and red signals can keep swapping with each other, without requiring any additional storage for performing the swap. Leveraging this principle, the microarchitecture of the SwapNoC router is shown in Fig. A.4(a). In this example, we enable the incoming flit being enqueued at the tail of the queue (Q_{tail}) to swap with the one at Q_{head}. We also allow swaps to occur from intermediate points inside the queue, as we describe later. Swaps occur at a packet granularity (i.e., if enabled, all flits of a blocked packet get swapped to Q_{head} one behind the other), as explained in Section A.3.3.

The swap control signals are setup by a swap policy controller. Different policies use different metrics to determine whether to swap or not, which are shown as inputs to the controller in Fig. A.4(a).

A.3.2 Swap Policies

In this work we present 5 policies for swapping. However, our proposed idea of swapping packets is quite powerful and there can be a lot more policies. For all the policies, we assume that the flit at Q_{head} is unable to leave due to zero credits at its outport. If not, a swap will not be triggered.

Tail_Swap. In tail_swap we swap the head of the queue with the incoming packet if the outport of the incoming packet (i.e., at Q_{tail}) is different than at Q_{head}. We trigger tail_swap if the current queue occupancy is greater than a preset *threshold* value. For this policy, we assume lookahead routing [24], i.e., the incoming packet comes with a specific output port that was computed at the previous router. If the output port of the incoming packet is different from that of the packet waiting at Q_{head}, it becomes the candidate for swapping. The swap is initiated if the current queue occupancy is greater than or equal to the threshold.

[2]Clock synthesis by CAD tools ensures this behavior for correct operation of all synchronous digital circuits.

Intel_Swap. In intel_swap we do not restrict ourselves to swap Q_{head} with the incoming (Q_{tail}) packet; instead upon reaching the threshold intel_swap proactively scans the queue from back to front to find any packet with outport different than that of its head.

On finding the first packet during scan with different outport than head, we swap it with Q_{head}.

Fig. A.4 demonstrates the tail_swap and intel_swap policies with an example. The *threshold* parameter is only used in the tail_swap and intel_swap policies, not by the other policies. The implicit difference between intel_swap policy and tail_swap policy is that swapping is done more often in intel_swap because after threshold is reached, there are more candidates (packet with different outport than Q_{head}) to choose from as compared to tail_swap.

The cost of scanning can be reduced using a per inport structure which holds the updated outport of all the packets present in the queue with their position. This structure will get updated whenever any packet enqueues, or leaves the queue or get swapped within the queue.

Credit_Swap. credit_swap is based on the insight that a flit with zero credits at its output port could get stuck for many cycles since zero credits is a likely indication of congestion at its downstream router. If such a flit were to move to the Q_{head}, it would cause HoL for other packets. The credit_swap policy is centered around finding such packets and pushing them to the tail of the queue.

credit_swap keeps track of the credit count at all the outports. Whenever any of the outport's credit becomes 0 (which means there is no buffer space available at the given inport of the downstream router), it scans the inport queue starting from front till back. During the scan, if it finds a packet with same outport as the one which has 0 credit, it swaps it with the tail of the queue. This is done at all the inports in the router, to make sure the outport which has no credit has its packets shifted towards the tail of the input queues. This also helps in reducing network congestion.

Random_Swap. random_swap policy tries to shuffle all the inport queues periodically. Shuffling is done by choosing a packet from the queue randomly and swapping it with the head of the queue. This is a heuristic, but can reduce the effect of HoL blocking as each packet comes at the head of the queue with equal probability.

Shuffle_Swap. Recall that there could be two reasons that the flit at Q_{head} is unable to leave the queue, as discussed before - HoL blocking or losing SA in the router. shuffle_swap policy also tries to shuffle all its inport queues periodically like the random_swap policy. The only difference is that when it selects the candidate packet to swap with Q_{head}, it makes sure that the selected packet does not have the same outport as the one at the head. This make head of all inport queues randomly distributed, thus not only reducing the effect of HoL blocking, but also helping increase the chance of winning the SA.

A.3.3 Multi-flit Packet Swaps

Full-packet Swap. Swaps nominally occur at a packet granularity (i.e., if enabled, all flits of a blocked packet get swapped to Q_{head} one behind the other). During a full packet swap, all flits of the packet can be swapped either serially (if there is only one bypass connection to Q_{head}) or in parallel if there are multiple connections. All flits of a packet are swapped in-order, so there is no re-ordering of flits within a packet.

Since our base design is a wormhole router, we do not allow partial swaps since the body and tail flits of a packet do not carry routing information and rely on following the flit right before it. This is because there is no "VC" to store the per-packet routing information. All links are allocated on a packet granularity. Swaps are disallowed under 2 conditions:

(1) If the flit at the Q_{head} is not a head flit of a packet, then the swap is not allowed. The reasoning is as follows: if the flit at Q_{head} is a body or tail flit, that means that the head flit of this packet already left the router. This is implemented by setting a `head_blk` whenever a head flit reaches Q_{head}, and resetting it when the tail flit leaves.

