Multicomputer Routing Techniques[†]

Melanie L. Fulgham

Technical Report UW-CSE-97-11-02

Department of Computer Science and Engineering
University of Washington

Department of Computer Science and Engineering, Box 352350
University of Washington, Seattle, WA 98195-2350 USA

[†]This technical report is an adaptation of Melanie Fulgham's Ph.D. dissertation. This work has been supported in part by an NSF Graduate Fellowship, an ARPA Graduate Research Fellowship, and by NSF Grant MIP-9213469.

University of Washington

Abstract

Multicomputer Routing Techniques

by Melanie L. Fulgham

Chairperson of Supervisory Committee
Professor Lawrence Snyder
Department of Computer Science and Engineering

Communication is a fundamental problem in parallel computing. Without high performance communication, applications cannot exploit available parallelism effectively. Communication time has three components: the overhead to send and receive a message, the time to move a message between the processor and the network, and the time in the network to deliver a message. Research has reduced the first two, producing machines where time in the network is over fifty percent of the total communication time.

We demonstrate methods to increase network performance by examining fundamental network issues such as routing algorithms, router implementation, and flow control. We also make progress towards solving the routing problem, a search for a router that provides high throughput, low latency network performance for all application traffic and network loads.

In this thesis, we formulate a methodology for evaluating multi-class routing algorithms and present Triplex, the first triple class routing algorithm. Multi-class algorithms increase performance and help solve the routing problem by providing several kinds or classes of routing. This gives applications the flexibility to select the

type of routing that best matches their communication needs. Multi-class routing also removes a major liability of adaptive routing, out-of-order message delivery. By providing a class for in-order delivery, multi-class routing can simplify the network interface since message reordering becomes unnecessary.

We examine the effects of router implementation on network performance by comparing two methods of implementing a router: as an input driven and as an output driven router. Although the methods are similar, the output driven routers almost always perform as well as or better than the input driven routers. Almost all routers are input driven, even though most routing algorithms can be implemented as either.

The effects of flow control on network performance are also explored. We compare a variety of packet routing algorithms and demonstrate the following by simulation. Packet routing provides throughput superior to traditional wormhole networks which have a limited amount of buffering. And although often overlooked, non-minimal fully adaptive routing with controlled randomization can achieve throughput and latency that is superior to oblivious and minimal fully adaptive routing algorithms.

TABLE OF CONTENTS

List of Figures **iv**

List of Tables **x**

Chapter 1: Introduction **1**

1.1 Background 1

1.2 Thesis Structure 10

1.3 Contributions 11

Chapter 2: Key Properties of Routers **12**

2.1 Deadlock-Freedom 12

2.2 Livelock-freedom 23

2.3 Starvation-freedom 26

Chapter 3: Simulation Methodology **28**

3.1 Routing Algorithms 28

3.2 Router Description 29

3.3 Workloads 31

3.4 Performance Measures 33

3.5 Random Number Generation 34

3.6 Gathering Statistics 37

Chapter 4: Triplex Routing **40**

4.1 Introduction 40

4.2 Motivation 40
4.3 Overview 41
4.4 Multi-class Routing 42
4.5 Evaluation 43
4.6 The Triplex Algorithm 46
4.7 Comparisons 56
4.8 Related Multi-class Work 70
4.9 Summary 72

Chapter 5: Input Versus Output Driven Routing 74

5.1 Input versus Output Driven 74
5.2 Methodology 76
5.3 Results 78
5.4 Summary 86

Chapter 6: Packet Routing Performance 88

6.1 Simulations 89
6.2 Summary 98

Chapter 7: Conclusions 100

7.1 Contributions 100
7.2 Future Work 102

Bibliography 108

Appendix A: Triplex 119

A.1 Proofs of Deadlock-freedom 119
A.2 Proof of Livelock-Freedom 130
A.3 Performance Comparisons 131

Appendix B: Input and Output Driven **142**

B.1 Peak throughput 142

B.2 Performance Comparisons 145

LIST OF FIGURES

1.1 An example of a two-dimensional torus multicomputer. 2

1.2 Clockwise from the top left are examples of k-ary n-cubes: a 2D mesh, a 2D torus, and a 3D hypercube. 3

1.3 An example of a minimal path between a source (S) and destination (D) on a 2D mesh. 8

1.4 An example of a non-minimal path between a source (S) and destination (D) on a 2D mesh. 9

2.1 An example of a message that has corrected dimension 0, is currently correcting dimension 1, and still needs to finish correcting both dimensions. 15

2.2 An example of a dimension-order oblivious path between a source (S) and destination (D) on a 2D mesh. This is the only path allowed between this source and destination. 16

3.1 An example of a router with input buffers, output, buffers, crossbar, injection buffer, and ejection buffer. 30

4.1 An example of a route allowed by Rule 1 between a source (S) and destination (D). 49

4.2 An example of a route allowed by Rule 2 between a source (S) and destination (D). 49

4.3 An example of a route allowed by Rule 3 between a source (S) and destination (D). 50
4.4 An example of a route allowed by Rule 4 between a source (S) and destination (D). 50
4.5 This is an example of a Triplex message path on a 3D mesh between a source (S) and destination (D). Each edge is labeled by the rules which apply to the route. 51
4.6 An example of a cyclic dependency that does not cause deadlock. One of the messages in the cycle waits on a message that is not in the cycle. This is an escape route for the cycle. 52
4.7 A potential cycle caused by Triplex routing. The proof shows that the turn highlighted by dashed lines is not allowed. 53
4.8 Wormhole performance on a 256-node 2D torus with 40-word messages. 60
4.9 Wormhole performance on a 256-node 2D torus with 40-word and 400-word messages in a 10:1 ratio. 63
4.10 Wormhole performance on a 256-node 2D torus with 40-word and 400-word messages in a 10:1 ratio. 63
4.11 Wormhole performance on a 256-node 2D torus with 40-word and 400-word messages in a 10:1 ratio. 64
4.12 Wormhole performance on a 256-node 2D torus with 40-word and 400-word messages in a 10:1 ratio. 64
4.13 Throughput and latency comparisons between oblivious and Triplex algorithms for a range of traffic mixes on a 256-node 2D torus using wormhole routing with 40-word and 400-flit messages in a 10:1 ratio. 66
4.14 The performance advantage of Triplex over the oblivious algorithm on a 256-node 2D torus with 40-word and 400-word messages in a 10:1 ratio. 67

4.15 Performance on a 256-node 2D torus with 20-word packets. 68

5.1 An example of an input driven router. The short arrow marks the current input buffer. The long arrows are the choices of empty output buffers for the message in the current input buffer. Messages can travel north (N), east (E), south (S), or west (W). Buffers that are filled in are not available, while the unfilled buffers are available. 75

5.2 An example of an output driven router. The short arrow marks the current output buffer. The long arrows are from messages that are competing for the current empty output buffer. Messages can travel north (N), east (E), south (S), or west (W). Buffers that are filled in are not available, while the unfilled buffers are available. 76

6.1 Throughput and latency for 256-node 2D torus networks with uniform random traffic. 92

6.2 Throughput and latency for 256-node 2D torus networks with bit reversal traffic. 93

6.3 Throughput and latency for 256-node 2D torus networks with complement traffic. 93

6.4 Throughput and latency for 256-node 2D torus networks with transpose traffic. 94

6.5 Throughput and latency for 256-node 2D torus networks with perfect shuffle traffic. 94

6.6 Throughput and latency for 256-node 2D torus networks with ten four-times hot spot traffic. 96

6.7 Throughput and latency for 256-node 2D torus networks with ten four-times hot spot traffic. 97

A.1 An example of a route that violates the rules of Triplex. (The illegal buffer is marked.) The shaded buffers are the restricted output buffers, while the unshaded buffers are unrestricted. 122

A.2 Throughput and latency on 256-node 2D torus with packet routing for 20-word messages. 132

A.3 Throughput and latency on a 256-node 2D torus with packet routing for 20-word messages. 133

A.4 Throughput and latency on a 256-node 2D torus with packet routing for 20-word messages. 134

A.5 Throughput and latency on a 256-node 2D torus with wormhole routing for 40-word messages. 135

A.6 Throughput and latency on a 256-node 2D torus with wormhole routing for 40-word messages. 136

A.7 Throughput and latency on a 256-node 2D torus with wormhole routing for 40-word and 400-flit messages in a 10:1 ratio. 136

A.8 Throughput and latency on a 256-node 2D torus with wormhole routing for 40-word and 400-word messages in a 10:1 ratio. 137

A.9 Throughput and latency comparisons between oblivious and Triplex algorithms for a range of traffic mixes on a 256-node 2D torus using wormhole routing with 40-word and 400-flit messages in a 10:1 ratio. 138

A.10 Throughput and latency comparisons between oblivious and Triplex algorithms for a range of traffic mixes on a 256-node 2D torus using wormhole routing with 40-word and 400-flit messages in a 10:1 ratio. 139

A.11 Throughput and latency comparisons between oblivious and Triplex algorithms for a range of traffic mixes on a 256-node 2D torus using wormhole routing with 40-word and 400-flit messages in a 10:1 ratio. 140

A.12 Throughput and latency comparisons between oblivious and Triplex algorithms for a range of traffic mixes on a 256-node 2D torus using wormhole routing with 40-word and 400-flit messages in a 10:1 ratio. 141

B.1 Throughput and latency on a 256-node 2D torus with fixed output buffer selection. 146

B.2 Throughput and latency on a 256-node 2D torus with fixed output buffer selection. 147

B.3 Throughput and latency on a 256-node 2D torus with fixed output buffer selection. 148

B.4 Throughput and latency on a 256-node 2D torus with fixed output buffer selection. 149

B.5 Throughput and latency on a 256-node 2D mesh with fixed output buffer selection. 150

B.6 Throughput and latency on a 256-node 2D mesh with fixed output buffer selection. 151

B.7 Throughput and latency on a 256-node 2D mesh with fixed output buffer selection. 152

B.8 Throughput and latency on a 256-node 2D mesh with fixed output buffer selection. 153

B.9 Throughput and latency on a 256-node torus with random output buffer selection. 154

B.10 Throughput and latency on a 256-node torus with random output buffer selection. 155

B.11 Throughput and latency on a 256-node torus with random output buffer selection. 156

B.12 Throughput and latency on a 256-node torus with random output buffer selection. 157
B.13 Throughput and latency on a 256-node mesh with random output buffer selection. 158
B.14 Throughput and latency on a 256-node mesh with random output buffer selection. 159
B.15 Throughput and latency on a 256-node mesh with random output buffer selection. 160
B.16 Throughput and latency on a 256-node mesh with random output buffer selection. 161
B.17 Throughput and latency on a 256-node torus comparing input driven routing with fixed and random selection. 162
B.18 Throughput and latency on a 256-node torus comparing input driven routing with fixed and random selection. 163
B.19 Throughput and latency on a 256-node torus comparing input driven routing with fixed and random selection. 164
B.20 Throughput and latency on a 256-node torus comparing input driven routing with fixed and random selection. 165
B.21 Throughput and latency on a 256-node mesh comparing input driven routing with fixed and random selection. 166
B.22 Throughput and latency on a 256-node mesh comparing input driven routing with fixed and random selection. 167
B.23 Throughput and latency on a 256-node mesh comparing input driven routing with fixed and random selection. 168
B.24 Throughput and latency on a 256-node mesh comparing input driven routing with fixed and random selection. 169

LIST OF TABLES

1.1 Summary of the percentage of time a message spends in an uncongested network for various computers [CKP+93]. Components of the total communication time are shown in units of cycles. Overhead includes processing time for sending and receiving a message. Words per message depends on the channel width and is shown for a 160-bit message. This represents the time for the tail of the message to catch-up to header once it has been delivered. Node latency (node lat) is the time to make a routing decision at each node. Hops is the average distance a message travels. Total is the total communication time, while net is the percentage of time a message spends in the network. The last two rows refer to machines using active messages (AM) instead of the commercially provided software. 4

2.1 Comparison of deadlock-free routing algorithms for k-ary n-cubes including topology (topo), virtual channels (VC), adaptivity class (adapt), type of path, method of achieving deadlock freedom, and flow control (flow). 24

4.1 Summary of the differences between the various routing algorithms. . 58

4.2 Single class traffic comparisons between each Triplex class and its corresponding single class routing algorithm. 59

4.3 Mixed class traffic comparisons between Triplex and the oblivious routing algorithm, given a traffic mix where $x\%$, $0 \leq x \leq 100$, of the traffic requires in-order delivery, while the remaining $(100 - x)\%$ may use adaptive routing. 60

4.4 Minimum normalized applied load at which saturation is detected. Short refers to 40-word messages and mix refers to 40-word and 400-word messages in a 10:1 ratio. 65

5.1 Summary of the differences between the various routing algorithms. The number in parentheses refers to the number of buffers required when no virtual channels are used. 77

5.2 Summary of the input driven versus output driven router experiments. The comparisons in the two sets of experiments are made for three different routing algorithms using seven different traffic patterns for both the mesh and torus topologies. 78

5.3 Minimum normalized applied load within .05 at which saturation is detected. 79

5.4 Minimum normalized applied load within .05 at which saturation is detected. 80

5.5 Minimum normalized applied load within .05 at which saturation is detected. 84

5.6 Minimum normalized applied load within .05 at which saturation is detected. If an entry is marked with an asterisk, the difference in saturation points between the input driven and output driven router is not significant. 85

6.1 Summary of the design differences between the packet routing algorithms simulated. 90
6.2 Minimum load at which saturation is detected for 256-node 2D networks. 91

B.1 Shows the normalized applied load (load) at which the maximum normalized throughput (xput) is achieved by the traffic pattern. The percent error (error) is also shown. 142
B.2 Shows the normalized applied load (load) at which the maximum normalized throughput (xput) is achieved by the traffic pattern. The percent error (error) is also shown. 143
B.3 Shows the normalized applied load (load) at which the maximum normalized throughput (xput) is achieved by the traffic pattern. The percent error (error) is also shown. 143
B.4 Shows the normalized applied load (load) at which the maximum normalized throughput (xput) is achieved by the traffic pattern. The percent error (error) is also shown. 144
B.5 Shows the normalized applied load (load) at which the maximum normalized throughput (xput) is achieved by the traffic pattern. The percent error (error) is also shown. 144
B.6 Shows the normalized applied load (load) at which the maximum normalized throughput (xput) is achieved by the traffic pattern. The percent error (error) is also shown. 145

Chapter 1

INTRODUCTION

The exchange of information over networks is essential in many aspects of computing. Some networks, like the Internet, allow email messages to be sent or web pages to be browsed from around the world. Other networks, called local area networks (LANS), support cooperative computing which enables sharing of computing resources within a small area such as a building or campus. And still others provide communication between processors in a parallel computer, enabling problems to be solved faster than with a uni-processor. Each of these communication domains has a unique set of communication requirements and constraints resulting in unique solutions for sharing information efficiently. This thesis considers methods of practical communication in parallel computer networks, where practical refers to factors such as algorithm complexity, ease of implementation, performance, and hardware costs.

1.1 Background

The model of computation considered in this work is the multicomputer [AS88]. See Figure 1.1 for an example of a multicomputer. The multicomputer is a single computer where each node is composed of a processor, local memory, network interface, and hardware router. The nodes are connected in a point to point network by *links* or *channels* in a regular topology such as a k-ary n-cube, shuffle-exchange, or cube-connected cycles. See Figure 1.2 for examples of two-dimensional (2D) and three-dimensional (3D) networks. Processors communicate by sending messages through

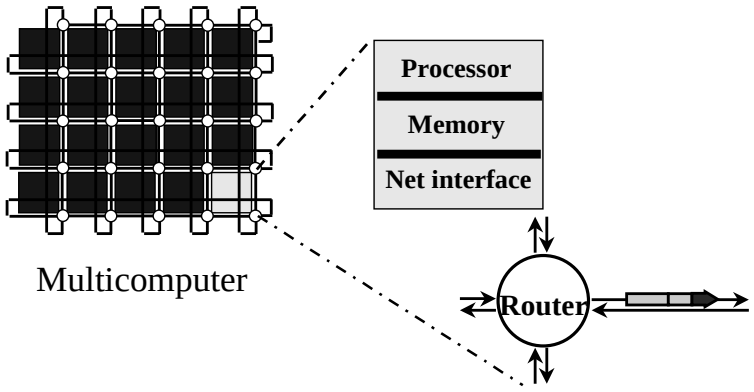

Figure 1.1: An example of a two-dimensional torus multicomputer.

the network via the network interface. The information used to route a message is contained in the message *header*; the remaining portion of the message, called the message *body* or *payload*, carries data. Hardware *routers* examine message headers and forward each message to a neighboring router according to the routing algorithm till it reaches its destination node. The *routing algorithm* represents the rules that specify the path a message may take through the network. The algorithm consists of two components: the *routing* relation, which computes the possible set of buffers that may be used by a message, and the *selection* function, which chooses one of the buffers from the set output by the routing relation. The selection function may make decisions by considering factors such as the number of other messages in local buffers. A routing algorithm is *connected*, if there is a path specified by the routing relation between every pair of nodes in the network. This is analogous to a graph which is strongly connected, but in this case, routing connectivity not edge connectivity is considered.

There are three components to total communication time: the message overhead (processing time) to send and receive messages, the time in the network interface to move messages between the processor and the router, and time in the network

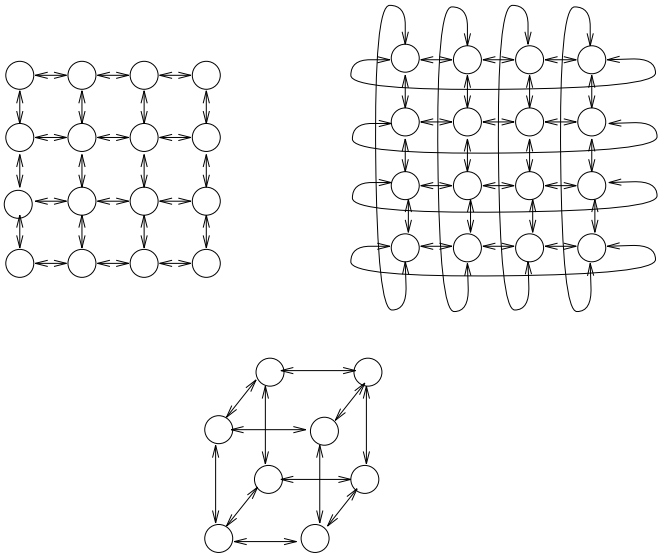

Figure 1.2: Clockwise from the top left are examples of k-ary n-cubes: a 2D mesh, a 2D torus, and a 3D hypercube.

to deliver the messages. As research continues to reduce the message overhead and network interface times [vEDCGS92, Dal90b, BLA+94, MBES94, BJM+96], time in the network is becoming a significant portion of total communication time [CKP+93]. This has motivated research in fundamental network issues and their effects on overall communication performance. This dissertation explores three key network issues: routing algorithms, router implementation, and flow control methods. See Table 1.1 for examples of the percentage of total communication time spent in the network in various computers.

1.1.1 Network Assumptions

Routers in the parallel computing domain are quite different from those in other networks. They are very simple, fast, low cost, and must be scalable since a multicomputer may contain hundreds or thousands of nodes. To achieve fine-grain parallelism, the latency constraints of multicomputers require routing decisions to be made very quickly with information local to the node (this may include status from a neighbor-

Table 1.1: Summary of the percentage of time a message spends in an uncongested network for various computers [CKP+93]. Components of the total communication time are shown in units of cycles. Overhead includes processing time for sending and receiving a message. Words per message depends on the channel width and is shown for a 160-bit message. This represents the time for the tail of the message to catch-up to header once it has been delivered. Node latency (node lat) is the time to make a routing decision at each node. Hops is the average distance a message travels. Total is the total communication time, while net is the percentage of time a message spends in the network. The last two rows refer to machines using active messages (AM) instead of the commercially provided software.

Machine	Topology	Overhead	Words	Node Lat	Hops	Total	Net
nCUBE/2	Hypercube	6400	160	40	5	6760	5
CM-5	Fattree	3600	40	8	9.3	3714	3
DASH	Torus	30	10	2	6.8	53	43
J-Machine	3D Mesh	16	20	2	12.1	60	73
Monsoon	Butterfly	10	10	2	5	30	67
nCUBE/2-AM	Hypercube	1000	160	40	5	1360	27
CM-5-AM	Fattree	132	40	8	9.3	246	46

ing node). Since the topologies are regular, a topology gathering phase is not needed. The routers simply route or forward messages from one node to the next. Topology gathering is useful in rapidly changing or more complex fault-tolerant systems since this allows the network routes to be reconfigured. For a practical, low cost approach to fault-tolerance in multicomputer networks, see [BY94]. To keep hardware costs down, routers have small (hardware) buffers from a word to several hundreds of words in size.

In parallel computers the distance between nodes is very small; and as a consequence, the channels have very low transient error rates. Most errors are detected by CRC codes. Messages are rarely lost or corrupted, so a higher-level protocol is used to handle an infrequent error. Also because the distances between nodes are small, hop-by-hop flow control is practical. This means it is not necessary to drop messages. In fact, multicomputers do not intentionally drop messages. To prevent traffic from backing up the network, each router is assumed to remove delivered messages in a finite amount of time. Flow control is described in more detail in the following section. Finally, it is assumed that the network is secure and that messages do not need to be encrypted to obtain protection from eavesdropping.

1.1.2 Flow Control

Flow control prevents the sender from over running the capacity of a receiver. Multicomputers use three types of flow control: circuit-switched, packet-switched, and wormhole routing. The latter is actually a method of flow control, though its name implies otherwise.

In *circuit-switched* flow control there are three phases. In the first phase, a connection between the source and destination is set-up. Once set-up is complete, the message is sent along the established path. When communication is finished, the connection is torn down. There is a high overhead for initiating communication, since the entire path between the source and destination must be reserved. If there is

substantial data to send, however, the initial set-up time becomes insignificant. A telephone call is an example of a circuit-switched connection.

With *wormhole* routing, a message is broken into small pieces called *flits* or *fl*ow control dig*its* [DS86]. The message header is contained in the initial flit(s). All the flits in the message follow the path of the message header in a pipeline fashion. As soon as the routing decision has been made, a message may be forwarded from its current node to the next. Since flit buffers are small, a blocked message may span several nodes and hold many links in the network simultaneously. A message in an uncongested wormhole network has lower latency than in a circuit-switched network, since a set-up phase is not required.

In *packet-switched* networks, a message is segmented into units called *packets*, any of which can fit completely in an empty buffer in the network. Therefore when a message gets blocked, it is completely buffered within one node. There are two main types of packet-switched flow control: store-and-forward and virtual cut-through. With *store-and-forward* flow control, a message must be completely received in a node before it can be forwarded to another node. Virtual-cut through is a hybrid of wormhole and store-and-forward flow control. With *virtual-cut through*, a message may be forwarded as soon as the routing decision has been made, regardless of whether the entire message has been completely received within the current node. Packet-switched networks have a maximum packet size and use more buffer space than wormhole routing, but messages hold fewer resources when they block. Uncongested message latency for packet-switched networks with virtual cut-through is equivalent to that of wormhole networks.

Mad postman is an unusual optimization of packet routing that is used to avoid the latency of the routing decision [YJ89]. When using mad postman routing, a message is immediately forwarded in the same direction in which it is currently routing. If the prediction is incorrect, the message is killed and forwarded out the appropriate direction. This technique only works in directed acyclic networks and is rarely used.

1.1.3 Routing Style

There are two major styles of routing: batch routing and continuous or dynamic routing. In *batch* routing, each processor injects a set of messages into the network. Then, all the processors wait till the last message can be delivered before communicating again. Batch routing algorithms are often synchronous. An important special case of batch routing is *permutation* routing. In permutation routing, each node sends one message to a unique destination. The development of batch routing algorithms has been heavily influenced by the desire to make them analyzable. This means that quantities like the maximum number of steps required to route a permutation, or perhaps, the maximum queue size needed at any node can be determined.

With *continuous* routing, messages can be injected into the network at any time. The quantities of interest are throughput and latency. *Throughput* represents messages delivered per time unit, often a cycle, while *latency* measures the time a message spends in the network. Continuous routing algorithms are usually considered in the context of real machines, and therefore must be practical.

Practical algorithms have several key properties. First, they use a reasonable number of resources per router. Second, they make decisions based on local information only. And finally, the algorithms must be deadlock-free, livelock-free, and starvation-free. *Deadlock-freedom* insures a message never waits indefinitely to progress in the network; *livelock-freedom* guarantees a message will never circulate in the network continuously without being delivered; and, *starvation-freedom* guarantees a waiting message will eventually be injected into the network.

Because the requirements of being analytically tractable and of being practical are usually conflicting, the literature in routing has been partitioned into two parts: routing algorithms that have been analyzed for permutation or dynamic random routing and those that present deadlock, livelock, and starvation-free continuous routing algorithms. Since we are interested in practical routers, we restrict our attention to

the latter. A survey of the former is presented in [Lei92b], and a more complete treatment can be found in [Lei92a].

1.1.4 Classes of Routing Algorithms

Routing algorithms are often categorized by the kinds of paths messages may take through a network. In a *minimal* algorithm, a message is required to take a shortest path between its source and destination. See Figure 1.3 for an example. A *non-minimal* algorithm, however, allows a message to take a path that is not a shortest path. See Figure 1.4 for an example. When a message routes away from its destination, it is called a *deroute* or *misroute*. A minimal algorithm always moves a message in a *profitable* direction, since it never deroutes a message.

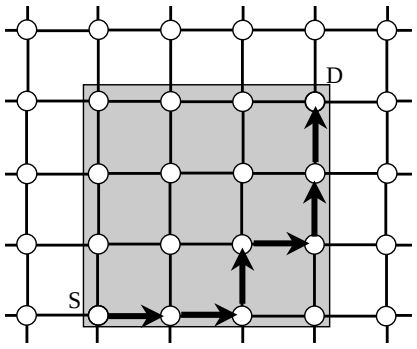

Figure 1.3: An example of a minimal path between a source (S) and destination (D) on a 2D mesh.

Routing algorithms are further categorized by the amount of adaptivity or flexibility allowed in the path selection. An *oblivious* algorithm specifies a single path between each source and destination in the network. Oblivious routers have no adaptivity. In *fully adaptive* algorithms, a message may take any shortest path between its source and destination, in *partially adaptive* algorithms a message may take a subset of the shortest paths, and in *non-minimal fully adaptive* algorithms a message

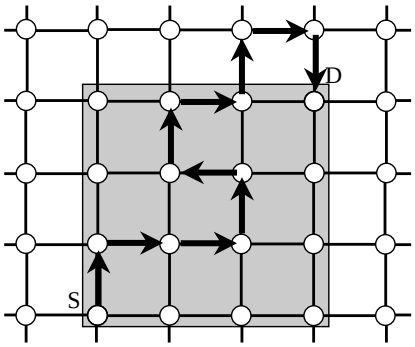

Figure 1.4: An example of a non-minimal path between a source (S) and destination (D) on a 2D mesh.

may take any shortest path as well as some longer paths. Specific routing algorithms are described later in Chapter 2 and in the literature [DYN97, GY93, NM93].

1.1.5 Router Properties

As mentioned earlier, deadlock, livelock, and starvation-freedom are three main properties every practical router must possess. We briefly discuss the different methods of achieving deadlock and livelock-freedom. All three properties are covered in more detail in Chapter 2.

There are several techniques for achieving deadlock-freedom. The first method, deadlock avoidance, ensures that deadlock can never occur. Currently, this is the most common method of providing deadlock-freedom.

Another method to achieve deadlock-freedom is pre-emption. Pre-emption allows messages to acquire resources like buffers already being held by other messages. This technique is more commonly used outside the domain of multicomputers, where routers are free to drop lower priority messages.

The last method, deadlock recovery, allows deadlock to occur. This method detects potential deadlocks and then invokes measures to resolve the deadlock.

For most routing algorithms livelock-freedom is not a problem. This is because most algorithms are minimal (restricted to shortest path routes) and (deadlock-free) minimal algorithms are naturally livelock-free.

Non-minimal algorithms usually maintain livelock-freedom in one of two ways. The first is by using a priority scheme, usually age, distance traveled, number of deroutes, or some other metric. When two messages conflict for a resource, the message with the highest priority takes the minimal route. The losing message gets derouted. An alternative technique uses randomization to select a message to deroute from the set of competing messages. In this case, the routing algorithm is *probabilistically livelock-free*. This means that the probability that a message remains in the network goes to zero in the limit.

1.2 Thesis Structure

This dissertation considers multicomputer network fundamentals and examines their effects on network performance. Chapter 2 describes three key properties of routing algorithms: deadlock-freedom, livelock-freedom, and starvation-freedom and surveys related work in multicomputer routing. Routing algorithms are almost always evaluated by simulation due to the complexity of an analytical analysis. The simulation methodology used in this work is detailed in Chapter 3. Chapter 4 defines and promotes the idea of multi-class routing. This chapter also introduces a novel multi-class routing algorithm called Triplex. Besides routing algorithms, another factor in network performance is the router implementation. Chapter 5 compares input and output driven routing which are two methods of implementing routing algorithms in a hardware router. Network performance is also effected by the type of flow control used. Chapter 6 evaluates several competitive routing algorithms using packet switching. Most algorithms have been evaluated with wormhole flow control, despite the higher throughputs achievable by packet routing. Finally, Chapter 7 discusses

conclusions and suggestions for future work.

1.3 Contributions

The main contributions of this thesis are summarized below.

- This thesis defines multi-class routing and formulates a method to evaluate multi-class routing algorithms. Multi-class routing algorithms increase performance by providing multiple kinds of routing in a single routing algorithm. This dissertation also introduces a novel integrated multi-class routing algorithm called Triplex. Triplex is the first triple-class routing algorithm. It supports oblivious, minimal fully adaptive, and non-minimal fully adaptive routing.

- This thesis presents the first non-minimal fully adaptive wormhole deadlock avoidance algorithm for tori.

- This thesis compares two methods of implementing a routing algorithm, as an input driven and as an output driven router. The two methods are similar, but surprisingly the output driven router almost always performs as well as or better than the input driven router. Most algorithms are implemented as input driven routers, even though almost all can be implemented as either.

- This thesis uses simulation to explore competitive packet routing algorithms. The results demonstrate the advantages of adaptive routing, that packet routing can achieve substantially higher throughputs than traditional wormhole networks, that packet routing algorithms could benefit from congestion control, and finally that non-minimal fully adaptive packet routing with controlled randomization is very effective at increasing throughput when compared with minimal fully adaptive and oblivious routers.

Chapter 2

KEY PROPERTIES OF ROUTERS

A large number of routing algorithms have been developed with varying complexity, resource requirements, and switching techniques. The complexity of the algorithms varies depending upon the techniques used to solve deadlock, livelock, and starvation. In multicomputer networks, substantial resources are required to resolve the problem of deadlocks. Because of this, multicomputer algorithms can generally be classified by the methods in which they provide deadlock-freedom.

2.1 Deadlock-Freedom

2.1.1 Deadlock Recovery

Routing algorithms can achieve deadlock-freedom in several ways. The first is by allowing deadlocks to occur and then to recover. This method is motivated by the observation that deadlock is a rare event, and hence should not require significant routing resources. Recovery schemes are used in both the Compressionless router [KLC94] and in the Disha router [KP95]. The former uses an abort and retry technique, similar to double-buffering [W+88], in which a message tries to acquire all the buffers between its source and its destination. If the message is successful, it will be delivered in time proportional to the length of the message, where all messages are assumed to be as long as or longer than one plus the number of buffers between the source and destination. Messages that are shorter than this are padded. If after some time, the message does not reach its destination, the source sends a kill signal along the message path causing the delivery attempt to be aborted. The aborted message is dropped

from the network, and another try is made at a later time. The Compressionless algorithm [KLC94] is adaptive, fault-tolerant, and does not require virtual channels, though it is more complex than traditional routers. If message lengths are short, bandwidth may be wasted on message padding. Compressionless routing, however, is one of the few adaptive routers that provides in-order delivery.

The Disha router [KP95] also has a time-out mechanism to detect potential deadlock situations. In Disha, a message may take an arbitrary route to its destination; but if it waits too long, it uses a special set of buffers called floating buffers to reach its destination. Besides the standard buffers, there is one floating buffer per routing node. In the sequential version, access to the floating buffers is mutually exclusive (by using a circulating token); and once a message obtains access, it takes a shortest path to its destination. In the concurrent version of Disha [KPD96], access to the floating buffers is no longer sequential. A message, however, must take a Hamiltonian path to its destination. The Disha algorithm is adaptive and does not use any virtual channels. The main drawback is that neither version scales efficiently, but like Compressionless routing, it may be suitable for small networks.

2.1.2 Deadlock Avoidance

Deadlock avoidance is a more commonly used method than deadlock recovery to achieve deadlock-freedom. Most of the schemes presented in the literature fall into this category. The simplest of these methods is deflection routing.

Deflection Routing

Deadlock is not a problem with traditional deflection routing since all messages route at every step. Because of this, deflection routers are sometimes referred to as hot-potato or desperation routers. Nevertheless, practical deflection routing has special requirements. With deflection routing, all routers must operate synchronously, and time-steps between routing decisions must be long enough to transmit an entire

packet to a neighboring node. This enables routers to forward every packet at each step. Deflection routing also requires livelock-protection since messages may deroute away from their destinations. Deflection routers do not require virtual channels and are classified as non-minimal adaptive routers. The HEP and Tera computers use deflection routing [Smi81, ACK+90].

Packet-Exchange Protocol

Another method used to avoid deadlock is the packet-exchange protocol [NS89]. The protocol is simple: if node a sends a packet to a neighboring node b, node a must also be able to accept a packet sent from node b. The packet-exchange protocol only works with packets and like deflection routing requires livelock protection, since an exchange may cause a packet to deroute. Routers using the packet-exchange protocol are non-minimal adaptive routers and do not require virtual channels. The Chaos router [KS94], as well as the priority router of Ngai and Seitz [NS89] employ the packet-exchange protocol to achieve deadlock-freedom. Both of these algorithms route packets on shortest paths whenever possible.

Directed Acyclic Graphs

One of the most common methods of avoiding deadlocks is to guarantee that buffer dependencies in the routing algorithm are acyclic. Informally, there is a *buffer dependency* from buffer a to buffer b if a message can use buffer a followed by buffer b. With wormhole routing, the term *channel dependency* is often used instead; though with special provisions, channels can be treated as buffers [SJ96, Dua95]. Dependencies are often analyzed by examining a *buffer dependency graph* (BDG) or *channel dependency graph* (CDG) of the routing algorithm. In this graph, the nodes are the buffers (channels) and the edges represent the dependencies between the buffers (channels). Many of the algorithms augment channels with multiple virtual channels. *Virtual channels* are a technique that creates the appearance of multiple physical channels.

Each virtual channel maintains a separate input and output buffer, and traffic on multiple virtual channels is multiplexed onto the (physical) channel. When virtual channels support non-blocking buffering for wormhole routing schemes, they are often referred to as (virtual) *lanes* [Dal92]. This idea is similar to lanes on a freeway, and allows messages to pass one another.

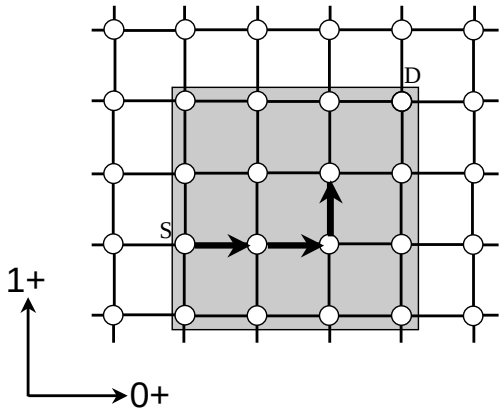

Figure 2.1: An example of a message that has corrected dimension 0, is currently correcting dimension 1, and still needs to finish correcting both dimensions.

The traditional method of avoiding deadlocks in wormhole networks is to insure that the channel dependency graph forms a directed acyclic graph (DAG). Dally and Seitz [DS87] showed that oblivious, wormhole algorithms are deadlock-free if and only if the channel dependency graph is acyclic. They extended a dimension-order oblivious hypercube algorithm, called e-cube routing [SB77], to other k-ary n-cubes. These algorithms, any of which is referred to as the oblivious algorithm, route a message by correcting its dimensions from the lowest to the highest dimension. A message *corrects* a dimension by routing on a shortest path towards its destination in this dimension. See Figure 2.1 for an example of correcting a dimension and Figure 2.2 for an example of a dimension-order oblivious route. The mesh and hypercube versions do not require virtual channels, while the torus algorithm requires two to break cycles within a single

dimension. These algorithms do not provide any adaptivity, but are simple and fast. There are many machines that use oblivious routing in their networks. Machines with oblivious hypercube networks include the Intel IPSC/2 [Nug88], oblivious mesh networks are found in the Intel Delta [Cor91], the Intel Paragon, the Alewife [ABC+95], the J-Machine [ND92], the M-Machine [FKD95], the Ametek 2010 [SAF88], and the SHRIMP [BLA+94] computers, while the Cray T3D [ST94] uses an oblivious torus.

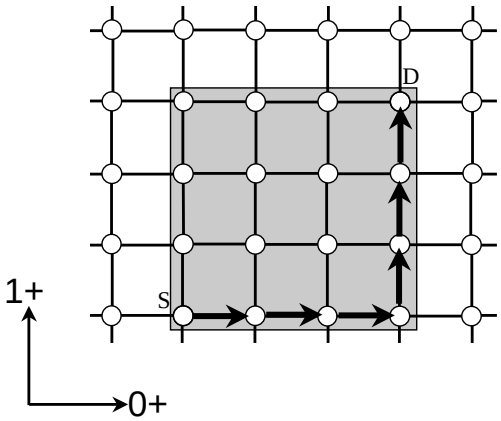

Figure 2.2: An example of a dimension-order oblivious path between a source (S) and destination (D) on a 2D mesh. This is the only path allowed between this source and destination.

Adaptive algorithms with acyclic channel dependencies are also deadlock-free. Dally presented a minimal fully adaptive wormhole algorithm for k-ary n-cubes without wrap edges (meshes) [Dal90a] which requires 2^{n-1} virtual channels per channel. The virtual channels are divided into 2^{n-1} groups. A message selects a group corresponding to the direction (positive or negative) in all but the lowest dimension that it needs to travel to reach its destination. This algorithm does not scale well, but is suitable for small-dimensional networks. The 2D mesh version, often referred to as the double-y algorithm [GN92b], is equivalent to algorithms that have appeared elsewhere [CK92, LH91].

Jesshope, Miller, and Yantchev observed that if networks are partitioned into

independent *virtual networks*, then it is sufficient to provide deadlock-freedom within each virtual network [YJ89]. Though presented in the context of packet routing, this idea was used by Linder and Harden to create minimal fully adaptive wormhole routing algorithms for k-ary n-cubes [LH91]. For torus networks, the Linder-Harden algorithm requires $(n+1)2^{n-1}$ virtual channels per channel, to support 2^{n-1} virtual networks, each with $n+1$ *levels*. A message selects a virtual network at injection corresponding to the direction (positive or negative) in all but the lowest dimension that it needs to travel to reach its destination. A message starts at level zero; and each time it uses a wrap edge, it switches to the next level. The mesh algorithm only requires 2^{n-1} virtual channels per channel since levels are unnecessary without wrap edges. As with the previous adaptive algorithm, the Linder-Harden algorithm does not scale well.

Using the Turn model [GN94], Glass and Ni developed routing algorithms which do not require the use of virtual channels. The Turn model restricts enough message turns (e.g. from one dimension to another) in the network so that the channel dependencies are acyclic. This results in minimal partially adaptive algorithms for mesh networks and non-minimal partially adaptive algorithms for tori [GN94]. One such algorithm for the 2D mesh is called west-first. With west-first routing, if a message needs to route west (in the negative x direction), it must finish routing west before routing in any other direction. Although the algorithm does not require virtual channels, the turn restrictions often cause non-uniformities in channel usage which result in poor performance that can be inferior to that of dimension-order routing. The 2D mesh algorithm can be extended to a minimal fully adaptive algorithm called mad-y by using two virtual channels per channel in the y dimension [GN92b]. The first set of virtual channels implements west-first routing and the second set prohibits east-south and east-north turns. Turns are allowed from the first set of virtual channels to the second, but not vice versa.

Planar adaptive [CK92] and hierarchical adaptive routing [LC94] (described later)

are partially adaptive algorithms that sacrifice adaptivity to reduce the crossbar complexity required of fully adaptive algorithms. The first algorithm, planar adaptive, achieves this for an arbitrary k-ary n-cube by restricting adaptivity to two dimensions (a plane) at a time. For example in a three-dimensional mesh, routing takes place in the first plane consisting of the lowest two dimensions. After the lowest dimension is corrected, routing proceeds in the second plane containing the highest two dimensions. Routing in each plane is fully adaptive, independent of the other planes, and uses a 2D version of Dally's algorithm (mentioned earlier) [Dal90a]. Nevertheless, almost any two-dimensional fully adaptive wormhole algorithm that uses a small number of virtual channels per channel could be substituted. The algorithm requires three virtual channels per channel for any mesh network with at least three dimensions. An extension to the torus topology is straight-forward and for example requires 6 virtual channels per channel if Duato's technique (described in the next section) is used. The planar adaptive algorithm can also support in-order delivery of messages, provided an extra bit is included in the message header. If the bit is set, the message takes an oblivious rather than an adaptive path through the network.

Boppana and Chalasani observed that all packet routing algorithms can be converted into wormhole routing algorithms in a straight forward manner. Each buffer in the node is replaced with a corresponding virtual channel for each channel of the node [BC93]. The packet routing algorithms (which count message hops) and the conversion are rather inefficient, resulting in algorithms requiring $O(nk)$ virtual channels per channel for k-ary n-cubes.

The routing algorithms in this section achieve deadlock-freedom by insuring that *all* the channel dependencies result in acyclic channel dependency graphs. This condition is not necessary and less severe restrictions will suffice. The following algorithms achieve deadlock-freedom even though their dependency graphs are not DAGs.

Underlying Deadlock-free Network

The next few algorithms are based on the informal observation that a message can route freely in one (or more) network(s), provided that the message can always move to a deadlock-free network if it gets blocked.

The first algorithm in this category is one presented by Dally and Aoki [DA93]. The algorithm is a non-minimal partially adaptive algorithm for arbitrary mesh networks [DA93]. Each message counts its dimension reversals, i.e. routes from a higher dimension to a lower dimension. The static version requires m virtual channels per channel and allows a message to be routed freely until it experiences m dimension reversals, the maximum number allowed. After this, the message switches to a (deadlock-free) dimension-order network*.

The dynamic version of the algorithm only requires two virtual channels per channel for a mesh; but before it can make a routing decision, it needs to obtain dimension reversal values of messages located in buffers in neighboring nodes. A message routes freely on the first channel until it must wait for another message with a smaller dimension reversal number. At this point, the message reverts to oblivious routing on the second virtual channel. Simulations have shown that the underlying dimension-order network quickly becomes a performance bottleneck. Deadlock-freedom is proved by demonstrating that the *packet-wait-for graph* (PWG) is acyclic. In this graph, the packets are the nodes of the graph, and there is an edge from packet p_i to packet p_j if packet p_i waits for a channel held by packet p_j.

The hierarchical adaptive routing algorithm [LC94] is similar to the Dally and Aoki algorithm, except that it only requires the last network to be deadlock-free. With hierarchical adaptive routing, a message is routed in a sequence of (linearly ordered) independent virtual networks, where the last virtual network is assumed to be deadlock and livelock-free by applying some other algorithm. As long as a message

*A similar scheme which counts battle-scars is used to prevent livelock in the HEP computer and is described in more detail in the section discussing livelock-freedom.

is guaranteed to progress from one virtual network to the next, it will arrive in the last virtual network and be routed in a deadlock-free manner. Liu and Chien also present a unique minimal fully adaptive algorithm for the two-dimensional mesh using this technique. The algorithm uses two virtual networks and requires two virtual channels per channel. The first network is minimal fully adaptive, while the latter uses the deadlock-free minimal partially adaptive positive-last algorithm which is based on the Turn model. With positive-last routing, a message must finish routing in the negative directions before it starts routing in the positive directions. Only messages that need to route in both dimensions in the same direction (i.e. either both in the positive x and y directions or both in the negative x and y directions) are allowed to move to the second network when blocked. This results in a deadlock-free minimal fully adaptive algorithm, since each network becomes fully adaptive and deadlock-free for messages which are allowed to block in the network.

The deadlock recovery algorithms are similar to algorithms employing an underlying deadlock-free network. With the deadlock-avoidance schemes, the decision to switch from one virtual network to another is based on the routing options available to a message, whereas with deadlock recovery switching occurs after a time-out. Time-outs are used either to abort message sends, to switch from one virtual network to another, or to switch to another set of buffers.

The Reliable Router is an implementation of an adaptive algorithm for a two-dimensional mesh [DDH+94]. The router is a combination of two dependent virtual networks: a minimal adaptive network and a fault-handling network and requires a total of five virtual channels per channel. The first network implements Duato's method (see the next section) and uses four virtual channels. Two of the four virtual channels provide adaptivity, while the other two support dimension-order routing for two priority classes. The second network uses a deadlock-free non-minimal partially adaptive Turn-based algorithm and requires one channel. The algorithm is very complex, but tolerates a single arbitrary fault in the network.

