UNIVERSITY OF CALIFORNIA, SAN DIEGO

Throughput-Driven Design of Networks-on-Chip

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Electrical Engineering (Electronic Circuits and Systems)

by

Rohit Sunkam Ramanujam

Committee in charge:

Professor Bill Lin, Chair
Professor Chung-Kuan Cheng
Professor Sujit Dey
Professor Andrew B. Kahng
Professor Michael B. Taylor

2011

The dissertation of Rohit Sunkam Ramanujam is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____

Chair

University of California, San Diego

2011

DEDICATION

To my wonderful wife, Rashmi, who has supported me at every step of my college life.

To my family, who have waited patiently for this day to arrive.

EPIGRAPH

Discovery consists of seeing what everybody has seen
and thinking what nobody else has thought.
—Jonathan Swift

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Prof. Bill Lin, for his guidance and support. His breadth of knowledge and intuition will always inspire me. Next, I would like to thank my co-authors, Kambiz Samadi, Vassos Soteriou, Prof. Li-Shiuan Peh and Prof. Andrew Kahng for their help and contributions towards this dissertation. I would like to thank my dissertation committee members, Prof. Andrew Kahng, Prof. C.K Cheng, Prof. Sujit Dey, and Prof. Michael Taylor for their useful comments and words of advice. I would also like to thank my colleagues and friends at UCSD, specially Hao Wang, Chia-Wei Chang, Mayank Kabra, Abhijeet Bhorkar, and Jayadev Acharya for making this journey through graduate school fun and memorable. I would like to thank my parents and everyone in my family who stood behind me for all these years and I consider myself extremely lucky to have their love and support. Finally, I would like to specially thank my wife, Rashmi, for patiently proof-reading all my papers and this thesis.

Chapter 2, in part, is a reprint of the material as it appears in the following publications:

- Rohit Sunkam Ramanujam and Bill Lin, "Randomized Partially-Minimal Routing on Three-Dimensional Mesh Networks", *IEEE Computer Architecture Letters*, vol. 7, no. 2, pp. 37-40, July 2008.

- Rohit Sunkam Ramanujam and Bill Lin, "Near-Optimal Oblivious Routing on Three-Dimensional Mesh Networks", *IEEE Conference on Computer Design (ICCD)*, October 2008, pp. 134-141.

- Rohit Sunkam Ramanujam and Bill Lin, "A Layer-Multiplexed 3D On-Chip Network Architecture", *IEEE Embedded Systems Letters*, vol. 1, no. 2, August 2009, pp. 50-55.

- Rohit Sunkam Ramanujam and Bill Lin, "A Novel 3D Layer-Multiplexed On-Chip Network", *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, Princeton, NJ, October 19-20, 2009.

Chapter 2, in part, has been submitted for publication of the material as it may appear in IEEE Transactions on Very Large Scale Integration, Rohit Sunkam Ramanujam, Bill Lin, "Randomized Partially-Minimal Routing: Near-Optimal Oblivious Routing for 3D Mesh Networks". The dissertation author was the primary investigator and author of the papers.

Chapter 3, in part, is a reprint of the material as it appears in the following publications:

- Rohit Sunkam Ramanujam and Bill Lin, "Weighted Random Routing on Torus Networks", *IEEE Computer Architecture Letters*, vol. 8, no. 1, January 2009.

- Rohit Sunkam Ramanujam and Bill Lin, "Weighted Random Oblivious Routing on Torus Networks", *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, Princeton, NJ, October 19-20, 2009.

Chapter 3, in full, has been submitted for publication of material as it may appear in IEEE Transactions on Computers, Rohit Sunkam Ramanujam, Bill Lin, "Randomized Throughput-Optimal Oblivious Routing for Torus Networks". The dissertation author was the primary investigator and author of the papers.

Chapter 4, in part, is a reprint of the material as it appears in ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2010, Rohit Sunkam Ramaujam, Bill Lin, "Destination-based Adaptive Routing on 2D Mesh Networks". Chapter 4, in full, has been submitted for publication of material as it may appear in ACM Transactions in Embedded Computing Systems, Rohit Sunkam Ramanujam, Bill Lin, "Destination-Based Congestion Awareness for Adaptive Routing in 2D Mesh Networks". The dissertation author was the primary investigator and author of the papers.

Chapter 5, in full, is a reprint of the material as it appears in the following publications:

- Vassos Soteriou, Rohit Sunkam Ramanujam, Bill Lin, and Li-Shiuan Peh, "A High-Throughput Distributed Shared-Buffer NoC Router", *Computer Architecture Letters*, April, 2009.

- Rohit Sunkam Ramanujam, Vassos Soteriou, Bill Lin, and Li-Shiuan Peh, "Design of a High-Throughput Distributed Shared-Buffer NoC Router", *The 4th ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, May 3-6, 2010, Grenoble, France.

- Rohit Sunkam Ramanujam, Vassos Soteriou, Bill Lin, and Li-Shiuan Peh, "Extending the Effective Throughput of NoCs with Distributed Shared-Buffer Routers", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol.30, no.4, pp.548-561, April 2011.

The dissertation author was the primary investigator and author of the papers.

VITA

2006	Bachelor of Technology (*hons.*) in Electronics and Electrical Communications Engineering, Indian Institute of Technology, Kharagpur
2008	Master of Science in Electrical Engineering (Electronic Circuits and Systems), University of California, San Diego
2007-2011	Graduate Research Assistant, University of California, San Diego
2011	Doctor of Philosophy in Electrical Engineering (Electronic Circuits and Systems), University of California, San Diego

PUBLICATIONS

Rohit Sunkam Ramanujam and Bill Lin, "Randomized Partially-Minimal Routing on Three-Dimensional Mesh Networks", *IEEE Computer Architecture Letters*, vol. 7, no. 2, pp. 37-40, July 2008.

Rohit Sunkam Ramanujam and Bill Lin, "Near-Optimal Oblivious Routing on Three-Dimensional Mesh Networks", *IEEE Conference on Computer Design (ICCD)*, October 2008, pp. 134-141.

Rohit Sunkam Ramanujam and Bill Lin, "Weighted Random Routing on Torus Networks", *IEEE Computer Architecture Letters*, vol. 8, no. 1, January 2009.

Vassos Soteriou, Rohit Sunkam Ramanujam, Bill Lin, and Li Shiuan Peh, "A High-Throughput Distributed Shared-Buffer NoC Router", *Computer Architecture Letters*, April, 2009.

Rohit Sunkam Ramanujam and Bill Lin, "A Layer-Multiplexed 3D On-Chip Network Architecture", *IEEE Embedded Systems Letters*, vol. 1, no. 2, August 2009, pp. 50-55.

Rohit Sunkam Ramanujam and Bill Lin, "Weighted Random Oblivious Routing on Torus Networks", *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, Princeton, NJ, October 19-20, 2009.

Rohit Sunkam Ramanujam and Bill Lin, "A Novel 3D Layer-Multiplexed On-Chip Network", *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, Princeton, NJ, October 19-20, 2009.

Rohit Sunkam Ramanujam, Vassos Soteriou, Bill Lin, and Li-Shiuan Peh, "Design of a High-Throughput Distributed Shared-Buffer NoC Router", *The 4th ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, May 3-6, 2010, Grenoble, France.

Andrew Kahng, Bill Lin, Kambiz Samadi, and Rohit Sunkam Ramanujam, "Trace-Driven Optimization of Network-on-Chip Configurations", *ACM/IEEE Design Automation Conference*, Anaheim, CA, June 2010.

Rohit Sunkam Ramaujam and Bill Lin, "Destination-based Adaptive Routing on 2D Mesh Networks", *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, Princeton, NJ, October 25-26, 2010.

Andrew Kahng, Bill Lin, Kambiz Samadi, and Rohit Sunkam Ramanujam, "Efficient Trace-Driven Metaheuristics for Optimization of Networks-on-Chip Configurations", *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, San Jose, CA, November 2010.

Rohit Sunkam Ramanujam, Vassos Soteriou, Bill Lin, and Li-Shiuan Peh, "Extending the Effective Throughput of NoCs with Distributed Shared-Buffer Routers", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol.30, no.4, pp.548-561, April 2011.

Rohit Sunkam Ramanujam and Bill Lin, "Randomized Partially-Minimal Routing: Near-Optimal Oblivious Routing for 3D Mesh Networks", *IEEE Transactions on Very Large Scale Integration*, in submission.

Rohit Sunkam Ramanujam and Bill Lin, "Randomized Throughput-Optimal Oblivious Routing for Torus Networks", *IEEE Transactions on Computers*, in submission.

Rohit Sunkam Ramanujam and Bill Lin, "Destination-Based Congestion Awareness for Adaptive Routing in 2D Mesh Networks", *ACM Transactions in Embedded Computing Systems*, in submission.

ABSTRACT OF THE DISSERTATION

Throughput-Driven Design of Networks-on-Chip

by

Rohit Sunkam Ramanujam

Doctor of Philosophy in Electrical Engineering (Electronic Circuits and Systems)

University of California, San Diego, 2011

Professor Bill Lin, Chair

With the increasing number of processing cores in many-core chip-multiprocessors (CMPs) and with the ever-growing number of IPs being integrated in multiprocessor systems-on-chips (MPSoCs), the need for a scalable high-bandwidth communication fabric becomes more evident. Networks-on-Chip (NoCs) have successfully catered to these needs and are fast emerging as the de-facto communication fabric in both the CMP and MPSoC domains. Both throughput and latency are important performance metrics in the design of NoCs. This thesis focuses on a throughput-driven NoC design paradigm where maximizing NoC throughput is the primary design objective. NoCs that can sustain high throughput are necessary to handle high on-chip traffic volumes expected in current and future many-core processors with a large number of processing cores and potentially running a larger number of concurrent threads.

For bandwidth-hungry applications, an increase in the sustainable throughput often translates into significant reductions in communication latency during temporary traffic bursts when the network is driven close to saturation.

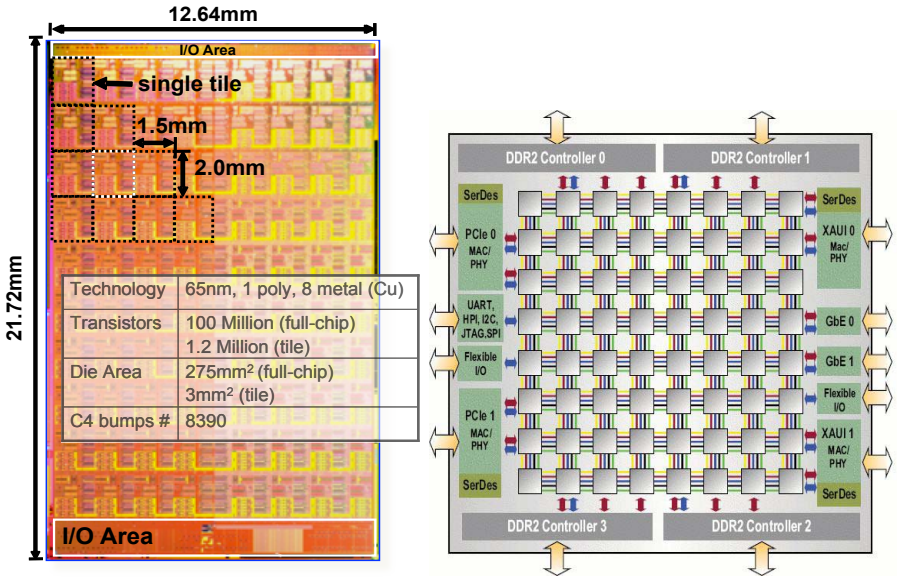
The problem of maximizing throughput in NoCs is tackled using two different approaches. First, novel oblivious and adaptive routing algorithms are proposed for mesh and torus topologies that maximize throughput by load-balancing traffic over all network links. The proposed oblivious routing algorithms guarantee optimal worst-case throughput, which is the throughput sustained under the most adversarial traffic. Providing such guarantees is important for general-purpose CMPs where target applications and resulting traffic patterns are not known at design time. Compared to existing solutions that achieve optimal worst-case throughput, the proposed algorithms achieve significantly lower latency and higher average-case throughput. The adaptive routing algorithm proposed for mesh networks improves the accuracy of congestion estimation over prior solutions by maintaining fine-grained destination-based congestion estimates that provide greater visibility into the global congestion state of the network. This results in better routing decisions and more efficient load-balancing. Next, a new router architecture is proposed that extends NoC throughput by more efficiently multiplexing packets onto network links. It deviates from the input-buffered router architecture traditionally used in NoCs. Instead, it practically emulates an output-buffered router that is known to achieve higher throughput.

Chapter 1

Introduction

1.1 Networks-on-Chip

Diminishing returns in the performance of uniprocessor architectures have led to the advent of multi-core chips. With the increasing number of on-chip cores, the need for a scalable and high-bandwidth communication fabric becomes more evident [4, 11]. Networks-on-chips (NoCs) have successfully catered to these needs and are fast



(a) Intel Teraflops (b) Tilera TILE64

Figure 1.1: Examples of multi-core chips with on-chip networks.

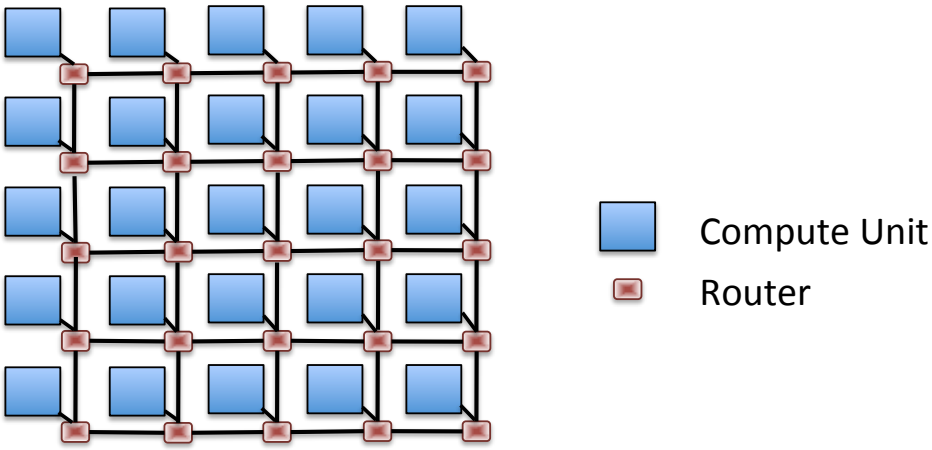


Figure 1.2: Network-on-Chip.

emerging as the de facto interconnection fabric for both general-purpose chip multiprocessors (CMPs) and application-specific systems-on-chips (SoCs). Figures 1.1(a) and 1.1(b) are examples of a 80-core research prototype from Intel [67] and a commercial 64-core chip for embedded applications from Tilera [3] that employ an on-chip network for inter-tile communication.

Packet-switched NoCs generally have a router connected to every node, which in turn are connected to neighboring routers via local on-chip wiring, as shown in Figure 1.2. Nodes in the network communicate with each other by transmitting and receiving packets to and from the local router. The routers multiplex several communicating flows onto a single physical network, making efficient use of the wires and hence, providing a scalable and high-bandwidth communication fabric.

Power and area are critical constraints while designing on-chip communication architectures. The distributed nature of the switches (routers) in a NoC sets it apart from centralized communication fabrics like shared-buses and crossbars whose feasibility greatly reduce with increasing core counts. Shared-buses suffer from reduction in throughput (per node) and increasing complexity of the centralized arbiter as the number of nodes increase. Similarly, the power and area of traditional crossbars increase quadratically with the number of communicating elements. These scalability issues with buses and crossbars have led to the broad adoption of NoCs for current and future many-core CMPs and MPSoCs [3, 23, 27, 63, 67]. Along with the scalability advantage,

NoCs are also built out of modular and repetitive structures, which enables extensive design reuse and greatly eases the burden of verification.

1.2 Motivation for throughput-driven NoC design

This thesis focuses on delivering high throughput in NoCs. In this section, we discuss the motivation behind solving this problem. Multi- and many-core processors will be commonplace in a variety of computing domains. Some potential applications include high performance graphics, high-performance computing applications like scientific simulations and modeling, server consolidation in data-centers and high throughput packet processing in multi-core network processors. Tera-scale CMP architectures consisting of several tens of physical cores and hundreds of hardware threads are highly suitable for high-performance throughput computing. Modern server workloads exhibit a high degree of thread-level parallelism which can be efficiently exploited by leveraging Moore's law to integrate a larger number of simpler cores within a chip rather than a few complex cores that only target single-thread performance. Towards this end, Intel recently created an experimental "Single-chip Cloud Computer," (SCC) [23] a research microprocessor containing 48 IA32 cores. It incorporates technologies intended to scale multi-core processors to 100 cores and beyond, and a 2D mesh-based on-chip network is an integral part of the system. With more threads running concurrently, there will be a corresponding increase in demand for both on-chip and off-chip memory bandwidth as well as increased competition for other shared resources within a chip. Hence, in order to keep the threads busy, a high bandwidth on-chip network connecting the processing cores to shared on-chip resources like caches and memory controllers is crucial.

Along with general-purpose CMPs, throughput is also critical for GPUs since achieving high computational throughput is the top priority in most graphics applications. GPUs are optimized for extremely parallel workloads and unlike current CMPs, GPUs like Nvidia's Tesla [42] and Fermi processors [19] already integrate hundreds of simple processing elements with thousands of parallel threads running concurrently to exploit data-level parallelism. Such highly parallel systems again

require a high-bandwidth on-chip interconnection network to access shared resources like caches and memory controllers within the chip. With the emergence of parallel programming languages that leverage the computational power of GPUs for general-purpose computing, the importance of co-operation between threads via inter-thread communication is also gradually increasing. This is likely to add further bandwidth strains on the interconnection network.

High-throughput NoCs are also gaining traction in the networking domain [22, 70] with the emergence of multi-core carrier-class network processors that take advantage of inherent packet-level parallelism in networking applications for feature-rich packet processing at throughputs of 100Gb/s or higher. Like server and GPU workloads, network processors also rely on extensive thread-level parallelism to attain high throughputs by spawning a new thread for processing a packet. These multi-core processors also need a high-bandwidth interconnection network to connect multiple RISC processors to other on-chip resources like embedded memory, various hardware accelerators, off-chip DRAM controllers and high-speed I/O interfaces.

All these application domains call for a throughput-oriented approach to NoC design to handle the high on-chip traffic volumes expected in current and future applications. In addition, for such bandwidth-hungry applications, throughput and latency often go hand in hand. Since NoCs in CMPs connect general-purpose computing elements, the exact application and the resulting network traffic are not known at design time. To add to the unpredictability, network traffic is often very bursty in nature. Due to the unpredictable nature of on-chip traffic, during the course of an application running on a CMP, there may be temporary traffic bursts that may drive the network close to saturation. Network saturation is characterized by a steep rise in communication latency as packet backlogs deplete network buffers at a fast pace and long queueing delays are incurred at intermediate routers. In many cases, local congestion hot-spots can rapidly overwhelm large portions of the network, an effect also known as tree saturation [12]. Hence, increasing the throughput at which the network saturates can translate into significant savings in communication latency, especially during periods of congestion. High throughput NoCs, therefore, can greatly boost application performance both by sustaining higher volumes of on-chip traffic and lowering communication delays during

periods of congestion.

Parallel computing has been around for a while in the form of shared-memory multiprocessors, multi-chassis supercomputers and clusters [24, 51]. Due to the high delay overheads associated with off-chip communication as well as packaging constraints like pin limitations, the processing elements in these parallel infrastructures are more loosely coupled compared to tightly-coupled processors integrated on the same die. In contrast to off-chip communication, on-chip communication is cheap since wires are plentiful within a chip and on-chip latencies are an order of magnitude lower than off-chip latencies. With significantly lower communication cost, many-core processors are capable of exploiting parallelism at a much finer granularity through fine-grained interaction between cores. Hence, the many-core wave is expected to spawn current and future applications that leverage the processing power of many tightly-coupled cores, and consequently demand high bandwidth from the on-chip communication fabric.

Although constraints on the number of wires and I/O pins are considerably eased in on-chip networks compared to their off-chip counterparts, several new constraints are also introduced that make NoC design starkly different from designing off-chip networks. Since the routers and wires share the power and area budgets with other components within the chip, NoCs have to be designed under strict power and area constraints. Along with high-throughput, fine-grained interaction between cores also requires low-latency communication. For example, in a multi-threaded workload that requires synchronization between threads, communication latency directly impacts application performance. Hence, minimizing the power, area, and latency overheads are important constraints while maximizing throughput in the throughput-driven NoC design paradigm.

1.3 Techniques for improving throughput in NoCs

There are several design knobs that can be tuned to improve throughput in NoCs.

1. **Network topology:** The network topology determines how the nodes are physically laid out and connected to each other through network links.[1] The

[1] Network links and channels are used interchangeably throughout the thesis.

topology characterizes important properties like the bisection bandwidth of the network, which plays an important role in determining overall throughput. Bisection bandwidth is essentially the minimum number of wires that are cut if the network is divided into two halves. A higher bisection bandwidth means higher throughput under uniform traffic. The choice of the best topology for general-purpose CMPs is a difficult one since the applications running on them and the resulting traffic patterns are not known at design time. Due to a number of factors like wire length, ease of physical layout, router complexity etc., general-purpose network topologies like meshes [3, 23, 63, 67] and rings [27, 52] have emerged as popular choices for on-chip networks. In this thesis, we assume a given topology and focus on exploiting the throughput maximization techniques that are discussed next.

2. **Routing algorithm:** Given a network topology, the routing algorithm determines the paths that packets take from the source node to the destination node. The routing algorithm's ability to balance network load even under non-uniform traffic patterns plays a key role in determining throughput. The overall throughput that can be sustained by the network is generally constrained by the maximum load on a network channel. A routing algorithm that does a poor job of load-balancing traffic may cause some channels in the network to saturate at very low packet injection rates. Once certain channels saturate, long packet backlogs are created that consume significant network resources (like buffers) before finally back-pressuring the source nodes and further throttling the injection rate. Hence, the problem of maximizing throughput is equivalent to minimizing the maximum channel load on any network channel. This objective is best realized by a routing algorithm that balances network load uniformly over all channels.

   The goal of load-balancing, however, is often at odds with minimizing the length of the routing paths, which is necessary to ensure low latency. Ideally, we would like a routing algorithm to just use minimal routing paths[2] and maximize

[2]For regular network topologies like rings, meshes and tori, a path is said to be minimal if its length in terms of the number of intermediate router hops is equal to the manhattan distance between the source and destination.

both worst-case and average-case throughputs. Here, worst-case throughput represents the throughput that can be sustained under the most adversarial traffic while average-case throughput signifies the average throughput over a large set of random traffic patterns. Attaining optimal throughput and minimum delay may not be always possible with *oblivious routing* where the routing paths are *oblivious* to the network congestion state [46, 53, 59, 61]. When optimal throughput cannot be attained with minimal routing, non-minimal routing paths need to be used to trade off latency for throughput. An alternate approach to designing routing algorithms is to adapt the routing paths to the current traffic conditions by routing around heavily congested links. Such routing algorithms are categorized as *adaptive routing* and employ more complex control hardware to sense and react to network congestion [14,20,60]. The performance of an adaptive routing algorithm depends to a great extent on the accuracy of its congestion estimates. In this thesis, we explore new oblivious and adaptive routing algorithms as means of improving network throughput.

3. **Router architecture:** A router's role lies in efficiently multiplexing packets onto the output ports/links determined by the routing algorithm. Buffers are generally used in routers to house incoming flits[3] that cannot be immediately forwarded due to contention. An ideal router is expected to behave like an output-buffered router where packets are queued at output ports and network links are always utilized whenever there are flits available to use the links. However, implementing output-buffering may not be practical for on-chip networks as it requires a switch speedup equal to the number of router ports or a large $P \times P^2$ crossbar, where P is the number of router ports. Both these techniques have prohibitively high power/area costs, which is the primary reason why low-cost input-buffered routers [21, 45, 67] are the preferred choice in NoCs. Input-buffered routers, however, cannot achieve the high levels of link utilization that can be attained using ideal output-buffered routers due to inefficiencies along their datapath introduced by hardware constraints. These inefficiencies stem from the fact that sophistication of matching algorithms that can be implemented in hardware to match input and output ports

[3] A flit is a fixed-size unit of a packetized message.

is quite limited under the stringent power and area budgets of on-chip routers. Therefore, the router architecture plays a central role in determining the efficiency of routers, which is a measure of how efficiently links in the network are utilized. The closer a router comes to emulating the behavior of an ideal output-buffered router, higher is the throughput it can sustain.

In addition to affecting throughput, a router is also the primary source of communication latency in NoCs, which rely on hop-by-hop transmission of packets. To meet the aggressive clocking needs of NoCs, routers are pipelined into one or more stages. A lot of prior research has focused on reducing the number of pipeline stages in input-buffered routers to minimize latency [21, 36, 45, 48]. Hence, any effort to improve throughput by attacking the bottlenecks along the datapath of input-buffered routers must also ensure that the number of pipeline stages and the complexity of each stage is not significantly increased during the process.

4. **Flow control:** Flow control determines how resources (buffers) are allocated to messages as they travel through the network. Unlike internet routers where packets can be dropped when buffers are full, on-chip networks cannot tolerate dropping of packets. This is because latency is important in most NoC applications and there may not exist a message retransmission protocol to recover from packet losses. To keep the buffering low, buffers are commonly allocated at the granularity of flits, an alternate name for a flow control unit. Credit-based flow control or On/Off flow control is generally used to keep track of buffering available in downstream routers and flits are forwarded from the current router only when free buffering is available downstream. The physical buffers are divided into virtual channels (VCs) to provide some degree of buffer isolation between network flows. Virtual channels improve network throughput and reduce packet latency by allowing unblocked packets to bypass blocked packets in the network. VCs provide an illusion of multiple virtual networks that are multiplexed onto the same physical network.

   Some schemes that leverage flow-control to improve throughput include Flit-Reservation Flow-Control [49], Express Virtual Channels [38] and Vichar [8].

Flit-reservation Flow-Control sends control flits ahead of data flits and timestamps these control flits so that buffers are allocated just-in-time when data flits arrive. Express Virtual Channels are essentially multi-hop VCs that allow packets to bypass the router pipeline stages at intermediate hops. Vichar extends throughput in NoCs by supporting a variable number of virtual channels, so that the router can use as many VCs as the current number of flows to maximize flow-multiplexing. The work presented in this thesis focuses on using routing algorithms and router architecture as primary means of improving throughput. Credit-based virtual channel flow control is assumed at the routers for all our evaluations. Changes to the router architecture may mandate some additional flow control between router pipeline stages, which is explored as a part of the router architecture design. Most of the flow-control related research is orthogonal to the thesis contributions and it is conceivable that many innovations in the flow-control domain can be used in conjunction with the ideas proposed in this thesis to further improve throughput.

1.4 Problem statement and contributions

The problem solved in this dissertation can be formally stated as follows:

How can we design NoC routers and routing algorithms that deliver high throughput while ensuring low latency, low power and low area overheads?

The above problem is solved using two of the four different approaches discussed in Section 1.3. First, efficient oblivious and adaptive routing algorithms are developed that load-balance traffic uniformly over all network links. In particular, oblivious routing algorithms for rings (1D torus), 2D torus, and 3D mesh networks are proposed that are optimal in terms of worst-case throughput. In addition to their throughput optimality, the proposed routing algorithms have the lowest latency (measured as the average number of hops between any pair of network nodes) among all known oblivious routing algorithms for the respective topologies that are also worst-case throughput optimal. As an alternative to oblivious routing, a minimal adaptive routing algorithm for 2D mesh

networks is also proposed, which uses fine-grained destination-based global congestion estimates to accurately sense and react to network congestion. Second, in the domain of router architecture design, a novel distributed-shared-buffer (DSB) router architecture is proposed that deviates from the traditional input-buffered router architecture broadly adopted by the NoC community and instead, practically emulates output buffering. In doing so, the DSB router inherits the high-throughput and predictable delay properties of output-buffered routers.

The main contributions of the dissertation are as follows:

- **Worst-Case Throughput Optimal Oblivious Routing on 3D Mesh Networks:** A new oblivious routing algorithm for three (and higher) dimensional mesh networks called Randomized Partially-Minimal routing (RPM) is proposed, which provably achieves optimal worst-case throughput for 3D meshes when the network radix k is even and within a factor of $1/k^2$ of optimal worst-case throughput when k is odd. On average latency, as measured in network hops, RPM does not achieve minimal-length routing because non-minimal routing is used in one of three dimensions of the 3D mesh network. However, whereas the best previously known worst-case throughput optimal routing for 3D meshes, Valiant load-balancing [66], achieves optimal worst-case throughput at the cost of twice-minimal latency, RPM achieves (near) optimal worst-case throughput with much lower latency of 1.33 times minimal for symmetric mesh topologies. In practice, the average latency of RPM is expected to be closer to minimal routing because 3D mesh networks are not expected to be symmetric in 3D chip designs. For practical asymmetric 3D mesh configurations where the number of device layers is far fewer than the number of nodes along the edge of a layer, the average latency of RPM reduces to just a factor of 1.11 times minimal. A variant of RPM, referred to as Randomized Minimal First (RMF) routing, is proposed to further reduce the latency overhead of RPM over minimal routing under traffic patterns that are inherently load-balanced. A novel layer-multiplexed (LM) architecture for 3D on-chip networks is also presented that exploits the optimality of RPM together with the short inter-layer wiring delays enabled in 3D technology by replacing the one-layer-per-hop routing in a 3D mesh with simpler vertical demultiplexing

and multiplexing structures. The LM architecture has several advantages over a traditional 3D mesh with regards to latency, power and area. The details of this work are discussed in Chapter 2.

- **Worst-Case Throughput Optimal Oblivious Routing on Torus Networks:** A new weighted random oblivious routing algorithm for one-dimensional rings (1D torus) referred to as Weighted Random Direction (WRD) is proposed that offers both optimal worst-case throughput and the minimum average hop count achievable while remaining worst-case throughput optimal for ring networks. A new closed-form oblivious routing algorithm based on a weighted random selection of paths with at most two turns is proposed for 2D torus topologies. This algorithm, referred to as W2TURN, achieves optimal worst-case throughput for 2D tori and has a simple deadlock-free implementation because of its 2-turn paths. In comparison to IVAL [65], the best previously known worst-case throughput optimal routing algorithm for 2D tori with a closed-form description, W2TURN achieves lower average hop count and higher average-case throughput. Both WRD and W2TURN are discussed in Chapter 3.

- **Destination-Based Minimal Adaptive Routing on Mesh Networks:** A destination-based adaptive routing algorithm (DAR) for 2D mesh networks is proposed. The performance of an adaptive routing algorithm is determined by its ability to accurately estimate congestion in the network. In this regard, maintaining global congestion state using a separate monitoring network offers better congestion visibility into distant parts of the network compared to solutions relying only on local congestion. DAR is a minimal destination-based adaptive routing strategy (DAR), where every node estimates the delay to every other node in the network, and routing decisions are based on these fine-grained per-destination delay estimates. DAR outperforms Regional Congestion Awareness [20], the best previously known adaptive routing algorithm for 2D meshes that uses non-local congestion state, both in terms of throughput and latency. DAR also significantly outperforms adaptive routing schemes based on local congestion estimates. A scalable version of DAR, referred to as SDAR, is also proposed to minimize the overheads associated with DAR in large network

topologies. The design, implementation, and evaluation of DAR and SDAR is described in Chapter 4.

- **Emulating Output-Buffering using a Distributed Shared-Buffer Router Architecture:** Finally, a new on-chip router design based on a Distributed Shared-Buffer (DSB) architecture [25, 50] is proposed that aims to emulate an ideal output-buffered router. The DSB router comes close to matching the ideal achievable throughput of output-buffered routers without any router speedup that is needed to naively implement output-buffering. DSB routers significantly outperform state-of -the-art input buffered routers in terms of the maximum throughput they can sustain. The proposed architecture introduces innovations to address the unique constraints of NoCs, including efficient pipelining and novel flow control. The work also explores practical DSB configurations with the objective of reducing the power overhead while ensuring negligible performance degradation. The DSB router architecture is described in Chapter 5.

Chapter 2

Near-Optimal Oblivious Routing for 3D Mesh Networks

2.1 Introduction

There has been considerable discussion in recent years on the benefits of three dimensional (3D) silicon integration in which multiple device layers are stacked on top of each other with direct vertical interconnects tunneling through them [6, 15, 29, 40, 72]. 3D integration promises to address many of the key challenges that arise from the semiconductor industry's relentless push into the deep nano-scale regime. Recent advances in 3D technology in the area of heat dissipation and micro-cooling mechanisms have alleviated thermal viability and reliability concerns regarding stacked device layers. Among the benefits, 3D integration promises the ability to provide huge amounts of communication bandwidth between device layers and integrate disparate technologies in the same chip.

The increasing viability of 3D technology has opened new opportunities for chip architecture innovations. One direction is in the extension of two-dimensional (2D) tiled chip-multiprocessor architectures [3, 21, 63, 67] into three dimensions [17, 31]. Many proposed 2D tiled chip-multiprocessor architectures have relied on a 2D mesh network topology as the underlying communication fabric. Extending mesh-based tiled chip-multiprocessor architectures into three dimensions represents a natural progression for

exploiting 3D integration. The focus of this chapter is on providing efficient routing for such 3D mesh networks.

As in the case of 2D mesh networks, throughput and latency are important performance metrics in the design of routing algorithms. Ideally, a routing algorithm should maximize both the worst-case and average-case throughput and minimize the length of routing paths. Although dimension-ordered routing (DOR) algorithm [59] achieves minimal-length routing, it suffers from poor worst-case and average-case throughput because it offers no route diversity. On the other hand, the routing algorithm proposed by Valiant (VAL) [66] achieves optimal worst-case throughput by load balancing globally across the entire network. However, it suffers from poor average-case throughput and long routing paths. ROMM [46] provides another alternative that achieves minimal routing and good average-case throughput by considering route diversity in the minimal direction, but it suffers from poor worst-case throughput.

For the case of 2D mesh networks, Seo et al. [53] described a novel routing algorithm called O1TURN that achieves both minimal-length routing and near-optimal worst-case throughput. O1TURN simply chooses between two possible minimal-turn paths (XY and YX) for routing. Despite the simplicity, it was shown that O1TURN achieves optimal worst-case throughput when the network radix k is even and within a factor of $1/k^2$ of optimal worst-case throughput when k is odd. However, as observed in [53], the near-optimal worst-case throughput property of O1TURN does not extend to higher dimensions[1]. Perhaps surprisingly, the worst-case throughput of O1TURN degrades tremendously for higher dimensional meshes. For example, in the 3D case for an $8 \times 8 \times 8$ mesh, the worst-case throughput of O1TURN degrades to just 30% of optimal. The corresponding worst-case throughput values for DOR and ROMM are even less at around 13% and 26% of optimal, respectively.

In this chapter, we introduce a new oblivious routing algorithm called *Randomized Partially-Minimal* (RPM) routing that achieves near-optimal worst-case throughput, higher average-case throughput than existing routing algorithms, and good average latency. Conceptually, RPM works as follows: In the 3D case, we use Z to denote the "vertical" dimension and XY to denote the two "horizontal" dimensions. RPM works

[1] Although technically the 3D version of O1TURN is called "O2TURN", we will simply refer to the algorithm as O1TURN so that the same name can be applied to all higher dimensional meshes.

by first routing a packet in the minimal direction to a random intermediate “layer" or “plane" in the vertical dimension; i.e., it first routes a packet in the minimal direction to a random intermediate Z position. It then routes the packet on the XY layer using either minimal XY or YX routing with equal probability. Finally, it routes the packet in the minimal direction in the Z vertical dimension to its final destination. The entire Z-XY-Z or Z-YX-Z path makes at most three turns. Effectively, RPM load-balances traffic uniformly across all k vertical layers and routes traffic minimally in the two horizontal dimensions[2].

Although RPM is worst-case throughput optimal, it is not minimal in terms of latency since it routes packets non-minimally in one of the three dimensions. For certain traffic patterns that are inherently load-balanced, the latency of RPM can be further reduced by preferentially choosing an intermediate routing layer in the minimal direction, while still ensuring uniform load-balancing across all layers. We propose a variant of RPM called *Randomized Minimal First* (RMF) routing that uses destination-aware intermediate layer selection to preferentially select intermediate layers in the minimal direction to reduce latency.

Finally, we propose a novel layer-multiplexed (LM) architecture for 3D on-chip networks that exploits the optimality of RPM together with the short inter-layer wiring delays enabled in 3D technology by replacing the one-layer-per-hop routing in a 3D mesh with simpler vertical demultiplexing and multiplexing structures. The adaptation of RPM routing to the LM architecture, referred to as RPM-LM, can achieve the same worst-case throughput as RPM on a 3D mesh while reducing latency, power and area.

The main contributions of the chapter are as follows:

- We propose a new routing algorithm for three (and higher) dimensional mesh networks called RPM which provably achieves optimal worst-case throughput for 3D meshes when the network radix k is even and within a factor of $1/k^2$ of optimal worst-case throughput when k is odd.

- RPM also significantly outperforms DOR, ROMM, O1TURN and VAL in average-case throughput by 90-109%, 45-54%, 28-35%, and 24-52%, respec-

[2]RPM can be equivalently defined by randomizing on any one dimension and minimally routing on the remaining two dimensions.

tively, on different symmetric and asymmetric mesh topologies that were evaluated.

- On average latency, as measured in network hops, RPM does not achieve minimal-length routing because non-minimal routing is used in one of three dimensions. However, whereas VAL achieves optimal worst-case throughput at the cost of twice-minimal latency, RPM achieves (near) optimal worst-case throughput with much lower latency of 1.33 times minimal for symmetric mesh topologies. In practice, the average latency of RPM is expected to be closer to minimal routing because 3D mesh networks are not expected to be symmetric in 3D chip designs. In particular, the number of available device layers is expected to be fewer than the number of processor tiles that can be placed along an edge of a device layer. For example, for a four layered $16 \times 16 \times 4$ mesh[3], the average latency of RPM reduces to just a factor of 1.11 times minimal.

- We propose a variant of RPM called Randomized Minimal First (RMF) routing, which uses the knowledge of a packet's destination while load balancing traffic across vertical layers and intelligently prefers minimal layers over non-minimal layers. RMF leverages the inherent load-balancing properties of the network traffic to reduce packet latency. On uniform traffic, the latency of RMF is only 0.4-2.6% higher than DOR. RMF also retains the worst-case throughput optimality property of RPM and performs at least as well or better than RPM on a wide range of traffic patterns.

- Finally, we propose a layer-multiplexed architecture for 3D ICs that takes advantage of the short inter-layer distances and abundance of vertical wiring in 3D ICs to replace the one-layer-per-hop routing in the vertical dimension with simpler demultiplexing and multiplexing structures. The LM architecture enables an efficient implementation of RPM with lower average latency and also reduces router power and area cost compared to a conventional 3D mesh.

[3] A $16 \times 16 \times 4$ 3D tiled chip-multiprocessor design is expected to be viable in the future. Already, a single-layer 16×16 multi-core design is in commercial use today in highend carrier class routers [70]. It is a 188-core design with some tiles used for dedicated functions.

The rest of the chapter is organized as follows: Section 2.2 provides a brief background on performance metrics. Section 2.3 describes the RPM routing algorithm and presents analytical results. Section 2.4 discusses how RPM can be efficiently integrated into a typical on-chip router. Section 2.5 describes the RMF routing algorithm. Section 2.6 presents the LM architecture and adaptation of RPM routing to the LM architecture, referred to as RPM-LM. Section 2.7 evaluates the performance of RPM, RMF, and RPM-LM and also estimates the power and area overheads for implementing them in on-chip routers. Section 2.8 concludes the chapter.

2.2 Background

In this section, we provide a brief overview of the analytical methods used to evaluate worst-case and average-case throughput. In particular, we elaborate on the concept of network capacity and a method to compute worst-case throughput for oblivious routing algorithms. We then elaborate on a method to compute average-case throughput. To simplify the discussion on throughput analysis, we ignore flow control issues and assume single flit packets that can be routed from a node to its adjacent neighbor in a single cycle.

Network capacity is defined by the maximum channel load γ^* that a channel at the bisection of the network needs to sustain under uniformly distributed traffic. Consider a network bisection that divides a $k \times k \times k$ network into two parts: one with $k^2\lfloor k/2 \rfloor$ nodes and the other with $k^2\lceil k/2 \rceil$ nodes. There are a total of k^2 channels that cross each direction of the bisection. To compute the load on the bisection channels, we equate the number of flits generated under uniform traffic that would cross the bisection with the number of flits that the bisection can sustain assuming all bisection channels are equally loaded.

$$\gamma^* k^2 = \frac{\left(k^2\lfloor \frac{k}{2} \rfloor\right)\left(k^2\lceil \frac{k}{2} \rceil\right)}{k^3} \text{ which implies } \gamma^* = \frac{\lfloor \frac{k}{2} \rfloor \lceil \frac{k}{2} \rceil}{k}$$

When k is even, the expression simplifies to $\gamma^* = k/4$. When k is odd, the expression

simplifies to $\gamma^* = (k^2-1)/4k$. For any k-ary n-mesh, independent of n,

$$\gamma^* = \begin{cases} \frac{k}{4} & k \text{ is even} \\ \frac{k^2-1}{4k} & k \text{ is odd} \end{cases} \tag{2.1}$$

The network capacity is the inverse of γ^*.

The maximum channel load $\gamma(R,\Lambda)$ for a routing algorithm R and traffic matrix Λ is the expected traffic load crossing the most heavily loaded channel under R and Λ. The worst-case channel load $\gamma_{wc}(R)$ for a routing algorithm R is the heaviest channel load that can be caused by any admissible traffic, where admissible traffic is defined to be any doubly sub-stochastic matrix Λ with all row and column sums bounded by 1. Suppose a network consists of N nodes, a traffic matrix $\Lambda = (\lambda_{ij})$ is an $N \times N$ matrix where λ_{ij} represents the expected traffic from node i to node j. The traffic matrix Λ is doubly sub-stochastic and hence admissible if

$$\sum_{i=1}^{N} \lambda_{ij} \leq 1, \forall j \text{ and } \sum_{j=1}^{N} \lambda_{ij} \leq 1, \forall i$$

and it is said to be doubly stochastic if

$$\sum_{i=1}^{N} \lambda_{ij} = 1, \forall j \text{ and } \sum_{j=1}^{N} \lambda_{ij} = 1, \forall i$$

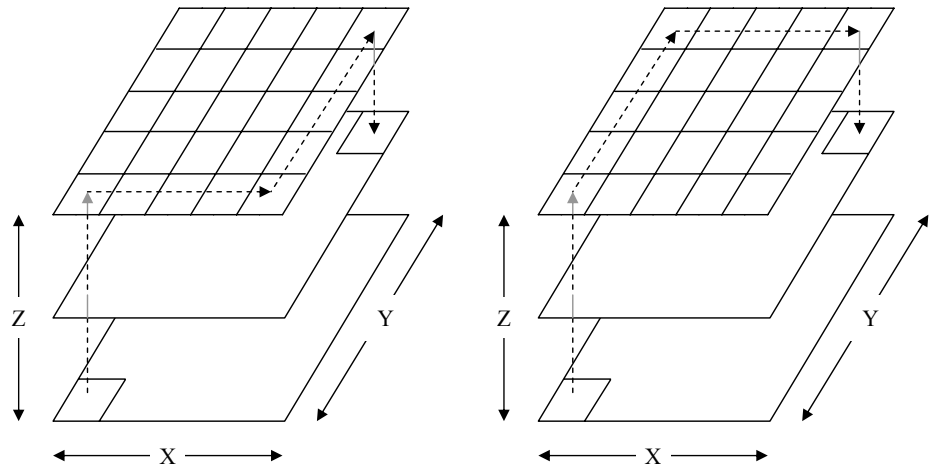
As shown in [64], an admissible traffic matrix that can cause the worst-case channel load for a routing algorithm R can be found by solving a derived maximum weighted matching problem. The worst-case saturation throughput for a routing algorithm R is the inverse of the worst-case channel load. The normalized worst-case saturation throughput, $\Theta_{wc}(R)$, is defined as the worst-case saturation throughput normalized to the network capacity:

$$\Theta_{wc}(R) = \left(\frac{\gamma_{wc}(R)}{\gamma^*} \right)^{-1} \tag{2.2}$$

Unless otherwise noted, we will simply refer to $\Theta_{wc}(R)$ as the worst-case throughput of R. Using the methodology used in [53, 64], the average-case throughput for a routing algorithm R can be computed by averaging the throughput over T, a large set of random traffic patterns:

$$\Theta_{avg}(R) = \frac{1}{|T|} \sum_{\Lambda \in T} \left(\frac{\gamma(R,\Lambda)}{\gamma^*} \right)^{-1} \tag{2.3}$$

For our evaluations, we use $|T|$ = 100,000.



(a) Z-XY-Z path (b) Z-YX-Z path

Figure 2.1: Examples of RPM routing.

2.3 Randomized partially-minimal routing

The idea behind RPM is fairly simple. Conceptually, RPM works by load-balancing flits uniformly across all k vertical layers along the Z dimension, just like VAL [66], but only along one dimension. RPM then routes flits on each XY plane using minimal XY or YX routing with equal probability. Finally, RPM routes flits to their final destinations along the Z dimension. Figure 2.1 depicts two possible RPM routing paths. In particular, let (x_1, y_1, z_1) be the source, (x_2, y_2, z_2) be the destination, and $\hat{z}$ be the randomly chosen intermediate Z position. The two corresponding Z-XY-Z and Z-YX-Z routing paths are $(x_1, y_1, z_1) \rightarrow (x_1, y_1, \hat{z}) \rightarrow (x_2, y_1, \hat{z}) \rightarrow (x_2, y_2, \hat{z}) \rightarrow (x_2, y_2, z_2)$ and $(x_1, y_1, z_1) \rightarrow (x_1, y_1, \hat{z}) \rightarrow (x_1, y_2, \hat{z}) \rightarrow (x_2, y_2, \hat{z}) \rightarrow (x_2, y_2, z_2)$, respectively, with at most three turns. When $x_1 = x_2$ and $y_1 = y_2$, the traffic is just uniformly randomized along the Z dimension. In this case, when $\hat{z}$ is greater than both z_1 and z_2, or when $\hat{z}$ is less than both z_1 and z_2, a loop is formed in the path $z_1 \rightarrow \hat{z} \rightarrow z_2$. These loops can be removed online before routing a packet to reduce hop count. When the source and destination are the same, no routing is necessary. It should be noted that although we use load-balancing along the Z dimension for this description, RPM can be equivalently defined by load-balancing uniformly along any one dimension and routing minimally in the two remaining dimensions.

2.3.1 Throughput analysis

In this section, we prove that RPM achieves optimal worst-case throughput when the network radix is even (k in a $k \times k \times k$ mesh network) and within a factor of $1/k^2$ of optimal when k is odd. Since the $1/k^2$ term diminishes quadratically, the worst-case throughput of RPM when k is odd rapidly converges to optimal with increasing k. We prove this near-optimality in three parts. We first prove that for any doubly sub-stochastic traffic matrix Λ for a $k \times k \times k$ mesh, RPM's uniform load-balancing along the Z vertical dimension will guarantee that the corresponding traffic matrix $\hat{\Lambda}$ for each $k \times k$ horizontal plane is also doubly sub-stochastic. We next prove that the worst-case channel load on each XY plane is bounded by $k/2$, meaning that the worst-case channel load on any channel in the X or Y dimension is bounded by $k/2$. We also prove that the worst-case channel load for the vertical channels is bounded by $k/2$. Finally, using the expression shown in Equation 2.2 for computing the worst-case throughput, we characterize the near optimal nature of RPM.

Claim 1. *Given any 3D doubly sub-stochastic traffic matrix, the 2D traffic that will traverse any XY plane using RPM will be doubly sub-stochastic.*

Proof. It suffices to show that for any doubly stochastic traffic matrix Λ, each corresponding 2D traffic matrix $\hat{\Lambda}$ will be doubly stochastic. Let $\lambda[(x_s, y_s, z_s), (x_d, y_d, z_d)]$ be the traffic from (x_s, y_s, z_s) to (x_d, y_d, z_d). By definition of doubly stochastic, the traffic *from* any source (x_s, y_s, z_s) or *to* any destination (x_d, y_d, z_d) in Λ must all sum to 1.

$$\sum_{x=0}^{k-1}\sum_{y=0}^{k-1}\sum_{z=0}^{k-1} \lambda[(x_s, y_s, z_s), (x, y, z)] = 1 \tag{2.4}$$

$$\sum_{x=0}^{k-1}\sum_{y=0}^{k-1}\sum_{z=0}^{k-1} \lambda[(x, y, z), (x_d, y_d, z_d)] = 1 \tag{2.5}$$

Let $\hat{\Lambda}$ be the 2D traffic matrix for a plane at some $z = \hat{z}$, and let $\hat{\lambda}[(x_s, y_s), (x_d, y_d)]$ be the traffic between two nodes (x_s, y_s) and (x_d, y_d) on this plane. Given the two-phase load balancing in the Z dimension, we have

$$\hat{\lambda}[(x_s, y_s), (x_d, y_d)] = \sum_{z_i=0}^{k-1}\sum_{z_j=0}^{k-1} \frac{\lambda[(x_s, y_s, z_i), (x_d, y_d, z_j)]}{k} \tag{2.6}$$

For $\hat{\Lambda}$ to be doubly stochastic, the row sum *from* any (x_s, y_s) or the column sum *to* any (x_d, y_d) in $\hat{\Lambda}$ must all be 1. These conditions can be stated as follows:

$$\sum_{x=0}^{k-1}\sum_{y=0}^{k-1}\hat{\lambda}[(x_s,y_s),(x,y)] = 1 \tag{2.7}$$

$$\sum_{x=0}^{k-1}\sum_{y=0}^{k-1}\hat{\lambda}[(x,y),(x_d,y_d)] = 1 \tag{2.8}$$

First substituting Equation 2.6 and then Equation 2.4 into the LHS of Equation 2.7, we have

$$\begin{aligned}
&\sum_{x=0}^{k-1}\sum_{y=0}^{k-1}\sum_{z_i=0}^{k-1}\sum_{z_j=0}^{k-1}\frac{\lambda[(x_s,y_s,z_i),(x,y,z_j)]}{k} \\
&= \sum_{z_i=0}^{k-1}\left(\sum_{x=0}^{k-1}\sum_{y=0}^{k-1}\sum_{z_j=0}^{k-1}\frac{\lambda[(x_s,y_s,z_i),(x,y,z_j)]}{k}\right) \\
&= \sum_{z_i=0}^{k-1}\frac{1}{k} = 1
\end{aligned}$$

Similarly, first substituting Equation 2.6 and then 2.5 into the LHS of Equation 2.8, we have

$$\begin{aligned}
&\sum_{x=0}^{k-1}\sum_{y=0}^{k-1}\sum_{z_i=0}^{k-1}\sum_{z_j=0}^{k-1}\frac{\lambda[(x,y,z_i),(x_d,y_d,z_j)]}{k} \\
&= \sum_{z_j=0}^{k-1}\left(\sum_{x=0}^{k-1}\sum_{y=0}^{k-1}\sum_{z_i=0}^{k-1}\frac{\lambda[(x,y,z_i),(x_d,y_d,z_j)]}{k}\right) \\
&= \sum_{z_j=0}^{k-1}\frac{1}{k} = 1
\end{aligned}$$

Since all rows and columns in $\hat{\Lambda}$ sum to 1, the 2D traffic matrix $\hat{\Lambda}$ is doubly stochastic. This analysis holds for all $z \in [0,k)$. □

Claim 2. *The maximum channel load in a $k \times k \times k$ mesh network with RPM routing is $k/2$.*

Proof. By Claim 1, load balancing uniformly in the Z dimension guarantees that the 2D traffic on each XY plane is doubly sub-stochastic. It has already been shown in the context of 2D meshes that minimal XY and YX routing with equal probability results in

a maximum channel load of $k/2$ [53]. Hence, it follows that the worst-case channel load on any channel in the X or Y dimension using RPM is bounded by $k/2$. Given the two-phase load balancing in the Z dimension, the one-dimensional (1D) traffic along any Z line is twice uniform. For 1D meshes, the maximum channel load for uniform traffic is $k/4$ when k is even and $(k^2-1)/4k$ when k is odd. The maximum channel load for twice uniform is simply $2(k/4) = k/2$ when k is even and $2(k^2-1)/4k = (k^2-1)/2k$ when k is odd. Since $k/2 \geq (k^2-1)/2k$, the worst-case channel load is upper bounded by $\gamma_{wc}(\text{RPM}) = k/2$.

