Packet Routing in Multiprocessor Networks

by

Donald D. Chinn

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

University of Washington

1995

Approved by_____
(Chairperson of Supervisory Committee)

Program Authorized
to Offer Degree_____

Date_____

In presenting this dissertation in partial fulfillment of the requirements for the Doctoral degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this dissertation is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to University Microfilms, 1490 Eisenhower Place, P.O. Box 975, Ann Arbor, MI 48106, to whom the author has granted "the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform."

Signature_____

Date_____

University of Washington

Abstract

Packet Routing in Multiprocessor Networks

by Donald D. Chinn

Chairperson of the Supervisory Committee: Professor Martin Tompa
Department of Computer Science
and Engineering

Large multiprocessor systems have the potential to solve large problems by breaking them into subtasks and solving them in parallel. When breaking a problem into subtasks and combining the results, processors must communicate by sending *packets* to each other. The time it takes for packets to move across the network that connects the processors will become more significant as multiprocessor systems get larger.

The mesh and torus, where processors are arranged in a grid pattern, are popular networks because of their simplicity and their efficient use of space when physically realized. There is a simple algorithm to route packets in these networks, called the *dimension order algorithm*. In this algorithm, the path of a packet is fixed once it enters the network, regardless of any other traffic in the network.

In an *adaptive* routing algorithm, the path a packet takes from its source to its destination may depend on packets it encounters. *Minimal* adaptive routing algorithms have the additional advantage that the path each packet takes is a shortest one.

One benchmark for a routing algorithm's performance is how quickly it can route an arbitrary *static permutation*, where each processor is the source for at most one packet and the destination for at most one packet. For a large class of minimal adaptive routing algorithms, this dissertation presents an $\Omega(n^2/k^2)$ bound on the worst case time to route a static permutation of packets on an $n \times n$ mesh or torus with nodes that can hold up to $k \geq 1$ packets each. This is the first nontrivial

lower bound on adaptive routing algorithms. The argument extends to a large class of dimension order routing algorithms, yielding an $\Omega(n^2/k)$ time bound.

To complement these lower bounds, this dissertation gives two upper bounds: an $O((n^2/k)+n)$ time dimension order routing algorithm and the first instance of a minimal adaptive routing algorithm that achieves $O(n)$ time with constant sized queues per node.

This dissertation also presents experimental results for two *nonminimal* routing algorithms. The results suggest that these algorithms route permutations similar to the ones in the lower bound above in time that is superlinear in n.

Table of Contents

List of Figures

List of Tables

ACKNOWLEDGMENTS

This dissertation is not the product of just one person, but the collective product of a large group of people.

Martin Tompa, my advisor, guided me through the morass of my half-baked ideas; out of the confusion eventually came fully-baked ideas. He also patiently and diligently read through drafts of this dissertation and other papers, always suggesting ways to improve the writing and presentation. My enthusiasm was influenced by his enthusiasm and encouragement.

Richard Anderson provided guidance during my early experiences in graduate school, which included the trauma of writing my first parallel program. Richard Ladner provided many good suggestions for improving this thesis. I also would like to thank the other theory faculty, Paul Beame and Larry Ruzzo, for their help in my stay at the University of Washington. Larry Snyder and Carl Ebeling were important influences, helping me see some of the practical aspects of the research. I am also indebted to Tom Leighton of MIT. Some of the ideas that he and I discussed evolved into Chapters 2 and 3 of this thesis.

Many friends and fellow graduate students made life in graduate school more enjoyable or helped in some other way. They include Craig Anderson, Gene Anderson, Ruth Anderson, Kathy Armstrong, Brian Bershad, Kevin Bolding, Jeff Dean, Susie Hashisaki, Jean Kaiser, David Keppel, Rakesh Sinha, Mitch Sundt, Raj Vaswani, and Lezlie Watkins. Some of these people will no doubt downplay their contributions; others may not even know what they did to help.

Finally, I would like to thank my parents, my brother, and his wife for their support throughout.

I hope to return as much as what all of these people so generously gave me.

Chapter 1

INTRODUCTION

Massively parallel computers bring large computational resources to bear on large problems. The full power of these machines can be realized if they are built and programmed in a way that overcomes two related problems. The first problem is the division of work and partitioning of data into many independent subproblems. The second problem arises when processing elements need to transmit data to each other. The cost in time for this communication has two sources: the overhead needed to construct a message and the transmission of the message through the interconnection network.

In current machines, the communication time is large enough so that adding more processing power to a problem can increase the time it takes to solve it, because the savings in time from the division of labor is more than offset by the extra communication time. This phenomenon will become worse as processors get faster relative to the rest of the machine. Although currently the overhead to create a message is much greater than the cost to transmit it through the interconnection network, in the future the transmission time will become a greater fraction of the communication time than it is today. As machines get bigger, messages will have to travel through a greater number of nodes to reach their destinations, incurring a greater delay. Also, the time it takes to create a message (a one-time cost per message) will decrease as more sophisticated techniques are employed (e.g., see [Fel93]).

The interconnection network is composed of *nodes*, which usually correspond to processing elements, and *links*, the wires that connect nodes. In one time step, a node can transmit one message along each of its links. Messages travel from node to node, and each node decides how to send messages it currently holds. How

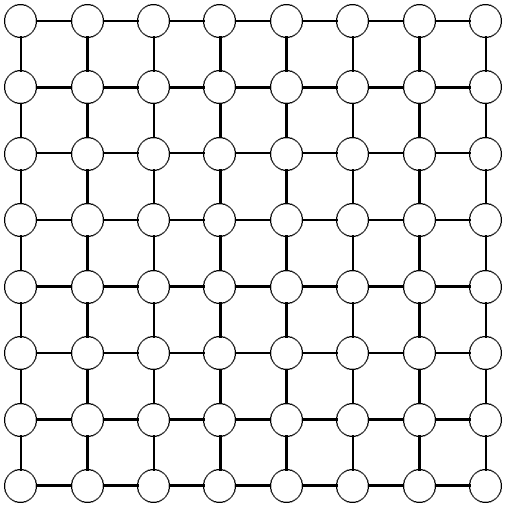

Figure 1.1: An 8×8 mesh.

fast these decisions are made and how good these decisions are determine how fast messages reach their destinations. If the decisions are poor, many messages will travel through the same nodes or compete for the same links, causing contention for resources and hence delays.

The way nodes and links are organized is called the network *topology*. For example, the (two-dimensional) *mesh* topology is the arrangement of nodes in a grid pattern (Figure 1.1). The (two-dimensional) *torus* topology is a mesh with the two ends of each row and column connected. The *N-node binary hypercube* topology is the topology where each node is labelled by a string of $\log N$ bits, and the node labelled $b_1 b_2 \ldots b_{\log N}$ ($b_i \in \{0, 1\}$) is connected to $b_1 \ldots b_{i-1} \overline{b_i} b_{i+1} \ldots b_{\log N}$, for all $1 \leq i \leq \log N$ (where $\overline{b_i} = 1 - b_i$).

The mesh and torus topologies have attracted much attention in multiprocessor network design because of their simplicity and their efficient use of space when physically realized. Examples of machines that use the mesh or torus topology include the MPP from Goodyear Aerospace [Bat80], the MP-1 from MasPar [MP-87], the Paragon from Intel Scientific, the J-machine from MIT [ND90], the Touchstone DELTA from Intel [Int91], the DASH from Stanford [LLJ+92], and the Mosaic from Cal Tech [SBSS93].

This thesis focuses on the time to transmit messages once they are in the network. The method by which messages are sent through the network is called the *routing algorithm*. Messages are typically subdivided into smaller entities called *packets*, but for the purposes of this thesis, we will consider messages and packets to be equivalent.

In practice, all packet routing algorithms are very simple. For instance, algorithms used in practice make routing decisions based only on each packet's preferred directions (that is, which links from the packet's current node bring it closer to its destination), rather than its full destination address. None of the algorithms used in practice works well with constant sized queues in the worst case.