Acyclic Subgraphs

Duato created a framework for developing deadlock-free adaptive [Dua93, Dua95] (and fault-tolerant [Dua94]) wormhole routing algorithms by noticing that the requirement of an acyclic channel dependency graph is too restrictive. In his framework only a *subset* of the virtual channels need to define a connected routing *subrelation* which has acyclic channel dependencies. An algorithm is deadlock-free if the *extended channel dependency graph* (ECDG) is acyclic, i.e. a particular subgraph of the channel dependency graph is acyclic. The extended channel dependency graph includes edges representing direct and indirect channel dependencies. A *direct* channel dependency is a channel dependency between channels in the routing subrelation. There is an *indirect* channel dependency from one channel to another in the routing subrelation, if there is a path a message can take between the two channels only using channels that are not used in the routing subrelation. This ensures that deadlock is not created by using edges that are not in the acyclic subgraph. The idea is that a message can take paths that may have cycles, provided that it can aways take an acyclic path if needed.

Duato's k-ary n-cube algorithms are minimal fully adaptive and have two classes of virtual channels, which we refer to as restricted and unrestricted. The restricted virtual channel(s) support dimension-order routing, while the unrestricted virtual channels provide adaptivity. The algorithm only requires three (two) virtual channels per channel for the torus (mesh and hypercube), but requires buffer status information from neighboring nodes before making a routing decision. This insures a message header only blocks in an input buffer where it can select from multiple output buffers of the virtual channels. An equivalent torus algorithm called *-channels was presented in [GPBS94].

Using Duato's framework, Schwiebert and Jayasimha presented a minimal fully adaptive wormhole algorithm for n-dimensional meshes [SJ95]. Their algorithm uses

the Turn model for the restricted virtual channels and uses an additional virtual channel in all but the lowest dimension to provide fully adaptive routing. For example, the 2D mesh algorithm, called opt-y since it is optimal with respect to the number of virtual channels and routing restrictions for a 2D mesh, uses west-first routing on the restricted channels and an extra set of virtual channels in the y dimension. The general mesh algorithm marks one direction in each dimension but the last. Before using the first virtual channel in a dimension, a message must complete routing in the marked direction of all lower dimensions. The second virtual channel is used for minimal adaptive routes.

Permitting Cyclic Channel Dependencies

Unlike the previously mentioned approaches, the framework of Schwiebert and Jayasimha [SJ96] does not require a connected routing subrelation with an acyclic extended channel dependency graph. This is because cyclic dependencies, called False Resource Cycles or unreachable configurations, may exist without causing deadlock. Those that cause deadlock are called True Cycles. For an algorithm to be deadlock-free in this framework, the *buffer waiting graph* cannot have any True Cycles. In this graph, the nodes are the buffers, and there is an edge, denoting a *waiting dependence*, from buffer b_i to buffer b_j if a message can occupy buffer b_i and wait for buffer b_j. Their framework results in algorithms with fewer routing restrictions than Duato's technique since the dependencies are based on waiting rather than usage. Moreover, in a recent development Schwiebert has shown, contrary to the claim of Dally and Seitz, that there exists a deadlock-free oblivious routing algorithm with cyclic channel dependencies [Sch97]. It is likely that Dally and Seitz only considered suffix-closed deterministic algorithms, since deadlock-free oblivious algorithms with cyclic channel dependencies cannot be suffix-closed [Sch97]. Algorithms that use local information are usually suffix closed, whereas table-based routing algorithms are not always suffix closed, because they often try to balance traffic from different sources among various

routes.

Triplex is a novel algorithm presented in Chapter 4 which uses the framework of Schwiebert and Jayasimha to achieve deadlock-freedom. Triplex is a deadlock-free and livelock-free, non-minimal fully adaptive wormhole algorithm for tori that uses three virtual channels per channel. Triplex is able to support oblivious, minimal, and non-minimal routing by using two extra bits in the message header. Like other non-minimal algorithms, Triplex uses minimal routes whenever possible.

Table 2.1 summarizes the techniques and resources used to achieve deadlock-freedom for many of the algorithms described. If the algorithm supports hypercube, mesh, and torus algorithms the torus version is described, since it is the most difficult of the three to achieve deadlock-freedom. We have been lenient in categorizing the following algorithms as fully adaptive: HEP, Dally-Aoki, and Disha since in the final stages of these routing algorithms the paths are not fully adaptive. The HEP algorithm uses an Euler path. In the Dally-Aoki algorithm, a dimension-order path is used; while in Disha, either a Hamiltonian path or a fixed shortest path is used. We have also omitted describing algorithms that only apply to hypercubes [SJ96, Kon90]. Any of the wormhole algorithms can be converted to packet routing algorithms by adding packet buffers.

2.2 Livelock-freedom

Livelock is a distinct problem that plagues non-minimal adaptive routers and can exist if there is no bound on the number of times a packet may be derouted. *Livelock* occurs when a message moves continually, but never reaches its destination. Livelock is not a problem for (deadlock-free) oblivious or minimal routers because deroutes are not allowed. There are two basic methods of dealing with livelock: priorities and randomness.

Table 2.1: Comparison of deadlock-free routing algorithms for k-ary n-cubes including topology (topo), virtual channels (VC), adaptivity class (adapt), type of path, method of achieving deadlock freedom, and flow control (flow).

Algorithm	Topo	VC	Adapt	Path	Deadlock	Flow
HEP	all	0	fully	non-min	deflect	pkt
TERA	all	0	fully	non-min	deflect	pkt
Ngai-Seitz	all	0	fully	non-min	pkt-exchange	pkt
Chaos	all	0	fully	non-min	pkt-exchange	pkt
Dally-Seitz	all	2	obliv	min	acyclic CDG	worm
Dally	n-mesh	2^{n-1}	fully	min	acyclic CDG	worm
Linder-Harden	all	$(n+1)2^{n-1}$	fully	min	acyclic CDG	worm
Turn	all	0	part	non-min	acyclic CDG	worm
Planar-adaptive	all	6	part	min	acyclic CDG	worm
Hop-based	all	$O(nk)$	fully	min	acyclic CDG	worm
Mad-y	2D mesh	2	fully	min	acyclic CDG	worm
Liu-Chien	2D mesh	2	fully	min	acyclic CDG	worm
Dally-Aoki	n-mesh	2 or r	fully	non-min	acyclic PWG	worm
Duato, *-channels	all	3	fully	min	acyclic ECDG	worm
Reliable Router	2D mesh	5	fully	min	acyclic ECDG	worm
Schwiebert-Jayasimha	n-mesh	2	fully	min	acyclic ECDG	worm
Triplex	all	3	fully	non-min	no True Cycles	worm
Disha	all	2	fully	min	recovery	worm
Compres-sionless	all	0	fully	min	recovery	worm

2.2.1 Priorities

Priority methods have two basic variations; one uses fixed priorities while the other uses moving priorities. The fixed priority methods are usually time-stamp protocols, while the moving priority schemes are usually methods that count deroutes.

Time-stamp protocols [Nga89] require that each message carry the time it was injected into the network. Whenever a message needs to be derouted, the router avoids selecting the oldest packet within the router. Since every router avoids derouting the oldest local packet, the oldest message in the network is routed on a minimal path to its destination. Livelock is prevented since a packet will reach its destination or eventually become the oldest packet which is guaranteed to be delivered. Time-stamp fields are expensive to compare, since they must be large enough to be unique.

Moving priority methods are similar to time-stamps, but each message carries information that is updated, for example a count indicating how many times it has been derouted. There are several counting methods, and despite their similarities they often use different terminology to count deroutes or some equivalent metric.

The HEP computer counts battle scars [Smi81]. The message with the smallest battle scar is selected for derouting. This gives priority to messages which have suffered more deroutes. When a message experiences the maximum number of deroutes allowed, it is routed on an Euler path to its destination.

A similar method for the mesh which was proposed by Dally and Aoki counts dimension reversals, i.e. when a message routes from a higher dimension to a lower dimension (see Section 2.1.2) [DA93]. In the static version of the algorithm, a message may route freely until it reaches the maximum number of dimension reversals, at which point it is routed deterministically by the oblivious dimension-order algorithm.

The R2 switch prevents livelock by counting adaptive credits [DHR+94]. Each message starts with an adaptive credit equal to one less than the minimal path length to its destination. Each time a message makes a hop, its adaptive credit count is

decremented. A message with zero adaptive credits must take a minimal path to its destination.

Priority methods suffer from two problems: complexity and overhead. The routing decision at each node becomes more complicated since priorities must be compared and possibly updated before derouting. Also, the time-stamp or counts must be transmitted in the header of the message, consuming bandwidth that could be used for additional data.

2.2.2 Randomization

Randomization is an alternative to priorities for achieving livelock-freedom. A message is selected for derouting at random from a set of competing messages. Randomization provides *probabilistic livelock-freedom*, i.e. the probability a packet is in the network at time t goes to zero as t goes to infinity [KS94]. Since the likelihood of livelock is low to begin with, this provides a practical, though not a deterministic solution. The Chaos router [Bol93b], the deflection router found in the Tera computer [ACK+90], and the Triplex routing algorithm [FS97b] use randomization to provide probabilistic livelock-freedom.

2.3 Starvation-freedom

In addition to avoiding deadlock and livelock, routers must also protect against starvation. *Starvation* occurs when a node continually fails to inject a waiting message into the network. Starvation develops when buffers in the node are full making it difficult to inject a message into the network. For most routers this is not a problem, because either the injection frame is serviced in a fair fashion along with the other input frames, or there is a positive probability that a message in an injection frame is selected over other messages in the input frames.

Routers that use deflection routing or the packet-exchange protocol can be subject

to starvation. The traditional solution is to have a special message tag or *injection token* [Nga89]. When the token is delivered to its destination, the node will get an opportunity to inject a message. A single token can be circulated among all nodes in the network or each node can have an individual set of tokens. In the latter case, a node tags some of its out-going packets as round-trip messages. When a message returns, the node will be able to inject another message.

The injection-synchronization protocol of Ngai [Nga89] is an alternative to injection tokens. This technique uses local synchronization between neighbors to insure that the total number of packets injected by a node is no more than a fixed positive constant K from those of its neighbors. If a node would exceed this limit by injecting additional messages, it must wait for its neighbors to catch-up. Idle nodes do not prevent busy nodes from injecting, since idle nodes inject the equivalent of null messages.

Chapter 3

SIMULATION METHODOLOGY

In this chapter we present the details of the routing algorithm simulations. This includes the basic methodology used to estimate throughput and latency of the networks, the general model of the routing nodes, and a description of the workloads considered.

3.1 Routing Algorithms

This section briefly describes the routing algorithms used in the simulations. Specific details of the algorithms can be found in the references.

oblivious The Dally-Seitz oblivious dimension order routing algorithm is, as its name suggests, an oblivious algorithm which routes a message in dimension order from the lowest dimension to the highest dimension (e.g. in the 2D case x, then y) [DS87]. The torus version requires two virtual channels per channel to avoid deadlocks, while the hypercube and mesh versions do not require virtual channels. The oblivious algorithm is one of the simplest routing algorithms and can be used with wormhole or packet routing.

Duato The Duato algorithm is a minimal fully adaptive routing algorithm [Dua93, GPBS94]. It requires two sets of virtual channels which we refer to as restricted and unrestricted. A message takes an oblivious path using the restricted virtual channels and minimal adaptive paths using the unrestricted virtual channels. Two virtual channels per channel are required for the mesh and hypercube,

while three virtual channels are required for the torus. The Duato algorithm can be used with wormhole or packet routing.

Triplex The Triplex algorithm is a non-minimal fully adaptive routing algorithm [FS97b] and is described in more detail in Chapter 4. The algorithm also has two sets of virtual channels, restricted and unrestricted. Messages are not always required to use the restricted channels in an oblivious fashion. Besides oblivious paths, adaptivity is provided by the restricted virtual channels. Adaptivity is also provided by the unrestricted channels. Messages take a minimal path whenever possible. When congestion is experienced, messages wait for minimal paths. If congestion persists, messages may be allowed to deroute with a small probability. Triplex requires two (three) virtual channels per channel for the mesh and hypercube (torus). The Triplex algorithm can be used with wormhole or packet routing.

Chaos The Chaos routing algorithm is a non-minimal fully adaptive routing algorithm [KS94, Bol93b]. Messages take a minimal path whenever possible. When congestion is experienced, messages move to and wait in a central queue called the multiqueue. If congestion persists and the multiqueue becomes full, a random message is selected from the multiqueue for derouting. The Chaos routing algorithm does not require virtual channels. The algorithm is a packet routing algorithm only.

3.2 Router Description

The following describes the high level design and operation of the routers. Each router has an injection buffer, an ejection or delivery buffer, and an input and an output buffer for each channel or virtual channel in each direction. See Figure 3.1 for an example. Nodes enter the network in the injection buffer and leave the network via

the delivery buffer. A router is connected to a set of neighboring nodes by physical channels which allow communication in both directions.

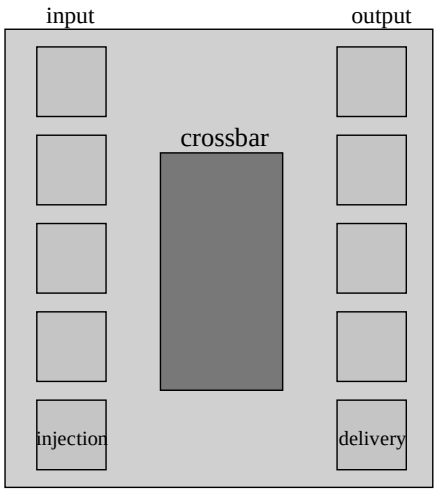

Figure 3.1: An example of a router with input buffers, output, buffers, crossbar, injection buffer, and ejection buffer.

Flow control is either wormhole or packet-switched. The latter uses virtual cut-through [KK79] routing to avoid store-and-forward latency penalties typically associated with packet routing. With virtual cut-through routing, a buffer may contain parts of two distinct messages, one that is being received and one that is being transmitted to another buffer. Nevertheless, when a message blocks, it is buffered entirely within a single buffer.

For wormhole routing, standard unidirectional (simplex) channels are used. For packet routing bidirectional (half-duplex) channels are shared between the two directions, though either unidirectional or bidirectional channels would suffice. See [Bol93a] for a discussion of the tradeoffs. To maintain a constant channel width, the unidirectional channels simulated are half as wide as the bidirectional channels. In the future it may be practical to use channels in both directions simultaneously [DLD93].

In this work, a *word* refers to the amount of data transferred in one direction between neighboring routers in one cycle. The buffers are one word for wormhole routing, and 20 words for packet routing. This is a minimal amount of buffering and greatly increases the throughput achieved by the network. The packet routing simulations use short messages of 20 words. Wormhole channels are half as wide as packet channels, and thus use 40-word short messages for the same amount of data. Some experiments use a combination of long and short messages, where the short messages are one-tenth the size and 10 times more frequent than the long messages.

Transmission of a word over a channel from an output buffer to a neighboring node's input buffer costs one simulated cycle. The actual cycle time depends on the technology used for implementation. Decoding and routing calculations are pipelined to allow the router cycle time to match that of the channel. The more complex algorithms require a larger pipeline depth. This study uses a *node latency* or routing decision time of three cycles for the oblivious router, and four cycles for the adaptive routers. These are based on the hardware design of several of the routers [Bol93b]. To keep the complexity manageable, during each cycle the router connects at most a single message from an input buffer to an output buffer. This does not sacrifice performance, since the probability of message arrivals (which is dependent on message length) is much smaller than the rate of service among the dimensions [Bol93b].

3.3 Workloads

Traffic is injected continuously. Messages are introduced at each node at every cycle with a constant probability specified by the applied load. For clarity, the applied load is normalized to the maximum sustainable load when an average of half the messages cross the network bisection, as with uniform random traffic. The *network bisection* is the minimum number of channels cut to divide the network in half. For a k-ary n-cube, a load of 1.0 specifies that a node can inject a message with average

length l every $kl/4$ cycles for tori, every $kl/2$ cycles for meshes, and every l cycles for hypercubes. This assumes half-duplex channels. The rate with full-duplex channels is twice as fast, since communication can take place in both directions simultaneously. For example on a 2D 256-node network with half-duplex channels and an average message length of 20 words, a load of 1.0 injects 1 message every 80 cycles for the torus and 1 message every 160 cycles for the mesh.

The traffic patterns considered are found in the literature, and are generally thought to be difficult, useful, or both. The following describes the traffic patterns simulated. Let the binary representation of the source node be $a_{n-1}a_{n-2}\ldots a_0$, and let $\overline{0} = 1$ and $\overline{1} = 0$.

Random - all destinations including the source are equally likely.

Bit Reversal permutation - destination is $a_0 a_1 \ldots a_{n-1}$.

Complement permutation - destination is to $\overline{a_{n-1}a_{n-2}\ldots a_0}$.

Perfect Shuffle permutation - destination is $a_{n-2}a_{n-3}\ldots a_0 a_{n-1}$.

Transpose permutation - destination is $a_{n/2-1}a_{n/2-2}\ldots a_0 a_{n-1}a_{n-2}\ldots a_{n/2}$.

Hot spot - ten randomly selected nodes are four times more likely to be chosen as destinations than the other nodes.

For the hot spot traffic, two different configurations were simulated. Assuming the nodes are labeled in row major order from 0 to 255, the hot spot nodes for case 1 are 6, 86, 121, 123, 152, 158, 186, 201, 216, and 236. For case 2 they are 51, 51, 70, 92, 124, 140, 155, 201, 245, and 254.

The traffic patterns illustrate different features. As mentioned earlier random traffic is simply a standard benchmark used in network routing studies. Hot spot traffic models cases where references to program data, such as synchronization locks,

bias packet destinations toward a few nodes. The complement is a particularly difficult permutation since all messages cross the network bisection. Given an imaginary x and y axis though the center of a mesh or torus network, the complement destination is the composition of the x and y axes reflection of the source. Because all messages cross the network bisection, the performance of the complement can be at most half that of random traffic. Perfect shuffle communication occurs in ascend/descend algorithms [PV81] while the transpose and bit reversal are important because they occur in computations such as matrix multiplication and fast-fourier transforms.

3.4 Performance Measures

The key performance measures for continuous traffic are saturation point, throughput, and latency. The *saturation point* is represented by the smallest applied load at which more messages are created than can be delivered by the network. The saturation point for a particular traffic pattern is approximated by running various simulations at applied loads in increments of .05.

Saturation point is important, since after saturation network performance is unpredictable and may even degrade. *Throughput* represents messages delivered per cycle and is normalized, like applied load, to the bisection bandwidth limitations of the network when half the processors inject messages that cross the network bisection. *Latency* measures the time a message takes to transit the network and does not include source queueing delay. Source queueing is measured separately since after saturation, the source queue length is not defined. Examining throughput and latency after saturation is useful even though real systems cannot sustain such loads, since it provides information about the systems ability to tolerate large bursts of traffic.

The simulator implements its own random number generator. This ensures that the methodology used is sound and guarantees repeatability of the streams of random numbers. The latter is useful for comparing various algorithms against the same

inputs and also for debugging purposes. The next section will briefly describe random number generation. Once the proper context and terminology are described, the random number generator employed by the simulator is presented.

3.5 Random Number Generation

The most widely used technique is a *linear congruential generator*. These generators use a recurrence relationship of the form

$$X_n = bX_{n-1} + c(\mathrm{mod}m). \tag{3.1}$$

where b, c, and m are non-negative integers called the *multiplier*, *increment*, and *modulus* respectively. The first term X_0 is called the *seed*. Successive applications of Equation 3.1 produce a sequence of integers X_1, X_2, ... between 0 and $m-1$. This sequence can be converted to numbers on the interval $(0, 1)$ by dividing by m, as specified in the following.

$$U_n = X_n/m$$

The algorithm produces a deterministic, dependent sequence. Nevertheless, if the parameters are chosen properly, the sequence will appear statistically like a random sequence. Numbers that appear random are often called *pseudo-random*. If c is positive, the generator is called a *mixed* congruential generator; and if c is zero, the recurrence is a *multiplicative* or *pure* congruential generator.

3.5.1 Multiplicative Congruential Generators

Multiplicative congruential generators are the best understood pseudo-random number generators and are also the most widely used. To generate a large source of numbers, it is also necessary to pick parameters that produce a long sequence before repeating. This repeating sequence is called a *cycle*, and its length is called its *period*.

There are number theoretic results that characterize the cycle length of these generators [Knu81]. In particular, if m is a prime integer and b is a primitive element modulo m, then the generator will cycle through the integers in the set $\{1, 2, \ldots, m-1\}$ and the period will be $m-1$. Zero is not included in the cycle, since after the first occurrence of zero, the generator will continue to produce zeros. A generator that satisfies these conditions is known as a *prime-modulus* multiplicative congruential generator.

Because computer words typically have 32 bits, the prime modulus $2^{31}-1$ has been well studied. There are several multipliers that have been found to have good statistical properties and large periods. If $b = 16807$, the generator is sometimes called the Learmonth-Lewis generator [LL74, Lea76] though it is attributed to Lewis, Goodman, and Miller [LGM69]. It has the form

$$X_n = 16807 X_{n-1} (\mathrm{mod} 2^{31}-1).$$

This generator is found in the APL, SPSS, and IMSL packages and is reliable and sufficient for most simulations. When it was discovered to have dependencies in its high (including three) dimensional structure, the multiplier $b = 397204094$ was proposed as an alternative [LL74]. This is considered one of the best for simulation work and is found in the GPSS and SAS packages [LO89].

Subsequently, Fishman and Moore [FM85] presented what they considered the five "best" multipliers for the modulus $2^{31}-1$ which produce generators with a cycle length of $m-1$. They selected them after searching through millions of multipliers (which are positive primitive roots of $2^{31}-1$) and running the latest set of statistical tests. The general form of the generator is

$$X_n = b X_{n-1} (\mathrm{mod} 2^{31}-1),$$

where b is one of the following multipliers:

950706376

742938285

1226874159

62089911

1343714438

The only caveat is that not all machines are able to evaluate bX_{n-1} without dropping digits or causing overflow. This can be tested manually by calculating the product $2147483646b$ of the multiplier and the largest element $(2^{31}-2)$ in the sequence, and comparing it with the computed result.

Recent work by Marsaglia and Zaman may soon become the preferred method of generating random numbers. They presented a new class of generators that can generate bits, integers, or reals, has provable uniformity for full sequences, and produces extremely long sequences, e.g. 2^{1376} 32-bit integers [MZ91].

3.5.2 Random Number Generation for the Simulations

For the simulations we used a multiplicative congruential generator (mentioned earlier) with multiplier 397204094 and modulus $2^{31}-1$. It has the form shown below.

$$X_n = 397204094 X_{n-1} (\mathrm{mod} 2^{31}-1).$$

Uniform random numbers on the interval $(0,1)$ were generated by multiplying the number generated by the inverse of the modulus.

$$U_n = X_n \frac{1}{2^{31}-1}.$$

Either multiplication or division can be used depending upon which operation is faster. The simulator also supports the Learmonth-Lewis generator by providing an optimized version of the algorithm developed for machines with 32-bit integers [BFS87].

Each of the nodes in the network models requires several independent streams of random numbers. For example, each node uses one stream for deciding whether to inject a message, and another to generate a random destination of a new message. A stream needs to use a non-overlapping section of the uniform random number sequence; otherwise, the numbers generated may no longer be uniform random in appearance. This is because random subsequences of the original sequence are not guaranteed to be independent or to have the same distribution as the original sequence. Because the simulator requires many long streams, the seeds for the streams for each node were computed offline and stored in a table. This was done by computing the interval size by dividing the period of the generator $2^{31} - 2$ by the number of steams required. Then the generator was called repeatedly and the values generated at the beginning of each interval were stored.

3.6 Gathering Statistics

For the simulations, we are primarily interested in obtaining values for the steady-state mean throughput and latency of the network.

The simulator uses a standardized time series procedure called batch means for computing confidence intervals for the steady-state mean. In this section, we briefly introduce the terminology needed and present the methodology used.

3.6.1 Batch Means Method

We chose the method of standardized time series since it constructs asymptotic confidence intervals for the steady-state mean from a single simulation run. The idea of the batch means method is as follows. If the simulation run is large enough, it can be divided into non-overlapping pieces of equal length called a *batch*, and each batch can be treated as an independent simulation. The mean of each batch is computed and treated as an independent estimate of the steady-state mean. The technique

applies to $\Re$-valued stochastic processes, but for the purposes of the simulator, only discrete-time stochastic processes are needed.

Given a discrete-time stochastic process $\{Z_k : k \geq 0\}$ that has steady-state mean r, the *sample batch means* are defined as follows:

$$\bar{X}_i(n) = \frac{m}{n} \sum_{j=1}^{k} Z_{(i-1)k+j},\ i = 1, 2, \ldots, m$$

where n is the length of the simulation run and k is the batch size. This results in m batches of length k. A sample batch mean is simply the mean of the observed data for a particular batch. In general the batch means $\bar{X}_1(n), \bar{X}_2(n), \ldots, \bar{X}_m(n)$ are not independent, identically distributed (i.i.d.) random values. If Z satisfies the appropriate conditions (a Functional Central Limit Theorem holds), as is often assumed, then the batch means tend to be normally, independent, and identically distributed [GI90]. Define $r(n)$ as follows.

$$r(n) = \sum_{i=1}^{m} \frac{\bar{X}_i(n)}{m}$$

If $\bar{X}_1(n), \bar{X}_2(n), \ldots, \bar{X}_m(n)$ are i.i.d. normally distributed with $E[\bar{X}_i(n)] = r$ (the expected value of a batch mean is equal to the steady-state mean r), the random variable

$$\frac{r(n) - r}{\frac{1}{\sqrt{m}} S_m(n)}$$

has a Student t distribution with $m - 1$ degrees of freedom where $S_m^2(n)$ is defined as follows

$$S_m^2(n) = \frac{1}{m-1} \sum_{i=1}^{m} (\bar{X}_i(n) - r(n))^2 \tag{3.2}$$

The quantity S_m^2 is the *sample variance* of the batch means. The $100(1-\alpha)\%$ asymptotic confidence interval for the steady-state mean r is

$$[r(n) - t_{(m-1,\alpha/2)} \frac{S_m(n)}{\sqrt{m}}, r(n) + t_{(m-1,\alpha/2)} \frac{S_m(n)}{\sqrt{m}}] \tag{3.3}$$

where $t_{(m-1,\gamma)}$ denotes the $(1-\gamma)$-percentile of the Student t distribution with $m-1$ degrees of freedom. Therefore,

$$\lim_{t\to\infty} P\{r(n) \in \hat{I}(n)\} = 1-\alpha$$

where $\hat{I}(n)$ is the interval in Equation 3.3. In other words, for large t the interval with random end points $r(n) - t_{(m-1,\alpha/2)}\frac{S_m(n)}{\sqrt{m}}$ and $r(n) + t_{(m-1,\alpha/2)}\frac{S_m(n)}{\sqrt{m}}$ contains the unknown constant r approximately $100(1-\alpha)\%$ of the time. Notice that the half-length of the confidence interval decreases at the rate of $O(1/\sqrt{n})$. Most estimators used in simulations are asymptotically normal, and hence have the same order of convergence.

3.6.2 Simulator Statistics

For the simulations we compute the sample batch mean for each batch as it is finished. These sample batch means are then used to compute the sample mean and variance of the batch means. From this, the confidence interval for the steady-state mean is computed. Instead of using Equation 3.2, however, the simulator computes the sample variance using a single pass over the data, though both methods are supported in the code. The simulator reports the estimate for the expected value of the steady-state mean as well as its 95% confidence interval. Values for $t_{(i,\alpha/2=.025)}$ are stored in a table for a range of values of i.

Chapter 4

TRIPLEX ROUTING

4.1 Introduction

This chapter describes multi-class routing and then introduces a novel multi-class routing algorithm called Triplex. A multi-class router has two functions. First it should provide several classes of routing. This is useful since no single class of routing is suitable for all applications. Second, multi-class routers should simplify the network interface or system software by providing services with the routing that might otherwise be implemented in these other layers. Nevertheless, nearly all known routers support a single class of routing, and few provide services such as in-order delivery required by many applications.

4.2 Motivation

Oblivious routing continues to be the routing algorithm of choice [Nug88, Cor91, ABC+95, ND92, FKD95, SAF88, BLA+94, ST94]. Oblivious routers are simple and provide in-order delivery of messages. Since oblivious routers have no adaptivity, they perform poorly in congested networks and fail when faults are present in the network.

Minimal adaptive routers are more complex than oblivious designs, but can achieve higher throughputs, even with small amounts of congestion. Minimal adaptive algorithms provide some fault-tolerance, though they rarely provide in-order delivery since the flexibility in route selection allows messages to pass one another. Minimal adaptive algorithms often suffer from severe throughput degradation in heavy congestion [NS94].

Non-minimal algorithms can provide the highest throughputs because they tolerate large bursts of traffic and heavy congestion. Like minimal adaptive routers, delivery is rarely in-order. Non-minimal routers are also fault-tolerant, but are more complex due to livelock-freedom requirements. In cases of extreme congestion, non-minimal routers occasionally experience large increases in latency due to the overuse of non-minimal routes.

There are performance-complexity trade-offs to consider when selecting a class of routing. If performance of the simplest oblivious algorithm is sufficient then the choice is easy. Otherwise, an adaptive routing algorithm is preferable. The price of adaptivity is increased complexity and out-of-order delivery. In-order delivery, however, is necessary to support efficient synchronization [WSC+95], checkpointing [LNP91], memory consistency [LHH91] (e.g. via cache coherency algorithms or barriers), and determinism in parallel computations. Furthermore, providing in-order delivery for adaptive algorithms requires special hardware to avoid the substantial latency costs of software reordering. Although a few such network interface designs have been demonstrated [BJM+96, MBES94], there is reluctance to provide such a complex solution.

Multi-class routing is an alternative method to provide the benefits of adaptive routing and is beginning to gain acceptance, as evidenced by the routing algorithms in the Cray T3E [ST96] and the lastest IBM SP2 routers [ASAA96]. Furthermore, since no router supports the needs of all applications, applications would benefit from a router that supports several kinds of routing simultaneously. Applications also benefit if services like in-order delivery are provided by routers. At present, almost all routers provide a single kind of routing.

4.3 Overview

A multi-class router is not simply a combination of several routers, but rather is an integrated router that shares resources to provide multiple kinds of routing. Multi-

class algorithms should also simplify the network interface or system software. This can simplify the network interface and reduce latencies for services that would otherwise be provided in the network interface or system software. Although the idea of a multi-class routing algorithm is perhaps obvious, and several routers have some of these features, the advantages of multi-class routing have not been promoted and multi-class performance has been ignored. Thus, our contribution lies in defining this notion, describing the methodology to evaluate such routers, demonstrating their usefulness, and introducing the Triplex routing algorithm.

Triplex provides oblivious, minimal fully adaptive, and non-minimal fully adaptive routing for both wormhole or packet-switched flow control, including virtual cut-through and store-and-forward techniques. The choice of class may be specified at system boot-time, by application, or individually by message. Dynamic class selection may be useful when compile time information is available [Fel93], enabling the system to select the best class for the expected traffic.

Although we consider point-to-point networks in a parallel (multi)computer context, the idea of multi-class routing is applicable to other types of networks as well.

4.4 Multi-class Routing

A multi-class router is a router designed specifically to provide routing for several classes of traffic, where each class of traffic requests distinct algorithmic constraints. Furthermore, messages of different classes may reside in the network at the same time.

The key to an multi-class router is the unification of the classes or modes of traffic it provides. Each routing class shares resources, such as buffers, with other classes thus reducing the cost and complexity of multiple classes. Although classes share resources, different classes of messages can exist in the network simultaneously. The typical method to distinguish messages of different classes only requires adding a few bits to the message header.

For example, consider the multi-class Triplex algorithm which will be described in more detail in Section 4.6. It provides three classes of routing: oblivious Triplex, minimal adaptive Triplex, and non-minimal adaptive Triplex. In-order delivery is provided by the oblivious class, which is the most restrictive class of the three. By providing support for an oblivious class, Triplex is an adaptive router that does not require a special network interface to reorder messages. Examples of other multi-class routers can be found in Section 4.8.

4.5 Evaluation

This section outlines a methodology for evaluating multi-class routing algorithms. Although multi-class algorithms have appeared in the literature, the performance of multiple classes has been ignored. Evaluating a multi-class routing algorithm is difficult because its advantages include both quantifiable properties such as improved throughput, as well as more qualitative properties such as in-order delivery. Obviously, a multi-class router's single class performance cannot exceed the best known single class router; otherwise, a faster router for the class would have been discovered*. The goal, however, is to be competitive for a combination of all the traffic classes. Conversely, if the ability to select routing classes significantly degrades performance from single class routing, then perhaps the versatility is not worth the price.

4.5.1 Single Class Traffic Performance

One figure of merit we use is the amount by which a multi-class router's performance departs from the best router for each class. This is a rather stringent criterion, unless a single class router provides sufficient performance†. This is because single class comparisons ignore the advantages of multi-class routing such as selective in-

*This may occur, however, when comparing a new multi-class router with an existing single class router.

†If the network will always be lightly loaded, a simple single class oblivious router will suffice.

order message delivery, as well as selective message adaptivity. The flexibility of multiple classes of routing can produce superior network performance compared with a single class router, even when each class of the multi-class router does not perform as well the comparable single class router. Nevertheless, single class comparisons give an indication of the quantitative costs of multi-class routing by estimating the performance range of the router.

4.5.2 Estimates of Mixed Class Traffic Performance

This is not the complete picture, though. A more complete evaluation considers the performance of mixed classes of traffic, i.e. when several classes of messages reside in the network simultaneously. Even if several classes of traffic can be used by the application, a single class router would need to provide the most restrictive kind of routing required. Therefore with mixed class traffic, the multi-class routing algorithm is compared with the most restrictive single class routing algorithm necessary.

Because the performance space is so large, the first approach is to estimate mixed class performance. Mixed class performance can be estimated by computing the average of the single class performance results of the multi-class routing algorithm. The average is simply weighted to reflect the traffic mix of interest. With the mixed class performance estimates, it is then possible to compute an estimate of the range of traffic mixes for which the multi-class routing algorithm maintains its advantage over the most restrictive single class routing algorithm.

4.5.3 Mixed Class Traffic Performance

Experiments with mixed class traffic can also be simulated. This gives a more accurate representation of performance, since each class of traffic actually shares router resources while the estimates do not assume this. Such experiments allow comparisons of the performance of the most restrictive single class router with the performance of the multi-class router under mixed class traffic loads. Although it may be tedious to

determine the range of loads and traffic mixes for which the multi-class router is superior to using only the most restrictive single class router, representative experiments can at least verify the estimates.

4.5.4 Other Considerations

There are also qualitative differences between the routers that are difficult to measure. This includes factors such as in-order delivery. Providing these services with the routing helps simplify the network interface or software. Simplifying the network interface can result in great savings in hardware complexity and may result in a faster, cheaper, more manageable network interface. Alternatively, providing features in routers that might otherwise be left to software provides a big latency savings. Evaluating the qualitative advantages is left to the architects and systems engineers.

A particularly useful class of multi-class routers is one that provides in-order delivery as well as fully adaptive routing. Fully adaptive routers generally achieve higher throughputs and saturate at higher loads than deterministic routers, while in-order delivery is usually assumed by programmers and helpful for providing efficient, low cost synchronization tasks.

Unlike the typical case where distinct applications use different classes of routing, there are also applications that may use several classes of routing simultaneously. Some of them may need to send additional messages to provide ordering among the different phases of adaptive computation. Alternatively, it is possible that if minimal adaptive routing is used, a small amount of additional hardware support in the router may be useful in providing such ordering. The evaluation of these issues, however, is not considered in this work.

4.6 The Triplex Algorithm

In this section, the Triplex multi-class algorithm is introduced and evaluated according to the method outlined in the previous section. Before describing the Triplex algorithm, a few additional definitions are defined. The routing terminology and network assumptions have already been described in Chapter 3.

A *waiting buffer* is a buffer that a message can wait to acquire when all other buffers specified by the routing relation cannot be selected. A routing algorithm is *wait-connected* if a message always has at least one waiting buffer [SJ96].

Triplex is a novel, non-minimal fully adaptive deadlock-free routing algorithm for k-ary n-cube networks. Although the algorithm is non-minimal, no message is forced to take a non-minimal route. Thus, a message may choose the kind of routing it prefers from the oblivious, minimal, or non-minimal class. The algorithm gains adaptivity by avoiding the traditional requirement of a connected set or subset of channels with static acyclic channel dependencies. Depending upon the implementation, the algorithm is also deterministically or probabilistically livelock-free. For presentation clarity the mesh algorithm is presented first, and the torus algorithm follows. There are two versions of the algorithm: one for wormhole routing and a slightly more flexible and less complex one for packet-switched routing. The mesh algorithm uses two virtual channels per channel, while the torus algorithm uses three virtual channels per channel.

The dimension-order Dally-Seitz oblivious, wormhole routing algorithm [DS87] (DO) will be used as a subroutine. This algorithm is wait-connected and deadlock-free, and routes a message from the lowest dimension to the highest dimension, where the direction in each dimension is chosen to make the message route minimal. Although the specific DO rules differ slightly for the mesh and the torus, the particular details are not relevant. Hence, the mesh algorithm will not be distinguished from the torus algorithm except by context.

The virtual channels are partitioned into two classes. The *restricted* class refers to the set of virtual channels used by the *DO* subroutine (as well as by other messages), two virtual channels per channel for the torus and one for the mesh, while the *unrestricted* class contains the remaining virtual channel for a total of three virtual channels for the torus and two for the mesh.

A buffer is *unrestricted* if its corresponding virtual channel is unrestricted, is *restricted* if its corresponding virtual channel is restricted, and is called a *wrap* buffer if its corresponding virtual channel is a wrap channel[‡, §]. For wormhole routing, a buffer representing a virtual channel is considered *empty* when both the input and output buffers that compose the virtual channel are empty. This requires state information from a neighboring node. For packet routing, a non-full buffer is considered *empty* since once a packet header progresses to a buffer, it is guaranteed to be completely accepted into this buffer, even if the packet becomes blocked.

4.6.1 The Mesh Algorithm

For ease of explanation, we describe the mesh algorithm first, both informally and then as a list of routing rules. The packet-switched version is presented first and is followed by the wormhole version.

Packet Triplex on the Mesh

A message can route according to dimension-order rules (DO) at any time using restricted buffers (Rule 1). A message may also take any minimal (Rule 2) or non-minimal route (Rule 3), using an empty unrestricted buffer. Moreover, a message can use any buffer in a dimension greater than the lowest dimension that needs correcting

[‡]A wrap channel in dimension i is simply a single distinguished channel in dimension i, though by convention it is often the channel between the last and first node of dimension i.

[§]For packet routing, only the output buffer corresponding to the wrap virtual channel is considered a wrap buffer.

provided that it needs to move in the negative direction of the lowest dimension it needs to correct (Rule 4). This includes both minimal and non-minimal routes.

Let l be the lowest dimension that a message still needs to correct. A message selects a path in the mesh at its current location subject to the following routing restrictions.

1. A message may route according to DO (using restricted buffers).

2. A message may use an empty unrestricted buffer on a minimal path.

3. A message may deroute using an empty unrestricted buffer.

4. A message that needs to route in the negative direction of l may use any (restricted or unrestricted) empty buffer in a dimension i, $i > l$. This includes non-minimal routes.

If none of these buffers is available, a message waits on all the buffers it needs including the one specified by DO. The unrestricted buffers provide most of the adaptivity, while the restricted buffers provide deadlock-freedom and in some cases extra adaptivity, since the restricted channels are not limited to messages taking dimension-order routes. For examples of the rules, see Figures 4.1 - 4.4. In these figures, the input buffers have been omitted. The restricted buffers appear shaded, while the unrestricted buffers are unshaded.

There are three classes provided by Triplex: oblivious, minimal, and non-minimal. The oblivious (minimal) class also called *oblivious Triplex* (*minimal Triplex*) results from restricting Triplex to oblivious (minimal) routes. Triplex refers to both the multi-class routing algorithm and in particular the non-minimal fully adaptive Triplex class. Unless otherwise specified, Triplex refers to the wormhole version of the algorithm.

Each class routes by selecting buffers specified by the routing rules of its class. The non-minimal Triplex class selects from all four rules, while the minimal class selects

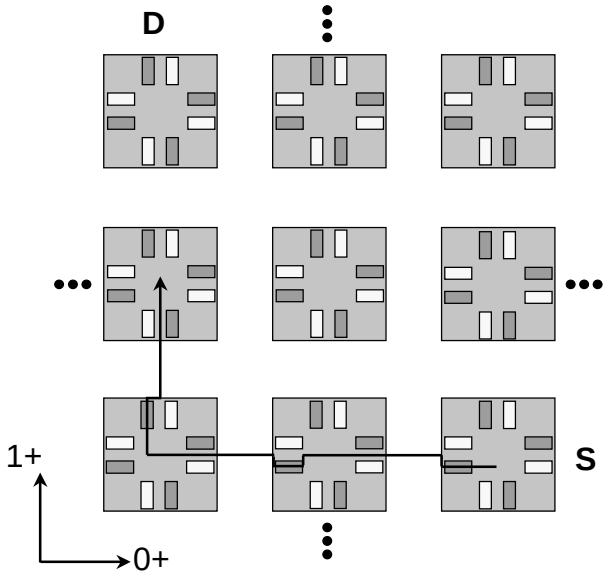

Figure 4.1: An example of a route allowed by Rule 1 between a source (S) and destination (D).

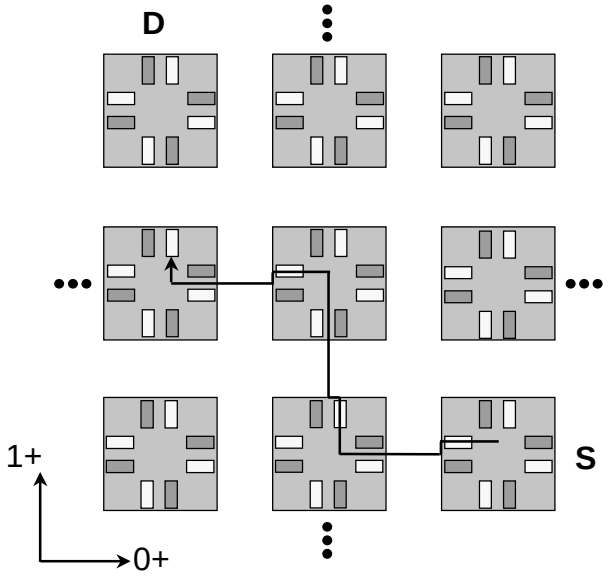

Figure 4.2: An example of a route allowed by Rule 2 between a source (S) and destination (D).

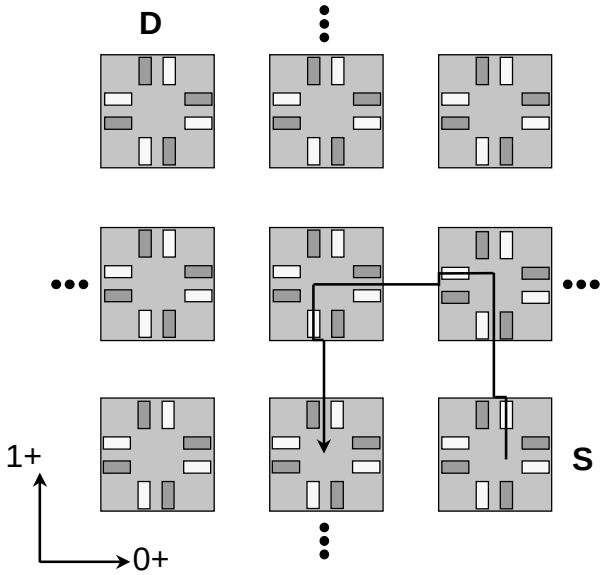

Figure 4.3: An example of a route allowed by Rule 3 between a source (S) and destination (D).

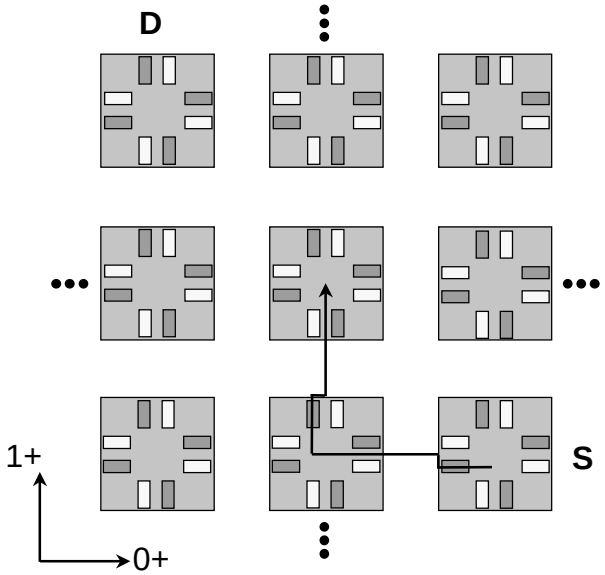

Figure 4.4: An example of a route allowed by Rule 4 between a source (S) and destination (D).

from Rules 1, 2, and the minimal routes of Rules 3 and 4. The oblivious Triplex class selects from the buffer specified by Rule 1. See Figure 4.5 for an example of a non-minimal class message route.

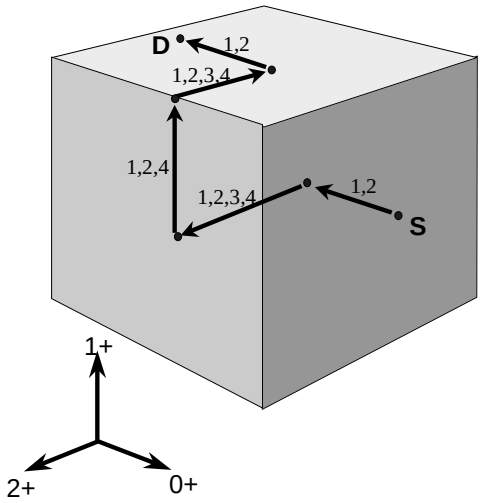

Figure 4.5: This is an example of a Triplex message path on a 3D mesh between a source (S) and destination (D). Each edge is labeled by the rules which apply to the route.