(2) If the queue does not have enough slots to hold the entire incoming packet, swaps to Q_{head} are not allowed since part of the packet would have been swapped to the front, while

the remaining would still be at the previous router waiting for credits.

Flit-level Swap. The two conditions listed above can be relaxed, i.e., partial swaps can be allowed, if each body and tail flit also carries the output port (encoded in 3-bits) at this router. The body and tail flits do not need to carry the full header (which would essentially make each body and tail flit a packet in itself reducing effective bandwidth). In such a scenario, the body and tail flits, that are no longer right behind the head flit, would know which output port to go out from when they eventually arrive at Q_{head} again.

This does not break the correctness of the design, as we describe with an example. Suppose a Packet A is waiting for the East output port and stalled. The head-flit of Packet B going towards South gets swapped and moves to the head of the queue and leaves. Now the East output port becomes free and flits from Packet A start getting sent out. Since the head of the queue is no longer blocked, the other flits of Packet B get queued behind Packet A. This is an acceptable outcome. The goal of the SwapNoC, like VCs, is to ensure that some flit can leave from an input port and output port every cycle. While Packet A was stuck, Packet B was allowed to swap and fulfill this requirement. Once Packet A becomes free, it satisfies this requirement. At the next router at the East output port, all flits of Packet A are still going to be together in the correct order.

A.3.4 Comparison to VCs.

The goal of SwapNoC is to emulate the behavior of VCs, i.e., the ability for flits to different output ports not get blocked by each other. To that end, different policies for swaps can emulate different behaviors. We give an intuition on how we can seamlessly model VC behavior without any control except the notion of a threshold and the ability for an input flit to swap with the head of the queue. **How to emulate shallow VCs?** If a network has a lot of 1-flit packets, a small value of threshold can essentially emulate the behavior of shallow VCs by allowing every new packet going to a different output port to have the ability to bypass a blocked packet.

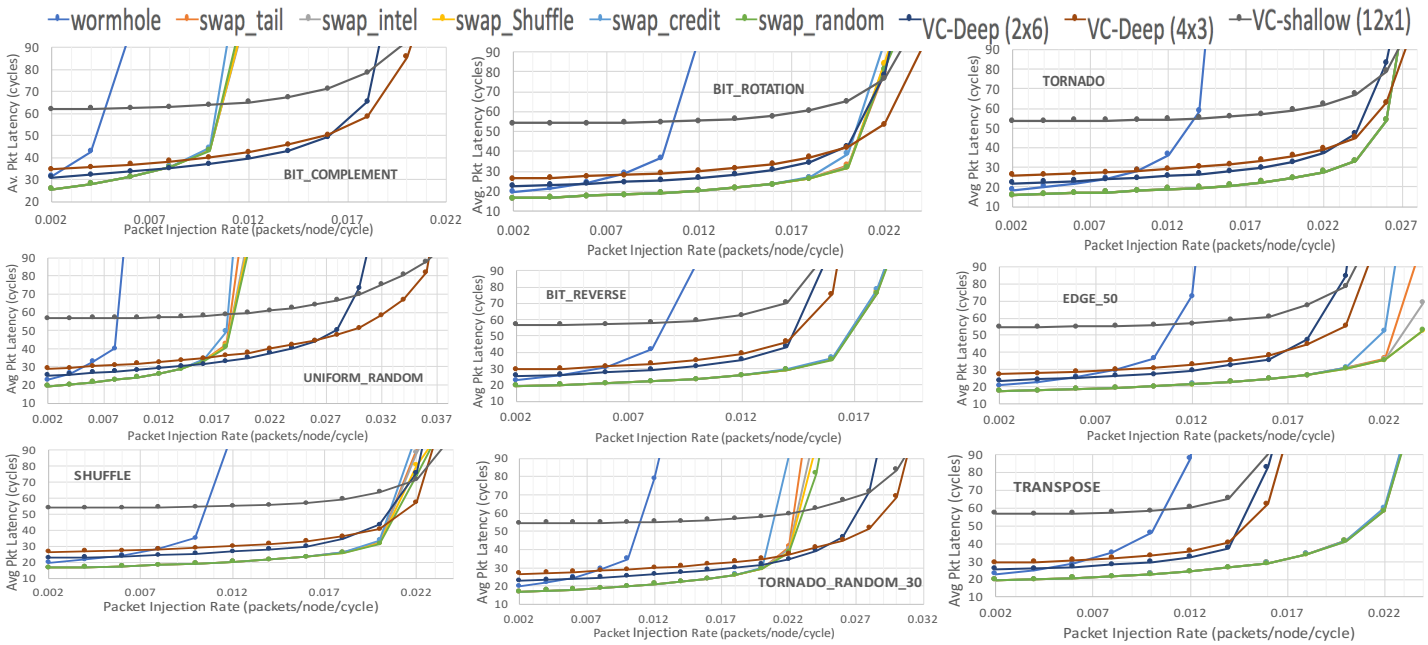
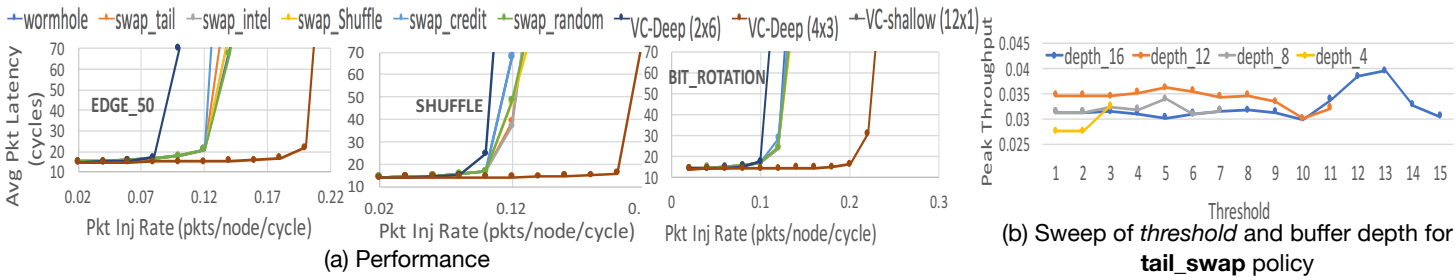