Routers in almost all state-of-the-art machines use *oblivious* algorithms, where a message's path depends only on the source and destination addresses of the packet. The simplest of these algorithms on the two-dimensional mesh is the *dimension order* algorithm: a packet moves along its row until it reaches its destination column, and then moves along that column until it reaches its destination. The logic to implement the routing decisions is relatively simple in this approach. Furthermore, the algorithm generalizes to higher dimensional meshes such as the hypercube. However, oblivious routing performance degrades quickly in the presence of congestion or faults, because the algorithm is not flexible enough to use the available bandwidth effectively. That is, although there might be unused links in the network at a given time, oblivious algorithms are unable to alter the paths of packets to use these available links. Machines that use oblivious algorithms include the Intel Touchstone [Int91] and Paragon, the MIT J-machine [ND90], and the Mosaic from Cal Tech [SBSS93].

Adaptive routing is an alternative to oblivious routing. In adaptive algorithms, the path a message takes from its source to its destination may depend on packets it encounters. Intuitively, adaptive routers potentially can use the available bandwidth to relieve the congestion or to route around faults. In *minimal* adaptive routing, the path a packet takes is a shortest one. An example of a minimal adaptive algorithm is the one of Cypher and Gravano [CG92] or of Chien and Kim [CK92]. In *nonminimal* adaptive routing, a packet may take any path between its source and its destination, possibly making moves in the network that place it farther from its destination than before the move. When a packet makes such a move, the packet

is said to have been *derouted*. Nonminimal routing allows the most flexibility in packet paths, but at a cost of more complex logic to avoid *livelock*, the situation in which a packet never reaches its destination because it is derouted frequently. Examples of adaptive routers include the Chaos router [BFS94, KS90, KS91] and the Ngai and Seitz router [NS89, NS91].

In theory, there are many fast algorithms for static[1] routing problems on synchronous networks[2], but all make use of large queues or information about destination addresses beyond just preferred directions. (See Sections 1.1 and 1.2.) An example of the latter would be those routing algorithms that are based on sorting packets according to their destination addresses. Often these complicating considerations render the algorithms impractical, particularly if one wants to generalize them to dynamic routing problems or asynchronous networks.

One of the simplest benchmarks for a router's performance is how it performs in the worst case on static *one-to-one* (or *partial permutation*) routing problems, where each processor sends at most one message and receives at most one message. The permutation problem models real-time systems where the communication pattern among processors is unknown, yet performance guarantees must be satisfied. It also models problems, such as scientific problems, that have "bursty" communication behavior, where processors compute locally for some time and then simultaneously communicate with each other (see [SWG92]).

An appealing feature of this metric of performance is that the traffic generated by a static permutation does not contain any "delivery hot spots," where one node is the destination of many packets. When there are delivery hot spots, performance can be limited trivially by delivery bandwidth. Another reason for interest in permutations is that those that arise in practice, such as the transpose permutation,

---

[1] A *static* problem is one in which each node is allowed to inject some fixed number of packets into the network, at most one per time step. In a *dynamic* routing problem, each node also can inject at most one packet per time step, but there is no limit on the total number of packets that a node injects.

[2] A *synchronous* network is one in which the behavior of the network is determined completely by the what packets are injected into the network and when they were injected. In contrast, an *asynchronous* network's behavior is not so determined, due to nondeterministic behavior of the clocks at the nodes.

cause poor performance on some networks.

For example, the dimension order algorithm on the N-node binary hypercube topology routes all packets in the transpose permutation (where $n = \log N$ and the packet at source $b_1 b_2 \ldots b_{n/2} b_{(n/2)+1} \ldots b_n$ is sent to destination $b_{(n/2)+1} \ldots b_n b_1 b_2 \ldots b_{n/2}$) in $\Omega(\sqrt{N}/\log N)$ steps, even though no packet need traverse more than $\log N$ nodes. To see this, observe that all packets whose source is $b_1 b_2 \ldots b_{n/2} 0 \ldots 0$ must travel through the node labelled $0 \ldots 0$. There are $\sqrt{N}$ such packets, and since the node labelled $0 \ldots 0$ can transmit at most $\log N$ packets per time step, the lower bound follows.

On the N-node three-dimensional mesh ($n \times n \times n$, $N = n^3$), Leighton [Lei] observes that the dimension order algorithm routes all packets in the bit-reversal permutation (where the packet at source $b_1 b_2 \ldots b_{3 \log n}$ is sent to destination $b_{3 \log n} \ldots b_2 b_1$) in $\Omega(n^2)$ steps, even though no packet need travel more than $3n$ nodes. If the first $\log n$ bits of the labelling of a node represent the x-coordinate of the node when realized in three-dimensional space, the next $\log n$ bits represent the y-coordinate, and the last $\log n$ bits represent the z-coordinate, then any packet whose source is $b_1 \ldots b_{\log n} \quad 0 b_{\log n + 2} \ldots b_{2 \log n - 1} 1 \quad 0 \ldots 0$ must cross the link that connects the node labelled $0 \ldots 0 \quad 01 \ldots 1 \quad 0 \ldots 0$ with the node labelled $0 \ldots 0 \quad 10 \ldots 0 \quad 0 \ldots 0$. There are $n^2/4$ such packets, and the link can only transmit one packet per time step, and so the lower bound follows.

These examples immediately raise the question of whether there is a permutation, for any given network, that causes poor performance. On the $n \times n$ mesh, there is theoretically enough bandwidth to deliver all packets in any permutation in time proportional to n. At the very least, a good routing algorithm should be able to route permutations efficiently (i.e., in $O(n)$ time).

For some time now, both theoreticians and practitioners have been trying to find a simple routing algorithm that works well in the worst case. In the present context, "simple" will be taken to mean deterministic, bounded queues (i.e., the size of the queues does not grow with the size of the network), no dependence on destination other than preferred directions, and minimal (i.e., shortest) routes from source to destination.

This thesis explores the limitations and possibilities of finding simple routing

algorithms in the restricted domain of permutation routing on the mesh. In Chapter 2, we will prove that it is impossible for any such simple routing algorithm to work well in the worst case. In particular, for any such simple routing algorithm on the $n \times n$ mesh, $\Omega(n^2/k^2)$ time is required to route all packets in some permutation routing instance, where $k \geq 1$ is the capacity of each queue. This is the first nontrivial lower bound for adaptive routing. We also provide a simple algorithm that routes any permutation on the $n \times n$ mesh in $O((n^2/k) + n)$ time. In Chapter 3, we present the first minimal adaptive routing algorithm that achieves $O(n)$ time with bounded queues. The algorithm exploits the full destination addresses (and, like the other known algorithms, it does so in a complicated and possibly impractical way). Thus, it is impossible to eliminate the assumption of no dependence on destination other than preferred directions from the lower bound of Chapter 2. Chapter 4 describes a series of experiments on nonminimal adaptive algorithms and gives the results, which suggest that perhaps none of those discussed can route arbitrary permutations in $O(n)$ time.

A preliminary version of portions of Chapters 2 and 3 has appeared previously [CLT94]. A preliminary version of portions of Chapter 4 has also appeared previously [Chi]. This chapter concludes by surveying some of the known results for permutation routing.

1.1 Routing with Unbounded Queues

Borodin and Hopcroft [BH85] prove an $\Omega(\sqrt{N}/d^{3/2})$ time bound for routing the worst case permutation on any N-node, degree d network using any oblivious routing algorithm. Kaklamanis *et al.* [KKT90] improve the bound to $\Omega(\sqrt{N}/d)$. These results are useful for networks such as the hypercube, whose diameter and degree are $\log_2 N$, but are no better asymptotically than the diameter lower bound of $2\sqrt{N} - 2$ on the two-dimensional mesh.

It is well known that dimension order paths can be used to route any permutation on the $n \times n$ mesh in $2n - 2$ steps, matching the diameter lower bound (see Leighton [Lei92, pages 159–162]). Unfortunately, this algorithm requires $\Theta(n)$ size queues at each node. (Leighton [Lei90] proves that if each packet has a random destination — i.e., the routing problem is not necessarily a permutation — then

with high probability all packets will be delivered in $2n + O(\log_2 n)$ steps and none of the queues ever contains more than four packets. However, this average case setting is not the one we consider here.) Our goal is to prove that $O(n)$ time routing of arbitrary permutations on the $n \times n$ mesh is impossible in a more practical setting, which includes bounding the queue size of each node.