□

Claim 3. *RPM achieves optimal worst-case throughput when k is even and within a factor of* $1/k^2$ *of optimal when k is odd.*

Proof. By Claim 2, $\gamma_{wc}(\text{RPM}) = k/2$. As discussed in Section 2.2, $\gamma^* = k/4$ when k is even. Therefore, using Equation 2.2, we have

$$\Theta_{wc}(\text{RPM}) = \left(\frac{k/2}{k/4}\right)^{-1} = 0.5$$

This is optimal since the optimal worst-case throughput has already been shown previously to be half of the network capacity [12]. When k is odd, $\gamma^* = (k^2-1)/4k$. Therefore, we have

$$\begin{aligned}
\Theta_{wc}(\text{RPM}) &= \left(\frac{k/2}{(k^2-1)/4k}\right)^{-1} \\
&= \left(\frac{k}{2} \cdot \frac{4k}{(k^2-1)}\right)^{-1} \\
&= \frac{k^2-1}{2k^2} \\
&= 0.5\left(1 - \frac{1}{k^2}\right)
\end{aligned}$$

This is within a factor of $1/k^2$ of optimal.

□

Figure 2.2 shows the worst-case throughput of RPM in comparison to VAL, DOR, ROMM, and O1TURN on symmetric 3D mesh networks with different radices. This graph was derived using the technique discussed in Section 2.2 for determining

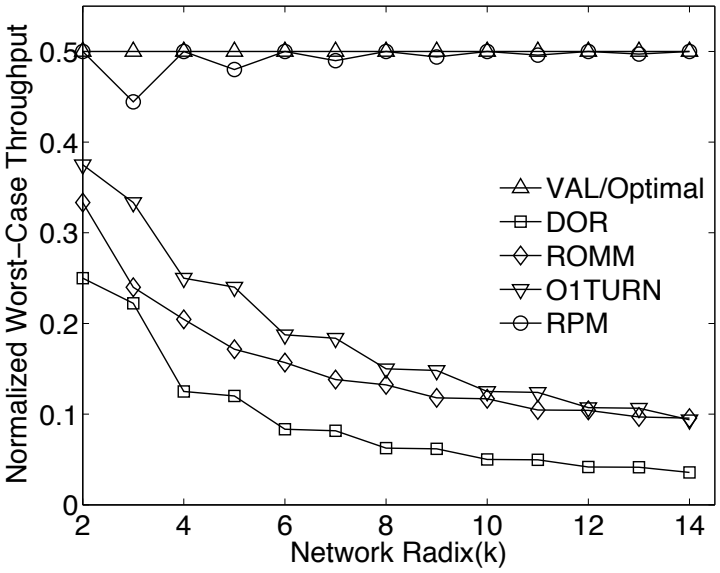


Figure 2.2: Normalized worst-case throughput of different routing algorithms on symmetric 3D mesh networks.

the worst-case throughput of oblivious routing algorithms. RPM achieves the same optimal worst-case throughput as VAL when the network radix is even. The worst-case throughput of RPM quickly converges to optimal for odd radices as the network radix is increased. Note that the performance of DOR, ROMM, and O1TURN all degrade tremendously with increasing radix. At $k = 14$, the worst-case throughput of RPM is 14 times higher than DOR and 5.26 times higher than both ROMM and O1TURN.

2.3.2 Latency analysis

Let $H_{avg}(R)$ denote the average packet latency of routing algorithm R measured as the number of router hops. The average hop count calculation assumes equal traffic between all source-destination pairs in the network. For a $k \times k \times k$ mesh, the average hop count for DOR is

$$H_{avg}(\text{DOR}) = 3\left(\frac{k^2-1}{3k}\right) = \frac{k^2-1}{k}$$

Each $(k^2-1)/3k$ component corresponds to the average hop count along a single dimension using DOR [12].

In RPM, minimal routing is used in the X and Y dimensions resulting in an average hop count of $(k^2-1)/3k$ along each of these two dimensions. When two-phase routing is used in the Z dimension without any loop removal, the average hop count for this dimension is two times minimal, i.e., $2[(k^2-1)/3k]$. However, as mentioned in Section 2.3, a possibility of loop removal exists in the Z dimension when the X and Y coordinates of the source and destination are the same. Once loops are removed, two-phases of Z routing reduce to a single phase of minimal routing. The X and Y coordinates of source and destination nodes are equal with a probability of $1/k^2$. This results in the following expression for the average hop count of RPM:

$$H_x(\text{RPM}) = H_y(\text{RPM}) = \left(\frac{k^2-1}{3k}\right)$$
$$H_z(\text{RPM}) = \left(\frac{k^2-1}{k^2}\right) \times \left(\frac{2(k^2-1)}{3k}\right) + \frac{1}{k^2} \times \left(\frac{k^2-1}{3k}\right)$$

$$\begin{aligned} H_{avg}(\text{RPM}) &= H_x(\text{RPM}) + H_y(\text{RPM}) + H_z(\text{RPM}) \\ &= \left(\frac{4}{3} - \frac{1}{3k^2}\right) \times H_{avg}(\text{DOR}) \end{aligned}$$

The penalty factor in average latency for using a partially minimal routing algorithm like RPM instead of a minimal routing algorithm like DOR can be quantified by computing the ratio of their average latencies. In particular, the latency ratio of two routing algorithms R_1 and R_2, $LR(R_1, R_2)$, is defined as

$$LR(R_1, R_2) = \frac{H_{avg}(R_1)}{H_{avg}(R_2)} \tag{2.9}$$

Therefore, the penalty factor for RPM is given as:

$$LR(\text{RPM}, \text{DOR}) = \frac{4}{3} - \frac{1}{3k^2} \tag{2.10}$$

$LR(\text{RPM}, \text{DOR})$ converges to 1.33 for relatively large values of k. This is much smaller than the penalty factor of $LR(\text{VAL}, \text{DOR}) = 2$ that VAL has to sacrifice to achieve optimal worst-case throughput.

2.3.3 RPM on asymmetric meshes

Claim 4. *RPM achieves optimal worst-case throughput when $k_{max} = \max(k_x, k_y, k_z) = k_z$ and k_{max} is not equal to k_x or k_y. When $k_{max} = k_x$ or $k_{max} = k_y$, RPM is optimal when*

k_{max} is even and is within a factor of $1/k_{max}^2$ of optimal when k_{max} is odd.

Proof. The maximum channel load and worst-case throughput of an asymmetric mesh topology is determined by its longest dimension. Claim 1 already shows that the 2D traffic on any XY plane using RPM will be doubly sub-stochastic. Suppose $k_{max} = \max(k_x, k_y, k_z)$ is equal to either k_x or k_y. Since routing on the XY plane using RPM is the same as O1TURN, it follows that RPM achieves optimal worst-case throughput when k_{max} is even, and within a factor of $1/k_{max}^2$ of optimal otherwise. If k_{max} is equal to k_z, the worst-case throughput of RPM only depends on the maximum channel load in the Z dimension, which is optimal because the traffic along the Z dimension is twice uniform (cf. proof of Claim 2). □

The latency analysis for asymmetric meshes is also very similar to the symmetric case. For an asymmetric $k_x \times k_y \times k_z$ mesh, the average hop count for DOR is given as the sum of hop counts along the three dimensions:

$$H_{avg}(\text{DOR}) = \left(\frac{k_x^2 - 1}{3k_x}\right) + \left(\frac{k_y^2 - 1}{3k_y}\right) + \left(\frac{k_z^2 - 1}{3k_z}\right) \tag{2.11}$$

The average hop count for RPM, after loop removal along the Z dimension, is given as:

$$H_{avg}(\text{RPM}) = \left(\frac{k_x^2 - 1}{3k_x}\right) + \left(\frac{k_y^2 - 1}{3k_y}\right) + \left(2 - \frac{1}{k_x k_y}\right)\left(\frac{k_z^2 - 1}{3k_z}\right) \tag{2.12}$$

The ratio of the latencies of RPM and DOR for the asymmetric case can then be calculated using Equation 2.9.

2.3.4 Extending RPM to higher dimensions

In this section, we extend RPM to consider higher dimensional asymmetric meshes. Let $a_0, a_1, \ldots, a_{n-2}, a_{n-1}$ be the dimensions of an n-dimensional mesh, and let $k_0, k_1, \ldots, k_{n-2}, k_{n-1}$ be the corresponding radices. RPM can be readily extended by first uniformly load-balancing flits across the last $(n-2)$ dimensions using dimension-ordered routing, namely along $a_2, a_3, \ldots, a_{n-2}, a_{n-1}$. RPM then routes flits minimally

along the two dimensions a_0 and a_1 using either the a_0a_1 or a_1a_0 order with equal probability. Finally, RPM routes flits to their destinations along a_2, a_3, ..., a_{n-2}, a_{n-1} in dimension order.

Claim 5. *RPM achieves optimal worst-case throughput when* $k_{max} = \max(k_0, k_1, \ldots, k_{n-1})$ *is equal to any one of* k_2, k_3, ..., k_{n-1} *and* k_{max} *is not equal to* k_0 *or* k_1. *If* $k_{max} = k_0$ *or* $k_{max} = k_1$, *RPM is optimal when* k_{max} *is even and within a factor of* $1/k_{max}^2$ *of optimal when* k_{max} *is odd.*

Proof. The proof for the above claim is a direct extension of the proof for Claim 4 and is not presented here to avoid repetition. □

2.4 Router implementation

In this section, we discuss how RPM can be efficiently integrated into a typical on-chip router. We first explain how RPM can be made deadlock-free using virtual channels. We then present some details about the modifications needed in existing on-chip routers to implement RPM.

2.4.1 Virtual channels and deadlocks

In general, RPM can be defined by load-balancing uniformly along any one dimension and routing minimally in the two remaining dimensions (Z-XY/YX-Z, X-YZ/ZY-X or Y-XZ/ZX-Y). For practical asymmetric mesh topologies where the number of device layers (Z dimension) is expected to be fewer than the number of nodes along the edge of a layer (X and Y dimensions), two-phase routing along the short Z dimension and minimal routing along the longer X and Y dimensions results in highest average-case throughput. On the other hand, for symmetric 3D mesh topologies, a randomized version of RPM where Z-XY/YX-Z, X-YZ/ZY-X, and Y-XZ/ZX-Y routings are used with equal probability, yields the highest average-case throughput as it balances channel load along all three dimensions. Randomization does not change the worst-case throughput of RPM because each one of the three routing algorithms combined has the same near-optimal worst-case throughput.

Virtual channels (VCs) are needed in on-chip routers to avoid cyclic resource dependencies, like buffer dependencies, which can result in deadlocks. If RPM is implemented by load-balancing only along one dimension (lets say the Z dimension), two virtual channels per physical channel are sufficient to achieve deadlock-free routing. One approach is to divide the input buffers on links along the Z dimension into two VCs − VC-0 reserved for phase-1 of Z routing and VC-1 reserved for phase-2 of Z routing. The buffers on links along the X and Y dimensions are also divided into two VCs − VC-0 reserved for packets using X-Y routing and VC-1 reserved for packets using Y-X routing. This VC allocation scheme ensures deadlock-free operation as the resulting channel dependency graph is acyclic [10].

The randomized version of RPM, which involves load-balancing packets along each dimension with equal probability and routing minimally along the two remaining dimensions, can be made deadlock-free using three VCs. One VC allocation approach for this case is to let packets start in VC-0 and increment the VC number after every YX, ZY or ZX turns. Since the routing paths have at most two of these three possible turns, three VCs are sufficient. This VC allocation scheme results in an acyclic channel dependency graph for randomized RPM.

2.4.2 RPM router

Baseline 3D router

Figure 2.3 shows the architecture of a typical 7-port router for 3D mesh networks. This architecture is a direct extension of 5-port routers used in 2D mesh networks, with the addition of two extra ports for vertical communication. At each input port, buffers are organized as separate FIFO queues, one for each VC. Flits entering the router are placed in one of these queues depending on their VC ID. The router is generally pipelined into five stages comprising of route computation, VC allocation, switch allocation, switch traversal and link traversal. The route computation stage determines the output port of a packet based on its destination. This is followed by VC allocation where packets acquire a virtual channel at the input of the downstream router. A packet that has acquired a VC arbitrates for the switch output port in the switch

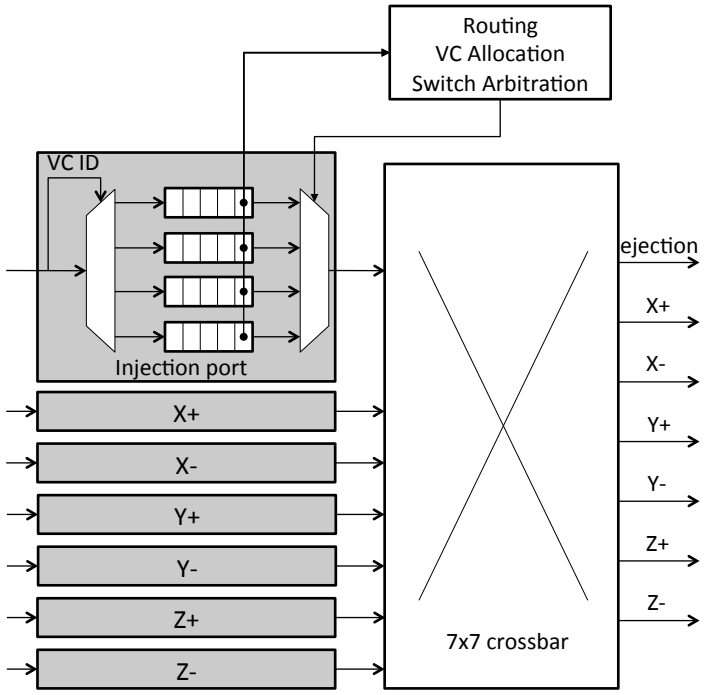


Figure 2.3: Baseline router architecture.

arbitration stage. Flits that succeed in switch arbitration traverse the crossbar before finally traversing the output link to reach the downstream router. Head flits proceed through all pipeline stages while the body and tail flits skip the route computation and VC allocation stages and inherit the output port and VC allocated to the head flit. The tail flit releases the reserved VC after departing the upstream router.

In order to implement a new routing algorithm like RPM in place of existing routing algorithms like DOR using this baseline architecture, only the route computation and VC allocation stages of the router pipeline need to be modified.

Choosing an intermediate layer

Since RPM involves load-balancing packets to a randomly chosen intermediate layer, some extra logic needs to be added to the baseline architecture to choose an intermediate layer. In most NoCs there is a strict requirement that flits of a packet need to arrive at the destination in-order. Hence, load-balancing across layers has to be carried out at the granularity of packets and not flits.

The logic for picking a random intermediate layer can be implemented using a simple linear feedback shift register (LFSR), which is often used to generate a pseudo-random sequence. Let us assume that packets are load-balanced uniformly along the Z dimension. If k_z is the number of layers in the topology, an LFSR with $\log_2 k_z$ bits is sufficient to generate a pseudo random sequence of k_z layer identifiers. In order to increase the randomness of the LFSR sequence for small values of k_z, a pseudo-random sequence with greater than $\log_2 k_z$ bits can be generated and the intermediate layer can be chosen by performing a modulo-k_z operation on the generated sequence. For example, in a $8 \times 8 \times 4$ mesh topology with 256 nodes, layer load balancing can be carried out using the last two bits of a 8-bit a pseudo-random sequence. To ensure that the random number generators at the different nodes work independently, the LFSR at a node can be initialized to the unique 8-bit node address. In general, for a network with N nodes, a $\log_2 N$ bit LFSR can be used to pick an intermediate layer. Each LFSR can then be initialized to an unique initial state (which is the unique $\log_2 N$ bit node address) resulting in different pseudo-random sequences being generated at different nodes. In the special case when the X and Y coordinates of the destination are same as the corresponding coordinates of the source, the decision of the LFSR is overridden and the intermediate layer is forced to be the Z coordinate of the destination. If the pseudo-random sequence generation process and the packet injection process (determining packet size) are independent, over a period of time the number of flits sent to different layers is expected to be equal.

If more accurate load-balancing is desired, a more sophisticated credit-based load balancing scheme can be employed for multi-flit packets. In this technique, every node in the network maintains k_z credit counters, one for each layer, all initialized to 0. The first layer with non-negative credits is always chosen to route a packet. When layer ℓ^* is selected to route an M flit packet, the credit counter corresponding to ℓ^* is decremented by $M - M/k_z$ since layer ℓ^* receives an excess of $M - M/k_z$ flits compared to ideal flit-level load-balancing across k_z layers. At the same time, counters corresponding to all other layers are incremented by M/k_z to account for the flit deficit with respect to ideal flit-level load-balancing. This layer selection approach requires k_z counters at each router to keep track of the credits associated with each layer. The

counter size depends on the maximum packet length and the number of layers in the topology. If the maximum packet length is L flits, the credit values can range from $-L$ to $(k_z - 1)L$. As explained earlier, when the X and Y coordinates of a packet's destination are same as those of the source, the Z coordinate of the destination is forced to be the intermediate layer in order to remove loops. The counters remain unchanged in this special case as no extra load is added to links along the X and Y dimensions of the destination layer.

After an intermediate layer is chosen at the time of packet injection, the intermediate layer number must be included in the packet header to enable the route computation stage to route packets to the appropriate layer. The intermediate layer selection can be carried out at the source router one cycle before packet injection to avoid increasing the critical path delay of the router.

Choosing XY/YX routing

RPM uses minimal XY or YX routing with equal probability on each horizontal layer. The logic to choose XY or YX routing paths can be the same as the approach described in [53], which uses a single signed counter to keep track of the excess/deficit of flits using XY or YX routing. Another approach is to use one of the bits generated by the LFSR used for layer selection to randomly choose between XY and YX routing. The decision to use XY or YX routing can be taken before packet injection, in parallel with intermediate layer selection, and the routing decision can be stored as a part of the packet header. This approach avoids adding extra delay at intermediate routers.

Routing and VC allocation

In a baseline router using dimension-ordered routing, the routing can be decomposed into the X, Y and Z dimensions. The route computation stage first routes a packet along the X dimension, followed by the Y dimension and finally, the Z dimension. For packets at the *injection*, X+ and X- inputs of a router (refer Figure 2.3), the route computation logic needs to determine the X, Y and Z offsets to a packet's

final destination and choose the first productive dimension.[4] For packets at the Y+ and Y- inputs, the routing decision is based on the Y and Z offsets to the packet's final destination and for packets at the Z+ and Z- inputs, the decision is based only on the Z offset.

Next, we describe one possible high-level implementation of the route computation stage for asymmetric 3D mesh networks where packets are load-balanced only along the *Z* dimension. Route computation is closely tied to the VC allocation scheme used in the router. Packets are routed using either Z-XY-Z or Z-YX-Z routing paths and two VCs are needed to make RPM deadlock-free in the asymmetric mesh case, as described in Section 2.4.1. We assume that the input buffers at all router ports are divided into two sets of VCs, VC set 0 and VC set 1. Each VC set can in turn have one or more VCs. The two VC sets at different physical channels are associated with different RPM routing segments, as summarized in Table 2.1. A packet is injected into either VC set 0 or VC set 1 at the *injection* port. The routing decisions for packets at different input ports and input VCs are taken as follows:

- For packets at any VC set of the *injection* port and VC set 0 of the Z+ and Z- ports, the offset along the Z dimension to the intermediate layer is determined along with the X and Y offsets to the final destination. If the packet has reached the destination, it is simply ejected from the network. If the Z offset to the intermediate layer is non-zero, packets are routed along the Z dimension on VC set 0. On the other hand, if the Z offset is zero and a packet is chosen to use XY routing, the packet is forwarded to the output port along the X dimension on VC set 0, if the X offset is non-zero. If the X offset is also zero, the packet is forwarded to the output port along the Y dimension on VC set 0. Alternatively, if the packet is chosen to use YX routing, it is forwarded to the output port along the Y dimension on VC set 1, if the Y offset is non-zero. If the Y offset is also zero, the packet is forwarded to the output port along the X dimension on VC set 1.

- For packets at VC set 0 of X+ and X- ports, the X, Y and Z offsets to the final destination are computed. If either X or Y dimensions are productive, packets

[4] A dimension is said to be productive if the packet has not yet reached the destination coordinate corresponding to the dimension.

Table 2.1: RPM VC allocation.

Physical Channel	Virtual Channel	Routing Segment
Injection	Set 0, Set 1	Packet Injection
Z+, Z-	Set 0	Z Phase-1
Z+, Z-	Set 1	Z Phase-2
X+, X-, Y+, Y-	Set 0	XY routing
X+, X-, Y+, Y-	Set 1	YX routing

are forwarded on VC set 0 using XY routing. Once packets reach the X and Y coordinates of the destination, they are either ejected, if the Z offset is 0, or forwarded along the appropriate Z output port on VC set 1. Packets at VC set 0 of Y+ and Y- ports are routed similarly, the only difference being that the X offset can be ignored for these packets as they are guaranteed to have reached the X coordinate of the destination.

- For packets at VC set 1 of Y+ or Y- ports, the Y, X and Z offsets to the final destination are computed. If either Y or X dimensions are productive, packets are forwarded on VC set 1 using YX routing. Once packets reach the Y and X coordinates of the destination, they are either ejected, if the Z offset is 0, or forwarded along the appropriate Z output port on VC set 1. Packets at VC set 1 of X+ or X- ports are routed similarly, the only difference being that the Y offset can be ignored for these packets as they are guaranteed to have reached the destination Y coordinate.

For symmetric mesh topologies, a randomized version of RPM that needs three sets of VCs is used, as discussed in Section 2.4.1. In addition to choosing an intermediate layer and the order of dimension traversal within a layer, the randomized variant also needs to select one of X, Y or Z dimensions for load-balancing packets. This decision has to be taken during packet injection and stored in the packet header. The route computation at intermediate routers can then be divided into three parallel planes to handle, X-YZ/ZY-X, Y-XZ/ZX-Y, and Z-XY/YX-Z routing, based on the routing plane chosen for a packet at the time of injection. The VC allocation stage looks at the

input port, the input VC set, and the output port of a packet to determine the output VC set. The output VC set is equal to input VC set + 1 if the packet is making a YX, ZY or ZX turn. Otherwise, the output VC set is equal to the input VC set.

2.5 Randomized minimal first routing

RPM achieves near-optimal worst-case throughput in 3D mesh networks by load-balancing packets uniformly across all vertical layers and routing minimally on the horizontal layers. However, RPM is not minimal in terms of latency since it needs to route packets to a randomly chosen intermediate layer. As described in Section 2.4.2, the intermediate layer selection process in RPM is oblivious to the packet's destination, which may result in routing in non-minimal directions. Non-minimal routing can be avoided to some extent by using a packet's destination in the layer-selection process and preferentially choosing an intermediate layer in the minimal direction. In this section, we present a destination-aware layer selection technique that can reduce the latency of RPM when the traffic is inherently load-balanced, such as uniform random traffic. We refer to this variant of RPM as Randomized Minimal First (RMF) routing.

In a simple preferential load-balancing approach where traffic is load-balanced along the vertical Z dimension, a node keeps track of the number of flits sent to all k_z vertical layers. For routing a packet from source (x_s, y_s, z_s) to destination (x_d, y_d, z_d), the route computation logic first looks at the number of flits sent to the minimal layers, i.e. layers between z_s and z_d. If any of these layers have a deficit of flits, the packet is sent to the first minimal layer with a flit deficit. If none of the minimal layers have a deficit, the packet is routed to a non-minimal layer. This approach requires k_z counters at a node, similar to the load-balancing technique described in Section 2.4.2, and some logic to identify minimal layers. The main intuition behind this approach is that if the traffic is inherently load-balanced, packets can be routed to intermediate layers in the minimal direction more often, without causing an overall imbalance in the traffic routed to different layers. This simple preferential load-balancing technique, however, is not provably worst-case throughput optimal.

As discussed in Section 2.3.1, the foundation for the worst-case throughput

optimality of RPM is laid using Claim 1, which shows that the projected 2D traffic on any XY plane using RPM is doubly sub-stochastic provided the 3D traffic matrix is doubly sub-stochastic. The preferential load-balancing technique described above satisfies Equation 2.4 because every node still sends only $1/k_z^{th}$ fraction of the total traffic it generates, to a particular layer. This approach, however, cannot guarantee that Equation 2.5 is satisfied. This is because, it is possible that a node (x_1, y_1, m) on the m^{th} layer receives more flits than (x_1, y_1, n) on the n^{th} layer as layer m might be in the minimal direction of more traffic flows to coordinates (x_1, y_1), compared to layer n. As a result, node (x_1, y_1, m) may receive more than $1/k_z^{th}$ fraction of the total traffic destined to the set of nodes at coordinates (x_1, y_1). A source node with just k_z counters may not see this imbalance as it may be sending more traffic to a different (x, y) coordinate of the n^{th} layer, (x_2, y_2, n), compared to (x_2, y_2, m). Therefore, in order to make RMF worst-case throughput optimal, preferential layer load-balancing needs to be done individually for each (x, y) column of the network. RMF routing, described next, does exactly that.

For implementing RMF, a node s in the network maintains a set of credit counters $C_s(x,y,z)$ that count the number of credits available to send flits to the (x,y) coordinate of layer z. The counters $C(x,y,z)$ are all initialized to zero. When source s at (x_s, y_s, z_s) needs to route an M flit packet to destination (x_d, y_d, z_d), the route computation stage looks at the values of k_z credit counters, $C_s(x_d,y_d,0 \ldots k_z-1)$. If any one of the minimal layers between z_s and z_d (including z_s and z_d) has non-negative credits, it is selected as the intermediate routing layer. If all minimal layers have negative credits, a non-minimal layer with non-negative credits is chosen as the intermediate layer. To guarantee flit-level load-balancing similar to the counter-based layer selection technique described in Section 2.4.2, if layer ℓ^* is chosen as the intermediate routing layer, $C_s(x_d,y_d,\ell^*)$ is decremented by $M - M/k_z$. At the same time, $C_s(x_d,y_d,z)$ is incremented by M/k_z for all $z \neq \ell^*$. Instead of choosing a minimal layer with non-negative credits, RMF can be generalized to choosing a minimal layer ℓ^* with $C_s(x_d,y_d,\ell^*) > -T$, where T is a positive threshold. This approach allows packets to be routed minimally even when the minimal layers are temporarily oversubscribed by at most T flits. However, over time, the imbalance caused by T flits is negligible for small values of T.

In this way, RMF satisfies Claim 1, which implies that the worst-case channel

load on any channel in the X or Y dimensions is $max(k_x, k_y)/2$. The load on the Z channels with RMF is strictly less than the corresponding load on the Z channels with RPM because the combined channel load of phases 1 and 2 of Z routing is reduced by preferring minimal paths. Therefore, Claims 2 and 3 with regard to worst-case throughput optimality also hold for RMF.

The above description can be extended to a randomized version of RMF, which load balances packets along all three dimensions with equal probability (instead of just the Z dimension) and routes minimally in the two remaining dimensions. The randomized variant of RMF is better suited for symmetric mesh networks because it balances the channel load equally between the X, Y, and Z channels. However, randomized RMF requires a separate set of counters for load balancing flits along each of the X, Y, and Z dimensions. As in the case of RPM, randomization is not needed for practical asymmetric topologies where load-balancing along the shorter Z dimension leads to higher throughput.

When the traffic is not inherently load-balanced, like permutation traffic patterns, where a source sends all its traffic to a single destination, RMF routing degenerates to RPM as the layer load-balancing requirement at each (x, y) column forces packets to use non-minimal intermediate layers. The performance comparison of RPM and RMF is presented in Section 2.7 along with an estimate of the overhead required for implementing RMF.

2.6 Layer-multiplexed 3D architecture

The 3D router described in Section 2.4.2 is a natural extension to a 2D mesh router where two extra ports are added in the 3D case to support vertical communication, resulting in a 7×7 crossbar compared to a 5×5 crossbar in the 2D case. However, since the crossbar power and area costs increase quadratically with the number of ports, 3D routers have higher power consumption and a larger area footprint than their 2D counterparts. Increased number of ports in 3D routers increases contention in switches and the complexity of arbiters. Also, in a 3D mesh, packets are routed in a *one-layer-per-hop* manner along the vertical dimension. When RPM routing is

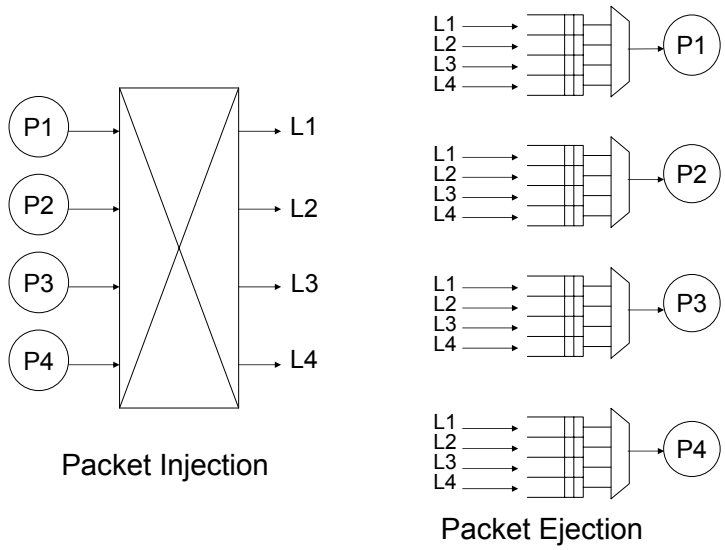


Figure 2.4: Packet injection and ejection stages of the layer-multiplexed architecture.

used, which involves two phases of vertical routing, a packet may need to traverse $2k$ layers (and hence $2k$ router hops) in the worst-case, resulting in long worst-case latencies. Along with long delays, *one-layer-per-hop* communication is also inefficient in terms of power consumption as a flit needs to traverse a router pipeline at every hop despite the short inter-layer distances in 3D designs. These factors motivated us to propose a new Layer-Multiplexed (LM) 3D on-chip network architecture that exploits the throughput optimality of RPM together with the short inter-layer wiring delays and the abundance of vertical wiring in 3D designs. The LM architecture has a lower power cost, a smaller area footprint and higher performance compared to a 3D mesh, as shown in Section 2.7.3.

2.6.1 Architecture

In the LM architecture, the one-layer-per-hop communication in a 3D mesh is replaced with simpler demultiplexing and multiplexing structures to take advantage of the short vertical wiring delays and the abundance of vertical wiring in 3D designs. The high level architecture of these structures is shown in Figure 2.4 for $k = 4$ layers. These demultiplexing and multiplexing stages are used to implement the layer load-balancing of RPM. RPM requires two-phase routing along the vertical dimension to load-balance

traffic across the layers. In the LM architecture, the two phases of vertical routing are carried out at packet injection and packet ejection, respectively. At packet injection, flits are *demultiplexed* uniformly to the k layers using the *packet injection stage*. The packet injection demultiplexer gives every processor direct access to all the layers, thus removing the necessity for one-layer-per-hop communication. Once demultiplexed to a horizontal layer, packets are routed on this plane using either minimal XY or YX routing with equal probability. At the destination (X, Y) coordinates, packets from different layers are *multiplexed* at the destination processor in the *packet ejection stage*.

With this architecture, each horizontal plane is effectively a conventional 2D mesh with just 5-port routers. Rather than connecting directly to these 5-port routers, a processor is connected to a corresponding packet injection stage at the same (X, Y) location. The packet injection stage is in turn connected to k 5-port routers at the same (X, Y) location. This is depicted in Figure 2.4 for $k = 4$, with processors P_1 to P_4 demultiplexing to layers L_1 to L_4. Similarly, on the packet ejection side, a packet ejection multiplexer is used at each processor to multiplex flits arriving from layers L_1 to L_4. We refer to this adaptation of the RPM routing algorithm on the LM architecture as RPM-LM.

As shown later in Section 2.7.3, the combined costs of these demultiplexers and multiplexers together with the 5-port router costs are significantly lower than the 7-port router costs of a conventional 3D mesh both in terms of power and area. The microarchitectural details of these packet injection and ejection stages and the routers are detailed next.

Packet injection

The details of the packet injection stage are shown in Figure 2.5 for $k = 4$ layers. It is essentially a 4-port switch with a typical router pipeline consisting of route selection, virtual channel (VC) allocation, switch arbitration, switch traversal, and link traversal. However, route selection is modified to implement layer load-balancing, which has to ensure that statistically traffic does indeed get uniformly distributed across all k layers. This is achieved by using a set of flit-counters for each ordered pair of input and output ports in the load-balancing logic. A flit-counter (i, j) records the total number

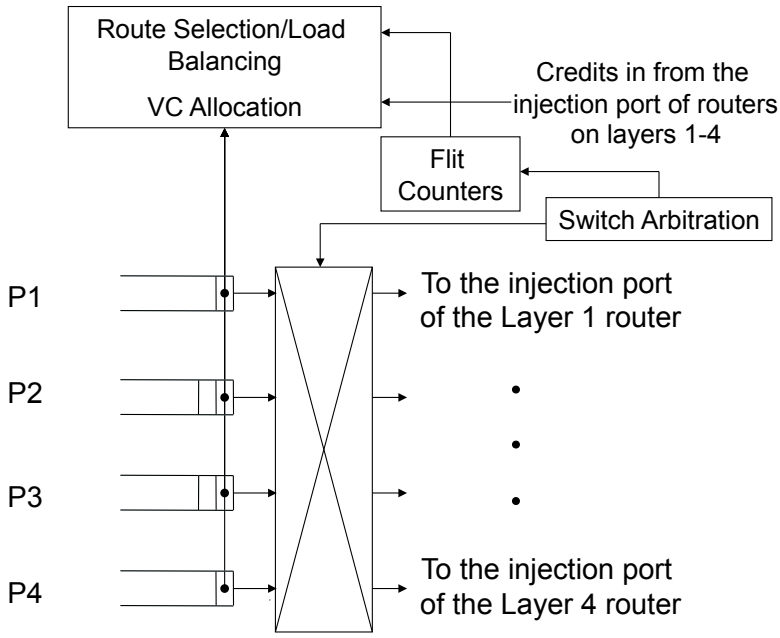


Figure 2.5: Packet injection/Demultiplexing stage.

of flits sent from input i to output j. When a new head flit is injected from processor P_i, the route selection logic selects the output layer L_j with the lowest flit-count (i, j). Ties between layers are broken by a rotating output priority pointer for P_i that gets advanced at each route selection.

The route selection logic keeps track of the ouput ports that are currently in use in order to avoid allocating the same output to more than one input, which would reduce throughput. The layer selection is done as follows:

- Each input port is given a unique priority that rotates every cycle over all inputs.
- When two or more inputs enter the route selection stage at the same cycle, the highest priority input is served first.
- Suppose input i requests for an output port and F is the set of free outputs in that cycle after the higher priority inputs have been served, the route selection logic simply chooses an output port from F that has received the least number of flits from input i.

This strategy guarantees that every input is always granted an unique output. It must

be noted that route computation is only performed on the head flit of a packet. Once a layer is chosen by the route selection stage for the head flit, all remaining flits of the same packet will be routed on the same layer. The VC allocation stage further allocates a virtual channel at the injection port of the 5-port router on the selected layer. Together, flits belonging to the same packet are guaranteed to be routed on the same layer through the same set of virtual channels, thus ensuring their in-order arrival at the destination.

Finally, we found that a single VC with a small amount of buffering (e.g. 5 flits) is sufficient at each input of the packet injection demultiplexing switch. For implementation, the input buffers can be located on the same layer as the corresponding processor. The crossbar of the packet injection demultiplexer can be spread across the k layers, i.e., the cross points for a particular layer can be located at that layer. The route selection, VC allocation and switch arbitration logic can be located on the middle layer (or one of the middle layers when k is even) so that the wiring for the control signals is minimized and also uniformly distributed across the layers.

Router microarchitecture

Figure 2.6 depicts the microarchitectural details of the 5-port routers used for routing on a horizontal plane in the LM architecture. Once a packet is injected into a router on one of the layers, RPM-LM uses either minimal XY or YX paths with equal probability to reach the (X, Y) coordinates of its destination. This is essentially O1TURN [53] routing on the horizontal plane. In turn, the microarchitecture of each 5-port router on the 2D plane is very similar to that of the O1TURN router [53]. The injection port of the router receives flits from the corresponding output port of the packet injection switch located at the same (X,Y) column. The virtual channels at each input port are divided into two sets–one set for XY routing and another for YX routing. After being injected into one of the routing layers (XY or YX), a packet is restricted to remain in the virtual channels of that layer while being routed on the 2D plane. The routing and VC allocation stages are duplicated as in the O1TURN router to independently handle the corresponding decisions in the XY and YX routings. The switch arbitration stage is common to both XY and YX routings since flits from all virtual channels at an input contend for the same crossbar port.

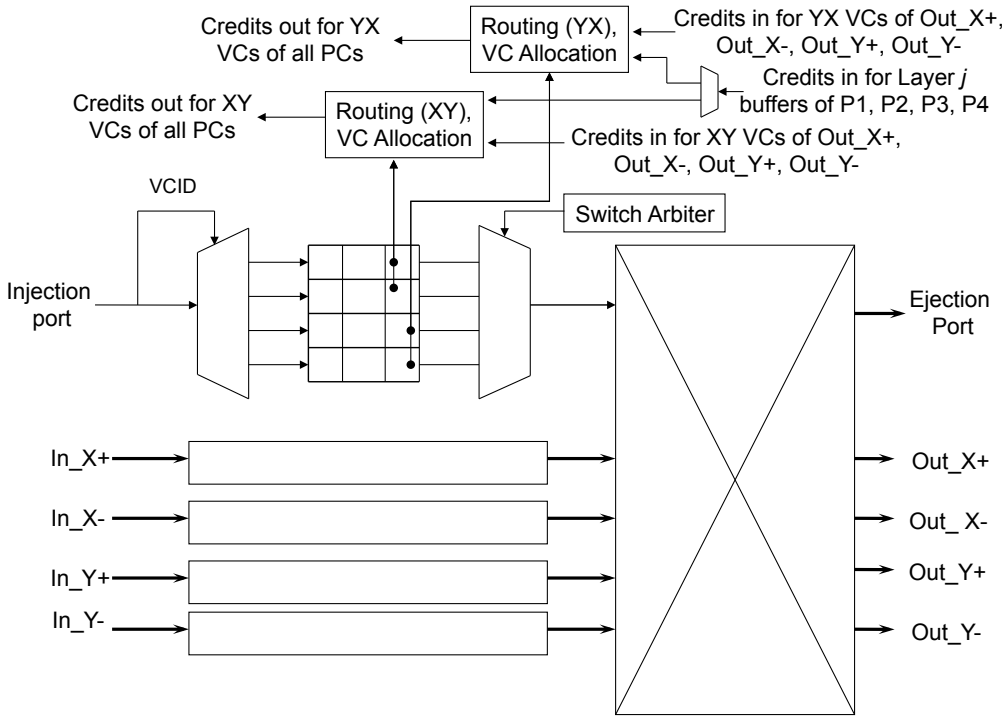


Figure 2.6: Microarchitecture of a 2D intra-layer router on layer j.

Packet ejection

The ejection port of each router is connected through packet ejection multiplexers to all processors at the same (X, Y) coordinate, as depicted in Figure 2.7. In particular, each horizontal plane router sees at its ejection port four virtual channels, each of which corresponds to an ejection queue of a processor connected to the router. In the example depicted in Figure 2.7, a router on the first layer (L_1) sees four virtual channels, namely L_1-P_1, L_1-P_2, L_1-P_3, and L_1-P_4 as its ejection channels located at processors P_1, P_2, P_3, and P_4, respectively. The VC allocation stage of the router chooses the appropriate output VC for a packet based on its destination. For example, a packet destined for processor P_2 is assigned VC L_1-P_2 while being ejected from the horizontal plane router. After leaving the horizontal plane router, the VC ID field of a flit is used to determine the packet ejection multiplexer into which the flit is inserted.

For implementation, a packet ejection multiplexer can be implemented on the corresponding processor layer. A flit ejected from a horizontal plane router can be broadcast to the packet ejection multiplexers on all layers and a decision to accept a

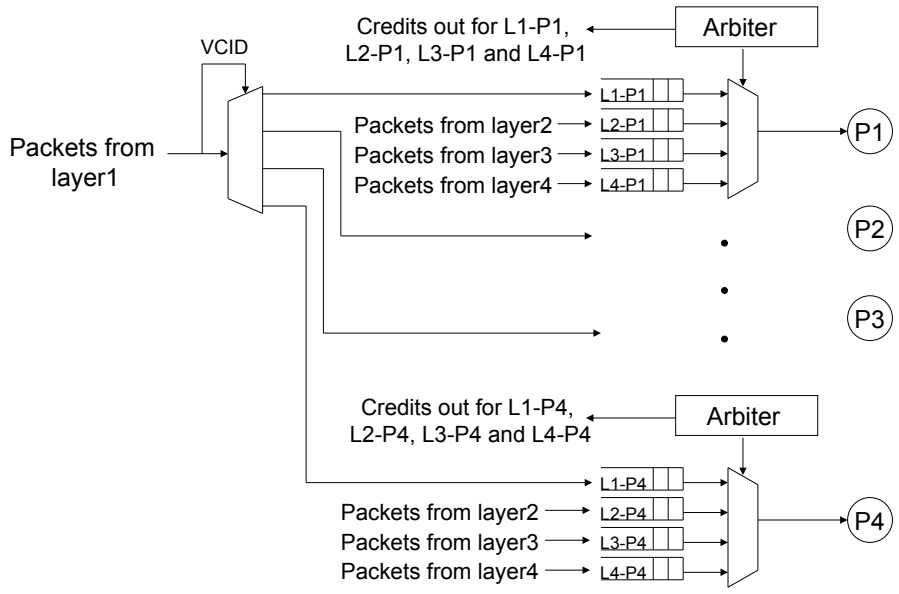


Figure 2.7: Packet ejection/Multiplexing stage.

flit can be made locally at the input buffer of each multiplexer, based on the flit's VC ID. Finally, each packet ejection multiplexer can independently choose a flit from one of its input queues to forward to the corresponding destination processor every cycle. The multiplexers also handle flow control for their respective input buffers. We found that only a small amount of buffering (e.g. 5 flits per queue) is sufficient at the multiplexer inputs to achieve good performance.

2.6.2 Analysis

Claim 6. *The worst-case throughput of RPM-LM is equal to the worst-case throughput of RPM on a 3D mesh, which is optimal when the network radix k is even and within a factor of* $1/k^2$ *of optimal when k is odd.*

Proof. For RPM-LM routing on the LM architecture, traffic is uniformly load-balanced across the vertical layers just as in RPM routing on a 3D mesh. Using the same analysis as Claim 1, the 2D traffic that will traverse any XY plane will be doubly sub-stochastic under an doubly sub-stochastic traffic matrix Λ. Therefore, the maximum channel load on the XY planes with RPM-LM will be the same as O1TURN and RPM. Given that the demultiplexing packet injection stage and the multiplexing packet ejection stage are

Table 2.2: Routing algorithms evaluated.

DOR	Dimension-order routing [59]. Packets routed minimally first in X dimension, then in Y, then in Z.
VAL	Valiant's routing algorithm [66]. Packets first routed from the source to a random intermediate node, followed by routing to the destination. Both phases use DOR.
ROMM	ROMM [46]. Packets routed using two-phase routing, like VAL, but intermediate nodes restricted to those within a cube bounded by the source and the destination.
O1TURN	O1TURN [53]. Packets routed using one of six minimal orthogonal paths (XYZ, XZY, YXZ, YZX, ZXY, ZYX).
DUATO	DUATO [16]. Minimal adaptive routing based on deadlock avoidance. Uses next-hop buffer occupancy estimate to select an output port in the minimal direction. It uses escape virtual channels [16] to avoid deadlocks.

both non-blocking under ideal load analysis, the maximum channel load is dictated by the load on the channels in the XY planes. Hence, RPM-LM on the LM architecture achieves the same worst-case throughput as RPM on a 3D mesh. This analysis holds for both symmetric and asymmetric meshes. □

2.7 Evaluation

In this section, we compare the performance of RPM with the routing algorithms described in Table 2.2. The first four are oblivious routing algorithms where the routing paths are independent of the network state. DUATO [16] is a deadlock-free minimal adaptive routing algorithm. We include an adaptive routing algorithm in our evaluation to show that oblivious routing algorithms with good load-balancing capabilities can perform as well and sometimes even better than adaptive routing algorithms, without incurring the extra overheads associated with adaptive routing.

We evaluate the performance of RPM on two symmetric ($4 \times 4 \times 4$ and $8 \times 8 \times 8$) and two asymmetric ($8 \times 8 \times 4$ and $16 \times 16 \times 4$) 3D mesh topologies. In practice, 3D mesh networks are not expected to be symmetric in 3D chip designs. The number of available device layers is expected to be fewer than the number of processor tiles that can

Table 2.3: Traffic patterns evaluated.

Worst-Case	Worst-case traffic that causes lowest throughput.
Average-Case	Average throughput over 100,000 random permutations.
Transpose (asymmetric)	Packet at (x,y,z) sent to (y,z,x). Destination obtained by left shifting the concatenated bit representation of the source xyz to yzx and repartitoning the result.
Complement	Packet at (x,y,z) sent to $(k_x-x-1, k_y-y-1, k_z-z-1)$.
DOR-WC (asymmetric)	Packet at (x,y,z) sent to $(k-z-1, k-y-1, k-x-1)$. If x is represented using b_x bits and z is represented using b_z bits, destination node of (x,y,z) is obtained by swapping the positions of the first b_x bits of the concatenated bit representation of the source with the last b_z bits, repartitioning the result, and taking its complement.
Uniform Random	Packet sent to a destination chosen at uniform random.

be placed along the edge of a device layer. Hence, we chose to evaluate the performance of RPM on asymmetric $8 \times 8 \times 4$ and $16 \times 16 \times 4$ mesh topologies, both with only 4 device layers. The randomized version of RPM discussed in Section 2.4.1 is used in the symmetric mesh topologies. For asymmetric meshes, we load-balance only along the short vertical dimension. Randomization improves the average throughput of RPM on symmetric meshes while retaining the same worst-case throughput, since it distributes traffic equally along all three dimensions. On the other hand, for asymmetric meshes, the links along the short vertical dimension are more lightly loaded compared to the links along the longer horizontal dimensions. Consequently, two-phase routing only along the lightly loaded vertical dimension offers better performance.

This section is divided into three main parts. First, Section 2.7.1 compares RPM with DOR, O1TURN, ROMM and VAL using a simplified throughput analysis technique based on channel load measurement (refer Section 2.2). This analysis assumes ideal single cycle routers with infinite buffers. Since these techniques are only applicable to oblivious routing algorithms, we do not consider comparisons with adaptive routing and destination-aware RMF routing in this section. We also defer comparisons with

RPM-LM on the LM architecture until the next section with cycle-accurate flit-level simulations. This is because under ideal analysis, the packet injection and ejection stages of the LM architecture are non-blocking structures and throughput is mainly determined by load on the X and Y channels. So ideal throughput analysis is not sufficient to reveal the performance advantages of the LM architecture over RPM routing on a 3D mesh. Moreover, since the multiplexing and demultiplexing structures of the LM architecture are tailored towards the two-phase load-balancing of RPM, we restrict ourselves to just comparing the performance of RPM on the LM and 3D mesh architectures. While other routing oblivious algorithms can be implemented with the LM architecture, both the demultiplexing and multiplexing stages may not be needed if the algorithm involves only a single phase of Z routing. Next, we back the results obtained using ideal throughput analysis with more realistic flit-level simulations in Section 2.7.2. Here, in addition to comparing RPM with existing oblivious routing algorithms, we include performance comparisons with adaptive routing, performance comparison of RMF with RPM and DOR and comparison of RPM-LM on the LM architecture with RPM on a 3D mesh. Finally, Section 2.7.3 evaluates the power and area overheads associated with implementing RPM and RMF in on-chip routers. Here, we also discuss the potential power/area benefits of using the LM architecture in place of a conventional 3D mesh.

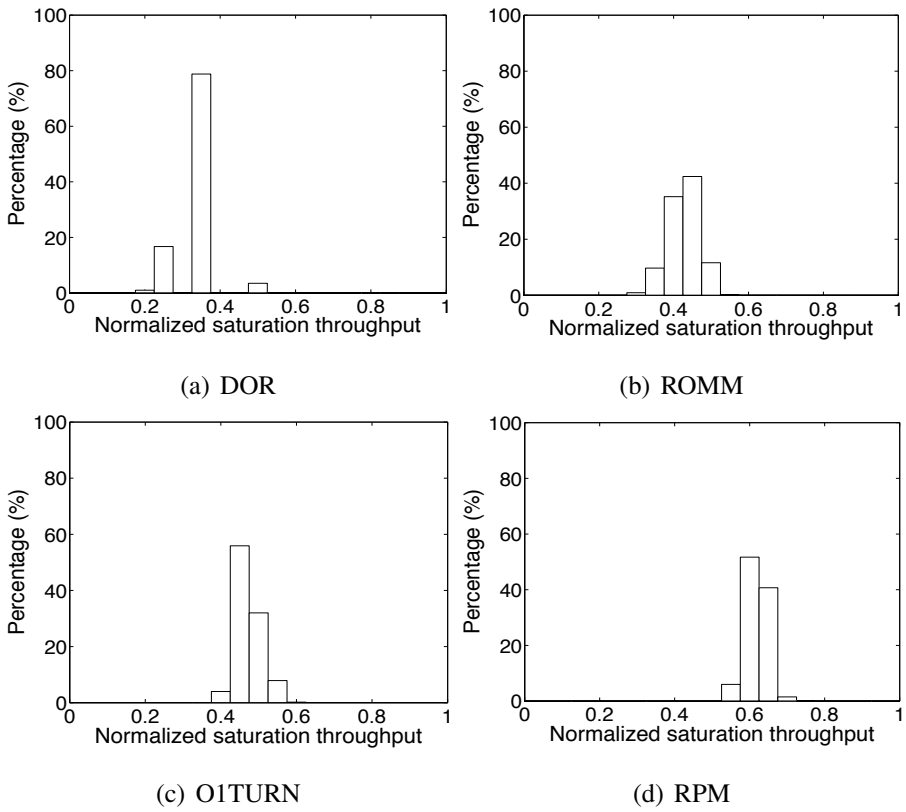
2.7.1 Performance evaluation

The saturation throughput results normalized to the network capacity for each oblivious routing algorithm on each traffic pattern of Table 2.3 are shown in Table 2.4 for the four different 3D mesh configurations. As discussed briefly in Section 2.2, we use the methodology proposed in [64] to determine worst-case throughput. Similarly, the average-case results were computed using the technique described in Section 2.2 by averaging the throughput over 100,000 randomly generated permutation traffic patterns. In addition to worst-case and average-case analysis, we also evaluate the throughput of the routing algorithms on a mixture of benign traffic patterns like uniform random traffic and adversarial traffic patterns like DOR-WC traffic, which is a worst-case traffic pattern for DOR.

Table 2.4: Comparison of normalized throughput of different routing algorithms.

	VAL	DOR	ROMM	O1TURN	RPM
	$4 \times 4 \times 4$ Network				
Worst-Case	0.5	0.125	0.205	0.25	0.5
Average-Case	0.5	0.322	0.427	0.472	0.62
Transpose	0.5	0.25	0.327	0.5	0.6
Complement	0.5	0.5	0.308	0.5	0.5
DOR-WC	0.5	0.125	0.214	0.25	0.5
Random	0.5	1	0.813	1	0.75
	$8 \times 8 \times 8$ Network				
Worst-Case	0.5	0.0625	0.13	0.15	0.5
Average-Case	0.5	0.32	0.45	0.52	0.67
Transpose	0.5	0.25	0.29	0.48	0.6
Complement	0.5	0.5	0.19	0.5	0.5
DOR-WC	0.5	0.06	0.15	0.15	0.5
Random	0.5	1	0.74	1	0.75
	$8 \times 8 \times 4$ Network				
Worst-Case	0.5	0.1	0.177	0.25	0.5
Average-Case	0.5	0.352	0.475	0.54	0.73
Transpose	0.5	0.25	0.313	0.333	0.5
Complement	0.5	0.5	0.242	0.5	0.5
DOR-WC	0.5	0.1	0.198	0.286	0.5
Random	0.5	1	0.777	1	1
	$16 \times 16 \times 4$ Network				
Worst-Case	0.5	0.083	0.148	0.25	0.5
Average-Case	0.5	0.4	0.525	0.597	0.762
Transpose	0.5	0.25	0.303	0.286	0.5
Complement	0.5	0.5	0.196	0.5	0.5
DOR-WC	0.5	0.083	0.192	0.267	0.533
Random	0.5	1	0.758	1	1

The worst-case analysis results validate that RPM indeed achieves (near) optimal worst-case throughput for both symmetric and asymmetric mesh topologies. For the symmetric $4 \times 4 \times 4$ and $8 \times 8 \times 8$ mesh topologies, RPM achieves the same optimal worst-case throughput as VAL and outperforms DOR, ROMM, and O1TURN by 300-700%, 144-284%, and 100-233%, respectively. In terms of average-case throughput for the two symmetric mesh topologies, RPM significantly outperforms VAL, DOR, ROMM, and O1TURN by 24-34%, 92-109%, 45-49%, and 29-31%, respectively.