Notice that the buffers are integrated. There are no buffers restricted to a single class. Also there are no buffers limited to messages taking dimension-order paths as in the Duato algorithm. Because a more flexible deadlock avoidance technique is used, Triplex provides additional adaptivity over fully adaptive algorithms like Duato. For example, the last rule increases the number of routing choices by allowing messages to violate dimension-order routing in the restricted buffers, either with minimal or non-minimal routes. This creates cycles in the buffer dependencies. Thus, there is no acyclic ordering of the restricted buffers as in dimension-order routing or Duato. Deadlock is avoided by providing an escape route [Gün81] for every message. See Figure 4.6 for an example of an escape route.

The idea of the deadlock-freedom proof follows. Details can be found in Appendix A. A message always maintains an escape route. For Triplex, the escape

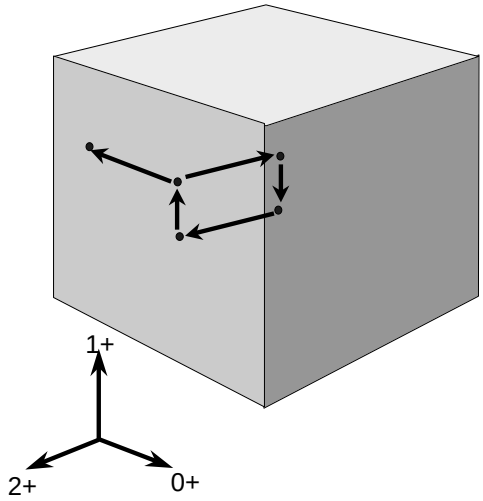

Figure 4.6: An example of a cyclic dependency that does not cause deadlock. One of the messages in the cycle waits on a message that is not in the cycle. This is an escape route for the cycle.

route is in the lowest dimension that it needs to correct. A message insures this path is deadlock-free by following special routing rules for dimension l which are slightly more restrictive than the general routing rules. When a message gets delayed, it avoids deadlock by waiting for buffers that include a specific deadlock-free buffer in dimension l. The proof shows that cycles like the one depicted in Figure 4.7 cannot exist unless a message violates one of the routing rules.

Theorem 1: The packet-switched version of the Triplex routing algorithm, *packet Triplex*, for the mesh is deadlock-free.

Wormhole Triplex on the Mesh

The (wormhole) Triplex algorithm is complicated by buffer dependencies caused by arbitrary length messages. Since messages can only wait on restricted buffers, any cycle in the buffer dependencies is created from waiting dependencies between restricted buffers. These dependencies can be direct, resulting from a message in one restric-

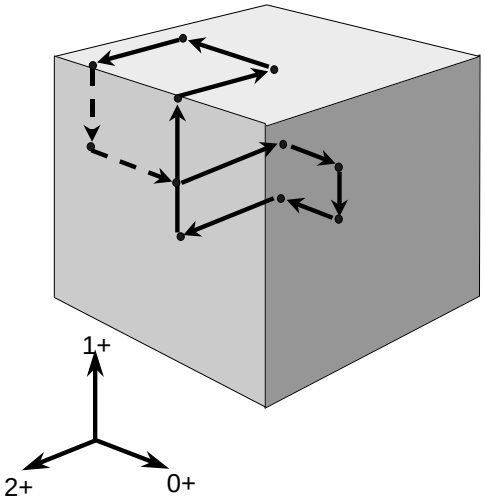

Figure 4.7: A potential cycle caused by Triplex routing. The proof shows that the turn highlighted by dashed lines is not allowed.

ted buffer waiting immediately for another restricted buffer; or they can be indirect, caused by a message which occupies a restricted buffer followed by one or more unrestricted buffers and waits for another restricted buffer. The wormhole algorithm is slightly more restrictive than the packet version. The empty buffer criterion is stricter and Rule 3 is replaced by Rule 3w. Rule 3w states that a message may deroute to an empty unrestricted buffer in a dimension greater than the lowest dimension it needs to correct.

3w. A message may deroute using an empty unrestricted buffer in a dimension i, $i > l$.

Theorem 2: The Triplex routing algorithm for the mesh is deadlock-free.

4.6.2 The Torus Algorithm

The torus algorithm is more restrictive than the mesh algorithm since the wrap edges pose an additional threat of deadlock. Let l be the lowest dimension in which a

message still needs to route. A message m satisfies the *wrap-free* property if m is guaranteed to have a minimal path specified by Rule 1 to its destination position in dimension l, where the path consists of waiting buffers that never contain a message that uses or has a waiting dependence on the wrap buffers in the negative direction of dimension l. Because a message always waits on a set of buffers that includes the one specified by DO, the wrap-free property is easy to determine. For Triplex, the wrap-free property tests whether a message does not use a wrap buffer or has already used a wrap buffer in dimension l in the negative direction. This is sufficient due to the structure of the waiting dependences of the DO algorithm. A message selects a path in the torus subject to the same routing restrictions as the mesh, except that Rule 4 is replaced with Rule 4t. Rule 4t permits a message to use any empty buffer in a dimension greater than l provided it needs to correct l in the negative direction, and it does not need or no longer needs a wrap buffer in dimension l. There is nothing special about the negative direction. The algorithm is simply trying to minimize the number of routing restrictions, and this additional flexibility can only be allowed in a single direction. Otherwise, there is a potential for deadlock.

4t. A message that needs to route in the negative direction of l may use any (restricted or unrestricted) empty buffer in a dimension i, $i > l$ if it satisfies the *wrap-free* property. This includes non-minimal routes.

Theorem 3: The packet Triplex routing algorithm for the torus is deadlock-free.

The wormhole Triplex algorithm on the torus is the same as packet Triplex on the torus, except that Rule 3 is replaced with Rule 3w.

Theorem 4: The Triplex routing algorithm for the torus is deadlock-free.

For Triplex, any route other than the one specified by DO is optional. Thus, the algorithm or class minimal Triplex (oblivious Triplex) class is also deadlock-free.

Corollary 5: The minimal Triplex and oblivious Triplex algorithms are deadlock-free for packet and wormhole routing.

Message delivery of the oblivious Triplex class is in-order since there is a unique path between each source and destination, and with blocking buffering, messages cannot overtake one another.

Corollary 6: Oblivious Triplex delivers messages in-order.

A message is *never* forced to take a non-minimal route. Therefore, livelock-freedom can be achieved in several ways. The first, not so interesting alternative, is to use the minimal adaptive version of the algorithm. The second method is to use a counting scheme which allows each message a maximum number of deroutes, after which the algorithm reverts to an alternative kind of routing like oblivious [DA93] or by following an Euler path [Smi81]. These schemes require a message to carry and update additional routing information that records the number of misroutes experienced. The third scheme is similar, but assigns a fixed priority or time-stamp [Nga89] to each message. It also requires a message to carry additional routing information. Fixed priorities are expensive to compare, since they must be large enough to be unique. The fourth alternative is to use a probabilistic scheme, like the Chaos router [KS94] or Tera computer router [ACK+90], which guarantees a message always has a chance of taking a minimal route. This is simple to implement but results in a probabilistically livelock-free algorithm. Nevertheless, this is sufficient in practice. See Section 2.2 for more details about achieving livelock-freedom and Appendix A.2 for the livelock proof.

Theorem 7: If Triplex deroutes a message with probability p, $0 \leq p < 1$, then Triplex is probabilistically livelock-free.

Unlike the Chaos or Tera computer router, the Triplex algorithm is never forced to deroute. Therefore the probability of derouting could be set at machine boot-up,

by application, or individually based on the specifications of a particular message. In the latter scheme, a message could specify oblivious, minimal, or non-minimal routing at the cost of two bits in the header. The cost may be reasonable, since routing possibilities can be computed before the selection bits arrive. Individual class selection might be useful when in-order delivery is required for some messages, the flexibility of adaptive routing is needed by others, or when compile time information is available [Fel93] and a particular kind of routing is preferred for the expected traffic.

4.7 Comparisons

Performance of the Triplex algorithm is explored by simulation. The algorithm is compared to three other routers: the Dally-Seitz oblivious router [DS87], the Duato [GPBS94, Dua93] router, and the Chaos [KS94] router[¶] using a flit-level simulator of a 256-node two dimensional (2D) torus (16-ary 2-cube) network. Each of the three routers represents one of the routing classes offered by the Triplex router. The first algorithm is the dimension order algorithm and has no adaptivity. The second is a minimal fully adaptive algorithm which allows any shortest path. The last is a non-minimal fully adaptive router which prefers any minimal path, but occasionally deroutes a message in the presence of severe congestion. More details are described in Chapter 3.

4.7.1 Router Description

The following briefly describes the high level design and operation of the routers. Each router has an injection buffer, a delivery buffer, and an input and an output buffer for each virtual channel in each dimension in each direction. The oblivious, Duato, and Triplex algorithms have 2, 3, and 3 virtual channels per channel, respectively, yielding

[¶]There was no qualified non-minimal algorithm for the torus, except the Chaos router which does not support wormhole routing.

a total of 18, 26, and 26 buffers per node for each router. The oblivious router could be given an extra set of virtual lanes [Dal92], but then it would not support in-order delivery as desired. The Chaos router does not use virtual channels. Instead it uses 10 buffers, one for each dimension in each direction, an injection and a delivery buffer, in addition to a central queue with 5 buffers, for a total of 15 buffers per node.

The node latencies to make a routing and selection decision are three cycles for the oblivious and four cycles for the more complex adaptive algorithms [BFSar]. Although the adaptive algorithms appear quite complex to describe, the additional calculations required beyond that of the oblivious router are actually easy to compute. The computation determines, in parallel, the dimension and direction of all of the minimal routes, the lowest dimension minimal route, and the non-minimal routes which are the remaining directions in each dimension. Then, two masks are created, one for the non-minimal routes and the other for all of the minimal routes, except for the lowest dimension minimal route which uses a restricted buffer. When the selection bits arrive, the appropriate mask is combined with the routing choices to obtain oblivious, minimal, or non-minimal routes.

The buffers are one word long for wormhole routing, while for packet routing, buffers are 20 words long. Besides increasing throughput, packet routing also enables the comparison of Triplex with a scalable non-minimal fully adaptive algorithm. Table 4.1 contains a summary of the differences among the routing algorithms examined.

Before moving a message from an input to an output buffer within a node, the Duato and (wormhole) Triplex algorithms require the full/empty status of the neighboring node's input buffer corresponding to the output buffer under consideration. This status is already present at the node, after a one cycle propagation delay, since it is used for flow control of the channels. Nevertheless, there is a penalty for using this information. The delay in status causes a two flit bubble between consecutive messages for packet and wormhole routing. This reduces the maximum throughput achievable by both the Duato and Triplex algorithms. The other algorithms including

Table 4.1: Summary of the differences between the various routing algorithms.

Router	Node Latency	Adaptivity	Buffers per Node
Oblivious	3	none	18
Obliv Triplex	4	none	18
Duato	4	min adaptive	26
Min Triplex	4	min adaptive	26
Chaos	4	non-min adaptive	15
Triplex	4	non-min adaptive	26

the packet Triplex algorithm avoid this penalty, since they only require status about the local output buffers.

4.7.2 Simulation Parameters

The standard routing workloads were simulated including random, bit reversal, transpose, and hot spot traffic. See Chapter 3 for details.

For non-minimal routers where derouting is forced, excessive derouting is a difficult problem. This occurs in deflection routers which may deroute every cycle; and to a lesser extent, in chaotic routers which must fill a special queue before deroutes are forced. Triplex prevents a message from derouting continuously and wasting bandwidth by routing minimally when possible, requiring a message to wait at least 160 cycles before becoming eligible for deroutes, and by derouting eligible messages with a probability of .1 when no minimal path is available. For packet routing the values used are 400 and .00001, respectively. These parameters were chosen by experimentation which found nearly equivalent results for a large range of parameter values. Probabilistic derouting is also used to guarantee probabilistic livelock-freedom. For packet routing, the deroute probability selected is extremely small. Thus, very few packets are derouted. This simply reflects the fact that the non-minimal routing permitted by

packet Triplex is not effective at improving throughput or decreasing latency beyond that achievable by taking minimal routes.

4.7.3 Results

The methodology to evaluate multi-class routers described earlier is used to examine the performance of Triplex and packet Triplex in each of its modes: oblivious, minimal fully adaptive, and non-minimal fully adaptive. Single class traffic comparisons are made between Triplex and a single class router using the same mode and flow control, while mixed class traffic experiments compare Triplex performance with the single class oblivious router. Tables 4.2 and 4.3 summarize the single class and mixed class traffic comparisons, respectively.

Table 4.2: Single class traffic comparisons between each Triplex class and its corresponding single class routing algorithm.

Single Class Traffic Comparisons		
Class	Single Class	Multi-class
oblivious	Oblivious	Obliv Triplex
min adaptive	Duato	Min Triplex
non-min adaptive	Chaos[‖]	Non-min Triplex

Results show the Triplex algorithm performs well for the oblivious and minimal adaptive modes and has a wide range of traffic loads and patterns for which it is advantageous compared with single class routing algorithms. Triplex does not perform as well as expected in the non-minimal mode. Representative wormhole and packet results are presented. The remainder of the graphs can be found in Appendix A.

[‖]Chaos is used for packet routing comparisons only.

Table 4.3: Mixed class traffic comparisons between Triplex and the oblivious routing algorithm, given a traffic mix where $x\%$, $0 \leq x \leq 100$, of the traffic requires in-order delivery, while the remaining $(100 - x)\%$ may use adaptive routing.

Mixed Class Traffic Comparisons	
Single Class	Multi-class
100% Oblivious	$x\%$ Oblivious Triplex $(100 - x)\%$ Min or Non-min Triplex

Single Class Traffic Results

Wormhole results are presented first. The first set of experiments consider short messages, while the subsequent set of simulations contains traffic with both short and long messages.

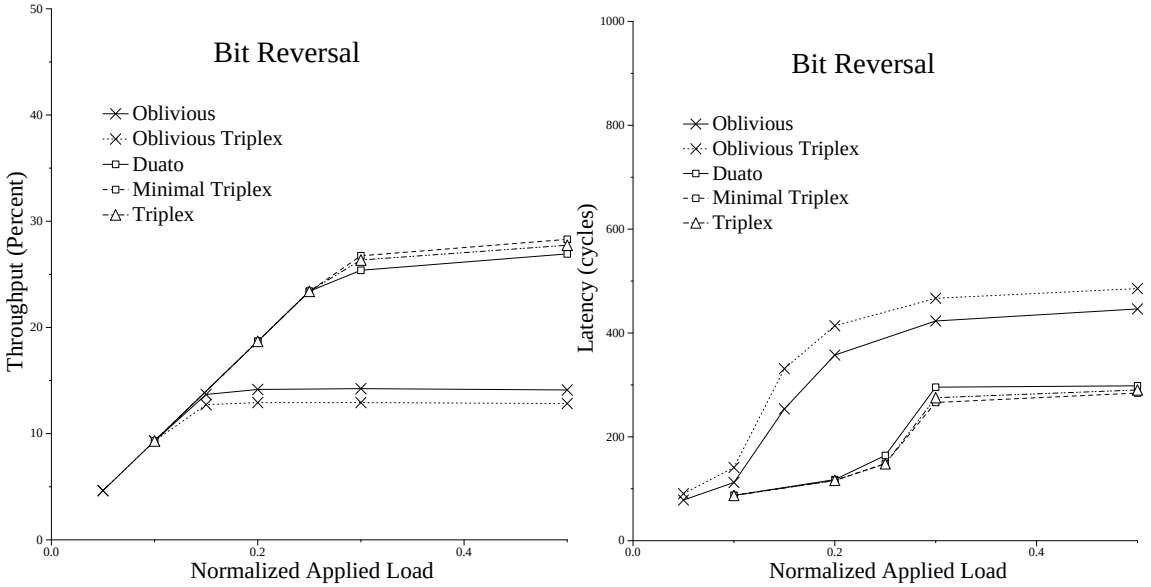

Figure 4.8: Wormhole performance on a 256-node 2D torus with 40-word messages.

Figure 4.8 depicts representative throughput and latency graphs. At very low loads where no congestion is present, all the routers achieve the same throughput.

The expected latency depends on the minimal latency of a message through the network. The oblivious router has the smallest node latency (three versus four cycles for all other routers); and therefore, has the best expected latency for the different traffic patterns.

At higher loads, the throughput of the oblivious Triplex router does not quite match that of the oblivious router. Oblivious Triplex throughput is within 23 percent of the throughput of the oblivious router at the same applied load. The effect on network throughput is not substantial though, since for all of the applied loads, the difference in the normalized throughput achieved is at most 4 percent. This is a small price to pay for the additional flexibility of Triplex. The difference in throughput occurs because the oblivious Triplex incurs an extra cycle of latency at each node due to its higher node latency. Though small, the difference between the two routers is greater for some traffic patterns than one might expect (see Section 4.7.3 for a comparison with packet routing). With wormhole routing a message holds multiple buffers in multiple nodes in the network. Thus, the extra cycle of latency charged to oblivious Triplex forces each message to hold scarce buffer resources across multiple nodes for a longer period of time than with the oblivious router. This increases the amount of message blocking in the network and may reduce the throughput and increase the latency of oblivious Triplex. The amount of performance degradation is dependent on the the congestion and contention caused by the traffic pattern. As expected, oblivious Triplex experiences slightly higher latency than the oblivious router.

The minimal Triplex algorithm matched or exceeded Duato throughput performance in half of the four cases (bit reversal and transpose), while in the other cases (random and hot spot) the Duato achieved a higher throughput, despite its more restrictive routing rules. Minimal Triplex is within 9 percent of Duato, while the difference in normalized throughput is 2 percent or less. Latency was slightly lower for the minimal router that achieved the higher throughput. We conjecture that minimal

Triplex does not always surpass Duato because Triplex allows the restricted buffers to be used in a manner which violates dimension-order routing. This increases the number of turns a message may make from one dimension to another which are a source of potential conflicts.

For the non-minimal wormhole case, there is no qualified algorithm for comparison. Triplex performs adequately, but not as well as the minimal Triplex algorithm. In the cases where derouting was particularly ineffective (random and hot spot), Triplex also experienced higher latencies than the minimal routers. It is likely that the asymmetries in the virtual channel usage makes the non-minimal routing less effective than it should be. This may be caused by both the non-uniformities from the deadlock avoidance scheme and the somewhat restricted non-minimal routes allowed for only a subset of the messages in the network.

When the traffic includes both long and short messages, the results are similar to those with only short messages, though the overall throughput is lower and latency is higher. Figure 4.9 shows representative throughput and latency graphs of the mixed length messages. Even when Triplex does not match or exceed the single class router, the difference is at most 2 percent of the normalized throughput for the minimal adaptive class (an 10 percent degradation with respect to the Duato) and within 3 percent for the oblivious class (within 21 percent with respect to the oblivious router). The routers have the same relative performance except for the non-minimal case, where long messages emphasize the ineffectiveness of non-minimal routing in the Triplex algorithm. The performance difference between oblivious Triplex and the single class oblivious routing algorithm should not be a big problem, since oblivious routing is only used when in-order delivery is needed, e.g. with synchronization. Otherwise adaptive routing is used.

Table 4.4 contains the saturation points for each of the traffic patterns and algorithms. The saturation point is important, since after saturation network performance is unpredictable. Although real systems cannot sustain such loads, it provides

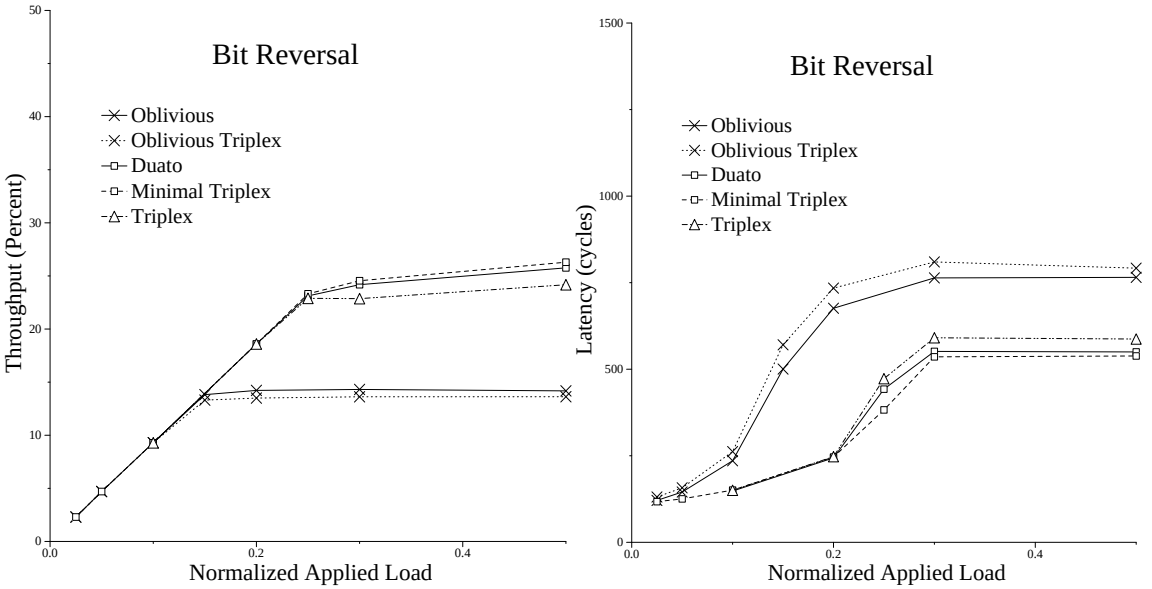

Figure 4.9: Wormhole performance on a 256-node 2D torus with 40-word and 400-word messages in a 10:1 ratio.

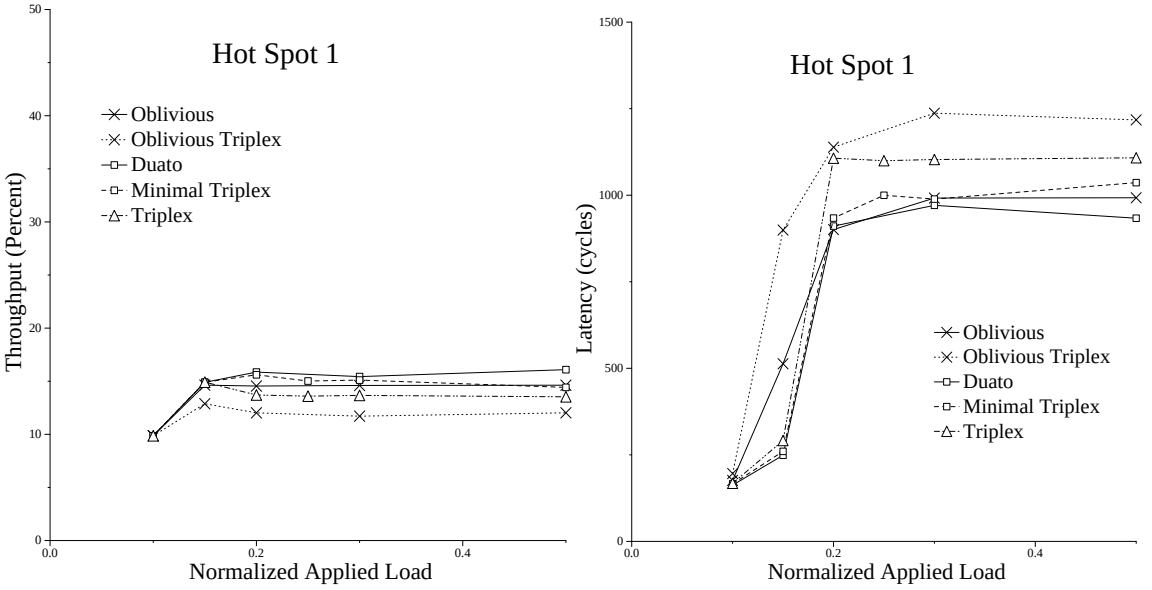

Figure 4.10: Wormhole performance on a 256-node 2D torus with 40-word and 400-word messages in a 10:1 ratio.

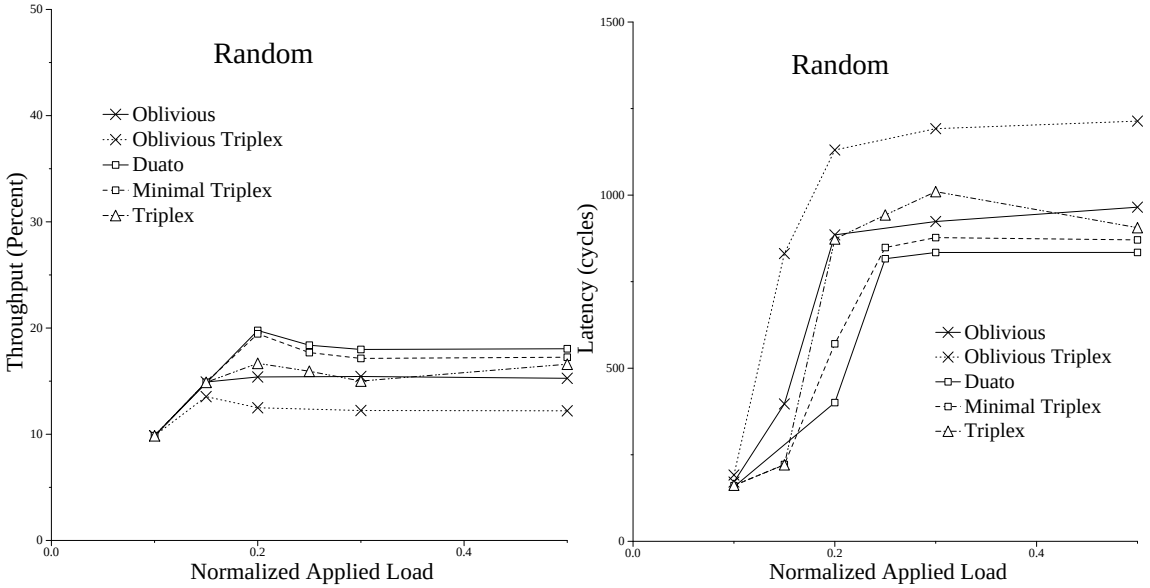

Figure 4.11: Wormhole performance on a 256-node 2D torus with 40-word and 400-word messages in a 10:1 ratio.

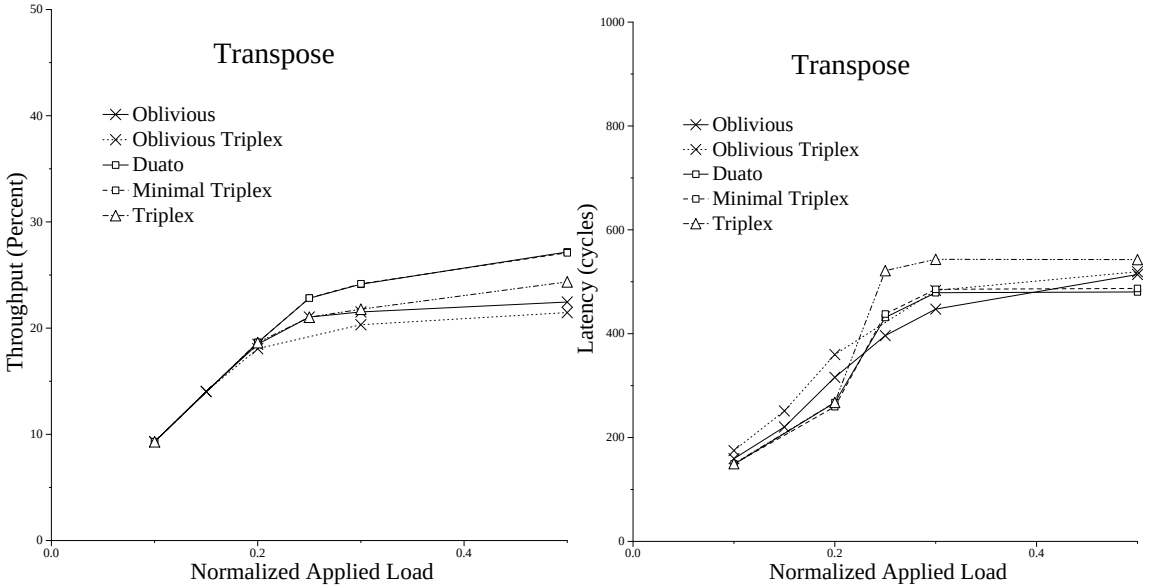

Figure 4.12: Wormhole performance on a 256-node 2D torus with 40-word and 400-word messages in a 10:1 ratio.

information about the system's ability to tolerate large bursts of traffic. The saturation points of Triplex and the corresponding single class routers are nearly identical. Long messages decrease the saturation point of the network slightly because large messages hold more resources simultaneously.

Table 4.4: Minimum normalized applied load at which saturation is detected. Short refers to 40-word messages and mix refers to 40-word and 400-word messages in a 10:1 ratio.

256-node 2D Torus Saturation Points for Wormhole Routing										
Traffic	Oblivious		Obliv Triplex		Duato		Min Triplex		Triplex	
Msg type	short	mix	short	mix	short	mix	short	mix	short	mix
Random	.20	.20	.20	.15	.30	.25	.30	.20	.30	.20
Bit reversal	.15	.15	.15	.15	.30	.25	.30	.30	.30	.25
Transpose	.20	.20	.20	.20	.25	.25	.30	.25	.30	.25
Hot Spot 1	.20	.15	.20	.15	.25	.20	.25	.20	.20	.20

Mixed Class Traffic Performance Estimates and Results

As mentioned previously, single class comparisons do not provide a complete performance evaluation since they ignore the benefits of multi-class routing. Mixed class traffic evaluation helps highlight the benefits of multi-class routing which includes the ability of selective messages to arrive in-order while other messages benefit from the higher throughputs achieved by adaptive routing.

The graphs in Figure 4.13 compare the oblivious algorithm with Triplex using the estimated and experimental results for a range of traffic mixes. The graphs show that Triplex performs close to the estimates. This demonstrates that with this multi-class router, examining the single class traffic can help provide a reasonable prediction of mixed class traffic behavior despite the shared resources of the multi-class router. The graphs also show that when almost all the traffic requires in-order

delivery, the single class oblivious routing algorithm has a slight advantage. But when more of the traffic may be delivered out-of-order, the adaptive class of Triplex helps the Triplex algorithm provide higher throughput and lower latency than the oblivious algorithm. The improvements are the greatest for traffic that benefits significantly from adaptive routing. See Figures A.9–A.12 for additional comparisons between Triplex and oblivious routing for a variety of traffic patterns and applied loads.

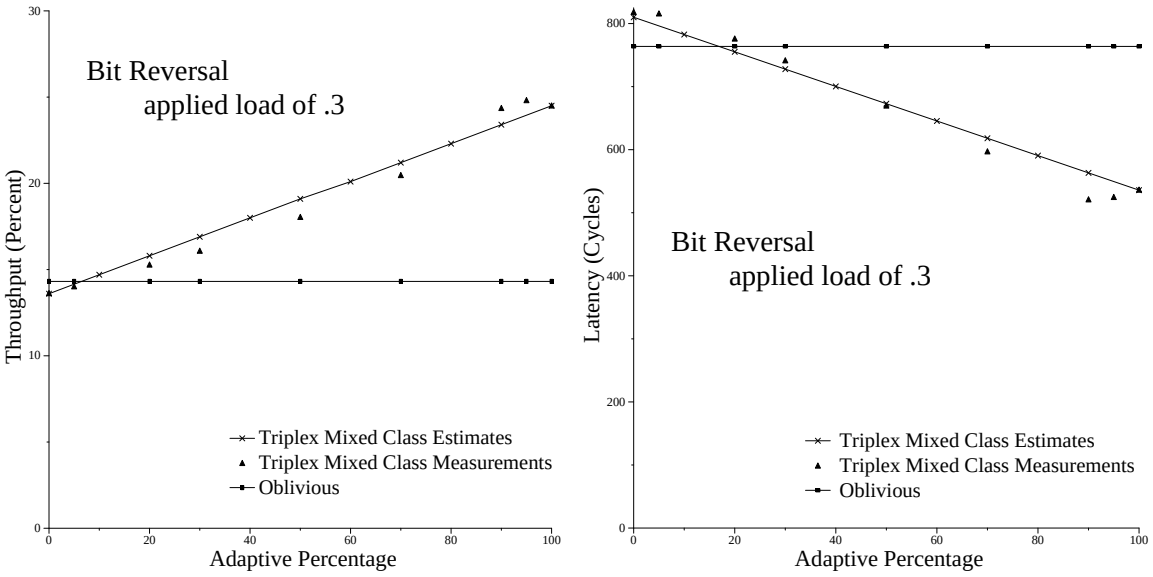

Figure 4.13: Throughput and latency comparisons between oblivious and Triplex algorithms for a range of traffic mixes on a 256-node 2D torus using wormhole routing with 40-word and 400-flit messages in a 10:1 ratio.

The next set of graphs in Figure 4.14 show the advantage of Triplex over the oblivious algorithm at a particular normalized load. The graph on the left represents the difference in normalized throughput between Triplex and the oblivious algorithm, while the graph on the right displays the difference in latency between the oblivious algorithm and Triplex. As observed before, at very small loads there is no performance advantage (actually a penalty). But the graphs show that the performance advantage of the multi-class router increases as the percentage of adaptive traffic in the traffic mix increases. Although the maximum differences may appear small, the multi-class

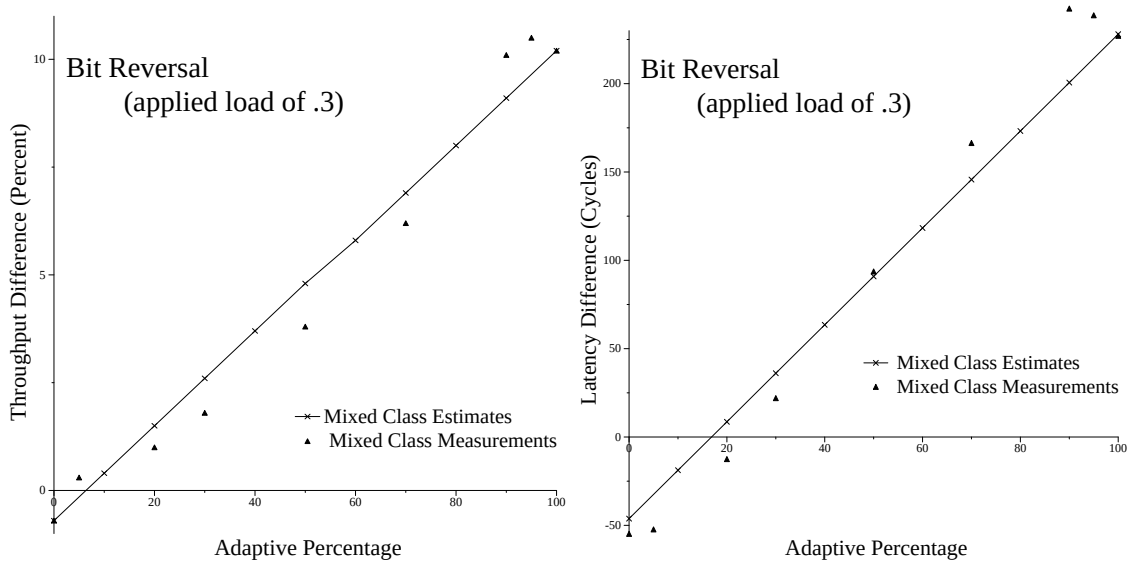

Figure 4.14: The performance advantage of Triplex over the oblivious algorithm on a 256-node 2D torus with 40-word and 400-word messages in a 10:1 ratio.

throughput (latency) improvement is 72 (30) percent with respect to the oblivious algorithm. These results also show that it is possible for a multi-class router to provide superior performance over a single class router, even when the single class performance of each mode of the multi-class router does not match or exceed that of the single class router.

Packet Results

This section presents single class comparisons of the different routers using packet routing. In this case, we can compare packet Triplex to Chaos, a competitive non-minimal packet routing algorithm. See Figures 4.15, and A.2–A.4 for the final comparisons between Triplex and the corresponding single class routers.

As in the wormhole case, at very low loads where no congestion is present, all the routers achieve the same throughput. The oblivious router has the smallest node latency; and thus, has the best expected latency for the different traffic patterns.

At higher loads, the throughput of the oblivious packet Triplex router does not

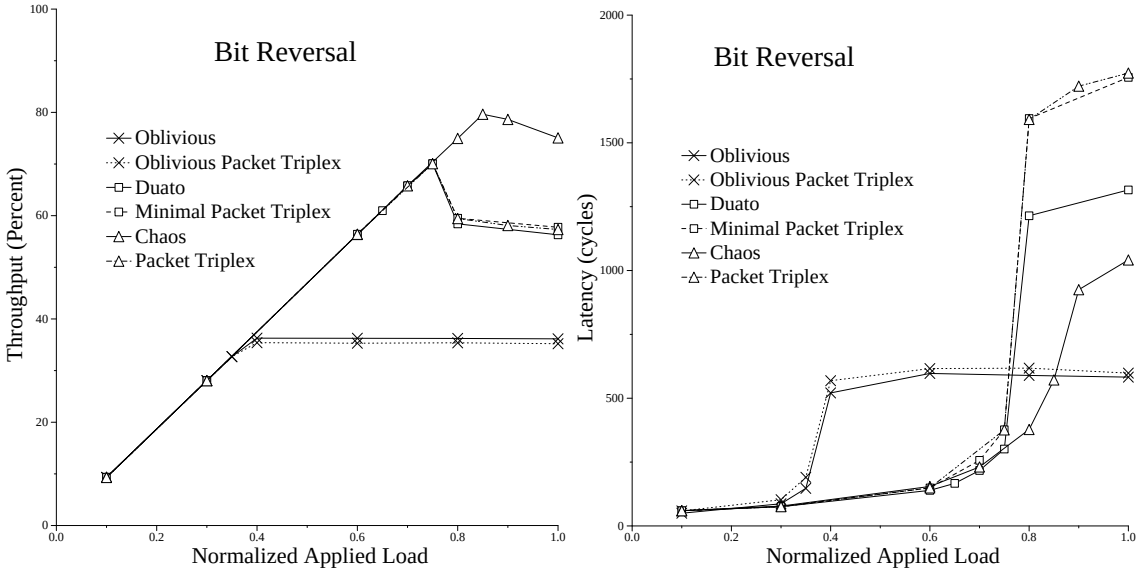

Figure 4.15: Performance on a 256-node 2D torus with 20-word packets.

quite match that of the oblivious router. Triplex throughput is within 5 percent (and usually less) of the throughput of the oblivious router at the same applied load while the difference in the normalized throughput achieved is at most 2 percent. As before, the only difference between the two oblivious classes is the extra cycle of latency that packet Triplex incurs at each node. The throughput difference is smaller with the packet network than with the wormhole network. With packet routing when the packet header is blocked (due to a routing decision, congestion, or conflict), the tail of the message continues to progress towards the buffer containing the message header. This helps minimize the amount of additional blocking caused by the higher latency charged to oblivious packet Triplex messages. In contrast, with wormhole routing the entire message is blocked when the header blocks**. Due to its higher node latency, the oblivious packet Triplex router experiences a slightly higher latency than the oblivious network.

Although the minimal packet Triplex is less restrictive than the Duato algorithm,

**This assumes single word buffers, like those used in the wormhole experiments.

it did not match the Duato performance. For all the traffic patterns, Duato matches or exceeds the maximum throughput of minimal Triplex. Nevertheless, for two traffic patterns bit reversal and perfect shuffle, the throughput of minimal Triplex did not degrade nearly as much as Duato. Latency for minimal Triplex is also worse. The only exception was for the perfect shuffle traffic pattern, where the latency of Duato is over twice that of minimal Triplex.

Performance of minimal packet Triplex is improved substantially by omitting (the minimal routes of) Rule 4t. In this case, the two algorithms have the same routing rules but differ in their empty buffer definition. The Duato algorithm has a stricter criterion resulting in bubbles between consecutive messages in the network. Consequently, the throughput of minimal Triplex equals or exceeds (bit reversal and perfect shuffle) that of Duato, although the additional occupied buffers of Triplex results in higher after saturation latency. We conjecture that Rule 4t, which allows messages to use the restricted buffers according to rules which violate dimension-order, permits more turns from one dimension to another in the restricted output buffers. This causes conflicts which impede the flow of messages in a congested network. Maximizing adaptivity seems like a good strategy to increase throughput; however, the results demonstrate that this is not always the case. From here on, we assume that the packet Triplex implementation has omitted the minimal routes of Rule 4t.

For the non-minimal case, packet Triplex performs well, equivalent to the minimal packet Triplex, but has higher latencies than and lacks the sustained throughput achieved by the Chaos router at high loads. We believe that this performance difference arises from the non-uniformities introduced into the network by the routing restrictions that prevent deadlock for the Triplex router. This includes both the underlying oblivious network and the restricted nature of the non-minimal routes. Some messages are not allowed to deroute, while others are but eventually lose this ability. This loss may be beneficial when the message is a hop or two away from its destination [Kon92], but not if it is far from its destination. The Chaos router has none

of these non-uniformities, since it relies on the packet-exchange protocol for deadlock prevention [NS89]. The protocol has a simple requirement: if node a sends a message to node b, node a must also accept a message from node b.

Next, packet Triplex was tested with Rule 4t completely omitted. Deroutes were still allowed by Rule 3. For all the cases tested, the throughput of the modified packet Triplex is equivalent to that of the packet Triplex algorithm. Likewise, latency was equivalent or lower than the Triplex algorithm. As before, we believe that allowing some messages to violate dimension-order rules, in buffers used primarily for dimension-order routing, impedes message flow in congested networks. In the following, we assume that the Triplex implementation has omitted Rule 4t entirely.

Finally, since the Chaos router does not allow messages in an injection buffer to deroute, this feature was disabled in packet Triplex. This reduced the standard deviation of the throughput and latency figures; but otherwise, was insignificant in improving the performance of Triplex.

4.8 Related Multi-class Work

A large number of single class k-ary n-cube routing algorithms have been developed [DS87, KP95, LH91, Dua95, GPBS94, GN94, CK92, LC94, SJ96, KS94]. See Chapter 2 for more details. Each balances algorithm flexibility against resource requirements and router complexity.

Only a few routers provide in-order message delivery. The Dally Seitz oblivious algorithm [DS87] is one and requires two virtual channels per channel. Messages follow a dimension-order path by correcting dimension from the lowest to highest. Another is the Compressionless router [KLC94]. Compressionless routing, which uses an abort and retry technique, is designed for small diameter networks with minimal buffering since it requires a message to own the entire path between its source and destination before sending is completed. Thus, if messages are sent in-order at the source, they

arrive in-order at their destination. This method suffers from high contention in large congested networks, and wastes bandwidth on padding required for short messages when the network diameter is large; but it achieves adaptivity and fault-tolerance without the use of virtual channels, and may be sufficient for small networks.

Even fewer routers offer several classes of routing. The planar-adaptive router [CK92] is a partially adaptive wormhole routing algorithm that sacrifices algorithm adaptivity to reduce the crossbar complexity required by fully adaptive algorithms. The algorithm restricts routing to two dimensions (a plane) at a time. The highest dimension in one plane is the lowest dimension in the next plane. By tagging messages and sending them on a dimension-order path, messages can also be delivered in-order.

The Cray T3E router [ST96] is a fully adaptive (in theory, not implementation) packet routing algorithm. The router uses a modified version of Duato's technique and provides four virtual channels for oblivious routes and a fifth virtual channel for adaptivity. Using packet-switching (either store-and-forward or virtual cut-through [KK79] as in the T3E) simplifies the routing and flow control requirements of using Duato's technique. For the T3E, the oblivious route is direction-order instead of dimension-order. Direction-order [YDG] routes the positive directions in dimension-order and then the negative directions in dimension-order. A message can be restricted from minimal adaptive routes to oblivious routes to avoid faults in the network. Oblivious routes can also be used to provide in-order delivery, as in the planar-adaptive router.

The Triplex router [FS97b, FS97a] described in this chapter is a non-minimal fully adaptive wormhole algorithm for the torus, that supports oblivious, minimal fully adaptive, and non-minimal fully adaptive routing. It uses three virtual channels per channel. Unlike the T3E, Triplex supports wormhole flow-control between all of the buffers. Triplex uses a deadlock-avoidance technique of Schweibert and Jayasimha [SJ96] that allows cyclic channel dependencies. This yields extra adaptivity

over traditional adaptive algorithms since there is no set of virtual channels that are restricted to messages taking acyclic routes. Unlike the traditional solution which uses two virtual channels for oblivious routing and a third for adaptive routing, Triplex makes no such distinction. Adaptive routes are allowed in all three virtual channels.

4.9 Summary

In this chapter we define and describe the idea of multi-class routers which allow applications, or different messages within applications, to use different classes of routing. Multi-class routers are useful because not all applications have the same network requirements. Applications can achieve greater performance if they are allowed to use a class of routing which is most advantageous. Typically systems provide a single class one-size-fits-all routing algorithm. This generally provides routing which is overly restrictive for some messages. In addition multi-class routers are an alternative method to provide adaptive routing which is attractive because it avoids the complexity of designing a network interface to reorder messages.

Further, this chapter describes how to evaluate multi-class routers. Comparing the multi-class router against the best single class algorithm for each of its modes is not sufficient. Despite the costs of flexibility, multi-class routers can be advantageous for a large combination of traffic classes because all the traffic is not forced to use a single class of routing.

In addition to promoting multi-class routing, we have mentioned a few routers that provide multi-class routing and have presented Triplex, a novel multi-class routing algorithm. Triplex is the first triple class router. It can select from oblivious, minimal fully adaptive, or non-minimal classes at boot-time, application invocation, or individually by message. Triplex is deadlock-free and livelock-free, and supports both packet and wormhole routing on k-ary n-cubes. Triplex provides additional adaptivity over traditional fully adaptive algorithms since it uses a more flexible deadlock

avoidance technique. Triplex is also the first non-minimal fully adaptive wormhole deadlock avoidance algorithm for tori. Although the algorithm is non-minimal, no message is forced to take a deroute. The algorithm is also deterministically or probabilistically livelock-free, depending upon the implementation chosen.