Figure A.5: Performance of SwapNoC with multi-flit packets.

How to emulate deep VCs? If a network has a lot of multi-flit packets, then having a threshold equal to or greater than the size of the packets can allow new packets to bypass blocked packets.

How is the threshold set? The *threshold* is meant to be a dynamic knob available with the router to tune the swap frequency for tail_swap and intel_swap based on traffic rate and size of packets. In our design, we support both a static version and a dynamic version. In the static version, the threshold is set in the router at reset, based on offline profiling of traffic. In the dynamic version, the router monitors the number of failed switch arbitrations and adjusts the threshold accordingly. When the number of failed arbitrations are high, the threshold is lowered, else it is raised. We demonstrate the impact of the threshold in our evaluations.

Adaptive Schemes. Schemes like credit_swap, rand_swap and shuffle_swap do not need the *threshold* knob to tune and aggressively try to adapt with traffic.



(a) Performance

(b) Sweep of *threshold* and buffer depth for **tail_swap** policy

Figure A.6: Performance of SwapNoC with single-flit packets.

Table A.1: **Network Configurations (1-cycle router in each)**

Wormhole	Wormhole router with a N-flit deep queue.
VC-shallow	N 1-flit VCs (max number of VCs).
VC-deep	2 $N/2$-flit deep and 4 $N/4$-flit-deep VCs.
SwapNoC	N-flit deep queue with Swaps.

Summary. SwapNoC cannot beat VCs cycle-by-cycle in terms of throughput, since fundamentally it operates on heuristics to get a non-blocked flit to Q_{head} while VCs can essentially arbitrate for and choose the best possible candidate every cycle. But the SwapNoC has a lower cycle time, and much lower power and area than VC routers, as we show next. Moreover, as we observe in our results, static partitioning of VCs actually performs worse than SwapNoC, especially with large packet sizes.

A.4 Evaluation

A.4.1 Methodology

Our target NoCs are described in Table A.1. We equalize the total buffers (say N) in each router across all designs. All other possible VC configurations should perform between VC-shallow and VC-deep which represent two extremes of the design space for VCs.

All routers - wormhole, Swap and VC have a state-of-the-art 1-cycle pipeline. This is an aggressive assumption for VC routers which typically take 2+ cycles due to input and output VC arbitrations [93] that are not required in wormhole and SwapNoC.

We implemented all NoCs in RTL to get pipeline delay, area, and power results post-synthesis using the 15nm Nangate FreePDK library [95]. For design-space exploration

inside multicores, we used the gem5 [7] simulator with the Garnet on-chip network model where we modeled the SwapNoC. We assume a 8×8 mesh in all our evaluations.

Traffic Patterns. We evaluate our designs across a suite of synthetic and real traffic patterns. We also define two new synthetic patterns to stress HoLs in the NoC. Tornado_Random_30 is the traditional Tornado pattern - which always sends traffic halfway across the mesh in the *same dimension* without turning, with 30% of the traffic being random, i.e., may want to turn. Edge_50 sends 50% of the traffic to the rightmost node in the same row as the source, and 50% to a random destination. The synthetic traffic runs are done with both single-flit and 5-flit packets, to demonstrate the impact of our NoCs. We also run full-system simulations with PARSEC benchmarks over a MOESI directory protocol. The protocol requires 4 virtual networks (request, response, forward, unblock) for deadlock-avoidance. Data (cacheline) packets are 5-flit, the rest of the packets are 1-flit. Wormhole and SwapNoC use a single FIFO within each vnet, while the VC-based designs use 4 VCs within each vnet.