1.2 Routing with Bounded Queues

Little is known about lower bounds that exploit the fact that nodes have bounded queues. Krizanc [Kri91] proves such a bound for any *source-oblivious* routing algorithm, which is one where the path a packet takes only depends on its current location and destination. He shows that for any source-oblivious algorithm on an N-node, degree d network each of whose nodes can hold up to k packets, there is a partial permutation that requires $\Omega(N/d^4k(8k)^{5k})$ time to route. Krizanc's model, however, is restrictive: if a node sends a packet to a neighboring node and causes that neighboring node to exceed its capacity, the network is in an illegal configuration. A more realistic model would allow the node to detect the state of its neighbor and not send the packet.

Maggs and Sitaraman [MS92] prove that for any nonpredictive routing algorithm on an N-node butterfly with queues of size k at each node, there exists a permutation that requires $\Omega(N/(k \log_2 N))$ time to route. A nonpredictive routing algorithm is one in which contention for links is resolved independent of destination addresses of packets.

Another approach to permutation routing is to sort blocks of packets by destination and then advance them to their destinations by the dimension order algorithm. Packets in these algorithms may take paths that are nonminimal (i.e., make moves that place them farther away from their destination during the sorting phases). For the $n \times n$ mesh, Kunde [Kun88] shows that such a deterministic algorithm can route every permutation in $2n + O(n/k)$ time using queues of size k. Using Kunde's approach, Leighton, Makedon, and Tollis [LMT89] and Rajasekaran and Overholt [RO92] improve the bound to $2n - 2$ steps using constant (albeit large) sized queues per node. However, these algorithms may be too complicated, and too specifically tailored to static permutations and synchronous networks to be practical for general

routing.

Han and Stanat [HS90] provide routing algorithms for the mesh that are not based on sorting, but do use nonminimal paths and knowledge of full destination addresses. Their algorithms can route any permutation in $O(n)$ time and require constant sized queues per node. However, like the sorting-based algorithms, their algorithms may be too specifically tailored to static permutations and synchronous networks to be practical.

The desire to have simple routing algorithms with constant sized queues per node has led to the growing body of literature on *hot potato* (or *deflection*) routing [BNRST93, BC91, FR92, Haj91, KKR93, NS92], where at each step every node in the network must send all packets it received during the previous step. In these algorithms, no extra queues are needed, and packets again typically take nonminimal paths. Newman and Schuster [NS92] give an algorithm that routes any permutation in $7n + o(n)$ steps, but the algorithm uses sorting. Bar-Noy *et al.* [BNRST93] provide a deterministic hot potato routing algorithm not based on sorting that routes any permutation in $n2^{O(\sqrt{\log_2 n \log_2 \log_2 n})}$ steps. In the same paper, they provide a simpler $O(n^{3/2})$ algorithm.

Because the known $O(n)$ time routing algorithms on the mesh may not be practical, there is still considerable interest in finding practical ones. Notice that the $O(n)$ time bounds mentioned earlier [HS90, Kun88, Lei92, LMT89, NS92, RO92] each violate either the assumption of bounded queues, or both the assumptions of minimal paths and using only preferred directions. We will see in Chapter 2 that there is no $O(n)$ time algorithm that obeys all of these restrictions.

Chapter 2

A LOWER BOUND FOR MINIMAL ADAPTIVE ALGORITHMS

We now show that for each algorithm in a large class of minimal adaptive algorithms for the $n \times n$ mesh, there exists a permutation for that algorithm that requires $\Omega(n^2/k^2)$ steps to route all of its packets, where k is the maximum number of packets a node is allowed to hold at any time. Section 2.1 defines the model of the network that we will use throughout Chapters 2 and 3. Section 2.2 describes how to construct a permutation that will be used to prove the lower bound. Section 2.3 shows that the permutation constructed in Section 2.2 takes $\Omega(n^2/k^2)$ steps to route all of its packets.

The techniques of this chapter can be used to obtain similar lower bounds on the torus, for h-h routing problems, for nonminimal algorithms, and for dimension order algorithms. This is done in Section 2.4. The permutations constructed in this chapter form the basis of the experiments performed in Chapter 4.

2.1 The Model

Consider an $n \times n$ mesh network. The network can be viewed as a directed graph $G = (V, E)$ such that if the edge (u, v) is in E, then the edge (v, u) is also in E. The edge (u, v) is an *outlink* of the node u, and the edge (v, u) is an *inlink* of u.

Each node has a central queue that can hold up to k packets. In one step, each node decides deterministically which packets to attempt to transmit along its outlinks (at most one per outlink), decides which of the incoming packets to accept, and then transmits those packets that were accepted by its neighbors. When a packet reaches its destination, it is considered delivered and removed from the network. This is a multi-port model, in the terminology of Borodin *et al.* [BRSU93]. For consistency with the existing literature, we use the word "queue" to denote the

set of waiting packets in a node. The packets do not have to be served in a first-in first-out (FIFO) fashion.

The *outqueue policy of a node* is the method by which a node decides which packets, of all the packets in its queue, to attempt to send out its outlinks. No more than one packet can be scheduled to each outlink. Examples of outqueue policies are FIFO or farthest-first [Lei92, page 159]. The *inqueue policy of a node* determines which packets, of all the packets that attempt to enter a node, will be accepted. The inqueue policy must guarantee that the queue does not overflow (i.e., accept more packets than it is capable of holding). A packet is transmitted exactly when the outqueue policy of its queue selects it and the inqueue policy of the target node accepts it.

Also defined for each node is a *state*, which each of the policies can use to make its decision. The state is allowed to change at the end of the step as a function of the current state and the packets in the node. An example of the use of state is the round-robin inqueue policy, where a node accepts packets from each neighboring node in turn when there is competition for available space in its queue.

The *state of a packet* consists of information that can be modified by a node when the packet is in the node. An example of this is the arrival time of a packet at the current node. This information is transmitted along with the packet.

For the lower bound that follows, we restrict ourselves to deterministic routing algorithms for the mesh that use minimal (shortest-distance) paths. In addition, the only part of a packet's destination address that routing decisions may use is the packet's profitable outlinks (i.e., those outlinks from the current node that move the packet closer to its destination). We impose no further restrictions on routing decisions.

We make this last restriction precise as follows:

- The outqueue policy of a node can be a function only of the states, source addresses, and profitable outlinks of the packets in the node; and the state of the node.

- The inqueue policy of a node can be a function only of the states, source addresses, and profitable outlinks of the packets in the node and the packets

scheduled to enter the node (where profitable outlinks of scheduled packets are measured as profitable from the node from which they are coming); and the state of the node.

- The state of a node at the beginning of step $t+1$ can be a function only of its state at the beginning of step t; and the states at the beginning of step t, source addresses, and profitable outlinks of the packets in the node at the end of step t. The initial state of a node can be a function only of its address and the profitable outlinks of the packet that originates there.

- The state of a packet at the beginning of step $t+1$ can be a function only of the state at the beginning of step t of the node it occupies at the end of step t; and the states at the beginning of step t, source addresses, and profitable outlinks of the packets that occupy the same node at the end of step t. The initial state of a packet can be a function only of the initial state of its node, and its own source address and profitable outlinks.

Let us call an algorithm that obeys this last restriction a *destination-exchangeable routing algorithm*. Note that the restriction to profitable outlinks is similar to the definition of a *nonpredictive* algorithm, given by Ranade [Ran87], Leighton [Lei92, page 556], and Maggs and Sitaraman [MS92]. One example of a destination-exchangeable algorithm is the dimension order algorithm with FIFO queues and round-robin inqueue policy. An adaptive example might be similar, except that each packet moves in one profitable direction until it is blocked by congestion, and then moves in its other profitable direction, continuing this alternation until it reaches its destination. Other minimal adaptive algorithms that could be implemented with a destination-exchangeable algorithm include those of Chien and Kim [CK92] and Cypher and Gravano [CG92]. An example of a non-minimal destination-exchangeable algorithm is the $O(n^{3/2})$ hot potato algorithm of Bar-Noy *et al.* [BNRST93].