Simulations show that the performance penalty for providing the flexibility of oblivious and minimal fully adaptive classes is reasonable and that multi-class routing is advantageous for a wide variety of loads and traffic patterns. The performance of the non-minimal option, however, is disappointing.

Chapter 5

INPUT VERSUS OUTPUT DRIVEN ROUTING

In this chapter, we experimentally compare two methods of implementing a particular router, as an input driven or as an output driven router. Although the two choices are conceptually similar, they result in noticeable performance differences when compared with each other. Three algorithms, the Dally-Seitz oblivious router, the Duato router, and the minimal Triplex router, are simulated on a 256-node two-dimensional mesh and torus. Each algorithm is configured as an input and as an output driven router. The output driven versions of the algorithms are almost always equivalent or superior to the input driven versions when the networks are congested. This is the critical area of network performance, since communication is often bursty and does not always present easy, light loads to the network.

5.1 Input versus Output Driven

Routers that make decisions based on local information can usually be classified as either input driven or output driven. The input driven algorithms make routing decisions from an occupied input buffer while the output driven algorithms select routes from an empty output buffer. Typically the input driven algorithms service the input buffers in round-robin order, and the output driven algorithms service the output buffers in round-robin order. After routing, the pointer to the current buffer is advanced to the next dimension where an *interesting* buffer resides. For input driven algorithms an interesting input buffer is one that contains a ready message that needs an available output buffer. For output driven algorithms an interesting output buffer

is one that is empty and is wanted by some ready input message.

An input driven router operates as follows. First it computes for the current input message which output buffers the message needs. Then it selects one of the available output buffers and routes the message to that output buffer. Many of the routing algorithms in the literature fall into this category. See Figure 5.1 for an example of an input driven router.

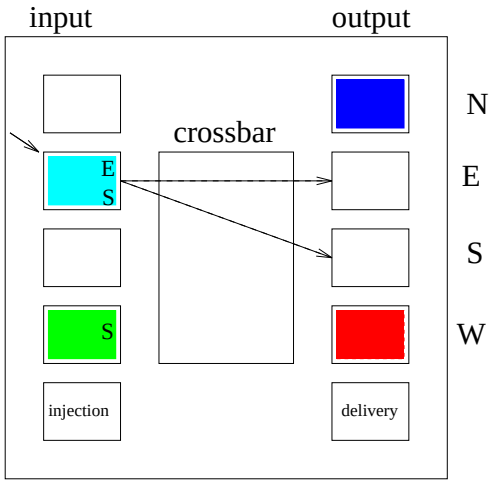

Figure 5.1: An example of an input driven router. The short arrow marks the current input buffer. The long arrows are the choices of empty output buffers for the message in the current input buffer. Messages can travel north (N), east (E), south (S), or west (W). Buffers that are filled in are not available, while the unfilled buffers are available.

An output driven router considers the current empty output buffer. The router tries to find a message in an input buffer that needs to be routed to the current output buffer. If messages are found, it selects one to be routed to the current output buffer. The Chaos router is a good example of an output driven router [KS94]. See Figure 5.2 for an example of an output driven router.

Many algorithms can be implemented as either input or output driven algorithms. For these algorithms, it would be advantageous to implement the router with the best performing routing mechanism. This chapter shows that for the algorithms examined, the output driven algorithm generally performs as well as, or better than the input

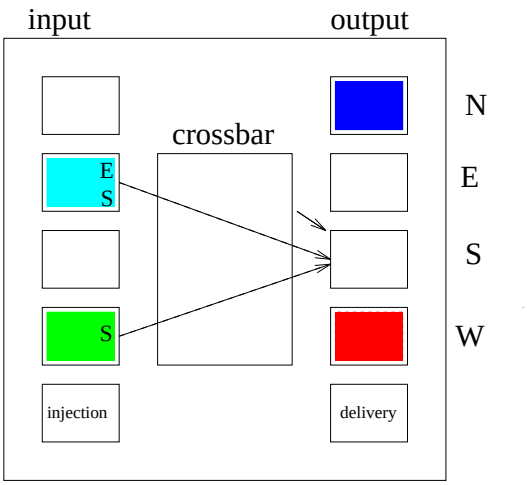

Figure 5.2: An example of an output driven router. The short arrow marks the current output buffer. The long arrows are from messages that are competing for the current empty output buffer. Messages can travel north (N), east (E), south (S), or west (W). Buffers that are filled in are not available, while the unfilled buffers are available.

driven algorithm. A few algorithms cannot easily be transformed into output driven algorithms, for example algorithms that consider probabilities over the possible output choices.

5.2 Methodology

Performance of input versus output driven routing algorithms is compared by experimentation. Three algorithms are simulated as both input and as output driven routers: the Dally-Seitz oblivious router [DS87], the Duato router [GPBS94, Dua93], and the minimal Triplex router* [FS97b] using a flit-level simulator of 256-node 2D torus and mesh networks.

The first algorithm, the Dally-Seitz oblivious router, provides no adaptivity and routes messages by correcting the dimensions from lowest to highest. Unless specified, the oblivious router uses two virtual lanes [Dal92] resulting in four virtual channels

*An early implementation of the Triplex router is used. Since then, performance has improved, though the differences between the input and output driven routers should remain unchanged.

per channel for the torus and two virtual channels for the mesh. When more than one lane is available, the one with the most space is chosen. The Duato algorithm is a minimal fully adaptive algorithm which uses three (two) virtual channels per channel for the torus (mesh). The minimal Triplex router is a minimal fully adaptive version of the Triplex router. It also uses three (two) virtual channels per channel for the torus (mesh). All the algorithms have routing restrictions that prevent deadlock.

The following describes the high level design and operation of the routers. Each router has an injection buffer, a delivery buffer, and an input and an output buffer for each virtual channel in each dimension in each direction. This yields 26 (18) buffers per node for the Duato and minimal Triplex routers on the torus (mesh), since each uses an injection buffer, a delivery buffer, and three (two) virtual channels per channel per direction. The oblivious router is configured with four (two) virtual channels per channel per direction for the torus (mesh) or 34 (18) buffers per node. Table 5.1 summarizes the differences between the routers. See Chapter 3 for other details about the methodology.

Table 5.1: Summary of the differences between the various routing algorithms. The number in parentheses refers to the number of buffers required when no virtual channels are used.

Router	Node Latency	Adaptivity	Buffers per Node	
			Torus	Mesh (no vc)
Oblivious	3	none	34	18 (10)
Duato	4	min adaptive	26	18
Min Triplex	4	min adaptive	26	18

Experiments were run to determine the first normalized applied load at which the network saturates for each of the routers and traffic patterns. Although saturation is tested in increments of .05, a difference in saturation points of .05 between an input and output driven router for a particular traffic pattern is significant, unless otherwise

noted. If the data for determining saturation point has overlapping 95% confidence intervals, it was marked as statistically insignificant.

5.3 Results

The results are presented in two parts. The first set of experiments compare output driven routers to input driven routers that choose deterministically the first available output buffer from the set of needed output buffers. Since any deterministic order suffices, the routers search in a simple dimension-order manner. In the second set of experiments the input driven router selects an available output buffer at random from the set of needed output buffers. For this set-up, the input driven router is very similar to the output driven router which chooses at random among interesting messages from the input buffers. Table 5.2 summarizes the comparisons in the two sets of experiments. In both sets of experiments, the output driven routers perform as well as or better than the input driven routers for all but a few cases. As expected, the results are more dramatic for the first case.

Table 5.2: Summary of the input driven versus output driven router experiments. The comparisons in the two sets of experiments are made for three different routing algorithms using seven different traffic patterns for both the mesh and torus topologies.

Set	Input Driven	Output Driven
1	fixed order output buffer selection	random input buffer selection
2	random output buffer selection	random input buffer selection

5.3.1 Fixed Order Output Buffer Selection

The first set of experiments considers input driven routers with a fixed order output buffer selection policy. Table 5.3 specifies the saturation point of each 256-node torus network for a variety of traffic patterns. For all three algorithms and all traffic patterns

except one, the complement, the saturation point for the output driven router is at least as great as the saturation point of the corresponding input driven router. The advantage ranges from zero to fifty-four percent of the input driven saturation point.

Table 5.3: Minimum normalized applied load within .05 at which saturation is detected.

16x16 Torus Saturation, input driven, order is fixed						
Traffic	Oblivious		Duato		Min Triplex	
	input	output	input	output	input	output
Random	0.80	0.80	0.85	0.95	0.80	0.85
Bit reversal	0.50	0.50	0.70	0.80	0.65	0.75
Complement	0.50	0.50	0.45	0.40	0.40	0.40
Perfect shuffle	0.50	0.50	0.50	0.50	0.40	0.45
Transpose	0.55	0.55	0.55	0.55	0.55	0.55
Hot Spot 1	0.65	0.65	0.70	0.90	0.65	0.85
Hot Spot 2	0.55	0.55	0.55	0.85	0.60	0.80

We conjecture that the output driven algorithms are better at balancing the load among the channels because the empty output buffers are filled by dimension in round-robin order which naturally tries to keep all the physical channels busy. More specifically, when a message has been routed to an empty output buffer in a dimension, the router will consider the next dimension that has an empty output buffer. The input driven algorithm, however, routes a message to the first available output buffer it finds. This buffer may share the same physical channel as a recently routed message. This cannot happen in an output driven router unless all the other output buffers in the other dimensions are full, or all the messages in the input buffers only need dimensions that are being used by previously routed messages. The advantage of the output driven router is that it balances the use of the channels by selecting interesting

messages from any of the input buffers. In contrast the input driven algorithms balance messages leaving the input buffers, but must select an output buffer based only on its availability.

Table 5.4 presents the saturation loads for the 256-node mesh and shows that output driven routers are superior to input driven routers for the mesh also. The advantage ranges from zero to thirteen percent. Again, the only exception is the complement traffic for the Duato and minimal Triplex adaptive routers.

Table 5.4: Minimum normalized applied load within .05 at which saturation is detected.

16x16 Mesh Saturation, input driven, order is fixed								
Traffic	Obliv (no vc)		Oblivious		Duato		Min Triplex	
	input	output	input	output	input	output	input	output
Random	0.90	0.90	0.95	0.95	0.95	0.95	0.90	0.90
Bit reversal	0.50	0.50	0.55	0.55	0.80	0.80	0.70	0.75
Complement	0.45	0.50	0.50	0.50	0.45	0.35	0.45	0.35
Perfect shuffle	0.75	0.80	0.90	0.90	0.90	0.95	0.80	0.90
Transpose	0.50	0.50	0.55	0.55	0.85	0.85	0.85	0.85
Hot Spot 1	0.75	0.75	0.80	0.80	0.80	0.85	0.75	0.85
Hot Spot 2	0.70	0.70	0.75	0.75	0.75	0.85	0.75	0.85

We expect the benefit of output driven routers to be less pronounced or non-existent in oblivious routers than in adaptive ones, since the two oblivious routers make the same routing decisions. The oblivious routers may, however, move a message from an input to an output buffer at a different time or select a different lane to route a message to, though the latter is unlikely to effect performance since lanes share the same underlying physical channel. The data supports this hypothesis since there is almost no difference between the saturation points of the input and the output driven oblivious routers on the torus and mesh.

Routing the complement permutation using the Duato algorithm is the one case where the output driven saturation point did not equal or exceed the input driven saturation load for either topology. The complement is unusual since dimension order routing helps prevent conflicts in this traffic pattern, i.e. when two messages compete for the same output buffer.

To see this, consider the following idealized scenario. Each node simultaneously injects a single message destined for the bit complement of its source id. When routing is by a synchronous, oblivious dimension-order algorithm, there are no conflicts for channels. With dimension-order routing, every message m with source (i, j) is routed in row i in exactly the same way as message m' with source (k, j) is routed in row k. All messages move within the rows in lock step. Conflicts can only occur when a message turns from its source row to its destination column. Messages from column j are the only messages that need to travel in destination column $\bar{j}$. Messages m and m' arrive at their destination column $\bar{j}$ at the same time, and hence do not impede each other from progressing to their respective destinations. Arbitrary injections are used in our routing experiments, however, perturbing this orderly flow of messages. Adaptive routing further destroys this property by letting messages take any minimal path.

For the Duato algorithm the input driven router has a slight advantage in exploiting this phenomenon since it prefers the lowest dimension output buffer needed, while the output driven algorithm selects a random message that needs the current output buffer. Nevertheless, the second set of experiments shows this advantage disappears when the input driven algorithm selects at random from the needed output buffers.

For the mesh, the oblivious algorithm was also simulated with no virtual channels (vc), i.e. with no extra lanes. In this case, a message waiting in an input buffer needs exactly one output buffer. Therefore, the input and output driven algorithms do not make different routing decisions or buffer choices. Rather, the difference in performance is due to the ordering of messages and the router's ability to keep the

channels busy. As hypothesized, the difference in saturation points between input and output driven routers is very modest, non-existent in five traffic patterns and within a normalized load of .05 for the two remaining ones. For the two lane oblivious routers, there is no difference in the saturation points between the input and output driven routers.

Figures B.1–B.7 show the expected throughput and latency versus the normalized applied load for each of the traffic patterns on the 256-node torus and mesh.

At low loads the input and output driven routers are indistinguishable. Nevertheless, the output driven router achieves an equivalent or higher peak throughput than the corresponding input driven router for all but one of the traffic patterns and routers simulated. The peak throughput of the output driven router on the torus (mesh) is up to 36 (13) percent better than that of the input driven router with fixed order selection. See Tables B.1–B.6 for details. As with the saturation data, the complement traffic is the only exception. For the minimal Triplex router on the mesh and torus, the peak throughput of the output driven router under the complement traffic does not quite reach that of the input driven router. The same is true for Duato on the mesh.

After saturation, the output driven router almost always maintains a higher throughput than the input driven router, though there are three exceptions. The output driven router degrades slightly more than the input driven router for a few of the applied loads for the perfect shuffle and second hot spot case with the oblivious router on the torus and for the second hot spot traffic with Duato on the mesh.

The latency curves have three phases. Initially the input and output schemes have equivalent latencies. Any router and routing decision will do when the router is lightly loaded. When the applied load is in the neighborhood of saturation and the network is congested, the output driven routers almost always exhibit an equivalent or lower latency (and latency variance) than the input driven router. This is apparent by observing that the output driven routers experience a steep increase in latency at

the same or higher load than the input driven routers. We believe this is because the output driven router is doing a better job at keeping the physical channels utilized. There are two exceptions. The adaptive output driven routers show an increase in latency at a slightly lower load than the input driven router for the complement on the torus and mesh and for random traffic on the mesh.

After saturation the network cannot keep up with the message arrivals; and again, any kind of routing will do. Furthermore, the latency differences are less predictable, though they tend to converge for many of the traffic patterns. In two instances, bit reversal and transpose on the torus, the oblivious input driven router shows a decrease in its after saturation latency. In these cases, many fewer messages with distant destinations are injected into the network, thereby lowering the expected latency of a message. This appears to be caused by the unfair access each node has to a congested network. A similar phenomena occurs with bit reversal traffic at a load of .8 using input driven Triplex routing.

5.3.2 Random Output Buffer Selection

To validate our conjecture that the advantage of output driven routers is not entirely from the non-deterministic search for available output buffers, the following changes were made to make the input driven routers as similar as possible to the output driven routers. Each input driven router was modified so that it selects a needed output buffer at random, instead of choosing the lowest dimension output buffer available. The output driven routers were unchanged and choose messages from the input buffers at random. Experiments showed the modified input driven algorithms to have better channel utilization resulting in improved performance[†].

Tables 5.5 and 5.6 compare the saturation points of the input and the output driven

[†]It is possible that round-robin selection could approximate this improvement without the use of randomization, since deterministically spreading the outgoing messages among the various channels would still help channel utilization.

routers for the torus and mesh. The change from fixed order to random output buffer selection does not change the buffer selection of the oblivious routers, and hence the results for input driven routing with random selection are the same as those with fixed order buffer selection.

Table 5.5: Minimum normalized applied load within .05 at which saturation is detected.

16x16 Torus Saturation, input driven, order is random				
Traffic	Duato		Min Triplex	
	input	output	input	output
Random	0.95	0.95	0.85	0.85
Bit reversal	0.80	0.80	0.70	0.75
Complement	0.40	0.40	0.35	0.40
Perfect shuffle	0.50	0.50	0.45	0.45
Transpose	0.55	0.55	0.55	0.55
Hot Spot 1	0.85	0.90	0.80	0.85
Hot Spot 2	0.75	0.85	0.75	0.80

For the adaptive routers the difference between the saturation points of the input and output driven algorithms on the mesh and torus has nearly been eliminated for almost all the traffic patterns. The output driven advantage ranges from 0 to 14 percent of the input driven saturation point for the torus and from 0 to 13 percent for the mesh. The input driven routers with random output buffer selection are superior to those with fixed order output selection, since removing the bias of fixed order selection improves the utilization of the physical channels of a node. The only exception is the complement. The complement saturates at a lower load than previously for both adaptive routers on the mesh and torus. This is because the fixed order input selection prefers the dimension-order route if available; and as mentioned earlier, the complement prefers dimension-order routing.

For the mesh hot spot traffic, the Duato input driven router with random selection has a higher saturation point than the output driven router, even though the output driven router nearly achieves the same throughput. The input driven router is able to prevent congestion around the hot spots from spreading as quickly into the whole network compared to the output driven router. The reasons for this are unclear, but most likely related to the lack of wrap edges in the mesh since the torus does not exhibit this behavior.

Table 5.6: Minimum normalized applied load within .05 at which saturation is detected. If an entry is marked with an asterisk, the difference in saturation points between the input driven and output driven router is not significant.

16x16 Mesh Saturation, input driven, order is random				
Traffic	Duato		Min Triplex	
	input	output	input	output
Random	0.95	0.95	0.90	0.90
Bit reversal	0.80	0.80	0.75	0.75
Complement	0.35	0.35	0.35	0.35
Perfect shuffle	0.95	0.95	0.90	0.90
Transpose	0.80	0.85	0.75	0.85
Hot Spot 1	0.90*	0.85*	0.85	0.85
Hot Spot 2	0.90	0.85	0.85	0.85

Figures B.9–B.15 compare the expected throughput and latency of the input and output driven algorithms on the torus and mesh. The oblivious comparisons are identical to the previous set of experiments.

The adaptive input driven routers improve their maximum achieved throughput, and in some cases now match the throughput of the corresponding output driven routers. This is observed with bit reversal and random traffic with Duato routing and

random traffic using Triplex routing on the torus. For the mesh, random selection closed the gap between input and output driven performance. This occurs with Triplex for bit reversal, random, perfect shuffle, and both hot spot cases, as well as for the perfect shuffle traffic pattern with the Duato algorithm. In addition, random selection sometimes results in less throughput degradation for the adaptive algorithms on the mesh, as with the perfect shuffle traffic pattern.

The output driven router achieves up to an 11 (6) percent greater peak throughput than the input driven router on the torus (mesh). The two exceptions, as with the saturation points, are the hot spot cases where the Duato input driven algorithm achieves a slightly higher peak throughput than the output driven algorithm. See Tables B.1–B.6 for details.

There are also latency improvements for the adaptive routers with many of the traffic patterns. In these cases, the steep increase in latency occurs at a higher load and latency after saturation is also smaller. In a few cases however, throughput degradation causes an increase in latency. See Figures B.17–B.23 in Appendix B for direct comparisons between the two input driven schemes for both the mesh and torus.

5.4 Summary

We have experimentally compared the performance of input and output driven algorithms on the mesh and torus. Although the two are conceptually similar, for almost all the cases examined, the performance of the output driven algorithms is equivalent to or superior to that of the input driven algorithms. We believe this is because the output driven algorithms implicitly capture some information about the status of all the output channels of the router, and thus use the channels more effectively. In contrast, the input driven algorithms have a more limited view of the system. They only examine the availability of an output buffer needed by an input

message and are less efficient at using the channels. The difference is diminished when randomization is added to the output buffer selection of the input driven algorithm, since this prevents the input driven algorithms from favoring a particular channel. Although the findings presented only apply to the routers considered, we believe that the results can be generalized to routers where the designer is indifferent to which approach to use.

An open problem is whether the performance of the input driven router can be increased to match that of the output driven router by keeping extra state. This information might include which output buffers have most recently received messages from the input buffers. This would provide the input driven algorithm with more complete information about the status of the channels, allow it to make routing choices that distribute the messages among the channels, and possibly increase the utilization of its channels.

Chapter 6

PACKET ROUTING PERFORMANCE

Chapter 4 introduced a new wormhole algorithm and compared it with other competitive wormhole algorithms. Wormhole algorithms gained popularity when architects combined the processor and router on the same chip, as in the MOSAIC [Sei92]. Since space was limited, the small flit buffers of wormhole routing were preferable over typical packet buffers. Nevertheless, wormhole algorithms continue to be popular despite the current trend of building routers separately from the processor. Now that routers are distinct, there is more than enough space to satisfy the requirements of competitive packet routing algorithms. Therefore, in this chapter we consider the often overlooked choice of packet routing.

The main attraction of packet routing is that it provides substantially higher throughput than wormhole routing. Packet routing also has the advantage that all wormhole algorithms can be implemented as packet routing algorithms. This means that a competitive wormhole algorithm, such as the oblivious router, can also be implemented as a packet algorithm. The simulations demonstrate that some of the most competitive packet routing algorithms are ones that are known as wormhole algorithms.

As expected, there is a cost to packet routing. First, virtual cut-through routing needs to be used to avoid the latency of store-and-forward flow control typically used with packet routing. This requires larger, more complex buffers than the single flit wormhole buffers, though they can be implemented effectively as demonstrated by the Chaos chip [Bol93b]. Long messages are a more serious problem, since they must be segmented into packets for network transmission and reassembled before being

delivered to the receiving node. If the packets arrive in the same order in which they were sent, this is trivial, provided there are no lost packets. On the other hand if packets may arrive out-of-order, mechanisms must be provided to reorder the packets into the original message.

Oblivious routers always deliver packets in order because there is a single path between a source and destination, so packets do not pass each other on the same route*. For adaptive routers, this is not true since packets may take different paths and arrive out-of-order. Re-ordering can be done by software at the receiving node, but this adds substantial latency to the communication time. A simple hardware solution to this problem has been proposed by McKenzie in which packets include information on their length and their destination in memory [MBES94]. The Cranium interface places packets into their proper place in memory and signals the processor as soon as the entire message has arrived. The Hamlyn interface is more flexible and complex than Cranium and also provides message reordering [BJM+96].

Though some of the most competitive packet routing algorithms are ones that are derived from wormhole algorithms, there is one notable exception. Chaotic routers are randomizing, non-minimal adaptive packet routers which combine the flexibility found in adaptive routing with a design simple enough to be competitive with the most streamlined oblivious routers. In this chapter, we compare oblivious, minimal adaptive, and chaotic routing by exploring the performance of comparable implementations through simulation. The results indicate that chaotic routers provide very effective and efficient high-performance packet routing.

6.1 Simulations

In this section, we present the results of simulating oblivious, minimal fully adaptive, and chaotic routing on several two dimensional 256-node networks under various

*If non-blocking buffering, such as virtual lanes, is added to an oblivious router, packets may pass each other and delivery could be out-of-order.

synthetic traffic workloads. In particular, the Dally-Seitz oblivious, Duato, and the Chaos algorithms were simulated. The oblivious algorithm was configured with two virtual lanes per virtual channel to minimize the disparity in buffer usage among the routers. Table 6.1 summarizes the design differences between the routers. Additional details can be found in Chapter 3. The oblivious and Chaos algorithms are among the few, if not the only, practical oblivious and non-minimal packet algorithms, respectively. The Duato algorithm was chosen over other minimal algorithms, because it is minimal fully adaptive, it uses a very small number of virtual channels for wrapped or unwrapped k-ary n-cubes, and it does not suffer severe performance degradation from non-uniformities in the network, like other minimal adaptive algorithms. The Duato algorithm is similar to the *modified* packet Triplex algorithm described earlier in Chapter 4. The only difference is the Duato algorithm introduces a slightly larger bubble between consecutive messages on a channel. The results are comparable though. Additional comparisons can be found elsewhere [BFS94] and include hypercube networks [FS93] and deflection routing [Bol93b].

Table 6.1: Summary of the design differences between the packet routing algorithms simulated.

Router	Node Latency	Adaptivity	Buffers per Node
Oblivious	3	none	34
Duato	4	min adaptive	26
Chaos	4	non-min adaptive	15

6.1.1 Saturation

The first set of results simply identifies the saturation points for the different traffic patterns. Table 6.2 reports saturation points for mesh and torus networks with various traffic patterns. The load where the system saturates is an important measure since after saturation it is not possible to predict the delivery time of messages.

Table 6.2: Minimum load at which saturation is detected for 256-node 2D networks.

	Mesh			Torus		
Traffic	Oblivious	Duato	Chaos	Oblivious	Duato	Chaos
Random	0.95	0.95	0.85	0.80	0.95	1.00
Bit reversal	0.55	0.80	0.80	0.50	0.80	0.90
Complement	0.50	0.35	0.35	0.50	0.40	0.35
Transpose	0.55	0.85	0.70	0.55	0.55	0.55
Perfect shuffle	0.90	0.95	0.85	0.50	0.50	0.45
Hot Spot 1	0.80	0.85	0.80	0.65	0.90	0.90
Hot Spot 2	0.75	0.85	0.80	0.55	0.80	0.95

The traffic patterns exhibit considerable diversity in the throughput they are able to sustain. With the mesh network, the oblivious router achieves equivalent or higher saturation levels for random and complement traffic. Bolding and Pertel have compared oblivious routing with minimal adaptive routing on mesh networks and found the oblivious router to perform better with random traffic [Bol93b, Per92]. The main reason is that when dimension-order routing is used, packets follow 'L'-shaped paths and are not as likely to use the paths near the center as intensively as in minimal adaptive routing (though the center paths will still be more congested than the edge paths). Thus, the hot spot in the center is not as "hot" when using oblivious routing as when using minimal adaptive routing, so performance is potentially better.

Similar results are observed for chaotic adaptive routing. The center becomes a very "hot" hot spot, resulting in severe congestion and excessive derouting. Because the mesh hot spot creates so many difficulties, and since the addition of only a few extra links to create a torus doubles the bisection bandwidth and halves the network diameter, we will concentrate on the torus for the remaining discussion. Simulation results for mesh networks which include both uniform and non-uniform traffic can be

found elsewhere [BFS94].

For torus networks, saturation almost always occurs earlier when using oblivious routing than when using minimal adaptive or chaotic routing. One exception to this is the complement permutation, which achieves an unusually high throughput under oblivious routing when compared to the other non-uniform traffic patterns. This occurs because the complement is routed nearly conflict-free using oblivious routing, but causes two large hot spots to form in chaotic and Duato routing. See Section 5.3.1 for an explanation.

6.1.2 Throughput and Latency

In this section the behavior of the three routers is compared by examining expected throughput and expected latency for the traffic patterns. The graphs in Figure 6.1 - 6.5 display the throughput and latency versus the offered load for uniform random, bit reversal, complement, transpose and perfect shuffle traffic patterns. Results of more detailed comparisons are found elsewhere [BFS94]. Throughput of the Chaos

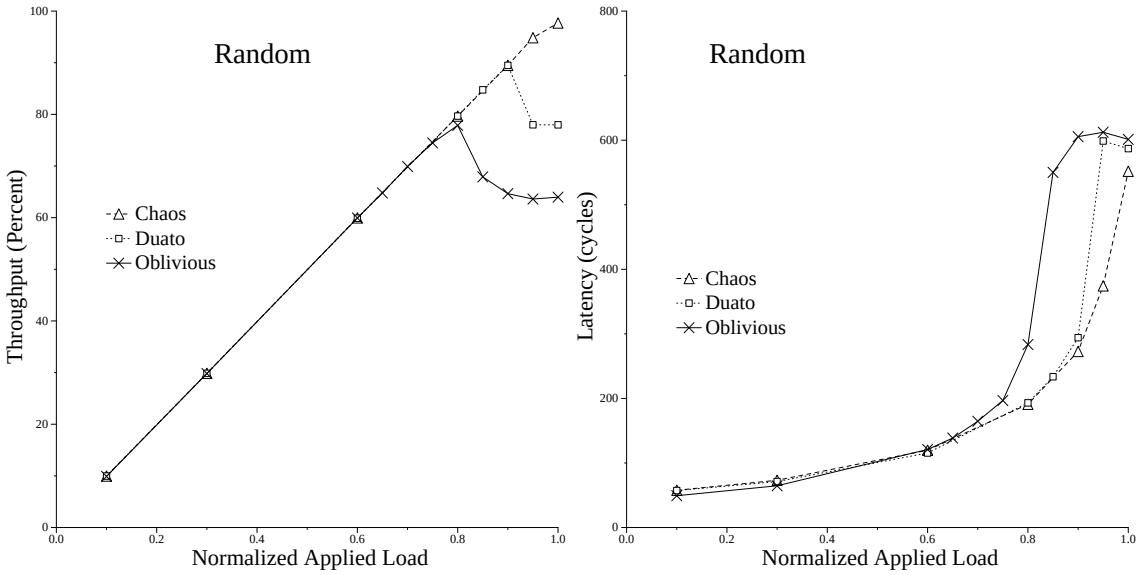

Figure 6.1: Throughput and latency for 256-node 2D torus networks with uniform random traffic.

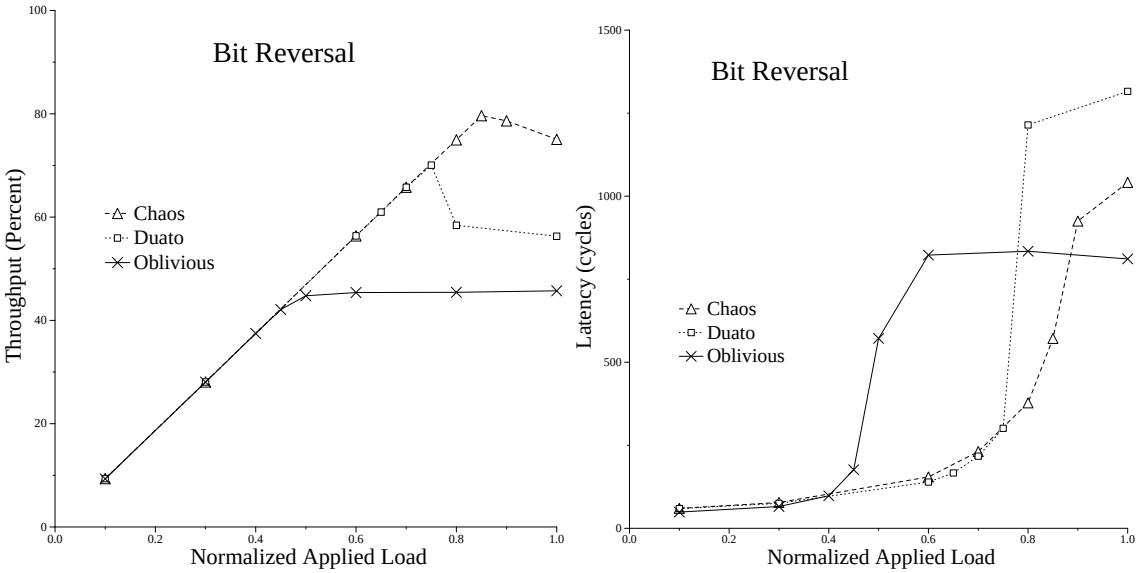

Figure 6.2: Throughput and latency for 256-node 2D torus networks with bit reversal traffic.

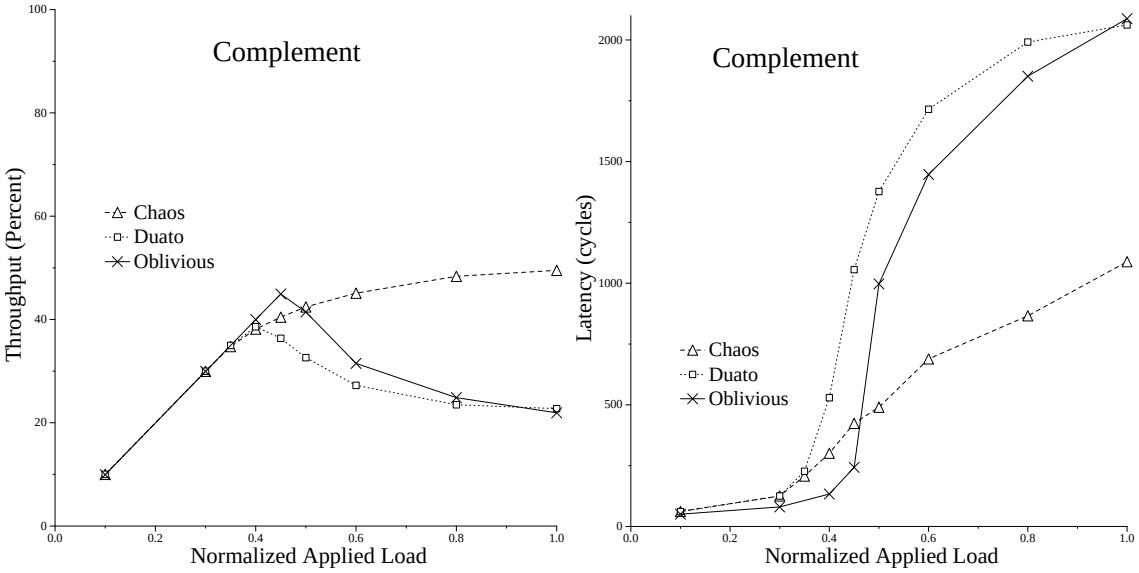

Figure 6.3: Throughput and latency for 256-node 2D torus networks with complement traffic.

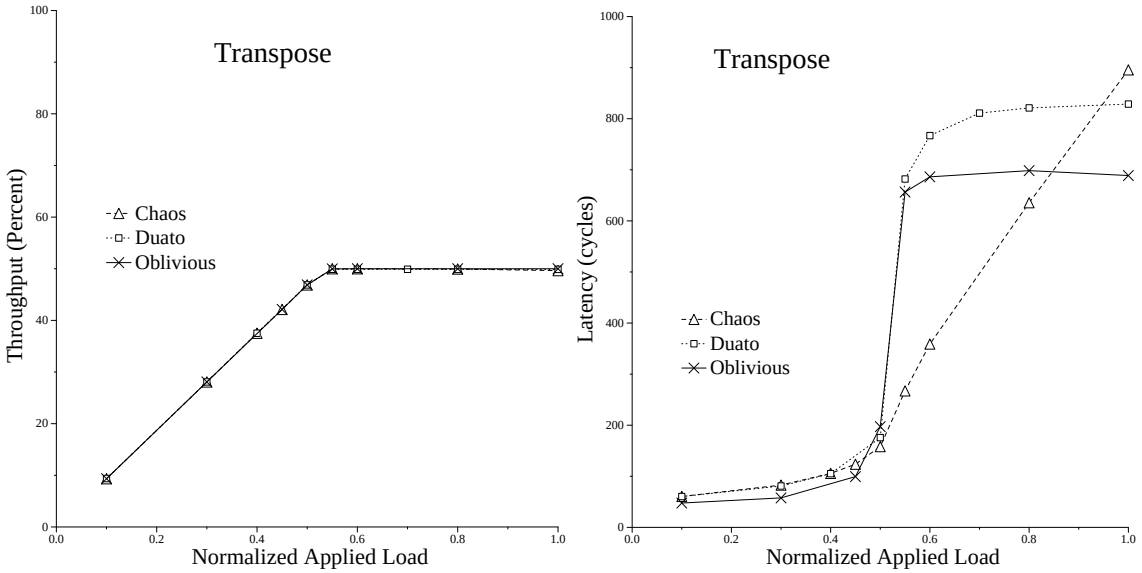

Figure 6.4: Throughput and latency for 256-node 2D torus networks with transpose traffic.

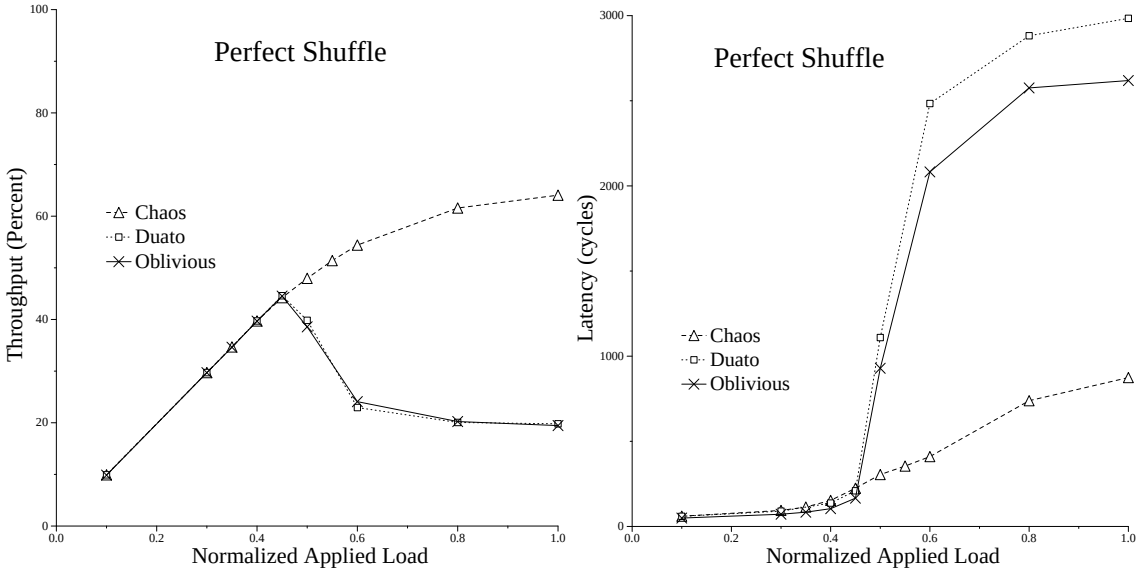

Figure 6.5: Throughput and latency for 256-node 2D torus networks with perfect shuffle traffic.

network is greater than or equal to that of the oblivious and Duato networks for the torus topology for all traffic patterns at all loads except two. These loads occur on the complement where the difference in normalized throughput of the routers is no more than five percent. The throughput for the bit reversal permutation on the Chaos torus is especially noteworthy. With the bit reversal permutation, the 79% throughput of the torus Chaos router at saturation is close to double the throughput of the oblivious router at 45% and nearly 10% better than the Duato algorithm at 70%.

The throughput and saturation points of the packet routing algorithm are substantially higher than those of traditional wormhole routing implementations which have limited buffering. Though these results do not show a fair comparison with respect to buffering or message header overhead if long messages were considered, it demonstrates the benefits of packet routing over traditional router configurations and shows that wormhole networks would benefit from additional buffering. It is likely that further studies would show there is a large range of message sizes for which packet routing is advantageous over wormhole routing because a blocking packet only holds a single resource in the network.

Throughput for the oblivious and Duato torus networks degrades after saturation with the complement, perfect shuffle, and random traffic patterns. For Duato, throughput degradation also occurs for the bit reversal traffic pattern. The most likely cause of this is the asymmetry in the oblivious and Duato networks introduced by the virtual channels used for deadlock avoidance in the torus [Bol92, AV94]. Throughput degradation has also been reported for other minimal adaptive routers [NS94].

At very low loads, latency for the torus is slightly higher for the Chaos and Duato routers than for the oblivious router, due to the lower per-hop latency of the oblivious router. At higher loads before Chaos saturation, the Chaos network generally has the lowest latency. After the Chaos router saturates, the relative latencies between the routers depend upon the traffic pattern under consideration. Due to the additional

buffering, the after saturation latency in packet routing is much higher than those experienced by wormhole routers. Thus, packet routers may benefit from congestion control to avoid this region of performance. For more on congestion control, see the future work described in Section 7.2.1.

6.1.3 Hot Spots

Simulations were run for six torus and six mesh hot spot arrangements. Results are only shown for two typical torus arrangements. The results of the others exhibit the same trends, although the particular values differ slightly from case to case. Full results are available elsewhere [BFS94]. When hot spots are added to random traffic,

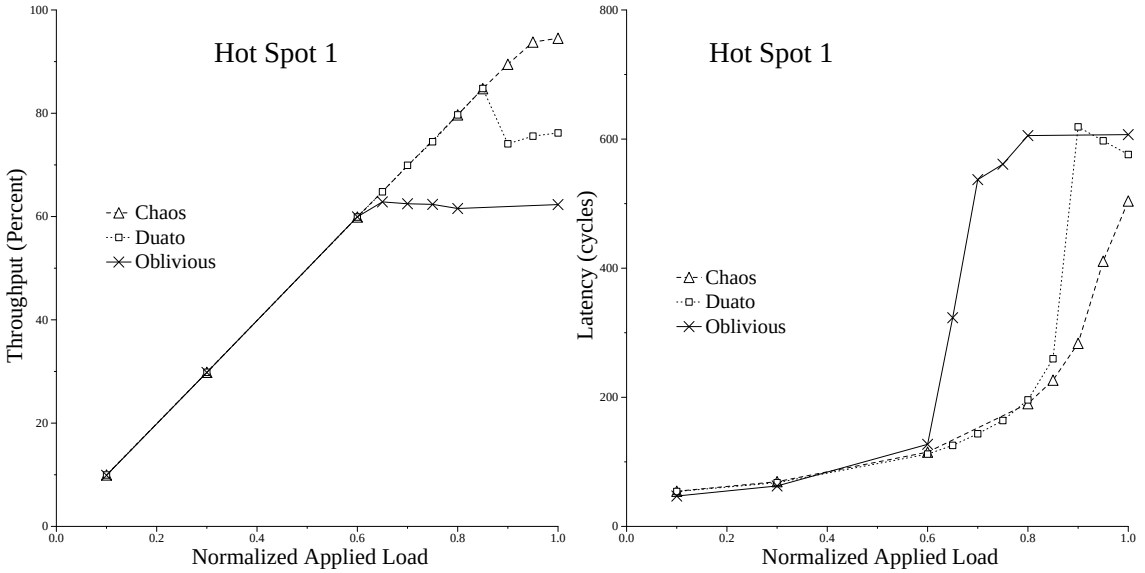

Figure 6.6: Throughput and latency for 256-node 2D torus networks with ten four-times hot spot traffic.

the maximum throughput of the oblivious torus is reduced from 76% to 63%, as shown in Figure 6.6. With the same hot spots, however, the Chaos torus achieves a maximum throughput of 95%, a slight reduction from its peak throughput of 98% achieved with random traffic. The Duato router experiences a drop in maximum throughput from

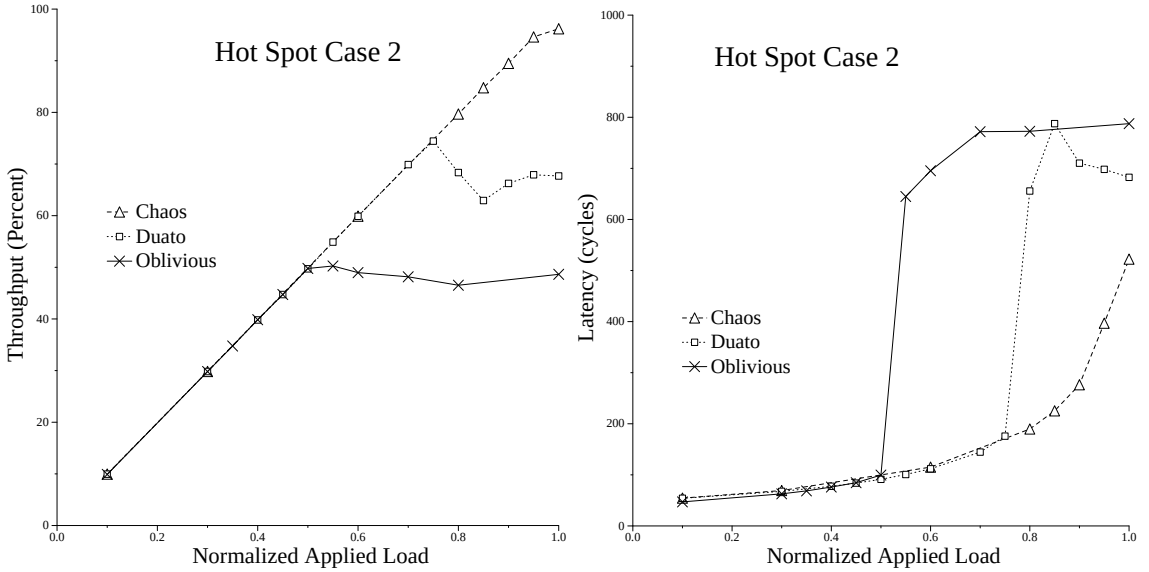

Figure 6.7: Throughput and latency for 256-node 2D torus networks with ten four-times hot spot traffic.

89% to 85%. For the second case, shown in Figure 6.7, the oblivious and Duato routers suffered further reductions in throughput. The Duato router reached a peak throughput of 74% and the oblivious router reached 50%. The Chaos router was comparable to the first case achieving a peak throughput of 96%.

The Duato router experiences throughput degradation with hot spot traffic. As mentioned earlier, this is most likely caused by the asymmetric use of the virtual channels employed for deadlock avoidance in the torus [Bol92, AV94].