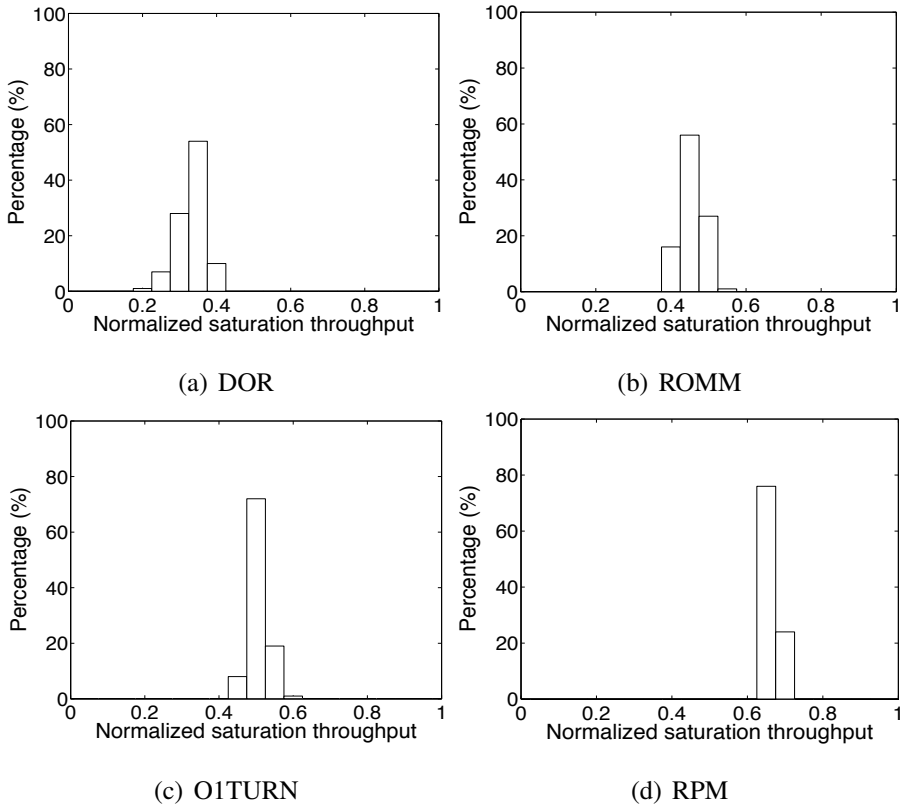


(a) DOR (b) ROMM

(c) O1TURN (d) RPM

Figure 2.8: Histograms of saturation throughputs under random permutation traffic patterns for a $4 \times 4 \times 4$ mesh.

Figures 2.8 and 2.9 present the histograms of the normalized saturation throughputs for the symmetric $4 \times 4 \times 4$ and $8 \times 8 \times 8$ topologies, respectively, under randomly generated permutation traffic patterns that were used to measure average-case throughput. The normalized saturation throughput of VAL is 0.5 for all traffic patterns. The histograms clearly show that the saturation throughput of RPM is higher than the minimal routing algorithms over the wide range of traffic patterns evaluated. Moreover, for all traffic patterns, the normalized saturation throughput of RPM is greater than 0.5, which validates its worst-case throughput optimality and also shows that it achieves higher throughput than VAL in all cases. RPM also performs very well under adversarial traffic patterns, namely on transpose and DOR-WC traffic. The throughput of DOR, ROMM, and O1TURN degrade tremendously under both these traffic patterns. Compared to the best of the three minimal routing algorithms evaluated, RPM performs 25% better on



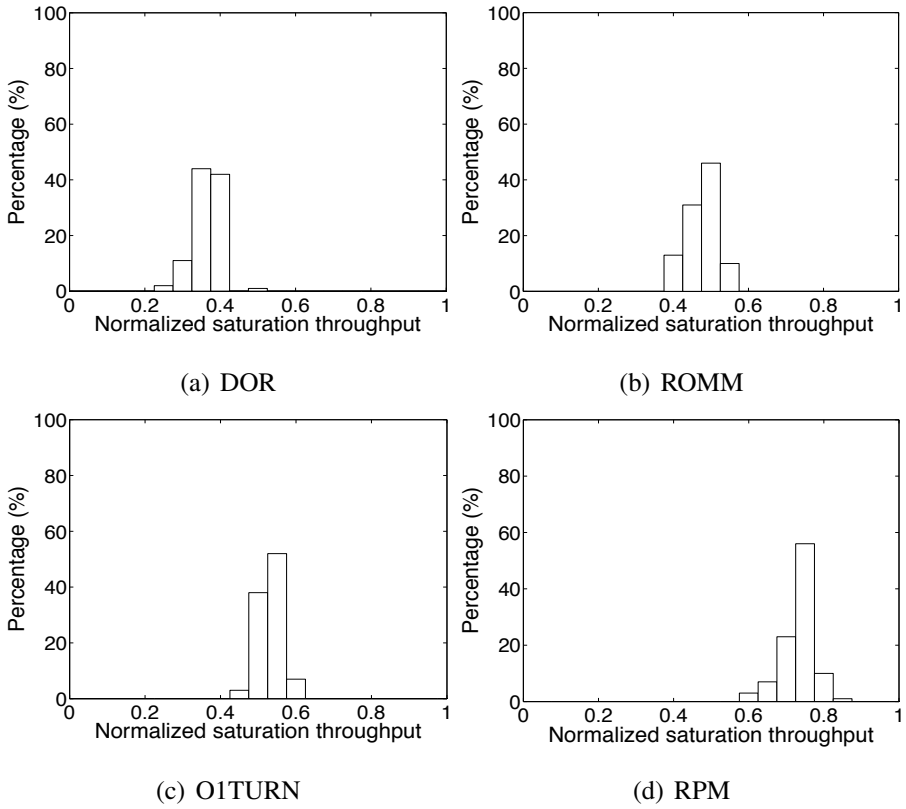
(a) DOR (b) ROMM

(c) O1TURN (d) RPM

Figure 2.9: Histograms of saturation throughputs under random permutation traffic patterns for a $8 \times 8 \times 8$ mesh.

transpose and 233% better on DOR-WC traffic for the $8 \times 8 \times 8$ configuration. Although DOR and O1TURN can achieve better normalized throughput than RPM on symmetric meshes when the traffic has already been uniformly randomized, the results for both are significantly worse in the average and worst cases to justify the use of RPM when worst-case performance is critical.

For the asymmetric $8 \times 8 \times 4$ and $16 \times 16 \times 4$ mesh topologies, RPM again achieves optimal worst-case throughput (same as VAL) and outperforms DOR, ROMM, and O1TURN by 400-500%, 182-238%, and 100% respectively. For these topologies, RPM performs strictly better than all other routing algorithms considered, both in terms of average-case throughput and throughput for the different traffic patterns except for uniform and complement traffic, where it performs as well as DOR and O1TURN. In terms of average-case throughput for the two asymmetric configurations, RPM



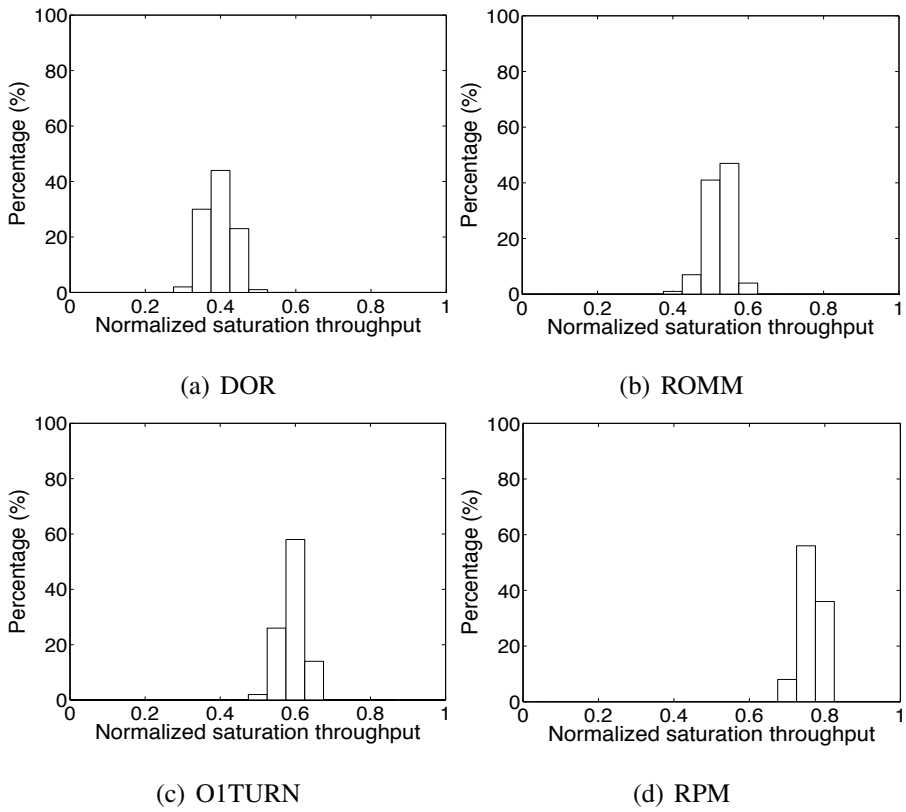
(a) DOR (b) ROMM

(c) O1TURN (d) RPM

Figure 2.10: Histograms of saturation throughputs under random permutation traffic patterns for a $8 \times 8 \times 4$ mesh.

outperforms VAL, DOR, ROMM, and O1TURN by 46-52%, 90-107%, 45-54%, and 28-35%, respectively. Figures 2.10 and 2.11 show the histograms of the normalized saturation throughputs for the asymmetric $8 \times 8 \times 4$ and $16 \times 16 \times 4$ topologies, respectively, under randomly generated permutation traffic patterns used to determine average-case throughput. The results show that over a large sample space of traffic patterns, RPM has higher saturation throughput compared to both the minimal routing algorithms and VAL, whose normalized saturation throughput is 0.5 independent of the traffic.

As stated in Equation 2.10, the latency of RPM is at most 1.33 times minimal for symmetric meshes. RPM pays a significantly smaller latency penalty compared to VAL (two times minimal) to achieve optimal worst-case throughput. The latency penalty of RPM is greatly reduced for practical asymmetric topologies like the ones



(a) DOR (b) ROMM

(c) O1TURN (d) RPM

Figure 2.11: Histograms of saturation throughputs under random permutation traffic patterns for a $16 \times 16 \times 4$ mesh.

evaluated. Using the expressions in Equations 2.11 and 2.12, the average hop count of RPM reduces to just a factor of 1.19 of DOR for the $8 \times 8 \times 4$ topology and 1.11 of DOR for the $16 \times 16 \times 4$ topology.

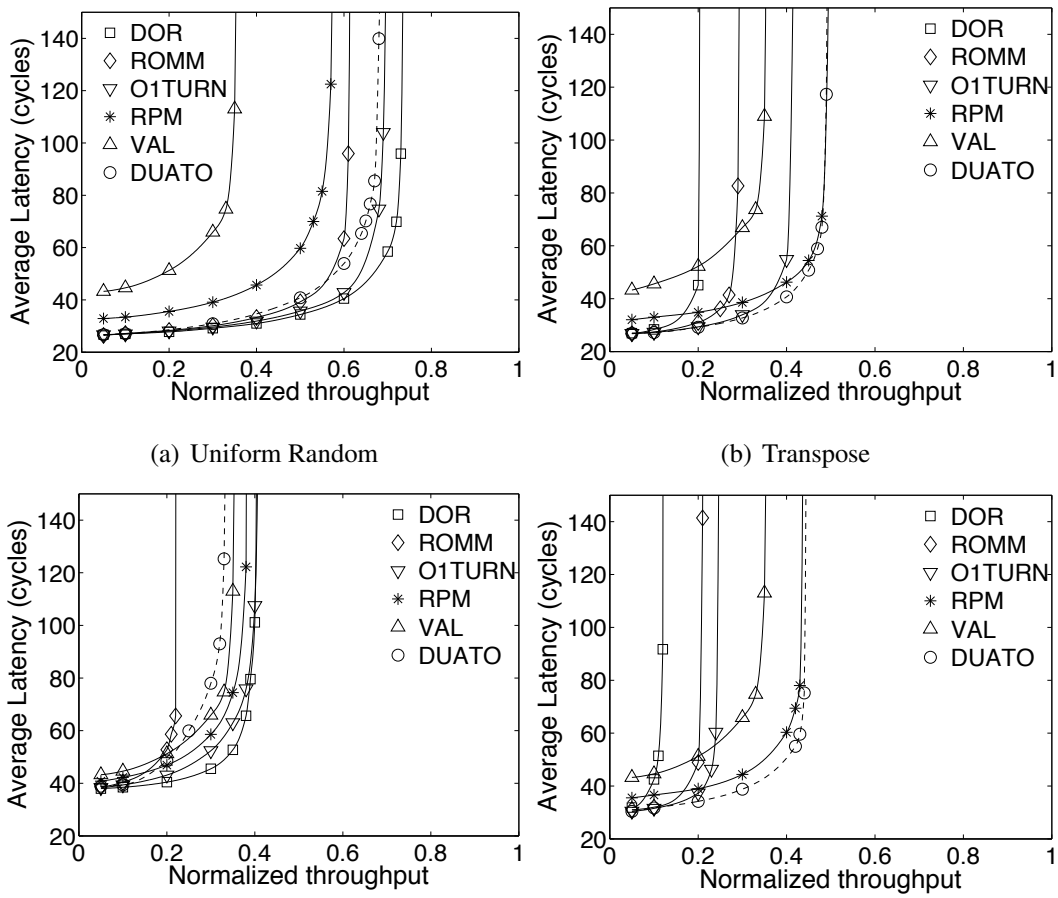
2.7.2 Detailed flit-level simulation

The results obtained using ideal throughput analysis represent upper bounds to the actual achievable throughput because it assumes ideal single-cycle routers with infinite buffers and ignores issues like flow control and contention in switches. In this section, we evaluate the performance of different routing algorithms using cycle-accurate flit-level simulations which provide more realistic insights into the performance of routing algorithms. We first describe the simulation setup and then evaluate the performance of RPM, RMF and RPM-LM.

Simulation setup

We modified the PopNet [54] on-chip network simulator to perform flit-level simulations. PopNet models a typical input-buffered VC router with five pipelined stages. Route computation is performed in the first stage followed by VC allocation, switch arbitration, switch traversal and link traversal. The head flit of a packet proceeds through all five stages while the body and tail flits bypass the first two stages and inherit the output port and output VC reserved by the head flit. Credit-based flit-level flow control is used between adjacent routers. We assume 8 virtual channels per physical channel, each 5 flits deep. For the asymmetric topologies, the 8 VCs are divided into two sets of four VCs to avoid deadlocks, as described in Section 2.4.1. Similarly, for the symmetric topologies, the 8 VCs are divided into two sets of 3 VCs and one set of 2VCs to avoid deadlocks. We include 8 VCs in our setup because it is well known that VCs improve the throughput of any routing algorithm by reducing head-of-line blocking and enabling better statistical multiplexing of flits. So, having a reasonably large number of VCs lets us compare the best performance of all routing algorithms. The injected packets are of a fixed size of 5 flits. The packet size and buffer size used are the same as the parameters identified by Wang et.al. [68] as representative approximations of the on-chip networks of RAW [63] and TRIPS [21].

We used PopNet to evaluate the average routing delays under different injection loads. For each simulation, we ran the simulator for 500,000 cycles. The latency of a packet is measured as the delay between the time the head flit is injected into the network and the time the tail flit is consumed at the destination. The four traffic patterns used in the previous section, i.e., Uniform random, Complement, Transpose and DOR-WC are also used for flit-level simulations. In addition to comparing RPM with the oblivious routing algorithms, we also implement a minimal adaptive routing algorithm based on deadlock avoidance (DUATO [16]) for comparison. The adaptive routing algorithm uses next-hop buffer occupancy to select the least congested output port. In this case, one out of the 8 VCs serves as an escape VC to avoid deadlocks.



(a) Uniform Random

(b) Transpose

(c) Complement

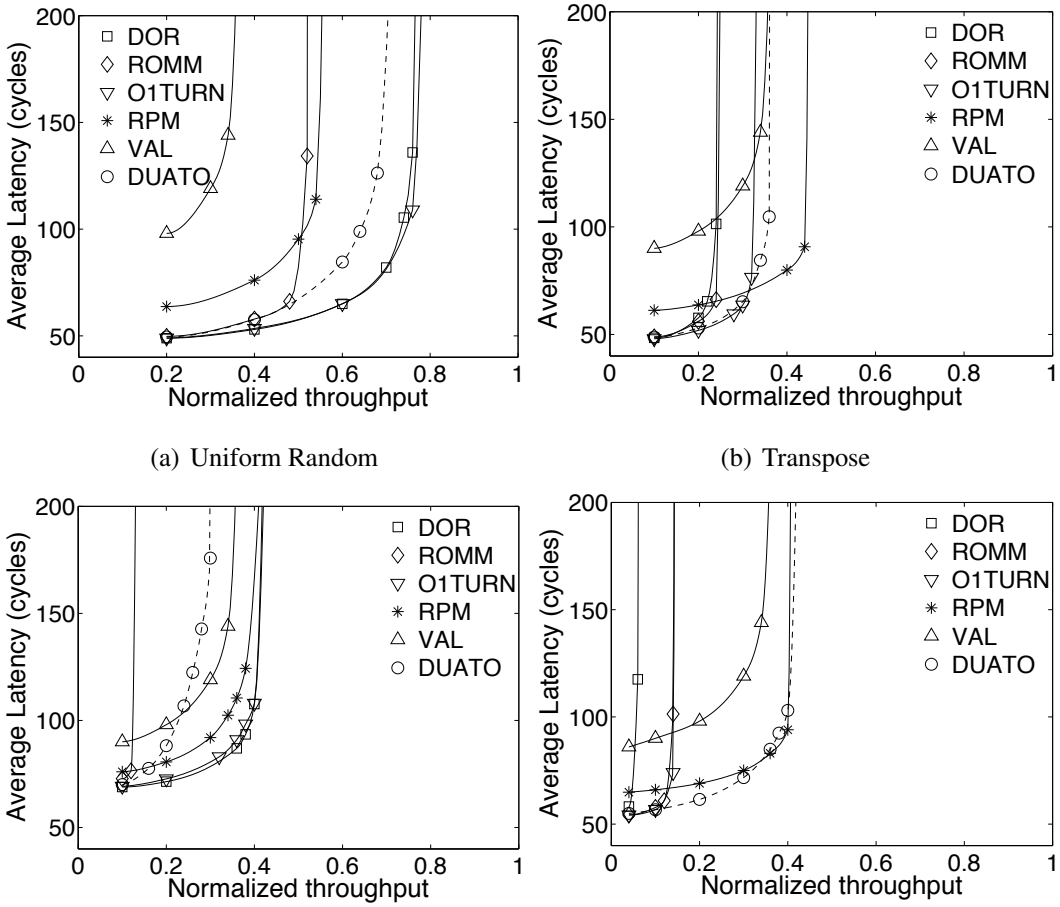
(d) DOR-WC

Figure 2.12: Performance of RPM on a $4 \times 4 \times 4$ mesh.

Evaluation of RPM

Figures 2.12, 2.13, 2.14 and 2.15 present the flit-level simulation results comparing the performance of RPM with existing oblivious and adaptive routing algorithms on the four 3D mesh configurations. The results follow a trend similar to the simplified throughput analysis presented in the previous section. RPM consistently achieves good throughput over all traffic patterns considered. The saturation throughput of RPM is higher than VAL and its latency is strictly lower than VAL over all traffic patterns evaluated and for all four topologies considered.

RPM can closely match the high throughput of adaptive routing (DUATO) on adversarial traffic like transpose and DOR-WC while other oblivious routing algorithms



(a) Uniform Random

(b) Transpose

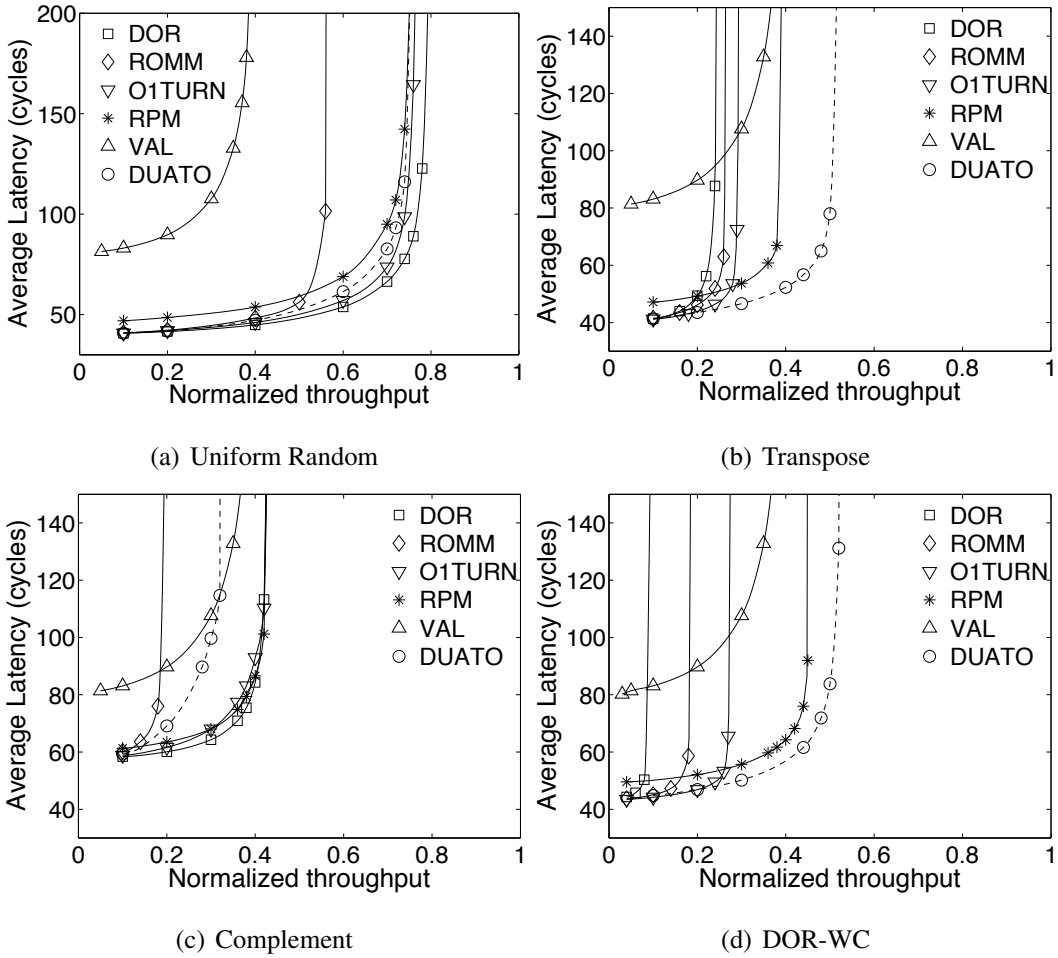
(c) Complement

(d) DOR-WC

Figure 2.13: Performance of RPM on a $8 \times 8 \times 8$ mesh.

perform poorly. RPM can, however, match DUATO's performance without adding any overhead associated with adaptive routing.

Although DOR and O1TURN perform well on uniform and bit-complement traffic, their performance degrades significantly under transpose and DOR-WC traffic, where both are clearly outperformed by RPM. The normalized throughput of O1TURN and DOR under DOR-WC traffic degrades with the increase in network radix from 4 to 8. On the other hand, the normalized throughput of RPM changes very little when the network radix is increased. RPM also outperforms ROMM on all traffic patterns considered, except uniform traffic on the $4 \times 4 \times 4$ topology. The poor performance of ROMM and DUATO on bit-complement traffic, despite having sufficient path diversity,



(a) Uniform Random

(b) Transpose

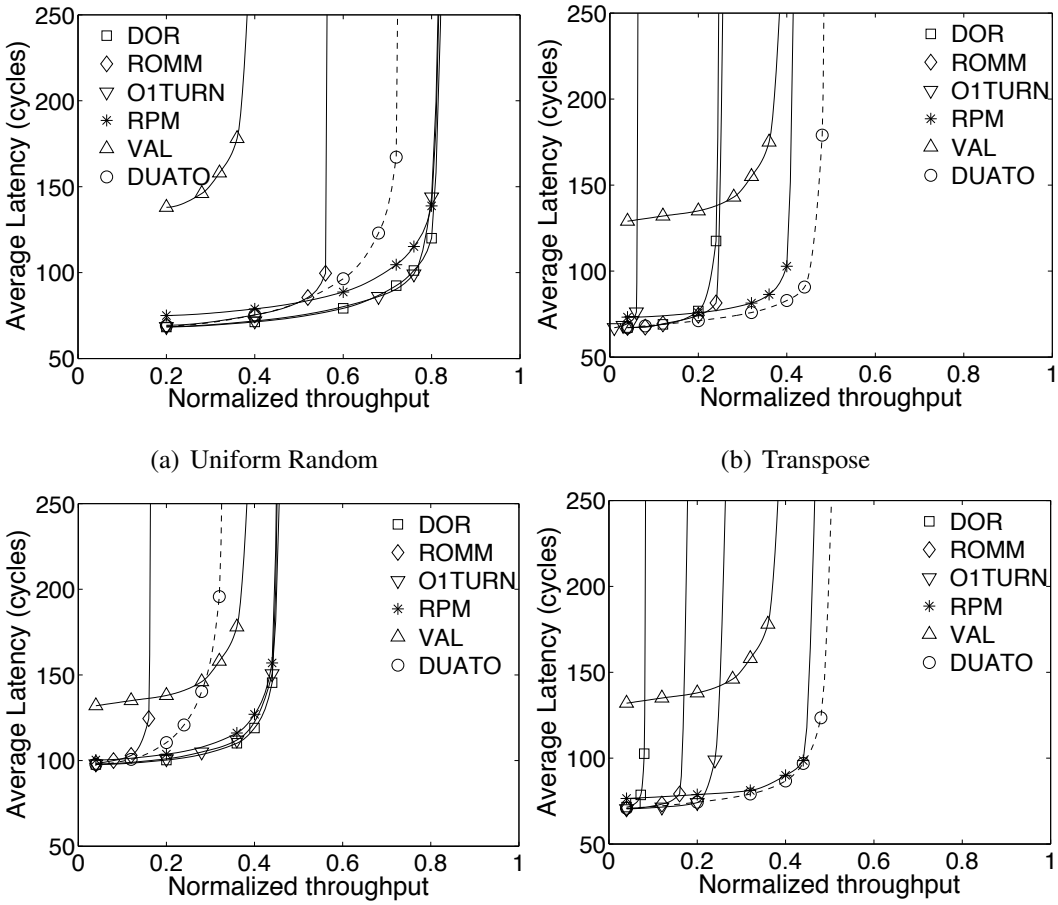
(c) Complement

(d) DOR-WC

Figure 2.14: Performance of RPM on a $8 \times 8 \times 4$ mesh.

can be attributed to the fact that they are restricted to routing in the minimal cube. This results in congestion on links in the middle of the network under complement traffic. RPM achieves better load balancing by using non-minimal paths.

For the asymmetric mesh topologies, RPM is comparable to DOR and O1TURN and better than ROMM on uniform traffic. This is because, for the asymmetric topologies, two phase routing on the short vertical dimension no longer forms a throughput bottleneck and the overall throughput is primarily determined by the saturation throughput of the horizontal channels. For the two asymmetric topologies, RPM sustains the highest (or close to highest) throughput over all oblivious routing algorithms for all four traffic patterns evaluated.



(a) Uniform Random

(b) Transpose

(c) Complement

(d) DOR-WC

Figure 2.15: Performance of RPM on a $16 \times 16 \times 4$ mesh.

Lastly, the latency of RPM is higher than the minimal routing algorithms when the network is lightly loaded. The average latency overhead of RPM over DOR can be deduced from the difference in latencies of the two routing algorithms under uniform random traffic. For the symmetric mesh configurations, the latency overhead under uniform random traffic is around 23% for the $4 \times 4 \times 4$ topology and 30.3% for the $8 \times 8 \times 8$ topology. The overheads are slightly lower than those predicted using hop-count analysis because packet delays in flit-level simulations include the transmission delay of multi-flit packets and the router pipeline delay at the destination, which are ignored while measuring hop count. The latency difference is considerably less when compared to VAL and reduces significantly for the two asymmetric configurations. In accordance

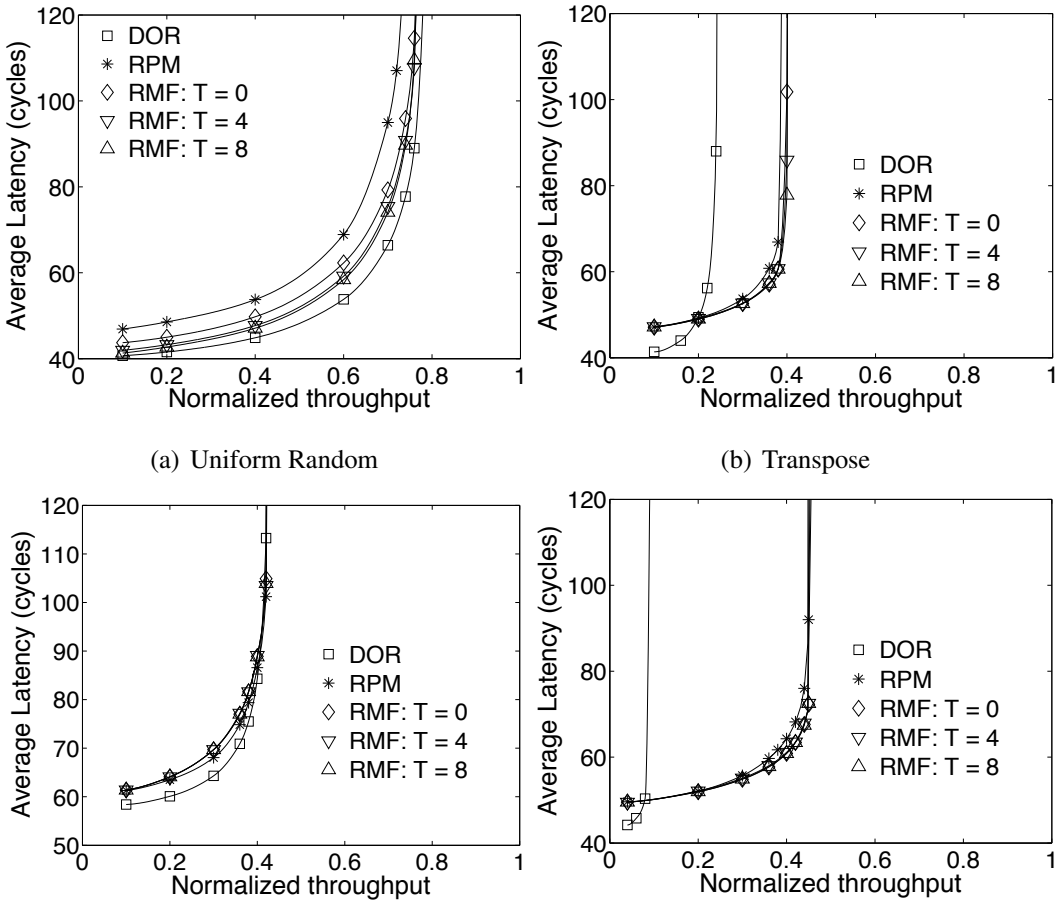
with our hop-count based calculations, the latency overhead of RPM over DOR under uniform traffic was observed to be only around 15% for the $8 \times 8 \times 4$ topology and 10% for the $16 \times 16 \times 4$ topology.

To summarize, the results clearly validate the claim that O1TURN, which achieves near-optimal worst case throughput for 2D meshes, performs poorly in the worst-case sense when extended to 3D. RPM, on the other hand, handles adversarial traffic much better than existing routing algorithms and it does so while paying a far smaller latency penalty compared to VAL. RPM is ideally suited for practical asymmetric 3D mesh configurations which have fewer device layers compared to the number of nodes along the edge of a layer. For such asymmetric topologies, RPM achieves the highest throughput among all oblivious routing algorithms and also pays negligible penalty in terms of latency over minimal routing.

Evaluation of RMF

In this section, we compare the performance of RMF with RPM and DOR under uniform random, transpose, bit complement, and DOR-WC traffic patterns. As described in section 2.5, RMF can reduce the average packet latency over RPM when the traffic is inherently load-balanced by preferentially choosing intermediate layers in the minimal direction in a destination-aware manner. Among the four traffic patterns considered, three of them (transpose, bit complement, and DOR-WC) are permutation traffic patterns where a source sends all its traffic to a single destination. Under such traffic patterns, we expect RMF routing to degenerate to RPM since RMF needs to load-balance flits across all k_z vertical layers at each (x, y) column (refer Section 2.5). On the other hand, for uniform random traffic, the packet destinations are already uniformly distributed across the vertical layers at every (x, y) column. RMF takes advantage of this fact and preferentially chooses intermediate layers in the minimal directions, thereby reducing latency without disrupting the overall balance.

Figures 2.16 and 2.17 compare the performance of RMF with RPM and DOR on the asymmetric $8 \times 8 \times 4$ and $16 \times 16 \times 4$ topologies, respectively. For these asymmetric topologies, Z-XY/YX-Z routing is used and destination-aware load-balancing is carried out at every node using a set of credit counters, as discussed in Section 2.5. The



(a) Uniform Random

(b) Transpose

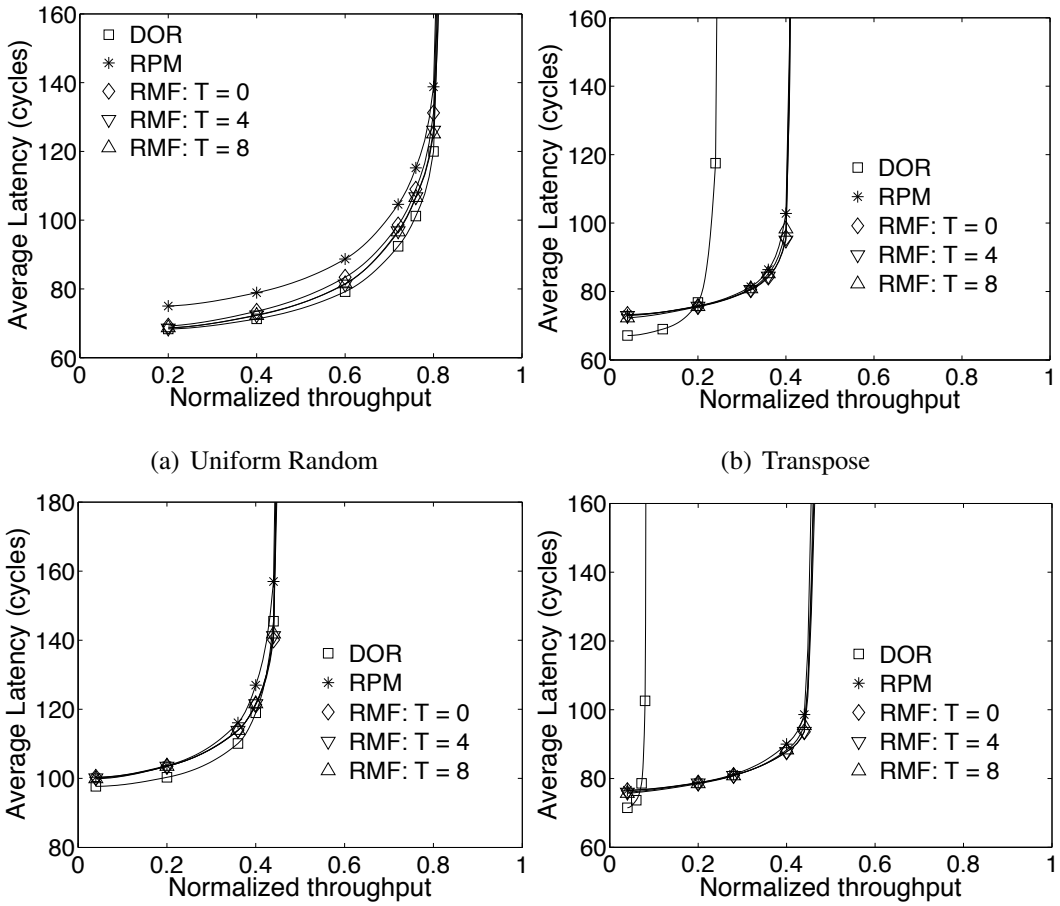
(c) Complement

(d) DOR-WC

Figure 2.16: Performance of RMF on a $8 \times 8 \times 4$ mesh.

performance of RMF using three different threshold values are presented. A threshold of T implies that the minimal layers are preferentially chosen even when they have a credit deficit of T flits with respect to the non-minimal layers.

As expected, the average latency of RMF is lower than RPM under uniform random traffic. The latency reduction increases as the threshold for preferring minimal layers is increased. For the $8 \times 8 \times 4$ topology, the latency of RMF is 7.5%, 11%, and 12.1% lower than RPM when the threshold is 0, 4, and 8 flits, respectively. This represents latency penalties over DOR of just 8%, 4%, and 2.6%, respectively, for the three thresholds. Slightly lower latency reductions of 8-9% over RPM are observed for the $16 \times 16 \times 4$ topology where the latency penalty of RPM over minimal routing is



(a) Uniform Random

(b) Transpose

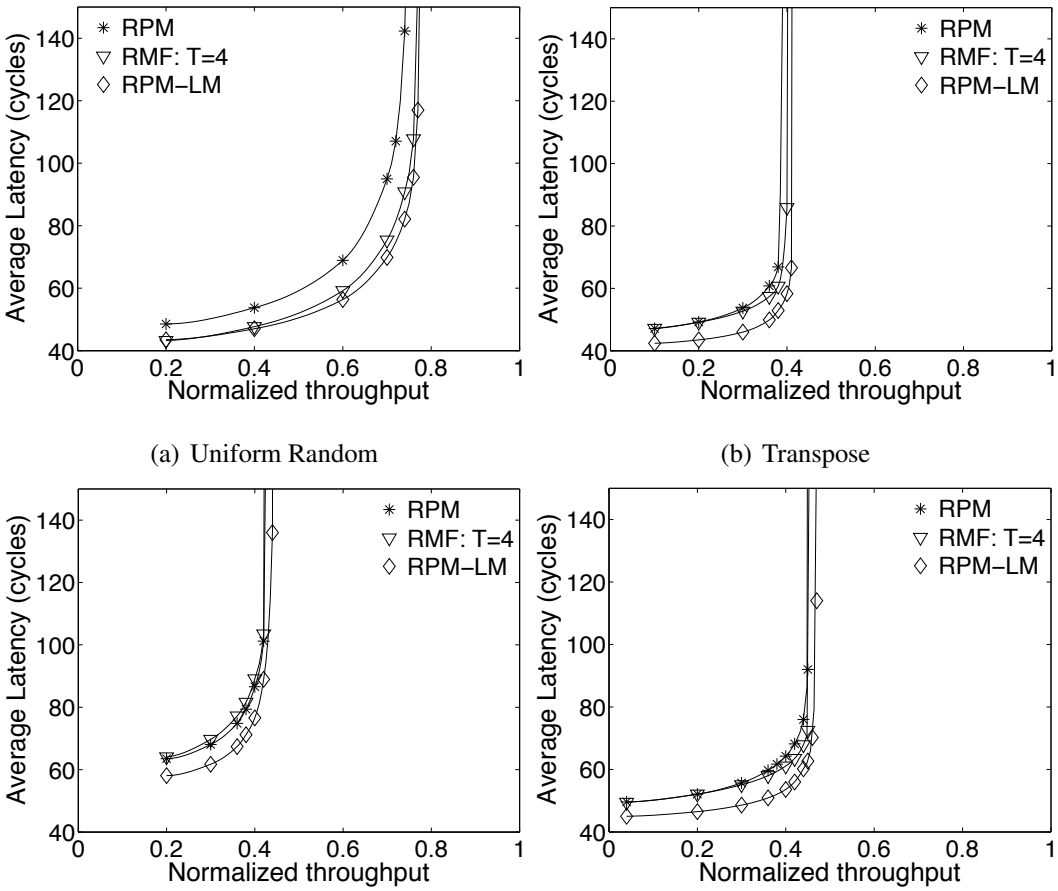
(c) Complement

(d) DOR-WC

Figure 2.17: Performance of RMF on a $16 \times 16 \times 4$ mesh.

even smaller. Here, the latency of RMF is just 0.4-1.2% higher than DOR for the three thresholds.

Finally, Figures 2.16(b), 2.16(c), 2.16(d), 2.17(b), 2.17(c), and 2.17(d) show that RMF is at least as good as RPM on the three permutation traffic patterns both in terms of latency and throughput. This proves that using RMF in place of RPM has no downside in terms of performance. Implementing RMF, however, requires the management of N counters, where N is the number of nodes in the network, as discussed in Section 2.5. The number of bits per counter depends on a number of factors like maximum packet size, threshold value (T) and the number of layers in the topology. Increasing the threshold T increases the number of bits per counter. Hence,



(a) Uniform Random

(b) Transpose

(c) Complement

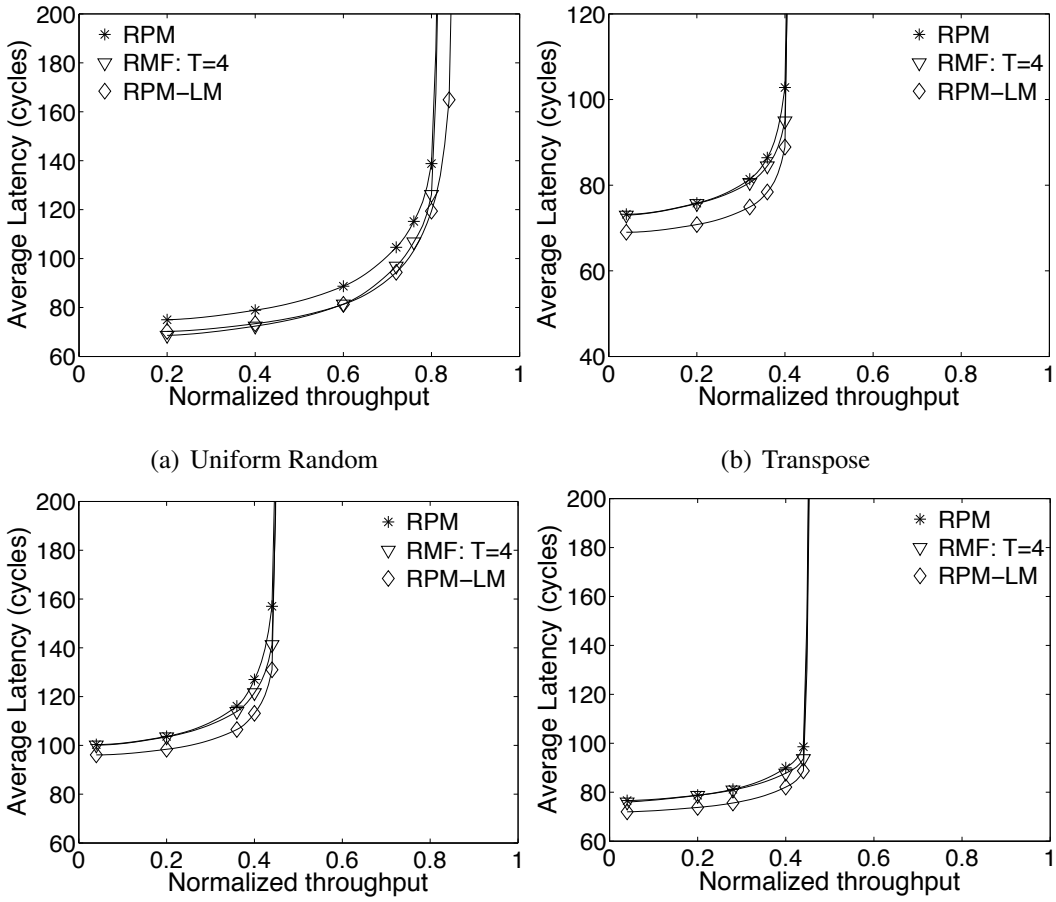
(d) DOR-WC

Figure 2.18: Performance of RPM-LM on a $8 \times 8 \times 4$ mesh.

selecting a threshold for preferential load-balancing is a tradeoff between latency and router overhead. Section 2.7.3 compares the power and area overheads associated with implementing RPM and RMF in on-chip routers.

Evaluation of RPM-LM

Next, we compare the performance of RPM-LM on the LM architecture with RPM and RMF routing on a conventional 3D mesh. Since the multiplexing and demultiplexing structures of the LM architecture are tailored towards the two-phase load-balancing of RPM, we restrict ourselves to evaluating RPM routing on the LM architecture. While other routing oblivious algorithms can be implemented with the LM



(a) Uniform Random

(b) Transpose

(c) Complement

(d) DOR-WC

Figure 2.19: Performance of RPM-LM on a $16 \times 16 \times 4$ mesh.

architecture, both the demultiplexing and multiplexing stages may not be needed if the algorithm involves only a single phase of Z routing.

Both the LM architecture and RMF routing can be considered as optimizations for reducing the latency penalty of RPM. RMF routing relies on maintaining extra state in the form of credit counters to implement preferential layer-load-balancing to reduce latency. On the other hand, the LM architecture leverages the short inter-layer distances and high vertical bandwidth in 3D ICs. Figures 2.18 and 2.19 compares the latency-throughput curves of RPM, RMF and RPM-LM on asymmetric $8 \times 8 \times 4$ and $16 \times 16 \times 4$ topologies, respectively. Both RMF and RPM-LM are well-suited for practical asymmetric topologies with few device layers to keep the overheads (i.e.,

low radix switches in the LM architecture and fewer counters for RMF routing) low. One advantage of the LM architecture is that it achieves lower latency than RPM under all traffic patterns by replacing one-layer-per-hop routing with the demultiplexing and multiplexing stages. In this regard, it is clearly superior to RMF routing which can reduce latency only when the traffic is inherently load-balanced.

From our simulations, we observe that RPM-LM can achieve comparable latency to RMF under uniform traffic but, unlike RMF, it achieves 9-10% lower latency than RPM under complement, transpose and DOR-WC traffic patterns on the $8 \times 8 \times 4$ topology. For the $16 \times 16 \times 4$ case, under low loads, the latency of RPM-LM is 4-6% lower than both RPM and RMF on complement, transpose and DOR-WC traffic patterns. Hence, lower packet latency is an important benefit of using the LM architecture in place of a 3D mesh. We evaluate the power and area benefits of the LM architecture in the next section.

2.7.3 Power and area evaluation

In this Section, we evaluate the power and area overheads associated with the RPM/RMF routers over a baseline 3D router. We also evaluate the power savings achievable by using the LM architecture in place of a conventional 3D mesh. To provide accurate comparisons, we implemented the baseline 3D router, RPM router, RMF router, LM router, LM demultiplexing stage and the LM multiplexing stage down to the gate level, and use post-layout power and area results for our comparisons. In particular, the Verilog RTL implementation for the baseline 3D router was generated using NetMaker [1], a fully-synthesizable parameterized router generator that implements an input-buffered pipelined virtual channel router. We consider a baseline 7-port router with 8 VCs per port, 5 flits per VC and a flit width of 16 bytes.

For the RPM and RMF routers, we extended the Verilog design of the baseline router by incorporating the additional logic needed to implement RPM and RMF, respectively. The routers are specifically designed for a $8 \times 8 \times 4$ mesh topology. For RPM, we used the last two bits of an 8 bit LFSR to select an intermediate layer, as discussed in Section 2.4.2. We used one of the remaining bits from the LFSR to choose between XY and YX routing. The additional logic required for RMF is a set of 256

Table 2.5: Power and area evaluation.

	Baseline router	RPM router	RMF router	LM router (amortized cost)			
				Router	Demux	Mux	Total
Power (mW)	413.46	413.53	420.56	246.3	17.75	26.73	290.78
Area (mm^2)	625987	626114	647411	375792	54895	72459	503146

counters arranged as 64 registers, each 20 bits wide. Each register contains four 5-bit counters associated with a particular (x, y) column of the network. The (x, y) coordinates of a packet's destination are used to extract the right set of counters and the operations discussed in Section 2.5 are performed to preferentially load-balance packets in minimal directions. The layer selection operation is carried out one cycle before packet injection to avoid adding to the critical path of the router pipeline.

As discussed in Section 2.6.1, the LM router is a 5-ported virtual-channel router with 8VCs per port and 5 flits per VC. For the packet injection (demultiplexing) stage of the LM architecture, we used the Verilog design for a 4-port router with a single VC per port and 5 flits per VC. We assume a $8 \times 8 \times 4$ mesh topology where the cost of a single demultiplexing stage is amortized across four processors on the four layers. Similarly, the total cost of the packet ejection (multiplexing) stages was estimated using a 4-port router with 4VCs per port and 5flits per VC. In a four-layered topology, each VC represents the ejection queue associated with one of the layers (refer Figure 2.7). The cost of a single multiplexing stage is again estimated by dividing the cost of the 4-port router by the number of layers. Since we assume inter-layer communication using Through-Silicon Vias (TSVs), the vertical wiring occupies some area on each device layer. For estimating the area overhead due to the extra wiring needed in the LM architecture, we assume vertical via pitch of 4 um [33] and eight 144 bit bundles (128 bit flits and 16 bit control signals, four bundles for demultiplexing and four for multiplexing) running vertically through all four device layers. The extra area overhead due to wiring is added to the amortized area of the demultiplexing and multiplexing structures.

The router RTLs were synthesized using Synopsys design compiler with TSMC

65 nm GP process libraries, and Cadence SoC Encounter was used for placement and routing. The frequency of operation was set at 1GHz. Table 2.5 presents the post-layout power and area for the baseline, RPM, RMF and LM routers. In comparison to the baseline 3D router, the post-layout results show that both the power and area overheads associated with implementing RPM are just around 0.02%, which are quite negligible. For the RMF router, the power and area overheads over the baseline router are 1.7% and 3.4%, respectively. The LM router requires considerably less power and area compared to 3D mesh routers. The amortized power and area per router for the LM architecture is 29.7% and 19.7% less than a baseline 3D router, which are significant improvements.

2.8 Conclusion

In this chapter, we proposed a new oblivious routing algorithm for 3D mesh networks called Randomized Partially-Minimal (RPM) routing. Although minimal routing with near-optimal worst-case throughput has already been achieved for the 2D mesh case using an algorithm called O1TURN [53], the optimality of O1TURN does not extend to 3D or higher dimensions. RPM provably achieves optimal worst-case throughput for 3D meshes when the network radix k is even and within a factor of $1/k^2$ of optimal worst-case throughput when k is odd. Furthermore, RPM significantly outperforms DOR, ROMM, O1TURN and VAL in average-case throughput by 90-109%, 45-54%, 28-35%, and 24-52%, respectively, on the different symmetric and asymmetric mesh topologies evaluated. Whereas Valiant's routing algorithm (VAL) [66] achieves optimal worst-case throughput at a penalty factor of 2 in average latency over DOR, RPM achieves (near) optimal worst-case throughput with a much smaller penalty of 1.33. In practice, the average latency of RPM is expected to be closer to minimal routing because 3D mesh networks are not expected to be symmetric in 3D chip designs. For practical asymmetric 3D mesh configurations where the number of device layers is far fewer than the number of nodes along the edge of a layer, the average latency of RPM reduces to just a factor of 1.11 of DOR. We also proposed a variant of RPM called Randomized Minimal First (RMF) routing which uses the knowledge of a packet's destination while load balancing traffic to intermediate layers.

RMF leverages the inherent load-balancing properties of the network traffic to reduce packet latency. On uniform traffic, the latency of RMF is only 0.4-2.6% higher than DOR. RMF also retains the worst-case throughput optimality property of RPM and performs at least as well or better than RPM on a wide range of traffic patterns. Finally, we proposed a new layer-multiplexed 3D architecture to efficiently implement RPM in 3D ICs. The LM architecture replaces the one-layer-per-hop routing in 3D meshes with simpler demultiplexing and multiplexing structures. This leverages the short inter-layer distances and abundance of vertical wiring in 3D ICs to reduce both power and area costs by around 30% and 20%, respectively, over a 3D mesh router while achieving comparable (or better) performance.

Chapter 2, in part, is a reprint of the material as it appears in the following publications:

- Rohit Sunkam Ramanujam and Bill Lin, "Randomized Partially-Minimal Routing on Three-Dimensional Mesh Networks", *IEEE Computer Architecture Letters*, vol. 7, no. 2, pp. 37-40, July 2008.

- Rohit Sunkam Ramanujam and Bill Lin, "Near-Optimal Oblivious Routing on Three-Dimensional Mesh Networks", *IEEE Conference on Computer Design (ICCD)*, October 2008, pp. 134-141.

- Rohit Sunkam Ramanujam and Bill Lin, "A Layer-Multiplexed 3D On-Chip Network Architecture", *IEEE Embedded Systems Letters*, vol. 1, no. 2, August 2009, pp. 50-55.

- Rohit Sunkam Ramanujam and Bill Lin, "A Novel 3D Layer-Multiplexed On-Chip Network", *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, Princeton, NJ, October 19-20, 2009.

Chapter 2, in part, has been submitted for publication of the material as it may appear in IEEE Transactions on Very Large Scale Integration, Rohit Sunkam Ramanujam, Bill Lin, "Randomized Partially-Minimal Routing: Near-Optimal Oblivious Routing for 3D Mesh Networks". The dissertation author was the primary investigator and author of the papers.