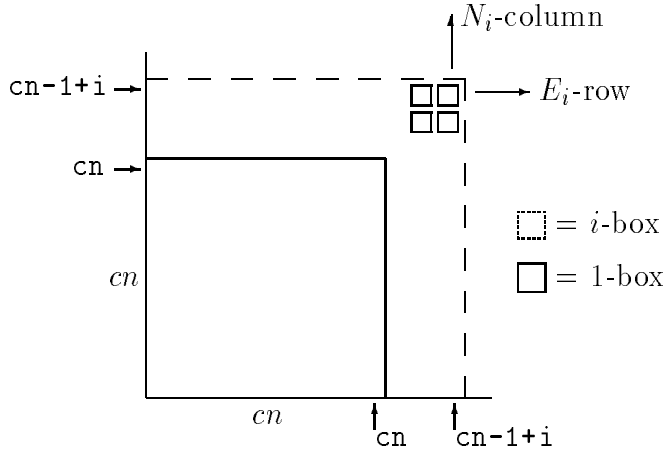

Figure 2.1: The $n \times n$ mesh.

Definitions

Number the columns of the mesh 1 to n from west to east and the rows 1 to n from south to north. Let c be a constant, to be determined later, so that $cn = \Theta(n/k)$ is an integer. An *N_i-packet* is a packet that starts in the $cn \times cn$ submesh located in the southwest corner of the mesh and is destined for the $(cn - 1 + i)$-th column (call this column the N_i-column) north of the $(cn - 1 + i)$-th row. An *E_i-packet* is a packet that starts in the $cn \times cn$ submesh located in the southwest corner and is destined for the $(cn - 1 + i)$-th row (call this row the E_i-row) east of the $(cn - 1 + i)$-th column. (See Figure 2.1.)

The *i-box* is the set of nodes west of and including the N_i-column and south of and including the E_i-row. Define the 0-box to be the set of nodes west of the N_1-column and south of the E_1-row. A packet is *in the i-box* if it is in a node of the i-box. A packet is *outside the i-box* if it is not in the i-box.

An *exchange of two packets x and x'* is a switching of their destination addresses. The remaining packet information (state and source address) remains unchanged.

A packet is *scheduled to enter a node v* during some step if the outqueue policy of its current node chooses it to advance into node v.

2.2 The Construction

For any given deterministic, destination-exchangeable, minimal adaptive routing algorithm, we will construct a permutation that forces the algorithm to take $\Omega(n^2/k^2)$ steps to deliver all of its packets. The idea behind the construction is that there are $\Omega(n^2/k^2)$ packets in the $cn \times cn$ submesh destined for nodes outside the submesh, but only a constant number will depart the 1-box during each of the first $\Theta(n)$ steps, a constant number will depart the 2-box during each of the next $\Theta(n)$ steps, etc., up to the l-box, where $l = \Theta(n/k^2)$.

We can do this by maintaining the following invariant: the only packets in the nodes of the north edge of the 1-box are E_1-packets, and the only packets in the nodes of the east edge of the 1-box are N_1-packets (except for the node in the northeast corner of the 1-box, which is allowed to contain both N_1- and E_1-packets). While this invariant holds, at most two packets — an N_1-packet and an E_1-packet — can escape the 1-box, since the routing algorithm is minimal. As we shall see, we can maintain this invariant for $\Theta(n)$ steps, because the routing algorithm is destination-exchangeable.

Then, for the next $\Theta(n)$ steps, we maintain a similar invariant for the 2-box, which allows at most one N_2-packet and one E_2-packet (and any number of N_1- and E_1-packets) to escape the 2-box. Successive phases of $\Theta(n)$ steps maintain similar invariants. Figure 2.2 illustrates the invariant of the construction in the general case.

We now present the construction.

1. Let $p = \left\lfloor (k+1)(cn + c^2 n) + dn \right\rfloor$, where c and d are constants to be determined later, and cn and dn are integers. For each $1 \leq i \leq \lfloor l \rfloor$, where $l = c^2 n^2/(2p)$, place p N_i-packets and p E_i-packets in the 1-box such that only N_1-packets are in the N_1-column at or south of the E_1-row, only E_1-packets are in the E_1-row west of the N_1-column, and there is no more than one packet per node. (Note that there must be N_1- and E_1-packets in the 0-box as well.) Assign unique row destinations in the N_i-column outside the i-box for N_i-packets, and unique column destinations in the E_i-row outside the i-box for E_i-packets. It is easy to see that such an arrangement is possible,

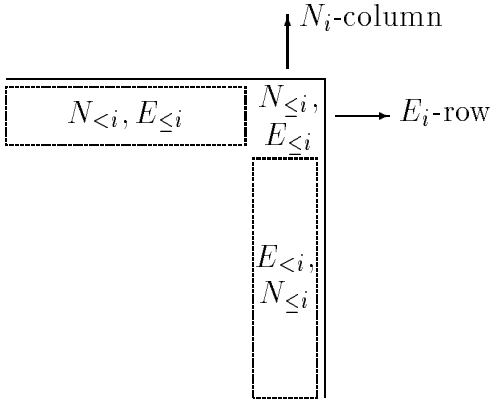

Figure 2.2: The i-box up to step idn. "$N_{<i}, E_{\leq i}$" indicates that the node contains only N_j-packets, where $j < i$, or E_j-packets, where $j \leq i$ (and similarly for "$E_{<i}, N_{\leq i}$" and "$N_{\leq i}, E_{\leq i}$").

provided $\lfloor (k+1)(cn + c^2 n) + dn \rfloor \leq (1-c)n - l$. (See Section 2.3.3.)

2. If desired, place additional packets in any way that forms a partial permutation. (At the extremes, no additional packets could be placed, or enough packets could be added to form a full permutation.)

3. Run the routing algorithm for $\lfloor l \rfloor dn$ steps, performing the following exchanges (in any order) as necessary. (Lemmas 2.3 and 2.4 will show that packets needed for the exchanges are always available, provided $l \leq c^2 n$, which is satisfied in Section 2.3.3.)

   - EX1. For $i \geq 1$, $j > i$, if an E_j-packet is scheduled by the outqueue policy of a node to enter the E_i-row west of the N_i-column during steps 1 to idn, then exchange that packet with an E_i-packet in the $(i-1)$-box that is not scheduled to enter the E_i-row.
   - EX2. For $i \geq 1$, $j > i$, if an N_j-packet is scheduled by the outqueue policy of a node to enter the N_i-column south of the E_i-row during steps 1 to idn, then exchange that packet with an N_i-packet in the $(i-1)$-box that is not scheduled to enter the N_i-column.
   - EX3. For $i \geq 1$, $j \geq i$, if an E_j-packet is scheduled by the outqueue policy of a node to enter the N_i-column south of the E_i-row during steps

1 to idn, then exchange that packet with an N_i-packet in the $(i-1)$-box that is not scheduled to enter the N_i-column.

- EX4. For $i \geq 1$, $j \geq i$, if an N_j-packet is scheduled by the outqueue policy of a node to enter the E_i-row west of the N_i-column during steps 1 to idn, then exchange that packet with an E_i-packet in the $(i-1)$-box that is not scheduled to enter the E_i-row.

The t-th step of the construction consists of the following sequence of activities for each node:

(a) The outqueue policy chooses packets to schedule along its outlinks.

(b) Exchanges are made, if necessary.

(c) The inqueue policy decides which packets to accept.

(d) Packets are transmitted.

(e) The state of the node and the states of packets in the node are updated.

Part (b) is executed only for the purposes of the construction and the lower bound argument. The actual routing algorithm ignores part (b). The first step corresponds to $t = 1$. The phrase "immediately after step 0" will mean at the beginning of the construction.

4. The *constructed permutation* of the routing algorithm is the set of packets (defined by their source and destination addresses after all exchanges) at the end of $\lfloor l \rfloor dn$ steps of the construction given above, including, of course, the packets that were delivered in those steps.

2.3 The Lower Bound

We now show that any destination-exchangeable routing algorithm takes $\Omega(n^2/k^2)$ steps to deliver all the packets in its constructed permutation. We begin by proving facts about the construction itself and then prove that when the routing algorithm is run on the constructed permutation, these facts also hold.