A.4.2 Critical Path, Area and Power

RTL synthesis of the SwapNoC router demonstrates that it increases the critical path over a wormhole router by only 8.4-9.4% across queue depths from 2 to 16. This is due to the mux that can read a flit from either the head, or the threshold size depth in the queue. This overhead was well within the timing slack at 1ns, enabling a 1-cycle operation at 1GHz. The VC router, on the other hand, has a critical path close to 2ns when N=16.

Fig. A.2 shows that the SwapNoC router is 2× smaller in area and consumes 2× lower power compared to VC routers, as the number of buffer slots in each port goes up. SwapNoC adds 1% power and 8% area overhead over the wormhole router.[3]

[3]We thank Hyoukjun Kwon from Georgia Tech for help with RTL implementation and synthesis of the Swap NoC

A.4.3 Performance: Synthetic Traffic

Multi-flit Packets. Fig. A.5 evaluates the performance of SwapNoC across synthetic traffic patterns using 5-flit packets. With multi-flit packets, the shallow VC design has the highest delay, due to heavy serialization. At low loads, the flit of each packet needs to wait for the credit round trip for sending every flit of the packet. At high loads, more VCs helps push the throughput. Thus, this design has the highest throughput across most patterns. The deep VC design on the other hand provides much better low-load latency, and saturates at the same or slightly lower injection rate than the shallow VC. Wormhole saturates the earliest across all patterns, which is its key shortcoming.

The SwapNoC policies provide the best low-load latencies, providing a 61% reduction in latency compared to the shallow VC design, and 28% lower than the deep-VC design. In terms of throughput, SwapNoC provides 88.2-87.6-88.1% better throughput than the VC-based designs for bit_reverse, transpose, and edge_50. With bit_rotation, shuffle and tornado, the throughput of SwapNoC is comparable to that of the VC-based designs. For uniform_random, bit_complement, and tornado_random_30 SwapNoC provides throughput that is in between that of wormhole and VCs.

tornado is an interesting traffic pattern for the SwapNoC since traffic never turns, which would seem to imply that there would never be any HoL blocking. However, there is still HoL blocking for packets that want to get ejected. This is the reason SwapNoC actually improves throughput over the wormhole design even for tornado.
Among the SwapNoC policies, random_swap and intel_swap have slightly better performance than the others.

In summary, SwapNoC provides the performance of wormhole and deep VCs at low-loads, and close to or better throughput than shallow VCs at high loads, essentially modeling a dynamic VC partitioning design without the overheads of managing each buffer slot independently.

Single-flit Packets. Fig. A.6(a) demonstrates the performance of SwapNoC with single-

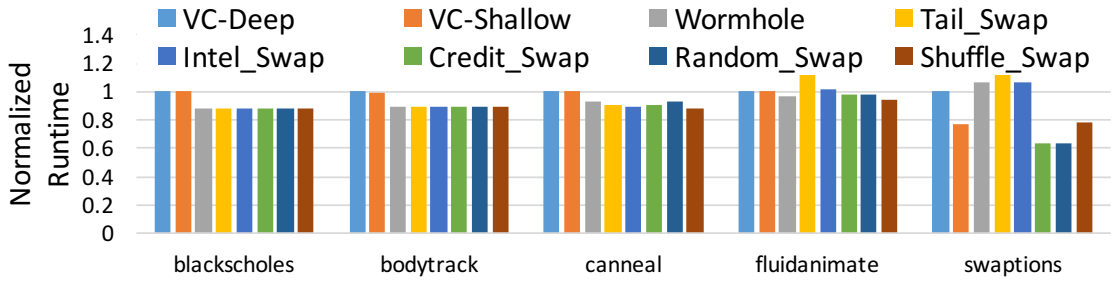


Figure A.7: Normalized Full-System Runtime with PARSEC.

flit packets. With single-flit packets, there are no credit turnaround issues for shallow VCs which provide the best latency and throughput. As discussed earlier in Section A.4.2, VCs provide much better opportunities for flits going out of unblocked outports to arbitrate for and leave, compared to the SwapNoC which relies on heuristics to come to the head of the queue. SwapNoC provides about 15% improvement in throughput over wormhole for edge_50 and shuffle, and comparable performance with bit_rotation and other patterns not shown in the interest of space. SwapNoC still beats the Deep-VC design in throughput by 40-50% in edge_50 and bit_rotation.