In some of the hot spot cases, the throughput after saturation degrades and then appears to improve. The apparent gains in throughput are not significant, but occur because the network has greater variability in these cases. For the oblivious and Duato routers, almost all the confidence intervals for throughput have less than a one percent error. The percentage is greater for hot spot traffic after saturation, where it occasionally reaches three percent. Experimental results suggest that the increase in variability is caused by unfair access to the network. Some nodes are able to inject a substantially greater number of messages than other nodes. Thus, some

messages enter the network quickly, while others languish in the injection buffers before proceeding. This unfairness is most likely caused by the asymmetric use of virtual channels which is accentuated by the hot spots. The Chaos router, which does not use virtual channels, does not exhibit an increase in variability for hot spot traffic after saturation. See Section 7.2.1 for a more detailed discussion about unfair access to the network.

6.2 Summary

Although many multicomputer routing algorithms have been proposed in the last decade, no algorithm has yet eclipsed the popularity of the simple-but-fast oblivious routing algorithm. The primary reservations about adaptive routing algorithms are implementation complexity and out-of-order delivery. Network interface solutions to out-of-order delivery were mentioned in the beginning of this chapter. Any adaptive router is more complex than a straightforward dimension-order oblivious router. To compete, a careful design is required to produce a router that can deliver messages with the low node-to-node latency of an oblivious router. The simulations in this chapter are based on hardware designs of the oblivious and Chaos routers and demonstrate the feasibility and advantages of adaptive routing.

The experiments in this chapter explore the potential advantages and disadvantages of adaptive packet routing algorithms. The findings show that packet networks can achieve substantially higher throughputs than traditional wormhole networks. They also show that because of the larger amount of buffering used in the network, packet algorithms need to avoid operating beyond saturation conditions due to the high latencies experienced in this range. Further, the simulations show that non-minimal adaptive routing with controlled random derouting can be very effective at tolerating congestion and hot spots, and is even better than minimal adaptive routers. In particular, although the Chaos router has a slightly larger per-node routing latency

than the oblivious router, the throughput it achieves is much higher than with oblivious or minimal adaptive routing.

Chapter 7

CONCLUSIONS

Communication performance continues to be a critical problem in parallel computing. This thesis examines fundamental network issues such as routing algorithms, flow control, and router performance. Besides questioning the traditional methods of flow control and router implementation, we promote the idea of multi-class routing algorithms which increase network performance and simplify the network interface. Multi-class routing also reflects the emerging trend for an end-to-end approach to achieving communication performance which includes the software overhead, time in the network interface, and time in the network.

7.1 Contributions

The main contributions of this thesis are as follows.

- This thesis defines and formulates a method to evaluate multi-class routing algorithms. Multi-class routing algorithms support multiple classes of routing simultaneously, thereby allowing different applications, and even different messages, to select the most advantageous kind of routing and avoid overly restrictive routing whenever possible. Multi-class routing algorithms are important since they can selectively provide in-order delivery of messages. Consequently, multi-class routing is also an alternative to traditional adaptive routing algorithms for achieving high throughput, low latency network performance. Almost all adaptive algorithms deliver messages out-of-order, and therefore have been slow to gain acceptance due to the additional complexity required to reorder mes-

sages. Multi-class routing provides the benefits of adaptive routing without the problems of out-of-order message delivery.

- This dissertation also introduces a novel integrated multi-class routing algorithm called Triplex. Triplex is the first triple class algorithm supporting oblivious, minimal fully adaptive, and non-minimal fully adaptive routing. Unlike traditional minimal fully adaptive routing algorithms, Triplex does not reserve a virtual network or virtual channels exclusively for oblivious routing. Triplex is also the first non-minimal fully adaptive wormhole torus routing algorithm using deadlock avoidance.

- This thesis also compares two methods of implementing a routing algorithm, as an input driven and as an output driven router. In particular these methods describe techniques to move messages from the input to the output buffers of a router. The two methods are similar, but surprisingly the output driven router almost always performs as well as or better than the input driven router. Most algorithms are implemented as input driven routers, even though almost all can be implemented as either. This work suggests that hardware designers select output driven routers whenever this is not precluded by other design considerations.

- Finally this thesis uses simulation to explore competitive packet routing algorithms for a variety of uniform and non-uniform traffic patterns. The results demonstrate the advantages of adaptive routing, that packet routing achieves higher throughput than traditional wormhole networks, that packet routing algorithms could benefit from congestion control to avoid the high latencies experienced after saturation, and most importantly that non-minimal fully adaptive routing that uses controlled random derouting is very effective at increasing throughput over oblivious and minimal fully adaptive routers.

7.2 Future Work

7.2.1 Extensions

The idea of a multi-class router appears to be very useful. The Triplex router works well as both an oblivious and a minimal adaptive router. Unfortunately, the non-minimal mode does not do significantly better than the minimal mode, making it difficult to justify. Thus, the question remains; is there a multi-class routing algorithm that effectively supports all three classes (or alternatively an oblivious and non-minimal class) of routing, has non-minimal performance similar to the high throughput Chaos packet router, scales well, and is practical to implement? There are two specific improvements that could be made. The first is to find an algorithm that allows every message to deroute in a less restrictive fashion. This is rather difficult with long, wormhole messages. The second possibility is to try to find an algorithm that performs a majority of its routing in a network free of virtual channels. Virtual channels cause asymmetries in channel use which result in performance degradation.

From the simulations it is clear that on a 2D torus, packet routing can achieve throughput which is sometimes twice as great as the throughput achieved with traditional wormhole routing. The main disadvantage of packet routing, however, is its high after saturation latencies which appear to be dependent upon the type of traffic present in the network. Thus, to take advantage of the high throughput and low latency packet performance before saturation, without experiencing the higher post-saturation latencies, some sort of congestion control is necessary.

Very little work has been done in congestion control since slow network interfaces in parallel machines have limited the rate at which traffic can enter the network. Now that network interfaces are receiving attention, routing performance will become more critical to overall communication performance. One of the reasons congestion control is difficult is that network saturation is a measure of global network traffic. A particular router cannot easily acquire such global knowledge without contributing

to congestion. Furthermore, the point of saturation not only differs from workload to workload, but some traffic patterns may have only a few nodes which place high demands on the network, while the other nodes remain idle. Patterns such as these may still require congestion control. Cherkasova, Davis and Hodgson have recognized the need for congestion control with non-minimal routers [CDH+96]. They suggest three techniques for congestion control including alpha scheduling of messages, back-pressure flow control, and balanced injection control.

Fair access to the network is a problem closely related to injection control which has almost been completely ignored in the literature. In an ideal situation, each node has equal access to network bandwidth, independent of its location in the network. If the network is partitioned with each job allocated to a different partition, fair access helps provide equivalent performance, regardless of the partition assigned to a particular job. Within a partition, unfair access may cause nodes to wait for slower nodes unnecessarily; and in some cases, throughput of the network may be constrained by the slowest part of the network.

There are three main sources of unfairness in networks. Often, the scheme used to achieve deadlock-freedom causes unfairness. In particular, using virtual channels can cause performance problems resulting from asymmetric channel use, even when traffic is random [Bol92, AV94]. Asymmetries cause congestion around some nodes, while others remain clear. Because injection depends upon the availability of buffers, congestion inhibits injection. Thus, some nodes have easy access to the network, while others often wait much longer. See the literature [NS94] for an example of unfairness caused by a routing algorithm. Nodes also experience unfair access to the network if traffic from one partition travels through another partition. Traffic that traverses a partition occupies buffers in the network needed by local messages for injection and may even reduce the throughput of the resident job. A solution to this likely requires the cooperation of the network and operating system.

Almost all experiments have been with continuous routing. Continuous routing is

ideal for observing the performance of a routing algorithm over a complete range of applied loads and extracting such measures as saturation point, peak throughput, and behavior after saturation. Another interesting possibility is to model bursty, batch traffic which is subject to the constraints of a network interface (either a particular interface or one that represents the common features of those likely to be designed in the near future). This model is appealing since it may be more representative of real workloads, though a combination of the two models would likely provide the most insight.

7.2.2 Other Work

As technology continues its rapid progress, the performance differences between personal computers (PCs), workstations, and parallel computers continues to blur. Recently networks of workstations (NOWs) or networks of PCs have become a popular alternative to parallel computing. In this model, commodity processors are connected by a high speed, low cost network, such as Myrinet. If the nodes are less than 3 meters apart, a system area network (SAN) can be used, otherwise a local area network (LAN) is used. Parallel programs are executed on available processors in the network. In this domain low latency is essential, since lower latency allows finer grain parallelism. In the past, these networks were designed for high throughput only. For example, ATM switches generally suffer from high latency communication due to lack of attention to the network interface implementation. Therefore, adapting the low latency, high throughput multicomputer network techniques to SANs and LANs may be beneficial.

There are multiple layers of routing in these fast, low cost networks. Typically there are switches in the network which are connected either to other switches or to processing nodes. Routing on a switch is similar to routing in multicomputers and can benefit from many of the techniques used in the parallel domain. This has been demonstrated by Myricom, a company that developed a switch by transforming a

Caltech routing chip to support the SAN and LAN environments [BCF+95]. Routing between switches is less complex and likely to be performed by a separate routing technique in order to support more general topologies than are typically found on a switch. Support for arbitrary networks, though, may not be necessary since most LANs will serve workstations located in a 3D building or campus. Nevertheless, routing between switches needs to be efficient.

Several interesting possibilities exist for future work in this area. The first is to explore routing on the switch. The Myrinet switch uses simple oblivious wormhole routing and implements a cross bar which requires a quadratic number of internal switching elements, but only allows a linear number of processors to be connected to the switch. This ensures incremental scalability (which is important since computing environments typically add tens of processors at a time), but makes large switches expensive to build. Perhaps a cheaper interconnect that still supports a reasonable bisection bandwidth across the switch might suffice. There is also a proposed design to use the Chaos packet router in a LAN switch [MBES97]. Unlike Myrinet, this design allows processors to be connected to every switching element.

Another interesting problem is special support for long messages to ensure high throughput, to prevent short message performance degradation, and in some instances, to provide low latency for large messages. Switches have not been optimized to provide high throughput and low latency, though recently the need has become more apparent. For example, video and multimedia applications, which require high throughput for large volumes of data and low latency for responsive control messages, are becoming more common as the trend to web-based applications continues. Another example is the global memory system which uses spare network memory as a cache for virtual memory and file pages to reduce the need for disk accesses [JFV+96]. The system uses both short control messages and much larger subpage or page size messages. After a fault, computations may proceed once a request for a subpage from remote memory has been satisfied. This system is much more effective in networks

that provide low latency for both short and long messages.

Another interesting problem in fast, low cost networks is determining the switch topology to facilitate efficient routing algorithms between the switches. Since the topologies are unlikely to be regular, popular multicomputer techniques do not apply directly. Nevertheless because paths between nodes are relatively short, source routing techniques may suffice. If adaptivity is the key to performance, it may be necessary to provide alternative routes for messages. This may be difficult to do with source routing, since the source does not have current information about the rapid changes in network traffic.

Though an optimal switch topology may be the most effective, other constraints such as ease of network management may demand support for arbitrary topologies. In this case, a topology gathering phase is needed to acquire information about the current topology of the network. With this information a set of deadlock-free routes between each host can be constructed. This information is stored in routing tables and used for route selection when forwarding messages through the network. See the literature [MCSW97] for an example of a network mapping algorithm. Open problems include developing algorithms that send a minimal number of messages, that terminate within a small fixed time bound, and that allow dynamic reconfiguration, i.e. recomputing the topology while allowing running applications to continue uninterrupted. Another open issue is to define switch primitives, such as queries for a unique switch id, that can be added to a switch to make it easier to configure the network.

Finally there is a need to provide quality of service in high speed, low cost networks. Quality of service is important for applications that require predictable network performance such as for voice, video, and other real-time applications. Quality of service can also be used to provide fair-access to the network (described earlier). Traditional solutions are complex and often use expensive priority queues and per-connection queueing. Besides limiting the number of simultaneous connections, it

makes switches very expensive. Rotating Combined Queueing (RCQ) is one proposed solution that reduces the costs of providing quality of service guarantees [KC96]. The RCQ switch uses a table look-up rather than a priority queue operation when a message arrives. This method avoids the use of hardware priority queues, per connection queueing, and allows best effort traffic to utilize unused bandwidth.

BIBLIOGRAPHY

[AA94] B. Abali and C. Aykanat. Routing algorithms for IBM SP1. In *Lecture Notes in Computer Science*, volume 853, pages 161–75, 1994.

[ABC+95] A. Agarwal, R. Bianchini, D. Chaiken, et al. The MIT Alewife machine: architecture and performance. In *Intl. Sym. on Computer Arch.*, pages 2–13, 1995.

[ABLL92] T.E. Anderson, B.N. Bershad, E.D. Lazowska, and H.M. Levy. Scheduler activations: Effective kernel support for the user-level management of parallelism. *ACM Transactions on Computer Systems*, 10:53–79, 1992.

[ACK+90] R. Alverson, D. Callahan, B. Koblenz, A. Porterfield, and B. Smith. The Tera computer system. In *Intl. Conf. on Supercomputing*, pages 1–6, 1990.

[AFNV90] M. Annaratone, M. Fillo, K. Nakabayashi, and M. Viredaz. The K2 parallel processor: Architecture and hardware implementation. In *Intl. Sym. on Computer Arch.*, pages 92–101, 1990.

[ALBL91] T.E. Anderson, H.M. Levy, B.N. Bershad, and E.D. Lazowska. The interaction of architecture and operating system design. In *Proc. of the Intl. Conf. on Arch. Support for Prog. Lang. and Op. Sys.*, pages 108–120, 1991.

[AS88] W.C. Athas and C.L. Seitz. Multicomputers: Message-passing concurrent computers. *Computer*, pages 9–23, August 1988.

[ASAA96] Y. Aydogan, C. Stunkel, C. Aykanat, and B. Abali. Adaptive source routing in multistage interconnection networks. In *Intl. Parallel Processing Sym.*, pages 258–67, 1996.

[AV94] Vikram S. Adve and Mary K. Vernon. Performance analysis of mesh interconnection networks with deterministic routing. *IEEE Transactions on Parallel and Distributed Systems*, 5(3):225–46, 1994.

[BC93] Rajendra V. Boppana and Suresh Chalasani. A comparison of adaptive wormhole routing algorithms. In *Proceedings of the 20th International Symposium on Computer Architecture*. IEEE, May 1993.

[BCF+95] N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, and Wen-King-Su. Myrinet: a gigabit-per-second local area network. *IEEE Micro*, 15(1):29–36, 1995.

[BFS87] P. Bratley, B.L. Fox, and L.E. Schrage. *A guide to simulation*. Springer-Verlag, 2nd edition, 1987.

[BFS94] K. Bolding, M.L. Fulgham, and L. Snyder. The case for chaotic adaptive routing. Technical Report TR CSE-94-02-04, University of Washington, Seattle, WA, 1994.

[BFSar] K. Bolding, M.L. Fulgham, and L. Snyder. The case for chaotic adaptive routing. *IEEE Transactions on Computers*, to appear.

[BJM+96] G. Buzzard, D. Jacobson, M. Mackey, S. Marovich, and J. Wilkes. An implementation of the Hamlyn sender-managed interface architecture. In *Proc. of the Sym. on Operating Sys. Design and Implementation*, pages 245–259, 1996.

[BLA+94] M.A. Blumrich, K. Li, R. Alpert, C. Dubnicki, and E.W. Felten. Virtual memory mapped network interface for the SHRIMP multicomputer. In *Proc. of the Intl. Sym. on Computer Arch.*, pages 142–153, 1994.

[Bol92] Kevin Bolding. Non-uniformities introduced by virtual channel deadlock prevention. Technical Report 92–07–07, University of Washington, Seattle, WA, July 1992.

[Bol93a] K. Bolding. Multicomputer interconnection network channel design. Technical Report UW-CSE-93-12-03, University of Washington, Seattle, 1993.

[Bol93b] Kevin Bolding. *Chaotic Routing: Design and Implementation of an Adaptive Multicomputer Network Router*. PhD thesis, University of Washington, Seattle, July 1993.

[BY94] K. Bolding and W. Yost. Design of a router for fault-tolerant networks. In *Lecture Notes in Computer Science*, volume 853, pages 226–40, 1994.

[CDH+96] L. Cherkasova, A. Davis, R. Hodgson, V. Kotov, I. Robinson, and T. Rokicki. Components of congestion control. In *Proc. of the Sym. on Par. Alg. and Arch.*, pages 208–210, 1996.

[CDS93] B. Coates, A. Davis, and K. Stevens. The post office experience: designing a large asynchronous chip. In *HICSS*, 1993.

[CHKM93] R. Cypher, A. Ho, S. Konstantinidou, and P. Messina. Architectural requirements of parallel scientific applications with explicit communication. In *Proc. of the International Symposium on Computer Architecture*, pages 2–13, 1993.

[Cho93] S. Choi. Worms and packets: A study of two switching techniques for oblivious routing networks. June 1993.

[CK92] A.A. Chien and J.H. Kim. Planar-adaptive routing: Low-cost adaptive networks for multiprocessors. In *Proc. of the Intl. Sym. on Computer Arch.*, pages 268–277, 1992.

[CKP+93] D. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, and T. vonEicken. LogP: Towards a realistic model of parallel computation. In *Proc. of Sym. on Prin. and Prac. of Par. Prog.*, May 1993.

[Cor91] Intel Corp. A touchstone DELTA system description. Technical report, 1991.

[CR94] L. Cherkasova and T. Rokicki. Alpha message scheduling for packet-switched interconnects. Technical Report TR HPL-94-72, Hewlett-Packard Labs, 1994.

[DA93] William J. Dally and Hiromichi Aoki. Deadlock-free adaptive routing in multicomputer networks using virtual channels. *IEEE Transactions on Parallel and Distributed Systems*, 4(4):466–75, 1993.

[Dal90a] W. Dally. Network and processor architecture for message-driven computers. In R. Suaya and G. Birtwistle, editors, *VLSI and Parallel Computation*, pages 140–222. Morgan Kaufmann Publishers, 1990.

[Dal90b] W.J. Dally. The J-Machine system. In P.Winston and S. Shellard, editors, *Artificial Intelligence at MIT: Expanding Frontiers*. MIT Press, 1990.

[Dal92] W. Dally. Virtual-channel flow control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194–205, March 1992.

[DDH+94] W.J. Dally, L.R. Dennison, D. Harris, K. Kan, and T. Xanthopoulos. The reliable router: A reliable and highh-performance communications substrate for parallel computers. In *Lecture Notes in Computer Science*, volume 853, pages 241–55, 1994.

[DDY95] B.V. Dao, J. Duato, and S. Yalamanchili. Configuarable flow control mechanisms for fault-tolerant routing. In *Proc. of the Intl. Sym. on Computer Arch.*, pages 220–29, 1995.

[DHR+94] A. Davis, R. Hodgson, I. Robinson, L. Cherkasova, V. Kotov, and T. Rokicki. R2: A damped adaptive router design. In *Lecture Notes in Computer Science*, volume 853, pages 295–317, 1994.

[DL94] J. Duato and P. Lopez. Performance evaluation of adaptive routing algorithms for k-ary n-cubes. In *Lecture Notes in Computer Science*, volume 853, pages 45–59, 1994.

[DLD93] L.R. Dennison, W.S. Lee, and W.J. Dally. High performacne bidirectional signalling in VLSI systems. In *Research on Integrated Systems: Proc. of the 1993 Sym.*, 1993.

[DS86] W. Dally and C. Seitz. The torus routing chip. *Journal of Distributed Computing*, 1(3), 1986.

[DS87] W. Dally and C. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36(5):547–553, May 1987.

[Dua93] J. Duato. A new theory of deadlock-free adaptive routing in wormhole networks. *IEEE Transactions on Parallel and Distributed Systems*, 4(4):466–475, April 1993.

[Dua94] J. Duato. A theory of fault-tolerant routing in wormhole networks. In *Intl. Conf. on Parallel and Dist. Sys.*, pages 600–7, 1994.

[Dua95] J. Duato. A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks. *IEEE Transactions on Parallel and Distributed Systems*, 6(10), October 1995.

[DYN97] J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Networks: An Engineering Approach.* IEEE Press, to appear 1997.

[Fel93] E.W. Felten. *Protocol Compilation: High-Performance Communication for Parallel Programs.* PhD thesis, University of Washington, Seattle, WA, 1993.

[FKD95] M. Fillo, S.W. Keckler, and W.J. Dally. The M-Machine multicomputer. In *Intl. Symp. on Micro.*, pages 146–56, 1995.

[FM85] G. Fishman and L. Moore. An exhaustive analysis of multiplicative congruential random number generators with modulus $2^{31} - 1$. *Journal of the American Statistical Association*, 77:129–186, 1985.

[FS93] M.L. Fulgham and L. Snyder. A study of chaotic routing with nonuniform traffic. Technical Report 93–06–01, University of Washington, Seattle, WA, 1993.

[FS96a] M. Fulgham and L. Snyder. Triplex router: a versatile torus routing algorithm. Technical Report UW-CSE-96-01-11, Univ. of Washington, Seattle, 1996.

[FS96b] M.L. Fulgham and L. Snyder. A comparison of input and output driven routers. Technical Report UW-CSE-96-06-01, Univ. of Washington, Seattle, 1996.

[FS96c] M.L. Fulgham and L. Snyder. A comparison of input and output driven routers. In *Lecture Notes in Computer Science, Proc. of Euro-Par '96*, volume 1123, pages 195–204, 1996.

[FS97a] M.L. Fulgham and L. Snyder. Integrated multi-class routing. In *Parallel Computer Routing and Communication Workshop. To appear in Lecture Notes in Computer Science*, 1997.

[FS97b] M.L. Fulgham and L. Snyder. Triplex: A multi-class routing algorithm. In *Sym. on Parallel Alg. and Arch.*, pages 127–138, 1997.

[GI90] P. Glynn and D.L. Iglehart. Simulation output analysis using standardized time series. *Mathematics Operations Research*, 15(1):1–16, February 1990.

[GN92a] C.J. Glass and L.M. Ni. Adaptive routing in mesh-connected networks. In *Proc. of the Intl. Conf. on Distributed Computing Systems*, pages 12–19, 1992.

[GN92b] C.J. Glass and L.M. Ni. Maximally fully adaptive routing in 2D meshes. In *Intl. Conf. on Parallel Proc.*, volume I, pages 101–104, 1992.

[GN94] Christopher J. Glass and Lionel M. Ni. The turn model for adaptive routing. *JACM*, 41(5):874–902, 1994.

[GPBS94] L. Gravano, G. Pifarré, P.E. Berman, and J.L.C. Sanz. Adaptive deadlock- and livelock-free routing with all minimal paths in torus networks. *IEEE Transactions on Parallel and Distributed Systems*, 5(12):1233–1251, 1994.

[Gün81] K.D. Günther. Prevention of deadlocks in packet-switched data transport systems. *IEEE Trans. on Communication*, COM-29:512–524, April 1981.

[GY93] P.T. Gaughan and S. Yalamanchili. Adaptive routing protocols for hypercube interconnection networks. *IEEE Computer Magazine*, 26(5):12–23, May 1993.

[JFV+96] H. Jamrozik, M. Feeley, G. Voelker, J. Evans, A. Karlin, H. Levy, and M. Vernon. Reducing network latency using subpages in a global memory environment. In *Proc. of the Intl. Conf. on Arch. Support for Prog. Lang. and Op. Sys.*, 1996.

[JMY89] C.R. Jesshope, P.R. Miller, and J. Yantchev. High performance communications in processor networks. In *ISCA*, pages 150–157, 1989.

[K+94] J. Kuskin et al. The Stanford FLASH multiprocessor. In *Proc. of ISCA*, pages 302–313, 1994.

[KC96] J.H. Kim and A.A. Chien. Rotating combined queueing (RCQ): Bandwiwdth and latency guarantees in low-cost, high-performance networks. In *Intl. Sym. on Computer Arch.*, pages 226–236, 1996.

[KK79] P. Kermani and L. Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks*, 3:267–286, 1979.

[KLC94] J.H. Kim, Z. Liu, and A.A. Chien. Compressionless routing: a framework for adaptive and fault-tolerant routing. In *Proc. of the Intl. Sym. on Computer Arch.*, pages 289–300, 1994.

[Knu81] D.E. Knuth. *The Art of Computer Programming*, volume 2. Addison-Wesley, 2nd edition, 1981.

[Kon90] Smaragda Konstantinidou. Adaptive, minimal routing in hypercubes. In *6th MIT Conference on Advanced Research in VLSI*, pages 139–153, 1990.

[Kon91] Smaragda Konstantinidou. *Deterministic and Chaotic Adaptive Routing in Multicomputers*. PhD thesis, University of Washington, Seattle, WA, May 1991.

[Kon92] S. Konstantinidou. Priorities in nonminimal, adaptive routing. In *Intl. Conf. on Par. Processing*, volume I, pages 67–71, 1992.

[Kon94] S. Konstantinidou. The segment router: a novel router design for parallel computers. In *Proc. of the Sym. of Parallel Algorithms and Architectures*, pages 364–373, 1994.

[KP95] Anjan K.V. and T.M. Pinkston. An efficient, fully adaptive deadlock recovery scheme: DISHA. In *Proc of the Intl. Sym. on Comp. Arch.*, pages 201–210, 1995.

[KPD96] Anjan K.V., T.M. Pinkston, and J. Duato. Generalized theory for deadlock-free adaptive wormhole routing and its application to Disha concurrent. In *Intl. Parallel Processing Sym.*, pages 815–821, 1996.

[KS94] S. Konstantinidou and L. Snyder. The chaos router. *IEEE Transactions on Computers*, 43(12):1386–97, December 1994.

[LAD+96] C.E. Leiserson, Z.S. Abuhamdeh, D.C. Douglas, et al. The network architecture of the Connection Machine CM-5. *Journal of Par. and Dist. Computing*, 33(2):145–58, 1996.

[LC94] Z. Liu and A.A. Chien. Hierarchical adaptive routing: A framework for fully adaptive and deadlock-free wormhole routing. In *Sym. on Par. and Distr. Processing*, pages 688–695, 1994.

[Lea76] G.P. Learmonth. Empirical tests of multipliers for the prime modulus random number generator $x_{i+1} = ax_i (\mathrm{mod} 2^{31} - 1)$. In *Proc. of the 9th Interface Symposium on Computer Science and Statistics*, pages 178–183, 1976.

[Lei92a] F. Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann Publishers, 1992.

[Lei92b] T. Leighton. Methods for message routing in parallel machines. In *Proc. of Sym. on the Theory of Computing*, pages 77–96, 1992.

[LGM69] P.A.W. Lewis, A.S. Goodman, and J.M. Miller. A pseudo-random number generation for the system 360. *IBM Systems Journal*, 8:136–146, 1969.

[LH91] D. H. Linder and J. C. Harden. An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes. *IEEE Transactions on Computers*, C-40(1):2–12, January 1991.

[LHH91] A. Landin, E. Hagersten, and S. Haridi. Race-free interconnection networks and multiprocessor consistency. In *Proc. of the Intl. Sym. on Computer Arch.*, pages 106–115, 1991.

[LL74] G.P. Learmonth and P.A.W. Lewis. Statistical tests of some widely used and recently proposed uniform ran dom number generators. In *Proc. of the 7th Conf. on Comp. Sci, and Stats. Interface*, 1974.

[LLG+92] D. Lenoski, J. Laudon, K. Gharachorloo, W.D. Weber, A. Gupta, J. Hennessy, M. Horowitz, and M.S. Lam. The Stanford Dash multiprocessor. *IEEE Computer*, 25(3):63–79, 1992.

[LNP91] K. Li, J.F. Naughton, and J.S. Plank. An efficient checkpointing method for multicomputers with wormhole routing. *International Journal of Parallel Programming*, 20:159–80, 1991.

[LO89] P.A.W. Lewis and E.J. Orav. *Simulation Methodology for Statisticians, Operations Analysts, and Engineers*, chapter Uniform Pseudo-Random Variable Generation, pages 65–99. Wadsworth Brooks/Cole, 1989.

[Max89] N.F. Maxemchuk. Comparison of deflection and store-and-forward techniques in the manhattan street and shuffle-exchange networks. In *Proc. of IEEE INFOCOM*, pages 800–809, 1989.

[MBES94] N. McKenzie, K. Bolding, C. Ebeling, and L. Snyder. CRANIUM: An interface for message passing on adaptive packet routing networks. In *Lecture Notes in Computer Science*, volume 853, pages 266–80, 1994.

[MBES97] N. McKenzie, K. Bolding, C. Ebeling, and L. Snyder. ChaosLAN: Design and implementation of a gigabit LAN using chaotic routing. In *Parallel Computer Routing and Communication Workshop. To appear in Lecture Notes in Computer Science*, 1997.

[MCSW97] A.M. Mainwaring, B.N. Chun, S. Schleimer, and D.S. Wilkerson. System area network mapping. In *Sym. on Parallel Alg. and Arch.*, pages 116–26, 1997.

[MCW97] P. May, S.M. Chai, and D.S. Wills. HiPER-P: An efficient, high-performance router for multicomputer interconnection networks. In *PCRCW, Lecture Notes in Computer Science*, to appear 1997.

[Muñ91] David Muñoz. Multivariate standardized time series in the analysis of simulation out put. Technical Report TR-68, Operations Research, Stanford University, Palo Alto, CA, April 1991.

[MZ91] G. Marsaglia and A. Zaman. A new class of random number generators. *The Annals of Applied Probability*, 1(3):462–480, 1991.

[ND92] P.R. Nuth and W.J. Dally. The J-Machine network. In *IEEE Intl. Conf. on Computer Design*, pages 420–23, 1992.

[Nga89] J. Y. Ngai. *A Framework for Adaptive Routing in Multicomputer Networks*. PhD thesis, California Institute of Technology, Pasadena, CA, May 1989.

[NM93] L.M. Ni and P.K. McKinley. A survey of wormhole routing techniques in direct networks. *IEEE Computer Magazine*, 26(2):62–76, February 1993.

[NS89] J. Y. Ngai and C. L. Seitz. A framework for adaptive routing in multicomputer networks. In *Proc. of the Sym. of Parallel Alg. and Arch.*, pages 1–9, 1989.

[NS94] T.D. Nguyen and L. Snyder. Performance analysis of a minimal adaptive router. In *Lecture Notes in Computer Science*, volume 853, pages 31–44, 1994.

[Nug88] S.F. Nugent. The iPSC/2 direct-connect communications technology. In *Conf. on Hypercube Concurrent Computers and Applic.*, pages 51–60, 1988.

[Pap93] G.M. Papadopoulos, September 1993. Presentation to DIMACS Workshop on Models, Architectures and Technologies for Parallel Computation.

[PC92] G.M. Papadopoulos and D.E. Culler. Monsoon: an explicit token-store architecture. In *Proc. of the International Symposium on Computer Architecture*, pages 82–91, May 1992.

[Per92] M.J. Pertel. A critique of adaptive routing. Technical Report CS-TR-92-06, California Institute of Technology, 1992.

[PV81] F. Preparata and J. Vuillemin. The cube-connected cycles: a versatile network for parallel computation. *Communications of the ACM*, 24(5):300–309, 1981.

[Ros90] Sheldon M. Ross. *A course in simulation.* Macmillan, 1990.

[Ros96] Sheldon M. Ross. *Stochastic processes.* Wiley, 2nd edition, 1996.

[SAF88] C.L. Seitz, W.C. Athas, and C.M. Flaig. The architecture and programming of the Ametek series 2010 multicomputer. In *Conf. on Hypercube Concurrent Computers and Applic.*, pages 33–6, 1988.

[SB77] H. Sullivan and T.R. Bashkow. A large scale, homogeneous, fully distributed parallel machine, I. In *Intl. Sym. on Computer Arch.*, pages 105–124, 1977.

[Sch97] L. Schwiebert. Deadlock-free oblivious wormhole roting with cyclic dependencies. In *Proc. of the Sym. on Par. Alg. and Arch.*, pages 149–158, 1997.

[Sei92] C.L. Seitz. MOSAIC C: an experimental fine-grain multicomputer. In *Future tendencies in computer science, control and applied mathematics*, pages 69–85, 1992.

[She93] G.S. Shedler. *Regenerative stochastic simulation.* Academic Press, 1993.

[SJ95] L. Schwiebert and D.N. Jayasimha. Optimal fully adaptive womhole routing for meshes. *Journal of Parallel and Distributed Computing*, 27(1):56–70, 1995.

[SJ96] L. Schwiebert and D.N. Jayasimha. A universal proof technique for deadlock-free routing in interconnection networks. *Journal of Parallel and Distributed Computing*, 32(1):103–17, 1996.

[Smi81] B.J. Smith. Architecture and applications of the HEP multiprocessor computer sytem. In *Proc. of SPIE*, pages 241–248, 1981.

[SS93] Charles L. Seitz and Wen-King Su. A family of routing and communication chips based on the Mosaic. In *Sym. on Integrated Systems: Proc. of the 1993 Washington Conf.*, 1993.

[ST94] S. Scott and G. Thorson. Optimized routing in the cray T3D. In *Lecture Notes in Computer Science*, volume 853, pages 241–55, 1994.

[ST96] S.L. Scott and G. M. Thorson. The cray T3E network: Adaptive routing in a high performance 3D torus. In *Proc. of the HOT Interconnects IV*, 1996.

[vEDCGS92] T. von Eicken D.E. Culler, S.C. Goldstein, and K.E. Schauser. Active messages: a mechanism for integrated communication and computation. In *Proc. of the Intl. Sym. on Computer Arch.*, pages 256–266, May 1992.

[W+88] D. Whobrey et al. A communications chip for multiprocessors. In *Proc. of CONPAR*, 1988.

[WSC+95] B. Wei, G. Stoll, D.W. Clark, E.W. Felten, and K. Li. Synchronization for a multi-port frame buffer on a mesh-connected multicomputer. In *Proc. of the Parallel Rendering Symposium*, pages 81–88, 1995.

[YDG] S. Yalamanchili, B.V. Dao, and P.T. Gaughan. Direction order oblivious routing in meshes and k-ary n-cubes. Unpublished paper, 1993.

[YJ89] J. Yantchev and C.R. Jesshope. Adaptive, low latency, deadlock-free packet routing for networks of processors. *IEE Proc., Part E*, 136(3):178–186, May 1989.

Appendix A

TRIPLEX

A.1 Proofs of Deadlock-freedom

We show the Triplex algorithm is deadlock-free by applying a theorem of Schwiebert and Jayasimha [SJ96]. To keep this section self-contained, we briefly review in this paragraph the definitions needed to prove the theorems. A *waiting buffer* is a buffer a message can wait to acquire when all other buffers specified by the routing relation cannot be selected. The *buffer waiting graph* (BWG) for a routing algorithm is a directed graph BWG $= (B, E)$ where the vertex set B represents the set of buffers in the network and the edge set E represents pairs of buffers (b_1, b_2) where a message occupying buffer b_1 can wait for buffer b_2. A routing algorithm is *wait-connected* if a message always has at least one waiting buffer.

There is a *waiting dependence* between two buffers a and b if a message can use buffer a and wait for waiting buffer b. Waiting dependencies can occur between buffers more than one hop away. There is also a *waiting dependence* between a and b if there is a sequence of buffers $(a = b_1, b_2, \ldots, b_s = b)$ such that there is a message m_i that uses buffer b_i and waits for waiting buffer b_{i+1} for $1 \leq i < s$. This is equivalent to having a path from a to b in the buffer waiting graph being considered.

Theorem 1: [SJ96] If a routing algorithm, R, is wait-connected and the BWG for R is acyclic, then R is deadlock-free.

Next we define the terminology and notation needed. Let DO be the dimension-order Dally-Seitz oblivious, deadlock-free wormhole routing algorithm [DS87]. This algorithm is minimal, makes routing decisions independent of the message source and input channel, and is suffix closed*. Furthermore, this algorithm is wait-connected,

*Informally, every path a message takes to a particular destination through a node a can be used

deadlock-free, and has an acyclic BWG. Let DO_i be the restriction of the Dally Seitz algorithm to dimension i. The restricted algorithm DO_i also makes routing decisions independent of dimensions other than i.

The DO algorithm routes a message from the lowest dimension 0 to the highest dimension $n-1$ by applying DO_0, then DO_1, ..., and finally DO_{n-1}, where the direction in each dimension is chosen to make the message route minimal. Although the specific DO_i rules differ slightly for the mesh and the torus[†], the particular details are not relevant. Hence, the mesh algorithm will not be distinguished from the torus algorithm except by context.

Let b_i^+ (b_i^-) denote the buffers corresponding to a virtual channel in the positive (negative) direction of dimension i. The direction will only be specified when a distinction between the positive and negative direction is necessary. The distinction between an input buffer b_i^{in} in dimension i and an output buffer b_i^{out} in dimension i will only be made for packet routing, and only when necessary. When using packet routing, the terms packet and message are used interchangeably.

For convenience, we assume that for all the routing algorithms presented, if a message waits, it waits on the buffer specified by DO. Thus with wormhole routing, all waiting buffers are restricted buffers; while with packet routing, all output waiting buffers are restricted buffers. For packet routing, a packet in an input buffer waits on the appropriate buffer in the lowest dimension it needs to correct, while a packet in an output buffer must wait on the corresponding input buffer (there is no other choice). We proceed by showing that each algorithm has an acyclic BWG, and hence by Theorem 1 is deadlock-free. Later, we show how to remove this waiting restriction so that if a message waits, it waits on all the buffers it needs.

A.1.1 The Mesh Algorithm

For ease of explanation, we describe the mesh algorithm first. The packet-switched version is presented first and is followed by the wormhole version.

---

by a message injected at node a to reach the same destination.

[†]The torus algorithm uses two virtual channels per direction per dimension, while the mesh uses one.

Packet Triplex on the Mesh

In packet routing each message in a cycle occupies a single buffer, regardless of whether store-and-forward or virtual cut-through flow control is used. Thus waiting dependencies between buffers occur only when a message in one buffer waits directly for another waiting buffer in the same or neighboring node. Furthermore, since a packet in an input buffer can only wait on a restricted buffer, a cycle in the BWG only contains restricted buffers.

Fact 1: For wormhole (packet) routing, when a message waits in an (input) buffer, it waits for the restricted buffer specified by *DO* which is in the *minimal* direction of l, the lowest dimension it needs to correct.

Lemma 2: Given a cycle in the BWG where l is the lowest dimension (input and output) buffer in the cycle, the lowest dimension any message needed to correct when it was routed to the output buffer corresponding to the input buffer it holds in the cycle is l.

Proof: Consider a cycle in the BWG where the lowest dimension in the cycle is l. A message in a cycle only occupies one buffer. So if a message m in a cycle needed to correct a dimension lower than l, when it was routed to the output buffer corresponding to the input buffer it holds in the cycle, m would wait on this dimension. Thus, l would not be the lowest dimension in the cycle. □

Lemma 3: Given a cycle in the BWG where l is the lowest dimension (input and output) buffer in the cycle, all input buffers in the cycle in dimension l have been used minimally according to *DO*.

Proof: Consider a cycle in the BWG where l is the lowest dimension buffer in the cycle. Let m be a message in a restricted (all buffers in the cycle are restricted) input buffer b_l^{in} in dimension l in the cycle. By Lemma 2 dimension l was the lowest dimension m needed to correct when it was routed to restricted output buffer b_l^{out} corresponding to input buffer b_l^{in}. And by definition of the routing algorithm, message m was routed to its restricted buffer in dimension l by *DO* rules, which are minimal.

□

Lemma 4: There are no cycles in the BWG where the lowest dimension in the cycle is used in the positive and negative direction.

Proof: Assume there is a cycle in the BWG. Let l be the lowest dimension of the cycle, and assume l is used in the positive and negative direction. Then, there exists a message m in an input buffer in the cycle that is not in the positive direction of dimension l, which waits for a restricted output buffer b_l^+ in the positive direction of l. We show that message m violates the routing rules, and hence no such cycle exists. See Figure A.1 for an example of such a configuration. There are two cases. The first case assumes m resides in a restricted buffer in a dimension greater than l, and the second assumes that m occupies a restricted buffer in the negative direction of dimension l. With packet routing, there are no other cases to consider, since each message in a cycle resides in exactly one restricted buffer.

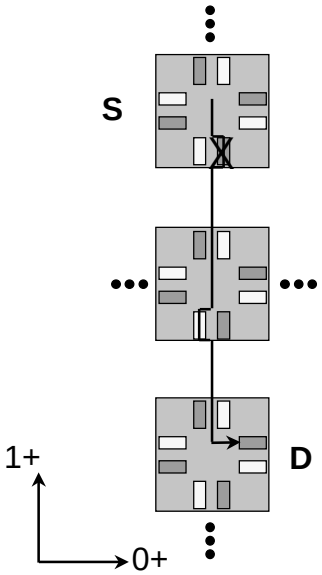

Figure A.1: An example of a route that violates the rules of Triplex. (The illegal buffer is marked.) The shaded buffers are the restricted output buffers, while the unshaded buffers are unrestricted.

First, message m holds a restricted input buffer b_h in the cycle, for some dimension $h > l$, and waits for restricted output buffer b_l^+ in the positive direction of dimension

l. By Fact 1, this direction is minimal. By Lemma 2, l is the lowest dimension m needed to correct when m was routed to buffer b_h. But m is not allowed to use a restricted buffer b_h in a dimension greater than l, when it needs to route in the positive direction of l.

Second, message m occupies a restricted input buffer b_l^- in the negative direction of dimension l in the cycle, and waits for the restricted output buffer b_l^+ in the positive direction of dimension l. By Lemma 3, m was routed minimally in the negative direction to its current input buffer b_l^- in the cycle. And by Fact 1, m waits in the minimal direction for b_l^+ which is positive. This is a contradiction. □

Theorem 5: The packet-switched version of the Triplex routing algorithm, *packet Triplex*, for the mesh is deadlock-free.

Proof: The algorithm is wait-connected since a message in a buffer can always wait on the waiting buffer specified by DO, which is wait-connected.

A cycle must use both directions of each dimension on a mesh. Thus, it follows from Lemma 4 that the BWG is acyclic. And by Theorem 1, the algorithm is deadlock-free, since the BWG is wait-connected and acyclic. □

Wormhole Triplex on the Mesh

The (wormhole) Triplex algorithm is complicated by buffer dependencies caused by arbitrary length messages. Since messages can only wait on restricted buffers, any cycle in the buffer dependencies is created from waiting dependencies between restricted buffers. These dependencies can be direct, resulting from a message in one restricted buffer waiting immediately for another restricted buffer; or they can be indirect, caused by a message which occupies a restricted buffer followed by one or more unrestricted buffers and waits for another restricted buffer. Thus, a message in a cycle occupies at least one restricted buffer and waits for another restricted buffer. Furthermore, the waiting dependencies between two restricted buffers do not have to be in the same or a neighboring node.

Another complication of the longer messages, is that the lowest dimension in a cycle in the BWG is no longer guaranteed to be the lowest dimension in the original

cycle in the network. Thus, we need to refer to the network cycle to reason about the routing decisions made.

Lemma 6: Given a cycle in the network corresponding to a cycle in the BWG where l is the lowest dimension buffer in the network cycle, the lowest possible dimension any message needed to correct when it was routed to a buffer it holds in the cycle is l.

Proof: Let m be a message in a network cycle corresponding to a cycle in the BWG. Also let l' be the lowest dimension message m needed to correct when it was routed to a buffer b it holds in the cycle. Furthermore, suppose l' is less than the lowest dimension l in the cycle. There are two possibilities. Either m corrected dimension l' sometime after acquiring buffer b, or m still needs to correct dimension l' and waits on a buffer in dimension l' in the cycle (a message must wait on the lowest dimension it needs to correct). Both, however, contradict the assumption that l is the lowest dimension buffer in the cycle. (Nevertheless, a message blocked in the cycle may have a tail that extends beyond the cycle and holds a buffer in a dimension lower than l.) □

Lemma 7: Consider a cycle in the network corresponding to a cycle in the BWG. Let l be the lowest dimension in the network cycle. Then all buffers in the network cycle in dimension l have been used minimally according to *DO* or by using unrestricted buffers in dimension l.

Proof: Consider a cycle in the network corresponding to a cycle in the BWG. Let l be the lowest dimension in the network cycle. By Lemma 6, l is the lowest possible dimension any message needed to correct when it occupied any buffer it currently holds in the cycle. A message can only deroute in a dimension greater than the lowest dimension that it needs to correct, which for messages in the cycle is at least as great as l. Thus, a message in the cycle was routed minimally when obtaining its buffers in the cycle in dimension l; and by definition of the routing algorithm, the buffers were chosen according to *DO* rules or by using unrestricted buffers in dimension l. □

Definition: A positive *turn* buffer is a buffer b_l^+ in the positive direction of dimension l in a cycle which resides in a node which has the smallest[‡] position or offset in dimension l of any node in the cycle.

Lemma 8: There are no cycles in the BWG where l, the lowest dimension in the corresponding network cycle is used in both the positive and negative direction.

Proof: Assume there is a cycle in the BWG. Let l be the lowest dimension in the corresponding network cycle, and suppose l is used in both the positive and negative directions. Then, there exists a message m which holds a buffer in the cycle that is not in the positive direction of l and later waits for or uses a positive turn buffer b_l^+ in the cycle, We show that message m violates the routing rules, and hence no such cycle exists. There are three cases.

First message m holds a restricted buffer b_h in the cycle, in some dimension h, $h > l$ and later waits for or uses and holds positive turn buffer b_l^+ in the positive direction of dimension l. By Fact 1, this direction of l is minimal for m. By Lemma 6, l is the lowest dimension m needed to correct when m used buffer b_h. But m is not allowed to use a restricted b_h buffer in a dimension greater than l, when it needs to route in the positive direction of l.

Second, message m holds a (restricted or unrestricted) b_l^- buffer in the cycle in the negative direction of dimension l and later waits for or uses and holds positive turn buffer b_l^+ in the cycle. By Lemma 7, message m was routed minimally to all buffers it holds in the cycle in dimension l. If m holds b_l^- and b_l^+, this is a contradiction since m cannot be routed minimally by being routed first in the negative, and then in the positive direction. If m holds b_l^- and waits for b_l^+, Fact 1 says that m waits in the minimal direction; and again, this is a contradiction.