2.3.1 Properties of the Construction

In the construction, for all $1 \leq i \leq \lfloor l \rfloor$, the following lemmas hold:

Lemma 2.1 *During step t, where $1 \leq t \leq (i-1)dn$, no N_j-packets or E_j-packets, for $j \geq i$, leave the i-box.*

Lemma 2.2 *During step t, where $(i-1)dn < t \leq idn$, at most one N_i-packet and one E_i-packet leave the i-box.*

Lemma 2.3 *During step t, where $1 \leq t \leq idn$, there is always an N_i-packet eligible for EX2 and EX3, i.e., in the $(i-1)$-box and not scheduled to enter the N_i-column, provided $l \leq c^2 n$.*

Lemma 2.4 *During step t, where $1 \leq t \leq idn$, there is always an E_i-packet eligible for EX1 and EX4, i.e., in the $(i-1)$-box and not scheduled to enter the E_i-row, provided $l \leq c^2 n$.*

Lemma 2.5 *For $j \geq i > 1$, no N_j-packet is outside the $(i-2)$-box immediately after step t, where $0 \leq t \leq (i-1)dn$.*

Lemma 2.6 *For $j \geq i > 1$, no E_j-packet is outside the $(i-2)$-box immediately after step t, where $0 \leq t \leq (i-1)dn$.*

Lemma 2.7 *No N_i-packet is at or north of the E_i-row and also west of the N_i-column immediately after step t, where $0 \leq t \leq idn$.*

Lemma 2.8 *No E_i-packet is at or east of the N_i-column and also south of the E_i-row immediately after step t, where $0 \leq t \leq idn$.*

Note that, because the paths are minimal and no exchanges take an N_i-packet outside the i-box, no N_i-packet can ever be east of the N_i-column in the construction. Similarly, no E_i-packet can ever be north of the E_i-row.

Proof: We will prove all of the lemmas simultaneously by induction on t.

BASIS : $t = 0$. Lemmas 2.1 through 2.4 are vacuously true, since they do not apply when $t = 0$. Lemmas 2.5 through 2.8 are verified by inspection of the initial arrangement of the packets.

INDUCTION : Assume Lemmas 2.5 through 2.8 are true for all steps at or before $t-1$ (where $t-1 < \lfloor l \rfloor dn$). We will prove all the lemmas true for any i at step t.

Lemma 2.1 follows from Lemmas 2.5 and 2.6 for step $t-1$. No N_j-packet leaves the i-box as a result of an exchange, part (b), of step t. Suppose such a packet existed. Then there is an N_m-packet, for some $m > j$, or an E_m-packet, for some $m \geq j$, that is scheduled to enter the N_j-column (i.e., it is in the N_{j-1}-column and hence outside the $(i-2)$-box immediately after step $t-1$), contradicting Lemmas 2.5 or 2.6.

Since no N_j-packet has left the $(i-2)$-box by the end of step $t-1$ by Lemma 2.5, then no such packet can leave the i-box during the transmission, part (d), of step t.

A similar argument holds for E_j-packets.

Lemma 2.2 follows from Lemmas 2.7 and 2.8 for step $t-1$. No N_i-packet leaves the i-box in part (b) of step t because the exchange rules never take an N_i-packet outside the i-box. The only N_i-packet that can leave the i-box during part (d) of step t is the one at the N_i-column and E_i-row, since all other nodes on the boundary of the i-box that could eject an N_i-packet out of the i-box during part (d) of step t do not have an N_i-packet immediately after step $t-1$ (Lemma 2.7). A similar argument holds for E_i-packets using Lemma 2.8.

Lemma 2.3 follows from Lemma 2.7 for step $t-1$, and Lemmas 2.1 and 2.2 for step t. The number of N_i-packets that begin the construction is $\Big\lfloor (k+1)(cn + c^2 n) + dn \Big\rfloor$. From Lemmas 2.1 and 2.2 for step t, we know that no more than $dn - 1$ N_i-packets have left the i-box before step t. By Lemma 2.7, the only N_i-packets that are in the i-box but outside the $(i-1)$-box are in the N_i-column, and there are at most $k(cn - 1 + i)$ queue positions for them. Any other N_i-packets ineligible for exchange must be scheduled to enter the N_i column. Suppose there are x packets that need to be exchanged with N_i-packets during step t; in particular, these x packets must also be scheduled to enter the N_i-column. Then there can be at most $cn - 1 + i - x$ N_i-packets scheduled to

enter the N_i-column. Putting these terms together, we find that the number of N_i-packets that are in the $(i-1)$-box and not scheduled to enter the N_i-column is at least $\lfloor (k+1)(cn+c^2n)+dn \rfloor - (dn-1) - k(cn-1+i) - (cn-1+i-x) \geq x$, since $i \leq l \leq c^2 n$. Thus, all x exchanges can be performed.

Lemma 2.4 follows from Lemma 2.8 for step $t-1$, Lemmas 2.1 and 2.2 for step t, and the number of E_i-packets that begin the construction, analogous to the proof of Lemma 2.3.

Lemma 2.5 follows from Lemmas 2.5 and 2.6 for step $t-1$ and Lemmas 2.3 and 2.4 at step t.

No N_j-packet can be outside the $(i-2)$-box immediately after part (b) of step t. Suppose such a packet existed. Then there is an N_m-packet, for some $m > j$, or E_m-packet, for some $m \geq j$, that is scheduled to enter the N_j-column (i.e., it was outside the $(i-2)$-box immediately after step $t-1$), contradicting Lemmas 2.5 or 2.6.

If an N_j-packet is scheduled to enter the N_{i-1}-column during part (d) of step t, then EX2 and Lemma 2.3 ensure that we can perform an exchange, leaving the N_j-packet in the $(i-2)$-box. If an N_j-packet is scheduled to enter the E_{i-1}-row during part (d) of step t, then EX4 and Lemma 2.4 ensure that we can perform an exchange, leaving the N_j-packet in the $(i-2)$-box.

Lemma 2.6 follows from Lemmas 2.5 and 2.6 for step $t-1$ and Lemmas 2.3 and 2.4 at step t, analogous to the proof of Lemma 2.5, except that EX1 and EX3 are the relevant exchange rules.

Lemma 2.7 follows from Lemma 2.7 at step $t-1$ and Lemma 2.4 at step t. The proof is similar to that of Lemma 2.5, except we observe that an exchange can never take an N_i-packet outside the $(i-1)$-box, and EX4 prevents an N_i-packet from entering the E_i-row west of the N_i-column.

Lemma 2.8 follows from Lemma 2.8 at step $t-1$ and Lemma 2.3 at step t. The proof is similar to that of Lemma 2.7, except that EX3 is the relevant exchange rule. □

Corollary 2.9 *Immediately after step $\lfloor l \rfloor dn$ of the construction, there is still an undelivered packet in the network.*

Proof: Choose $i = j = \lfloor l \rfloor$ in Lemmas 2.1 and 2.2. Then at least $\left\lfloor (k+1)(cn + c^2 n) \right\rfloor$ N_i-packets and $\left\lfloor (k+1)(cn + c^2 n) \right\rfloor$ E_i-packets remain in the i-box, and hence are undelivered, at time $\lfloor l \rfloor dn$. □

2.3.2 Properties of the Constructed Permutation

We must now show that the analogue of Corollary 2.9 holds when routing the constructed permutation, where, of course, no exchanges are performed.

The *configuration of a node* is the description of the packets in the node (their states, sources, and destinations) and the state of the node.

The *configuration of a network* is the collective configurations of all of its nodes.

The *transition function* $\delta(S, t)$ of a routing algorithm maps a configuration S of a network and a number of steps t into a configuration of the network, the configuration that results after executing t steps of the routing algorithm (with no exchanges) starting from configuration S. Since the routing algorithms we are considering are deterministic, this function is well-defined. Also, define $\delta(S, 0) = S$ for any configuration S. Note that, for any configuration S, $\delta(S, i+j) = \delta(\delta(S, i), j)$.

We now show that when certain pairs of packets are exchanged, the routing algorithm behaves essentially in the same way.