In summary, with single-flit packets, a deep-VC design suffers tremendously while a shallow VC design performs the best. The SwapNoC is an elegant design choice for providing better throughput than deep VC designs.

Impact of buffer depth and threshold

Fig. A.6(b) plots the throughput as a function of the *threshold* parameter across multiple buffer depths for the tail_swap policy. For a buffer depth of 4, a threshold of 3 gives a spike in performance. For depths of 8 and 12, the impact of threshold is almost negligible. But with a buffer depth of 16, a threshold of 11-13 gives the best performance.

A.4.4 Performance: Full-System PARSEC

Fig. A.7 demonstrates the full-system performance with PARSEC benchmarks. For benchmarks such as blackscholes and bodytrack, SwapNoC provides 12% lower runtime than VCs, primarily due to the faster router. Its performance is same as that of wormhole as

there is not enough NoC traffic to cause much HoL blocking. This is the same reason both deep VC and shallow VC have similar performance. With canneal and fluidanimate we see about 5% reduction in overall runtime with shuffle_swap compared to wormhole. But in some cases, tail_swap actually results in a drop in performance. swaptions shows the most dynamic behavior, demonstrating up to 36% reduction in runtime over both VCs and wormhole with credit_swap and random_swap.

In summary, for applications with low network traffic, which is often the case with real workloads, the overheads of VCs is an overkill for many-core systems. SwapNoC has similar overheads as a wormhole router, but can step in to provide higher performance than the wormhole in case of higher traffic, making it a win-win.

A.5 Conclusions

We provide a light-weight technique to mitigate HoL without requiring VCs. Our key novelty is the ability for blocked flits to swap with the head of the queue in a wormhole router, without any additional buffers to manage this swap. We add minimal control overhead to perform this swap, and also describe multiple heuristic policies for managing when swaps occur. The SwapNoC shows significant performance, energy, and area benefits over VC-based router designs. We believe that the idea of leveraging swaps goes beyond the policies presented in this paper, and can open up a suite of optimizations for NoC architects and designers.

REFERENCES

[1] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of ion-implanted mosfet's with very small physical dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, 1974.

[2] P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: A 32-way multithreaded sparc processor," *IEEE micro*, vol. 25, no. 2, pp. 21–29, 2005.

[3] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy, "Introduction to the cell multiprocessor," *IBM journal of Research and Development*, vol. 49, no. 4.5, pp. 589–604, 2005.

[4] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown, *et al.*, "Tile64-processor: A 64-core soc with mesh interconnect," in *2008 IEEE International Solid-State Circuits Conference-Digest of Technical Papers*, IEEE, 2008, pp. 88–598.

[5] T. Krishna, "Enabling dedicated single-cycle connections over a shared network-on-chip," PhD thesis, Massachusetts Institute of Technology, 2014.

[6] *The dining philosophers problem.* `https://en.wikipedia.org/wiki/Dining_philosophers_problem`.

[7] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[8] M. Parasar *et al.*, "Brownian bubble router: Enabling deadlock freedom via guaranteed forward progress," in *International Symposium on Networks on Chip*, 2018.

[9] M. Parasar and T. Krishna, "Bindu: Deadlock-freedom with one bubble in the network," in *International Symposium on Networks on Chip*, 2019.

[10] M. Parasar, H. Farrokhbakht, N. E. Jerger, P. V. Gratz, T. Krishna, and J. San Miguel, "Drain: Deadlock removal for arbitrary irregular networks," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, 2020, pp. 447–460.

[11] M. Parasar *et al.*, "Synchronized weaving of adjacent packets (swap) for network deadlock prevention," in *MICRO*, 2019.

[12] D. K. Schroder and J. A. Babcock, "Negative bias temperature instability: Road to cross in deep submicron silicon semiconductor manufacturing," *Journal of applied Physics*, vol. 94, no. 1, pp. 1–18, 2003.

[13] J. McPherson and H. Mogul, "Underlying physics of the thermochemical e model in describing low-field time-dependent dielectric breakdown in sio 2 thin films," *Journal of Applied Physics*, vol. 84, no. 3, pp. 1513–1523, 1998.

[14] E Takeda and N Suzuki, "An empirical model for device degradation due to hot-carrier injection," *IEEE electron device letters*, vol. 4, no. 4, pp. 111–113, 1983.

[15] J. R. Black, "Electromigration: A brief survey and some recent results," *IEEE Transactions on Electron Devices*, vol. 16, no. 4, pp. 338–347, 1969.

[16] I. T. R. for Semiconductors (ITRS), *More moore*, 2014.

[17] U. R. Karpuzcu *et al.*, "The bubblewrap many-core: Popping cores for sequential acceleration," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, ACM, 2009, pp. 447–458.

[18] J. Yin *et al.*, "Modular routing design for chiplet-based systems," in *Proceedings of the International Symposium on Computer Architecture*, 2018.