Third, message m does not satisfy either of the above cases. There are two possibilities. If m waits for positive turn buffer b_l^+, all the buffers m holds are unrestricted buffers in dimensions greater than l, and m is not part of the cycle. If m holds positive turn buffer b_l^+, all the buffers m holds after positive turn buffer b_l^+ are unrestricted

[‡]Without loss of generality, we assume the nodes are ordered in row major order.

buffers in dimensions greater than l. Hence, there must be another message in the cycle that waits on positive turn buffer b_l^+. □

Theorem 9: The Triplex routing algorithm for the mesh is deadlock-free.

Proof: The algorithm is wait-connected since a message can always wait on the waiting buffer specified by *DO*, which is wait-connected.

A cycle must use both the positive and negative directions of each dimension when routing on a mesh. Thus, it follows from Lemma 8 and Theorem 1 that the BWG is acyclic, and the algorithm is deadlock-free. □

A.1.2 The Torus Algorithm

The proof for the torus is similar to the mesh but requires three additional lemmas to show that cycles are not created by routing on the wrap edges. Since a message that needs to correct the lowest dimension l in the negative direction has more freedom than one that needs the positive direction of l, we need to consider two separate cases, one where dimension l is used only in the negative direction, and the other where l is used in the positive direction. We start with the base case, which considers a cycle in a single dimension l and follow with the more general case. Again, the packet algorithm is presented first.

Packet Triplex on the Torus

Lemma 10: There are no single dimension cycles in the BWG.

Proof: Assume there is a cycle in the BWG that only uses buffers in dimension l. By Lemma 3, all messages in input buffers in the cycle in dimension l were routed minimally by *DO* rules using restricted buffers. *DO* is suffix-closed, and independent of the current input buffer. So given a message m in an input buffer in the cycle, there exists some packet that always follows *DO* routing which could reside in m's buffer and wait for the same buffer as message m in the cycle. The output buffers in the cycle can be filled identically. Thus in this case, the edges or waiting dependencies in the BWG are equivalent to those of *DO* which is acyclic. □

Lemma 11: On the torus, there are no multi-dimensional cycles in the BWG which use the lowest dimension in the cycle in the positive direction only.

Proof: Assume there is a multi-dimensional cycle in the BWG which uses l, the lowest dimension in the cycle in the positive direction only. Then, there exists a message m that holds a restricted input buffer b_h in the cycle, in some dimension h, $h > l$ and waits for an output buffer b_l^+ in the positive direction of dimension l. By Fact 1, this direction in dimension l is minimal for m. By Lemma 2, l is the lowest dimension m needed to correct when m used buffer b_h. But m is not allowed to use restricted buffer b_h in a dimension greater than l, when it needs to route in the positive direction of l. □

Lemma 12: On the torus, there are no multi-dimensional cycles in the BWG where the lowest dimension in the cycle is used in the negative direction only.

Proof: Assume there is a multi-dimensional cycle in the BWG where l, the lowest dimension in the cycle, is used in the negative direction only. Since dimension l is used in a single direction, the cycle must use a wrap buffer w in dimension l. Also, there exists a restricted input buffer b_h in some dimension h, $h > l$, in the cycle immediately before the wrap buffer w, but excluding buffers in dimension l. Let m be the message that holds input buffer b_h. Since a message in a cycle occupies a single restricted buffer, m waits on a restricted output buffer in dimension l at or before the wrap buffer w. But m is not allowed to use a restricted buffer in a dimension greater than l unless m is guaranteed to have a minimal path to its destination position in dimension l, where the path consists of waiting buffers which never have waiting dependencies on the wrap buffers in dimension l (wrap-free property). □

Theorem 13: The packet Triplex routing algorithm for the torus is deadlock-free.

Proof: The algorithm is wait-connected since a message can always wait on the waiting buffer specified by DO, which is wait-connected. The result follows from Lemmas 4, 10, 11, 12, and Theorem 1. □

Wormhole Triplex on the Torus

Lemma 14: There are no single dimension cycles in the BWG in dimension l, the lowest dimension of the corresponding cycle in the network.

Proof: Assume there is a single dimension cycle in the BWG which only uses (restricted) buffers in dimension l, the lowest dimension in the corresponding cycle in the network. If buffers in dimensions greater than l are used in the cycle, they must be unrestricted buffers.

By Lemma 7, all the buffers in the cycle in dimension l were routed minimally by DO rules or by using unrestricted buffers in dimension l. The waiting dependencies of DO are acyclic. Routing by DO_l is independent of dimensions other than l. Thus taking unrestricted buffers in dimensions other than l does not cause a cycle in dimension l in the BWG. These routes only add the following edges. If there exists a route from node a to node b in dimension l, using an unrestricted buffer in a dimension i not equal to l results in an edges in the BWG from node a to b', where b' is any node with the same positions as b in each dimension, except dimension i.

Furthermore, since DO is suffix-closed and independent of the current buffer (and whether it's restricted or unrestricted), routing a message minimally in dimension l either by DO or unrestricted buffers does not alter the subsequent route or waiting buffer used by the message in dimension l, as specified by DO rules. So routing in unrestricted buffers in dimension l only adds new dependencies in the BWG from an unrestricted buffer in dimension l to restricted waiting buffers specified by DO. There are no edges from restricted buffers to unrestricted buffers. Thus, there can be no cycle in dimension l. □

Lemma 15: On the torus, there are no cycles in the BWG which contain a dimension greater than l, the lowest dimension in the corresponding cycle in the network, provided that l is used in the cycle in the positive direction only.

Proof: Assume there is a cycle in the BWG containing a restricted buffer in some dimension h, $h > l$, where l is the lowest dimension in the corresponding cycle in the network. Also suppose l is used in the network cycle in the positive direction only.

Then, there exists a message m that holds a restricted buffer b_h in the cycle, and later waits for or uses and holds a buffer b_l^+ in the positive direction of dimension l. By Fact 1, this direction in dimension l is minimal for m. By Lemma 6, l is the lowest dimension m needed to correct when m used buffer b_h. But m is not allowed to use a restricted b_h buffer in a dimension greater than l, when it needs to route in the positive direction of l. □

Lemma 16: On the torus, there are no cycles in the BWG which contain a dimension greater than l, the lowest dimension in the corresponding cycle, provided that l is used in the negative direction only.

Proof: Assume there is a cycle in the BWG containing a restricted buffer in some dimension h, $h > l$, where l is the lowest dimension in the corresponding cycle in the network. Also suppose l is used in the negative direction only. Since dimension l is used in a single direction, the cycle must use a wrap buffer w in dimension l. Let b_h be a restricted buffer in dimension h, in the cycle immediately before the wrap buffer w, but excluding unrestricted buffers and buffers in dimension l. Let m be the message that holds buffer b_h. Message m cannot wait for an unrestricted buffer, so one of the following describes m. Message m waits on a restricted buffer in dimension l at or before the wrap buffer w. Alternatively, m uses the wrap buffer w in dimension l and waits on a restricted buffer in the cycle after the wrap edge. Both are impossible. First, m is not allowed to use a restricted buffer in a dimension greater than l unless m is guaranteed to have a minimal path to its destination position in dimension l, where the path consists of waiting buffers which never have waiting dependencies on the wrap buffers in dimension l (wrap-free property). Second as a consequence of the wrap-free property, m cannot use restricted buffers in a dimension greater than l, if it needs to use a wrap buffer in dimension l. □

Theorem 17: The Triplex routing algorithm for the torus is deadlock-free.

Proof: The algorithm is wait-connected since a message can always wait on the waiting buffer specified by DO, which is wait-connected. The result follows from Lemmas 8, 14, 15, 16, and Theorem 1. □

It is not necessary to restrict a message to wait on the buffer specified by DO. A message may actually wait on all the buffers it needs. To see this, we define a subgraph BWG' of the BWG for each algorithm as follows. Let BWG' be the subgraph of the BWG obtained by removing all the edges from the restricted buffers to the unrestricted buffers. Furthermore, remove all edges between restricted buffers that violate DO routing. The resulting buffer waiting graph BWG' is still wait-connected. The proof is similar to those presented, except that the following definition and theorem are needed instead. A *True Cycle* is a cycle in the BWG which can be created without the simultaneous use of any buffer.

Theorem 18: [SJ96] A routing algorithm, R, that allows a blocked message to wait for multiple output buffers is deadlock-free iff R is wait-connected for some subgraph BWG' of BWG and BWG' has no True Cycles.

A.2 Proof of Livelock-Freedom

To prove livelock-freedom for Triplex, we generalize an argument that first appeared in [KS94]. Our argument works for many networks besides the Chaos hypercube.

Theorem 19: Given a strongly connected, deadlock-free network with diameter d where the indegree equals the outdegree of each node and the maximum indegree of any node is g, if the probability a message does not deroute at every step is at least p, $0 < p \leq 1$, then the network is livelock-free.

Proof: Consider a strongly connected, deadlock-free network with diameter d where the indegree equals the outdegree of each node, the maximum indegree of any node is g, and the probability a message does not deroute at every step is at least p, $0 < p \leq 1$.

Since the network is deadock-free a message can be delayed in a node at most a finite amount of time t. Thus each message is subjected to at most t (usually substantially less) deroute decisions.

Let m be a message. Let a superstep consist of d hops of m. Let S_i be the event that m is not delivered in its ith superstep. If we consider a superstep in isolation,

as in the first superstep, $P[S_1] \leq 1 - p^{td}$, since in the worst case m may only reach its destination in d hops after avoiding deroutes for t time steps at every node on its path to its destination.

Consider the probability that m is not delivered after i supersteps.

$$\begin{aligned} P[S_i \& S_{i-1} \& \cdots \& S_1] &= P[S_i | S_{i-1} \& \cdots \& S_1] P[S_{i-1} \& \cdots \& S_1] \\ &\leq (1 - p^{td}) P[S_{i-1} \& \cdots \& S_1] \\ &\leq (1 - p^{td})^i \end{aligned}$$

Taking the limit as supersteps (and hence time) goes to infinity, shows that the probability m is not delivered goes to zero in the limit. Since

$$\lim_{i \to \infty} (1 - p^{td})^i = 0$$

m is eventually delivered and livelock does not occur. □

Corollary 20: If Triplex deroutes a message with probability p, $0 \leq p < 1$ at each step, then Triplex is livelock-free.

This is a general argument and can be used for Triplex, chaotic or deflection routers. For a chaotic router, let $p = (q-1)/q$ where q, an integer greater than one, is the multiqueue size. Chaotic routers deroute when the multi-queue is full by selecting one message at random to be derouted. The remaining $q-1$ packets are not derouted. For a deflection router, let $t = 1$ since packets move at every step. For example, if the deflection router uses a random one-pass greedy assignment, let $p = 1/g$ since a packet has, at the very least, a $1/g$ chance of getting its first choice.

Notice that this is a very pessimistic analysis. Even if we assume $p \geq 1/2$, i has to be about 2^{td} before $(1 - 1/2^{td})^i$ is a constant (approximately $1/e$). It is not known whether this is the best one could do with such a general argument and topology.

A.3 Performance Comparisons

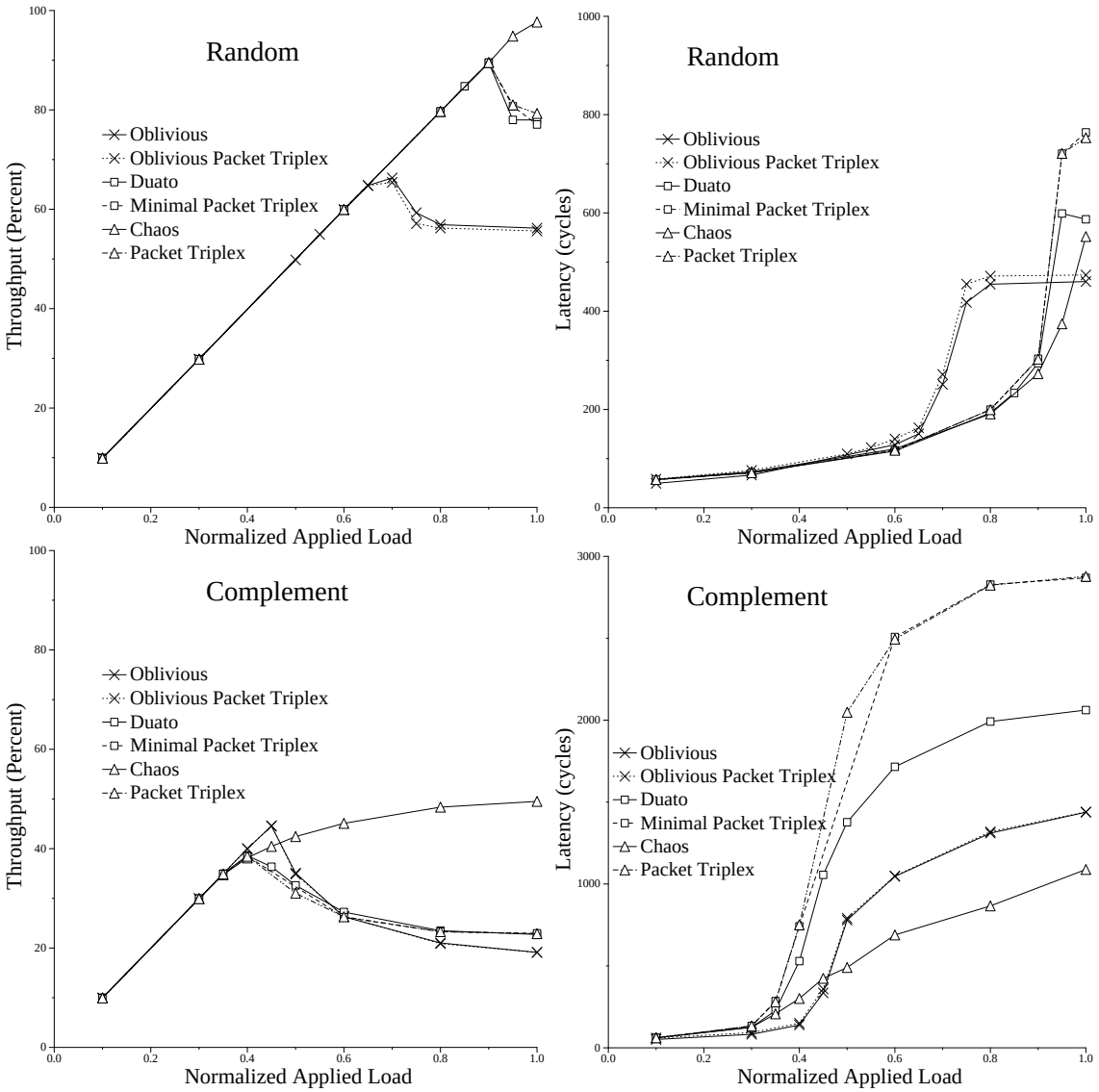

Figure A.2: Throughput and latency on 256-node 2D torus with packet routing for 20-word messages.

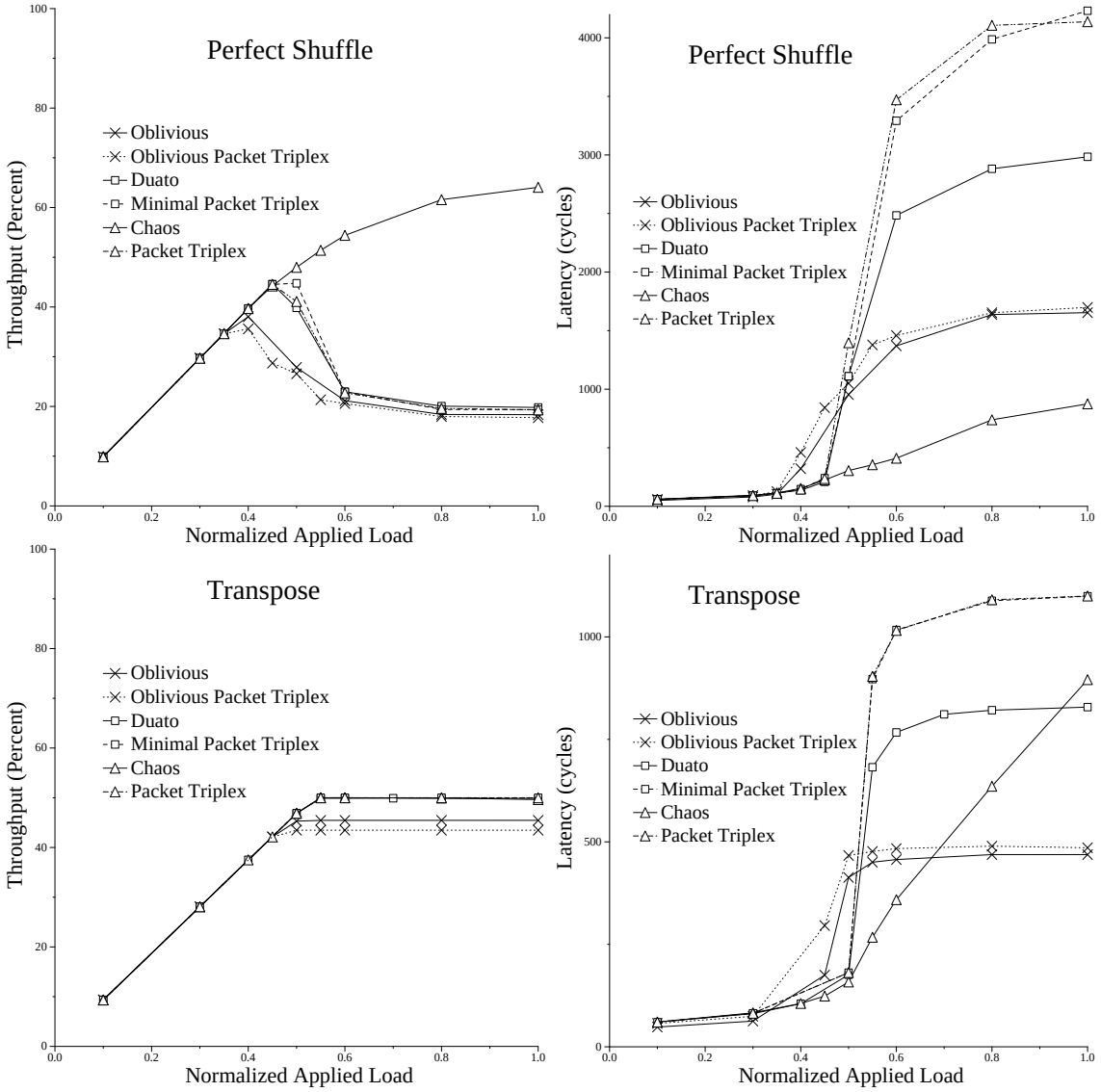

Figure A.3: Throughput and latency on a 256-node 2D torus with packet routing for 20-word messages.

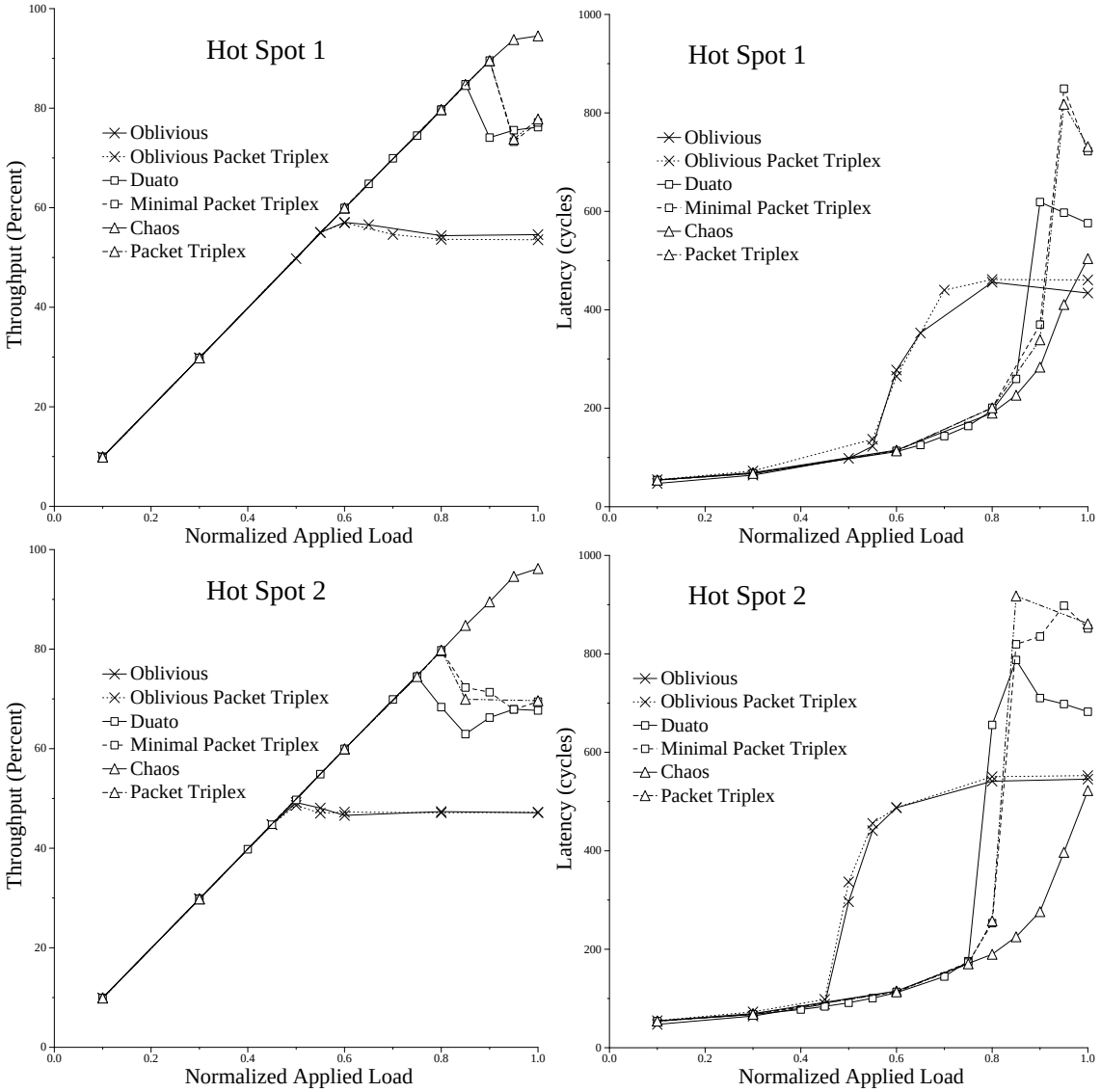

Figure A.4: Throughput and latency on a 256-node 2D torus with packet routing for 20-word messages.

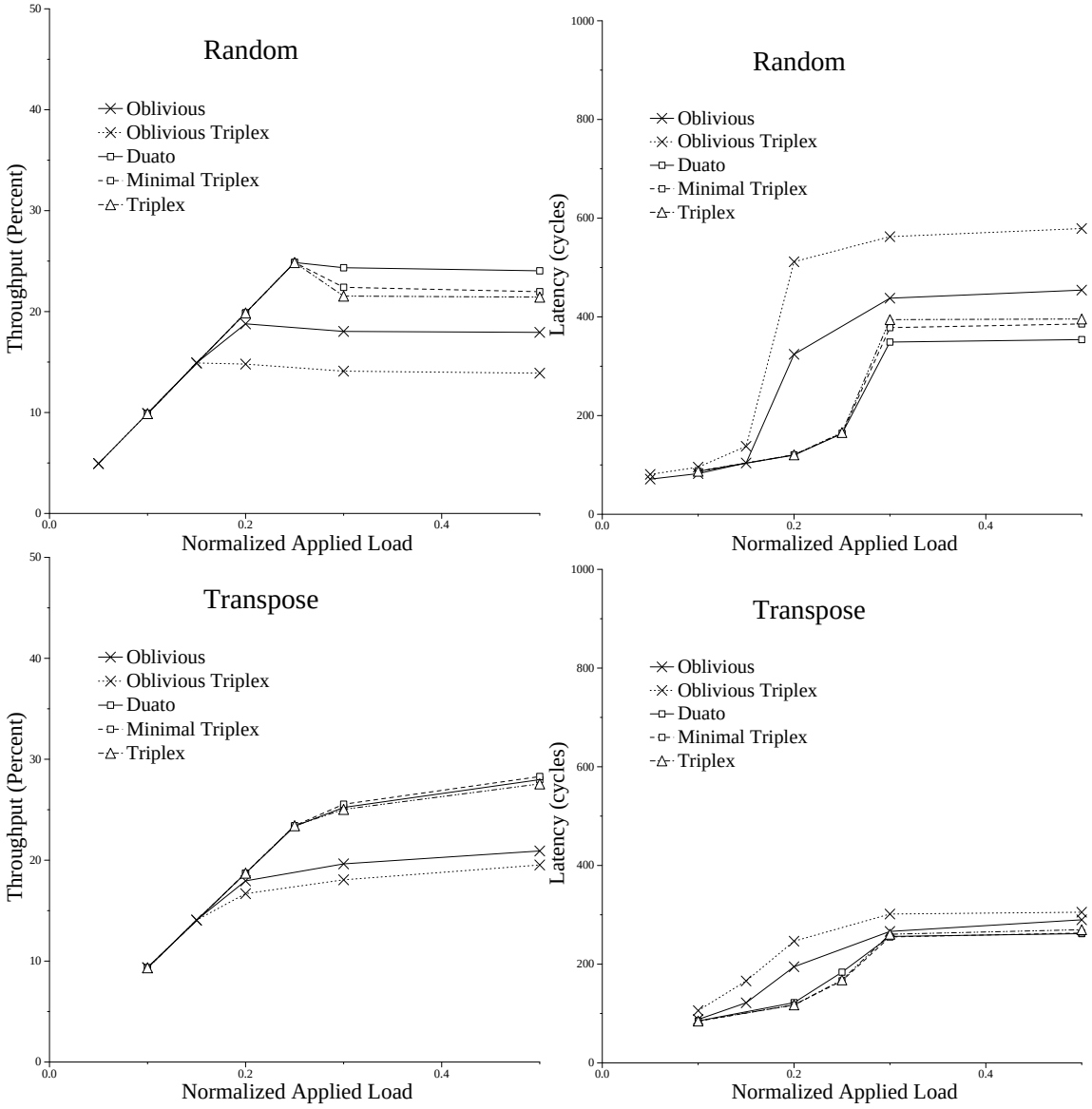

Figure A.5: Throughput and latency on a 256-node 2D torus with wormhole routing for 40-word messages.

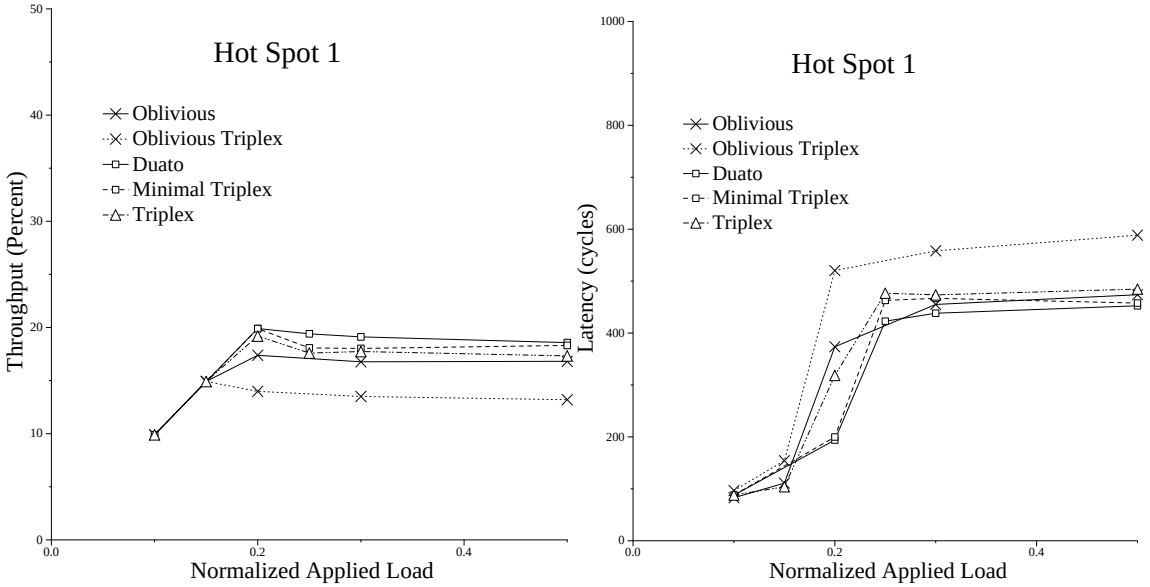

Figure A.6: Throughput and latency on a 256-node 2D torus with wormhole routing for 40-word messages.

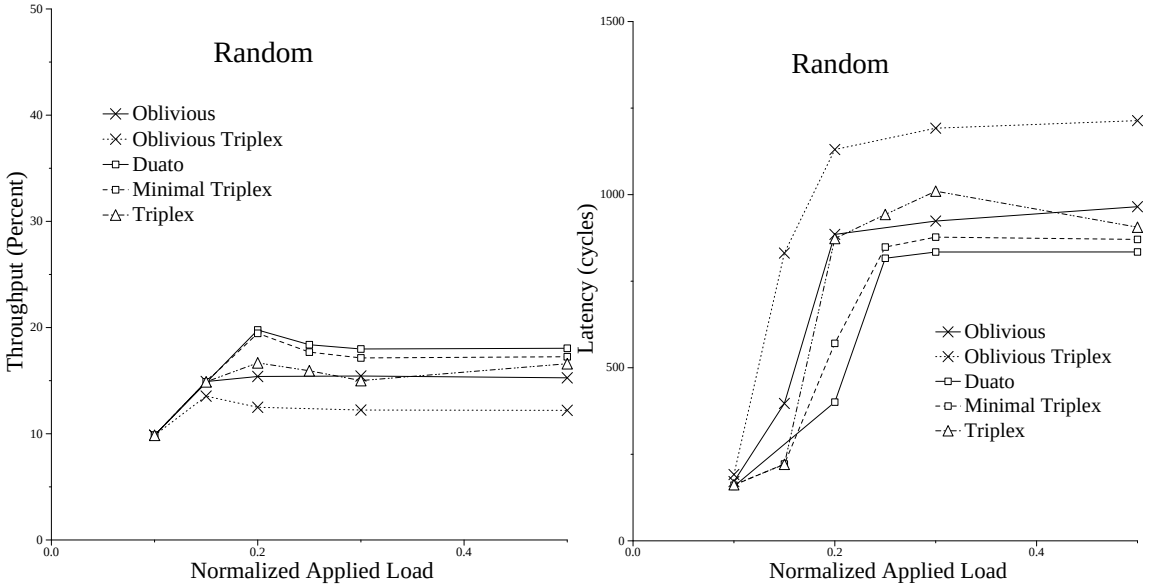

Figure A.7: Throughput and latency on a 256-node 2D torus with wormhole routing for 40-word and 400-flit messages in a 10:1 ratio.

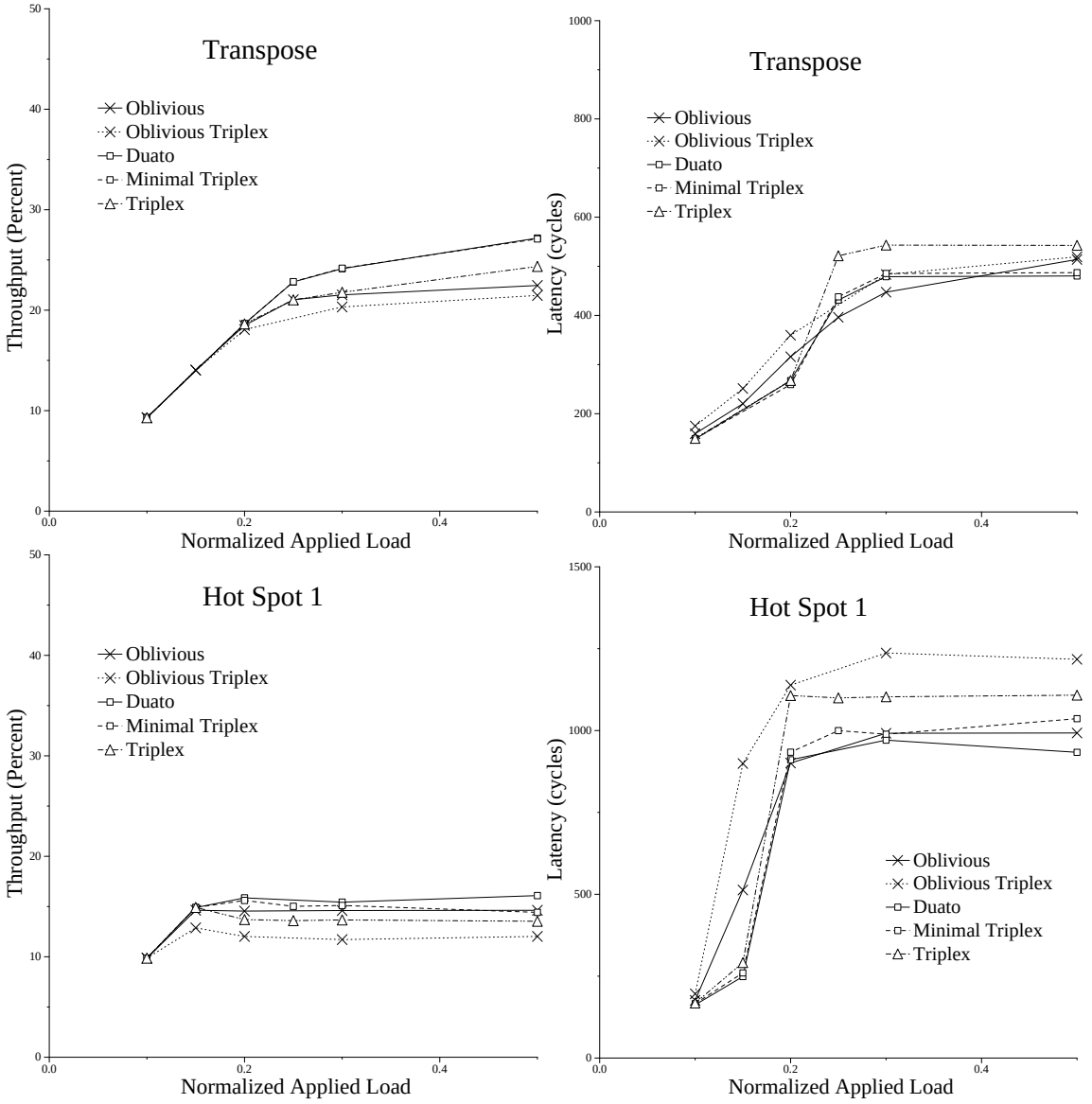

Figure A.8: Throughput and latency on a 256-node 2D torus with wormhole routing for 40-word and 400-word messages in a 10:1 ratio.

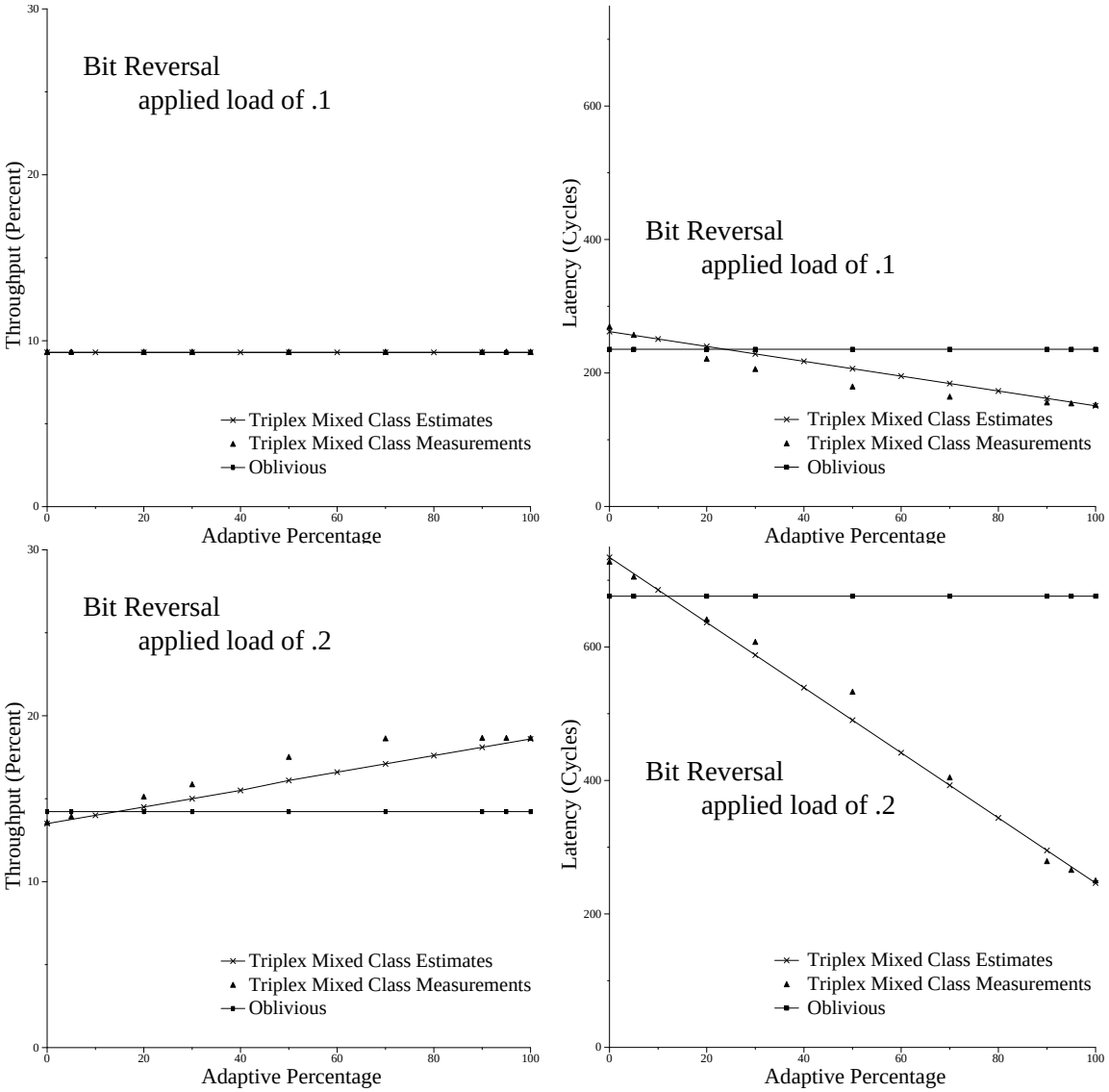

Figure A.9: Throughput and latency comparisons between oblivious and Triplex algorithms for a range of traffic mixes on a 256-node 2D torus using wormhole routing with 40-word and 400-flit messages in a 10:1 ratio.

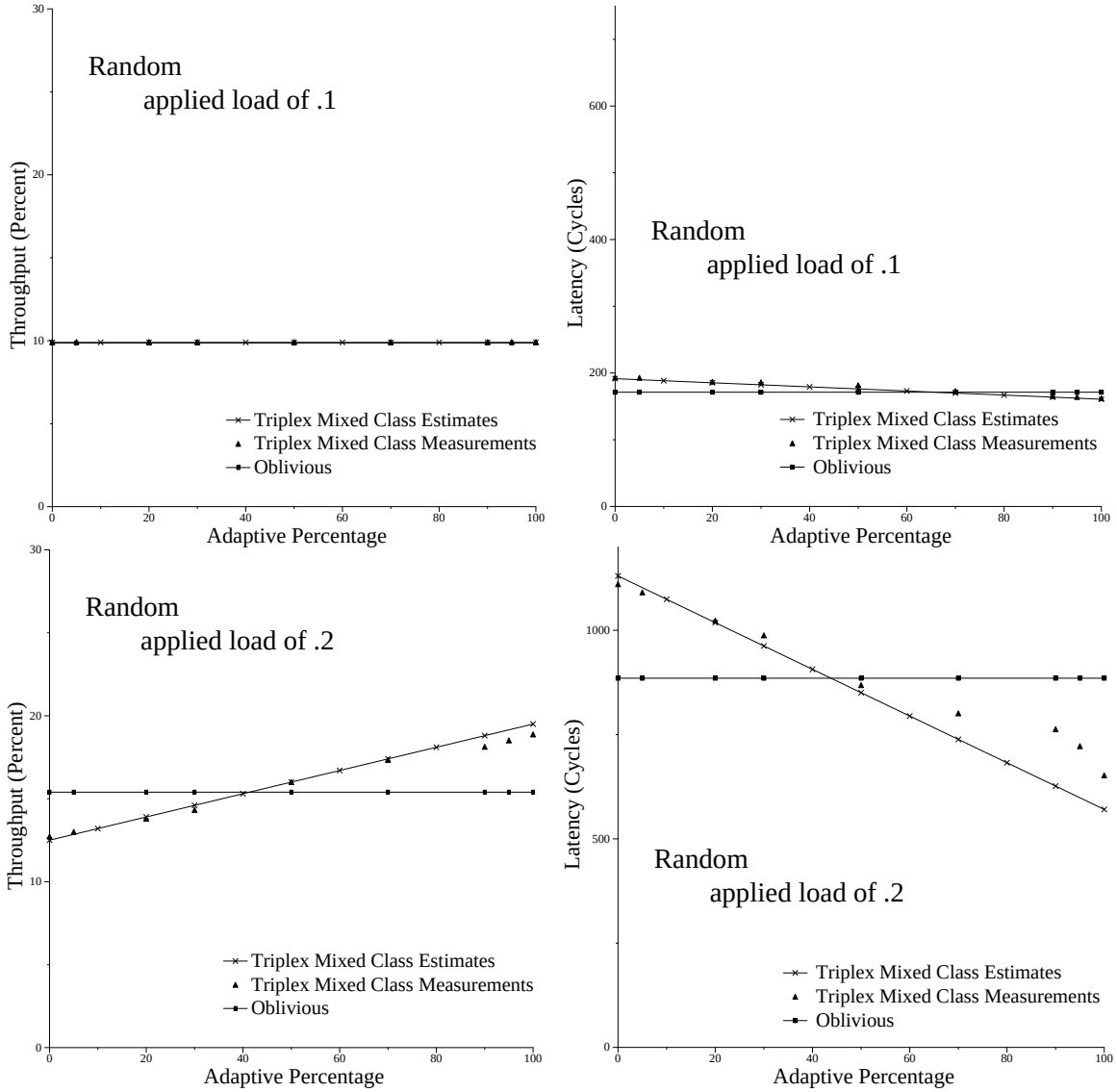

Figure A.10: Throughput and latency comparisons between oblivious and Triplex algorithms for a range of traffic mixes on a 256-node 2D torus using wormhole routing with 40-word and 400-flit messages in a 10:1 ratio.

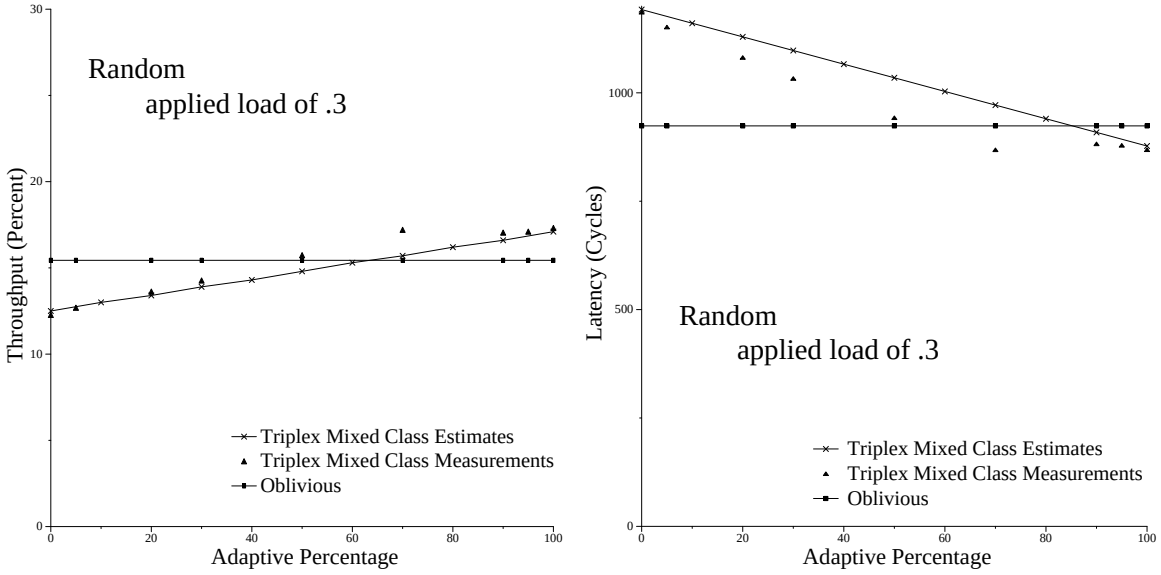

Figure A.11: Throughput and latency comparisons between oblivious and Triplex algorithms for a range of traffic mixes on a 256-node 2D torus using wormhole routing with 40-word and 400-flit messages in a 10:1 ratio.