Lemma 2.10 *Let S be the configuration of a network. For any $1 \leq i \leq \lfloor l \rfloor$, let x and x' be two packets in the $(i-1)$-box in S and whose destinations are at or east of the N_i-column and at or north of the E_i-row. Let $S_{x,x'}$ be S with x and x' exchanged. Then $\delta(S_{x,x'}, 1)$ is $\delta(S, 1)$ with x and x' exchanged.*

Proof: Note that both packets can move north or east profitably while they are in the $(i-1)$-box. Recall that an exchange of x and x' interchanges the destination addresses only, and does not alter any other packet information. Because of this, the routing decisions the algorithm makes cannot distinguish between S and $S_{x,x'}$. The outqueue policy and inqueue policy depend only on the state of the node, and the states, source addresses, and profitable outlinks of packets. The state of nodes in the next step is a function only of its previous state, and the states, source

addresses, and profitable outlinks of packets in the node. The state of packets is also a function only of quantities that do not change when an exchange is performed.

Since the decisions made by the routing algorithm are the same whether x and x' are exchanged or not, then $\delta(S_{x,x'}, 1)$ is $\delta(S, 1)$ with x and x' exchanged. □

Lemma 2.11 *Let S be the configuration of a network. Let X be a sequence of pairs of packets such that both packets in each pair are in the $(i-1)$-box in S, for some i, and have destinations at or east of the N_i-column and at or north of the E_i-row. (The value of i can be different for each pair.) Let S_X be S with each pair in X exchanged. Then $\delta(S_X, 1)$ is $\delta(S, 1)$ with each pair in X exchanged.*

Proof: The proof is by induction on the size of X using Lemma 2.10. □

Note that $(S_X)_Y = S_{<X,Y>}$ for any configuration S and sequences of pairs of packets X and Y, where $< X, Y >$ denotes the concatenation of sequences X and Y.

We now show that the routing algorithm, when run using the constructed permutation, behaves essentially like the construction.

Lemma 2.12 *Let S_t be the configuration of the network immediately after step t in the construction. Let S' be the configuration of the network with the constructed permutation immediately after step 0. Then for $t = \lfloor l \rfloor dn$, $\delta(S', t) = S_t$.*

Proof: For $1 \leq i \leq \lfloor l \rfloor dn$, let X_i be the sequence of pairs of packets that are exchanged during step i of the construction. We prove a stronger statement: $\delta(S', t)$ is S_t with $< X_{t+1}, ..., X_{\lfloor l \rfloor dn} >$ exchanged, for all $0 \leq t \leq \lfloor l \rfloor dn$.

The proof is by induction on t. Figure 2.3 illustrates the induction step.

Basis : $t = 0$. $\delta(S', 0) = S'$ is S_0 with all of the pairs of packets in $< X_1, X_2, \ldots, X_{\lfloor l \rfloor dn} >$ exchanged (by the definition of S').

Induction : Assume the statement is true for $t - 1 < \lfloor l \rfloor dn$, so that $\delta(S', t-1)$ is S_{t-1} with $< X_t, \ldots, X_{\lfloor l \rfloor dn} >$ exchanged. Define S^*_{t-1} to be S_{t-1} with each pair of X_t exchanged. Therefore, $\delta(S', t-1)$ is S^*_{t-1} with $< X_{t+1}, \ldots, X_{\lfloor l \rfloor dn} >$ exchanged.

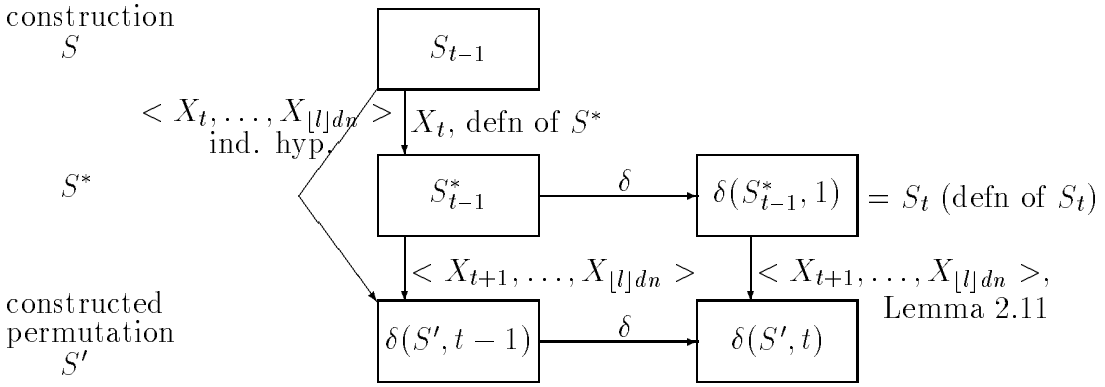

Figure 2.3: S_t, S^*_t, and $\delta(S', t)$ in Lemma 2.12.

All of the exchanges in $< X_{t+1}, \ldots, X_{\lfloor l \rfloor dn} >$ are of the type in the preconditions of Lemma 2.11: for any pair of packets exchanged during any step of the construction, each packet of the pair at the time of the exchange is in the $(i-1)$-box for some i and destined for nodes northeast of the $(i-1)$-box, and neither of the two packets could have escaped the $(i-1)$-box before step t (by Lemmas 2.5 through 2.8). Then by Lemma 2.11, $\delta(S', t)$ is $\delta(S^*_{t-1}, 1)$ with $< X_{t+1}, \ldots, X_{\lfloor l \rfloor dn} >$ exchanged.

But $\delta(S^*_{t-1}, 1) = S_t$, since S_t is the routing algorithm run for one step on S_{t-1} with X_t exchanged. (That is, since all pairs in X_t are of the type in Lemma 2.11, then starting in S_{t-1} and performing an exchange of X_t either before or after part (a) of the t-th step results in the same configuration, S_t.) Therefore, $\delta(S', t)$ is S_t with $< X_{t+1}, \ldots, X_{\lfloor l \rfloor dn} >$ exchanged, which proves the induction step. □

Since $S_{\lfloor l \rfloor dn}$ still has an undelivered packet in the network (Corollary 2.9), we have:

Theorem 2.13 *The configuration $\delta(S', \lfloor l \rfloor dn)$ contains an undelivered packet. That is, it takes at least $\lfloor l \rfloor dn$ steps for any deterministic, destination-exchangeable, minimal adaptive routing algorithm to deliver all of the packets in its constructed permutation.*

2.3.3 Choosing the Constants c and d

All that is left to complete the analysis is to find constants c and d that satisfy the following constraints:

1. There are enough distinct rows (columns) for the destinations of all N_i-packets (respectively, E_i-packets),

2. cn and dn are integers, and

3. $l \leq c^2 n$ (needed in Lemmas 2.3 and 2.4).

The first constraint is

$$\left\lfloor (k+1)(cn + c^2 n) + dn \right\rfloor \leq (1-c)n - l.$$

Rewritten, this becomes

$$\left\lfloor (k+1)(cn + c^2 n) + dn \right\rfloor + \frac{c^2 n^2}{2\left\lfloor (k+1)(cn + c^2 n) + dn \right\rfloor} \leq (1-c)n.$$

This is satisfied, provided

$$(k+2)c + (k+1)c^2 + d + \left(\frac{c^2}{2((k+1)(c+c^2)+d)} \right) \leq 1, \tag{2.1}$$

since $\lfloor x \rfloor + (a/\lfloor x \rfloor) \leq x + (a/x)$ when $\lfloor x \rfloor \geq \sqrt{a}$.

Choosing $c = 1/(2 \cdot (k+2))$ and $d = 2/5$ satisfies Inequality (2.1) for $k \geq 1$. Taking derivatives of the left hand side of Inequality (2.1) with respect to c and d shows that it is monotonically increasing in c and d, and so choosing any $c \leq 1/(2 \cdot (k+2))$ and any $d \leq 2/5$ satisfies Inequality (2.1).

To satisfy the second constraint, choose the largest $c \leq 1/(2 \cdot (k+2))$ such that cn is an integer, and do likewise for $d \leq 2/5$. Then $c \geq 1/(2 \cdot (k+2)) - 1/n$ and $d \geq 2/5 - 1/n$. For $n \geq 24 \cdot (k+2)^2$ (which we will need in the proof of Theorem 2.14) and $k \geq 1$, $c \geq 2/(5 \cdot (k+2))$ and $d \geq 1/3$.