[19] H. Yang *et al.*, "Dodec: Random-link, low-radix on-chip networks," in *Proceedings of the International Symposium on Microarchitecture*, 2014.

[20] D. Fick *et al.*, "Vicis: A reliable network for unreliable silicon," in *Proceedings of the 46th Annual Design Automation Conference*, ACM, 2009, pp. 812–817.

[21] C. Iordanou *et al.*, "Hermes: Architecting a top-performing fault-tolerant routing algorithm for networks-on-chips," in *Computer Design (ICCD), 2014 32nd IEEE International Conference on*, IEEE, 2014, pp. 424–431.

[22] A. DeOrio *et al.*, "A reliable routing architecture and algorithm for nocs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 5, pp. 726–739, 2012.

[23] K. Bhardwaj *et al.*, "Towards graceful aging degradation in nocs through an adaptive routing algorithm," in *Proceedings of the 49th Annual Design Automation Conference*, ACM, 2012, pp. 382–391.

[24] N. E. Jerger, T. Krishna, and L.-S. Peh, *On-chip Networks*. Morgan & Claypool Publishers, 2017.

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

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

[27] M. Galles, "Scalable pipelined interconnect for distributed endpoint routing: The sgi spider chip," in *Proceedings of the International Symposium on High-Performance Interconnects (HOTI'96)*, 1996, pp. 141–146.

[28] L.-S. Peh and W. J. Dally, "A delay model and speculative architecture for pipelined routers," in *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, IEEE, 2001, pp. 255–266.

[29] A. Kumar, P. Kundu, A. P. Singh, L.-S. Peh, and N. K. Jha, "A 4.6 tbits/s 3.6 ghz single-cycle noc router with a novel switch allocator in 65nm cmos.," in *ICCD*, Citeseer, vol. 7, 2007, pp. 63–70.

[30] P. Sweazey and A. J. Smith, "A class of compatible cache consistency protocols and their support by the ieee futurebus," *ACM SIGARCH Computer Architecture News*, vol. 14, no. 2, pp. 414–423, 1986.

[31] A. GEGO, *Study and performance analysis of cache-coherence protocols in shared-memory multiprocessors.* `https://dial.uclouvain.be/memoire/ucl/en/object/thesis:6679/datastream/PDF_01/view`.

[32] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The splash-2 programs: Characterization and methodological considerations," in *Proceedings 22nd Annual International Symposium on Computer Architecture*, 1995, pp. 24–36.

[33] C. Bienia *et al.*, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, ser. International Conference on Parallel Architectures and Compilation Techniques (PACT) '08, Toronto, Ontario, Canada: ACM, 2008, pp. 72–81, ISBN: 978-1-60558-282-5.

[34] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. Comput.*, vol. 36, no. 5, pp. 547–553, May 1987.

[35] A. Singh, "Load-balanced routing in interconnection networks," PhD thesis, Stanford University, 2005.

[36] C. L. Seitz, W. C. Athas, C. M. Flaig, A. J. Martin, J. Seizovic, C. S. Steele, and W.-K. Su, "The architecture and programming of the ametek series 2010 multicom-

puter," in *Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications: Architecture, Software, Computer Systems, and General Issues - Volume 1*, ser. C3P, Pasadena, California, USA: ACM, 1988, pp. 33–37, ISBN: 0-89791-278-0.

[37] C. Carrion *et al.*, "A flow control mechanism to avoid message deadlock in k-ary n-cube networks," in *Proceedings of the Fourth International Conference on High-Performance Computing*, 1997.

[38] A. Ramrakhyani and T. Krishna, "Static bubble: A framework for deadlock-free irregular on-chip topologies," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 253–264.

[39] C. Lizhong *et al.*, "Critical bubble scheme: An efficient implementation of globally aware network flow control," in *25th IEEEIPDPS*, 2011, pp. 592–603.

[40] L. Chen and T. M. Pinkston, "Worm-bubble flow control," in *Proceedings of the 19th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2013, pp. 366–377, ISBN: 978-1-4673-5585-8.

[41] A. Ramrakhyani *et al.*, "Synchronized progress in interconnection networks (SPIN) : A new theory for deadlock freedom," in *ISCA*, 2018.

[42] T. Moscibroda and O. Mutlu, "A case for bufferless routing in on-chip networks," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ser. ISCA '09, Austin, TX, USA: ACM, 2009, pp. 196–207, ISBN: 978-1-60558-526-0.

[43] P. Baran, "On distributed communications networks," *IEEE transactions on Communications Systems*, vol. 12, no. 1, pp. 1–9, 1964.

[44] C. Fallin *et al.*, "MinBD: Minimally-buffered deflection routing for energy-efficient interconnect," in *2012 Sixth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, 2012.

[45] C. Fallin *et al.*, "Chipper: A low-complexity bufferless deflection router," in *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture*, ser. HPCA '11, Washington, DC, USA: IEEE Computer Society, 2011.