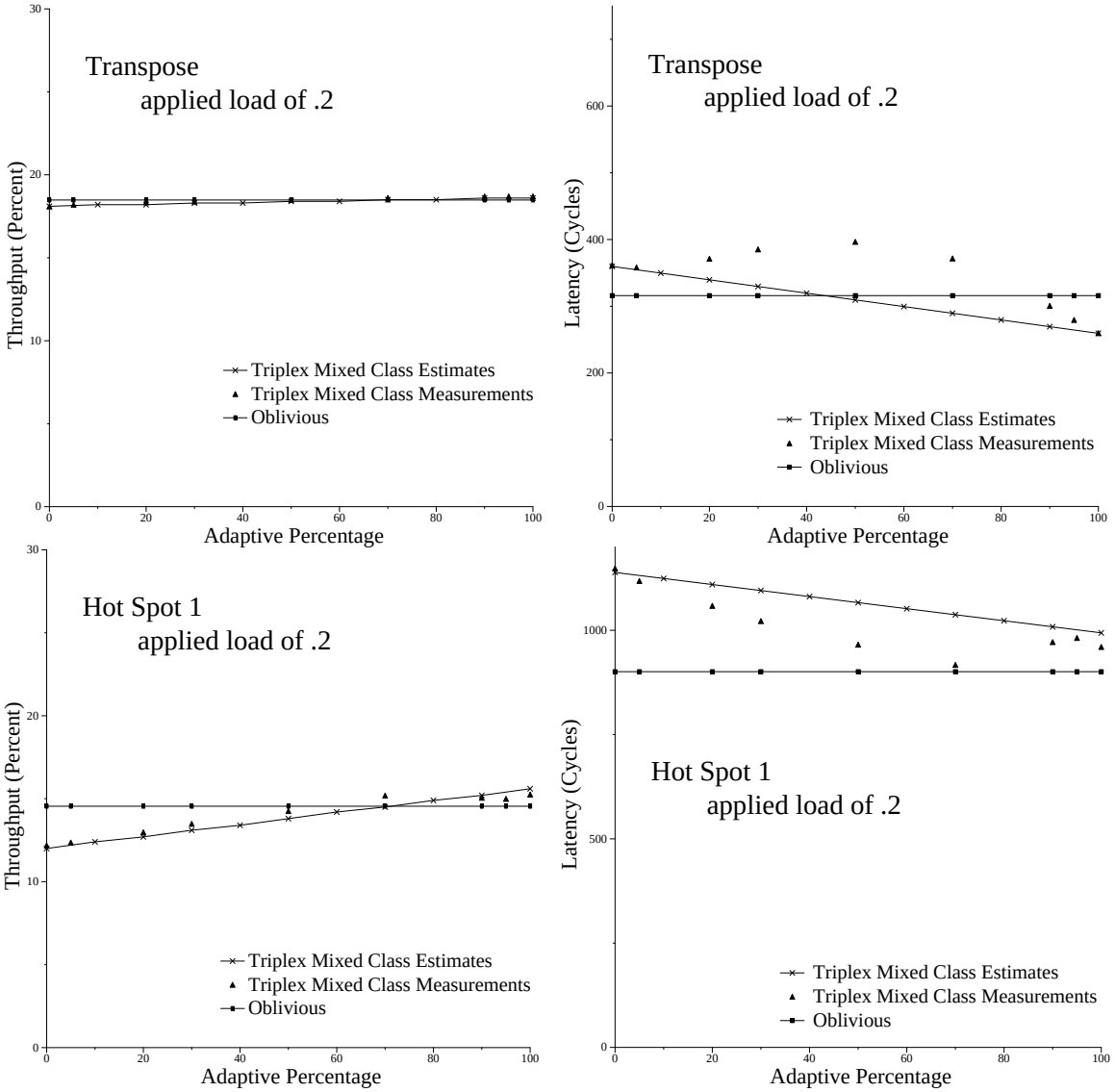

Figure A.12: Throughput and latency comparisons between oblivious and Triplex algorithms for a range of traffic mixes on a 256-node 2D torus using wormhole routing with 40-word and 400-flit messages in a 10:1 ratio.

Appendix B

INPUT AND OUTPUT DRIVEN

B.1 Peak throughput

Tables B.1–B.6 contain the peak normalized throughput, rounded to the nearest whole number, achieved by each of the routers for the various traffic patterns.

Table B.1: Shows the normalized applied load (load) at which the maximum normalized throughput (xput) is achieved by the traffic pattern. The percent error (error) is also shown.

16x16 Torus, Oblivious						
Traffic	output			input fixed or random		
	load	xput	error	load	xput	error
Random	0.80	78	0.3	0.80	77	0.4
Bit reversal	1.00	46	0.2	0.60	44	0.3
Complement	0.45	45	0.2	0.45	44	0.3
Perfect shuffle	0.45	45	0.4	0.45	45	0.3
Transpose	0.55	50	0.1	0.60	50	0.1
Hot Spot 1	0.65	63	1.1	0.65	63	1.1
Hot Spot 2	0.50	50	0.2	0.55	51	2.6

Table B.2: Shows the normalized applied load (load) at which the maximum normalized throughput (xput) is achieved by the traffic pattern. The percent error (error) is also shown.

16x16 Torus, Duato									
Traffic	output			input fixed			input random		
	load	xput	error	load	xput	error	load	xput	error
Random	0.90	89	0.2	0.80	70	1.3	0.90	90	0.2
Bit reversal	0.75	70	0.2	0.65	61	0.3	0.75	70	0.2
Complement	0.40	39	0.3	0.40	40	0.3	0.40	39	0.4
Perfect shuffle	0.45	45	0.3	0.45	45	0.6	0.45	45	0.3
Transpose	0.55	50	0.1	0.55	50	0.1	0.55	50	0.2
Hot Spot 1	0.85	85	0.4	0.80	70	2.1	0.80	80	0.2
Hot Spot 2	0.80	80	1.2	1.00	59	2.2	0.75	72	1.9

Table B.3: Shows the normalized applied load (load) at which the maximum normalized throughput (xput) is achieved by the traffic pattern. The percent error (error) is also shown.

16x16 Torus, Minimal Triplex									
Traffic	output			input fixed			input random		
	load	xput	error	load	xput	error	load	xput	error
Random	0.80	80	0.2	0.75	74	0.2	0.80	80	0.2
Bit reversal	0.70	66	0.2	0.60	56	0.3	0.65	61	0.3
Complement	0.40	36	0.6	0.40	40	0.5	0.35	34	0.5
Perfect shuffle	0.45	44	0.6	0.40	39	1.2	0.40	40	0.3
Transpose	0.55	50	0.1	0.55	50	0.2	0.55	50	0.1
Hot Spot 1	0.80	80	0.3	0.65	61	3.7	0.75	74	0.2
Hot Spot 2	0.75	75	0.5	0.55	55	2.2	0.70	70	0.2

Table B.4: Shows the normalized applied load (load) at which the maximum normalized throughput (xput) is achieved by the traffic pattern. The percent error (error) is also shown.

16x16 Mesh, Oblivious						
Traffic	output			input fixed or random		
	load	xput	error	load	xput	error
Random	0.95	93	0.4	1.00	94	0.5
Bit reversal	1.00	61	0.2	1.00	61	0.2
Complement	0.45	44	0.5	0.45	45	0.5
Perfect shuffle	0.90	86	0.7	0.90	86	0.9
Transpose	1.00	72	0.2	1.00	72	0.2
Hot Spot 1	0.80	76	0.8	0.80	77	0.9
Hot Spot 2	0.75	71	1.2	0.75	71	1.2

Table B.5: Shows the normalized applied load (load) at which the maximum normalized throughput (xput) is achieved by the traffic pattern. The percent error (error) is also shown.

16x16 Mesh, Duato									
Traffic	output			input fixed			input random		
	load	xput	error	load	xput	error	load	xput	error
Random	1.00	93	0.4	1.00	92	0.5	0.95	92	0.4
Bit reversal	0.90	77	0.3	0.80	74	0.6	0.90	75	0.4
Complement	0.40	36	0.9	0.40	40	0.5	0.35	34	0.4
Perfect shuffle	0.90	89	0.4	0.85	84	0.6	0.90	89	0.5
Transpose	0.95	81	0.3	1.00	76	0.9	1.00	78	0.2
Hot Spot 1	0.95	87	0.8	0.95	87	0.8	0.90	89	0.6
Hot Spot 2	1.00	83	0.8	0.90	83	1.0	0.85	85	0.3

Table B.6: Shows the normalized applied load (load) at which the maximum normalized throughput (xput) is achieved by the traffic pattern. The percent error (error) is also shown.

16x16 Mesh, Minimal Triplex									
Traffic	output			input fixed			input random		
	load	xput	error	load	xput	error	load	xput	error
Random	0.90	89	0.3	0.85	85	0.3	0.90	88	0.8
Bit reversal	0.70	65	0.4	0.65	61	0.4	0.70	65	0.7
Complement	0.35	34	0.5	0.40	40	0.5	0.35	32	2.8
Perfect shuffle	0.90	85	1.3	0.75	75	0.3	0.85	84	0.3
Transpose	1.00	82	0.2	0.90	76	0.6	1.00	78	0.2
Hot Spot 1	0.80	80	0.3	0.75	72	3.4	0.80	80	1.7
Hot Spot 2	0.80	80	0.3	0.75	71	2.7	0.80	80	0.3

B.2 Performance Comparisons

Figures contain the throughput and latency graphs of the three routers for both input and output driven modes for six traffic patterns. The first set of graphs compare output driven routers to input driven routers with a fixed order output buffer selection policy. The second set of graphs compare output driven routers to input driven routers with a random output buffer selection policy. The last set of graphs directly compare the performance of the input driven routers with fixed versus random selection.

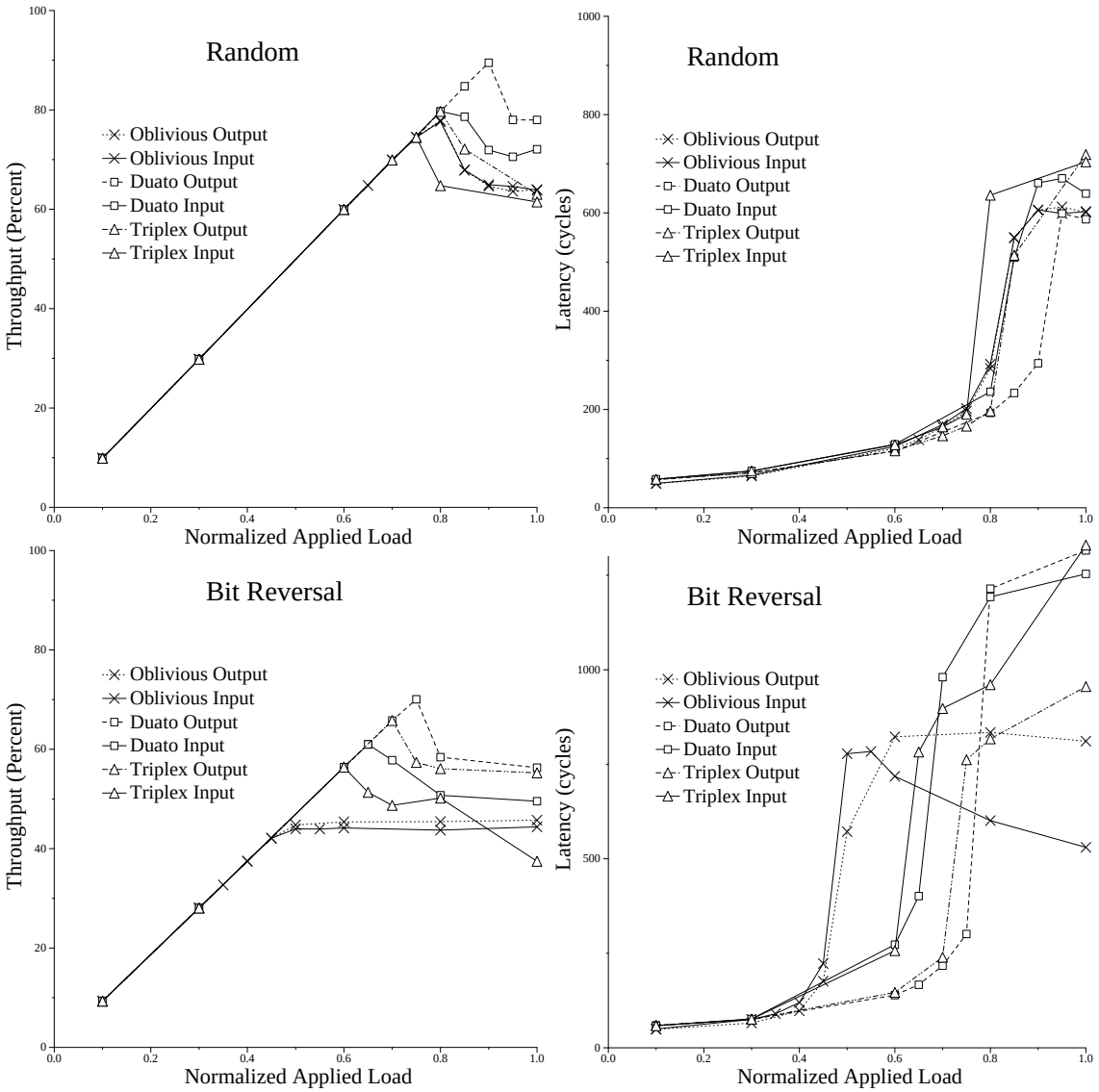

Figure B.1: Throughput and latency on a 256-node 2D torus with fixed output buffer selection.

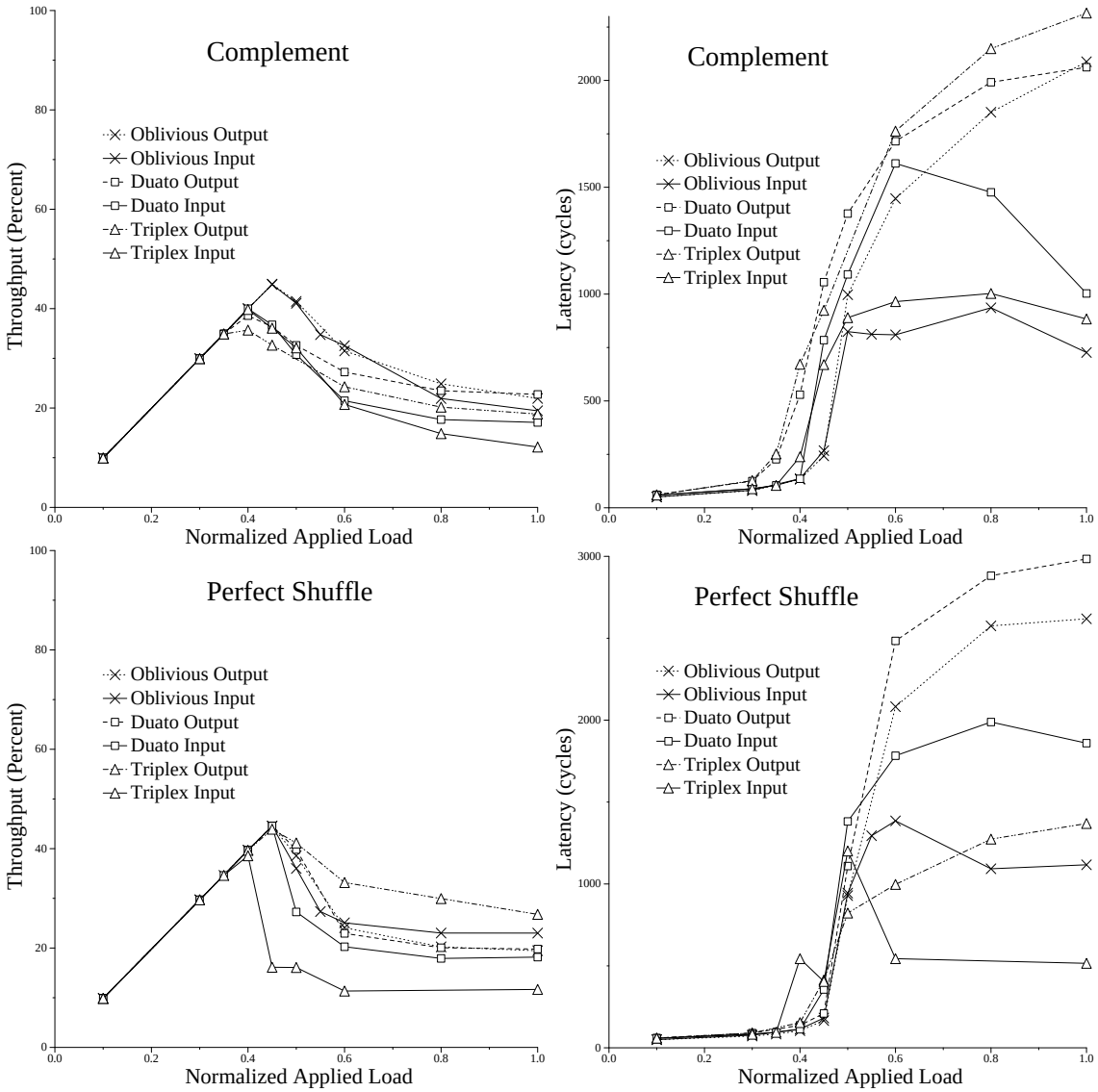

Figure B.2: Throughput and latency on a 256-node 2D torus with fixed output buffer selection.

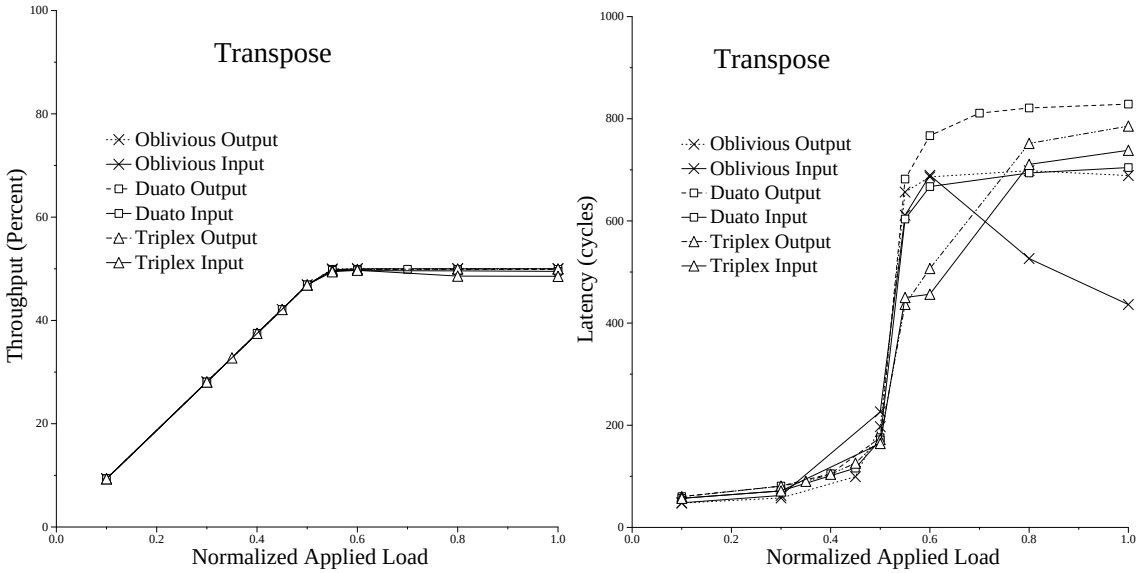

Figure B.3: Throughput and latency on a 256-node 2D torus with fixed output buffer selection.

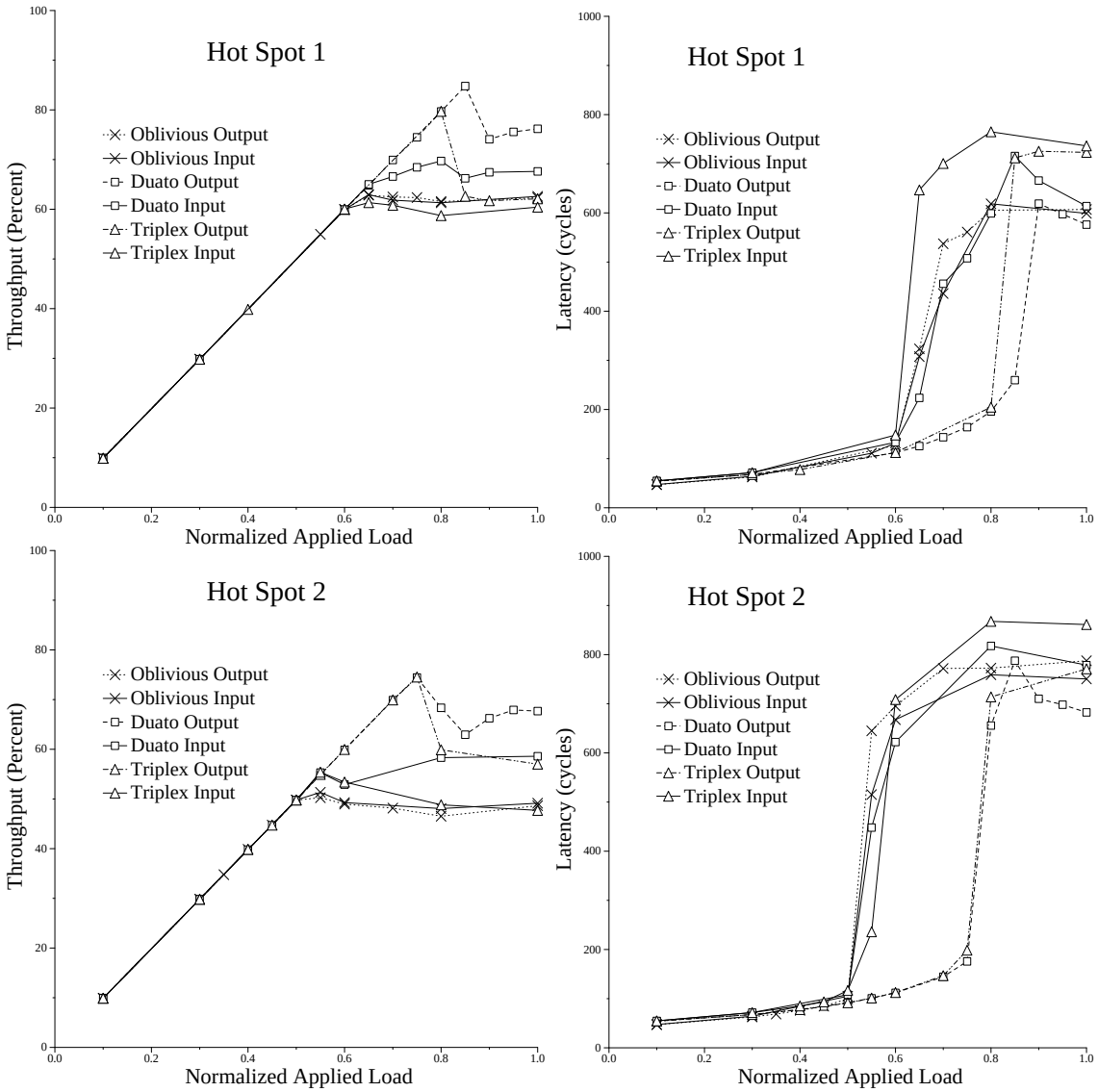

Figure B.4: Throughput and latency on a 256-node 2D torus with fixed output buffer selection.

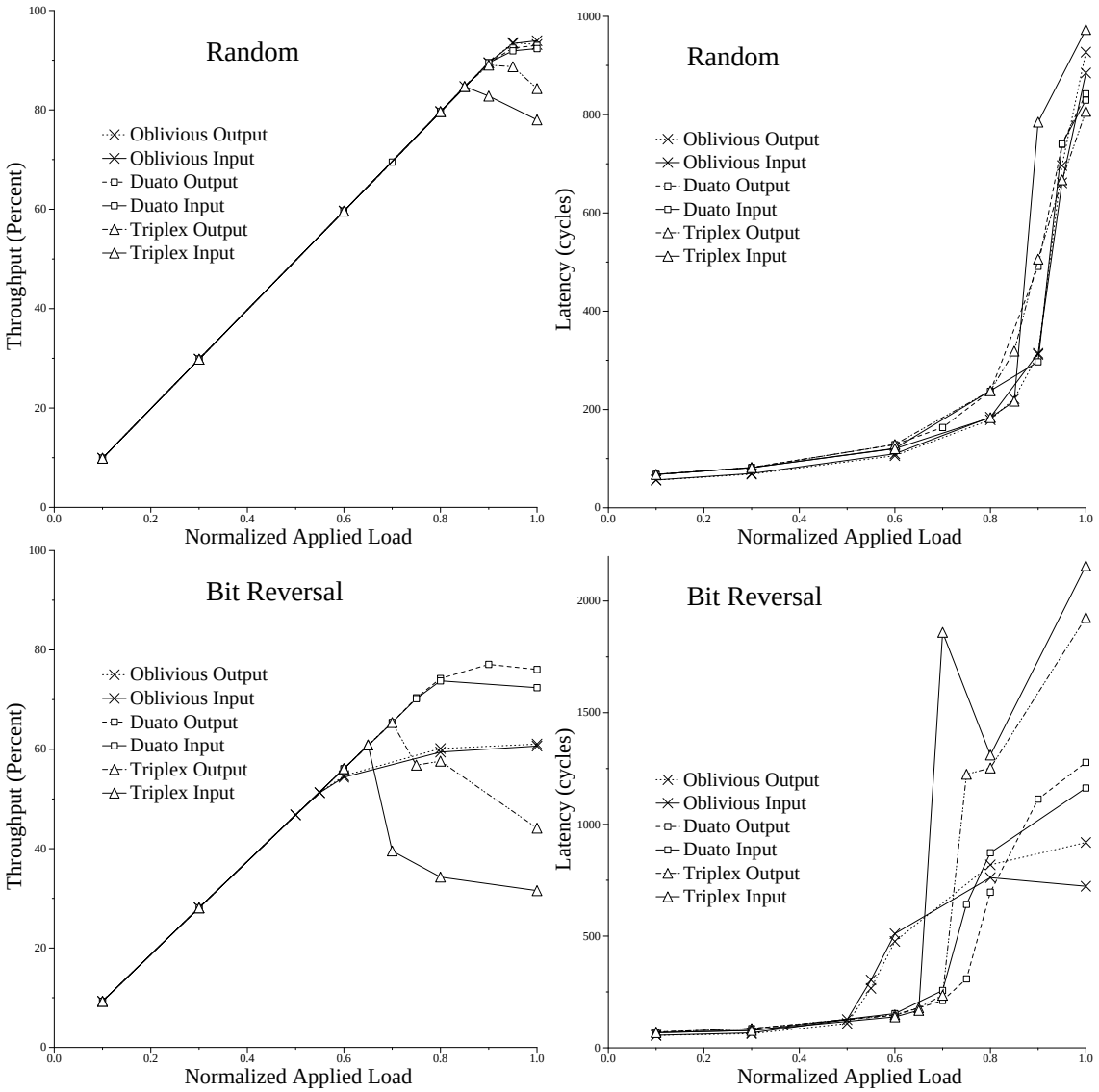

Figure B.5: Throughput and latency on a 256-node 2D mesh with fixed output buffer selection.

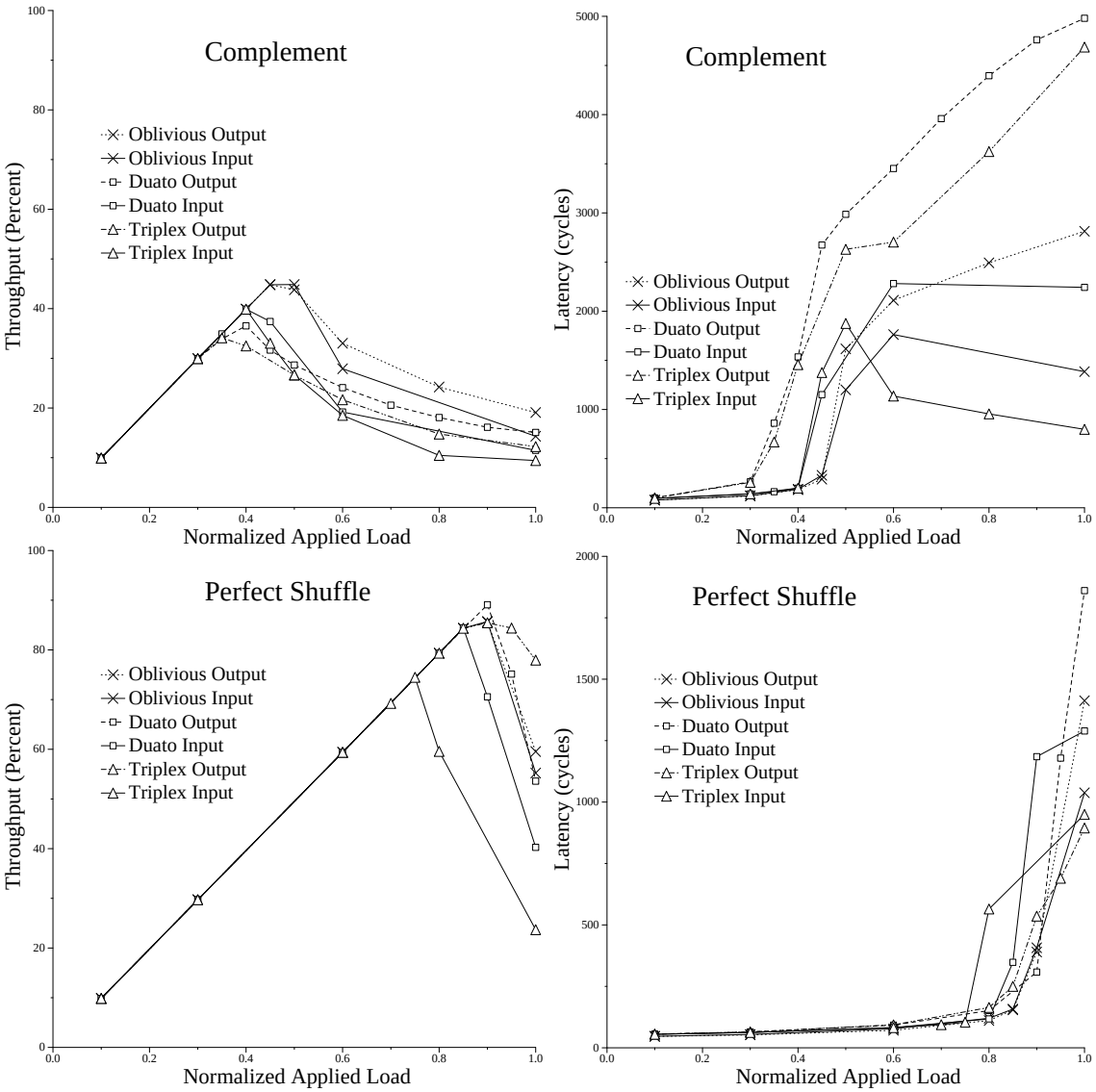

Figure B.6: Throughput and latency on a 256-node 2D mesh with fixed output buffer selection.

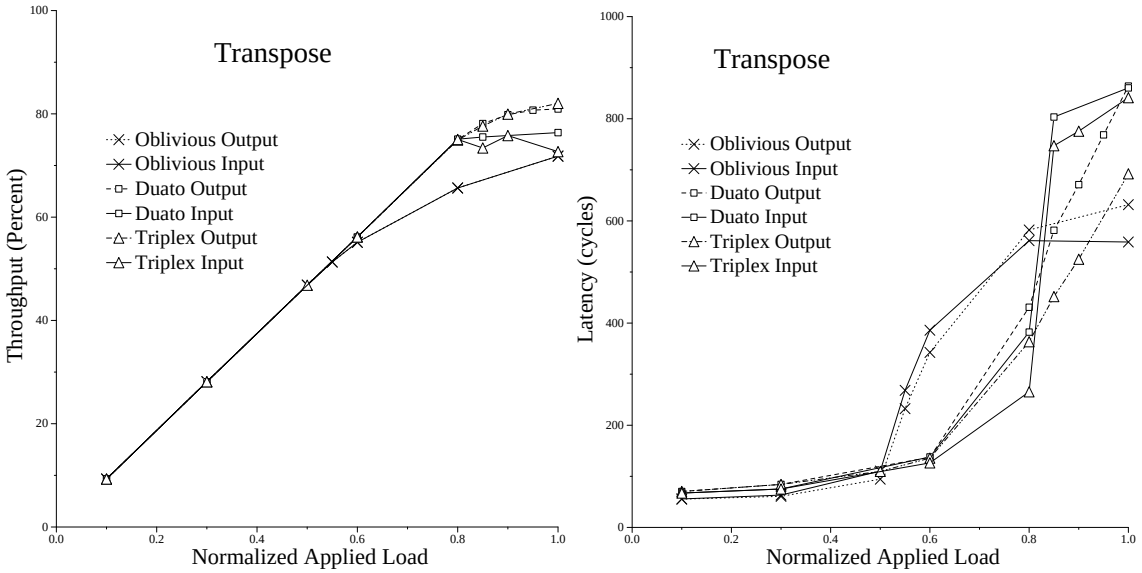

Figure B.7: Throughput and latency on a 256-node 2D mesh with fixed output buffer selection.

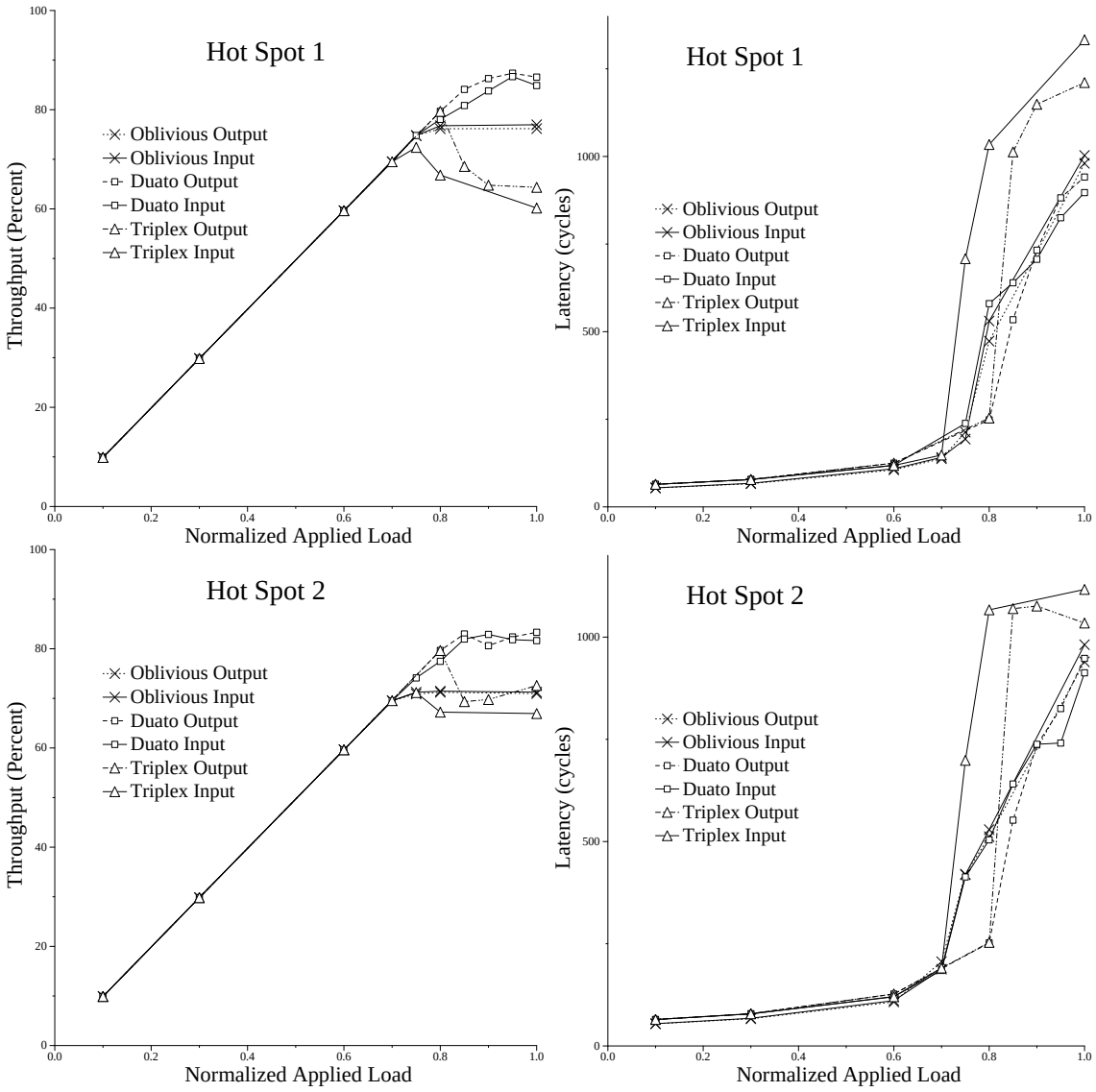

Figure B.8: Throughput and latency on a 256-node 2D mesh with fixed output buffer selection.

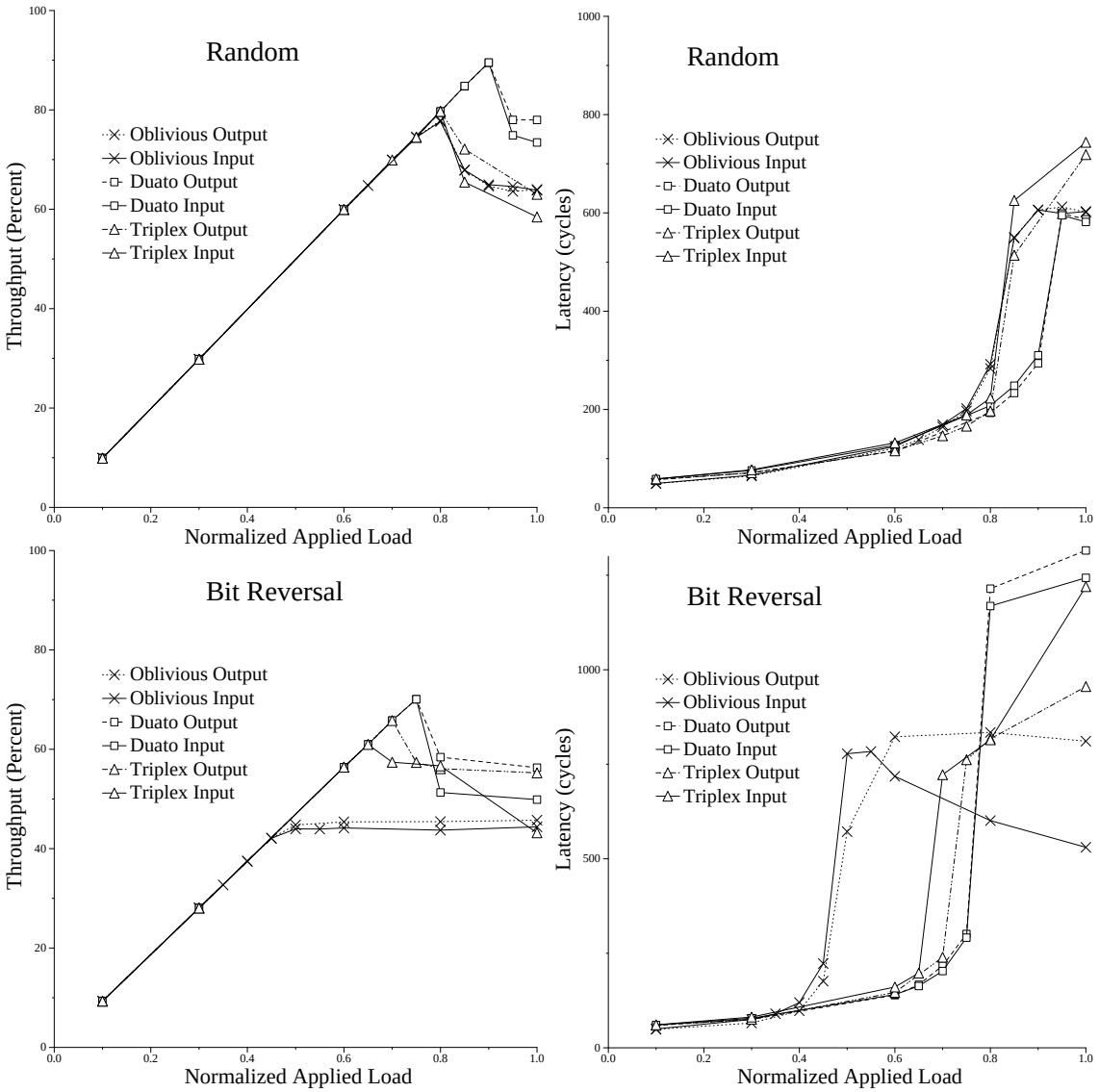

Figure B.9: Throughput and latency on a 256-node torus with random output buffer selection.

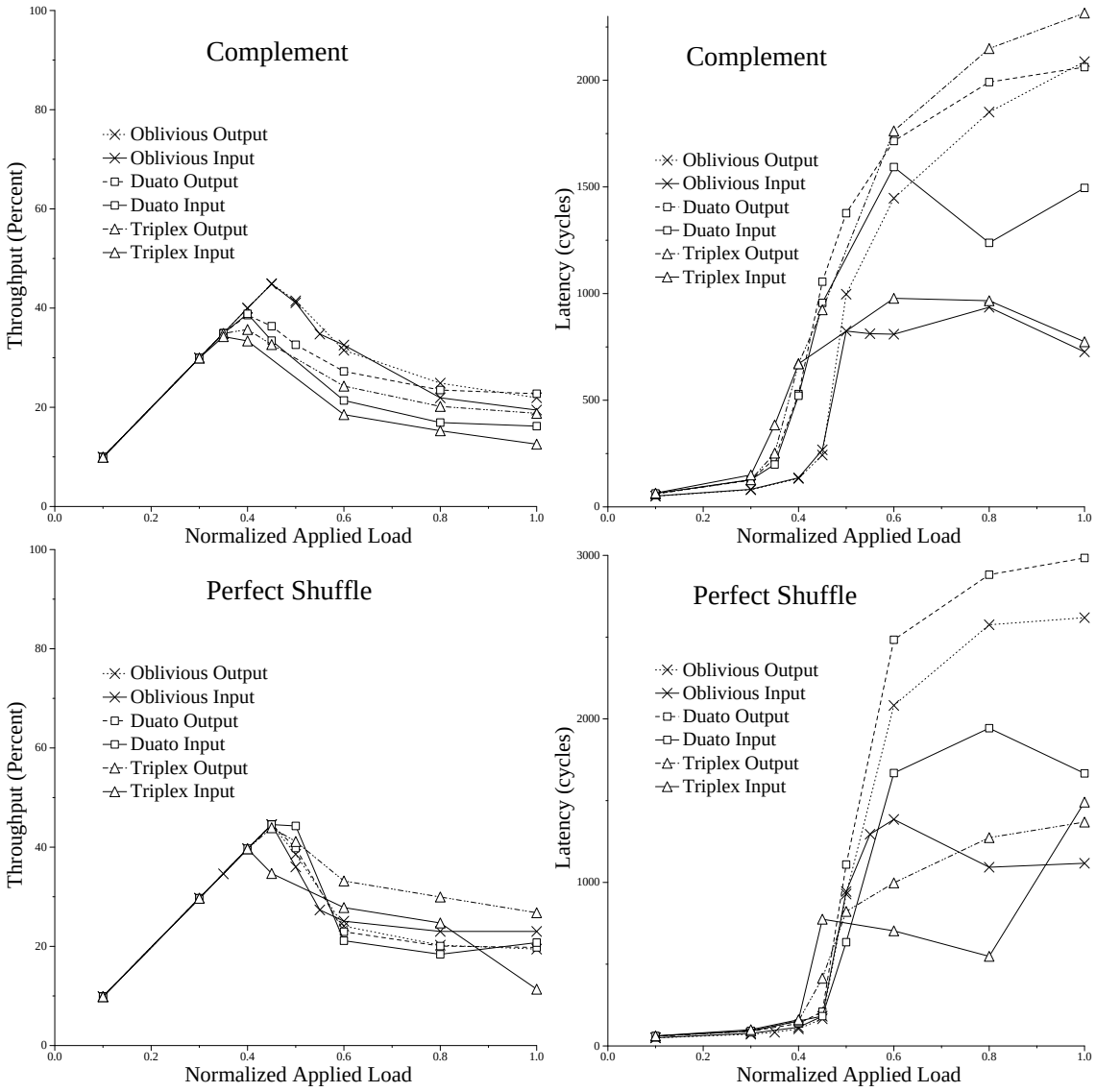

Figure B.10: Throughput and latency on a 256-node torus with random output buffer selection.

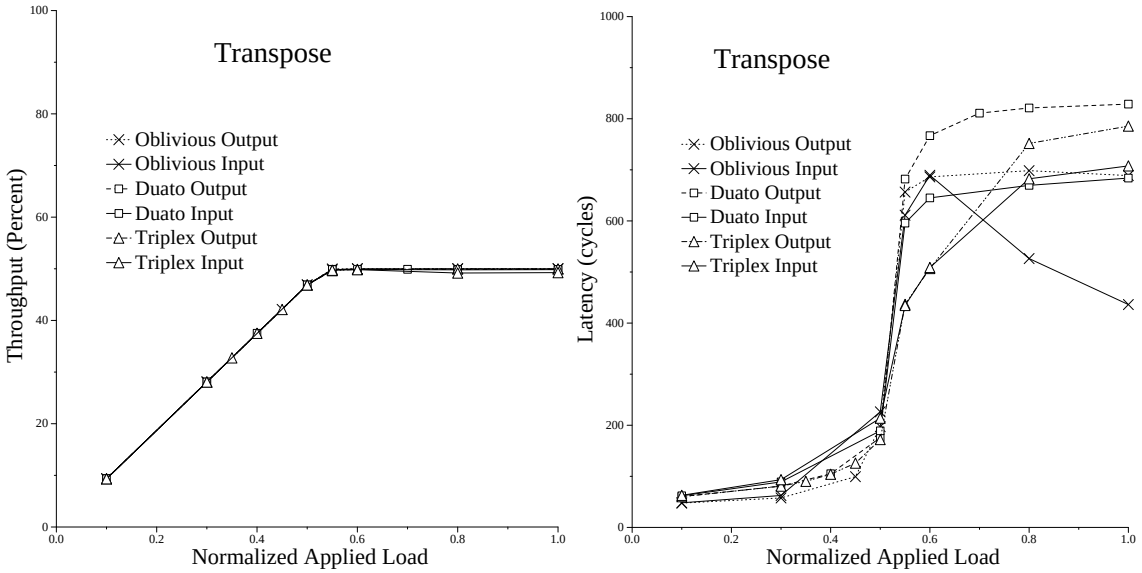

Figure B.11: Throughput and latency on a 256-node torus with random output buffer selection.