The third constraint is verified for these new values of c and d (and again for $n \geq 24 \cdot (k+2)^2$) as follows:

$$\begin{aligned} l &= \frac{c^2 n^2}{2 \cdot \lfloor (k+1)(cn + c^2 n) + dn \rfloor} \\ &\leq \frac{c^2 n^2}{2 \cdot (k+1)cn + 2dn} && (cn,\ dn \text{ integers}) \\ &\leq \frac{c^2 n}{\frac{4 \cdot (k+1)}{5 \cdot (k+2)} + \frac{2}{3}} \\ &< c^2 n. \end{aligned}$$

We can now calculate the lower bound of Theorem 2.13 in terms of n and k by substituting our choices of c and d.

Theorem 2.14 *For any deterministic, destination-exchangeable, minimal adaptive routing algorithm on the $n \times n$ mesh with queues of size $k \geq 1$, it takes $\Omega(n^2/k^2)$ steps to deliver all of the packets in its constructed permutation.*

Proof:

Case 1: $n \geq 24 \cdot (k+2)^2$. By Theorem 2.13, the number of steps is at least

$$\begin{aligned} \lfloor l \rfloor dn &\geq \left\lfloor \frac{c^2 n}{2 \cdot ((k+1)(c + c^2) + d)} \right\rfloor dn \\ &\geq \left\lfloor \frac{2n}{25(k+2)^2 ((k+1)(\frac{2}{5(k+2)} + \frac{4}{25(k+2)^2}) + \frac{2}{5})} \right\rfloor \cdot \frac{1}{3} n \\ &= \left\lfloor \frac{n}{(k+1)(5(k+2)+2) + 5(k+2)^2} \right\rfloor \cdot \frac{1}{3} n \\ &\geq \left\lfloor \frac{n}{12(k+2)^2} \right\rfloor \cdot \frac{1}{3} n \\ &\geq \left(\frac{n}{12(k+2)^2} - 1 \right) \cdot \frac{1}{3} n \\ &\geq \left(\frac{n}{12(k+2)^2} - \frac{n}{24(k+2)^2} \right) \cdot \frac{1}{3} n && (n \geq 24(k+2)^2) \end{aligned}$$

$$= \Omega(n^2/k^2).$$

CASE 2: $n < 24 \cdot (k+2)^2$. The diameter bound immediately yields a $2n - 2 = \Omega(n^2/k^2)$ bound. □

2.4 Extensions of the Lower Bound

We now give some extensions to the argument that apply to other models and routing problems.

2.4.1 Other Queue Types

Suppose a node, instead of containing a single central queue, consists of four *incoming queues*, one associated with each inlink, such that when a packet enters a node, it is placed in the appropriate incoming queue. We can also consider *outgoing queues*, where a packet in an outgoing queue means that it is waiting to be sent along the node's appropriate outlink.

A node using a central queue of size $4k$ can simulate a node with four incoming queues each of size k. The key to the simulation is to use the state of the node to record in which queue each packet is located. The simulation also can be done for outgoing queues. The simulation can be done in a destination-exchangeable way, so that a destination-exchangeable routing algorithm that uses incoming or outgoing queues can be simulated by a destination-exchangeable algorithm that uses central queues.

After recalculating constants c and d in the analysis, we can conclude that the lower bound applies asymptotically to networks whose nodes' queues consists of incoming or outgoing queues.

2.4.2 The Torus

The bound of Section 2.3.3 also holds asymptotically for the torus. The construction simply is applied to a contiguous $(n/2) \times (n/2)$ submesh of the torus, also yielding a lower bound of $\Omega(n^2/k^2)$ steps.

2.4.3 h-h Routing Problems

In h-h routing problems, each node is the source for up to h packets and the destination for up to h packets. The construction is modified to have h packets in each of the c^2n^2 nodes of the 1-box. (Assume for the moment that $h \leq k$.) As before, we define $p = \left\lfloor (k+1)(cn + c^2n) + dn \right\rfloor$, but we define $l = hc^2n^2/(2p)$. We again have three constraints similar to those of Section 2.3.3 to satisfy. The first constraint is

$$\left\lfloor (k+1)(cn + c^2n) + dn \right\rfloor \leq h\left((1-c)n - l\right).$$

Rewritten, this becomes

$$\left\lfloor (k+1)(cn + c^2n) + dn \right\rfloor + \frac{h^2c^2n^2}{2\left\lfloor (k+1)(cn + c^2n) + dn \right\rfloor} \leq h(1-c)n.$$

This inequality is satisfied if

$$(k+1)(cn + c^2n) + dn + \frac{h^2c^2n^2}{2((k+1)(cn + c^2n) + dn - 1)} \leq h(1-c)n,$$

since $\lfloor x \rfloor + a/\lfloor x \rfloor \leq x + a/(x-1)$ for $x > 1$.

Thus, it suffices to satisfy

$$(k+1+h)c + (k+1)c^2 + d + \frac{h^2c^2}{2((k+1)(c+c^2) + d - \frac{1}{n})} \leq h.$$

Now suppose

$$(k+1)(c+c^2) + d \geq \frac{1}{2}h + \frac{1}{n}. \tag{2.2}$$

Then it suffices to satisfy

$$(k+1+h)c + (k+1+h)c^2 + d \leq h. \tag{2.3}$$

Choosing $c = h/(3 \cdot (k+1+h))$ and $d = 5h/9$ satisfies Inequality (2.3). Since the left hand side of Inequality (2.3) is increasing in c and d, choosing any $c \leq h/(3 \cdot (k+1+h))$ and any $d \leq 5h/9$ satisfies Inequality (2.3).

To satisfy the second constraint (cn and dn are integers), choose the largest $c \leq h/(3\cdot(k+1+h))$ such that cn is an integer, and do likewise for $d \leq 5h/9$. Then $c \geq h/(3\cdot(k+1+h)) - 1/n$ and $d \geq 5h/9 - 1/n$. For $n \geq 52(k+1+h)^2/h^2$, $k \geq 1$, and $h \geq 2$, $c \geq h/(4\cdot(k+1+h))$ and $d \geq 11h/20$. Note that $11h/20 \geq h/2 + 1/n$ when $n \geq 52(k+1+h)^2/h^2$, so that Inequality (2.2) is satisfied.

The third constraint ($l \leq c^2 n$) immediately is satisfied, since Inequality (2.2) implies $l \leq hc^2n^2/(2\lfloor(hn/2)+1\rfloor) \leq c^2 n$.

As in the proof of Theorem 2.14, we have that for $n \geq 52(k+1+h)^2/h^2$,

$$\begin{aligned}
\lfloor l \rfloor dn &\geq \left\lfloor \frac{hc^2 n}{2\cdot((k+1)(c+c^2)+d)} \right\rfloor dn \\
&\geq \left\lfloor \frac{h^3 n}{32(k+1+h)^2((k+1)(\frac{h}{4(k+1+h)} + \frac{h^2}{16(k+1+h)^2}) + \frac{5}{9}h)} \right\rfloor \cdot \frac{11}{20} hn \\
&= \left\lfloor \frac{h^2 n}{8(k+1)(k+1+h) + 2h(k+1) + \frac{160}{9}(k+1+h)^2} \right\rfloor \cdot \frac{11}{20} hn \\
&\geq \left\lfloor \frac{h^2 n}{26(k+1+h)^2} \right\rfloor \cdot \frac{11}{20} hn \\
&\geq \left(\frac{h^2 n}{26(k+1+h)^2} - 1 \right) \cdot \frac{11}{20} hn \\
&\geq \left(\frac{h^2 n}{26(k+1+h)^2} - \frac{h^2 n}{52(k+1+h)^2} \right) \frac{11}{20} hn \\
&= \Omega\left(\frac{h^3 n^2}{(k+h)^2} \right).
\end{aligned}$$

Theorem 2.15 *For any deterministic, destination-exchangeable, minimal adaptive routing algorithm on the $n \times n$ mesh with queues of size $k \geq 1$, there exists an h-h routing problem that requires $\Omega(h^3n^2/(k+h)^2)$ steps, for $n \geq 52\cdot(k+1+h)^2/h^2$ and $h \geq 2$, to deliver all of the packets in the problem.*

Note that this bound also applies to dynamic problems where no more than h packets originate from or are destined to a single node and packets are injected deterministically into the network at potentially different times, as long as the time

before a packet is injected does not depend on its full destination address (although it can depend on its profitable directions). In fact, if $h > k$ this dynamic setting would be necessary to accommodate the h packets in the k queue locations of their source node.