[46] D. H. Linder and J. C. Harden, "An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes," *IEEE Trans. Comput.*, vol. 40, no. 1, pp. 2–12, Jan. 1991.

[47] K. V. Anjan and T. M. Pinkston, "An efficient, fully adaptive deadlock recovery scheme: DISHA," in *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, 1995, pp. 201–210.

[48] P. Lopez *et al.*, "A very efficient distributed deadlock detection mechanism for wormhole networks," in *Proceedings of the Fourth International Symposium on High-Performance Computer Architecture*, 1998, pp. 57–66.

[49] 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.

[50] R. Wang *et al.*, "Bubble coloring: Avoiding routing- and protocol-induced deadlocks with minimal virtual channel requirement," in *ICS '13*, 2013.

[51] W. J. Dally and B. P. Towles, *Principles and practices of interconnection networks*. Elsevier, 2004.

[52] 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.

[53] S. S. Mukherjee *et al.*, "The alpha 21364 network architecture," *IEEE Micro*, vol. 22, no. 1, pp. 26–35, 2002.

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

[55] D. Ziakas, A. Baum, R. A. Maddox, and R. J. Safranek, "Intel® quickpath interconnect architectural features supporting scalable system architectures," in *2010 18th IEEE Symposium on High Performance Interconnects*, IEEE, 2010, pp. 1–6.

[56] J. Duato, "Hypertransport; technology tutorial," in *2009 IEEE Hot Chips 21 Symposium (HCS)*, IEEE, 2009, pp. 1–53.

[57] B. A. Hechtman and D. J. Sorin, "Evaluating cache coherent shared virtual memory for heterogeneous multicore chips," in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, IEEE, 2013, pp. 118–119.

[58] J. Power, A. Basu, J. Gu, S. Puthoor, B. M. Beckmann, M. D. Hill, S. K. Reinhardt, and D. A. Wood, "Heterogeneous system coherence for integrated cpu-gpu systems," in *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2013, pp. 457–467.

[59] N. Agarwal *et al.*, "GARNET: A detailed on-chip network model inside a full-system simulator," in *ISPASS*, 2009, pp. 33–42.

[60] "Garnet synthetic traffic," gem5.org.

[61] *Https://www.youtube.com/watch?v=wugwx0nc4ny*.

[62] E. Nuutila and E. Soisalon-Soininen, "On finding the strongly connected components in a directed graph," *Information Processing Letters*, vol. 49, no. 1, pp. 9–14, 1994.

[63] I. S. Duff and J. K. Reid, "An implementation of tarjan's algorithm for the block triangularization of a matrix," *ACM Transactions on Mathematical Software (TOMS)*, vol. 4, no. 2, pp. 137–147, 1978.

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

[65] T. M. Pinkston, "Flexible and efficient routing based on progressive deadlock recovery," *IEEE Transactions on Computers*, vol. 48, no. 7, pp. 649–669, 1999.

[66] H. Kwon and T. Krishna, "OpenSMART: Single-cycle multi-hop NoC generator in BSV and Chisel," in *Proc of the IEEE International Symposium on Performance Analysis of Systems and Software*, 2017.

[67] C. Jackson and S. J. Hollis, "Skip-links: A dynamically reconfiguring topology for energy-efficient nocs," in *SoC*, 2010, pp. 49–54.

[68] J. Hu and R. Marculescu, "Dyad: Smart routing for networks-on-chip," in *DAC*, 2004, pp. 260–263.

[69] A. Konstantinos *et al.*, "ARIADNE: agnostic reconfiguration in a disconnected network environment," in *PACT*, 2011.

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

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

[72] R. Parikh *et al.*, "Power-aware NoCs through routing and topology reconfiguration," in *Design Automation Conference (DAC)*, 2014.

[73] C. Sun, C.-H. O. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L.-S. Peh, and V. Stojanovic, "Dsent-a tool connecting emerging photonics with electronics for

opto-electronic networks-on-chip modeling," in *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, IEEE, 2012, pp. 201–210.

[74] J. Duato, "A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 6, no. 10, pp. 1055–1067, Oct. 1995.

[75] J. Shun and G. E. Blelloch, "Ligra: A lightweight graph processing framework for shared memory," *SIGPLAN Not.*, vol. 48, no. 8, pp. 135–146, 2013.

[76] K. A. Hawick and H. A. James, "Enumerating Circuits and Loops in Graphs with Self-Arcs and Multiple-Arcs," Massey University, Computational Science Technical Note CSTN-013, 2008.

[77] L.-S. Peh and W. J. Dally, "Flit-reservation flow control," in *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture*, 2000, pp. 73–84.

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

[79] N. Enright Jerger, L.-S. Peh, and M. Lipasti, "Circuit-switched coherence," in *International Symposium on Networks-on-Chip*, 2008.