Chapter 3

Weighted Random Oblivious Routing on Torus Networks

3.1 Introduction

Several different network architectures such as torus, mesh, and flattened butterfly networks [2, 11, 63, 67] have been considered as candidates for on-chip interconnection networks. Chapter 2 focused on designing a routing algorithm with optimal worst-case throughput for 3D mesh networks. In this chapter, we study the problem of optimal oblivious routing for another important architecture class, namely, the torus network.

In the design of routing algorithms, both throughput and latency are important metrics. Although dimension-ordered routing (DOR) [59] can achieve minimal-length routing on torus networks, it suffers from poor worst-case throughput as it offers no route diversity. On the other hand, it is well known that Valiant routing (VAL) [66] can achieve optimal worst-case throughput by load-balancing globally across the entire network, but it does so at the expense of destroying locality. Other oblivious routing algorithms such as ROMM [46] and RLB [57] have good locality, but they fail to achieve optimal worst-case throughput.

To the best of our knowledge, among the closed-form oblivious routing algorithms that can guarantee optimal worst-case throughput for torus networks, an

improved Valiant routing algorithm called IVAL [65] achieves the lowest average hop count. Like two-phase Valiant routing, IVAL load-balances packets to a randomly chosen intermediate node, but reverses the order of traversal of dimensions between the two routing phases (e.g., XY routing, followed by YX routing). In doing so, loops are often formed, and IVAL improves over Valiant routing by removing such loops at runtime.

In this chapter, we introduce a new closed-form oblivious routing algorithm called W2TURN that achieves optimal worst-case throughput for 2D torus networks. W2TURN is based on a weighted random selection of paths with at most two turns. The restriction imposed on the number of allowed turns results in a simple deadlock-free implementation. In comparison to IVAL, W2TURN achieves lower average hop count and higher average-case throughput. We also present another weighted random routing algorithm based on selecting paths with at most two turns, called I2TURN. We show that I2TURN is in fact equivalent to IVAL in the sense that packets are routed over the same set of paths with the same probabilities. However, I2TURN eliminates the need for loop removal at runtime and provides a closed-form analytical expression for evaluating the average hop count. The latter enables us to demonstrate analytically that W2TURN does indeed strictly outperform IVAL (and I2TURN) in average hop count.

W2TURN also performs well in comparison to optimization-based solutions. Optimal routing for 2D torus networks has been formulated as a multicommodity flow problem [65], which can be expressed as linear programs. Using this formulation, worst-case throughput optimal routing with minimum average hop count can be computed. However, it is difficult to guarantee deadlock-free operation for this approach since the resulting solution may include arbitrary paths with arbitrary number of turns. Motivated in part by this difficulty, Towles et al. proposed a modified formulation called 2TURN that guarantees optimal routing when the choice of routing paths is restricted to those with at most two turns. As noted in [65], the key advantage of 2TURN over the optimal solution is the fact that its paths can be described in simple terms, allowing for a simple deadlock-free implementation. However, like the optimal solution, 2TURN does not have a closed-form description, thus requiring a separate linear program for each instance of network size. These linear programs grow quickly, making them difficult to

scale to large networks.[1] When the network radix is odd, W2TURN achieves the same average hop count as optimal-2TURN, but this optimal result is achieved with a closed-form algorithm without the issues mentioned above. When the network radix is even, W2TURN comes very close to optimal-2TURN in terms of average hop count, within just 0.72% of optimal-2TURN on a 12×12 torus.

We also present a new weighted random oblivious routing algorithm for one-dimensional rings called WRD (Weighted Random Direction). WRD offers both optimal worst-case throughput and the minimum average hop count achievable while remaining worst-case throughput optimal for ring networks. RLB routing [57] on rings achieves these optimality conditions when the network radix is odd, but we are unaware of any previous oblivious routing algorithms that can achieve the same for even-radix ring topologies.

Finally, we present detailed evaluations comparing the performance of WRD and W2TURN with the best previously known worst-case throughput optimal routing algorithms with closed-form descriptions for ring and torus topologies, respectively. In this regard, we compare W2TURN with IVAL/I2TURN in terms of average hop count, average-case throughput and throughputs under a wide range of benign and adversarial traffic patterns. Similarly, we compare WRD with RLB over the same set of performance metrics. We observe that W2TURN can achieve up to 13.4% reduction in hop count and a similar increase in throughput over I2TURN under uniform random traffic. WRD can significantly reduce average hop count over RLB by up to 25% when the network radix is even. By means of cycle-accurate flit-level simulations, we also demonstrate that the reduction in hop count together with the improvement in saturation throughput achieved by W2TURN and WRD can translate into significant latency reductions under moderate to high network loads over a wide range of traffic patterns.

The rest of this chapter is organized as follows: Section 3.2.1 provides a brief background on the torus topology. Section 3.2.2 discusses the techniques used to

[1] The largest 2D torus networks solved in [65] had $k = 11$ and $k = 13$, respectively, for optimal and optimal-2TURN routing, where k is the network radix. Interconnection networks with thousands of nodes are already in use today. Although larger instances may be solved with increasing computing power, the size of interconnection networks continues to grow as well.

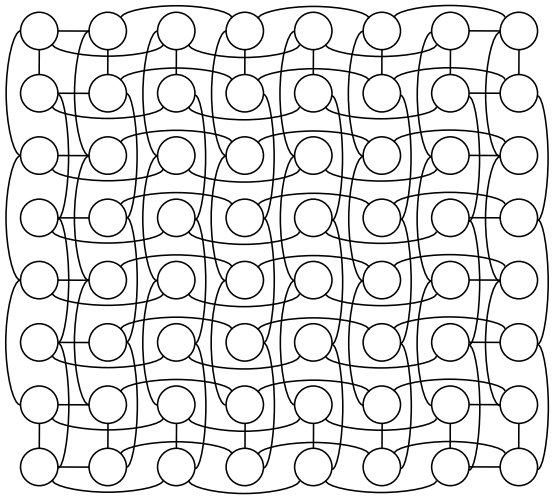


Figure 3.1: Layout of a 8×8 folded torus.

evaluate the performance of a routing algorithm. Section 3.3 then presents our optimal routing algorithm, WRD, for the case of rings. Section 3.4 describes the I2TURN routing algorithm and shows its equivalence to IVAL. Section 3.5 describes W2TURN for the case of 2D torus networks. Finally, Section 3.6 evaluates the performance of WRD and W2TURN and Section 3.7 concludes the chapter.

3.2 Background

In this section, we first provide a brief overview of the torus topology. We follow it up with some preliminaries about computing the worst-case and average-case throughput of routing algorithms.

3.2.1 Torus networks: A candidate on-chip network topology

Torus networks can be described as k-ary n-cubes, where k is the number of nodes along each dimension and n is the number of dimensions. Rings belong to the torus family of network topologies denoted as k-ary 1-cubes and have been often used as the interconnection fabric in commercial multi-core chips [27, 52]. One and two dimensional torus topologies are well suited for on-chip networks as they map well to a planar substrate. A 2D torus has to be physically arranged in a folded form to

equalize wire lengths (as shown in Figure 3.1) and avoid employing long wrap-around links between edge nodes. A torus is a regular topology with symmetric links, which makes it easier to load-balance traffic over all links in the network. This is different from a network with asymmetric links like a 2D mesh, where links at the center of the network are generally more heavily loaded compared to the links at the edge of the network, even under uniform traffic.

3.2.2 Preliminaries

As discussed previously in Section 2.2, the worst-case throughput of a routing algorithm is typically defined relative to the capacity of a network, which is in turn defined by the maximum channel load γ^* that a channel at the bisection of the network needs to sustain under uniform traffic. For any n-dimensional tori with radix k, using the results in [12],

$$\gamma^* = \begin{cases} \dfrac{k}{8} & k \text{ is even} \\ \dfrac{k}{8} - \dfrac{1}{8k} & k \text{ is odd} \end{cases}$$

The network capacity is the inverse of γ^*.

As shown in [64], the worst-case channel load for a routing algorithm R over all admissible traffic matrices can be found by solving a derived maximum weighted matching problem for each channel in the network. The worst-case saturation throughput for a routing algorithm R is the inverse of the worst-case channel load. Further, using the same notations as Section 2.2, the normalized worst-case saturation throughput, $\Theta_{wc}(R)$, is defined as the worst-case saturation throughput normalized to the network capacity:

$$\Theta_{wc}(R) = \frac{\gamma^*}{\gamma_{wc}(R)} \tag{3.1}$$

Valiant routing (VAL) [66] is known to be worst-case throughput optimal with $\Theta_{wc}(\text{VAL}) = 0.5$. Therefore, to show that a routing algorithm $\hat{R}$ is worst-case throughput optimal for a torus network with radix k, it is sufficient to show that the maximum channel load under the worst-case traffic pattern identified using maximum weighted

matching is at most

$$\gamma_{wc}(\hat{R}) = \frac{\gamma^*}{0.5} = \begin{cases} \frac{k}{4} & k \text{ is even} \\ \frac{k}{4} - \frac{1}{4k} & k \text{ is odd} \end{cases} \tag{3.2}$$

which could be demonstrated analytically or empirically over a wide range of network sizes.

In order to show that a routing algorithm $\hat{R}$ provides the minimum average hop count achievable while remaining worst-case throughput optimal, we use the multicommodity flow formulation proposed by Towles et al. [65] to derive worst-case throughput optimal routings with minimum hop count over a range of network sizes. We then compare the average hop counts of $\hat{R}$ with those of the optimal routing solutions over the same range of network sizes.

Finally, along with worst-case throughput, average-case throughput is also an important performance metric for routing algorithms. Using the methodology used in [53, 64], the average-case throughput of a routing algorithm R can be computed by averaging the throughput over T, a large set of random traffic patterns:

$$\Theta_{avg}(R) = \frac{1}{|T|} \sum_{\Lambda \in T} \left(\frac{\gamma(R,\Lambda)}{\gamma^*} \right)^{-1} \tag{3.3}$$

3.3 Optimal routing on rings with WRD

In this section, we consider the optimal oblivious routing problem for one-dimensional rings. Our proposed algorithm called WRD works as follows. Suppose source s sends traffic to destination d, the minimal distance around the loop is given as:

$$\Delta(s,d) = \min(|s-d|, k-|s-d|) \tag{3.4}$$

where k is the number of nodes in the ring. When there is no confusion, we will simply refer to $\Delta(s,d)$ as Δ. We consider two cases: first when k is odd, second when k is even.

For odd k, WRD routes traffic in the minimal and non-minimal directions with

the following probabilities:

$$P_{odd} = \begin{cases} \dfrac{k-\Delta}{k} & \text{in minimal direction} \\ \\ \dfrac{\Delta}{k} & \text{in non-minimal direction} \end{cases} \tag{3.5}$$

This is precisely what the RLB algorithm [57] does in the case of rings, and this has already been shown to be worst-case throughput optimal. Given the above routing probabilities and the fact that the minimal direction has Δ hops while the non-minimal direction has $k-\Delta$ hops, the average hop count for the odd-radix case can be computed as follows:

$$H_{odd}(\text{WRD}) = E\left[\frac{2\Delta(k-\Delta)}{k}\right] = \frac{k}{3} - \frac{1}{3k} \tag{3.6}$$

where $E[.]$ denotes the expectation operator over all possible destination nodes for a given source.

For even k, WRD routes traffic using the following probabilities when $\Delta > 0$ and $k > 2$:

$$P_{even} = \begin{cases} \dfrac{k-\Delta-1}{k-2} & \text{in minimal direction} \\ \\ \dfrac{\Delta-1}{k-2} & \text{in non-minimal direction} \end{cases} \tag{3.7}$$

When $\Delta = 0$ (i.e., $s = d$), no routing is necessary. When $k = 2$ and $\Delta > 0$, WRD routes in both directions at equal distance with equal probability, which is the same as RLB. Note that the traffic distribution in the minimal and non-minimal directions for the even radix case when $k > 2$ is different from RLB. The average hop count of this route distribution can be computed as follows:

$$\begin{aligned} &H_{even}(\text{WRD}) = \\ &E\left[\left(\frac{\Delta(k-\Delta-1)}{k-2} + \frac{(k-\Delta)(\Delta-1)}{k-2}\right)\middle|\, \Delta > 0\right] P(\Delta > 0) \\ &= \frac{k}{3} \times \left(\frac{k-1}{k}\right) = \frac{k}{3} - \frac{1}{3} \end{aligned}$$

WRD achieves lower average hop count than RLB when the network radix k is even and when $k > 2$ since

$$\frac{k}{3} - \frac{1}{3} < \frac{k}{3} - \frac{1}{3k} \qquad \forall k > 1$$

We next show that WRD indeed achieves optimal worst-case throughput for all network radices.

Claim 7. *WRD is worst-case throughput optimal.*

Proof. For the odd-radix case, WRD is the same as RLB, which has already been shown to be worst-case throughput optimal in [55]. For the even-radix case, we use the same proof methodology that was used in [55] for showing RLB is worst-case throughput optimal on a ring. The proof uses the method in [64] to identify a worst-case traffic pattern for WRD. We then verify that the maximum channel load using WRD on this worst-case traffic pattern is indeed at most $k/4$, as shown necessary and sufficient for worst-case throughput optimality for even k in Equation 3.2 of Section 3.2. Using the technique described in [64], a worst-case traffic pattern for WRD is Tornado traffic [12]. Suppose under the tornado traffic pattern each node sends all its traffic to a node $k/2-1$ hops away in the clockwise direction ($\Delta = k/2-1$), the corresponding load on every clockwise channel is given by the sum of the contributions from the $k/2-1$ nodes preceding the channel. Using the probability for routing in the minimal direction from Equation 3.7, each of these $k/2-1$ preceding nodes route in the clockwise (minimal) direction with a probability of $k/(2(k-2))$. Therefore, the maximum channel load on a clockwise channel, $\gamma_{clk}(\text{WRD})$, is given as:

$$\gamma_{clk}(\text{WRD}) = \frac{k/2}{k-2} \times \left(\frac{k}{2}-1\right) = \frac{k}{4} \;\; \forall k > 2$$

Similarly, the maximum channel load on a counter-clockwise channel can be computed as the sum of the contributions from $(k/2+1)$ nodes preceding the channel. Using the probability of routing in the non-minimal direction from Equation 3.7, each of these nodes contribute $(k-4)/(2(k-2))$ of their traffic, and the sum of their contributions, $\gamma_{cclk}(\text{WRD})$, is given as:

$$\gamma_{cclk}(\text{WRD}) = \frac{(k+2)(k-4)}{4(k-2)} < \frac{k}{4} \;\; \forall k > 2$$

The worst-case channel load with WRD for even k is then given as:

$$\gamma_{wc}(\text{WRD}) = max(\gamma_{clk}(\text{WRD}), \gamma_{cclk}(\text{WRD})) = \frac{k}{4} \;\; \forall k > 2$$

Hence, WRD is worst-case throughput optimal. □

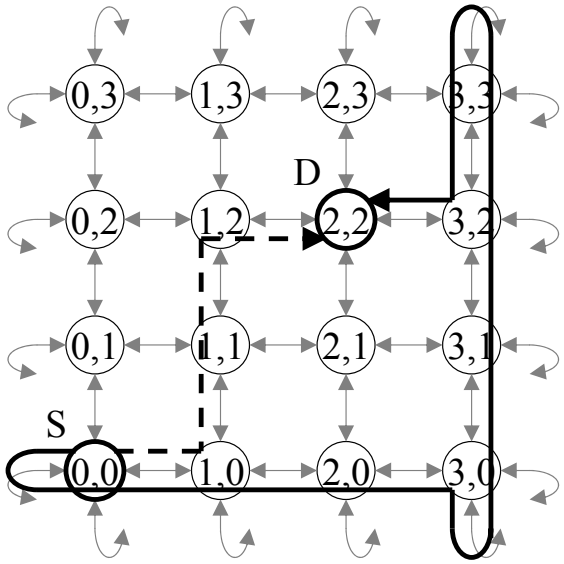


Figure 3.2: Routing with 2-turn paths.

Claim 8. *WRD achieves the minimum average hop count achievable while remaining worst-case throughput optimal.*

Proof. Using the methodology discussed in Section 3.2, we have verified this claim by comparing the average hop counts of WRD with those of optimal routing, which were computed using a multicommodity flow formulation [65]. □

3.4 The I2TURN routing algorithm

In this section, we describe the I2TURN routing algorithm for two-dimensional torus networks. As the name suggests, I2TURN considers routing paths with at most two turns, as shown in Figure 3.2. The dashed line shows an XYX 2-turn path that starts from $(0,0)$, makes the first turn at $(1,0)$, makes the second turn at $(1,2)$, and goes finally to $(2,2)$. The solid line shows an alternative XYX 2-turn path that starts from $(0,0)$, loops left and around to first turn at $(3,0)$, loops down and around to $(3,2)$, and goes finally to $(2,2)$.

The idea of using 2-turn paths was proposed in [65] in their optimal 2TURN algorithm. However, the proposed 2TURN algorithm does not have a closed-form algorithmic description. It only has a closed-form description of the possible paths that

a packet may take through the network, but requires solving a separate linear program to determine the path distribution for each given network radix. The size of these linear programs grow quickly making them difficult to scale to large networks. The I2TURN algorithm and the W2TURN algorithm, which is described next, have closed-form descriptions and can be easily extended to arbitrarily large networks.

We first consider a version of I2TURN that only uses XYX 2-turn paths. Suppose (x_1, y_1) is the source and (x_2, y_2) is the destination. The three segments of the XYX 2-turn paths are generated as follows:

1. X-segment: Choose at uniform random an X position $x^* \in [0, k-1]$ and route in the X dimension from (x_1, y_1) to (x^*, y_1) in the minimal direction.

2. Y-segment: Next, route in the Y dimension from (x^*, y_1) to (x^*, y_2) in the minimal and non-minimal directions with the following probabilities:

$$P = \begin{cases} \dfrac{k-\Delta y}{k} & \text{in minimal direction} \\ \\ \dfrac{\Delta y}{k} & \text{in non-minimal direction} \end{cases}$$

   Here Δy is the minimum distance in Y from (x^*, y_1) to (x^*, y_2) computed using Equation 3.4. The routing along the Y dimension is identical to RLB and WRD for the odd-ring case (Equation 3.5).

3. X-segment: Finally, route in the X dimension from (x^*, y_2) to (x_2, y_2) in the minimal direction.

There are several degenerate cases. When $x^* = x_1$, there is no need to route on the first X-segment. Similarly, when $x^* = x_2$, then there is no need to route on the last X-segment. When $y_1 = y_2$, packets only need to be routed along the X dimension with no turns and any loop formed as a result of two-phase minimal routing on the X ring can be removed. In this case, the packet is routed with probability $(k - \Delta x)/k$ in the minimal direction, where Δx is the minimum distance in X between x_1 and x_2, and with probability $(\Delta x)/k$ in the non-minimal direction. Finally, when the source and destination are the same, no routing is necessary. Unless otherwise noted, we will regard these “degenerate" cases as “2-turn" paths as well.

3.4.1 Equivalence to IVAL

Claim 9. *I2TURN routes packets using the same statistical distribution of paths as IVAL.*

Proof. IVAL routes packets from the source to the destination via a random intermediate node, using minimal XY and YX routing in the two phases. IVAL identifies and removes any loop formed at runtime, resulting in the construction of loop-free 2-turn XYX paths. To show that IVAL and I2TURN route packets along the same paths with the same probabilities, we consider three cases. Suppose (x_1, y_1) is the source and (x_2, y_2) is the destination. In the first case, when the source and destination are the same, no routing occurs in both IVAL and I2TURN.

In the second case, when $y_1 \neq y_2$, I2TURN chooses at uniform random the intermediate X position x^*. The routing in the X dimension from x_1 to x^* and x^* to x_2 are each unique in the corresponding minimal directions. For the Y segment, the packet is routed in either the minimal or non-minimal Y direction. In IVAL, there are k^2 possible intermediate nodes (x_i, y_i) that can be chosen at uniform random. It follows that the probability of choosing an intermediate node with $x_i = x^*$ is uniformly $1/k$. As in I2TURN, the routing for IVAL in the X dimension from x_1 to x^* and x^* to x_2 are in the same corresponding minimal directions. Since IVAL paths are loop-free after runtime loop removal, we are guaranteed that the path will be a 2-turn XYX path where the packet will be routed in the Y dimension at X position x^* in either the minimal or non-minimal direction. Since routing in the X dimension is equivalent, we can reduce the proof to equivalent path selection on the Y ring.

For I2TURN, there are two possible acyclic paths on the Y ring – a minimal path in the short direction with a distance of Δy that is chosen with probability $(k - \Delta y)/k$, and a non-minimal path in the long direction with a distance of $(k - \Delta y)$ that is chosen with probability $\Delta y/k$. For IVAL, any of the k nodes on the Y ring can be chosen as the intermediate node. Since the definition of minimal and non-minimal paths is relative to y_1 and y_2, it suffices to consider the case where $y_1 = 0$ and $y_2 = \Delta y$, effectively shifting the origin to the coordinates of the source. By definition of minimal distance, $\Delta y \leq k/2$.

In the subsequent discussion, minimal and non-minimal directions (paths) refer to the short and long paths, respectively, between the source and the destination. Let i be the Y coordinate of the intermediate node chosen by IVAL. There are two situations

when a packet is guaranteed to be routed along the minimal path after loop removal: when $0 \leq i < k/2$ or when $(i - \Delta y) > k/2$. There are $\lceil k/2 \rceil$ possible intermediate nodes that satisfy $0 \leq i < k/2$, and there are $\lceil k/2 \rceil - \Delta y - 1$ possible intermediate nodes that satisfy $(i - \Delta y) > k/2$, with a combined total of $\lceil k/2 \rceil + \lceil k/2 \rceil - \Delta y - 1$ intermediate nodes that will always result in IVAL routing along the minimal path after loop removal. When k is even and $i = k/2$, the distance between $y_1 = 0$ and i will be the same in both directions, giving it a 50% chance that a packet will be routed in the minimal direction after loop removal. Similarly, when k is even and $i - \Delta y = k/2$, the distance between i and $y_2 = \Delta y$ will be the same in both directions, again giving it a 50% chance that a packet will be routed in the minimal direction following loop removal. Assuming all intermediate nodes are chosen with equal probability, the total probability of choosing the minimal path along the Y ring in IVAL is given by $(\lceil k/2 \rceil + \lceil k/2 \rceil - \Delta y - 1)/k$ when k is odd and $(\lceil k/2 \rceil + \lceil k/2 \rceil - \Delta y)/k$ when k is even. When k is odd, $\lceil k/2 \rceil + \lceil k/2 \rceil = k + 1$. When k is even, $\lceil k/2 \rceil + \lceil k/2 \rceil = k$. Therefore, the probability of choosing the minimal path is equal to $(k - \Delta y)/k$ when k is either even or odd.

Finally, in the third case, when $y_1 = y_2$, but $x_1 \neq x_2$, the proof reduces to showing that IVAL will choose the same loop-free path on the X dimension as I2TURN. The same analysis presented above for the Y ring can be applied to the X dimension to show that both IVAL and I2TURN will choose the minimal (and non-minimal) paths with the same probability. □

Claim 10. *I2TURN is worst-case throughput optimal.*

Proof. The proof follows from its equivalence to IVAL. □

The above discussion applies to I2TURN with XYX 2-turn paths. I2TURN can be equivalently defined using YXY 2-turn paths by swapping dimensions. Also, I2TURN can be implemented using a randomization of XYX and YXY paths. When XYX and YXY routings are used with equal probability, I2TURN routing is symmetric and balances load equally between the X and Y channels.

3.4.2 Average hop count of I2TURN

The I2TURN routing algorithm for torus networks is described in terms of weighted random selection of 2-turn paths. Although I2TURN is equivalent to IVAL, its description of 2-turn paths based on probabilities makes it easier to derive an analytical expression for its average hop count. The average hop count for I2TURN can be expressed as the sum of the average number of hops for the three routing segments of the 2-turn paths: minimal routing on the first and last X segments, and weighted random routing on the middle Y segment. Let H_{min} denote the average hop count for minimal routing on a ring [12].

$$H_{min} = \begin{cases} \dfrac{k}{4} & k \text{ is even} \\ \dfrac{k}{4} - \dfrac{1}{4k} & k \text{ is odd} \end{cases}$$

Since the routing on the middle Y segment is same as the WRD algorithm for the odd-ring case, the average hop count for this segment can be computed using Equation 3.6. As analyzed in the proof of Claim 9, we have to consider 1-in-k cases when the source and destination have the same Y coordinate. For these cases, loops formed on the X ring can be removed and the routing along the X ring becomes identical to the weighted random routing used in the Y dimension. Taken together, the average hop count for I2TURN is presented as follows:

$$\begin{aligned} H_{avg}(\text{I2TURN}) &= H_x(\text{I2TURN}) + H_y(\text{I2TURN}) \\ H_x(\text{I2TURN}) &= \left(1 - \frac{1}{k}\right) 2H_{min} + \left(\frac{1}{k}\right)\left(\frac{k}{3} - \frac{1}{3k}\right) \\ H_y(\text{I2TURN}) &= \frac{k}{3} - \frac{1}{3k} \\ H_{avg}(\text{I2TURN}) &= 2\left(1 - \frac{1}{k}\right) H_{min} + \left(1 + \frac{1}{k}\right)\left(\frac{k}{3} - \frac{1}{3k}\right) \end{aligned}$$

It must be noted here that I2TURN routing described using YXY routing paths or a randomization of XYX and YXY routings will have the same average hop count as computed above.

3.5 The W2TURN routing algorithm

Next, we describe the W2TURN routing algorithm for 2D-torus networks. Like I2TURN, W2TURN also considers different routing paths with at most two turns, as shown in Figure 3.2. However, the probabilities with which the 2TURN paths are chosen are different, giving W2TURN an edge over I2TURN in terms of average hop-count. The W2TURN algorithm was developed in part from examining the path distribution derived out of the optimal 2TURN formulation. W2TURN was also based on the intuition gained from studying optimal routing for the 1D ring case (WRD) and the I2TURN algorithm.

In the remainder of this section, we present the W2TURN routing algorithm and analyze its worst-case throughput and average hop count. Like WRD, we consider the odd-k and even-k cases separately.

3.5.1 When k is odd

We first describe the weighted random selection of XYX routing paths in W2TURN. $\Delta(x_1,x_2)$ refers to the minimum distance on the X-ring between nodes having X-coordinates x_1 and x_2 and the same Y-coordinate. $\Delta(y_1,y_2)$ refers to the minimum distance on the Y-ring between nodes having Y-coordinates y_1 and y_2 and the same X-coordinate. The definition of minimum distance along a dimension follows from Equation 3.4.

Suppose (x_1,y_1) is the source and (x_2,y_2) is the destination, the three segments of the XYX 2-turn paths are generated as follows:

1. X-segment: Choose at uniform random an X position $x^* \in [0,k-1]$. Then, consider two cases:

   (a) Route in the minimal direction from (x_1,y_1) to (x^*,y_1) if *any* of the following conditions are satisfied:

   - $\Delta(x_1,x^*) < \lfloor \frac{k}{2} \rfloor$,
   - (x_2,y_1) is not on the minimal path from (x_1,y_1) to (x^*,y_1), or
   - $\Delta(x_1,x_2) = \lfloor \frac{k}{2} \rfloor$.

(b) Otherwise, route from (x_1, y_1) to (x^*, y_1) on the X ring with the following probabilities:

$$P = \begin{cases} \frac{k - \Delta(x_1, x_2)}{k} & \text{in minimal direction} \\ \frac{\Delta(x_1, x_2)}{k} & \text{in non-minimal direction} \end{cases}$$

where minimal and non-minimal directions refer to the short and long paths, respectively, on the X ring from (x_1, y_1) to (x^*, y_1).

2. Y-segment: Next, route in the Y dimension from (x^*, y_1) to (x^*, y_2). We again consider two cases:

(a) Route in the minimal direction from (x^*, y_1) to (x^*, y_2) if *all* of the following conditions are satisfied:

- $x_1 \neq x_2$,
- $\Delta(y_1, y_2) < \lfloor \frac{k}{2} \rfloor$, and
- $(x^* = x_1$ or $x^* = x_2)$.

(b) Otherwise, route from (x^*, y_1) to (x^*, y_2) on the Y ring using WRD.

3. X-segment: Finally, route in the X dimension from (x^*, y_2) to (x_2, y_2), again with two cases:

(a) Route in the minimal direction from (x^*, y_2) to (x_2, y_2) if *any* of the following conditions are satisfied:

- $\Delta(x^*, x_2) < \lfloor \frac{k}{2} \rfloor$,
- (x_1, y_2) is not on the minimal path from (x^*, y_2) to (x_2, y_2), or
- $\Delta(x_1, x_2) = \lfloor \frac{k}{2} \rfloor$.

(b) Otherwise, route from (x^*, y_2) to (x_2, y_2) on the X ring with the following probabilities:

$$P = \begin{cases} \frac{k - \Delta(x_1, x_2)}{k} & \text{in minimal direction} \\ \frac{\Delta(x_1, x_2)}{k} & \text{in non-minimal direction} \end{cases}$$

where minimal and non-minimal directions refer to the short and long paths, respectively, on the X ring from (x^*, y_2) to (x_2, y_2).

There are several degenerate cases. When $x^* = x_1$, there is no need to route on the first X-segment. Similarly, when $x^* = x_2$, there is no need to route on the last X-segment. When $y_1 = y_2$, packets only need to be routed along the X dimension with no turns using WRD. Finally, when the source and destination are the same, no routing is necessary.

Given the above XYX routing algorithm, the version with YXY routing can be equivalently defined by swapping dimensions. To achieve worst-case throughput optimality for the odd k case, W2TURN requires using both XYX and YXY routing with equal probabilities.

3.5.2 When k is even

For the even-radix case, we first describe the weighted random selection of XYX routing paths. Suppose (x_1, y_1) is the source and (x_2, y_2) is the destination, the three segments of the XYX 2-turn paths are generated as follows:

1. X-segment: First, choose at uniform random an X position $x^* \in [0, k-1]$. Then route minimally from (x_1, y_1) to (x^*, y_1). If the number of hops in both directions are equal, choose the direction that does not contain the node (x_2, y_1). If $x^* = x_2$, and the number of hops in both directions are equal, choose either direction with equal probability.

2. Y-segment: Route from (x^*, y_1) to (x^*, y_2) using WRD.

3. X-segment: Route minimally from (x^*, y_2) to (x_2, y_2). If the number of hops in both directions are equal, choose the direction that does not contain the node (x_1, y_2). If $x^* = x_1$, and the number of hops in both directions are equal, choose either direction with equal probability.

There are several degenerate cases. When $x^* = x_1$, there is no need to route on the first X-segment. Similarly, when $x^* = x_2$, then there is no need to route on the last X-segment. When $y_1 = y_2$, the packet only needs to be routed along the X dimension using the same algorithm described above. For this case, any loop formed as a result of an overlap in the routing paths of the two X segments should be removed. Following

loop removal, the probabilities of routing in the minimal and non-minimal directions are given as follows when $\Delta(x_1,x_2) < k/2$ and $\Delta(x_1,x_2) > 0$:

$$P = \begin{cases} \dfrac{k-\Delta(x_1,x_2)-1}{k} & \text{in minimal direction} \\ \dfrac{\Delta(x_1,x_2)+1}{k} & \text{in non-minimal direction} \end{cases}$$

When $\Delta(x_1,x_2) = k/2$ a packet is routed in either direction with equal probability. Finally, when the source and destination are the same, no routing is necessary.

The YXY routing paths can be equivalently defined by swapping dimensions. To achieve worst-case throughput optimality for the even k case, W2TURN requires interpolating over the following four routings with the corresponding specified probabilities:

- XYX routing with probability $\frac{k}{2(k+1)}$
- YXY routing with probability $\frac{k}{2(k+1)}$
- Dimension-ordered XY routing with probability $\frac{1}{2(k+1)}$
- Dimension-ordered YX routing with probability $\frac{1}{2(k+1)}$

3.5.3 Throughput optimality

In this section, we show that W2TURN is indeed worst-case throughput optimal.

Claim 11. *W2TURN is worst-case throughput optimal.*

Proof. We again use the same proof methodology that was used in [55], which uses the method in [64] for identifying a worst-case traffic pattern. We then show that the maximum channel load using W2TURN on this worst-case traffic pattern is indeed at most $k/4$ when k is even and at most $(k/4 - 1/(4k))$ when k is odd, as shown necessary and sufficient in Equation 3.2. For a network with radix k, a worst-case traffic pattern for W2TURN is shown as follows:

$$\text{Node}(x,y) \text{ sends packets to } (x+\lfloor k/2 \rfloor, y+\lfloor k/2 \rfloor)$$

The above traffic pattern is same as Tornado traffic [12] when k is odd. Using worst-case load analysis, the maximum channel load for the worst-case traffic pattern was found to be the same as Equation 3.2 for all values of k analyzed[2]. □

3.5.4 Latency analysis

Next, we express the average hop count of W2TURN in terms of the average hop count expressions derived for WRD and the network radix k. Later, in Section 3.6 we show that W2TURN indeed outperforms I2TURN in average hop count. We treat the even and odd k cases separately.

Odd k

The average hop count of XYX routing is presented in this section. YXY routing will have identical hop count due to symmetry. The average hop count for XYX routing is given by the sum of the average hop counts of the two X segments and the Y segment. We first consider the average hop count for the X segments. Suppose (x_1, y_1) is the source and (x_2, y_2) is the destination, when $y_1 = y_2$, which is one of the degenerate cases, WRD is used to route on the X ring. The average hop count for this case is equal to the average hop count of WRD with odd radix.

$$H_{case1} = \frac{k}{3} - \frac{1}{3k}$$

When $y_1 \neq y_2$, minimal routing is used on both X segments if either of the three conditions stated in Section 3.5.1 are satisfied. In this case, we first compute the probability of routing non-minimally in the X dimension (when all three conditions are not satisfied) and multiply it by the extra hops added (over minimal) as a result of non-minimal routing. We denote this penalty paid over minimal routing by routing non-minimally as $H_{penalty}$.

$$H_{penalty} = \frac{2}{k}\left[\frac{1}{k}\sum_{\Delta(x_1,x_2)=0}^{\lfloor \frac{k}{2} \rfloor - 1} \frac{\Delta(x_1,x_2)}{k}\right]$$

[2]Maximum channel load was verified for k up to 40.

The latency of each X segment is then given as:

$$H_{case2} = H_{min} + H_{penalty}$$

H_{case1} denotes the combined latency of the two X segments when $y_1 = y_2$ and H_{case2} denotes the average latency of each X segment when $y_1 \neq y_2$. Hence, the average latency of the two X segments can be expressed as follows:

$$H_x(\text{XYX}) = \frac{1}{k} H_{case1} + \frac{(k-1)}{k}(2 \times H_{case2}) \tag{3.8}$$

Next, we consider the average hop count of the Y segment. In the Y dimension, a packet is routed minimally if all the conditions stated in Section 3.5.1 are satisfied. Else, it is routed using WRD. The probability that the first condition is true, i.e. $x_1 \neq x_2$ is $(k-1)/k$ and the probability that the third condition is true, i.e. $x^* = x_1$ or $x^* = x_2$, given $x_1 \neq x_2$ is $2/k$. Using these results, the average hop count savings by routing minimally in the Y dimension instead of using WRD when all three conditions are true is given as follows:

$$H_{savings} = \frac{2}{k}\frac{(k-1)}{k}\left[\frac{2}{k}\sum_{\Delta(y_1,y_2)=0}^{\lfloor \frac{k}{2} \rfloor - 1} \frac{\Delta(y_1,y_2)}{k}(k - 2\Delta(y_1,y_2))\right]$$

The average latency in the Y dimension can then be expressed as:

$$\begin{aligned} H_y(\text{XYX}) &= H_{odd}(\text{WRD}) - H_{savings} \\ &= \frac{k}{3} - \frac{1}{3k} - H_{savings} \end{aligned} \tag{3.9}$$

From Equations 3.8 and 3.9 and using the fact that YXY routing will have the same average hop count as XYX routing,

$$H_{odd}(\text{W2TURN}) = H_x(\text{XYX}) + H_y(\text{XYX})$$

Even k

W2TURN routing for an even network radix is an interpolation of four different routings - XYX, YXY, XY and YX. We first consider the average hop count for the XYX

routing paths. When $y_1 = y_2$, no routing is necessary along the Y dimension and there is a possibility of loop removal on the X ring after two phases of X routing. Following loop removal, using the probabilities described in Section 3.5.2 the combined average hop count of the two X segments is given as follows:

$$H_{case1} = \frac{1}{2} + \frac{k}{3} - \frac{4}{3k}$$

For the case when $y_1 \neq y_2$, a packet is routed in the minimal direction on both the X segments. Hence, the average hop count for each X segment in this case is given as:

$$H_{case2} = H_{min} = \frac{k}{4}$$

Therefore, the average hop count for the two X segments of the XYX routing paths can be expressed as:

$$H_x(\text{XYX}) = \frac{1}{k} H_{case1} + \frac{(k-1)}{k}(2 \times H_{case2}) \tag{3.10}$$

Since WRD is used along the Y dimension, the average hop count along this dimension is given as:

$$H_y(\text{XYX}) = H_{even}(\text{WRD}) = \frac{(k-1)}{3} \tag{3.11}$$

From Equations 3.10 and 3.11,

$$H(\text{XYX}) = H_x(\text{XYX}) + H_y(\text{XYX})$$

The hop count for YXY routing is identical to XYX routing due to symmetry. The average hop counts for minimal XY and YX routings are given as:

$$H(\text{XY}) = H(\text{YX}) = \frac{k}{2}$$

The average hop count of W2TURN when the network radix is even, $H_{even}(\text{W2TURN})$, is given by the weighted mean of the average hop counts of XYX, YXY, XY and YX routings with weights $k/2(k+1)$, $k/2(k+1)$, $1/2(k+1)$, and $1/2(k+1)$, respectively.

3.5.5 Deadlock-free implementation

W2TURN uses the same set of 2-turn paths as the optimal 2TURN formulation proposed in [65]. When k is odd, W2TURN distributes traffic over these 2-turn paths

with the same probabilities as optimal 2TURN. When k is even, W2TURN also uses the same set of 2-turn paths, although with different probabilities. Since W2TURN uses the same set of 2-turn paths as optimal-2TURN, a deadlock-free implementation requires exactly the same number of virtual channels, which has been shown to be four virtual channels per physical channel (the same requirement for VAL [66]). In particular, W2TURN can be made deadlock-free by incrementing a packet's virtual channel set after each turn from the Y to the X dimension. Since any 2-turn path has at most one turn from Y to X, this approach requires two virtual channel sets. Each set requires two virtual channels to resolve intra-dimension deadlocks, therefore requiring four virtual channels per physical channel in total. As stated earlier, the key advantage of W2TURN over optimal-2TURN is that W2TURN provides a closed-form algorithm that can achieve comparable performance with the same simple deadlock-free implementation. Further, the W2TURN algorithm only involves simple conditional checks and probability calculations that can be readily implemented in parallel.

3.6 Performance Evaluation

In this section, we evaluate the performance of WRD and W2TURN in terms of latency and throughput. Both WRD and W2TURN have already been shown to be worst-case throughput optimal for one-dimensional and two-dimensional torus topologies, respectively. This section focuses on other throughput metrics like average-case throughput and throughputs under different benign and adversarial traffic patterns.

3.6.1 Evaluation of WRD

Hop count analysis

Figure 3.3 compares the average hop count of WRD with RLB [57] and optimal routing [65]. All three routing algorithms considered achieve optimal worst-case throughput for ring networks. WRD and RLB have closed-form algorithmic descriptions, while the optimal routing results were obtained using the multicommodity flow formulation proposed in [65]. The average hop counts are normalized to minimal

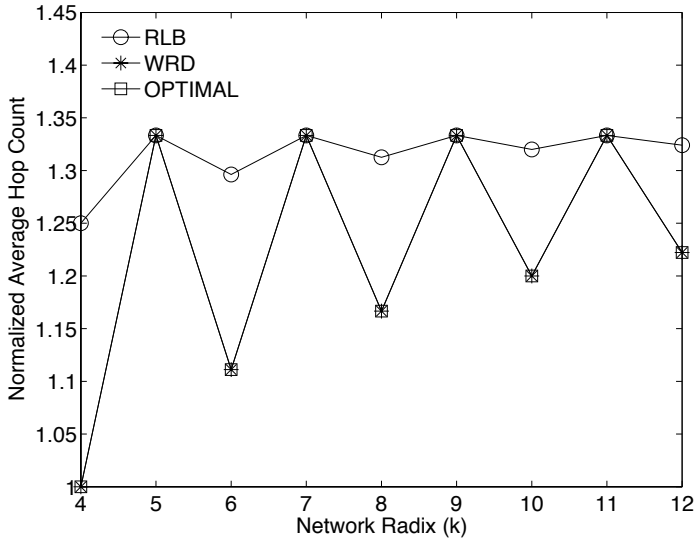


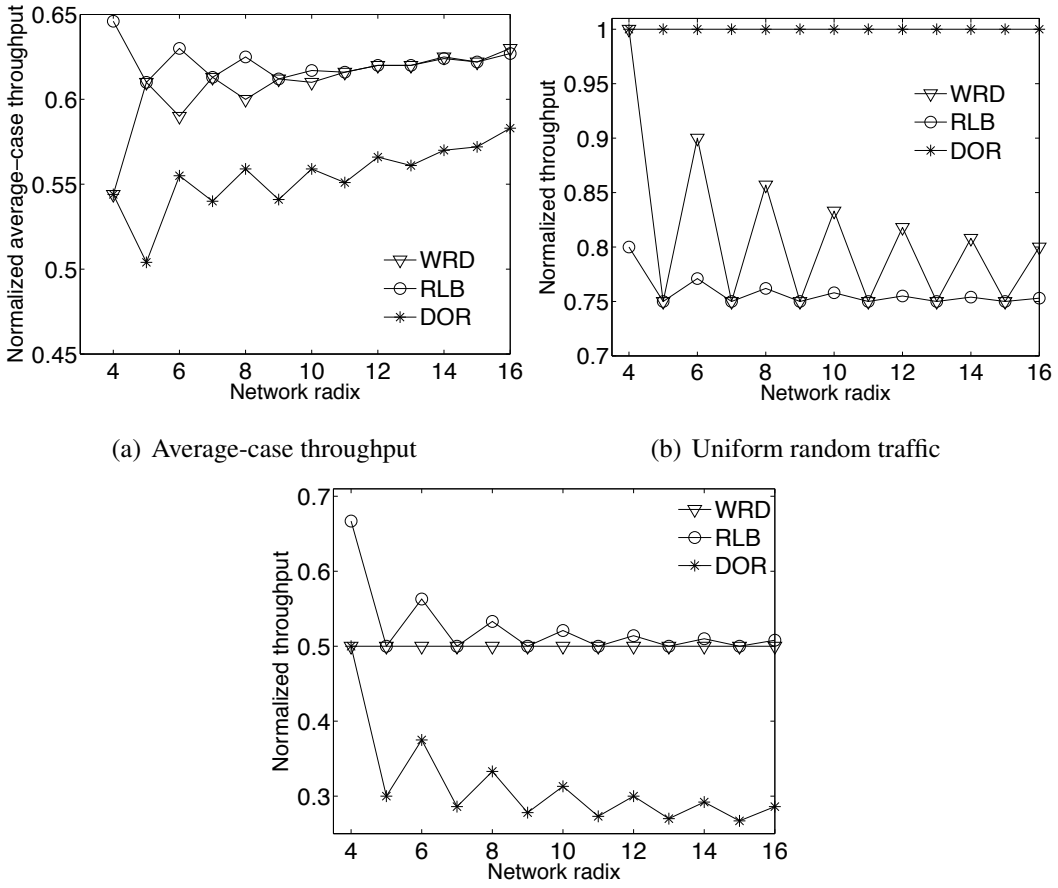
Figure 3.3: Comparison of the average hop counts of WRD with RLB [57] and optimal routing [65] on ring networks with different radices.

routing. As shown in Figure 3.3, WRD achieves the same hop-count as optimal routing for all network radices.

When k is odd, WRD and RLB are equivalent and therefore have the same hop count, as shown in Figure 3.3. When k is even, WRD outperforms RLB because WRD routes in the minimal direction more often. In this case, WRD can achieve up to 25% reduction in average-hop count over RLB.

Throughput evaluation

WRD is optimal in terms of worst-case throughput, but is not minimal in terms of latency as it employs non-minimal routing paths. Here, we compare the throughput of WRD with two other routing algorithms for rings with closed-form descriptions, namely, RLB and DOR. RLB also achieves the same optimal worst-case throughput as WRD but has a higher average hop count when the network radix is even. DOR, on the other hand, achieves minimal hop count while sacrificing worst-case throughput. The throughput metrics used in this section are average-case throughput and throughput under uniform random traffic and tornado traffic [12]. Uniform random traffic is a benign traffic pattern which is easy to route since it is inherently load-balanced. In contrast, tornado traffic



(a) Average-case throughput

(b) Uniform random traffic

(c) Tornado traffic

Figure 3.4: Throughput evaluation of WRD for different network sizes.

is an adversarial pattern, which is also a worst-case traffic pattern for both DOR and WRD.

The throughput analysis is carried out in two steps. Initially, we perform a simplified throughput analysis for a range of network sizes from 4 nodes to 16 nodes. This analysis assumes ideal single-cycle routers with infinite buffers. In the next section, we back these results with more realistic flit-level simulations for an 8-node ring topology. All throughput results presented subsequently are normalized to the network capacity (refer Section 3.2).

Figure 3.4(a) compares the average-case throughput of WRD with RLB and DOR. As discussed in Section 3.2, average-case throughput of a routing algorithm is computed by averaging the throughput over a large set of randomly generated

permutation traffic patterns.[3] We average the throughput over a set of 10,000 randomly generated permutation traffic matrices. WRD is identical to RLB when the network radix is odd and this is reflected in the average-case throughput results as well. RLB is slightly better than WRD for even radices when the network radix is low. For larger topologies (beyond 10 nodes), the difference in throughputs is negligible and by radix 16, WRD even slightly outperforms RLB. Both WRD and RLB outperform DOR in terms of average-case throughput. On average, over all network radices evaluated, WRD outperforms DOR by around 9.8% in average-case throughput.

As shown in Figure 3.4(b), DOR is the best routing algorithm under uniform random traffic, where traffic from a node is equally distributed to all nodes in the network. Non-minimal routing in WRD and RLB prevent these algorithms from sustaining throughputs as high as DOR when the traffic is uniform. However, for even network radices, the lower average hop count of WRD compared to RLB directly translates into a corresponding gain in throughput. For even radices, the throughput of WRD is 12.3% higher than RLB on average under uniform traffic.

Finally, Figure 3.4(c) proves that the throughput of a minimal routing algorithm like DOR can degrade tremendously under an adversarial traffic pattern like tornado traffic. Therefore guaranteeing a level of worst-case performance using optimal routing algorithms like WRD is important. As discussed in Section 3.3, tornado traffic is a worst-case traffic pattern for WRD, which sustains optimal worst-case throughput of half the network capacity under the tornado traffic pattern. On an average over all radices, this is 64% higher than what DOR can sustain under the same traffic pattern. RLB performs comparably to WRD and its throughput converges to half the network capacity for high radices.

Flit-level simulations

The results obtained using ideal throughput analysis represent upper bounds to the actual achievable throughput because it assumes ideal single-cycle routers with infinite buffers and ignores issues like flow control and contention in switches. Hence,

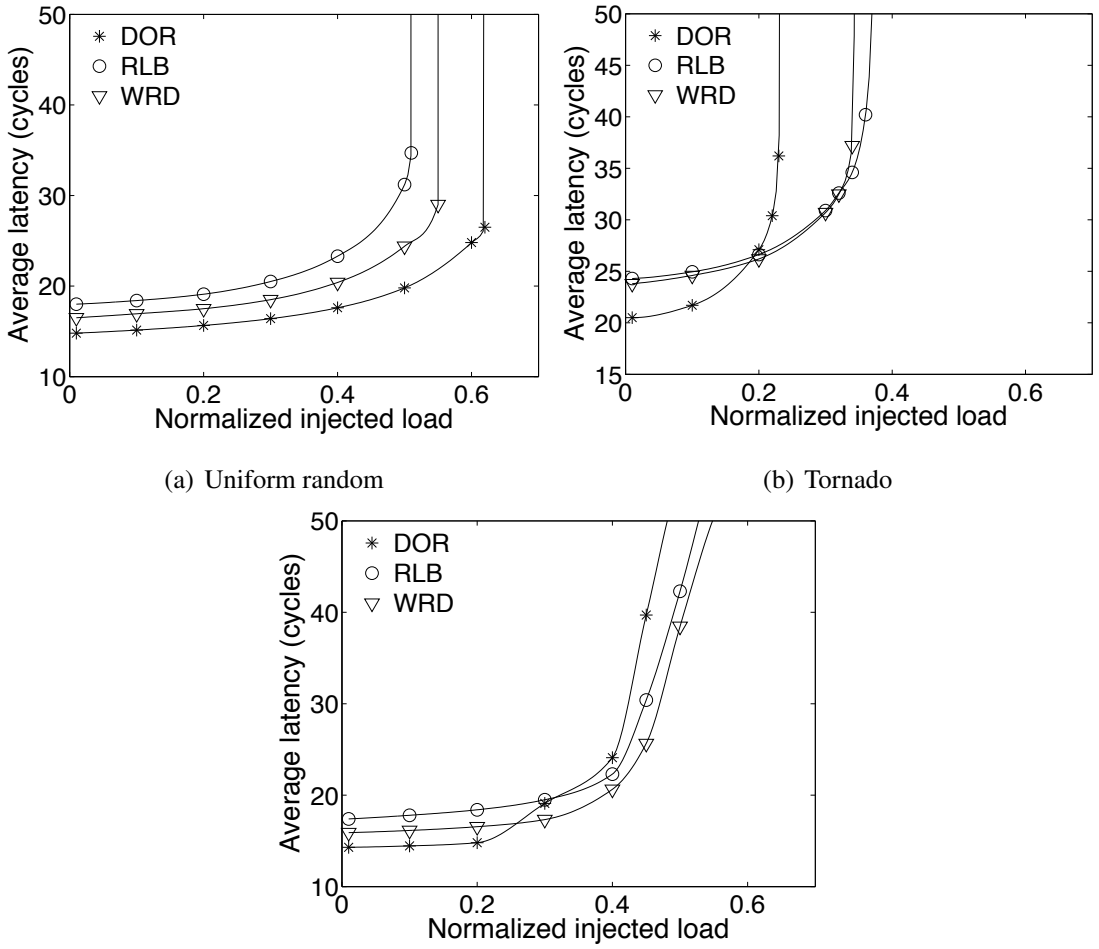
[3] A permutation traffic matrix is one in which a source sends all its traffic to a single destination, obtained from a one-to-one mapping between pairs of nodes in the network [12].

we use cycle-accurate flit-level simulations to gain more realistic insights into the performance of the routing algorithms. We restrict ourselves to an 8-node ring topology. A topology with odd radix is not considered because WRD is identical to RLB for odd radices.

We modified the PopNet [54] on-chip network simulator to perform flit-level simulations. PopNet models a typical input-buffered VC router with five pipelined stages. Route computation is performed in the first stage followed by VC allocation, switch arbitration, switch traversal and link traversal. The head flit of a packet proceeds through all five stages while the body and tail flits bypass the first two stages and inherit the output port and output VC reserved by the head flit. Credit-based flit-level flow control is used between adjacent routers. We assume 8 virtual channels (VCs) per physical channel, each 5 flits deep. For ring topologies, two virtual channels are sufficient to avoid intra-dimension deadlocks. However, it is well known that VCs improve the throughput of any routing algorithm by reducing head-of-line blocking and enabling better statistical multiplexing of flits. So, having a reasonably large number of VCs lets us compare the best performance of all routing algorithms. 3-flit packets are injected into the network and we use PopNet to evaluate the average routing delays under different injection loads. For each simulation, we ran the simulator for 500,000 cycles. The latency of a packet is measured as the delay between the time the head flit is injected into the network and the time the tail flit is consumed at the destination.

Uniform random traffic and tornado traffic are used for comparing WRD with RLB and DOR. In order to capture average-case performance, we generated 250 random permutation traffic patterns and ran the simulation on each pattern for 20,000 cycles. Finally, we report the average delay over the 250 patterns under different injection loads. The maximum latency used for averaging is clipped at 75 cycles in order to keep the impact of a single observation on the computed average within bounds. We refer to this traffic pattern as *dynamic random* traffic as the averaged results correspond to the performance of the routing algorithms under a traffic matrix that dynamically changes with time.

Figure 3.5 presents the flit-level simulation results for the three traffic patterns. The actual throughput obtained is around 60-70% of the throughput predicted using



(a) Uniform random

(b) Tornado

(c) Dynamic random

Figure 3.5: Performance of WRD on a 8-node ring.

ideal analysis, but the saturation throughput trends remain unchanged. As expected from the ideal throughput analysis, DOR outperforms WRD, which in turn outperforms RLB in terms of saturation throughput under uniform random traffic. The latency of WRD under low loads is 8.3% lower than RLB and 11.5% higher than DOR. These numbers are slightly less than the corresponding numbers obtained using hop count analysis, where the hop count of WRD is 11.2% lower than RLB and 16.5% higher than DOR. This is because packet delays in flit-level simulations include the transmission delay of multi-flit packets and the router pipeline delay at the destination, which are ignored while measuring hop count.