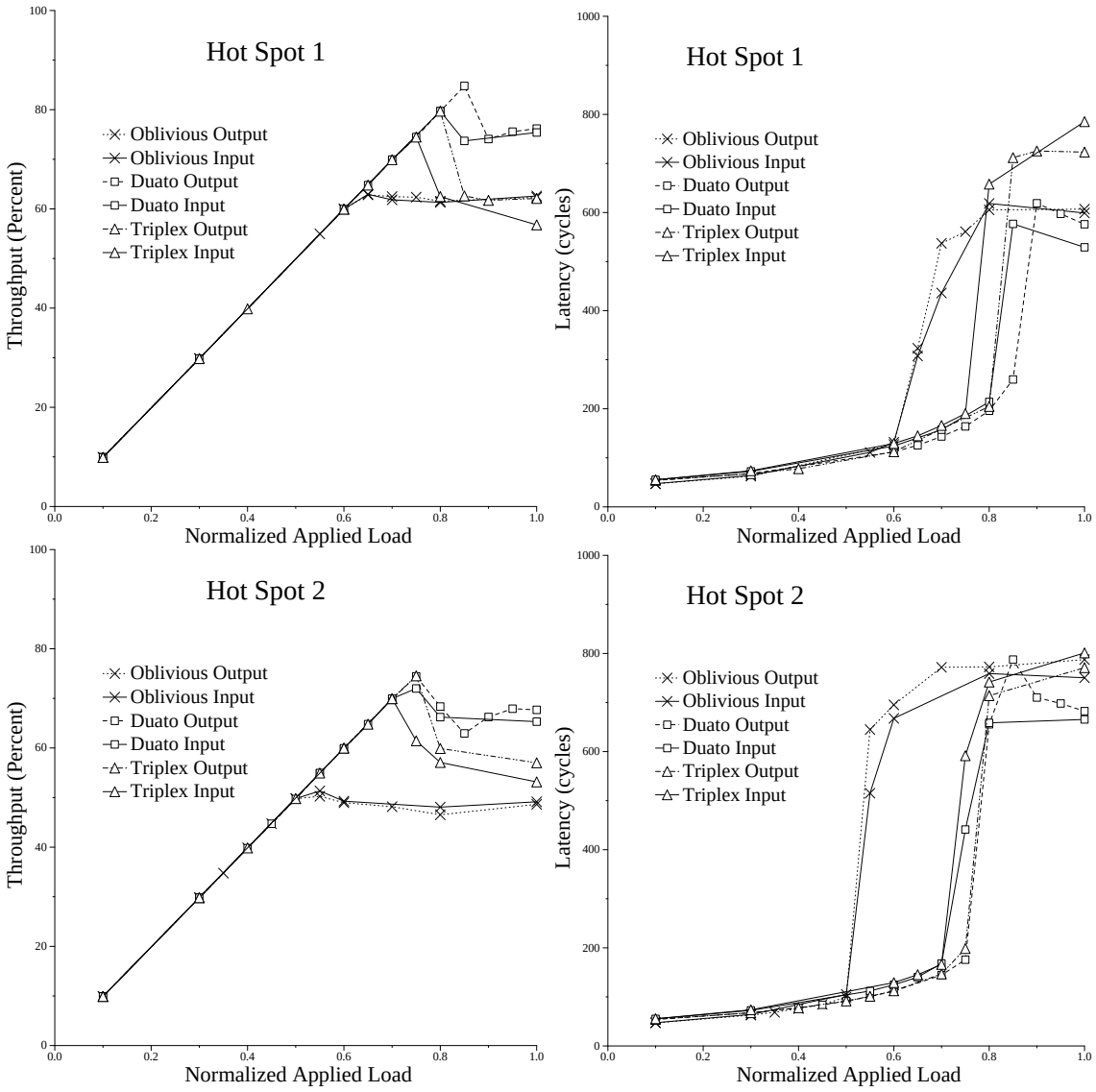

Figure B.12: Throughput and latency on a 256-node torus with random output buffer selection.

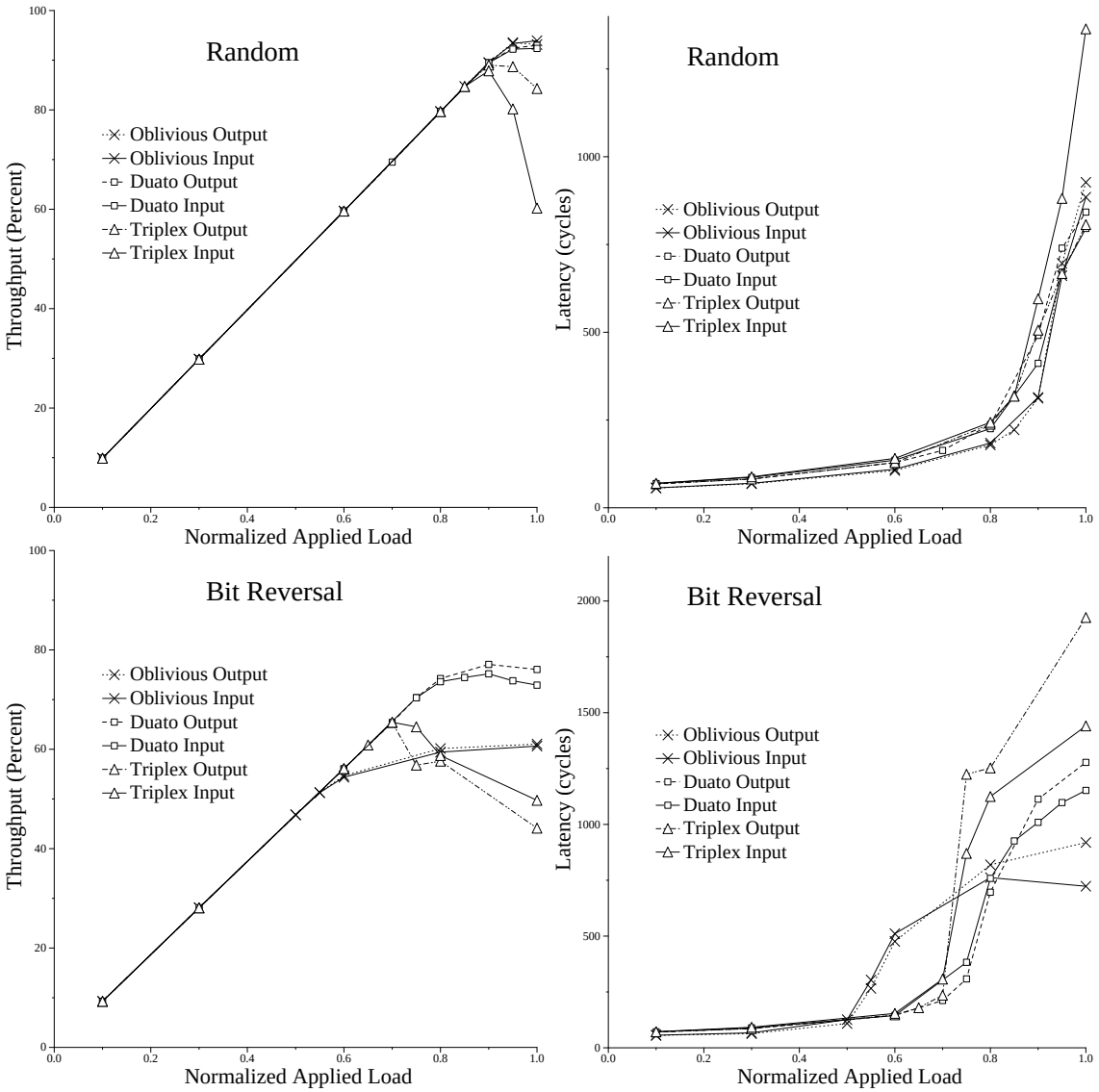

Figure B.13: Throughput and latency on a 256-node mesh with random output buffer selection.

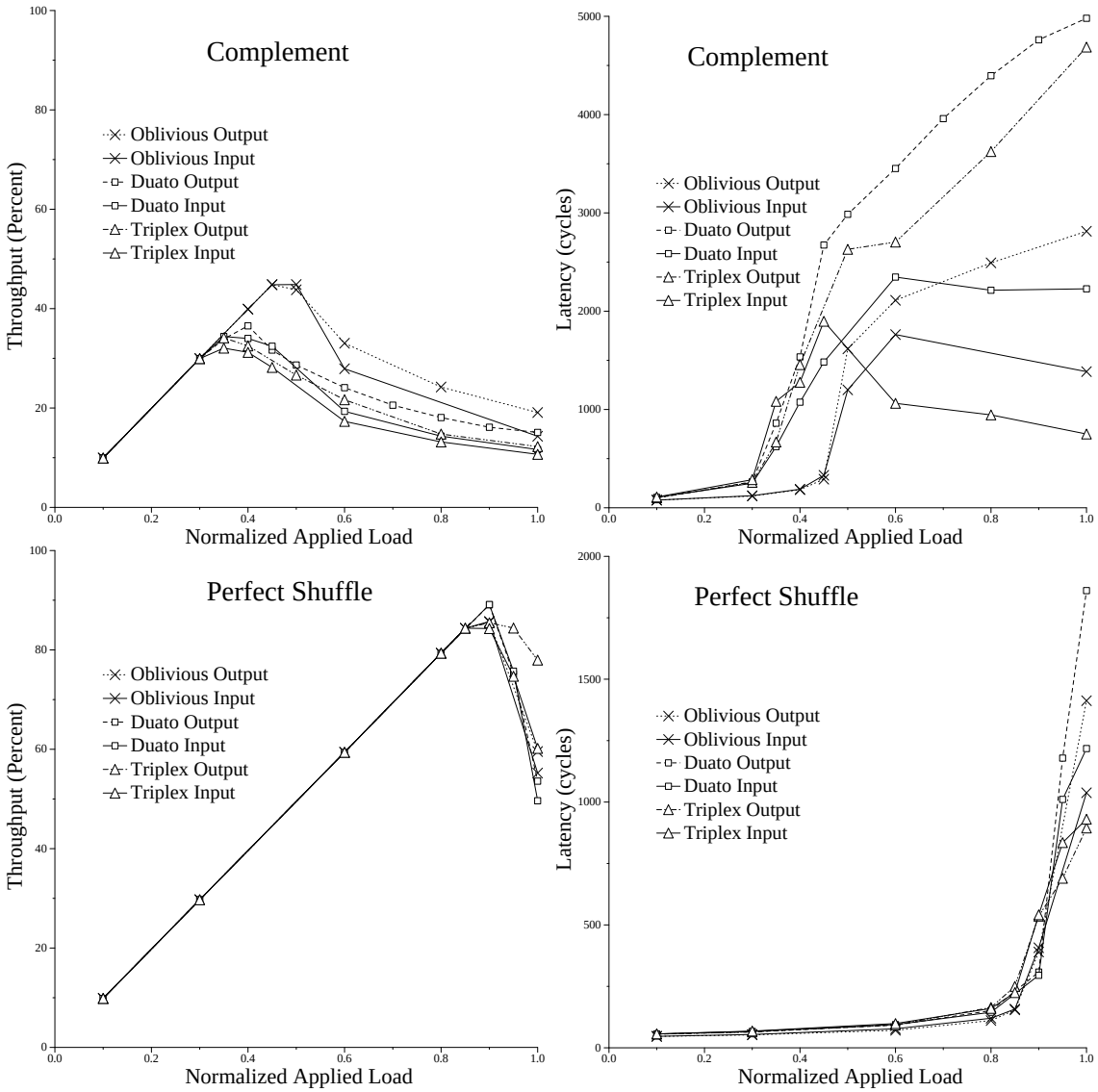

Figure B.14: Throughput and latency on a 256-node mesh with random output buffer selection.

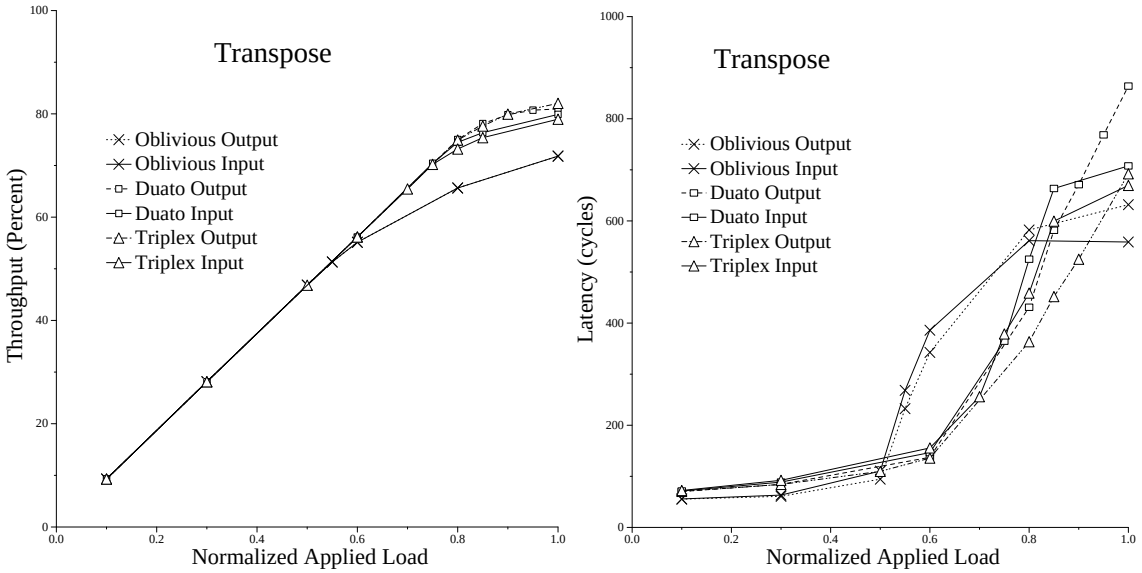

Figure B.15: Throughput and latency on a 256-node mesh with random output buffer selection.

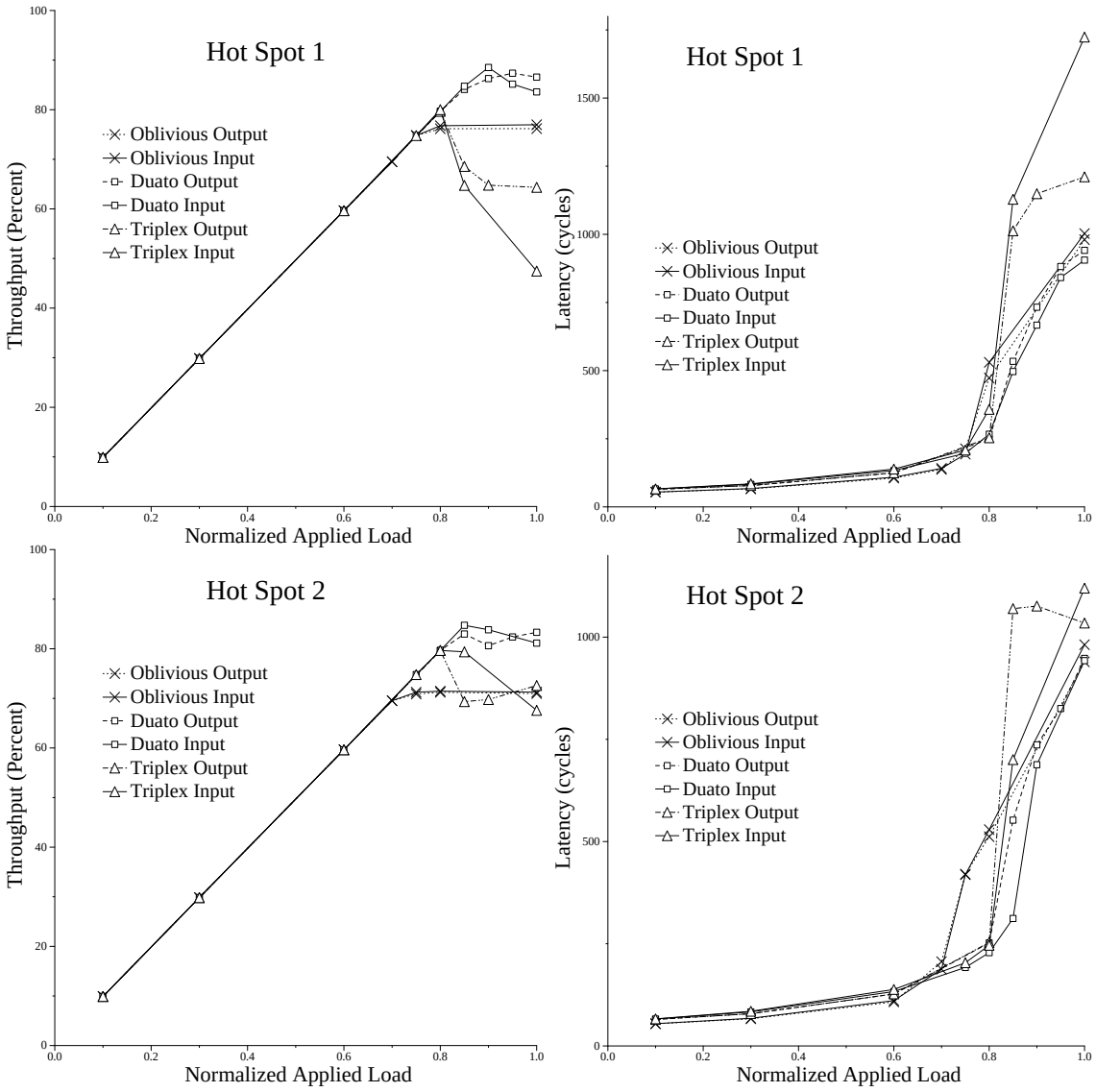

Figure B.16: Throughput and latency on a 256-node mesh with random output buffer selection.

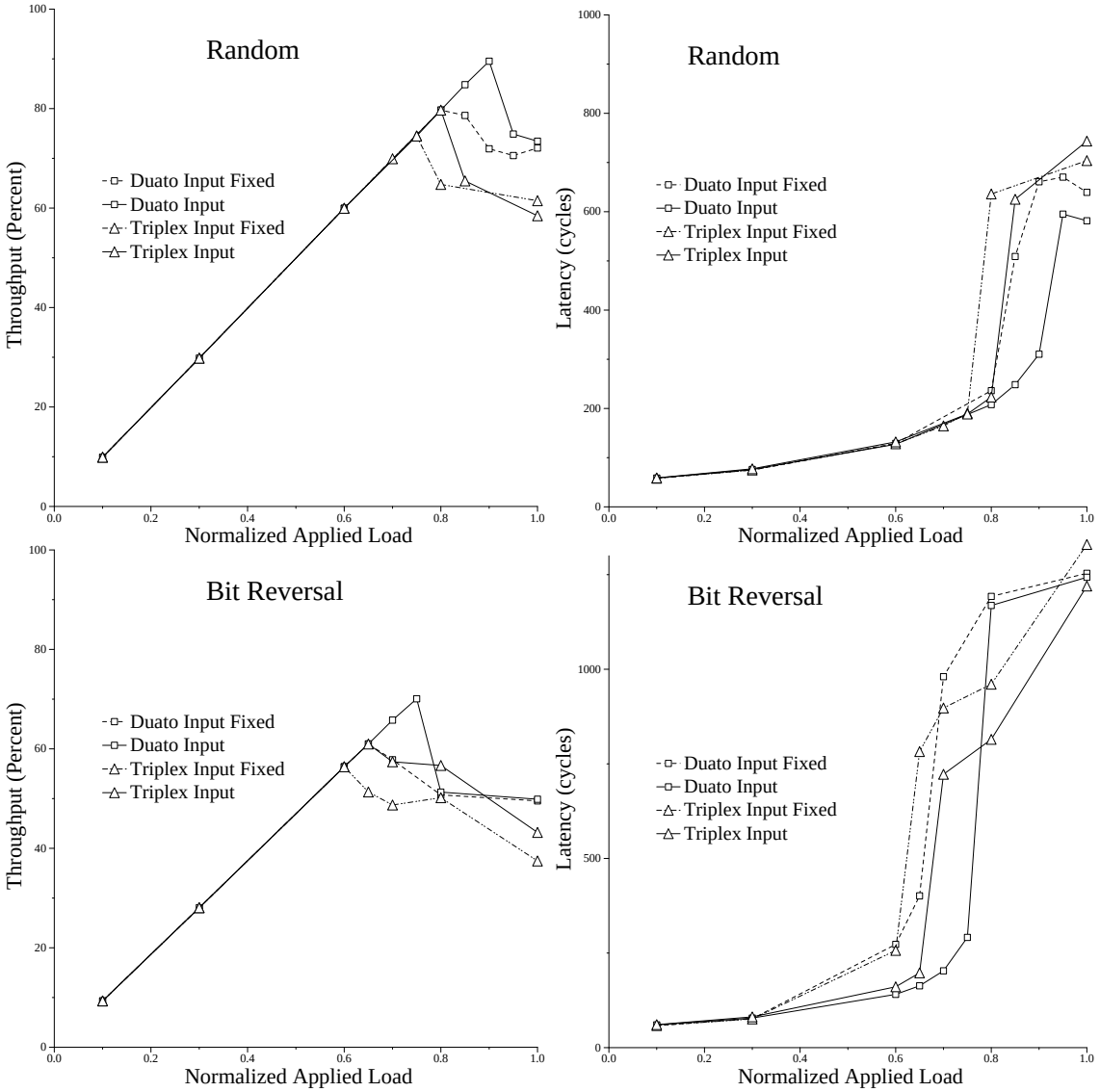

Figure B.17: Throughput and latency on a 256-node torus comparing input driven routing with fixed and random selection.

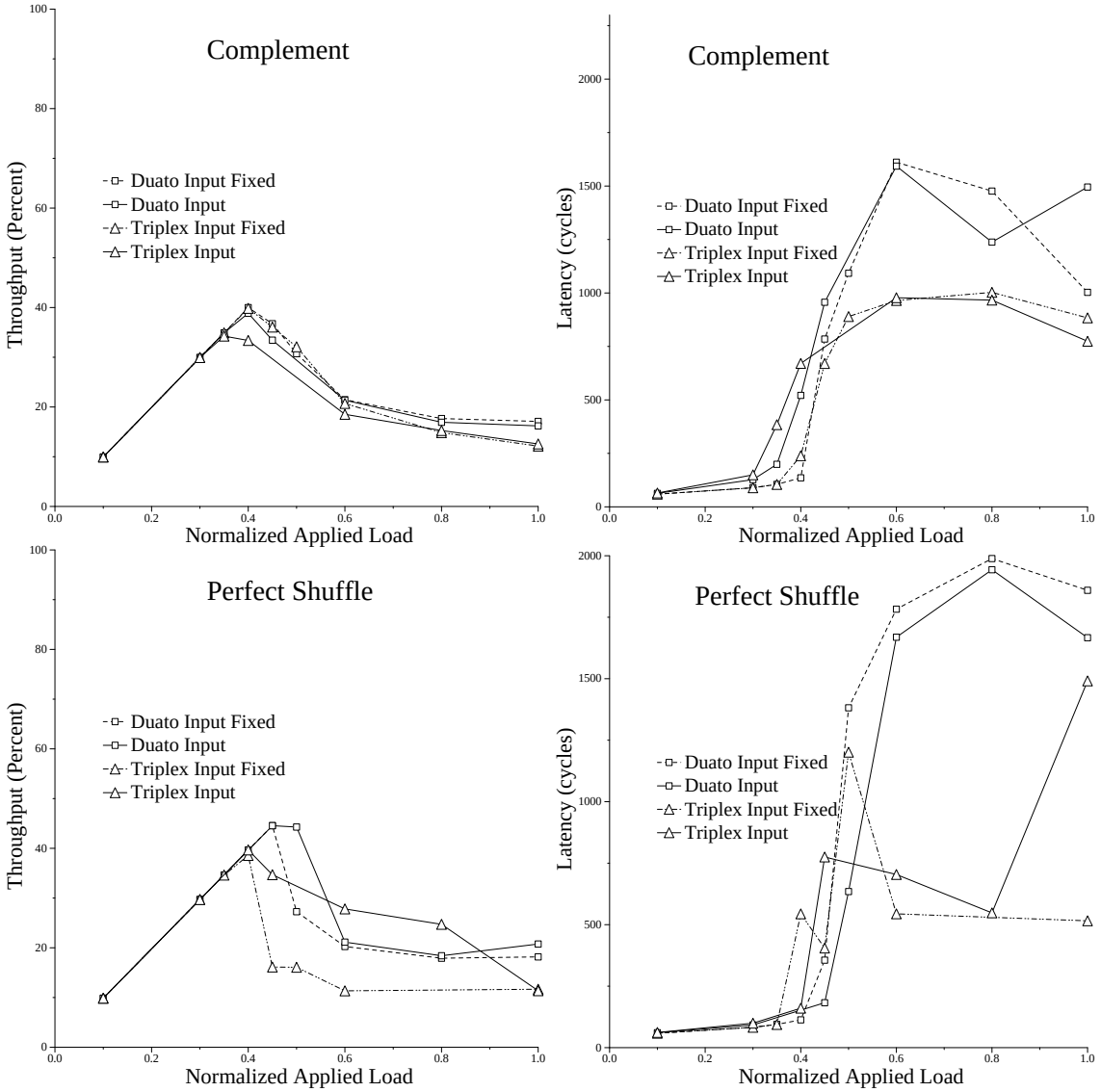

Figure B.18: Throughput and latency on a 256-node torus comparing input driven routing with fixed and random selection.

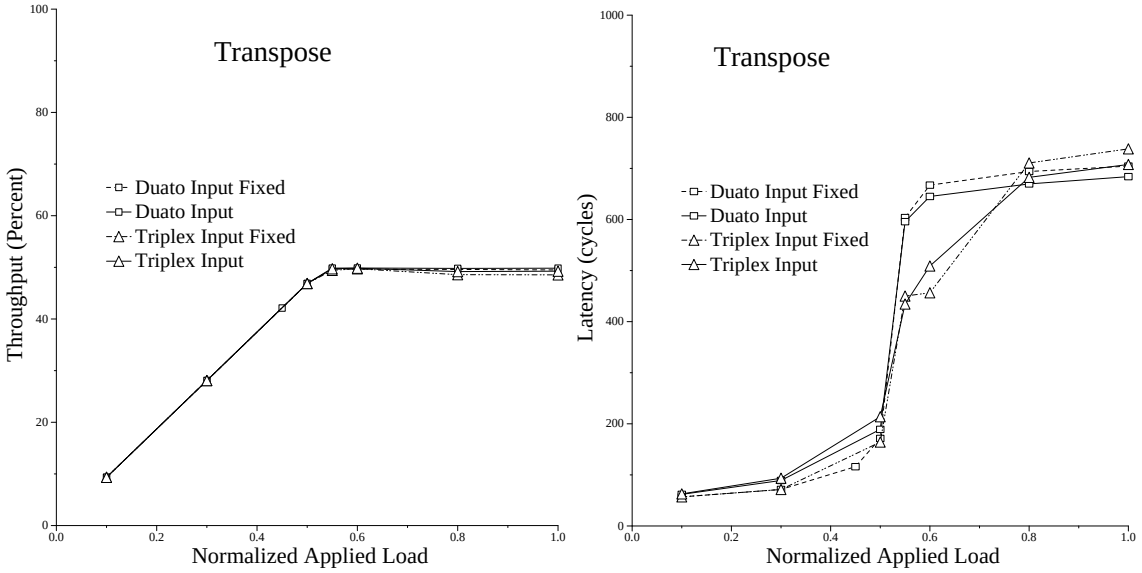

Figure B.19: Throughput and latency on a 256-node torus comparing input driven routing with fixed and random selection.

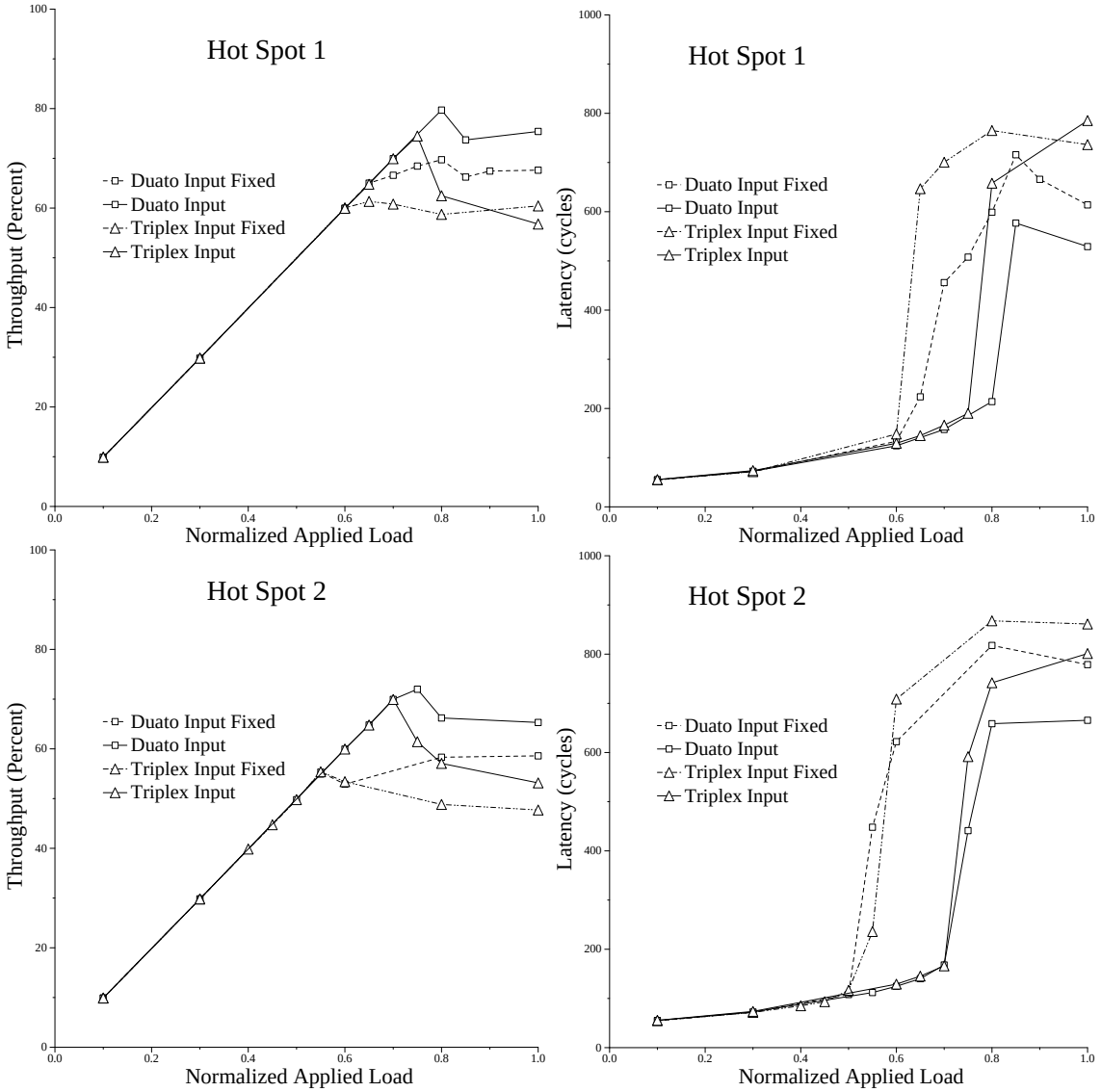

Figure B.20: Throughput and latency on a 256-node torus comparing input driven routing with fixed and random selection.

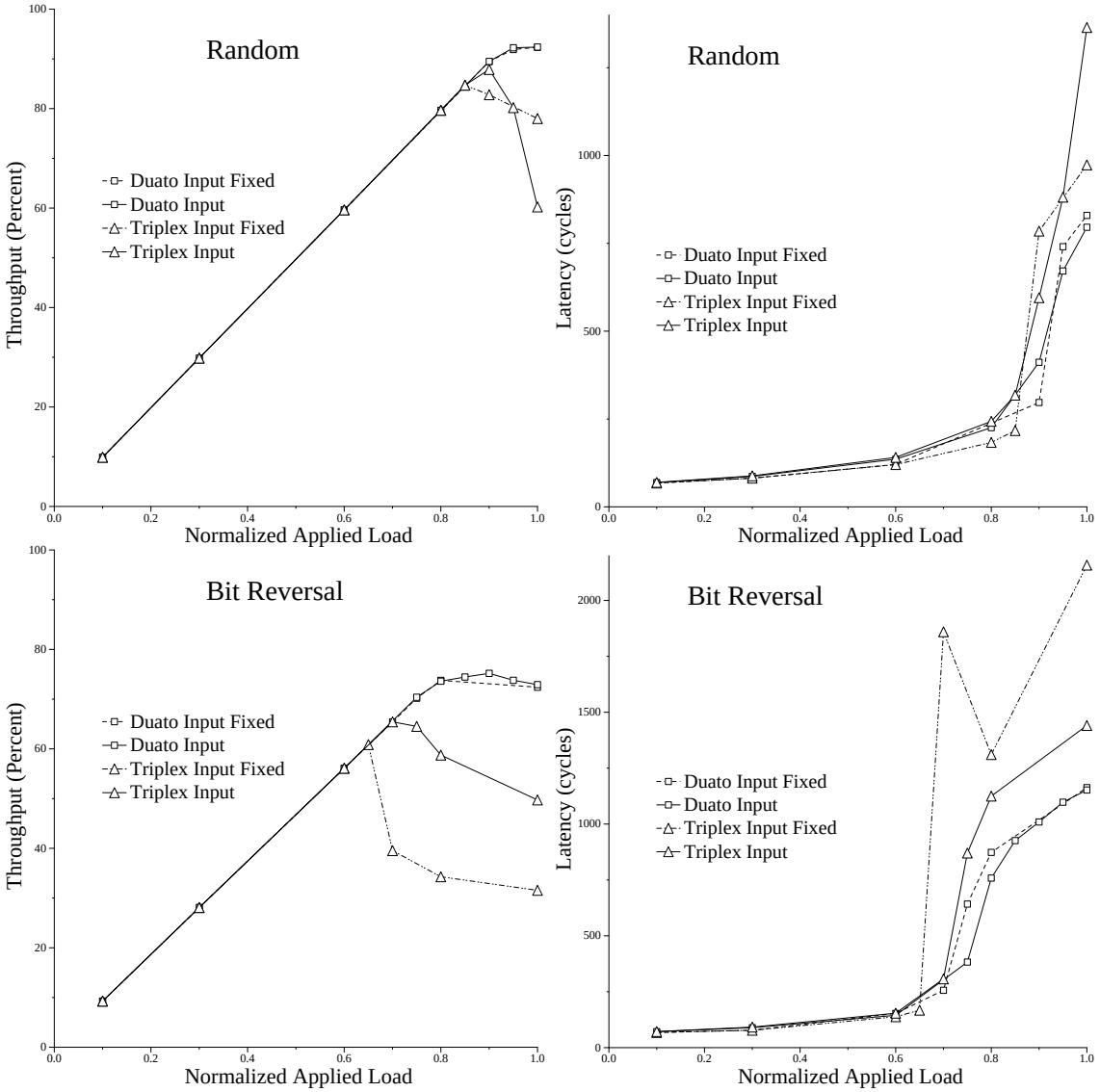

Figure B.21: Throughput and latency on a 256-node mesh comparing input driven routing with fixed and random selection.

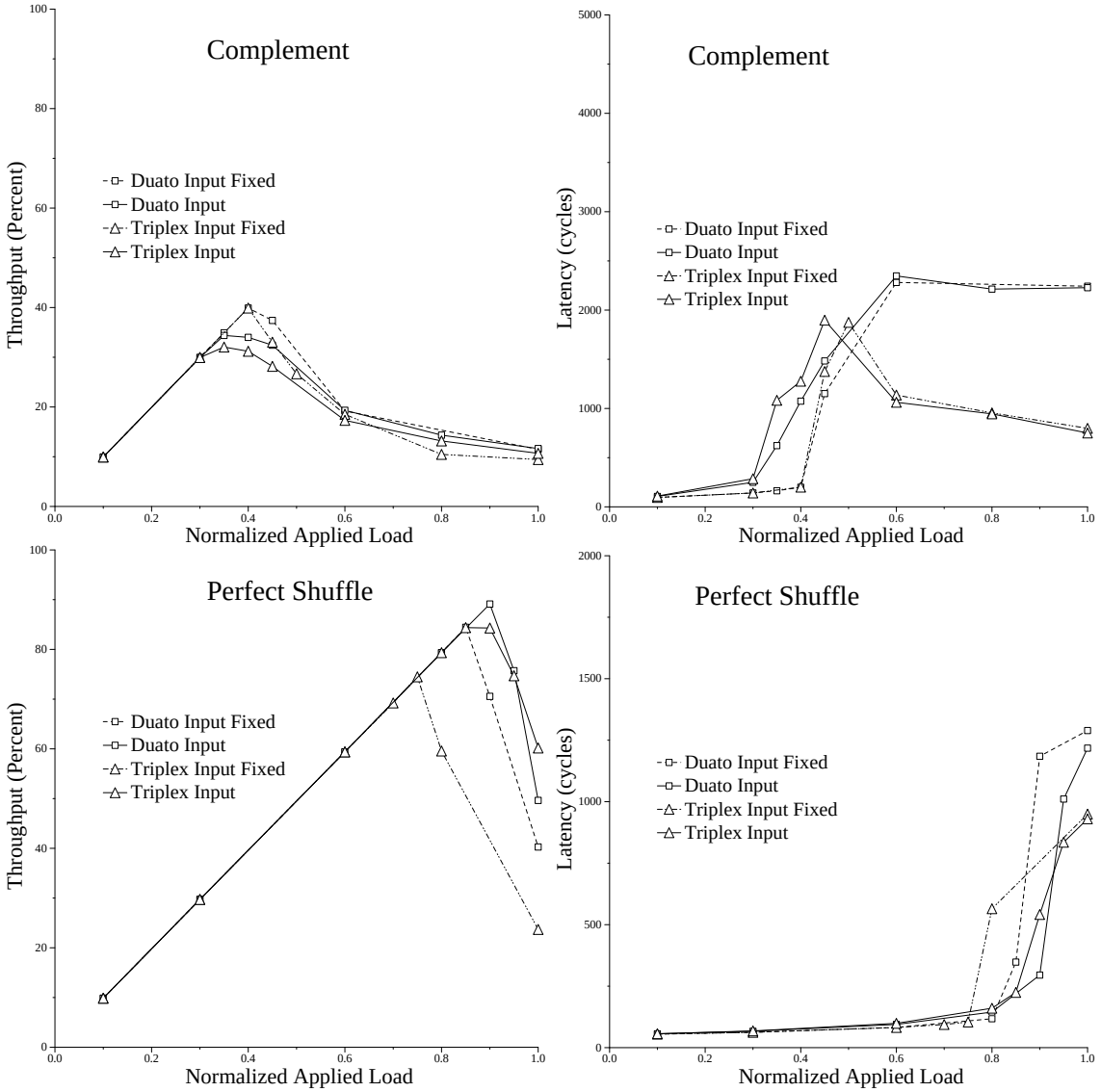

Figure B.22: Throughput and latency on a 256-node mesh comparing input driven routing with fixed and random selection.

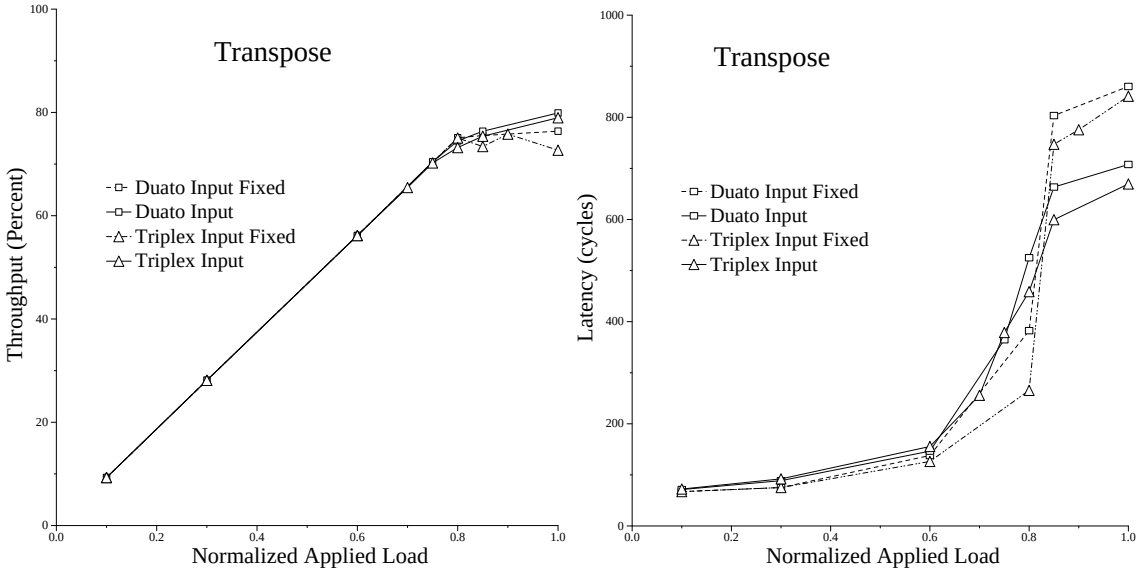

Figure B.23: Throughput and latency on a 256-node mesh comparing input driven routing with fixed and random selection.

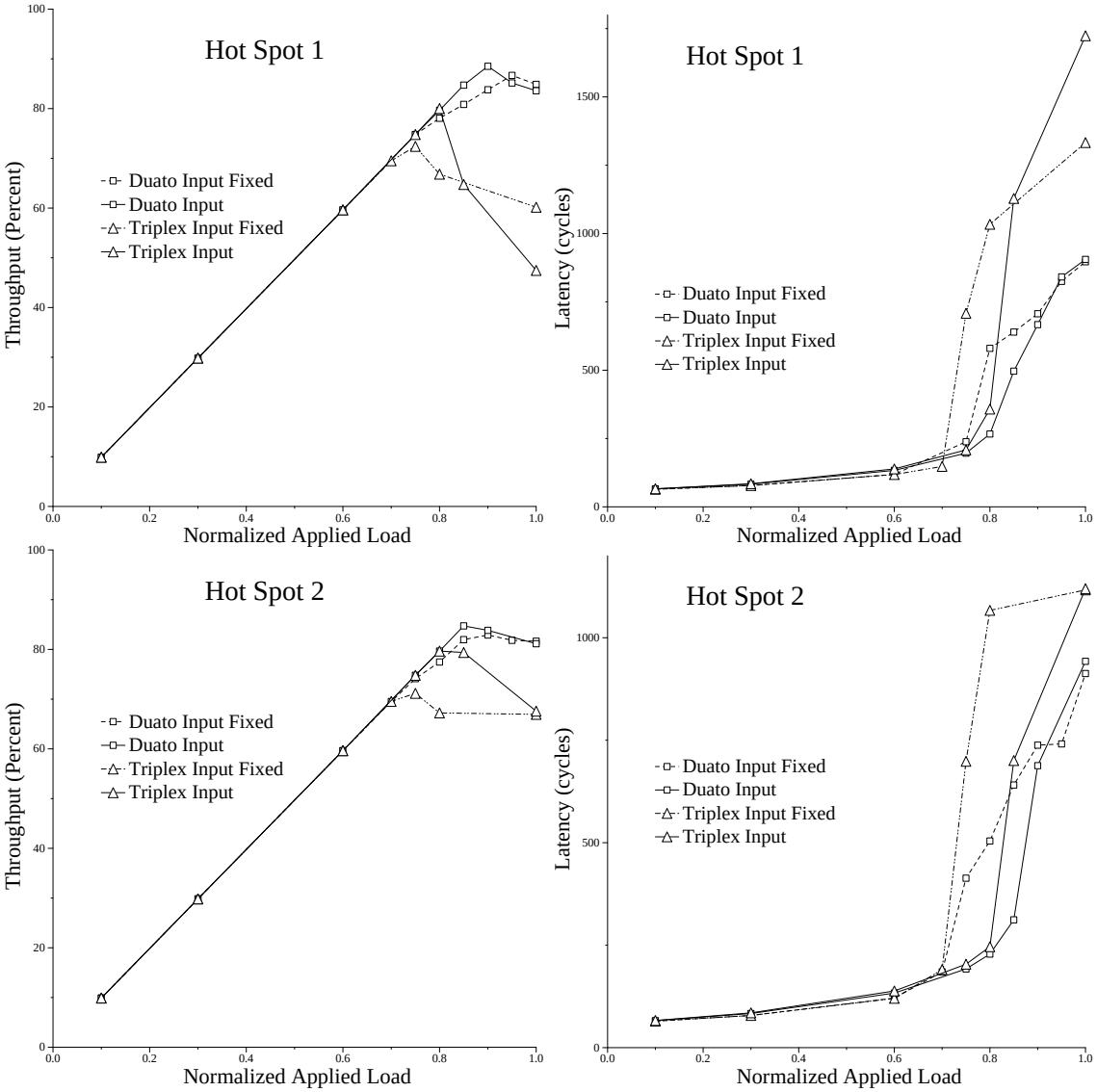

Figure B.24: Throughput and latency on a 256-node mesh comparing input driven routing with fixed and random selection.