[80] P. Wolkotte, G. Smit, G. Rauwerda, and L. Smit, "An energy efficient reconfigurable circuit-switched network-on-chip," in *International Parallel and Distributed Processing Symposium*, 2005.

[81] J. Duato, P. Lopez, F. Silla, and S. Yalamanchili, "A high-performance router architecture for interconnection networks," in *International Conference on Parallel Processing*, 1996.

[82] A. Kumar, L.-S. Peh, P. Kundu, and N. K. Jha, "Express virtual channels: Towards the ideal interconnection fabric," in *ACM SIGARCH Computer Architecture News*, ACM, vol. 35, 2007, pp. 150–161.

[83] A. Kumar, L.-S. Peh, and N. K. Jha, "Token flow control," in *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*, IEEE Computer Society, 2008, pp. 342–353.

[84] T. M. Pinkston and S. Warnakulasuriya, "On deadlocks in interconnection networks," in *Proceedings of the 24th Annual International Symposium on Computer Architecture*, 1997, pp. 38–49.

[85] W. J. Dally and H. Aoki, "Deadlock-free adaptive routing in multicomputer networks using virtual channels," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 4, pp. 466–475, Apr. 1993.

[86] C. Xiao *et al.*, "Dimensional bubble flow control and fully adaptive routing in the 2-d mesh network on chip," in *2008 IEEE/IPIP International Conference on Embedded and Ubiquitous Computing (EUC*, 2008, pp. 353–358.

[87] M. Garcia *et al.*, "On-the-fly adaptive routing in high-radix hierarchical networks," in *Proceedings of the 41st International Conference on Parallel Processing*, 2012.

[88] *Https://github.com/georgia-tech-synergy-lab/gem5_drain*.

[89] V. Nagarajan, D. J. Sorin, M. D. Hill, and D. A. Wood, "A primer on memory consistency and cache coherence," *Synthesis Lectures on Computer Architecture*, vol. 15, no. 1, pp. 1–294, 2020.

[90] M. Koibuchi *et al.*, "A case for random shortcut topologies for hpc interconnects," in *Proceedings of the International Symposium on Computer Architecture*, 2012.

[91] M. Parasar and T. Krishna, "Lightweight emulation of virtual channels using swaps," in *Proceedings of the 10th International Workshop on Network on Chip Architectures*, ser. NoCArc'17, Cambridge, MA, USA: ACM, 2017, 1:1–1:6, ISBN: 978-1-4503-5542-1.

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

[93] C. Clauss *et al.*, "Evaluation and improvements of programming models for the intel scc many-core processor," in *HPCS*, 2011.

[94] B. Daya *et al.*, "Scorpio: A 36-core research chip demonstrating snoopy coherence on a scalable mesh noc with in-network ordering," in *ISCA*, 2014.

[95] M. Martins *et al.*, "Open cell library in 15nm freepdk technology," in *ISPD*, ACM, 2015, pp. 171–178.

[96] Y. Tamir and G. L. Frazier, *High-performance multi-queue buffers for VLSI communications switches*. IEEE Computer Society Press, 1988.

[97] C. A. Nicopoulos *et al.*, "ViChaR: A dynamic virtual channel regulator for network-on-chip routers," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, 2006, pp. 333–346, ISBN: 0-7695-2732-9.

[98] A. Psarras *et al.*, "Shortpath: A network-on-chip router with fine-grained pipeline bypassing," *IEEE Transactions on Computers*, vol. 65, no. 10, pp. 3136–3147, 2016.

[99] I. Seitanidis *et al.*, "Elastistore: Flexible elastic buffering for virtual-channel-based networks on chip," *TVLSI*, vol. 23, no. 12, pp. 3015–3028, 2015.

[100] R. Ramanujam *et al.*, "Design of a high-throughput distributed shared-buffer noc router," in *NOCS*, 2010.

VITA

Mayank Parasar received B.Tech degree in Instrumentation Engineering (Electrical Engineering department) from the Indian Institute of Technology (IIT) Kharagpur, West Bengal, India, in 2013 and an M.S. degree in Electrical and Computer Engineering (ECE) from Georgia Institute of Technology in 2017. Between 2013 and 2015, he worked in the CPU architecture validation group at Nvidia, Bangalore, India as an engineer. He is currently a Ph.D. candidate in the School of Electrical and Computer Engineering at Georgia Institute of Technology, Atlanta, Georgia, working on new efficient solutions to provide routing level and protocol level deadlock freedom in interconnection networks. His interests span the area of virtual memory management and support in computer architecture, network on chip, accelerator architecture, hardware/software, and hardware/algorithm co-design. He held the position of AMD Student Ambassador at Georgia Tech in the year 2018-19. He received the Otto & Jenny Krauss Fellow award in the year 2015-16.