For tornado traffic, although DOR has lower latency under very low loads,

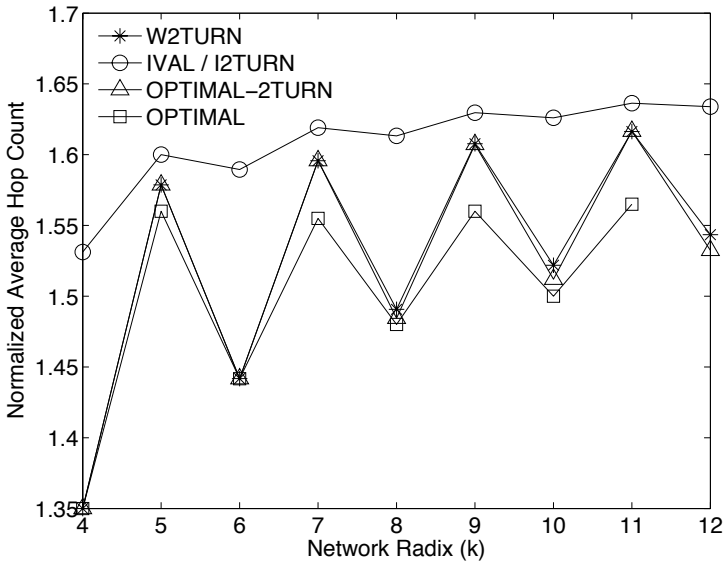


Figure 3.6: Comparison of average hop counts for several routing methods with optimal worst-case throughput on 2D-torus networks with different radices. Optimal routing and optimal routing with restriction to 2-turn paths are included.

it saturates earlier compared to WRD and RLB. WRD and RLB have comparable saturation throughput and latency under tornado traffic. Finally, for dynamic random traffic, we observe that WRD in fact achieves 9-10% lower latency compared to RLB over the entire range of injection rates. WRD also starts achieving lower latency compared to DOR beyond injection loads of 25% of network capacity. Hence, in addition to being worst-case throughput optimal, WRD performs well in the average case.

3.6.2 Evaluation of W2TURN

Hop count analysis

Figure 3.6 compares the average hop count of W2TURN with I2TURN, optimal 2TURN routing [65], and optimal routing [65]. The average hop counts are again normalized to minimal dimension-ordered routing. The optimal and optimal-2TURN routing results are obtained using the corresponding multicommodity flow formulations proposed in [65]. As shown in Figure 3.3, the average hop count of W2TURN is lower

than I2TURN for all network radices. When the network radix is odd, W2TURN achieves the same average hop count as optimal-2TURN, but this optimal result is achieved with a closed-form algorithm. When the network radix is even, W2TURN comes very close to optimal-2TURN, within just 0.72% in average hop count for k up to 12. Also, as shown in Figure 3.6, the hop count of W2TURN is very close to optimal routing when k is even[4], within just 1.4% for $k = 10$. Although optimal routing performs noticeably better when k is odd, it is difficult to guarantee deadlock-free operation for optimal routing because the resulting solution may include arbitrary paths and turns.

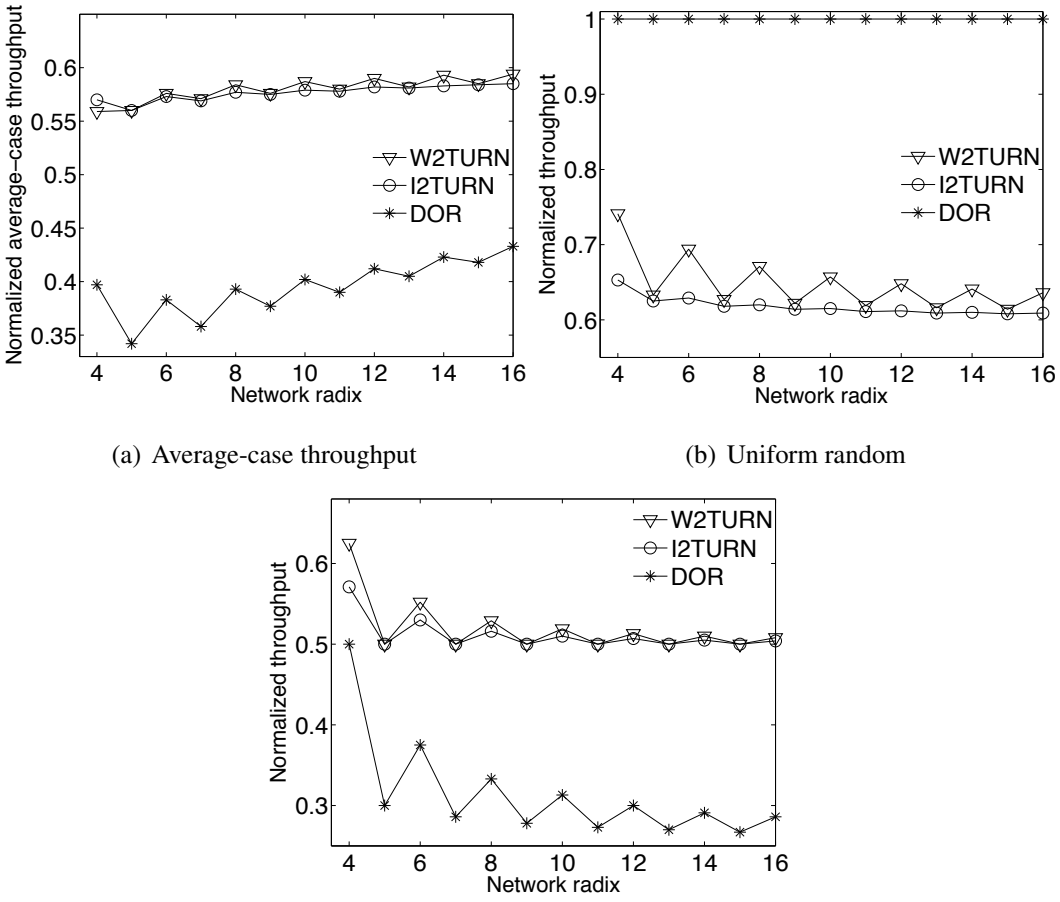
Throughput evaluation

Similar to our evaluation of WRD, in this section, we compare the throughput of W2TURN with IVAL/I2TURN, the best previously known worst-case throughput optimal routing algorithm with a closed-form description for 2D torus networks. We evaluate a randomized version of I2TURN that uses XYX and YXY routing paths with equal probabilities. The randomization balances the load between the X and Y channels and improves the average-case throughput of I2TURN over a non-randomized version using just XYX (or YXY) routing paths. We also include DOR in our evaluation to compare W2TURN with a minimal routing algorithm and to emphasize the importance of worst-case throughput optimality.

Throughput analysis is again carried out in two steps. We first present results using ideal throughput analysis over a range of network radices from 4 (16 nodes) to 16 (256 nodes) and then back these results using cycle-accurate flit-level simulations for an even-radix 8×8 topology and an odd-radix 7×7 topology. The traffic patterns used for evaluating W2TURN are two-dimensional versions of the patterns used for evaluating WRD, i.e., uniform random traffic and tornado traffic. In addition, we present average-case throughput results based on averaging the throughput of the routing algorithms over 10,000 randomly generated permutation traffic patterns.

Figure 3.7(a) compares the average-case throughput of W2TURN with I2TURN and DOR. W2TURN performs marginally better than I2TURN over all the network sizes considered. The average-case throughput of W2TURN is slightly higher than I2TURN

[4]The largest 2D-torus network with an even radix solved for optimal routing in [65] was $k = 10$.



(a) Average-case throughput

(b) Uniform random

(c) Tornado

Figure 3.7: Throughput evaluation of W2TURN for different network sizes.

when the network radix is even, but the difference is negligible when the network radix is odd. Both W2TURN and I2TURN, however, significantly outperform DOR in terms of average-case throughput. On average, the average-case throughput of W2TURN is 47.3% higher than DOR. This shows that although W2TURN (and I2TURN) are designed for optimal worst-case performance, they also perform well in the average case.

As in the case of one-dimensional rings, DOR achieves the highest throughput when the traffic is inherently load-balanced, as shown in Figure 3.7(b). W2TURN and I2TURN achieve lower throughputs due to their non-minimal nature. The average hop-count reduction of W2TURN over I2TURN helps it achieve a proportional increase in

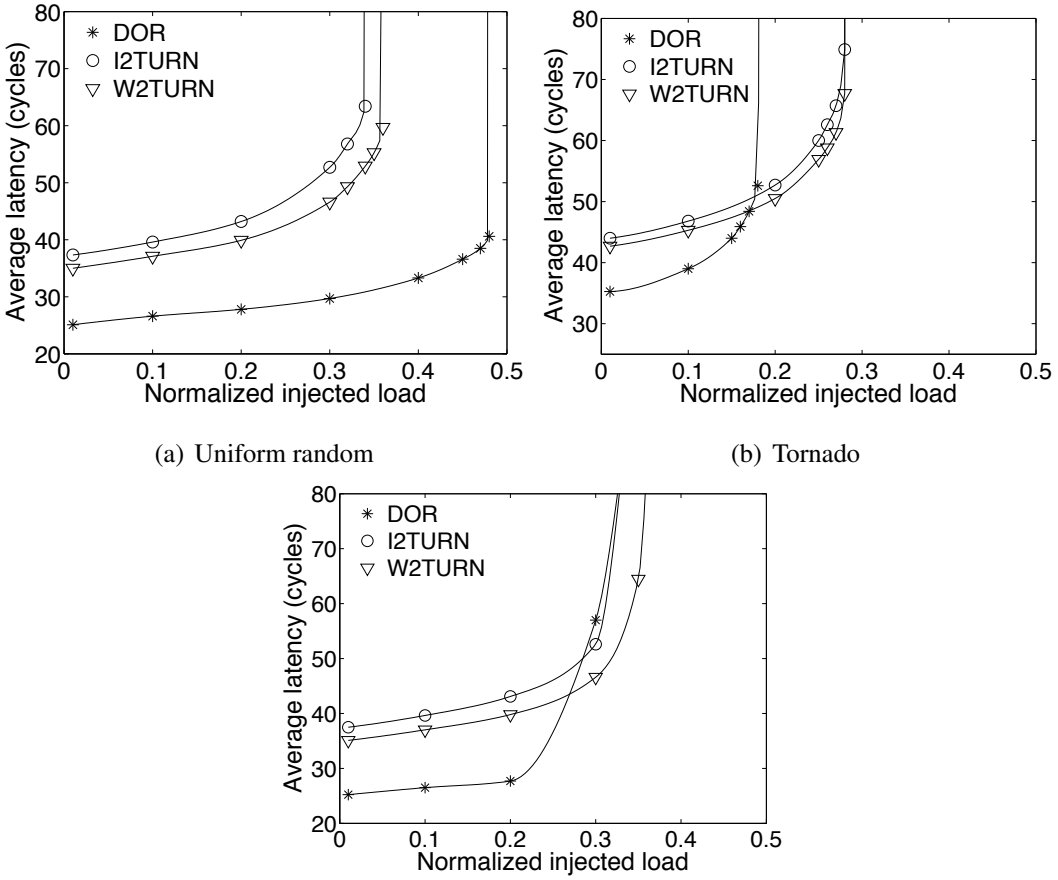
throughput under uniform traffic. As shown in Figure 3.6, the hop count reduction is higher when the network radix is even, resulting in higher throughput improvements. On average, under uniform random traffic, the saturation throughput of W2TURN is 7.75% higher than I2TURN for even-radix networks and 1.25% higher than I2TURN for odd-radix networks. Maximum improvement of up to 13.5% over I2TURN can be observed for the 4×4 topology.

Finally, tornado traffic is an adversarial traffic pattern for all three routing algorithms. In fact, it is the worst-case traffic pattern for W2TURN, I2TURN and DOR when the network radix is odd. Therefore, for odd radices, W2TURN and I2TURN achieve the optimal worst-case throughput of half the network capacity. The worst-case throughput of DOR is significantly lower. For odd network radices, W2TURN degenerates to I2TURN under tornado traffic. This can be deduced from the descriptions of the two algorithms in Sections 3.4 and 3.5.1 assuming $\Delta(x_1, x_2) = \Delta(y_1, y_2) = \lfloor k/2 \rfloor$. When the network radix is even, W2TURN can outperform I2TURN by up to 9.4% for low network radices. However, for high network radices, the throughput of W2TURN and I2TURN converge to half the network capacity for even-radix topologies.

Flit-level simulations

Next, we compare the performance of W2TURN, I2TURN and DOR using flit-level simulations. We use an even-radix 8×8 torus topology and an odd-radix 7×7 torus topology for our experiments. The simulation setup is similar to the one described in Section 3.6.1. The simulator models 5-port pipelined routers for the two-dimensional networks. The number of VCs used is still 8 but the buffering is increased to 8 flits per VC to accommodate the increased traffic volume in two-dimensional networks. As discussed in Section 3.5.5, 4VCs are sufficient to avoid deadlocks in 2-turn paths. However, increasing the number of VCs helps in significantly improving the performance of all the routing algorithms.

Figures 3.8(a), 3.8(b) and 3.8(c) present the flit-level simulation results for a 8×8 torus topology under uniform random traffic, tornado traffic and dynamic random traffic, respectively. The actual throughput sustained is around 50-60% of the throughput predicted using ideal analysis due to the non-idealities in the routers. However, the



(a) Uniform random

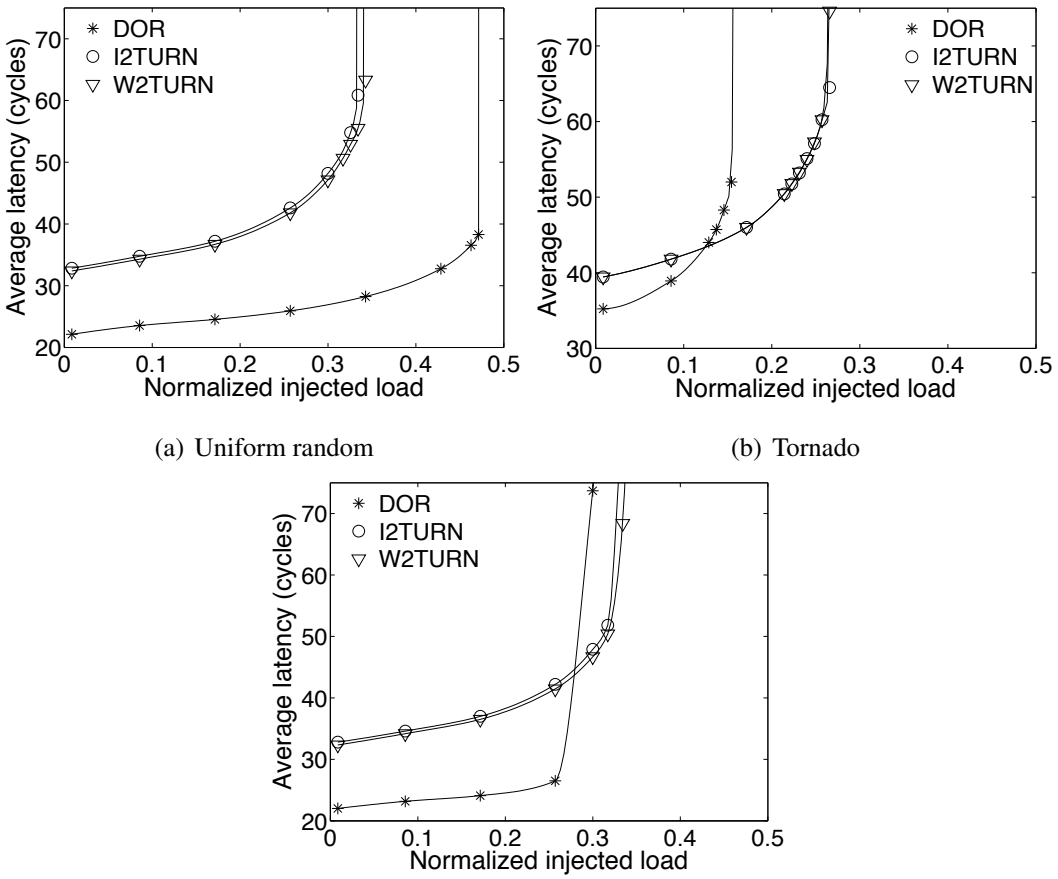
(b) Tornado

(c) Dynamic random

Figure 3.8: Performance of W2TURN on a 8×8 torus topology.

throughput trends are consistent with the ideal results. For uniform traffic, W2TURN achieves around 6% higher saturation throughput compared to I2TURN and at the same time, the latency under low loads is reduced by 6.5%. The latency reduction is quite close to the hop count difference of 8.2% predicted in Figure 3.6. The observed difference in latency is lower because the transmission delay of the 3-flit packets and the router pipeline delay at the destination node are ignored while calculating hop count. Both W2TURN and I2TURN, however, pay a latency penalty of 40% and 50%, respectively, over DOR as they select both minimal and non-minimal routing paths.

Tornado traffic is an adversarial traffic pattern for all three routing algorithms. Therefore, the maximum throughputs sustained are less than the throughputs sustained under uniform random traffic. The reduction is drastic for DOR, which does not



(a) Uniform random

(b) Tornado

(c) Dynamic random

Figure 3.9: Performance of W2TURN on a 7×7 torus topology.

guarantee optimal worst-case throughput. The maximum throughput sustained with DOR under tornado traffic falls by more than 60% compared to the maximum throughput sustained under uniform traffic. On the other hand, the throughput reduction for W2TURN and I2TURN are less severe at just 22% and 18%, respectively. W2TURN marginally outperforms I2TURN under tornado traffic, both in terms of latency and saturation throughput. W2TURN outperforms DOR significantly by 55% in terms of saturation throughput.

Figure 3.8(c) presents results for dynamic random traffic, which captures the average-case behavior of the routing algorithms. The results correspond to the average packet latency measured over 250 randomly generated permutation traffic patterns. The maximum latency of each observation is clipped at 125 cycles to prevent biasing

the average to a few large delays observed when the network saturates or approaches saturation. Under low loads of less than 20% of the network capacity, the average latency of W2TURN is 6-7% lower than I2TURN. However, for moderate to high loads, latency reductions of 12-40% can be achieved. This is because the saturation throughput of W2TURN is higher than I2TURN for most of the traffic patterns evaluated. The packet latencies tend to increase rapidly when the network approaches saturation, resulting in much higher latency reductions under high loads. W2TURN also outperforms DOR in terms of latency at and beyond injection rates of 30% of network capacity.

Finally, Figure 3.9 compares W2TURN with I2TURN and DOR on an odd-radix 7×7 topology. The throughput and latency differences between W2TURN and I2TURN are smaller for the odd-radix case, as expected from the ideal throughput analysis. W2TURN has marginally lower latency and marginally higher throughput under uniform and dynamic random traffic patterns. Under tornado traffic, W2TURN and I2TURN algorithms are identical.

3.7 Conclusion

This chapter presented an optimal closed-form routing algorithm for rings called WRD and a closed-form worst-case throughput optimal routing algorithm for 2D torus networks called W2TURN. WRD can achieve the minimum average hop count on rings while remaining worst-case throughput optimal. When the network radix is even, WRD achieves lower latency than RLB, the best-known worst-case throughput optimal routing algorithm for rings, under a wide range of traffic patterns. W2TURN routing algorithm for 2D torus networks is based on a weighted random selection of paths with at most 2 turns, which enables a simple deadlock-free implementation with just 4 virtual channels. W2TURN is shown to achieve optimal-2TURN routing when the network radix is odd and is within just 0.72% of the average hop-count of optimal-2TURN routing when the network radix is even. However, unlike optimal-2TURN, which requires solving large linear programs that do not scale, W2TURN has a closed-form algorithmic description that can scale to arbitrarily large networks. The chapter also presented an algorithm

called I2TURN that, like W2TURN, is based on a weighted random selection of 2-turn paths. We prove that I2TURN is equivalent to IVAL and hence, is worst-case throughput optimal. We also derive analytical expressions for the average hop counts of I2TURN and W2TURN. These are used to show that the average hop count of W2TURN is strictly less than I2TURN (and IVAL), the best previously known worst-case throughput optimal algorithm with a closed-form description. Finally, we show that W2TURN outperforms I2TURN both in terms of latency and throughput over a wide range of traffic matrices.

Chapter 3, in part, is a reprint of the material as it appears in the following publications:

- Rohit Sunkam Ramanujam and Bill Lin, "Weighted Random Routing on Torus Networks", *IEEE Computer Architecture Letters*, vol. 8, no. 1, January 2009.

- Rohit Sunkam Ramanujam and Bill Lin, "Weighted Random Oblivious Routing on Torus Networks", *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, Princeton, NJ, October 19-20, 2009.

Chapter 3, in full, has been submitted for publication of material as it may appear in IEEE Transactions on Computers, Rohit Sunkam Ramanujam, Bill Lin, "Randomized Throughput-Optimal Oblivious Routing for Torus Networks". The dissertation author was the primary investigator and author of the papers.

Chapter 4

Destination-Based Adaptive Routing on 2D Mesh Networks

4.1 Introduction

Oblivious routing algorithms where routing paths are independent of the network state have been the focus of Chapters 2 and 3. Oblivious routing has generally been preferred for on-chip networks because they are easy to implement and enable simple and fast router designs. These algorithms are designed to provide certain worst-case and average-case performance guarantees under relatively static network loads [53, 61, 65, 66]. However, network traffic in CMPs tends to be bursty in nature and oblivious routing algorithms sometimes perform poorly under such conditions [21]. On the other hand, adaptive routing, where routing paths can be dynamically changed depending on the network's congestion state, can achieve significantly better performance under bursty conditions. Along with the performance benefits, adaptive routing also offers better tolerance towards link and router failures compared to oblivious routing.

Ideally, every router in the network needs to have a perfect picture of global network congestion in order to make the best local routing decisions. Traditionally, adaptive routing algorithms have relied on local congestion metrics that are readily available at the routers in the form of flow control state. Although this approach adds very little router overhead (over oblivious routers), it is too slow to react to congestion

in more distant parts of the network as it relies on network backpressure to propagate congestion state.

More recently, an adaptive routing algorithm called regional congestion awareness (RCA) [20] was proposed for 2D mesh networks, which uses a lightweight monitoring network to propagate non-local (regional) congestion state from more distant parts of the network, like along a dimension or quadrant. While knowledge of regional congestion is better than just local congestion, it does not always accurately reflect delays along the actual paths a packet can take to its destination. We consider a minimal adaptive routing model for 2D mesh topologies where packets are only routed along the minimal directions. Under this model, the regional congestion estimates used to route a packet may encompass congestion on links outside the admissible paths to a packet's destination, thereby, degrading the quality of the estimates. To counter this drawback, we propose an adaptive routing algorithm that takes into account a packet's destination while making routing decisions. In our proposed destination-based adaptive routing (DAR), every node estimates the average delay to every other node in the network through each of the candidate output ports (assuming minimal adaptive routing). Routing is then carried out using per-destination *traffic split ratios* at every router, which provide fine-grained control for independently load-balancing traffic flows to different destinations based on the per-destination delay estimates. This is different from the coarser approach used in RCA where all flows going to a common region in the network use the same congestion estimate. In our evaluations, we find that maintaining finer congestion state helps DAR outperform RCA and local adaptive routing in terms of latency and throughput on both real and synthetic workloads.

The main contributions of this chapter are as follows:

- We propose a destination-based adaptive routing algorithm called DAR for on-chip mesh networks.

- We describe the router microarchitecture to implement DAR and show that the logic, storage and bandwidth overheads are quite reasonable for moderately sized networks. We explore different implementations of the monitoring network, which affects the rate at which congestion state is measured and updated in DAR.

- We also propose a scalable version of DAR called SDAR that reduces the implementation overhead of DAR for large networks.

- Finally, we evaluate the performance of DAR and SDAR on a number of real and synthetic workloads. DAR achieves latency reductions of 58% maximum and 13% average over RCA, 79% maximum and 21% average over local adaptive routing, and 96% maximum and 40% average over O1TURN [53] on eight SPLASH-2 benchmarks. SDAR also outperforms existing adaptive and oblivious routing algorithms under a wide range of synthetic traffic workloads on a 16×16 mesh topology.

The rest of the chapter is organized as follows. Section 4.2 discusses related work. Section 4.3 describes the DAR algorithm. Section 4.4 discusses the router microarchitecture for practically implementing DAR. Section 4.5 presents a scalable version of DAR called SDAR. Section 4.6 compares the performance of DAR and SDAR with existing adaptive and oblivious routing algorithms. Finally, Section 4.7 concludes the chapter.

4.2 Related work

A lot of prior research has focused on designing good oblivious routing algorithms for on-chip networks [53, 61, 65, 66]. These algorithms are simple to implement and are designed to achieve a certain level of worst-case or average-case performance under relatively static network loads. However, network traffic in CMPs is generally bursty in nature [21] and dynamic traffic conditions can cause temporary congestion hot spots in the network. Oblivious routing fails to deliver high performance under such dynamic conditions since it cannot react to network congestion by adapting its routing paths.

Adaptive routing has been extensively studied for multi-chip interconnection networks [24, 26, 43, 51]. However, the constraints for off-chip adaptive routing are vastly different from the constraints for the on-chip scenario and the best solutions in both domains are likely to be very different. This is because on-chip networks have

to operate under very tight power and area budgets and as a consequence, the routing overheads at both the end-nodes and network switches need to be kept to a minimum.

Adaptive routing algorithms can be classified as either minimal or non-minimal, based on whether non-minimal routing is permitted. Non-minimal adaptive routing [26, 56] has the potential to improve performance over minimal routing, but it is more complex to implement and results in longer packet latencies due to extra router complexity and non-minimal routes. Most non-minimal adaptive routing approaches have been proposed for networks with greater path diversity than a 2D mesh, like torus and dragonfly topologies.

In [20], Gratz. et al. present a taxonomy of adaptive routing policies based on the measured congestion state. Traditionally, most adaptive routing algorithms that sense network congestion only use local congestion metrics like free VC count [14], downstream buffer count [34] and output queue lengths [56]. Although local congestion metrics are easy to sense, greedy routing decisions based on these metrics can result in global imbalances in network load. Gratz et al. propose an adaptive routing algorithm called Regional Congestion Awareness (RCA) [20] that uses a lightweight monitoring network to propagate remote congestion state from more distant parts of the network. It was shown in [20] that sensing regional congestion along a dimension or quadrant in a 2D mesh can significantly improve performance over local adaptive routing. Other approaches that use remote congestion state are Token Flow Control (TFC) [37] and Indirect Adaptive Routing (IAR) [26]. TFC is complementary to DAR as it is conceivable that it can be used in conjunction with any adaptive routing algorithm. IAR deals with the challenges of large scale multi-processor systems as opposed to DAR, which is designed for on-chip 2D mesh networks.

4.3 Destination-based adaptive routing

To the best of our knowledge, Regional Congestion Awareness (RCA) is the only known adaptive routing algorithm for on-chip mesh networks that uses non-local congestion state. Although RCA shows significant performance improvements over local adaptive routing, it still faces a difficult challenge of balancing remote and local

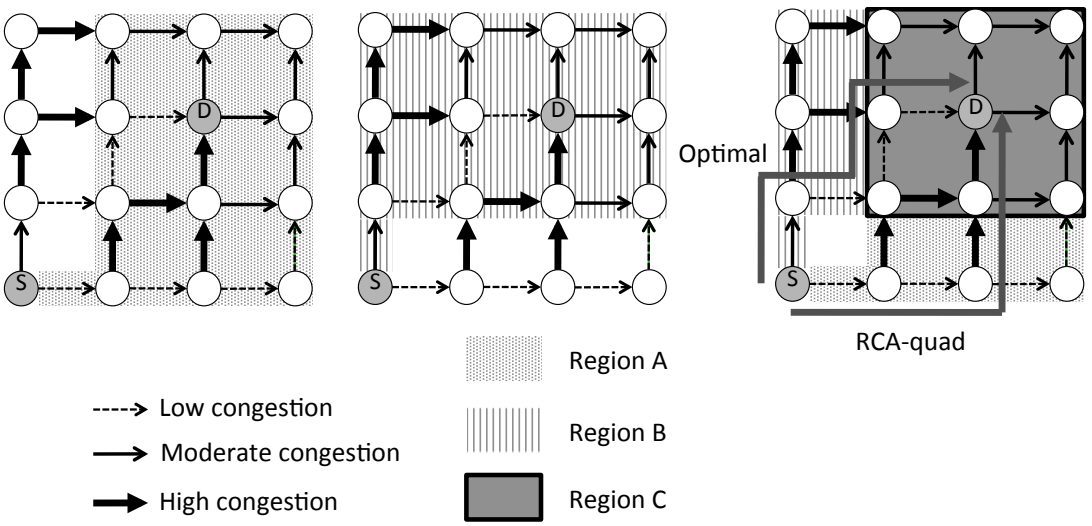


Figure 4.1: Drawback of region-based congestion estimation.

congestion state. This is because under minimal adaptive routing, the relevance of congestion on a downstream link while routing a packet is greatly dependent on whether the link is likely to be used by the packet to reach its destination. RCA deals with this issue by weighing local and non-local congestion state equally, but this may not lead to the best routing decisions. For example, consider the congestion snapshot of a 4×4 mesh network shown in Figure 4.1. At this instant of time, source S needs to send a packet to destination D and the RCA-quadrant algorithm is used for routing. There are two candidate output ports for routing the packet, i.e., the east and north ports. The routing decision is taken based on the congestion estimated on the two output ports. For simplicity, the congestion on links is categorized into only three categories−low, moderate, and high. The congestion on the east port is measured as the mean of the congestion on the local link and an exponentially weighted average of the congestion on all links in the north-east quadrant, represented by region A. Similarly, the congestion on the north port is a measured as the exponentially weighted average of the congestion on links in region B. The exponential averaging is done based on distance to ensure that links further away from a node have less impact on its congestion estimates. Since the links in region C contribute equally to the congestion estimates along the north and east ports, the routing decision depends only on the difference in congestion between regions $(A-C)$ and $(B-C)$. In this example, congestion (or lack of congestion) on links that are

outside the minimal routing paths to the destination (two links at the north-west corner of region ($B-C$) are congested while two links at the south-east corner of region ($A-C$) are not congested) mislead the router at node S into taking the wrong routing decision. As a result, RCA-quadrant chooses a sub-optimal path from S to D with much higher congestion, as shown. This example illustrates that the relevance of remote congestion state is greatly dependent on the packet's destination and the congestion on links outside the admissible paths to a packet's destination can corrupt the congestion estimates in region-based schemes.

This leads us to Destination-Based Adaptive Routing (DAR), where every node maintains per-destination congestion state in the form of *expected* delays to all other nodes through the candidate output ports permitted by minimal adaptive routing. The measured delays more accurately represent the congestion along paths from the source to the destination without being corrupted by congestion on links outside the admissible paths. However, the main challenge in estimating and maintaining such fine-grained congestion state in on-chip routers lies in having a lightweight implementation that can fit into the tight on-chip power, delay, and area budgets. In this section, we describe the adaptive routing strategy used in DAR and deal with the implementation issues later in Section 4.4.

4.3.1 Minimal adaptive routing model

In this chapter, we focus on adaptive routing using only minimal paths in a 2D mesh topology. Minimal routing is preferred due its simplicity and lower latency compared to non-minimal routing. Under this assumption, if a packet needs to be routed from source i to destination j, at every intermediate node starting from i, it has a choice of at most two output ports along minimal routes to j. For example, in Figure 4.2, a packet originating at node 1 can use either the east (E) or the north (N) ports to reach node 10, but not the west (W) port. The candidate output ports at intermediate routers for routing packets along minimal routes from node 1 to node 10 are shown using arrows in Figure 4.2.

At a high level, DAR works as follows:

- Every node periodically estimates the expected delay to all other nodes in

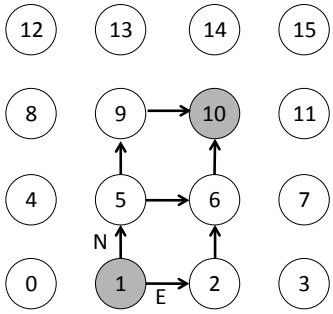


Figure 4.2: Adaptive routing along minimal routes.

the network through the candidate output ports. Due to constraints of time, bandwidth, and hardware resources required for distributed delay measurement, all of which are scarce in on-chip networks, delays are estimated periodically with a period T, rather than being estimated every cycle.

- The router associated with a node controls the distribution of traffic to its next-hop routers using per-destination traffic split ratios. This means that at a given router, all packets destined for the same node are distributed in the same ratio to the downstream routers. Packets going to different destinations can, however, be distributed independently in different ratios. The split ratios are also updated every T cycles, based on the measured delays. In DAR, traffic split ratios are used instead of simply choosing the output port with lower delay, like in RCA or local adaptive routing. This is done in order to prevent overloading the output port with lower delay during the relatively longer periods between delay measurements in DAR compared to RCA or local adaptive routing.

Based on this high-level description, there are two main tasks involved in DAR. First, every node estimates the delay to every other node in the network through the candidate output ports in a distributed manner. Second, the measured delays are used to determine the ratio in which traffic for a given destination is split among the candidate outputs at a router. Next, we explain the first task, which is distributed delay measurement.

Table 4.1: Notations.

$W[p][j]$	Fraction of traffic for destination j forwarded on port p.
$l[p]$	Estimated delay to downstream router through port p.
$A[p][j]$	Average delay from downstream router connected to port p to node j.
$L[p][j]$	Average delay from current node to node j through port p.
$Avg_i[j]$	Average delay from node i to node j.

4.3.2 Distributed delay measurement

Consider a mesh topology with N nodes. Each node maintains the following state information that is used in the delay measurement process:

- $W[p][j] \quad \forall j \in 1$ to N: Fraction of traffic for destination j forwarded on port p. This is maintained for all ports p permitted by minimal adaptive routing to destination j.

- $l[p] \quad \forall p \in \{N, S, E, W, Ej\}$: Estimated delay to downstream router through port p. N, S, E, W and Ej represent the north, south, east, west and ejection ports, respectively.

With minimal adaptive routing, a packet[1] can choose between at most two output ports at each intermediate router. If p_x and p_y denote the two candidate output ports to reach destination j, the weights $W[p_x][j]$ and $W[p_y][j]$ control the ratio in which traffic destined for node j is split between output ports p_x and p_y, and are called the traffic split ratios for destination j.

The delay measurement process starts with each router periodically estimating the local delays on all five output ports. This is denoted by $l[p]$ and it represents the delay incurred on the link connecting output port p to the downstream router. We assume an input buffered router model, where exact delays depend on the implementation of the virtual channel (VC) allocator and the switch arbiter modules of the router. However, since we are mainly interested in the relative delays through the candidate output ports, we use buffer occupancy estimates to approximate link delays. There are two main sources of contention in an input buffered router. First, flits destined for the same

[1]Adaptive routing is done at the granularity of packets. Like any adaptive routing algorithm, a reordering buffer is needed at the end-node to order packets, if ordering is required.

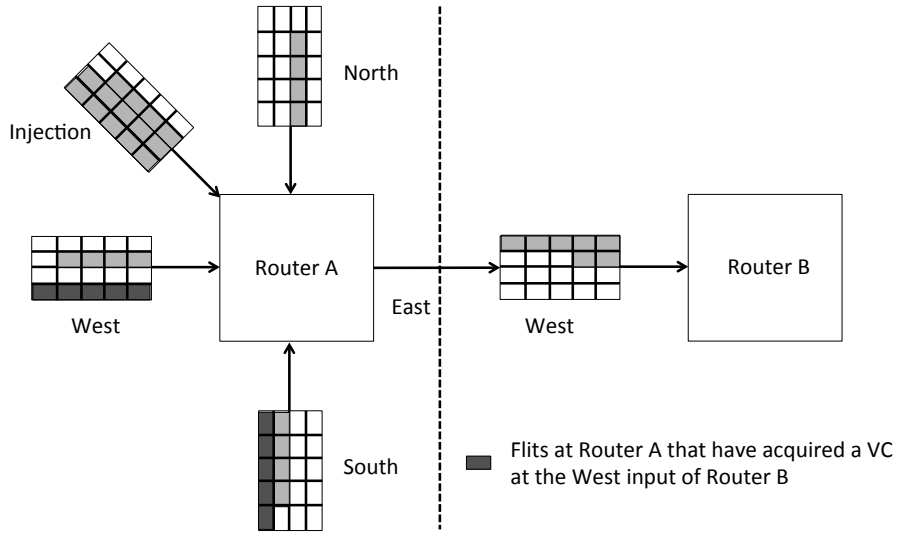


Figure 4.3: Example of local delay measurement.

output port contend for output VCs[2] during VC arbitration and access to the crossbar output during switch arbitration. Second, flits stored in different VCs at the same input port contend for access to the crossbar input. We take both these contentions into consideration while estimating local delays. The local delay through output port p, $l[p]$, is measured as the sum of the number of flits in the current router that have acquired a VC at the input of the downstream router connected to port p and the number of flits already stored in the input buffers of the downstream router at port p. For example, consider the pair of routers shown in Figure 4.3. The flits at router A that are shaded in dark gray have already acquired a VC at the west input of router B. Hence, in this case, the local delay on the east port of router A is estimated as 17, which is the sum of 10 flits in router A that have acquired a VC at the west input of router B and 7 flits already stored in the west input of router B. The first part of the sum estimates the delay due to output port and output VC contention while the second part contributes to the contention between input VCs when flits try to leave router B. Since output port selection and VC allocation are done in the same cycle in the DAR router (refer Section 4.4.2), the output port is fixed only for packets that have acquired a VC at a downstream router. Hence, only these packets are counted while estimating the delay through output port p.

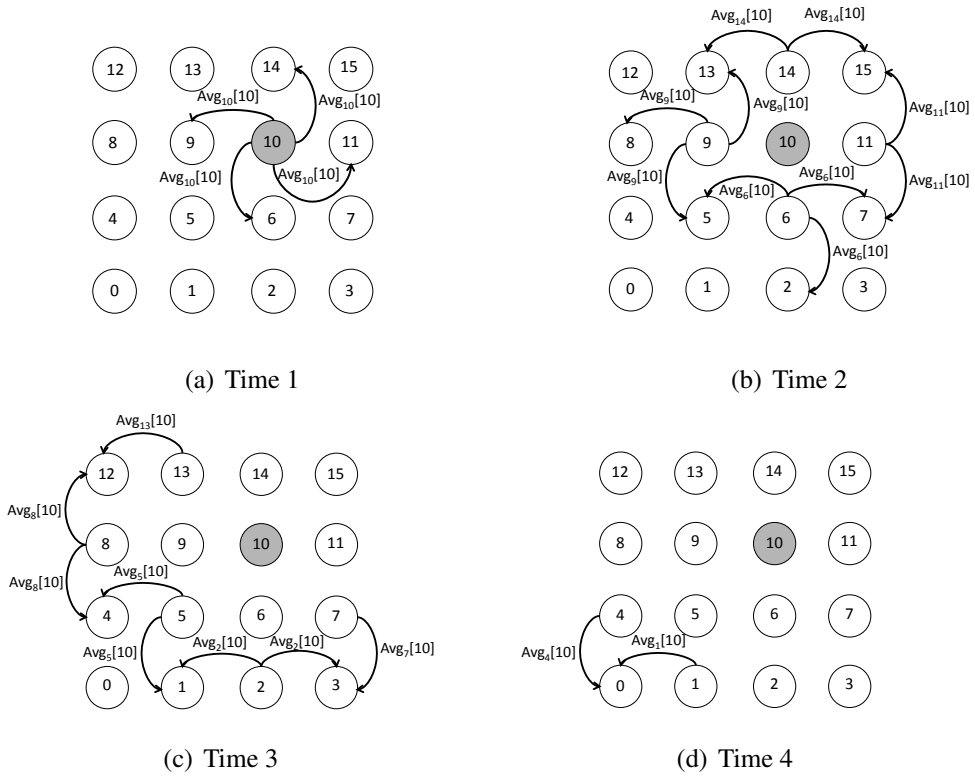
[2]Output VCs refer to VCs at the input of the downstream router.

All nodes in the network periodically estimate the delay to all other nodes through the candidate output ports in a distributed manner by sharing delay information with neighbors over a separate monitoring network. The distributed delay measurement algorithm works as follows. Suppose all nodes need to measure the delay to node j. The first node that has this information is node j itself, which periodically measures the delay on its ejection port. Node j then shares this information with all its neighbors. Let us say that node k, which is a neighbor of j receives the delay update from node j on port p. Node k can now estimate its delay to j through output port p by adding its locally measured delay on port p with the update received from j. Since node k is only one hop away from j, it can reach j through only one port under the minimal routing assumption. So the average delay from k to j is equal to the delay from k to j through port p. Node k shares this average delay information with all neighboring nodes other than j. A node s further away from j can receive delay updates to j through more than one port (at most two). Let us call the two ports p_x and p_y. In such a situation, node s estimates the delay to j through both ports by adding the local delays on ports p_x and p_y to the corresponding delay updates received along these ports. It then computes its average delay to j by weighing the delays through the ports by the traffic split ratios W[p_x][j] and W[p_y][j]. The average delay from s to j is then transmitted to nodes further upstream. In this manner, average delay information to j is propagated from j, first to all nodes one hop away, then to all nodes two hops away and so on until every node in the network has estimated its delay to j. At every hop, a node aggregates its local delay with the delay update received from downstream neighbors and propagates its average delay to j to nodes further upstream.

Next, we illustrate the computations involved in the delay measurement algorithm using an example. The notations used in this context are explained in Table 4.1. Consider a 4×4 mesh topology shown in Figure 4.4(a). Suppose all nodes in the network need to measure the delay to node 10.

- Delay from node 10 to itself is just the delay on the ejection port of node 10. $A_{10}[10]$ denotes the average delay from node 10 to itself and is computed at node 10 as:

$$A_{10}[10] = l[\mathrm{Ej}]$$



(a) Time 1

(b) Time 2

(c) Time 3

(d) Time 4

Figure 4.4: Delay measurement to node 10 in a 4×4 mesh.

- $A_{10}[10]$ is propagated to the neighbors of node 10. Assuming it takes 1 unit of time for delay updates to propagate, at the next time slot, nodes 9, 11, 6, and 14 receive $A_{10}[10]$ through their east (E), west (W), south (S), and north (N) ports, respectively, as shown in Figure 4.4(a).

- Each of these nodes that are one hop away estimate their delay to node 10 by adding their locally measured delays on the port leading to node 10 to $A_{10}[10]$. For example, at node 9, the delay to node 10 through the east port, L[E][10], is estimated as follows:

$$L[\mathrm{E}][10] = l[\mathrm{E}] + A[\mathrm{E}][10]$$

where $A[\mathrm{E}][10] = A_{10}[10]$ is the delay update received from node 10 through the east port. Since the nodes that are one hop away can reach node 10 through exactly one port, the average delay from each of these nodes to node 10 is given by $L[p][10]$ where p is the port leading to node 10. For example the average delay

from node 9 to node 10 is given as:

$$Avg_9[10] = L[\mathrm{E}][10]$$

The nodes that are one hop away from node 10 share their average delay estimate to node 10 with their neighbors that are two hops away from node 10, as shown in Figure 4.4(b).

- At time slot 2, nodes 5, 8, 13, 15, 7, and 2 receive updates for the delay to node 10. For example, node 5 receives updates about the average delay to node 10 from nodes 6 and 9 connected to the east and north ports, respectively.

$$A[\mathrm{E}][10] = Avg_6[10], \quad A[\mathrm{N}][10] = Avg_9[10]$$

Node 5 then estimates the delay to node 10 through its east and north ports by adding its locally measured delay on these ports to the delay update received from the downstream node.

$$L[\mathrm{E}][10] = A[\mathrm{E}][10] + l[\mathrm{E}], \quad L[\mathrm{N}][10] = A[\mathrm{N}][10] + l[\mathrm{N}]$$

Finally, node 5 estimates its average delay to node 10 by computing a weighted mean of the delays through the east and north ports, the weights given by the traffic split ratio for destination 10 at node 5.

$$Avg_5[10] = W[\mathrm{E}][10] * L[\mathrm{E}][10] + W[\mathrm{N}][10] * L[\mathrm{N}][10]$$

$Avg_5[10]$ is then propagated to the neighbors of node 5 that are three hops away from node 10, as shown in Figure 4.4(c). Similarly, nodes 8, 13, 15, 7, and 2 also compute and propagate their average delay to node 10 to their neighbors which are 3 hops away from 10.

- Carrying on in this manner, after 4 time units, all nodes in the network are able to measure their delay to node 10 through the candidate output ports permitted by minimal adaptive routing.

4.3.3 Adaptation of split ratios

After delay measurement, the next task is to use the measured delays through the valid output ports to update the per destination traffic split ratios. In this section, we describe the adaptation of traffic split ratio at a router for a single destination node. The same algorithm is repeated for all destination nodes and at all routers. As discussed earlier, due to various on-chip resource constraints and the nature of the delay measurement algorithm, delays are measured periodically with a period T. The split ratios are also adapted with period T, whenever delays are updated.

Suppose at node i, there are two candidate output ports p_x and p_y along minimal routes to destination j. Using the delay estimation mechanism described previously, node i can estimate $L[p_x][j]$ and $L[p_y][j]$, the delay to node j through ports p_x and p_y, respectively. In DAR, the ultimate goal is to load-balance traffic uniformly over all minimal paths to the destination. Since the router at node i only controls the distribution of traffic to the next-hop routers, it individually contributes towards the overall load-balancing goal by attempting to equalize the delay to node j through output ports p_x and p_y. So periodically, with a period T, the split ratio for traffic to node j is adapted such that the volume of traffic being sent on the port with a higher delay to j is decreased and vice versa. Let p_l denote the port with lower delay to j and p_h denote the port with higher delay to j. The split ratios $W[p_l][j]$ and $W[p_h][j]$ are normalized such that their sum is always equal to 1.

$$W[p_l][j] + W[p_h][j] = 1$$

We first determine Δ, the change in the split ratios in time period T. The magnitude of Δ is computed as follows:

$$\Delta = min(\lambda \left(\frac{(L[p_h][j] - L[p_l][j])}{L[p_h][j]} \right), W[p_h][j]) \tag{4.1}$$

Intuitively, Δ is proportional to the ratio of the delay difference between the ports to the magnitude of the delay. Since the split ratios cannot be negative, Δ can be at most equal to $W[p_h][j]$. In the special case when Δ equals $W[p_h][j]$, all traffic to node j is sent through port p_l. The new split ratios are then computed as:

$$W[p_h][j] = W[p_h][j] - \Delta, \quad W[p_l][j] = W[p_l][j] + \Delta$$

The quantity λ in Equation 4.1 represents the maximum possible shift in the split ratios in time period T. The value of λ is dependent on T and is empirically chosen to ensure that the delay values through the two ports do not oscillate much and converge to the same value. The parameter T, which is the period over which the split ratios are updated, is determined by the bandwidth and logic requirement of the delay estimation algorithm. We further discuss this parameter in Section 4.4.

It must be noted here that split ratios are updated as a reaction to the measured delays. When a node receives a delay update for destination j (once every T cycles), it first computes and its delay to j through the candidate output ports ($L[p][j]$) and the average delay value that needs to be propagated to its upstream neighbors ($Avg[j]$). Following delay computation, it updates the split ratio for j. Therefore, the delay value propagated is based on the split ratio that was updated T cycles back. This is done because the delay measured in the current time period is mainly a reflection of the split ratio that was used in the last T cycles. In response to the delay imbalance caused by the existing split ratios, each node then individually tries to update its split ratio to achieve better load balancing.

4.4 Router implementation

In this section, we discuss the modifications to the baseline adaptive router architecture required for practically implementing DAR. On-chip routers operate under tight power, area and delay budgets. Compared to existing adaptive routing algorithms, DAR measures congestion state at a much finer granularity of a single destination, as opposed to local or regional congestion. Hence, we expect to add some overhead in storage, logic and bandwidth. However, our goal is to keep these overheads to a minimum and ensure that the extra logic added does not increase the critical path delay of the router. First, in Section 4.4.1 we describe the architecture of a baseline adaptive router. Section 4.4.2 then presents the modified architecture for implementing DAR. Finally, Section 4.4.3 discusses a practical implementation of the DAR router for an 8×8 mesh.

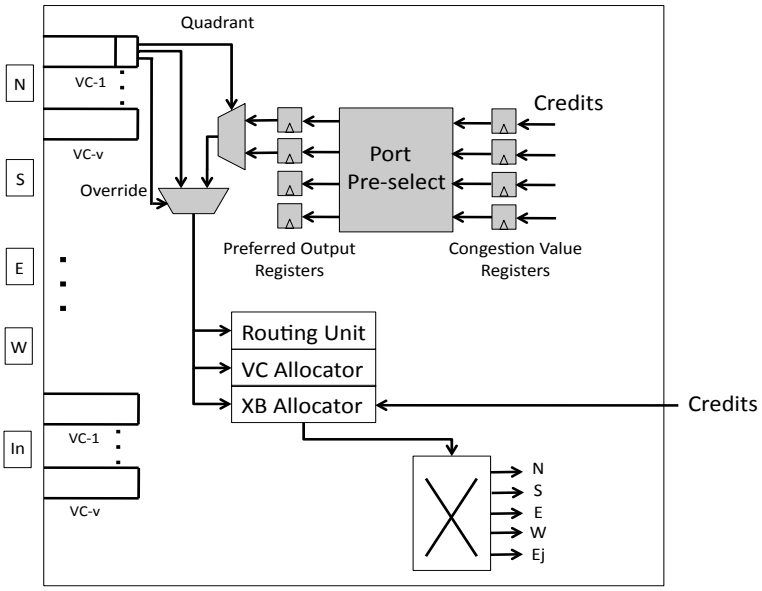


Figure 4.5: Baseline adaptive router.

4.4.1 Baseline adaptive router

Figure 4.5 shows the architecture of a baseline adaptive router for minimal routing on a 2D mesh topology based on Kim et al.'s design [34]. The original design proposed in [34] and used in [20] has a two-stage router pipeline with an additional cycle required for link traversal. For our evaluation, we use a router architecture with a three-stage pipeline as our baseline. The three pipeline stages include port-preselection, look-ahead route computation, and VC allocation in the first stage, switch arbitration in the second stage and switch traversal in the third stage. The head flit of a packet passes through all three stages and the body flits bypass the first pipeline stage and directly enter switch arbitration. As demonstrated by Mullins et al. [45], speculation can be used to reduce router latency to a single cycle. However, our work on adaptive routing is orthogonal to reducing the zero-load latency of routers. Hence, we use a non-speculative router as a baseline for our comparisons.

In Figure 4.5, the extra logic needed for adaptivity is shaded. The local congestion values for each output port are stored in the Congestion Value Registers. At the start of a cycle, the port pre-select module does a pairwise comparison of the

values stored in the Congestion Value Registers to determine the preferred output port for each of the four quadrants in a 2D mesh. When minimal adaptive routing is used, a packet arriving at an input port can have a choice of at most two output ports which maps to one of the four quadrants. So based on the packet's destination quadrant the preferred output port for that quadrant is chosen for routing the packet. When the destination is along the same dimension, a packet has no choice of output ports and the decision of the port-preselect logic is overridden. The port-preselection is done in the same cycle as VC allocation and route computation for the head flit of a packet. The route computation stage uses look-ahead routing to determine the destination quadrant of a packet at the next-hop router. The switch arbitration and switch traversal stages make up the rest of the router pipeline followed by link traversal. This baseline adaptive router architecture has very little overhead over routers used for oblivious routing. Next, we look at the router architecture for implementing DAR.

4.4.2 Router architecture for implementing DAR

Figure 4.6 shows the router architecture for implementing DAR. It uses the same 3-stage router pipeline with an additional cycle for link traversal. The storage and logic overheads over the baseline router are shaded in light and dark gray, respectively. In this architecture, instead of just maintaining congestion state for each output port like the baseline implementation, the DAR router stores a set of traffic split ratios, $W[p][j]$, for every destination j in the network. At the start of a cycle, the port-preselection logic computes a *preferred output port* for every destination node in the network using the traffic split ratio for that node. These values are latched onto the *preferred output port* registers. When a packet arrives at an input, its destination field is directly used to choose the *preferred output port* corresponding to its destination. After the port-preselect stage, the data path of the packet is identical to the baseline adaptive router.

Next, we take a look at the logic needed to determine these per-destination split ratios. As described in Section 4.3, there are two different computations being performed in the DAR router. First, there are the delay measurement and propagation blocks for carrying out the computations described in Section 4.3.2. The delay measurement block estimates the delay to all other nodes in the network through the

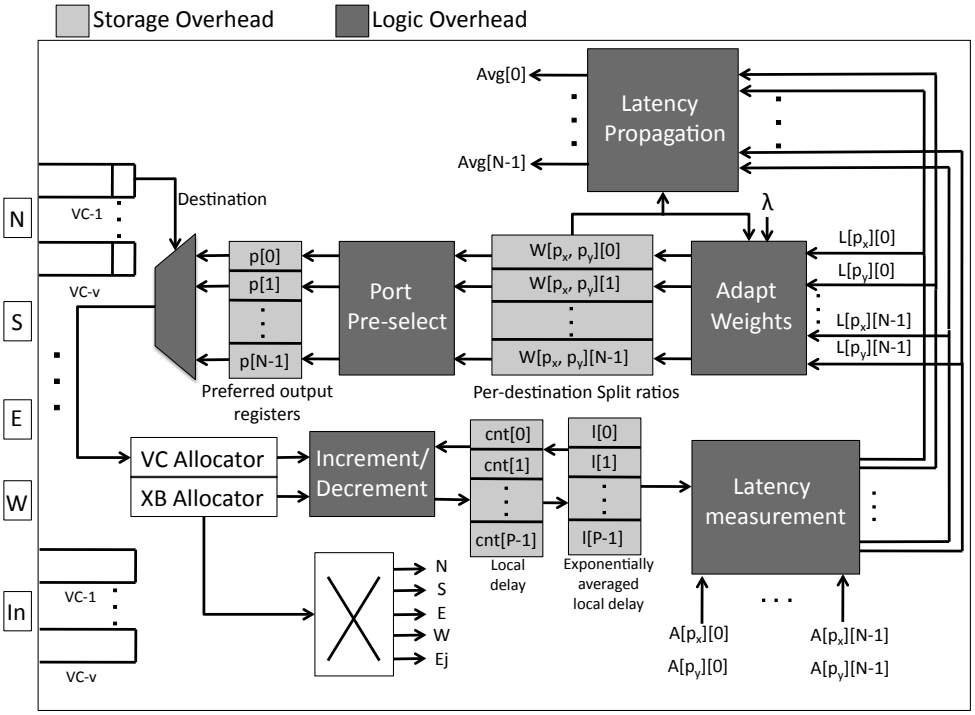


Figure 4.6: Architecture of the DAR router.

candidate output ports using updates received from downstream routers ($A[p][j]$) and the local delays measured for each output port ($l[p]$). An exponentially weighted average of the local delay to each output port is maintained in order to make the estimates more robust to sudden changes in traffic. The delay propagation block computes the average delay from the current router to all other nodes ($Avg[j]$) and propagates this information to upstream neighbors. The measured delays ($L[p][j]$) are then used to adapt the traffic split ratios using the equations described in Section 4.3.3.

Storage requirement

The extra storage required for the DAR router can be categorized as follows:

Split Ratios $W[p][j]$: As described in Section 4.3.3, the split ratios are normalized such that they always add up to one. Hence, storing the fraction of traffic through one of the output ports is sufficient as the traffic through the other port can be computed by subtracting the stored value from one. We assume a 5-bit

representation of the split ratios. So, for a network with N nodes, the storage overhead for split ratios is $5N$ bits.

Local delays $l[p]$: In a 2D mesh, a router needs to store local delays for each of its five output ports. The number of bits used for representing local delays depends on the amount of buffering at the input ports. In addition to the current delay values, an older copy of the delay values also need to be stored for maintaining an exponentially weighted average of local delays.

Preferred output ports $p[j]$: For 2D mesh routers with 5 ports, this field requires 3 bits per destination.

Assuming 6 bits for storing local delays, the total storage overhead of the DAR router over a baseline adaptive router is $(8N+60)$ bits, where N is the number of nodes in the network. For a network with 64 nodes, the storage overhead is 71.5 bytes per router. Assuming a typical input buffered router has 200 flits of buffering (8 VCs/port, 5flits/VC) and assuming a modest flit width of 64 bits, the extra storage required for DAR is only around 4.5%. However, since the storage increases linearly with the number of nodes, for a network with 1024 nodes, the overhead can be more than 60% of the total buffering in an input buffered router.

Bandwidth requirement

DAR uses a separate congestion monitoring network to propagate delay updates between nodes. As described in Section 4.3.2, the delay to a node j is propagated throughout the network in a hop-by-hop manner once every T cycles. During this process, an intermediate node i propagates its average delay to node j to only those neighbors whose hop-count to j is greater than the hop-count from i to j. This propagation scheme ensures that a link in the network propagates an update for node j at most once every T cycles. Hence, in a network with N nodes, a network link may have to carry at most N updates every T cycles. In fact, we can get a much tighter upper bound on the maximum bandwidth required and show that in a symmetric bidirectional

mesh with N nodes, a link in the monitoring network will have to carry at most $N - \sqrt{N}$ updates every T cycles. Hence, the value of T and the width of the monitoring network are two inter-dependent parameters and fixing one of them determines the value of the other.

4.4.3 Practical implementation of DAR

In this section, we discuss the details of the DAR router and the design of the monitoring network for a 8×8 mesh topology. In DAR, the delay measurement and propagation block and the split ratio adaptation block need to perform computations on a per destination basis. However, since computations for a destination need to be performed once every T cycles, T can be set to be large enough to stagger computations in such a way that at a given time slot, a router in the network works on measuring the delay and adapting the split ratio of exactly one destination. This provides a straightforward strategy for reusing hardware, which is essential for making the on-chip implementation of DAR feasible. Before stating the problem of staggering the computations, we first define a time slot as follows:

Definition 1 (Time slot)**.** *A time slot is defined as the total time taken to process delay updates for a single destination at a router and to transmit the aggregated update to upstream routers.*

Next, the problem of staggering the computations at nodes can be defined as:

Problem 1. Find a vector $Start\_update$ whose element, $Start\_update[j]$, represents the time slot at which node j starts propagating its own delay ($Avg_j[j]$) to its neighbors such that when the updates are propagated, no node in the mesh receives updates for more than one destination in the same time slot.

The time taken for one round of delay updates from all nodes to reach all other nodes in the network determines the value of T. Figure 4.7 shows an example of a $Start\_update$ vector for a 8×8 mesh, where the time slot at which a node in the mesh starts propagating its delay update is marked on the node. Such an assignment ensures that under deterministic hop-by-hop update propagation, where each hop takes

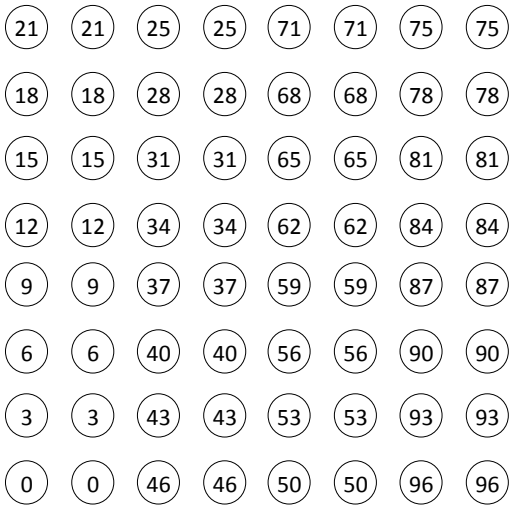


Figure 4.7: *Start_update* vector example for a 8×8 mesh.

one time slot, no node receives updates for more than one destination in the same slot. The deterministic update propagation scheme also ensures that updates for the same destination (through different ports) arrive at the same time slot. Using the assignment shown in Figure 4.7, the value of T has to be 103 time slots.[3]

The number of cycles per time slot is determined by two factors−the number of cycles needed to compute the aggregated update that needs to be propagated and the width of the monitoring network, which determines the time taken for propagating the aggregated update to the upstream router. We first look at the detailed implementation of the different blocks to determine the number of cycles required for the computations.

Delay measurement and propagation

Using the same notations used in Section 4.3.2, for every destination j, the delay measurement block needs to compute:

$$L[p_x][j] = A[p_x][j] + l[p_x], \quad L[p_y][j] = A[p_y][j] + l[p_y]$$

For a 8×8 mesh we use 9 bits to represent $L[p][j]$, which means that delays of up to 512 cycles can be measured.

[3] T = 96 + 7 time slots to avoid a conflict between updates from nodes starting at time slot 96 and time slot 0.

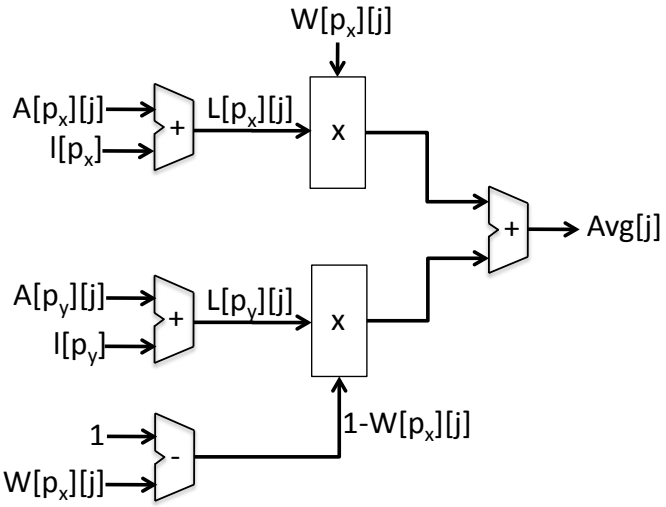


Figure 4.8: Delay measurement and propagation logic.

In order to make the routing algorithm more robust to sudden changes in local delays, an exponentially weighted average of the local delay is maintained for all output ports. This weighted average can be calculated at a finer timescale compared to T (e.g. $T/8$) as it is not in the critical path of the delay computation. Let $cnt[p]$ represent the current delay for output port p. This can be measured using a set of counters which count the number of flits in the input buffers that have already reserved buffering at the next-hop router connected to output port p and the credit counters which keep track of the number of flits at the input buffers of downstream routers. Let $l[p]$ represent the exponentially weighted average delay estimate for port p,

$$l[p] = 0.5 * (l[p] + cnt[p])$$

The exponentially weighted averaging of the local delays can be done sequentially for the five output ports using a single adder.

The delay propagation block needs to compute:

$$Avg[j] = L[p_x][j] * W[p_x][j] + L[p_y][j] * W[p_y][j] \tag{4.2}$$

$$= L[p_x][j] * W[p_x][j] + L[p_y][j] * (1 - W[p_x][j]) \tag{4.3}$$

Figure 4.8 shows details of the delay measurement and propagation block. It uses 4 adders and two multipliers which multiply 9 bit delay values with 5 bit split ratios. Only

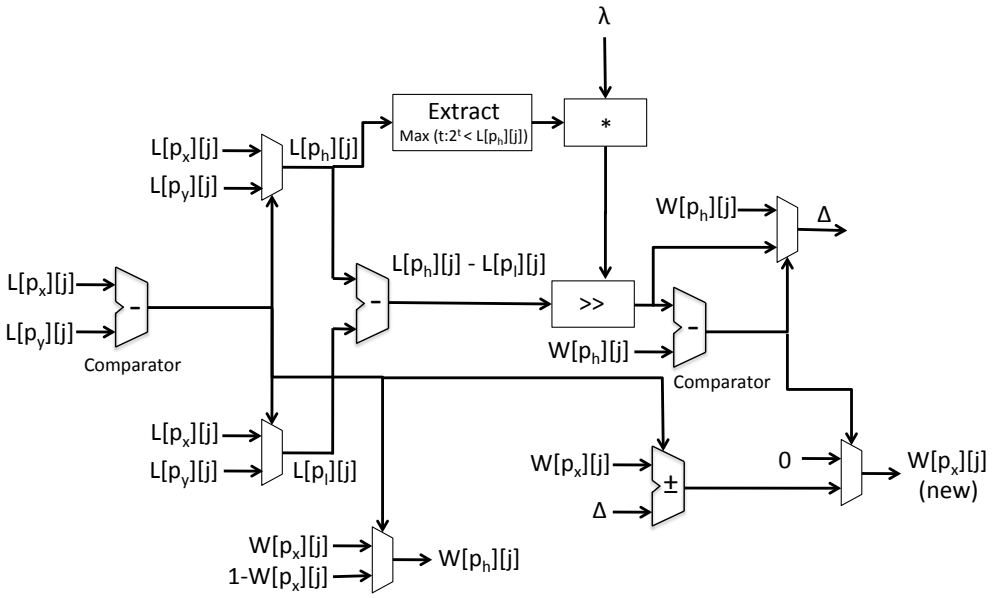


Figure 4.9: Logic for adaptation of weights.

the first 9 bits of the multiplier output are used for the final addition. Since the critical path contains a multiplier, the logic can be pipelined and implemented over 2 clock cycles to avoid adding to the critical path delay of the router. The choice of the number of stages, however, will also depend on the target frequency. For lower frequency of operation, alternative implementations that minimize logic and power can be used.

Split ratio adaptation

Using the same notations as in Section 4.3.3, the computations involved with adaptation of split ratios are given as follows:

$$\Delta = min(\lambda \left(\frac{L[p_h][j] - L[p_l][j])}{L[p_h][j]} \right), W[p_h][j])$$

$$W[p_x][j] = W[p_x][j] \pm \Delta$$

To simplify the implementation of these computations in hardware we always assume λ to be a power of 2 (eg. $\lambda = 0.25$), which reduces the multiplication to a shift operation. Division is also reduced to a shift operation by extracting only the most significant bit of $L[p_h][j]$ that is set and ignoring the remaining less significant bits. This optimization

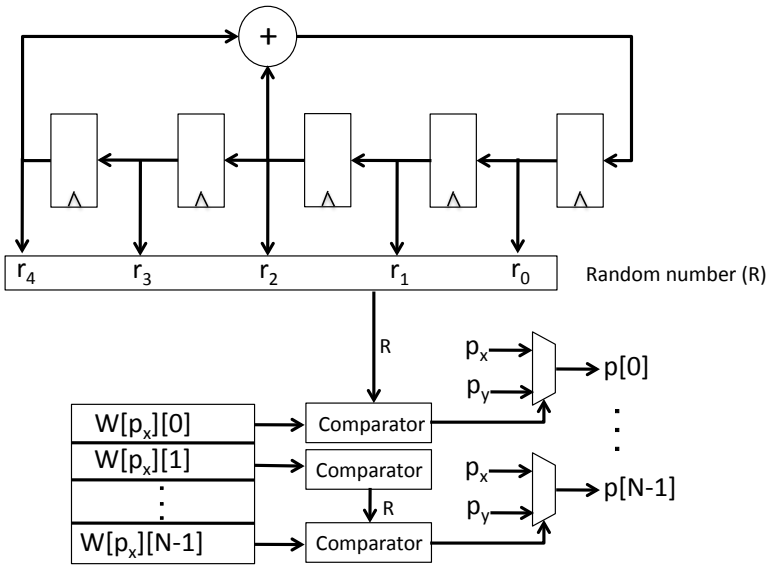


Figure 4.10: Port preselection logic.

can cause an error of a factor of at most two and can be viewed as having a variable λ (e.g. λ = 0.25 to 0.5). In our evaluations, we observe that DAR performs well even with these approximations. Figure 4.9 shows the hardware implementation of the block. It requires four 9-bit adders, five 2-input multiplexers (three 5-bit multiplexers and two 9-bit multiplexers), two barrel shifters, and block to determine the first bit of $L[p_h][j]$ that is 1. The logic to adapt the split ratios can also be pipelined across two cycles.

Port pre-select

The port-preselect stage shown in Figure 4.10 uses a 5-bit pseudorandom number generator to generate a single random number every cycle. All split ratios are compared against the same random number to select an output port for each destination. This requires 64 5-bit comparators that run in parallel and a shift-register based pseudorandom number generator.

Power and area overheads

We synthesized the additional logic (gates+registers) used in the DAR router using Synopsys Design Compiler with TSMC 65nm GP process libraries at 1V and an

Table 4.2: Power and area overheads of DAR router.

	Router	Additional DAR logic	DAR overhead
Power (mW)	246	10.62	4.3%
Area (um^2)	375792	10912	2.9%

operating frequency of 1GHz. Table 4.2 compares the post-synthesis power and area overheads of the DAR router with the power consumed by a typical 5-ported mesh router with 8 VCs per port, 5 flits per VC and 128-bit flit width. The Verilog RTL implementation for the router was generated using NetMaker [1], a fully-synthesizable parameterized router generator that implements an input-buffered pipelined virtual channel router. The mesh router assumes an oblivious routing algorithm and does not include the additional logic for adaptive routing. The router pipeline stages are similar to the ones described in Section 4.4.1 and comprise route selection, VC and switch allocation, and switch traversal. The router RTL was synthesized using the same TSMC 65 nm GP process libraries at 1V and 1GHz. The power and area consumed by the additional logic in the DAR router is 10.6 mW and 10912 um^2, respectively, which represent corresponding overheads of just 4.3% and 2.9% over a typical 5-ported mesh router.

Determining the value of T

The main computation in the critical path of the DAR router that determines the length of a time slot is the delay computation and propagation block that requires two cycles. In addition to the two cycles for computation, a time slot also needs to include the number of cycles needed to transmit the delay update from one node to the next. A delay update needs to include the 9-bit delay field and possibly a 6-bit destination field making it 15 bits. The 6-bit destination field is not mandatory as a router can decipher the destination from the time slot at which the update is received. This involves using the $Start\_update$ vector to generate a lookup table at every router that maps time slots to destinations. However, if delay computation takes two cycles to complete, it is possible to hide the transmission time of the destination field from the critical path by overlapping

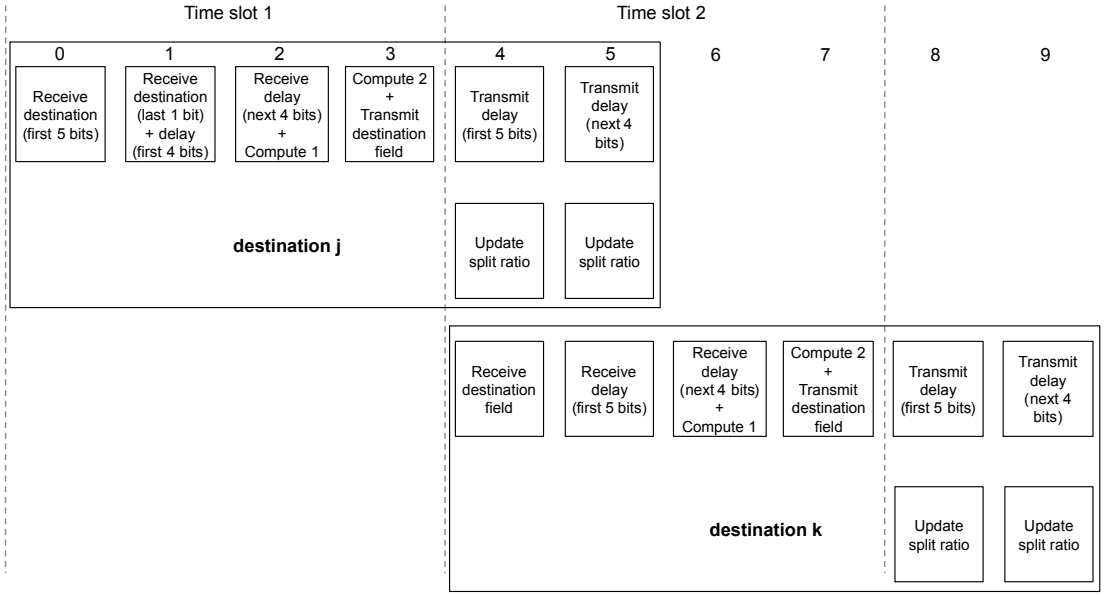


Figure 4.11: Processing of delay updates using a 5-bit monitoring network.

it with the ongoing delay computation, thereby avoiding the use of an extra lookup table at routers. Various possible scenarios leading to different values of T are described next.

- **Monitoring network is 5 bits wide:** In this case, it takes two cycles to transfer 9-bit delay updates over the 5-bit control network. The sequence of operations performed at a node for processing delay updates received in successive time slots for two destination nodes j and k are shown in Figure 4.11. The minimum length of a time slot is 4 cycles and using the *Start_update* vector of Figure 4.7, T needs to be 412 cycles (103 timeslots $\times$ 4 cycles) to ensure that a router processes delay updates for a single destination every time slot. The split ratio for a given destination is updated one time slot after delay computation and propagation, as shown.

- **Monitoring network is 9 bits wide:** In this case, only one cycle is required to transfer delay updates between neighboring routers. The transmission of the destination field can again be overlapped with one of the two computation cycles, resulting in a time slot of 3 cycles and T = 309 cycles. One way of further

reducing the value of T in order to make delay updates more frequent is to reduce the critical path of the delay computation stage to just one cycle, resulting in a 2-cycle time slot and $T = 206$ cycles.

- **Monitoring network is 3 bits wide:** In this scenario, 3 cycles are required to transfer 9-bit delay updates between adjacent routers. The transmission of the destination field requires two cycles, which can be overlapped with the two cycles required for delay computation. This results in a time slot of 5 cycles and $T = 512$ cycles.

Assuming 64 bit data flits, a 5 bit monitoring network results in a wiring overhead of only around 8% while 3-bit monitoring network represents a mere 4.6% overhead. In comparison, RCA-1D requires 8 bit wires per link and RCA-quadrant requires 16 bit wires per link for the congestion propagation network in a 8×8 mesh [20]. We show in Section 4.6 that even with a narrower control network, DAR can outperform RCA-quadrant under both real and synthetic traffic. In Section 4.6, we analyze the impact of T on performance of DAR to help us reach the right tradeoff between overhead and performance.

4.5 Scaling DAR to larger networks

As discussed in Section 4.4.2, both storage and bandwidth requirements for DAR do not scale to networks with a large number of nodes. In this section, we describe a variation of DAR called SDAR that can easily scale to large 2D mesh networks. We use a lookahead window approach for implementing SDAR. In this scheme, for a $k \times k$ mesh, instead of maintaining per-destination split ratios, a node i only maintains accurate delay information for a $M \times M$ mesh centered at i where $M < k$. For example, in Figure 4.12, a 7×7 lookahead window is shown for node S. When a packet's destination, node j, lies outside the lookahead window, it uses the delay estimate to the node closest to j within the lookahead window for determining its output port. The node within the lookahead window, whose delay estimates are used by traffic to node j, is called the proxy of j.

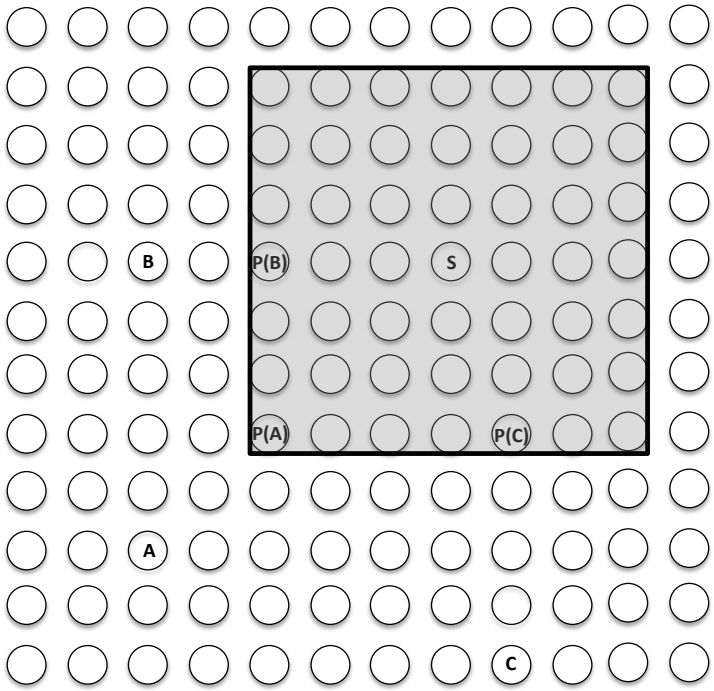


Figure 4.12: Lookahead window for implementing SDAR.

Every node outside the lookahead window can be uniquely mapped to a proxy node that is closest to it (in terms of number of hops) within the window. For example, in Figure 4.12, nodes $P(A)$, $P(B)$, and $P(C)$ represent the proxies of nodes A, B, and C, respectively. The lookahead window is different for every node in the network. As a packet is routed from the source to the destination node, the lookahead window shifts at every hop. The main intuition behind why such an approach works is that the final destination is guaranteed to be inside the look ahead window when the packet is at least $(M-1)/2$ hops away from it. If $(M-1)/2$ is large enough, the packet has enough time to route around any congestion hot-spot on its path to the destination.

In SDAR, the storage and bandwidth requirement at a node is determined by the dimensions of the lookahead window and not by the dimensions of the network. The value of M is always odd as the window for node j is symmetric with respect to j and offers a visibility of at least $(M-1)/2)$ hops (whenever possible) in all directions. For any destination node d that lies outside the window, the proxy of d with respect to the lookahead window of j can be easily calculated by truncating the coordinate of

d that is greater than $(M-1)/2$ hops away from the corresponding coordinate of j, to $(M-1)/2$. The delay propagation scheme is similar to DAR except for the fact that a node propagates a delay update for node j further upstream only if the upstream node is within the lookahead window of j. The $Start\_update$ vector used for staggering the latency updates for different destinations looks the same as that of a $(M+1) \times (M+1)$ mesh, repeated multiple times to cover the entire $k \times k$ mesh.

The choice of window size, M, is a tradeoff between performance and overheads associated with storage, bandwidth, and logic complexity. Increasing the window size increases the accuracy of the delay estimates but also forces the latency update frequency $(1/T)$ to be low due to bandwidth constraints. In Section 4.6 we explore the effect of the lookahead window size on the performance of SDAR on a 16×16 mesh.

4.6 Evaluation

In this section, we compare the performance of DAR and SDAR with known adaptive and oblivious routing algorithms.

4.6.1 Experimental setup

The adaptive routing algorithms used for comparison are local adaptive and RCA-quadrant. The local congestion metric measured in both these adaptive routing algorithms is the same metric used in DAR to measure delays on the local output ports. We experimented with a few other local congestion metrics like downstream free VC count, free buffer count, demand for crossbar outputs, and different combinations of these metrics, but found that free buffer count used in conjunction with crossbar output demand performs best. In addition to the adaptive routing algorithms, we also compare the performance of DAR and SDAR with O1TURN routing [53], which is known to be the best oblivious routing algorithm for 2D mesh networks.

We modified PopNet [54], a cycle-accurate flit-level on-chip network simulator to implement the routing algorithms. The simulator models the router pipeline described in Sections 4.4.1 and 4.4.2 for local adaptive routing and DAR, respectively. The RCA router includes minor modifications to the baseline router for aggregation and

propagation of remote congestion information along quadrants (RCA-quadrant) as described in [20]. The routers employ credit-based virtual channel flow control. Virtual channels (VCs) are also used for deadlock avoidance using escape VCs [16]. For our evaluation, input buffers comprise of 8 VCs, each 5-flit deep. One out of 8 VCs is used as an escape VC to ensure deadlock freedom. The simulator computes the average packet latency over all packets injected into the network. The latency of a packet is measured as the delay between the time the head flit is injected into the network and the tail flit is consumed at the destination.

We evaluate the routing schemes under both real and synthetic workloads. For real workloads, we use memory traces from eight SPLASH-2 benchmarks [71], representing a typical chip multiprocessor (CMP) scientific workload.The traces used are for a 49-node shared memory CMP [38] arranged as a 7×7 mesh. The SPLASH-2 traces were gathered by running the corresponding benchmarks with 49 threads[4] on Bochs [39], a multiprocessor simulator with an embedded Linux 2.4 kernel. The memory trace was captured and fed to a memory system simulator that models the classic MSI (Modified, Shared, Invalid) directory-based cache coherence protocol, with the home directory nodes statically assigned based on the least significant bits of the tag, distributed across all processors in the chip. Each processor node has a two-level cache (2MB L2 cache per node) that interfaces with a network router and 4GB off-chip main memory. Access latency to the L2 cache is derived from CACTI to be six cycles, whereas off-chip main memory access delay is assumed to be 200 cycles. For synthetic traffic, we generate *uniform*, *transpose*, and *perfect shuffle* traffic traces with different injection rates. The traces are generated using a self-similar injection process [47] with a Hurst constant of 0.8 and a standard deviation of 30% about the mean injection rate. A self-similar injection process accurately models the burstiness or temporal correlations in network traffic. A 7×7 mesh topology is used to evaluate DAR under the SPLASH workloads and a 8×8 mesh is used for the synthetic traffic traces. For evaluating the performance of SDAR, we use a larger 16×16 mesh topology. Since it is difficult to generate real traffic traces for the large 256-node topology, we tried to emulate the

[4]Two of the eight traces, `fft` and `radix` could not be run with 49 threads. They were run with 64 threads instead. The memory traces of the first 49 threads were captured and the addresses were mapped onto a 49-node system.

spatial traffic characteristics of real workloads by generating a set of 20 traffic patterns where traffic from a node is equally distributed to a randomly chosen subset of 16 nodes in the network. We also ensured that each node receives traffic from at most 16 other nodes. The SPLASH traces were simulated for 10 million cycles or the entire duration of the trace, whichever came first. All synthetic traces were simulated over a million cycles and latency was measured after a warm up period of 20,000 cycles.

For DAR, we experiment with two different values of λ, $\lambda = 0.25$ and $\lambda = 0.5$. We also evaluate the four possible configurations described in Section 4.4.3 resulting in four different values of T ranging from 206 cycles to 512 cycles for the 8×8 mesh topology. For the 7×7 mesh, a *Start_update* vector similar to the one shown in Figure 4.7 is derived, which requires 83 time slots for completing a round of delay updates. Varying the number of cycles per time slot again results in four different values of T for the 7×7 topology ranging from 166 cycles to 415 cycles. In subsequent discussions, a DAR configuration is described using parameters T and λ as DAR(T, λ). We evaluate SDAR with different lookahead window sizes, ranging from a 3×3 window to a 9×9 window. The value of T increases with increasing window size. A SDAR configuration is described using three parameters, T, λ and length of a side of the lookahead window, M, as SDAR(T, λ, M).

4.6.2 Performance on SPLASH-2 benchmarks

In this section, we first compare the performance of DAR with known adaptive routing algorithms and then with oblivious O1TURN routing [53].

Comparison of DAR with adaptive routing algorithms

Figure 4.13 compares the average packet latency of DAR with RCA-quadrant and local adaptive routing on eight SPLASH-2 benchmarks. The latencies are normalized to the average latency of local adaptive routing. The DAR configuration used for comparison is one that uses a 5-bit monitoring network, with $T = 332$ cycles and $\lambda = 0.25$. DAR significantly outperforms both RCA and local adaptive routing on three of the eight benchmarks, namely, `fft`, `waterns`, and `waters`. In [20], the authors categorize the benchmarks into *contended* and *uncontended* categories based

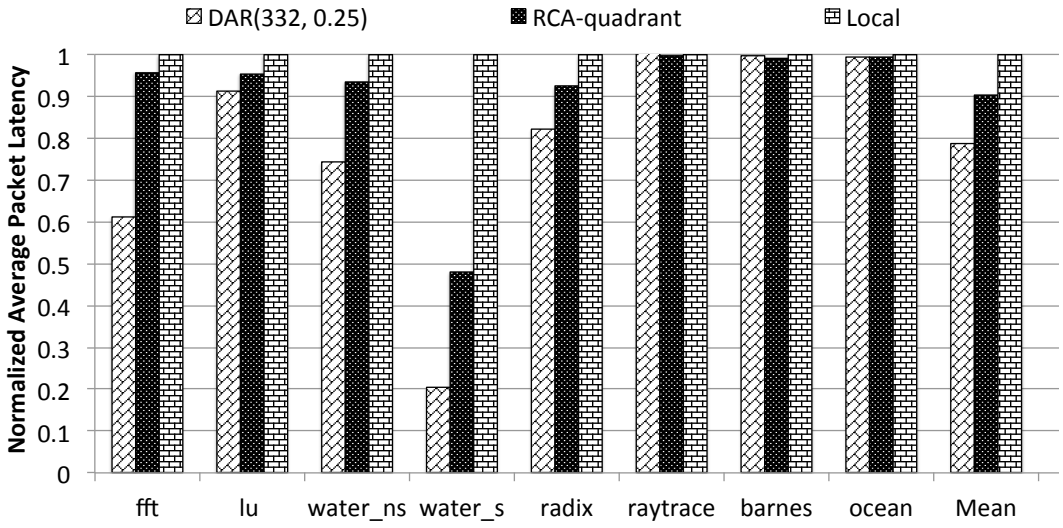


Figure 4.13: Comparison of DAR with adaptive routing algorithms on SPLASH-2 benchmarks.

on the fraction of packet latency that can be attributed to contention in routers. For the *contended* traces (`fft`, `waters`, `waterns`, and `lu`), DAR outperforms RCA by a maximum of 58% on the `waters` trace and by 28% on average over all four traces. The maximum improvement is seen for the `waters` benchmark. This trace contains a single traffic hot spot and is the same benchmark for which RCA has the maximum benefit over local adaptive routing. We see that DAR, as a result of maintaining more fine-grained congestion state, can outperform RCA significantly under such conditions. DAR outperforms local adaptive routing on the contended benchmarks by a maximum of 79% on the `waters` trace and by 39% on average over the four traces. For the *uncontended* benchmarks (`radix, raytrace, barnes`, and `ocean`), the scope for improvement is limited as contention doesn't contribute significantly to packet delays in these benchmarks. However, the average packet latency of DAR is comparable or better than the other adaptive routing algorithms even on these traces. The final set of bars show the arithmetic mean of the latencies over all eight benchmarks. DAR outperforms RCA by 13% and local adaptive routing by 21% on average over all eight traces.

Figure 4.14 presents the variation of the mean packet latency (normalized to the latency of local adaptive routing) over all eight SPLASH benchmarks with parameters

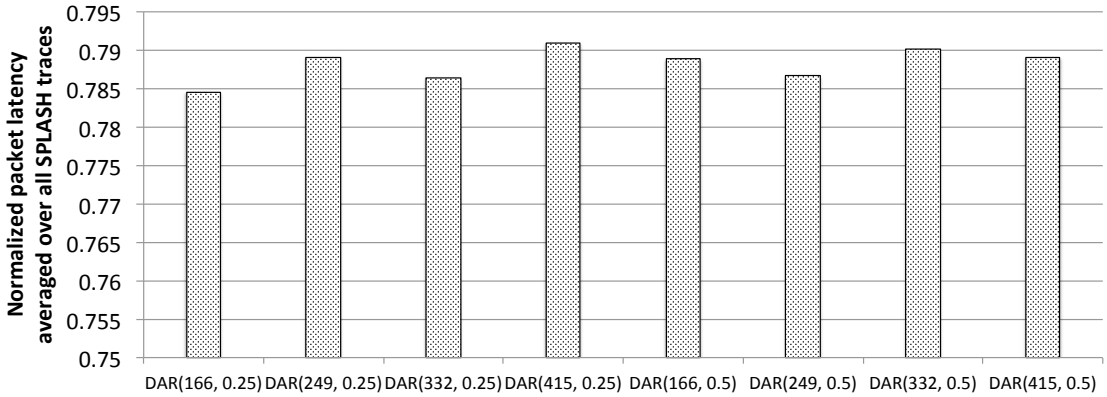


Figure 4.14: Performance sensitivity of DAR to λ and T. The DAR configurations are annotated as DAR(T, λ).

T and λ. As explained in Section 4.6.1, T is varied between 166 cycles and 415 cycles, corresponding to time slots ranging from 2 cycles to 5 cycles. λ varies between 0.25 and 0.5. The results show the variation between the different DAR configurations is negligible compared to the reductions achieved over existing adaptive routing algorithms. This indicates that spatial and temporal variations in traffic in the SPLASH benchmarks are generally much slower than the range of time periods used for updating delays and split ratios. Moreover, maintaining an exponentially weighted average of local delays at a much finer time scale (local delays are measured every 8 cycles) in all configurations makes DAR less sensitive to transient traffic variations.

Comparison with oblivious routing

Figure 4.15 compares the performance of DAR with O1TURN, the best performing oblivious routing for 2D mesh networks. DAR outperforms O1TURN by up to 96% maximum and by 40% on average over all eight benchmarks. This shows that under real workloads with time varying injection rates, adaptive routing performs considerably better than even the best oblivious routing solution.

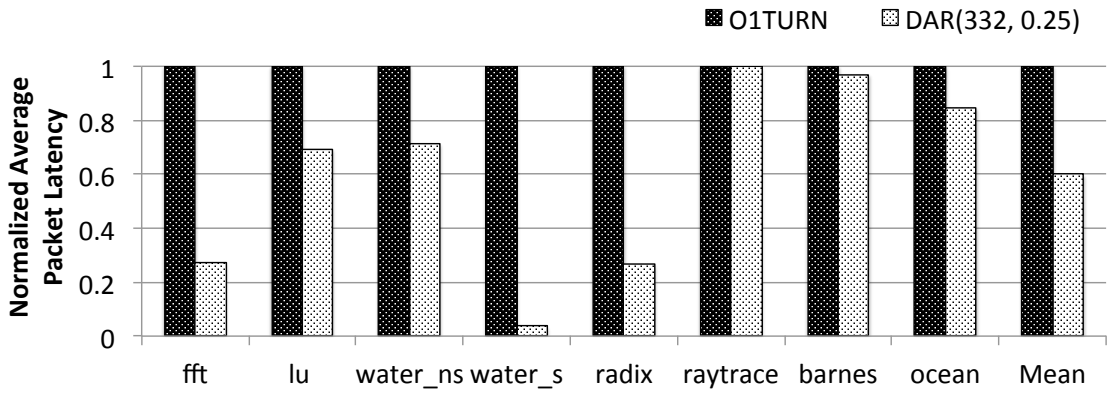
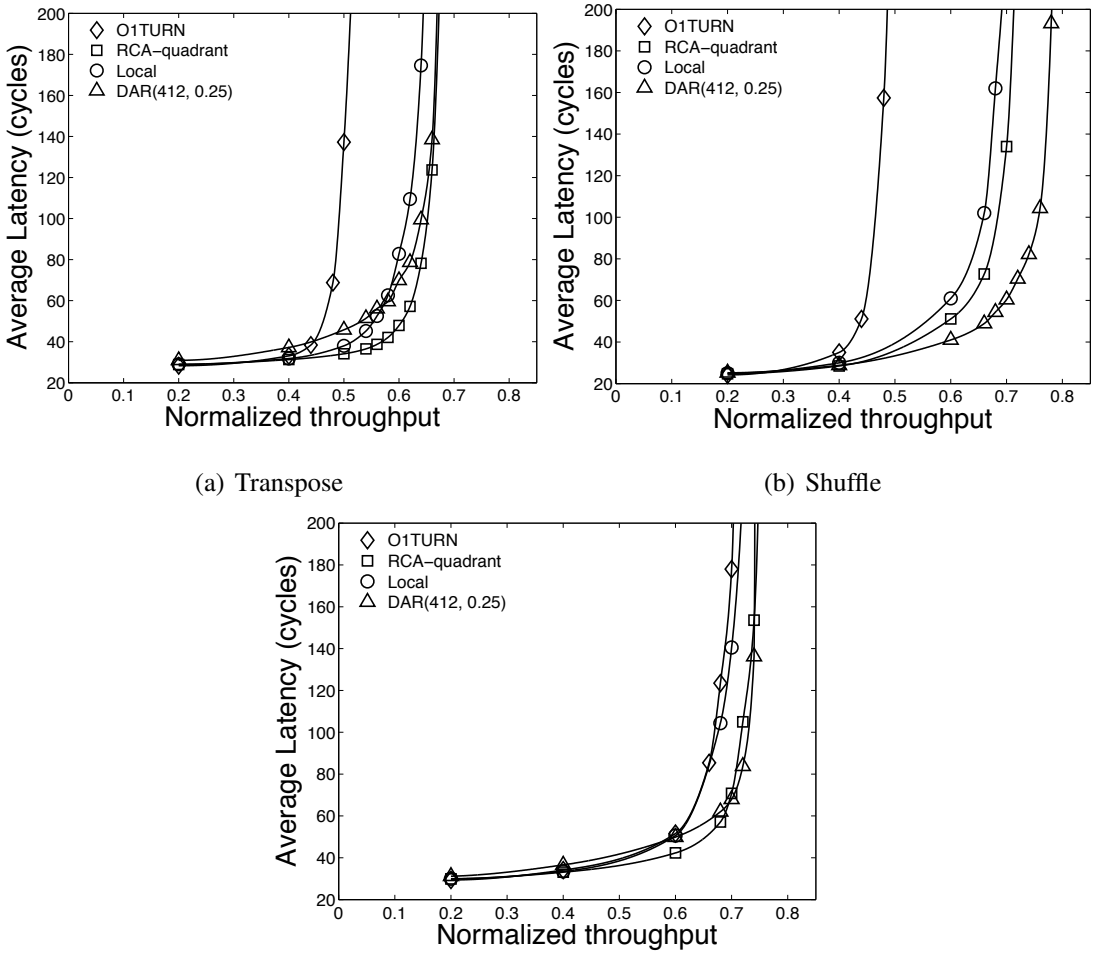


Figure 4.15: Comparison of DAR with O1TURN on SPLASH-2 benchmarks.

4.6.3 Performance on synthetic traffic

Figure 4.16 compares the performance of DAR with RCA-quadrant, local adaptive routing, and O1TURN under synthetically generated *transpose*, *shuffle*, and *uniform* traffic traces. A self-similar injection process with a standard deviation of 30% about the mean is used to emulate the bursty nature of network traffic. The throughput is normalized to the network capacity, which is defined as the maximum throughput that can be ideally sustained under uniform traffic.

In $transpose$ traffic, a node with coordinates (x,y) sends packets to a node with coordinates (y,x). *Transpose* is an adversarial traffic pattern, especially for oblivious routing using O1TURN. O1TURN performs poorly under this workload while DAR and RCA have the highest saturation throughput. In *perfect shuffle* traffic, assuming nodes are indexed from 0 to $N-1$, a node with index i sends packets to a node with index $2i$ when i is less than $N/2$. Otherwise, node with index i sends packets to a node with index $(2i+1-N)$. This traffic pattern is interesting because DAR outperforms RCA-quadrant by more than 12% in terms of saturation throughput. This is because the *shuffle* traffic pattern has a number of source-destination pairs that are close to each other and separated by only one or two hops along one of the dimensions. For these cases, misleading congestion estimates from links beyond the permitted minimal paths (as shown in Figure 4.1) can force a packet to reach one of the two destination coordinates, following which the packet is forced to travel along the remaining productive dimension.



(a) Transpose

(b) Shuffle

(c) Uniform

Figure 4.16: Performance of DAR on a 8×8 mesh under synthetic traffic patterns.

This route may be more congested than other available routes to the destination but packets lose the choice of output ports once a destination coordinate is reached. Local adaptive routing also performs poorly on this traffic pattern because *shuffle* has a mix of source-destination pairs with long and short separations and local adaptive routing is too slow to react to remote congestion.

For *uniform* traffic, the destination is chosen at uniform random. It is a benign traffic pattern since network load is implicitly balanced over the links by randomizing the destination. All routing algorithms perform well on this workload. DAR has a slightly higher latency than RCA at moderate loads because it is slower than RCA in

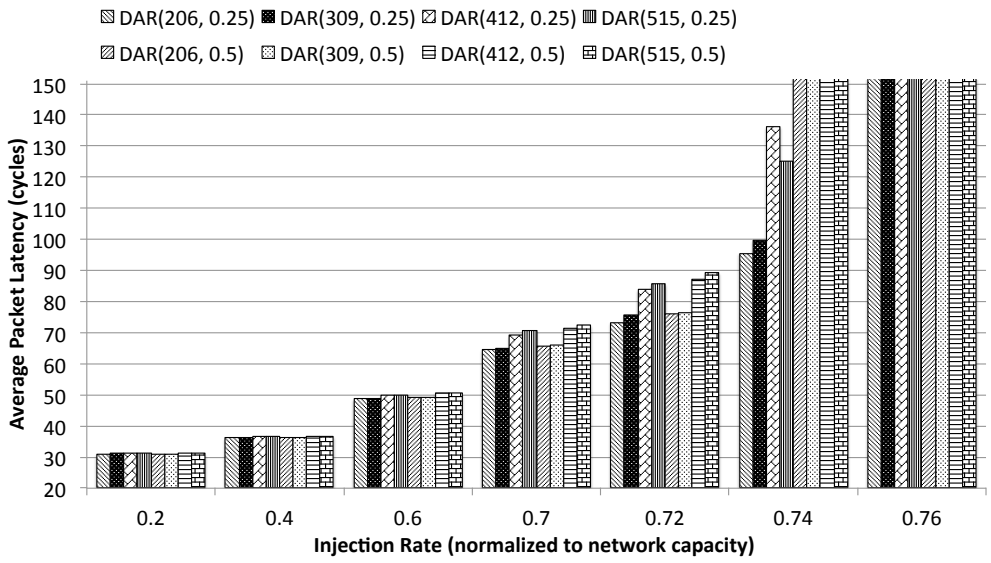
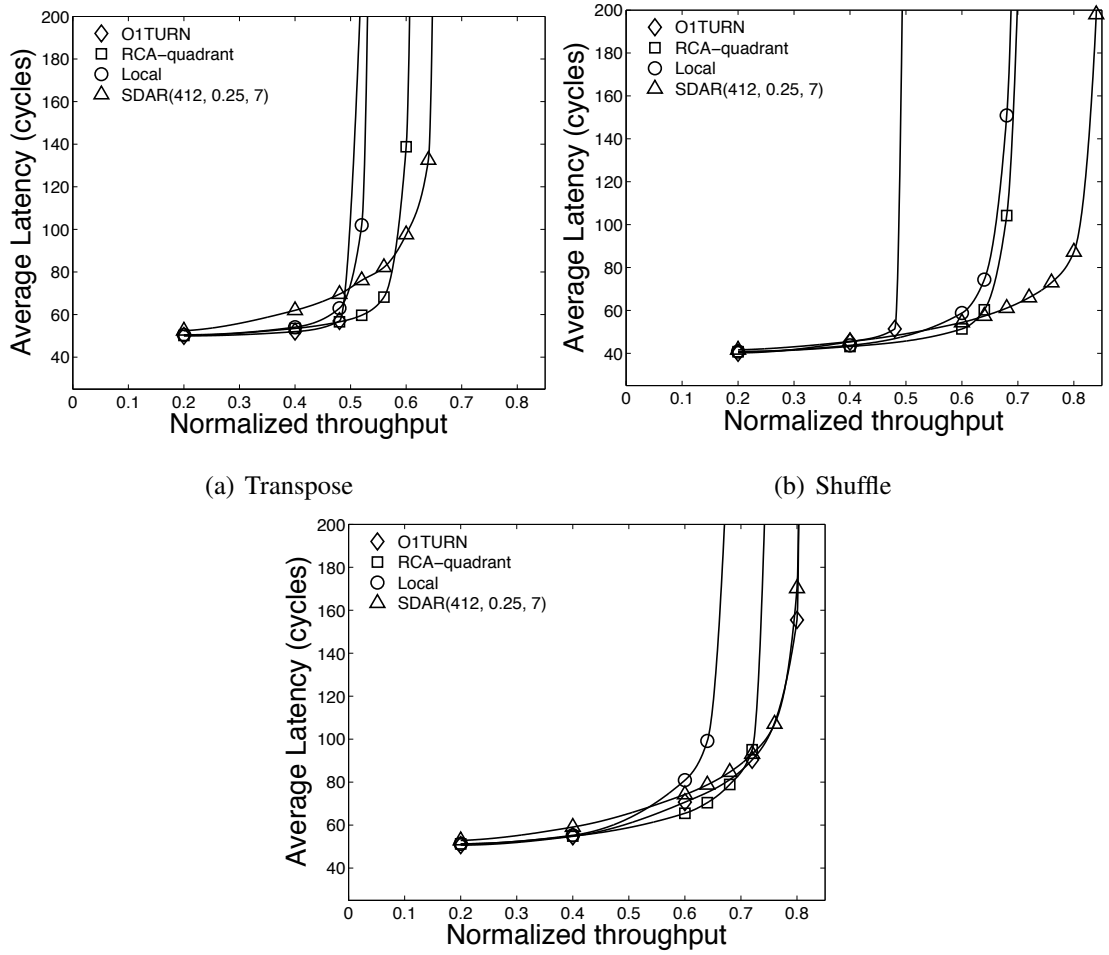


Figure 4.17: Performance sensitivity of DAR to λ and T under uniform traffic. The DAR configurations are annotated as DAR(T, λ).

adapting congestion state. DAR adapts the split ratios every 412 cycles compared to RCA which updates congestion state every cycle. However, the latency difference is very small, which justifies our case for maintaining more fine-grained congestion state than RCA, and updating the state at a slower rate.

Figure 4.17 compares the average packet latency of eight DAR configurations with different values of T and λ under uniform traffic. Since a bursty injection process is used, the variations in latency between different configurations is larger than those seen for the SPLASH benchmarks, especially under injection loads greater than 70% of network capacity. As a general trend, for moderate to high injection loads, the latency increases slightly with increasing T, which is expected, since lowering T results in faster delay computation and more frequent split ratio updates. DAR configurations with $T = 309$ cycles and 206 cycles can in fact match the slightly lower delays of RCA-quadrant under uniform traffic. The variation with λ is negligible until very high injection rates, when small changes in split ratio cause significant shifts in traffic and hence, large changes to path delays. Therefore, a smaller value of λ results in better load-balancing. However, if λ is too small, the algorithm may be too slow to react to congestion.



(a) Transpose

(b) Shuffle

(c) Uniform

Figure 4.18: Performance of SDAR on a 16×16 mesh under synthetic traffic patterns.

4.6.4 Performance of SDAR on a 16×16 mesh

In this section, we evaluate the performance of SDAR on a 16×16 mesh topology. Figure 4.18 compares the performance of SDAR with a 7×7 lookahead window with RCA-quadrant, local adaptive routing and O1TURN under *transpose*, *shuffle*, and *uniform* traffic workloads. The SDAR configurations are described using three parameters, T, λ and M, where M is the length of one side of the lookahead window. SDAR implemented using a 7×7 lookahead window gives a packet at least three hops to accurately detect and avoid congestion since a destination within 3 hops of a node in any direction is guaranteed to be within the lookahead window. In terms of

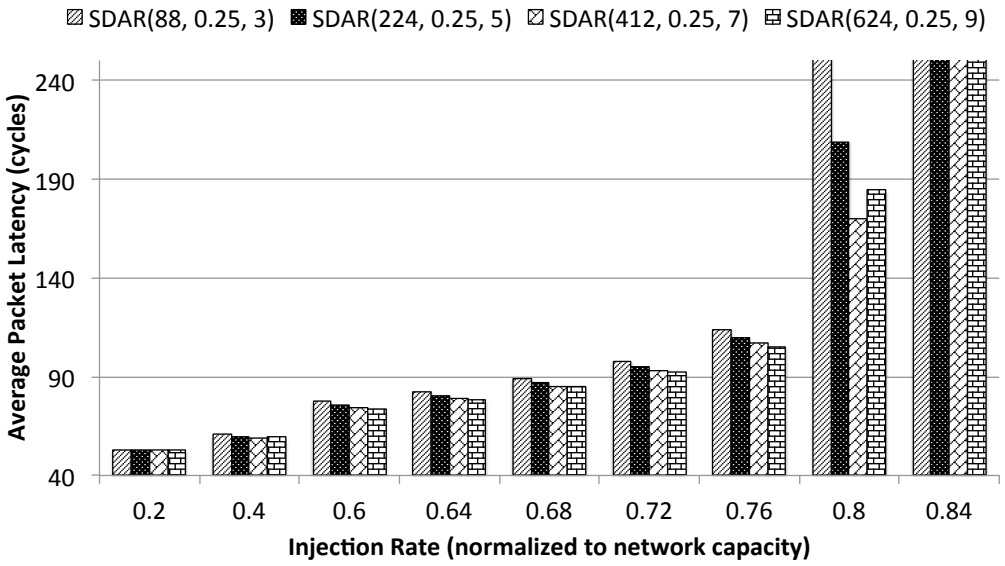


Figure 4.19: Performance sensitivity of SDAR to lookahead window size under uniform traffic. The SDAR configurations are annotated as SDAR(T, λ, M), where M is the length of a side of the lookahead window.

saturation throughput, SDAR outperforms RCA and local adaptive routing algorithms under all three traffic traces. The saturation throughput of SDAR under transpose, shuffle and uniform traffic are 6%, 18%, and 5% higher, respectively, than RCA. These improvements are better than the improvements seen for the smaller 8×8 mesh topology, indicating that measuring congestion only along the relevant routing paths becomes more important as the network size increases. Although the improvements in saturation throughput may seem small, they can significantly reduce average packet latency during temporary bursts of congestion when the network is driven close to saturation, as seen in some of the SPLASH traces for the smaller mesh topology.

Next, we look at the effect of the lookahead window size on average packet latency under uniform traffic. Figure 4.19 presents the average packet latency of different SDAR configurations with lookahead window size ranging from a 3×3 window to a 9×9 window. A larger window size has greater congestion visibility but suffers from slower delay measurement and slower split ratio refresh rates compared to configurations with smaller windows, as indicated by the values of T for the different configurations. We derived the values of T based on the assumption that the width of the

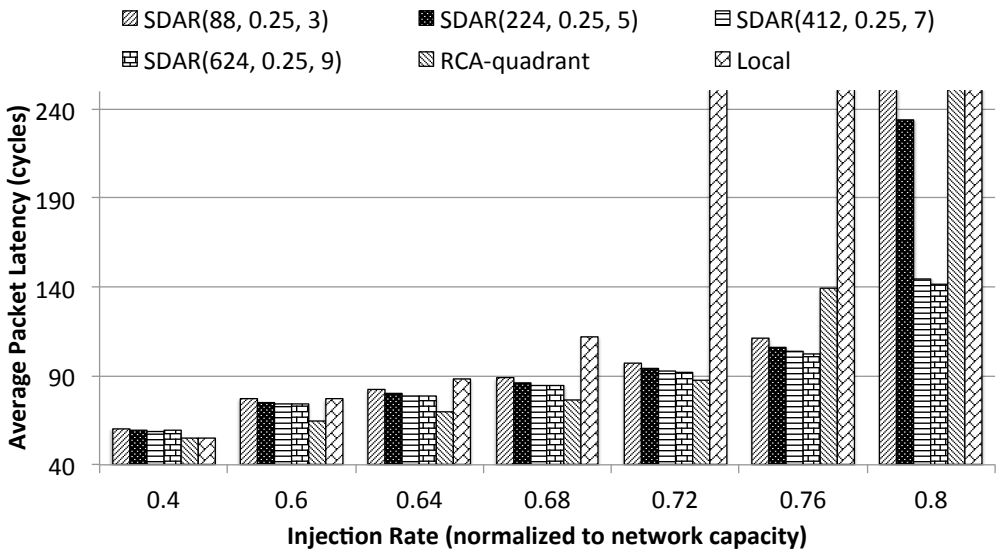


Figure 4.20: Performance of SDAR on uniform subset traffic. The SDAR configurations are annotated as SDAR(T, λ, M), where M is the length of a side of the lookahead window.

monitoring network is the same for all configurations. As expected, the average packet latency decreases with increase in window size, at least at moderate injection rates of 40-75% of network capacity. The magnitude of difference, however, is almost negligible between configurations with 5×5, 7×7, and 9×9 windows. Moreover, we observe that under high injection load of 80% capacity, the configuration with a 7×7 window slightly outperforms the configuration with a 9×9 window. This can be attributed to the lower value of T associated with the smaller window size. From these observations we can conclude that although a lookahead window has relatively low visibility compared to a 31×31 window required to have accurate delays to all nodes in a 16×16 mesh, as long as packets have enough room to route around newly detected congestion, the performance is not greatly affected.

Finally, to analyze the performance of SDAR on a broader spectrum of traffic traces, we generate *uniform subset* traffic patterns where every node sends packets to a randomly chosen subset of 16 nodes in the network. By mapping nodes to different subsets, we generate 20 different traffic patterns. Figure 4.20 compares the mean packet latency averaged over the 20 traces for SDAR (with different lookahead window sizes),

RCA and local adaptive routing. The latency bars are clipped at 250 cycles. We observe that SDAR configurations with 7×7 and 9×9 lookahead windows perform consistently well up to very high injection rates of 80% of capacity. The latency of SDAR under low and moderate loads is slightly higher than RCA, which benefits from having faster congestion updates compared to SDAR. However SDAR outperforms RCA significantly by 25-90% in terms of average packet latency under high injection rates. As mentioned earlier, during the course of an application running on a CMP, there may be temporary periods of congestion during which the network may be driven close to saturation. SDAR is better suited than existing routing algorithms to tackle such bursts, which can result in significant improvements in average communication latency over the entire duration of an application, as was seen in the experiments with SPLASH benchmarks for the 7×7 mesh.

4.7 Conclusion

In this chapter, we proposed a new minimal destination-based adaptive routing algorithm called DAR for 2D mesh networks. DAR makes routing decisions on a per-destination basis using delay estimates along the minimal paths to the destination. With such fine-grained destination-based congestion state, DAR can sense global network congestion more accurately than existing adaptive routing algorithms, which only sense regional or local congestion. On SPLASH-2 benchmarks, DAR outperforms Regional Congestion Awareness (RCA) by up to 58% and local adaptive routing by up to 79% in average latency. We discuss the challenges in implementing DAR under tight on-chip power and area constraints and propose a scalable version of DAR called SDAR for large mesh topologies. We show that SDAR outperforms RCA and local adaptive routing under a wide range of synthetic workloads on a 16×16 mesh.

Chapter 4, in part, is a reprint of the material as it appears in ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2010, Rohit Sunkam Ramaujam, Bill Lin, “Destination-based Adaptive Routing on 2D Mesh Networks”. Chapter 4, in full, has been submitted for publication of material as it may appear in ACM Transactions in Embedded Computing Systems, Rohit Sunkam

Ramanujam, Bill Lin, “Destination-Based Congestion Awareness for Adaptive Routing in 2D Mesh Networks”. The dissertation author was the primary investigator and author of the papers.

Chapter 5

Extending the Effective Throughput of NoCs with Distributed Shared-Buffer Routers

5.1 Introduction

In the design of NoCs, high throughput and low latency are both important design parameters and the router microarchitecture plays a vital role in achieving these performance goals. High throughput routers allow an NoC to satisfy the communication needs of multi- and many-core applications, or the higher achievable throughput can be traded off for power savings with fewer resources being used to attain a target bandwidth. Further, achieving high throughput is also critical from a delay perspective for applications with heavy communication workloads because queueing delays grow rapidly as the network approaches saturation.

A router's role lies in efficiently multiplexing packets onto the network links. Router buffering is used to house arriving flits that cannot be immediately forwarded to the output links due to contention. This buffering can be done either at the inputs or the outputs of a router, corresponding to an input-buffered router (IBR) or an output-buffered router (OBR). OBRs are attractive for NoCs because they can sustain higher throughputs and have lower queueing delays than IBRs under high loads. However,

a direct implementation of an OBR requires each router to operate at a speedup of P, where P is the number of router ports. This can either be realized with the router being clocked at P times the link clock frequency, or the router having P times more internal buffer and crossbar ports. Both of these approaches are prohibitive given the aggressive design goals of most NoC applications, such as high-performance CMPs. This is a key reason behind the broad adoption of IBR microarchitectures as the preferred design choice and the extensive prior effort in the computer architecture community on aggressively pipelined IBR designs.

In this chapter, we propose a new router microarchitecture that aims to emulate an OBR without the need for any router speedup. It is based on a distributed shared-buffer (DSB) router architecture that has been successfully used in high-performance Internet packet routers [25, 50]. Rather than buffering data at the output ports, a DSB router uses two crossbar stages with buffering sandwiched in between. These buffers are referred to as middle memories. To emulate the first-come-first-served (FCFS) order of an output-buffered router, incoming packets are timestamped with the same departure times as they would depart in an OBR. Packets are then assigned to one of the middle memory buffers with two constraints. First, packets that are arriving at the same time must be assigned to different middle memories. Second, an incoming packet cannot be assigned to a middle memory that already holds a packet with the same departure time[1]. It has been shown in [25, 50] that a DSB router can emulate a FCFS output-buffered router if unlimited buffering is available.

However, just as the design objectives and constraints for an on-chip IBR are quite different from those for an Internet packet router, the architecture tradeoffs and design constraints for an on-chip DSB router are also quite different. First, limited power and area budgets restrict a practical router microarchitecture implementation to contain small amounts of buffering. It is therefore imperative to explore power- and area-efficient DSB configurations suitable for on-chip design. Next, a flow-control protocol which can work with few buffers is necessary since NoC applications and protocols, such as cache coherency, cannot tolerate dropping of packets. A suitable flow-control mechanism is also needed to support a wide range of delay-sensitive applications with

[1]This is necessary to avoid switch contention.

ultra-low latency requirements. This can be achieved with flit-level flow control that allows allocation of network resources at the granularity of flits. This is different from internet routers which typically employ store-and-forward packet-level flow control. Finally, another key difference is the need for on-chip routers to operate at aggressive clock frequencies. This can be achieved with efficient router pipelining where circuit delay and complexity are balanced across all router pipeline stages. Our proposed router microarchitecture tackles all these challenges with appropriate solutions and new designs.

Our evaluation shows that the proposed on-chip DSB router achieves up to 19% higher saturation throughput in comparison to a state-of-the-art pipelined IBR and up to 94% of the ideal saturation throughput for the synthetic traffic workloads evaluated. On the set of SPLASH-2 benchmarks [71] that exhibit high contention and demand high bandwidth, our results further show that the proposed DSB router reduces packet latency by 61% on average when compared with IBRs.

The remainder of this chapter is organized as follows. Section 5.2 provides background information on throughput analysis and on existing router architectures. Section 5.3 describes our proposed distributed shared-buffer router microarchitecture for NoCs. Next, Section 5.4 provides extensive throughput and latency evaluations of our proposed DSB architecture using a detailed cycle-accurate simulator on a range of synthetic network traces and traffic traces gathered from real system simulations, while Section 5.5 evaluates the power and area overhead of DSB routers. Section 5.6 reviews related work. Finally, Section 5.7 concludes the chapter.

5.2 Background

In this section, we first provide a brief background on throughput analysis. We then present a short description of OBR and IBR microarchitectures, focusing on their deficiencies in practically attaining ideal throughput, before discussing distributed-shared-buffer Internet routers and how they mimic output buffering [25, 50].

5.2.1 Throughput analysis

Here, we provide a brief overview of the analysis techniques used to evaluate ideal network throughput. In particular, we elaborate on the concepts of network capacity, channel load, and saturation throughput. These concepts are intended to capture what could be *ideally* achieved for a routing algorithm R on a given traffic pattern Λ. To decouple the effects of the router microarchitecture, including buffer sizing and the flow control mechanism being used, ideal throughput analysis is based on channel load analysis. We first review the concept of network capacity.

Network capacity: Network capacity is defined by the maximum channel load, γ^*, that a channel at the bisection of the network needs to sustain under uniformly distributed traffic. As shown in [12], for any $k \times k$ mesh,

$$\gamma^* = \begin{cases} \frac{k}{4} & \text{for even } k \\ \frac{k^2-1}{4k} & \text{for odd } k \end{cases}$$

The network capacity, $\mathscr{N}$, in flits per node per cycle is then defined as the inverse of γ^*:

$$\mathscr{N} = \frac{1}{\gamma^*} = \begin{cases} \frac{4}{k} & \text{for even } k \\ \frac{4k}{k^2-1} & \text{for odd } k \end{cases}$$

For example, for a $k \times k$ mesh, with $k = 8$, $\mathscr{N} = 4/8 = 0.5$ flits/node/cycle. Next, we review the concept of saturation throughput.

Saturation throughput: For a routing algorithm R and a given traffic pattern Λ, the expected *channel load* on a channel c is denoted as $\gamma_c(R,\Lambda)$. The *normalized worst-case channel load*, $\gamma_{wc}(R,\Lambda)$, is then defined as the expected number of flits crossing the most heavily loaded channel, normalized to γ^*.

$$\gamma_{wc}(R,\Lambda) = \frac{\max_{c \in C} \gamma_c(R,\Lambda)}{\gamma^*}$$

where C is the set of all channels in the network.

Given this definition of normalized worst-case channel load, the *saturation throughput* corresponds to the average number of flits that can be injected per cycle by all the nodes in the network so as to saturate the most heavily loaded channel to its unit capacity. This is given as:

$$\Theta(R,\Lambda) = \frac{1}{\gamma_{wc}(R,\Lambda)}$$

Table 5.1: Traffic patterns and their corresponding ideal saturation throughput under dimension-ordered XY routing.

Traffic	Description	Saturation throughput
Uniform	Destination chosen at random, uniformly	1.0
Tornado	(x,y) to $((x+\lceil \frac{k}{2} \rceil - 1)\%k,\ (y+\lceil \frac{k}{2} \rceil - 1)\%k)$	0.66
Complement	(x,y) to $(k-x-1,\ k-y-1)$	0.5

Saturation throughput is defined specifically for a given routing algorithm R and traffic pattern Λ. Table 5.1 shows a few commonly used traffic patterns and their corresponding saturation throughput under dimension-ordered XY routing (DOR-XY). Note that 100% capacity cannot always be achieved with DOR-XY routing even under an *ideal* router design, defined as the one that can handle injection loads up to the saturation throughput. For example, for an adversarial traffic pattern like bit-complement traffic, it is well-known that DOR-XY routing saturates at 50% of network capacity. To decouple the effects of the routing algorithm on network performance, we assume DOR-XY routing throughout the remainder of this chapter. The goal of our router design is to reach the ideal router performance and thus approach the achievable saturation throughput.

5.2.2 Output-buffered routers

Fact 1. *An OBR with unlimited buffering can achieve the theoretical saturation throughput.*

Fact 2. *OBRs with unlimited buffering have predictable and bounded packet delays when the network is below saturation.*

Emulating the first-come, first-served (FCFS) behavior of OBR architectures is important for exploiting their attractive high-throughput and low-latency properties. Throughput guarantees offered by all oblivious routing algorithms [12], which are often used in NoCs because of their simplicity, assume ideal output-buffered routing with infinite buffers when performing throughput analysis. When the network topology

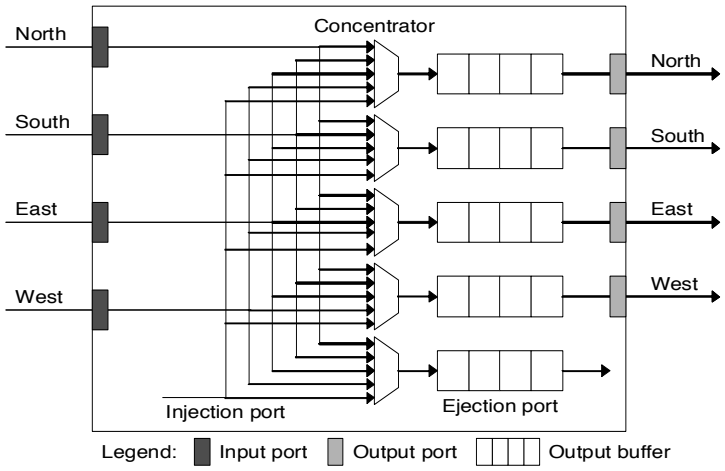


Figure 5.1: Output-buffered router (OBR) architecture model.

and the traffic matrix are both known, the saturation throughput for oblivious routing algorithms can be computed based on worst-case channel load analysis (as described in Section 5.2.1). Even when no information about the spatial characteristics of the traffic is available, which is often the case, worst-case throughput guarantees of oblivious routing functions can be provided by solving bipartite maximum-weight matching problems for each channel [64]. These throughput guarantees do not hold if the routers used do not emulate an OBR. Generally, using IBRs, the worst-case saturation throughput of an oblivious routing algorithm can be quite far off from the value predicted by worst-case throughput analysis (Figure 5.3). So one key advantage of OBR emulation is to provide and retain such guarantees with the limited hardware resources available in on-chip routers.

OBRs also have lower and more predictable queueing delays than IBRs because of their FCFS servicing scheme. Flits are not delayed in OBRs unless the delay is unavoidable due to multiple flits arriving at the same time at different input ports destined for the same output port. On the other hand, the switch arbitration schemes used in IBRs for multiplexing packets onto links are sub-optimal and result in unpredictable packet delays. The predictability of packet delays is an important concern for delay-sensitive NoC applications and OBR emulation is a step forward in this direction.

Figure 5.1 depicts the OBR architecture. In this architecture, incoming flits are directly written into the output buffers through a concentrator. Since up to P flits may

arrive together for a particular output in the same cycle, a direct implementation of an OBR would require a router speedup of P, where P is the number of router ports (i.e., $P = 5$ in Figure 5.1). Router speedup can be realized in two ways. First, by clocking the router at P times the link frequency, which is highly impractical with today's aggressive clock rates. Even if realizable, this will lead to exorbitant power consumption. Second, it can be realized with higher internal port counts at the buffers and the crossbar: each output buffer needs P write ports, with input ports connected to the output buffers through a $P \times P^2$ crossbar. This scenario leads to huge CMOS area penalties. High power and area requirements for OBR implementation are the key reasons behind the broad adoption of IBR microarchitectures as the principal design choice and the extensive prior effort in the computer architecture community on aggressively pipelined IBR designs (see Section 5.6), despite the very attractive property that an OBR can theoretically reach the ideal saturation throughput. Subsequently, in Section 5.2.4 we review a distributed shared-buffer (DSB) router architecture that emulates an OBR, inheriting its elegant theoretical properties, without the need for P times router speedup. We first review the IBR microarchitecture that is widely used in on-chip interconnection networks.

5.2.3 Input-buffered router microarchitecture

Figure 5.2-A sketches a typical input-buffered router (IBR) microarchitecture [12] that is governed by virtual-channel flow control [13]. We adopt this as the baseline input-buffered router for comparisons with our DSB architecture. The router has P ports, where P depends on the dimension of the topology. In a 2-dimensional mesh, $P = 5$, which includes the 4 North, South, East, West ports and the injection/ejection port from/to the processor core. At each input port, buffers are organized as separate FIFO queues, one for each virtual channel (VC). Flits entering the router are placed in one of these queues depending on their VC ID. All VC queues at an input port typically share a single crossbar input port, as shown in Figure 5.2-A, with crossbar output ports directly connected to output links that interconnect the current (upstream) router with its neighboring (downstream) router.

Figure 5.2-B shows the corresponding IBR router pipeline. The router uses look-

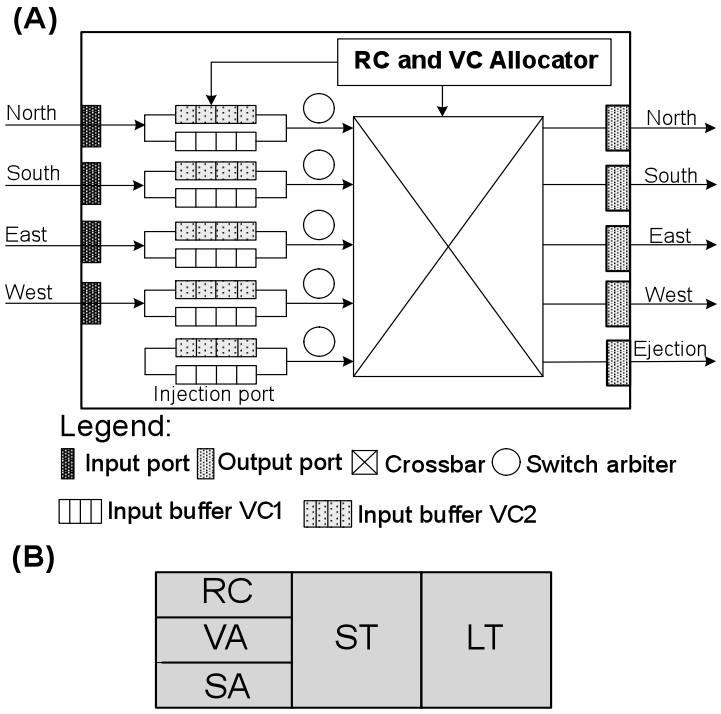


Figure 5.2: (A) A typical input-buffered router (IBR) microarchitecture with virtual-channel flow control and 2 virtual channels (VCs), and (B) its 3-stage pipeline: (1) route computation (RC) + virtual-channel allocation (VA) + switch arbitration (SA), (2) switch traversal (ST), and (3) link traversal (LT).

ahead routing and speculative switch allocation resulting in a short three-stage router pipeline. The route computation (RC) stage determines the output port based on the packet destination and is done one hop in advance to shorten the length of the router pipeline by executing RC concurrently with other router functions. Speculative switch allocation (SA) and VC allocation (VA) are also carried out in parallel in the first pipeline stage and priority is given to non-speculative switch requests to ensure that performance is not hurt by speculation. Once SA and VA are completed, flits traverse the crossbar (ST) before finally traversing the output link (LT) towards the downstream router. Head flits proceed through all pipeline stages while the body and tail flits skip the RC and VA stages and inherit the VC allocated to the head flit. The tail flit releases the reserved VC after departing the upstream router.

To attain high throughput, an IBR relies on several key microarchitectural

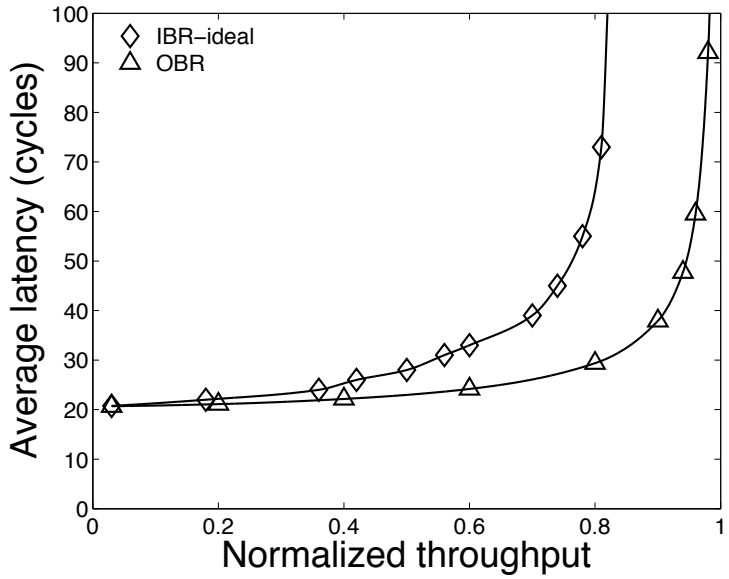


Figure 5.3: Latency-throughput curve of an input-buffered router with unlimited buffering, virtual channels and perfect allocation (Ford-Fulkerson matching algorithm [18]).

components to effectively multiplex flits onto the output links. First, additional buffering allows a greater number of flits to be housed at a router during high contention. This increases the number of competing flit candidates to eventually traverse a link towards a downstream router, while also alleviating congestion at upstream routers. Second, a higher number of VCs allows a greater number of individual packet flows to be accommodated, increasing buffer utilization and ultimately link utilization. Finally, the VC and switch allocators need to have good matching capability, i.e. they should ensure that VCs and crossbar switch ports never go idle when there are flits waiting for them.

To explore the throughput limits of an IBR, we simulated an 8×8 mesh network of IBRs with a pipelined microarchitecture as described above, but with unlimited buffering and VCs, and perfect matching (Ford-Fulkerson allocation [18]). Four-flit packets were sent uniformly to random destinations and routed using DOR-XY routing. Section 5.4 describes the detailed simulation methodology. Figure 5.3 compares the latency-throughput curve of the ideal IBR with the curve of an OBR having the same number of pipeline stages. Despite unlimited buffering, VCs and perfect matching,

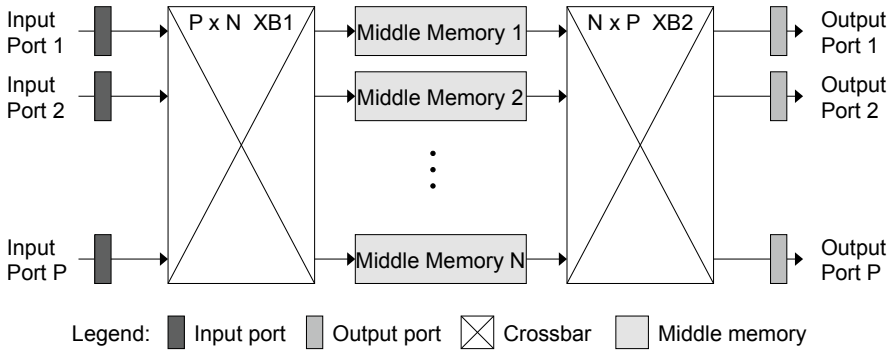


Figure 5.4: Distributed shared-buffer router architecture model.

all of which are *impractical* as they command impossibly huge area, power and delay overheads, the IBR obtained only about 81% of ideal saturation throughput. This is because of the bottlenecks that exist along the datapath of an IBR: buffers are partitioned per input port, and all VC queues at an input port share a single port into the crossbar. So when multiple packet flows at an input port wish to depart through different output ports, they need to queue up and leave the input buffer one at a time. Output-buffered routers precisely tackle this deficiency and can achieve near-ideal saturation throughput, as Figure 5.3 shows.

5.2.4 Distributed shared-buffer routers

DSB routers have been successfully used in high-performance Internet packet routers [25, 50] to emulate OBRs without any internal router speedup. Rather than buffering data directly at the output ports, a DSB router uses two crossbar stages with buffering sandwiched in between. Figure 5.4 depicts the DSB microarchitecture used in Internet routers. The input ports are connected via a $P \times N$ crossbar to N middle memories. These N middle memories are then connected to the output ports via a second $N \times P$ crossbar. At every cycle, one packet can be read from and written to each middle memory. It should be noted here that when contention occurs in Internet routers, packets can be dropped. This scenario is not allowed in our proposed on-chip DSB architecture (see Section 5.3).

To emulate the first-come, first-served (FCFS) packet servicing in OBRs, a DSB

router has to satisfy two conditions: (a) a packet is dropped by the DSB router if and only if it will also be dropped by an OBR, and (b) if a packet is not dropped, then the packet must depart the DSB router at the same cycle as the cycle in which it would have departed an OBR. To achieve this emulation, each packet arriving at a DSB router is *timestamped* with the cycle in which it would have departed from an OBR (i.e., in FCFS order). When a packet arrives, a scheduler chooses a middle memory to which to write this incoming packet and to appropriately configure the corresponding first crossbar (XB1). Also, at each cycle, packets whose timestamp is equal to the current cycle time are read from the middle memories and transferred to the outputs through the second crossbar (XB2).

In [25, 50], it was shown that a middle memory assignment can always be found and a DSB router can exactly emulate an OBR if the following condition is satisfied:

Fact 3. *A distributed shared-buffer router with $N \geq (2P - 1)$ middle memories and unlimited buffering in each middle memory can exactly emulate a FCFS OBR with unlimited buffering.*

At least $(2P-1)$ middle memories are needed to ensure that two types of conflicts can always be resolved: The first type of conflict is an *arrival conflict*. A packet has an arrival conflict with all packets that arrive simultaneously at other input ports since packets arriving at the same time cannot be written to the same middle memory. With P input ports, the maximum number of arrival conflicts a packet can have is $(P-1)$. The second type of conflict is a *departure conflict*. A packet has a departure conflict with all other packets that have the same timestamp and need to depart simultaneously through different output ports. With P output ports, a packet can have departure conflicts with at most $(P-1)$ other packets. Therefore, by the pigeonhole principle, $N \geq (2P-1)$ middle memories are necessary and sufficient to find a conflict-free middle memory assignment for all incoming packets.

5.3 Distributed shared-buffer (DSB) router for NoCs

In this section, we describe the details of the proposed DSB router for NoCs. In particular, Section 5.3.1 presents the key architectural contributions that differentiate

our DSB router for on-chip networks from DSB routers used for routing in the Internet. Section 5.3.2 then goes on to describe the router microarchitecture and pipeline. Finally, Section 5.3.3 discusses lower level details associated with practically implementing the router pipeline stages.

5.3.1 Key architectural contributions

The proposed DSB NoC router architecture addresses the bottlenecks that exist in the data path of IBRs which lead to lower than theoretical ideal throughput. At the same time, it tackles the inherent speedup limitations and area penalties of OBRs while harnessing their increased throughput capabilities. The middle memories decouple input virtual channel queueing from output channel bandwidth, as any flit can acquire any middle memory provided that there are no timing conflicts with other flits already stored in the same middle memory. Essentially, middle memories provide path diversity between the input and output ports within a router. Although based on the DSB architecture used in Internet routers, the proposed NoC router architecture faces a number of challenges specific to the on-chip domain.

First and foremost, NoC applications such as cache coherence protocols cannot tolerate dropping of packets unlike in Internet protocols. As a result, the DSB architecture used in Internet routers cannot be directly applied to the on-chip domain. To guarantee packet delivery, a flow control mechanism needs to be in place. The proposed DSB router uses credit-based flit-level flow control. To implement credit-based flow control, we introduce input buffers with virtual channels and distribute the available router buffers between the input ports and the middle memories. Flow control is applied on a flit-by-flit basis, advancing each flit from an input queue towards any time-compatible middle memory and ultimately to the output link. Flits are timestamped and placed into a compatible middle memory only when the next-hop router has buffers available at its corresponding input port. Further, since the middle memory buffering is limited due to power and area constraints, flits are held back in the input buffers when they fail to find a compatible middle memory.

Next, since power is of utmost importance in the NoC domain, the power-performance tradeoff of different DSB configurations needs to be explored. Although,

theoretically, $2P-1$ middle memories are needed in a P-ported router to avoid all possible arrival and departure conflicts, having a large number of middle memories increases power and area overheads by increasing the crossbar size. Therefore, we evaluate DSB configurations with fewer than $2P-1$ middle memories and estimate the impact of the reduced number of middle memories on performance.

Finally, on-chip routers need to operate at aggressive clock frequencies, pointing to the need for careful design of router pipelines with low complexity logic at each stage. Our design assumes a delay and complexity-balanced 5-stage pipeline. The proposed DSB architecture can achieve much higher performance than virtual-channel IBRs with comparable buffering while adding reasonable power and area overheads in managing middle memories and assigning timestamps to flits.

5.3.2 DSB microarchitecture and pipeline

Figure 5.5 shows the router microarchitecture of the proposed DSB router and its corresponding 5-stage pipeline. Incoming flits are first buffered in the input buffers which are segmented into several atomic virtual channels (VCs). The route computation stage (RC) employs look-ahead routing like the baseline IBR architecture, where the output port of a packet is computed based on the destination coordinates one hop in advance. Only the head flit of a packet participates in route computation. The remaining pipeline stages of a DSB router are substantially different from those of IBRs. Instead of arbitrating for free virtual channels (buffering) and passage through the crossbar switch (link), flits in a DSB router compete for two resources: middle memory buffers (buffering) and a unique time at which to depart from the middle memory to the output port (link).

The timestamping stage (TS) deals with the timestamp resource allocation. A timestamp refers to the future cycle at which a flit will be read from a middle memory, through the second crossbar, XB2, onto the output port. Timestamping is carried out in conjunction with lookahead routing. A flit (head, body, or tail) enters the TS stage and issues a request to the timestamper if it is at the head of a VC or if the flit ahead of it in the same VC has moved to the second stage of the pipeline. A flit can also re-enter the TS stage if it fails to find either a conflict-free middle memory or a free VC at the

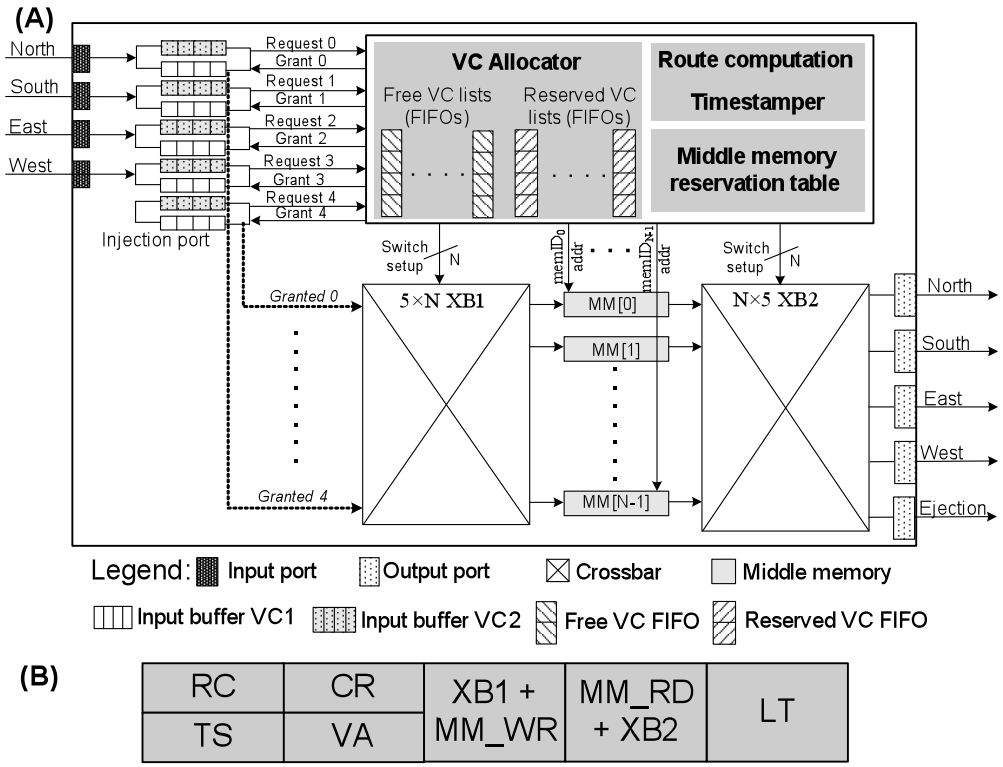


Figure 5.5: (A) Distributed shared-buffer router microarchitecture with N middle memories and (B) its 5-stage pipeline: (1) Route computation (RC) + timestamping (TS), (2) conflict resolution (CR) and virtual-channel allocation (VA), (3) first crossbar traversal (XB1) + middle memory write (MM_WR), (4) middle memory read (MM_RD) and second crossbar traversal (XB2), and (5) link traversal (LT).

input of the next-hop router, as will be explained later. If multiple VCs from the same input port send simultaneous requests to the timestamper, it picks a winning VC and assigns the earliest possible departure time for the output port requested by the flit in the selected VC. Let us assume that the output port requested is p. In order to find the earliest possible departure time through port p, the timestamper first computes the time the flit would leave the middle memory assuming there are no flits already stored in the middle memories that need to depart through port p. Let us denote this time as $T[p]$,

$$T[p] = \text{Current Time} + 3$$

since two pipeline stages, namely, CR+VA (Conflict resolution + VC allocation) and

XB1+MM_WR (Crossbar 1 and Middle Memory Write) lie between the TS and MM_RD + XB2 (Middle Memory Read and Crossbar 2) stages in the DSB pipeline (see Figure 5.5). Next, we consider the case when there are flits in the middle memories destined for output port p. To handle this case, the timestamper remembers the value of the last timestamp assigned for each output port until the previous cycle. The last assigned timestamp for output port p is denoted as $LAT[p]$. As timestamps are assigned in a strictly increasing order, the assigned timestamp for output port p in the current cycle must be greater than $LAT[p]$. In other words, a flit that is currently being timestamped can depart the middle memory through output port p only after all flits that are destined for the same output port and were timestamped at an earlier cycle, depart the middle memory. This emulates the FCFS servicing scheme of OBRs. Hence, the earliest timestamp that can be assigned to a flit for output port p is given as:

$$\text{Timestamp} = max(LAT[p]+1, T[p]) \tag{5.1}$$

If flits at more than one input port request a timestamp for output port p in the same cycle, the timestamper serves the inputs in an order of decreasing priority (either fixed or rotating). The timestamp of a flit at the highest priority input is computed as above and the remaining flits at other inputs are assigned sequentially increasing timestamps in the order of decreasing priority.

Conflict resolution (CR) and virtual-channel allocation (VA) comprise the second pipeline stage of the DSB router. The CR and VA operations are carried out in parallel. The task of the CR stage is to find a conflict-free middle memory for flits that were assigned timestamps in the TS stage. As mentioned earlier, there are two kinds of conflicts in shared-buffer routers – *arrival conflicts* and *departure conflicts*. Arrival conflicts are handled by assigning a different middle memory to every input port with timestamped flits. Departure conflicts are avoided by ensuring that the flits stored in the same middle memory have unique timestamps. Figures 5.6 and 5.7 illustrate how arrival and departure conflicts are resolved in a simplified DSB architecture with two input ports (no VCs), two output ports, and two middle memories, MM[0] and MM[1], each with a 3-flit capacity. Figure 5.6 shows how an arrival conflict is resolved by having two middle memories. In this example, two flits F_A and F_B need to depart through the same output port. They are timestamped at the same current cycle i. F_A is assigned a

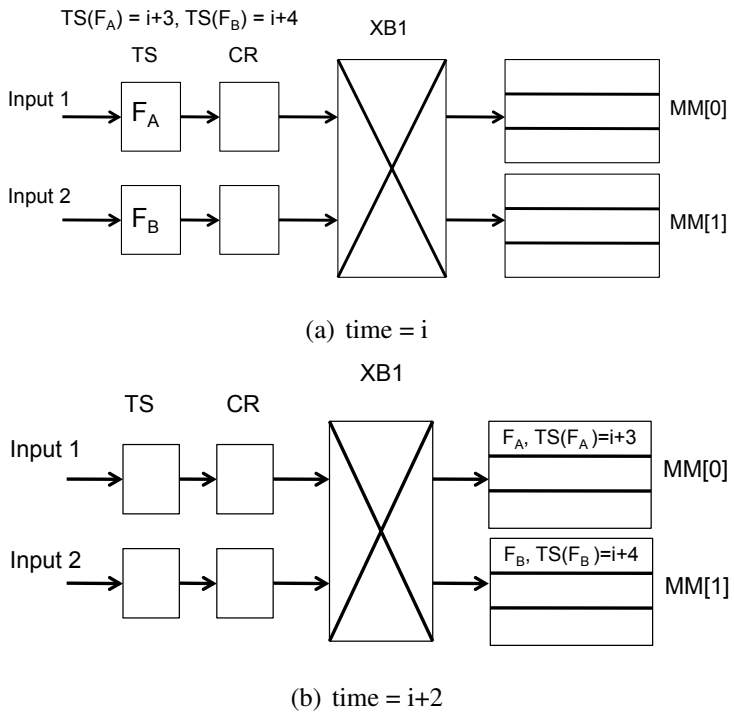


Figure 5.6: Arrival conflict resolution in a simplified DSB architecture.

timestamp of $i+3$ while F_B is assigned a timestamp of $i+4$, assuming input 1 has a higher priority than input 2 and there are no flits already stored in the middle memory buffers. Although F_A and F_B are in different input ports and have different departure times, they cannot be simultaneously written into the same middle memory, as each memory has a single write port. So F_A and F_B are assigned to MM[0] and MM[1], respectively. In Figure 5.7, a departure conflict is illustrated. Flit F_A enters the TS stage at current time i for output port 1 and is assigned a timestamp of $i+3$ since there are no flits in any middle memory destined for output port 1. When it enters the CR stage in the next cycle, there are two flits already stored in MM[0] having timestamps $i+2$ and $i+3$ destined for output port 2. Although MM[0] has an empty slot to store an additional flit, F_A cannot be assigned to MM[0] since it has a conflicting departure time with a flit already stored in MM[0] and MM[0] has one read port that can read out only a single flit at a given time. So, flit F_A has to be assigned to MM[1]. From these examples, it can be seen that conflicts are caused because middle memories are uni-ported[2] and only

[2]Single-ported memories are power and area-efficient.

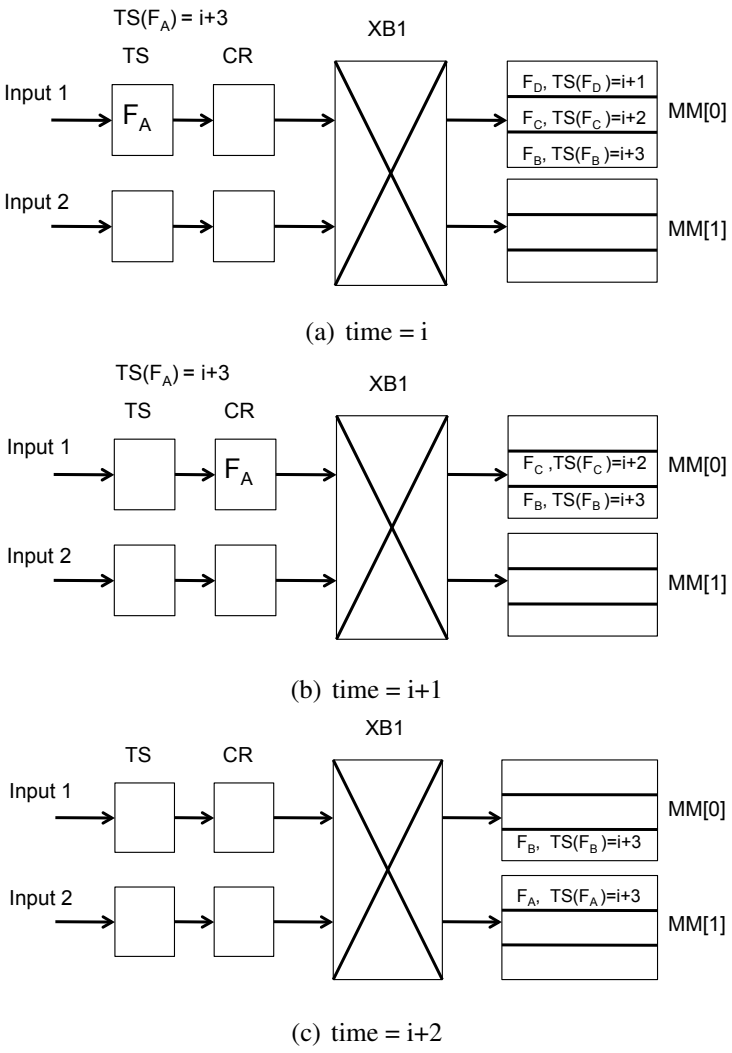


Figure 5.7: Departure conflict resolution in a simplified DSB architecture.

one flit can be written into (via XB1) and read from (via XB2) a middle memory in a given cycle. The implementation details of the TS and CR stages are explained in the next section.

The virtual-channel allocator arbitrates for free virtual channels at the input port of the next-hop router in parallel with conflict resolution. VC allocation is done only for the head flit of a packet. The VC allocator maintains two lists of VCs – a *reserved* VC pool and a *free* VC pool. VC allocation is done by picking the next free output VC from the *free* VC list of the given output port, similar to the technique used in [36]. Additionally, when output VCs are freed, their VC number is moved from the *reserved*

VC list to the end of the *free* VC list. If a free VC exists and the flit is granted a middle memory, it subsequently proceeds to the third pipeline stage, where it traverses the first crossbar (XB1) and is written to its assigned middle memory (MEM_WR). If no free VC exists (all VCs belong to the reserved VC list), or if CR fails to find a conflict-free middle memory, the flit has to be re-assigned a new timestamp and it therefore re-enters the TS stage.

When the timestamp of a flit matches the current router time, the flit is read from the middle memory (MM_RD) and passes through the second crossbar (XB2) in the fourth pipeline stage. We assume that the output port information is added to every flit and stored along with it in the middle memory. Finally, in the link traversal (LT) stage, flits traverse the output links to reach the downstream router.

5.3.3 Practical implementation

In this section, we describe the implementation details of the timestamping and conflict resolution stages, which are unique to the proposed DSB architecture. It must be noted here that the proposed implementation is only one among a range of possible design implementation choices that span a spectrum of area/delay tradeoffs. We specifically focus on the implementation of a 5-ported 2D mesh router. However, our design can be extended to higher or lower radix routers.

The high level block diagrams of the logic used in the TS stage are shown in Figures 5.8(a)- 5.8(c). The five input ports are labeled from i_0 to i_4. First, as shown in Figure 5.8(a), when multiple VCs from the same input port send simultaneous requests to the timestamper, a winning VC is selected using a matrix arbiter. In addition to the timestamping requests, the arbiter also takes as input the size of the $free$ VC lists for the output ports requested by each of the flits in the TS stage. The free VC count is used to give priority to body and tail flits that have already acquired a VC at the next-hop router over head flits that are likely to fail in the VC allocation stage. This optimization avoids wasted cycles resulting from re-timestamping of flits.

After choosing a winning VC at each input port i, the output ports requested by the flits in the selected VCs, OP_i, are used to generate timestamp offsets for each of these flits, as shown in Figure 5.8(b). Timestamp offsets are required to assign sequentially

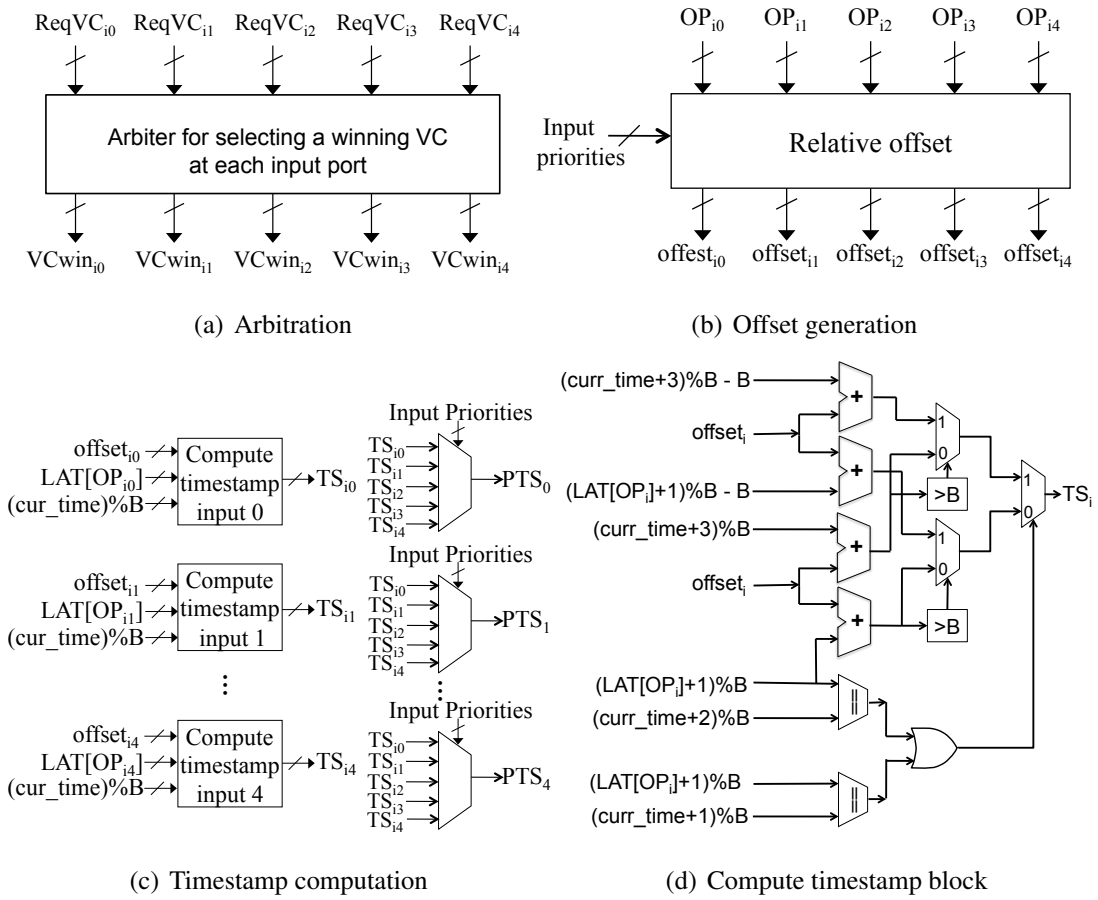


Figure 5.8: Block diagrams of the TS stage.

increasing timestamps to flits based on input priority when flits from more than one input port simultaneously request a timestamp for the same output port, as discussed in Section 5.3.2. Internally, the offset generator block comprises of separate sub-blocks, one for each output port. The offsets are generated on a per output port basis and the offset value for a flit is the number of higher priority flits requesting a timestamp for the same output port in the same cycle. The offset generation sub-block for an output port is a simple combinational function of the requests received from the input ports (OP_i) and the input priorities. The final timestamp assigned to a flit at input i is the sum of the timestamp assigned to the highest priority flit requesting the same output port (given by Equation 5.1) and the computed offset.

$$TS_i = max(LAT[OP_i] + 1, \text{Current time} + 3) + \text{offset}_i \tag{5.2}$$

In the DSB architecture, flits are stored in middle memories only after they have reserved buffering at the next-hop router. Let the total buffering at each input port be B flits. If the current time is denoted by *curr_time*, we restrict the maximum timestamp assigned for an output port to $curr\_time + B - 1$. This is because, assigning a timestamp equal to $curr\_time + B$ means that there are B flits before the current flit (with timestamps *curr_time* to $curr\_time + B - 1$) that have been timestamped for the same output port and have not yet departed the router. If all timestamped flits succeed in acquiring an output VC and a conflict-free middle memory, these B flits would reserve all available buffering at the input of the next-hop router and any flit with a timestamp greater than $curr\_time + B - 1$ would fail to reserve an output VC. Hence, assigning timestamps greater than $curr\_time + B - 1$ is not necessary. This fact is used to simplify the hardware for detecting departure conflicts. From this discussion, at most B unique timestamps are assigned for an output port, which can be represented using $\lceil \log_2 B \rceil$ bits. We ensure that each middle memory has exactly B flits of buffering so that a flit with timestamp T is always stored at the T^{th} location within the middle memory. In this way, a flit with timestamp T can only have departure conflicts with flits stored at the T^{th} location of any one of the N middle memories.

With timestamps represented using $\lceil \log_2 B \rceil$ bits, the timestamp assignment has to be carried out using *modulo-B* arithmetic. Under this scheme, the current time rolls over every B clock cycles, implemented using a *mod-B* counter. The assigned timestamps can take B unique values and also roll over beyond B. Hence, if $curr\_time\%B$ has a value t, flits stored in the middle memories can have B unique timestamps between t and $(t-1)\%B$, representing times from *curr_time* to $curr\_time + B - 1$. The last assigned timestamp for an output port can fall behind *curr_time* when the output port is not used for a while. If the last assigned timestamp for output port OP falls behind $curr\_time\%B$ (i.e. $LAT[OP] = curr\_time\%B$), it is advanced along with the current time to ensure that the last assigned timestamp is always either equal to or ahead of $curr\_time\%B$. This prevents old values of $LAT[OP]$ from appearing as future timestamps after rollover. Figure 5.8(d) presents the logic diagram for the timestamp computation block shown in Figure 5.8(c). When assigning a timestamp for output port OP_i, $(LAT[OP_i] + 1)\%B$ is simultaneously compared to $(curr\_time + 1)\%B$

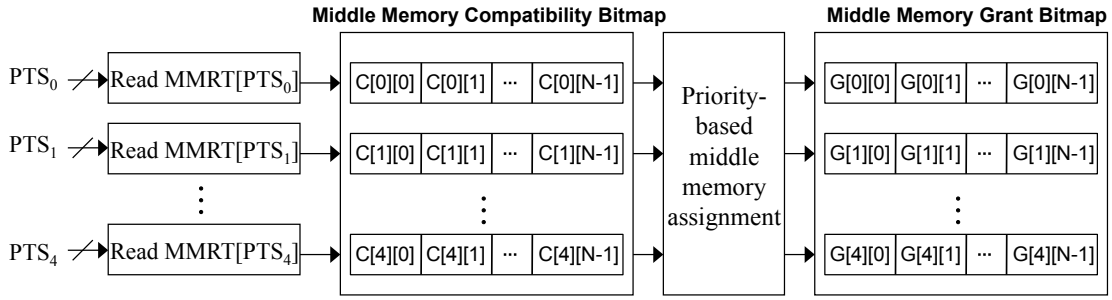


Figure 5.9: Block diagram of the CR stage.

and $(curr\_time + 2)\%B$, and the results are *ORed* together. A logic 1 at the output of the *OR* gate signifies that $(LAT[OP_i] + 1)\%B$ is behind $(curr\_time + 3)\%B$ and vice-versa. The greater of the two times is chosen and the corresponding flit offset is added to obtain the final timestamp according to Equation 5.2. If the timestamp computed using Equation 5.2 is greater than B, it is rolled over by subtracting B from the result, as shown.

In the last block of Figure 5.8(c), the timestamps are shuffled according to input priority, which is assumed to be a rotating priority over all inputs. In this respect, PTS_0 is the timestamp of the input with priority 0, PTS_1 is the timestamp of input with priority 1, and so on. This helps with the priority-based middle memory assignment during the CR stage. If an input does not hold a flit that needs to be timestamped, an invalid timestamp value is stored instead.

The task of the conflict resolution stage (CR) is to detect arrival and departure conflicts. To keep track of the occupancy of the middle memory buffers, we use an auxiliary data structure called the middle memory reservation table (MMRT). For N middle memories, with B flits of buffering per middle memory, the MMRT is an array of B registers, each N bits wide. The registers are indexed from 0 to $B-1$. If bit $MMRT[i][j]$ is set, it implies that memory bank j holds a flit with timestamp i and vice versa. Departure conflicts are resolved using the middle memory reservation table. For each timestamp that needs to be assigned a middle memory (PTS_0 ... PTS_4), the MMRT register indexed by the timestamp represents the middle memory compatibility bitmap for the timestamp. In Figure 5.9, the bits $C[i][0]$ to $C[i][N-1]$ represent the individual bits of the N-bit register, $MMRT[PTS_i]$. If bit $C[i][j]$ is 1, it means that middle memory

j already has a flit with timestamp PTS_i and hence, has a departure conflict with any flit with this timestamp. On the other hand, if $C[i][j]$ is 0, the flit with timestamp PTS_i is compatible with middle memory j. If an input does not have a flit that needs to be timestamped, the compatibility bits for all middle memories are set to 1 (meaning incompatible).

Next, arrival conflicts are resolved in the middle memory assignment stage. The middle memories are assigned fixed priorities with memory $N-1$ given the highest priority and memory 0 the lowest priority. In the middle memory assignment stage, the inputs are granted the highest priority compatible middle memory in the order of decreasing input priority while ensuring that more than one input is not granted the same middle memory. Bit $G[i][j]$ denotes the grant bit and it is set to 1 if the input with priority i has been granted middle memory j. This memory assignment scheme was specifically designed to have low middle memory miss rates when the number of middle memories is fewer than $2P-1$ (P being the number of ports) for 5-ported mesh routers. Having less than $2P-1$ middle memories is necessary to reduce the power and area of DSB routers as shown in Section 5.5. When the number of middle memories is at least $2P-1$, memory assignment schemes with less delay can be implemented as it is much easier to find conflict-free middle memories.

The above logic distribution between the TS and CR stages was architected to even-out the Fan-Out-of-4 (FO4) delays across the four stages (excluding LT) of the DSB pipeline. The FO4 calculations were carried out using the method of Logical Effort [62], and was applied to each logic block. For a 5-ported DSB router with 5 VCs per input port, 4 flits per VC ($B = 20$ flits) and 5 middle memories with 20 flits per middle memory, the critical path delays of the TS and CR pipeline stages were estimated at 19 FO4s and 18 FO4s, respectively. A delay of less than 20 FO4 for each stage in the proposed architecture enables an aggressively-clocked high-performance implementation. In particular, assuming a FO4 delay of 15ps for Intel's 65nm process technology, our proposed design can be clocked at a frequency of more than 3GHz.

5.4 Throughput and latency evaluation

5.4.1 Simulation setup

To evaluate the effectiveness of our proposed DSB router against a baseline input-buffered router (IBR) architecture with virtual-channel (VC) flow control, we implemented two corresponding cycle-accurate flit-level simulators. The baseline IBR simulator has a three-stage pipeline as described in Section 5.2.3. The DSB simulator models the five-stage router pipeline described in Section 5.3.2. Both simulators support k-ary 2-mesh topologies with their corresponding 5-ported routers. DOR-XY routing is used for all our simulations where packets are first routed in the X-dimension followed by the Y-dimension. We use DOR-XY because our main focus is on highlighting the improvement in performance due to the DSB router architecture, rather than the routing algorithm.

We present results for both synthetic and real traffic traces. The three synthetic traffic traces used are `uniform`, `complement` and `tornado` traffic, shown in Table 5.1. These three traces represent a mixture of benign and adversarial traffic patterns. The ideal saturation throughputs that can be achieved for these three traffic patterns using DOR-XY (based on channel load analysis) are also shown in Table 5.1. All throughput results presented subsequently are normalized to the ideal saturation throughput for the given traffic pattern. An 8×8 mesh topology is used for our simulations with synthetic traffic. Multi-flit packets composed of four 32-bit flits are injected into the network and the performance metric considered is the average packet latency under different traffic loads (packet injection rates). The latency of a packet is measured as the difference between the time the head flit is injected into the network and the time the tail flit is ejected at the destination router. The simulations are carried out for a duration of 1 million cycles and a warm-up period of ten thousand cycles is used to stabilize average queue lengths before performance metrics are monitored.

In addition to the synthetic traces, we also compare the performance of the two router architectures on eight traffic traces from the SPLASH-2 benchmark suite [71]. The traces used are for a 49-node shared memory CMP [38] arranged as a 7×7 mesh. The SPLASH-2 traces were gathered by running the corresponding benchmarks with 49

Table 5.2: Nomenclature of the router microarchitectures compared.

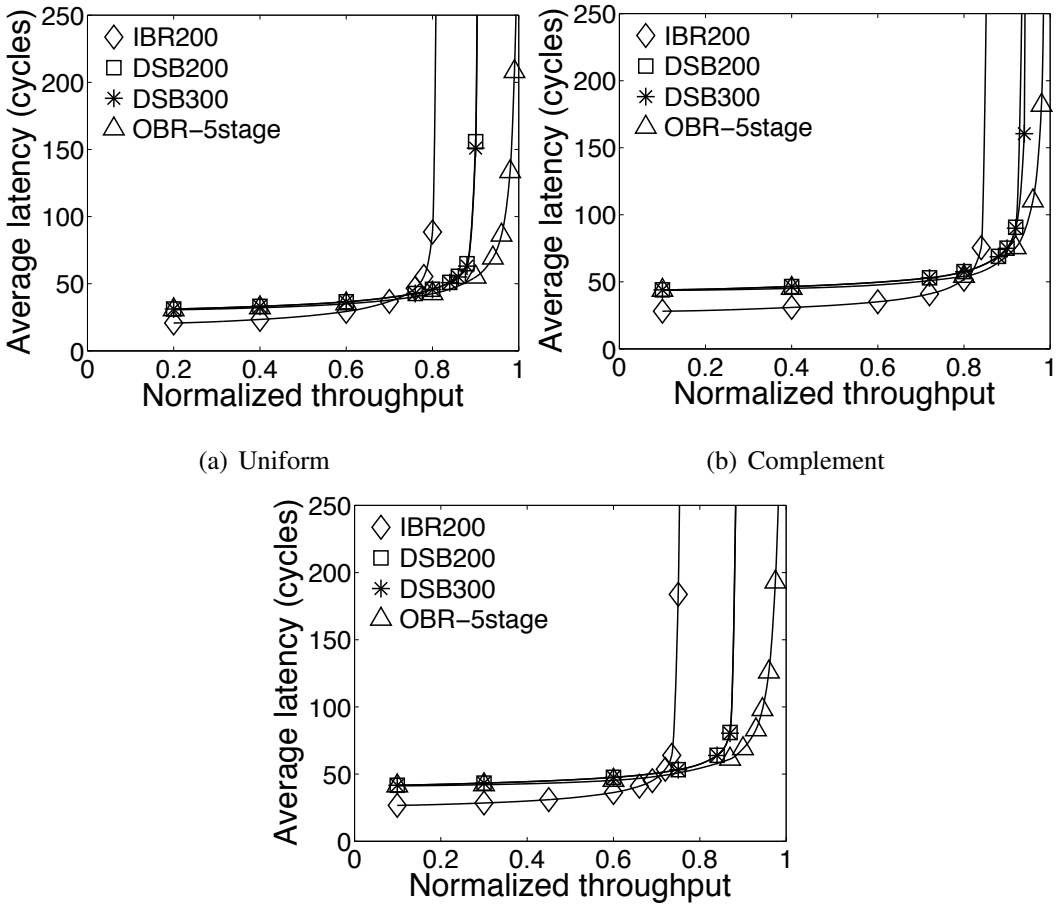
Config.	Input buffers (per port)		Middle memory buffers		Total buffers
	#VCs	#Flits/VC	#Middle Memories (MM)	Flits/MM	
IBR160	8	4	-	-	160
IBR200	8	5	-	-	200
IBR240	8	6	-	-	240
DSB160	4	4	5	16	160
DSB200	5	4	5	20	200
DSB240	6	4	5	24	240
DSB300	5	4	10	20	300

threads[3] on Bochs [39], a multiprocessor simulator with an embedded Linux 2.4 kernel. The memory trace was captured and fed to a memory system simulator that models the classic MSI (Modified, Shared, Invalid) directory-based cache coherence protocol, with the home directory nodes statically assigned based on the least significant bits of the tag, distributed across all processors in the chip. Each processor node has a two-level cache (2MB L2 cache per node) that interfaces with a network router and 4GB off-chip main memory. Access latency to the L2 cache is derived from CACTI to be six cycles, whereas off-chip main memory access delay is assumed to be 200 cycles. Simulations with SPLASH-2 traces are run for the entire duration of the trace, typically in the range of tens of millions of cycles, which is different for each trace.

5.4.2 Performance of the DSB router on synthetic traces

The nomenclature of IBR and DSB configurations referred to in this section is presented in Table 5.2, along with details of the buffer distribution in each configuration. For the DSB architecture, the number of flits per middle memory is always equal to the number of input buffers to simplify departure conflict resolution, as discussed in Section 5.3.3. Theoretically, a 5-ported DSB router needs 9 middle memories to avoid

[3]Two of the eight traces, `fft` and `radix` could not be run with 49 threads. They were run with 64 threads instead. The memory traces of the first 49 threads were captured and the addresses were mapped onto a 49-node system.



(a) Uniform

(b) Complement

(c) Tornado

Figure 5.10: Performance comparison of different router architectures under various synthetic traffic patterns. The X axis normalized to the ideal saturation throughput that can be achieved for a given traffic pattern.

all conflicts in the worst case, i.e., when all possible arrival and departure conflicts occur simultaneously. However, keeping the power overhead in mind, we evaluate a DSB configuration with only 5 middle memories (DSB200) and compare its performance to a configuration with 10 middle memories (DSB300).

In addition to the configurations shown in Table 5.2, we also simulated an OBR with a very large number of buffers (10,000-flit buffers) at each output port, emulating infinite output buffer capacities. Redundant pipeline stages are introduced in the OBR simulator totaling a pipeline depth of five, to ensure the same pipeline length as our

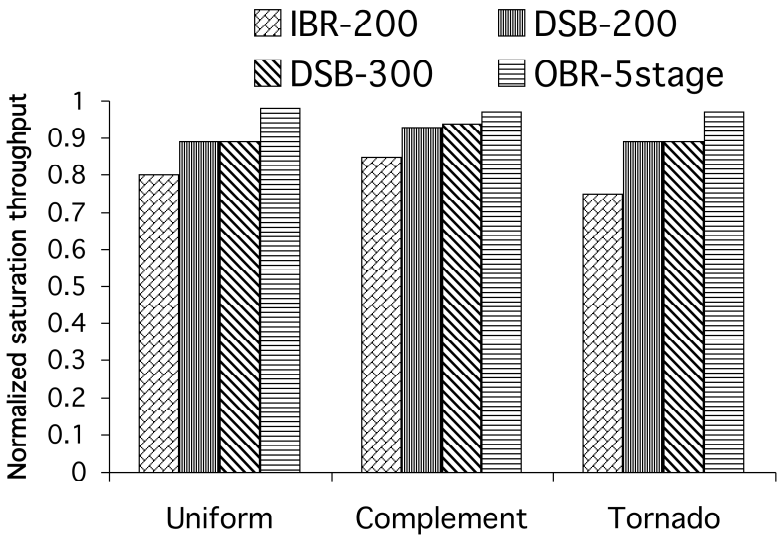


Figure 5.11: Normalized saturation throughput comparison for different NoC router microarchitectural configurations. The throughputs are normalized to the ideal saturation throughput that can be achieved for a given traffic pattern.

DSB router for fair comparisons. This helped us to compare the performance of DSB configurations with an OBR with the same number of pipeline stages, but which behaves ideally and incurs no delay due to switch arbitration and buffer capacity limitations. We refer to this OBR configuration as OBR-5stage.

We first compare the performance of IBR200, DSB200, DSB300 and OBR-5stage. IBR200 and DSB200 have the same number of buffers (200 flits). DSB300 has the same number of input buffers but double the number of middle memories compared to DSB200. The average packet latencies at different packet injection rates are shown in Figure 5.10 for the three synthetic traffic patterns and the saturation throughput values are presented in Figure 5.11. The saturation throughput is assumed to be the injection rate at which the average packet latency is three times the zero-load latency.

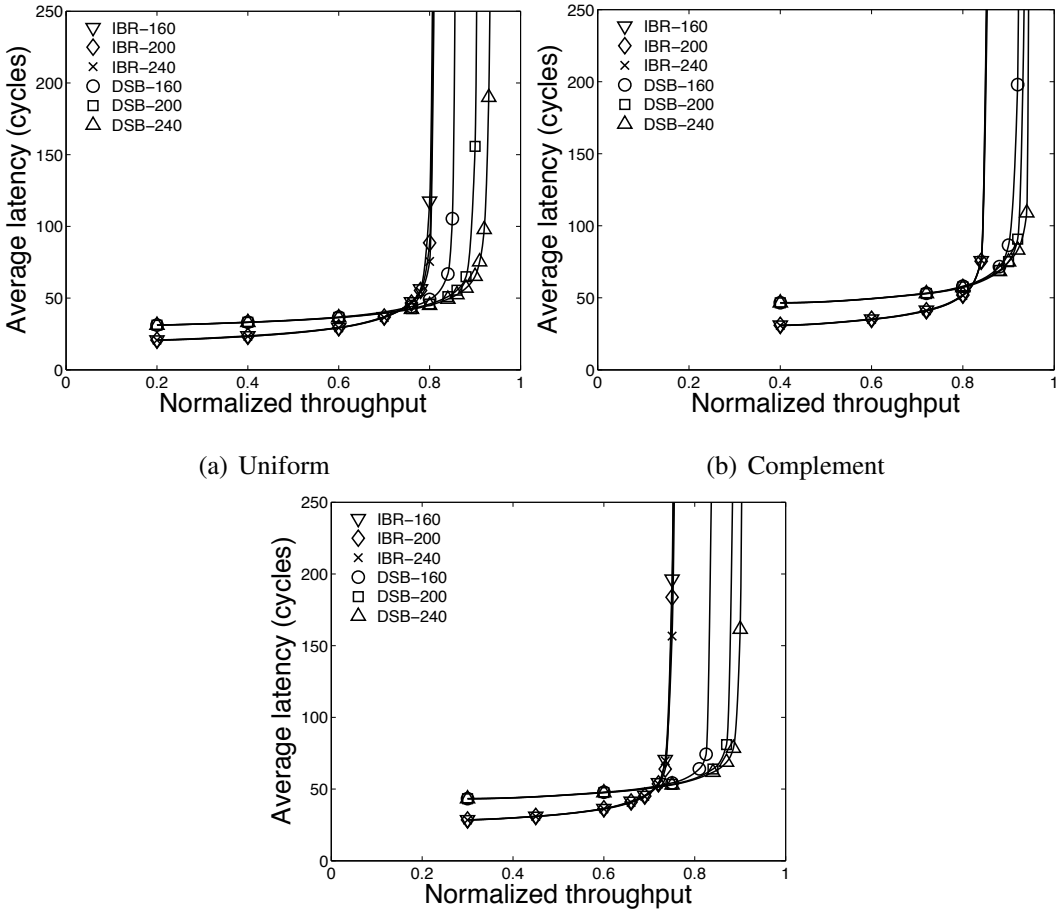
Figure 5.11 shows that with an aggregate buffering of 200 flits, DSB200 outperforms IBR200 by 11.25%, 9.5% and 18.5% on uniform, complement and tornado traffic, respectively, in terms of saturation throughput. Although IBR200 has a slightly lower latency than DSB200 under low loads due to the shorter router pipeline, the higher saturation throughput of DSB200 gives it a definite edge under moderate to

high loads. It must be noted here that during the course of an application running on a CMP, there may be transient periods of high traffic or localized traffic hotspots during which parts of the network are driven close to (or past) saturation. A router with higher saturation throughput can minimize the occurrence of such transient hotspots and provide tremendous latency savings during these periods, which can far outweigh the slightly higher latency observed under low loads. This is more clearly depicted in the SPLASH-2 results presented in Section 5.4.3.

The saturation throughput of DSB200 is also close to that of OBR-5stage. For uniform, tornado and complement traffic, the saturation throughput of the DSB200 architecture is within 9%, 4% and 8%, respectively, of the throughput of OBR-5stage. The slightly lower saturation throughput of DSB routers is a result of having far fewer buffers compared to OBR-5stage's infinite buffering.

The performance of DSB200 is nearly identical to DSB300, with negligible difference in saturation throughputs for all three traffic patterns. This is because the probability of more than five arrival and departure conflicts occurring simultaneously is very low. We observe in our experiments that even under very high injection loads, in the worst case, less than 0.3% of the flits failed to find a conflict-free middle memory over all traffic patterns. Hence, it can be concluded that although theoretically nine middle memories are needed to resolve all conflicts, in practice, five middle memories result in very little degradation in throughput. Fewer middle memories are very attractive from a power and area standpoint because they lead to smaller crossbars and fewer buffers.

Next, we explore the sensitivity of performance to the aggregate router buffering for both the IBR and DSB architectures. Figure 5.12 shows the average packet latency curves under different injection loads and Figure 5.13 compares the saturation throughput of IBR and DSB routers with 160, 200 and 240 flits of buffering for the three synthetic workloads. For both the IBR and DSB architectures, the partitioning of input buffers into VCs is done in such a way that the throughput is maximized. The IBR configurations show very little variation in saturation throughput with change in the number of buffers. However, for the case of DSB routers, we observe that the effective throughput can be improved by increasing the aggregate buffering. DSB240 has a saturation throughput of 92% for uniform traffic which is a 3.4% gain over



(a) Uniform

(b) Complement

(c) Tornado

Figure 5.12: Performance sensitivity of IBR and DSB architectures to total buffering under various synthetic traffic patterns.

DSB200. DSB200 in turn has higher throughput than DSB160 under all workloads. Therefore, while the performance of IBRs saturate beyond a certain number of buffers, the performance of DSB routers can be improved further at the expense of more buffering (power). This is because the performance of IBRs is limited by the efficiency of the switch arbitration stage. On the other hand, the throughput of DSB routers, which aim to emulate an OBR, approaches the ideal saturation throughput as buffering is increased. For the three synthetic traffic traces evaluated, the saturation throughput of DSB240 is within 7% of the saturation throughput of OBR-5stage. We also observe that DSB160, with a total buffering capacity of 160 flits can significantly outperform

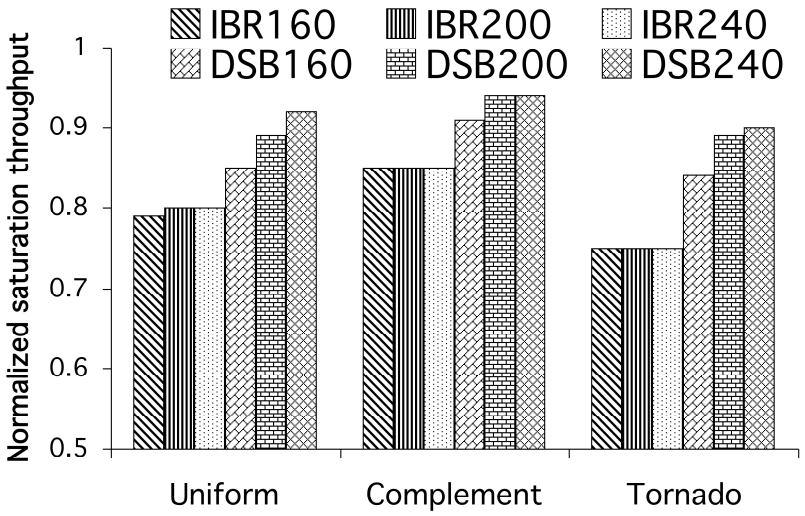


Figure 5.13: Sensitivity of saturation throughput to buffer size.

IBRs with more buffers. This indicates that if power consumption is critical, DSB configurations with fewer buffers can be used to minimize the router's power overhead.

5.4.3 Performance of the DSB router on real traffic traces

In this section, we present simulation results using the eight benchmark traces from the SPLASH-2 suite [71]. For uniform, tornado, and complement traffic, the traffic matrix Λ is assumed to be fixed and stationary. As discussed in Section 5.2.1, using channel load analysis, the ideal saturation throughput for these traffic patterns can be computed based on just Λ and the routing algorithm R. However, in the case of real traffic traces, like the SPLASH-2 traces, the traffic pattern is space- and time-varying. Therefore, the notion of ideal saturation throughput can neither be clearly defined nor easily determined. Instead, we compare the average packet latencies over the entire trace duration using our cycle-accurate flit-level simulators. The communication latency for memory accesses is directly proportional to the application performance on shared-memory many-core systems.

Figure 5.14 shows the latency results of all the IBR and DSB router configurations, normalized to the average packet latency of IBR200. The first observation we make is that the average packet latency of the DSB configurations is comparable

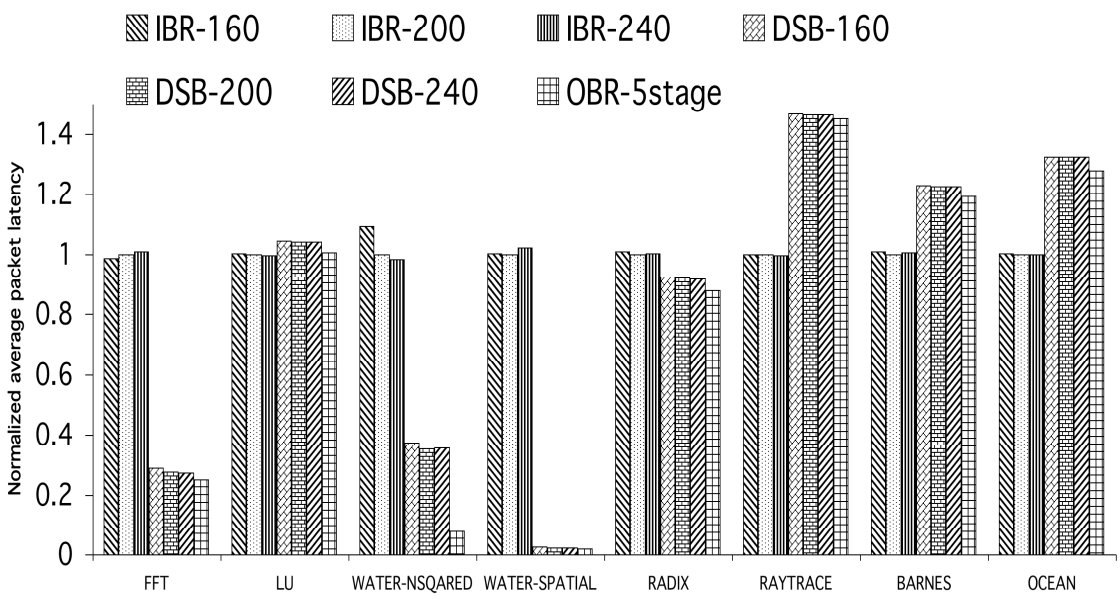


Figure 5.14: Network latency for SPLASH-2 benchmarks.

to OBR-5stage for seven of the eight SPLASH-2 benchmarks evaluated. For the `water-nsquared` trace, OBR-5stage has a lower latency than DSB routers because this trace has traffic hot spots where packets are injected at rates that saturate both OBR and DSB routers. In such cases, the large number of output buffers available in OBR-5stage help in attaining a lower average latency. On average, across all eight traces, OBR-5stage has 14% lower average packet latency than DSB-240. In most traces, the ability of a DSB router to closely match an OBR in terms of latency illustrates its ability to emulate OBRs, even with limited buffering.

Next, we observe that DSB200 outperforms IBR200 on `fft`, `water-nsquared`, `water-spatial` and `radix` traces by 72%, 64.5%, 97.5% and 8%, respectively. For the three traces where the performance improvement is over 50%, i.e., `fft`, `water-nsquared` and `water-spatial`, IBR200 saturates during portions of all three traces while DSB200 saturates only in the case of the `water-nsquared` trace. It is during these transient periods of congestion that the communication delay can shoot up and the network can become a serious performance bottleneck. The high relative packet delays of the IBR configurations seen in these three traces is expected to directly impact application performance as communication delays will constitute the bulk of

the application runtime. It can therefore be inferred from these results that a relatively small increase in saturation throughput translates into tremendous reductions in packet latency for applications that demand high bandwidth. IBR200, however, has 32%, 18% and 24% lower latency for `raytrace`, `barnes` and `ocean` benchmarks. This is because these traces have negligible output port contention and the higher delay of DSB routers can be attributed to their longer pipeline depth of 5 as compared to 3 for IBR200. This is proved by the fact that even OBR-5stage, which has no extra delay introduced as a result of switch contention, has higher average packet latency than IBR200. For the SPLASH-2 benchmarks with high output port contention (`fft`, `water-nsquared`, `water-spatial` and `radix`), on average, DSB200 has 61% lower latency than IBR200. Averaging across all eight traces, DSB200 has 17% lower latency than IBR 200. Hence, for applications that demand high throughput and drive the network towards saturation or close to saturation, DSB routers are clearly superior to IBRs.

Although DSB160 has lower saturation throughput compared to the DSB200 and DSB240 configurations under all three synthetic traffic traces, the difference in average packet latency in real application traces, like the SPLSH benchmarks, is less notable. The biggest difference in average packet latencies of DSB160 and DSB240 configurations is seen for the `water-spatial` trace, where DSB240 has 18% lower latency than DSB160. On average, the difference in average packet latency between the DSB160 and DS240 configurations across all eight traces is less than 4%. However, compared to IBR240 and IBR200 configurations with much higher aggregate buffering, DSB160 achieves 60% lower average packet latency for the four SPLASH-2 benchmarks with high contention.

Lastly, there is negligible difference in performance between DSB200 with 5 middle memories and DSB300 with 10 middle memories. This further shows that even with real application traces, more than five simultaneous arrival and departure conflicts occur very rarely. Hence, 5 middle memories are sufficient to achieve comparable performance to a 10-middle memory configuration, with the added advantage of significantly lower power and area overheads, which will be shown in the next section.

Table 5.3: Router power and area comparison with full-swing crossbars.

Router	Power						Area
	Buffer	Xbar	Arbiter	MM	Clock	Total	
IBR160	74	52.7	53.4	–	52	232	0.17
DSB160	53.2	105.4	48	30.4	70	307	0.28
IBR200	76.4	52.7	53.4	–	54.2	237	0.19
DSB200	58.4	105.4	58.7	34.6	71.5	328.6	0.3
IBR240	79.3	52.7	53.4	–	57	242.4	0.21
DSB240	63.5	105.4	68	38.9	73	348.8	0.32

Table 5.4: Power overhead of using DSB routers.

Router config.	Total power	Power penalty			
		Per router	Per tile, NoC power =		
	(mW)		10%	15%	20%
IBR160	232				
DSB160	307	1.32	1.032	1.048	1.064
IBR200	237				
DSB200	328.6	1.39	1.039	1.06	1.08
IBR240	242.4				
DSB240	348.8	1.44	1.044	1.066	1.088

5.5 Power and area evaluation

Table 5.3 compares the power consumption and area requirements of IBR and DSB router microarchitectures with the same aggregate buffering. We use the power and area models in Orion 2.0 [28, 69] for our analysis. The models use parameters from TSMC 65nm process libraries and include both dynamic and leakage power components. The operational frequency used is 3GHz at 1.2V. A typical flit arrival probability of 0.3 is assumed at each input port and the flit width is set to 32 bits. SRAMs are used as input and middle memory buffers and low threshold voltage transistors are used to ensure low delays for 3GHz operation. The VC allocator is configured to simply select a free VC from a free VC list as described in Section 5.3.2 in both DSB and IBR architectures while a matrix arbiter is used for switch arbitration in IBRs. The power and area of the arbitration logic of the DSB router were extrapolated from the power and

area numbers of the arbitration logic of an IBR, based on the the number of 2-input gates required to implement the logic blocks in the two architectures. The arbiter power of the DSB routers includes the power of all logic blocks needed to implement timestamping and conflict resolution, as described in Section 5.3.3.

As shown in Table 5.4, DSB160, DSB200 and DSB240 consume 32%, 39% and 44% more power and occupy 64%, 58% and 52% more area than an IBR router with the same amount of buffering. The higher power consumption and area requirements of the DSB router is due to the presence of an extra crossbar and a more complex arbitration scheme (involving timestamping and conflict resolution) compared to the switch arbiter in an IBR. The DSB160 configuration, which uses fewer buffers than IBR240, achieves significantly better performance on both synthetic and real traffic, but exhibits a 27% power and a 33% area overhead over IBR240.

Although the power cost per router for a DSB router is substantially greater than that of an IBR, the overall power increase for an NoC-based CMP application is often relatively smaller. In Table 5.4, the power penalty per tile (processor + router) of using a DSB router is presented for three different scenarios where the NoC consumes 10%, 15% and 20% of the total CMP tile power. Even for applications where the router consumes as high as 20% of the tile power, the power per tile with a DSB200 router is only 8% higher than the tile power with IBR200. On the other hand, if the router consumes only 10% of the tile power, the power per tile with a DSB200 router is just 3.9% higher than the tile power with IBR200. We believe that the increased power cost is justified for applications that demand high bandwidth and exhibit high contention since latency reductions of more than 60% can be achieved using DSB routers. As with power, in the case of area requirements, the increase in area of the entire tile as a result of using a DSB router in place of an IBR is again very low since the router area is only a small portion of the total tile area.

As discussed earlier, the DSB200 configuration with 5 middle memories and the DSB300 configuration with 10 middle memories exhibit similar performance. However, DSB200 consumes 37% less power and occupies 40% less area compared to DSB300 (Table 5.5). The power and area savings are achieved by using fewer buffers and two 5×5 crossbars in DSB200 instead of 5×10 and 10×5 crossbars used in DSB300.

Table 5.5: Comparison of DSB routers with 5 and 10 middle memories.

Router Config.	Power (mW)						Area (mm^2)
	Buffer	Xbar	Arbiter	MM	Clock	Total	
DSB200	58.4	105	58.7	34.6	71.5	329	0.3
DSB300	58.4	203	84	70.9	104	520	0.5

Table 5.6: Router power comparison of DSB and IBR architectures with low-swing crossbars.

Router config.	Power (mW)	Router Config.	Power (mW)	Penalty (per router)
IBR160	206	DSB160	254	1.23
IBR200	210	DSB200	276	1.31
IBR240	216	DSB240	296	1.37

Another technique with which the power overhead of DSB routers can be further reduced is by employing customized low-swing crossbars [35]. It has been shown in [35] that crossbar power can be reduced by approximately 60% compared to a baseline unoptimized design by using differential low-voltage signaling. As a conservative estimate, our extrapolated low-swing crossbar power model reduces the power of the 5×5 crossbar used in the DSB and IBR architectures by 50% compared to a full-swing crossbar. The power consumption of the two architectures with optimized low-power crossbars is presented in Table 5.6. The power penalty per router for DSB160, DSB200 and DSB240 routers can be reduced to 23%, 31% and 37%, respectively, compared to an IBR router with the same aggregate buffering.

5.6 Related work

5.6.1 On-chip routers

Most previous works focus on efficient buffer organization as a means of extending the effective throughput of NoCs. Sophisticated extensions to input-buffered router microarchitectures have been proposed for improving throughput, latency and

power. For throughput, techniques like flit-reservation flow control [49], variable allocation of virtual channels [8] and express virtual channels [38] have been proposed. As these designs are input-buffered, they are only able to multiplex arriving packets from their input ports across the crossbar switch, unlike our proposed DSB architecture which can shuffle incoming packet flows from all input ports onto the middle memories and then onto the crossbar switch. The DSB architecture offers better opportunities for packet multiplexing and improved packet flow, which helps in mimicking the high throughput and predictable delay characteristics of output-buffered routers. The Xpipes NoC architecture [5] targeted towards gigascale systems-on-chip uses output buffering without any switch speedup. The Xpipes switch implementation and flow control mainly target low latency rather than high-throughput, the latter being the primary objective of DSB routers.

There have been several input-buffered router proposals that target network latency, making single-cycle routers feasible, such as speculative allocation [44, 45, 48] and lookaheads [21, 36]. For power savings, techniques such as row-column separation [32] and segmentation of crossbars and straight-through buffers [68] have been proposed. These latency and power optimization techniques are orthogonal to our proposal as they do not target throughput. Some of these techniques can be applied to the DSB router as well to reduce network latency and energy consumption.

As discussed in Chapters 2, 3 and 4, another way to improve throughput in NoCs is by designing routing algorithms that balance the network load over all links. In general, any oblivious routing algorithm can be used with the DSB architecture as the route computation stage operates in the same way as in an IBR. Also, adaptive routing algorithms can be easily incorporated into the DSB architecture as follows: adaptive routing decisions are usually based on the next-hop input queue length as an indication of downstream congestion during output-port selection. Since the DSB architecture emulates an OBR, instead of input queue length, the corresponding output queue length can be used as an indicator of downstream congestion. The output queue length can be determined by considering the "Last-Assigned-Timestamp" of the already timestamped flits. The local delay measurement technique can be extended to estimate regional or destination-based congestion to implement more sophisticated

adaptive routing algorithms like Regional Congestion Awareness [20] and Destination-Based Adaptive Routing [60] that monitor global network congestion.

5.6.2 Off-chip routers

As already mentioned, distributed shared-buffer routers [25, 50], which can emulate an output-buffered router without router speedup, have been successfully used for Internet routing. Stunkel et al. [58] proposed the IBM Colony router which is a customized architecture for off-chip interconnection networks with large central buffers and three crossbars. Although the architecture is similar to DSB, it does not use timestamping of flits for OBR emulation. Instead, packets potentially incur large de-serialization and serialization latencies to support wide SRAM accesses.

Chuang et al. [9] showed that an input-buffered router can also emulate an output-buffered router. This emulation requires a router speedup of 2 and an impractical complex matching problem, both of which are hard to achieve with on-chip designs. In addition, load-balanced routers [7, 30, 41] have been proposed as scalable architectures for Internet routing. To ensure packet ordering, they typically employ aggregation or scheduling schemes that incur long latencies, which is not acceptable in the context of on-chip networks.

5.7 Conclusions

In this chapter, we proposed a distributed-shared-buffer (DSB) router for on-chip networks. DSB routers have been successfully used in Internet routers to emulate the ideal throughput of output-buffered routers, but porting them to on-chip networks with more stringent constraints presents tough challenges. The proposed DSB router achieves up to 19% higher saturation throughput than input-buffered routers (IBRs) and up to 94% of the ideal saturation throughput for synthetic traffic patterns. The higher saturation throughput translates to large reductions in network latency with SPLASH-2 benchmarks. For SPLASH-2 applications which exhibit high contention and demand high communication bandwidth, DSB routers on average have 61% lower network latency than IBRs.

Chapter 5, in full, is a reprint of the material as it appears in the following publications:

- Vassos Soteriou, Rohit Sunkam Ramanujam, Bill Lin, and Li-Shiuan Peh, "A High-Throughput Distributed Shared-Buffer NoC Router", *Computer Architecture Letters*, April, 2009.

- Rohit Sunkam Ramanujam, Vassos Soteriou, Bill Lin, and Li-Shiuan Peh, "Design of a High-Throughput Distributed Shared-Buffer NoC Router", *The 4th ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, May 3-6, 2010, Grenoble, France.

- Rohit Sunkam Ramanujam, Vassos Soteriou, Bill Lin, and Li-Shiuan Peh, "Extending the Effective Throughput of NoCs with Distributed Shared-Buffer Routers", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol.30, no.4, pp.548-561, April 2011.

The dissertation author was the primary investigator and author of the papers.

Chapter 6

Conclusion and Future Directions

The many-core era has reached a phase where widespread adoption of NoCs is inevitable to facilitate efficient communication between the increasing number of components integrated within a chip. This dissertation tackles the critical problem of throughput-driven NoC design where communication throughput is the primary performance metric that is maximized, while minimizing costs related to latency, power and area. We propose two different approaches to achieve this goal, first, by designing efficient routing algorithms that load-balance traffic uniformly over all network links and second, by improving the router microarchitecture to alleviate bottlenecks along the datapath of routers and improve their packet-multiplexing capabilities. Both these approaches provide substantial improvements in both worst-case and average-case throughputs over existing solutions. In bandwidth-hungry workloads, throughput and latency often go hand in hand, i.e., increasing the saturation throughput of the network results in significant improvements in average packet latencies brought about by avoiding or delaying network saturation during temporary bursts of congestion. Hence, throughput-driven NoC design can significantly improve application performance in many-core chips both by increasing communication throughput and reducing communication delays.

A key challenge in exploring and evaluating routing algorithms and router architectures is to accurately model the on-chip traffic that needs to be sustained by the communication fabric. One approach is to run full-system simulations that accurately model the entire system and can directly evaluate how changes in the

NoC affect application performance. Full system simulations, however, are extremely time-consuming and do not scale well to large systems. One option to expedite the exploration process is to run a single full-system simulation (assuming a certain underlying interconnect model) to derive network traffic traces for different applications, and use these traces to drive NoC design. Significant portions of the evaluations carried out in this dissertation involve such trace simulations. Simple traffic traces, however, cannot keep track of dependencies between packets and fail to model the self-throttling nature of most CMP applications. This is due to the absence of a feedback mechanism that changes the injection times of packets in response to communication delays currently experienced in the network. Hence, one open problem in this area is to explore trace-based evaluation techniques that are more accurate than trace simulations currently being used in NoC research, but that are significantly faster than a full-blown system simulation or emulation. Such trace-based methods need to incorporate abstract models for different chip components within the NoC simulator for deriving approximate packet service times. They also need to include information about inter-packet dependencies (like request-response dependencies) within a trace to model the self-throttling nature of applications under network congestion.

As discussed in Chapter 1, potential application domains for throughput-centric NoCs will rely on hundreds of processor cores and thousands of threads to exploit different forms of parallelism like data-level parallelism in GPUs, packet-level parallelism in network processors, request-level parallelism in servers, etc. With a large number of threads accessing shared resources, ensuring fairness and guaranteeing QoS for network flows becomes important for most applications. Some of these capabilities can be incorporated into the on-chip network while others can be implemented at higher layers. The work presented in this dissertation does not differentiate between network flows and treats all flows equally. Depending on the requirements of applications and the way fairness between communicating flows is defined, bandwidth and latency fairness issues need to be suitably addressed in NoCs.

Finally, 3D ICs promise “more than Moore” integration by packing a great deal of functionality into small form factors, while improving performance and reducing costs. Research on 3D integration is still in its early stages and new capabilities need to

be added to existing toolkits to provide an unified design and verification environment for 3D ICs. In the context of NoCs, 3D ICs are expected to have heterogenous 2D layers interconnected by through-silicon vias. The short inter-layer distances and low latency vertical interconnects provide new opportunities to innovate with new interconnect topologies for 3D ICs. One such architecture using crossbars for inter-layer communication has been proposed in this dissertation, but this architecture still assumes homogenous communicating elements. Depending on the organization of 3D ICs, the all-to-all communication model often used in NoC research may not apply for heterogenous 3D chips. There is certainly scope for developing new routing algorithms and network architectures specifically tailored for such heterogenous systems.

Bibliography

[1] Netmaker. `http://www-dyn.cl.cam.ac.uk/~rdm34/wiki/index.php?title=Main_Page`.

[2] J. Balfour and W. J. Dally. Design tradeoffs for tiled CMP on-chip networks. In *Proceedings of the 20th annual International Conference on Supercomputing*, ICS '06, pages 187–198, New York, NY, USA, 2006. ACM.

[3] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, Liewei Bao, J. Brown, M. Mattina, Chyi-Chang Miao, C. Ramey, D. Wentzlaff, W. Anderson, E. Berger, N. Fairbanks, D. Khan, F. Montenegro, J. Stickney, and J. Zook. TILE64 - processor: A 64-core SoC with mesh interconnect. In *IEEE International Solid-State Circuits Conference*, ISSCC '08, pages 88 –598, Feb. 2008.

[4] L. Benini and G. De Micheli. Networks on chips: a new SoC paradigm. *Computer*, 35(1):70 –78, Jan. 2002.

[5] D. Bertozzi and L. Benini. Xpipes: a network-on-chip architecture for gigascale systems-on-chip. *IEEE Circuits and Systems Magazine*, 4(2):18 – 31, 2004.

[6] B. Black, D. Nelson, C. Webb, and N. Samra. 3D processing technology and its impact on IA32 microprocessors. In *Proceedings of International Conference on Computer Design*, pages 316–318, Stanford, CA, USA, 2004.

[7] C-S. Chang, D-S. Lee, and Y-S. Jou. Load balanced Birkhoff-von Neumann switches, part 1: one-stage buffering. *Computer Communications*, 25(6):611 – 622, 2002.

[8] A. N. Chrysostomos, P. Dongkook, K. Jongman, N. Vijaykrishnan, S. Y. Mazin, and C. R. Das. ViChaR: A dynamic virtual channel regulator for network-on-chip routers. *IEEE/ACM International Symposium on Microarchitecture*, 0:333–346, Dec. 2006.

[9] S-T. Chuang, A. Goel, N. McKeown, and B. Prabhakar. Matching output queueing with a combined input/output-queued switch. *IEEE Journal on Selected Areas in Communications*, Jun. 1999.

[10] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, 36:547–553, May 1987.

[11] W. J. Dally and B. Towles. Route packets, not wires: on-chip inteconnection networks. In *Proceedings of the 38th annual Design Automation Conference*, DAC '01, pages 684–689, New York, NY, USA, 2001. ACM.

[12] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[13] W.J. Dally. Virtual-channel flow control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194 –205, Mar. 1992.

[14] W.J. Dally and H. Aoki. Deadlock-free adaptive routing in multicomputer networks using virtual channels. *IEEE Transactions on Parallel and Distributed Systems.*, 4(4):466 –475, Apr. 1993.

[15] W.R. Davis, J. Wilson, S. Mick, J. Xu, H. Hua, C. Mineo, A.M. Sule, M. Steer, and P.D. Franzon. Demystifying 3D ICs: the pros and cons of going vertical. *IEEE Design Test of Computers*, 22(6):498 – 510, Nov.-Dec. 2005.

[16] J. Duato. A new theory of deadlock-free adaptive routing in wormhole networks. *IEEE Transactions on Parallel Distributed Systems*, 4:1320–1331, Dec. 1993.

[17] L. Feihui, C. Nicopoulos, T. Richardson, Y. Xie, V. Narayanan, and M. Kandemir. Design and management of 3D chip multiprocessors using network-in-memory. In *33rd International Symposium on Computer Architecture*, ISCA '06, pages 130 –141, 2006.

[18] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. In *Classic Papers in Combinatorics*, pages 243–248. 1987.

[19] M. Garland and D. B. Kirk. Understanding throughput-oriented architectures. *Communications ACM*, 53:58–66, Nov. 2010.

[20] P. Gratz, B. Grot, and S.W. Keckler. Regional congestion awareness for load balance in networks-on-chip. In *IEEE 14th International Symposium on High Performance Computer Architecture*, HPCA '08, pages 203 –214, Feb. 2008.

[21] P. Gratz, C. Kim, R. McDonald, S.W. Keckler, and D. Burger. Implementation and evaluation of on-chip network architectures. In *International Conference on Computer Design*, pages 477 –484, Oct. 2006.

[22] T. R. Halfhill. Netlogic broadens XLP family. Microprocessor Report, Jul., 2010.

[23] J. Howard, S. Dighe, Y. Hoskote, S. Vangal, D. Finan, G. Ruhl, D. Jenkins, H. Wilson, N. Borkar, G. Schrom, F. Pailet, S. Jain, T. Jacob, S. Yada, S. Marella, P. Salihundam, V. Erraguntla, M. Konow, M. Riepen, G. Droege, J. Lindemann, M. Gries, T. Apel, K. Henriss, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, R. Van Der Wijngaart, and T. Mattson. A 48-core IA-32 message-passing processor with DVFS in 45nm CMOS. In *IEEE International Solid-State Circuits Conference. Digest of Technical Papers*, ISSCC '10, pages 108 –109, Feb. 2010.

[24] IBM. IBM Blue Gene project. `http://www.research.ibm.com/bluegene/`.

[25] S. Iyer, R. Zhang, and N. McKeown. Routers with a single stage of buffering. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '02, pages 251–264, New York, NY, USA, 2002. ACM.

[26] N. Jiang, J. Kim, and W. J. Dally. Indirect adaptive routing on large scale interconnection networks. In *Proceedings of the 36th annual International Symposium on Computer Architecture*, ISCA '09, pages 220–231, New York, NY, USA, 2009. ACM.

[27] 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*, 49(4.5):589 –604, Jul. 2005.

[28] A.B. Kahng, B. Li, L-S. Peh, and K. Samadi. ORION 2.0: A fast and accurate NoC power and area model for early-stage design space exploration. In *Design, Automation Test in Europe Conference Exhibition, 2009.*, DATE '09, pages 423 –428, Apr. 2009.

[29] M. Kawano, N. Takahashi, Y. Kurita, K. Soejima, M. Komuro, and S. Matsui. Three-dimensional packaging technology for stacked DRAM with 3-Gb/s data transfer. *IEEE Transactions on Electron Devices*, 55(7):1614 –1620, Jul. 2008.

[30] I. Keslassy, S-T. Chuang, K. Yu, D. Miller, M. Horowitz, O. Solgaard, and N. McKeown. Scaling internet routers using optics. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '03, pages 189–200, New York, NY, USA, 2003. ACM.

[31] T. Kgil, S. D'Souza, A. Saidi, N. Binkert, R. Dreslinski, T. Mudge, S. Reinhardt, and K. Flautner. PicoServer: using 3D stacking technology to enable a compact energy efficient chip multiprocessor. In *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, ASPLOS-XII, pages 117–128, New York, NY, USA, 2006. ACM.

[32] J. Kim, C. Nicopoulos, and D. Park. A gracefully degrading and energy-efficient modular router architecture for on-chip networks. In *Proceedings of the 33rd annual International Symposium on Computer Architecture*, ISCA '06, pages 4–15, Washington, DC, USA, 2006. IEEE Computer Society.

[33] J. Kim, C. Nicopoulos, D. Park, R. Das, Y. Xie, V. Narayanan, M. S. Yousif, and C. R. Das. A novel dimensionally-decomposed router for on-chip communication in 3D architectures. In *Proceedings of the 34th annual international symposium on Computer architecture*, ISCA '07, pages 138–149, New York, NY, USA, 2007. ACM.

[34] J. Kim, D. Park, T. Theocharides, N. Vijaykrishnan, and C. R. Das. A low latency router supporting adaptivity for on-chip interconnects. In *Proceedings of the 42nd annual Design Automation Conference*, DAC '05, pages 559–564, New York, NY, USA, 2005. ACM.

[35] T. Krishna, J. Postman, C. Edmonds, L-S. Peh, and P. Chiang. SWIFT: A swing-reduced interconnect for a token-based network-on-chip in 90nm CMOS. In *2010 IEEE International Conference on Computer Design (ICCD)*, pages 439 –446, Oct. 2010.

[36] A. Kumar, P. Kundu, A.P. Singh, L.-S. Peh, and N.K. Jha. A 4.6Tbits/s 3.6GHz single-cycle NoC router with a novel switch allocator in 65nm CMOS. In *International Conference on Computer Design*, pages 63 –70, Oct. 2007.

[37] A. Kumar, L-S. Peh, and N. K. Jha. Token flow control. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 41, pages 342–353, Washington, DC, USA, 2008. IEEE Computer Society.

[38] A. Kumar, L-S. Peh, P. Kundu, and N. K. Jha. Express virtual channels: towards the ideal interconnection fabric. In *Proceedings of the 34th annual International Symposium on Computer Architecture*, ISCA '07, pages 150–161, New York, NY, USA, 2007. ACM.

[39] K. P. Lawton. Bochs: A portable PC emulator for Unix/X. *Linux Journal*, 1996(29):7, 1996.

[40] K.W. Lee, T. Nakamura, T. Ono, Y. Yamada, T. Mizukusa, H. Hashimoto, K.T. Park, H. Kurino, and M. Koyanagi. Three-dimensional shared memory fabricated using wafer stacking technology. In *International Electron Devices Meeting, 2000. IEDM Technical Digest*, pages 165 –168, 2000.

[41] B. Lin and I. Keslassy. The concurrent matching switch architecture. *IEEE/ACM Transactions on Networking*, 18(4):1330 –1343, Aug. 2010.

[42] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym. NVIDIA Tesla: A unified graphics and computing architecture. *IEEE Micro*.

[43] D. Lugones, D. Franco, and E. Luque. Dynamic and distributed multipath routing policy for high-speed cluster networks. In *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 396–403, Washington, DC, USA, 2009. IEEE Computer Society.

[44] S. S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb. The Alpha 21364 network architecture. In *Proceedings of the The Ninth Symposium on High Performance Interconnects*, pages 113–, Washington, DC, USA, 2001. IEEE Computer Society.

[45] R. Mullins, A. West, and S. Moore. Low-latency virtual-channel routers for on-chip networks. In *Proceedings. 31st Annual International Symposium on Computer Architecture*, pages 188 – 197, Jun. 2004.

[46] T. Nesson and S. L. Johnsson. ROMM routing on mesh and torus networks. In *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures*, SPAA '95, pages 275–287, New York, NY, USA, 1995. ACM.

[47] V. Paxson. Fast, approximate synthesis of fractional Gaussian noise for generating self-similar network traffic. *SIGCOMM Computer Communications Review.*, 27:5–18, Oct. 1997.

[48] L-S. Peh and W. J. Dally. A delay model and speculative architecture for pipelined routers. In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, HPCA '01, pages 255–266, Washington, DC, USA, 2001. IEEE Computer Society.

[49] L-S. Peh and W.J. Dally. Flit-reservation flow control. In *Proceedings. Sixth International Symposium on High-Performance Computer Architecture*, pages 73 –84, 2000.

[50] A. Prakash, A. Aziz, and V. Ramachandran. Randomized parallel schedulers for switch-memory-switch routers: analysis and numerical studies. In *Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3 of *INFOCOM '04*, pages 2026 –2037, Mar. 2004.

[51] S. L. Scott and G. Thorson. The Cray T3E network: Adaptive routing in a high-performance 3D torus. In *Hot Interconnects-4*, 1996.

[52] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan. Larrabee: a many-core x86 architecture for visual computing. *ACM Transactions on Graphics*, 27:18:1–18:15, Aug. 2008.

[53] D. Seo, A. Ali, W-T. Lim, N. Rafique, and M. Thottethodi. Near-optimal worst-case throughput routing for two-dimensional mesh networks. In *Proceedings of*

the 32nd annual international symposium on Computer Architecture, ISCA '05, pages 432–443, Washington, DC, USA, 2005. IEEE Computer Society.

[54] L. Shang, L-S. Peh, and N. K. Jha. Dynamic voltage scaling with links for power optimization of interconnection networks. In *Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, HPCA '03, pages 91–, Washington, DC, USA, 2003. IEEE Computer Society.

[55] A. Singh. Load-balanced routing in interconnection networks. Ph.D thesis, Stanford University, Mar. 2003.

[56] A. Singh, W. J. Dally, A. K. Gupta, and B. Towles. Adaptive channel queue routing on k-ary n-cubes. In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, SPAA '04, pages 11–19, New York, NY, USA, 2004. ACM.

[57] A. Singh, W. J. Dally, B. Towles, and A. K. Gupta. Locality-preserving randomized oblivious routing on torus networks. In *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, SPAA '02, pages 9–13, New York, NY, USA, 2002. ACM.

[58] C.B. Stunkel, J. Herring, B. Abali, and R. Sivaram. A new switch chip for IBM RS/6000 SP systems. In *ACM/IEEE 1999 Conference on Supercomputing*, page 16, Nov. 1999.

[59] H. Sullivan and T. R. Bashkow. A large scale, homogeneous, fully distributed parallel machine. In *Proceedings of the 4th annual symposium on Computer architecture*, ISCA '77, pages 105–117, New York, NY, USA, 1977. ACM.

[60] R. Sunkam Ramanujam and B. Lin. Destination-based adaptive routing on 2D mesh networks. In *Proceedings of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '10, pages 19:1–19:12, New York, NY, USA, Oct. 2010. ACM.

[61] R. Sunkam Ramanujam and Bill Lin. Near-optimal oblivious routing on three-dimensional mesh networks. In *IEEE International Conference on Computer Design*, pages 134 –141, Oct. 2008.

[62] I. Sutherland, B. Sproull, and D. Harris. *Logical Effort: Designing Fast CMOS Circuits*. Morgan Kaufmann Publishers, 1999.

[63] M.B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, Jae-Wook Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal. The Raw microprocessor: a computational fabric for software circuits and general-purpose programs. *IEEE Micro*, 22(2):25 – 35, Mar./Apr. 2002.

[64] B. Towles and W. J. Dally. Worst-case traffic for oblivious routing functions. In *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, SPAA '02, pages 1–8, New York, NY, USA, 2002. ACM.

[65] B. Towles, W. J. Dally, and S. Boyd. Throughput-centric routing algorithm design. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, SPAA '03, pages 200–209, New York, NY, USA, 2003. ACM.

[66] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, STOC '81, pages 263–277, New York, NY, USA, 1981. ACM.

[67] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar. An 80-tile 1.28TFLOPS network-on-chip in 65nm CMOS. In *IEEE International Solid-State Circuits Conference, 2007. Digest of Technical Papers*, ISSCC '07, pages 98 –589, Feb. 2007.

[68] H. Wang, L-S. Peh, and S. Malik. Power-driven design of router microarchitectures in on-chip networks. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, pages 105–116, Washington, DC, USA, 2003. IEEE Computer Society.

[69] H-S. Wang, X. Zhu, L-S. Peh, and S. Malik. Orion: A power-performance simulator for interconnection networks. In *Proceedings. 35th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, pages 294 – 305, 2002.

[70] R. Wilson. Cisco taps processor array architecture for NPU. EE Times., August 9, 2004.

[71] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. *SIGARCH Comput. Archit. News*, 23:24–36, May 1995.

[72] L. Xue, C.C. Liu, H-S. Kim, S.K. Kim, and S. Tiwari. Three-dimensional integration: technology, use, and issues for mixed-signal applications. *IEEE Transactions on Electron Devices,*, 50(3):601 – 609, Mar. 2003.