2.4.4 Nonminimal Extensions

The techniques of this chapter can also be used to show a lower bound on the time it takes algorithms that are nonminimal, but nearly minimal, to route arbitrary permutations. For example, deterministic destination-exchangeable routing algorithms that allow packets to have nonminimal behavior before reaching the edge of the i-box, but are otherwise minimal, do not contradict the proofs of this chapter. Ben-Aroya and Schuster [BAS94] give a precise characterization of the kinds of nonminimal algorithms for which the technique applies. Although they describe the lower bound in the context of hot potato algorithms, the technique applies more generally to the model described in Section 2.1.

Since the $O(n^{3/2})$ time hot potato algorithm of Bar-Noy *et al.* [BNRST93] is destination-exchangeable, the restriction in Theorem 2.14 of minimal routing cannot be eliminated entirely.

2.4.5 Dimension Order Routing

Destination-exchangeability. The arguments presented in Sections 2.2 and 2.3 also apply to dimension order routing. Even though the paths are fixed in dimension order routers, the algorithms still have the flexibility to have different inqueue and outqueue policies and different ways the nodes keep state. However, because of the regularity in the paths, one can prove an $\Omega(n^2/k)$ bound for routing a worst case permutation in any destination-exchangeable dimension order router as follows.

The construction is similar to that of Section 2.2, except that we consider the westernmost $(1-c)n$ nodes in each of the cn southernmost rows of the mesh. Each of these nodes will send a packet to some node in the northernmost $(1-c)n$ nodes of the cn easternmost columns (see Figure 2.4, left). Define the N_i-column to be the $((1-c)n+i)$-th column of the mesh, and the i-box to be the set of nodes

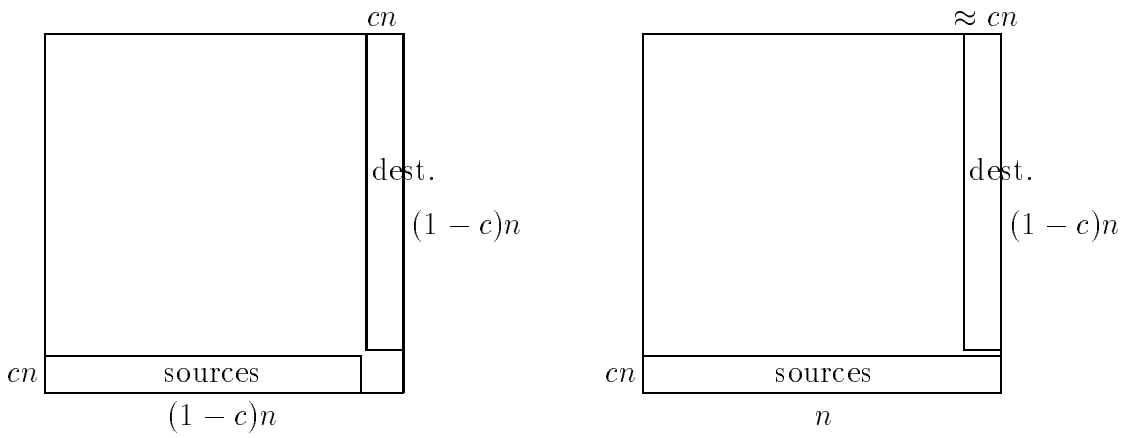

Figure 2.4: The construction for destination-exchangeable dimension order algorithms (left) and farthest-first algorithms (right).

west of and including the N_i-column and south of and including row cn. We define $p = (k+1)cn + dn$, where cn and dn are integers, and $l = (1-c)cn^2/p$. The construction proceeds as before, but there are no E_i-packets and only one exchange rule: for $i \geq 1$ and $j > i$, if an N_j-packet is scheduled by the outqueue policy of a node to enter the N_i-column during steps 1 to idn, then exchange that packet with an N_i-packet in the $(i-1)$-box that is not scheduled to enter the N_i-column.

Following an analysis similar to that of Section 2.3.3, we can find constants $2/(5 \cdot (k+2)) \leq c \leq 1/(2 \cdot (k+2))$ and $2/5 \leq d \leq 1/2$, for $n \geq 10 \cdot (k+2)$. Note that we avoid many of the complications in choosing constants c and d, because we do not need to satisfy $l \leq c^2 n$. We conclude that $\lfloor l \rfloor dn \geq \lfloor 3n/(8 \cdot (k+2)) \rfloor (2n/5) \geq 11n^2/(100 \cdot (k+2))$, for $n \geq 10 \cdot (k+2)$. When $n < 10 \cdot (k+2)$, the diameter bound yields a $2n - 2 = \Omega(n^2/k)$ bound.

For h-h routing with destination-exchangeable dimension order routers, we can find constants $2h/(5 \cdot (k+1+h)) \leq c \leq h/(2 \cdot (k+1+h))$ and $2h/5 \leq d \leq h/2$ for $n \geq 10 \cdot (k+1+h)/h$. For this choice of constants, $\lfloor l \rfloor dn \geq \lfloor 4hn/(15 \cdot (k+1+h)) \rfloor (2hn/5)$, giving a bound of $\Omega(h^2 n^2/(k+h))$ when $n \geq 10 \cdot (k+1+h)/h$.

These analyses are summarized in the following theorem.

Theorem 2.16 *For any deterministic, destination-exchangeable, dimension order routing algorithm on the $n \times n$ mesh with queues of size $k \geq 1$, there exists a*

permutation that requires $\Omega(n^2/k)$ steps to deliver all of the packets in the permutation. Also for such an algorithm, there exists an h-h routing problem that requires $\Omega(h^2n^2/(k+h))$ steps, for $n \geq 10 \cdot (k+1+h)/h$ and $h \geq 2$, to deliver all of the packets in the problem.

Farthest-first Outqueue Policy. The lower bound technique also holds for dimension order routing with a farthest-first outqueue policy, where the next packet to be advanced in a dimension is the one that has the farthest to go in that dimension (ties are broken by FIFO ordering). All other aspects of the routing algorithm — the inqueue policy, the way the state of a node changes, and the way the state of a packet changes — still are restricted to be destination-exchangeable, as in the definition in Section 2.1. Call any algorithm that meets these restrictions a *farthest-first dimension order* algorithm. In this case the lower bound holds even though this algorithm makes use of the entire destination address, and hence is not destination-exchangeable. The dimension order algorithm with farthest-first policy and unbounded queues can route any permutation on the mesh in $2n-2$ steps [Lei92, pages 159–162].

The construction is similar to the one above, except that we define $p = (2k+1)cn + dn$ and $l = cn^2/p$. Also, define the N_i-column to be the $(n+1-i)$-th column and the i-box to be the nodes west of and including the N_i-column and south of and including row cn. Each of the nodes in the southernmost cn rows will send one packet (see Figure 2.4, right).

The N_i-columns are reversed from the original construction, because we want the first packets to leave the i-box to be the ones whose destinations are farthest to the east in their rows. If the columns were arranged in the order of the original proof, then we would not be able to enforce the invariant that for any row, packets are ordered by how far they need to go in that row (see below). This invariant allows us to show that the construction at step t and the algorithm running on the constructed permutation at step t differ by a set of exchanges.

The initial arrangement of packets is one in which no N_i-packet, for $i \geq 2$, is in the N_i-column and for which no N_j-packet is further east in its row than any N_i-packet in that row for $j > i$.

The only exchange rule for the construction is as follows. For $i \geq 1$ and $j > i$, if an N_j-packet is scheduled by the outqueue policy of a node to enter the N_j-column during steps 1 to idn, then exchange that packet with an N_{j-1}-packet in the $(j+1)$-box not scheduled to enter the N_j-column. Furthermore, the N_{j-1}-packet chosen for the exchange is one that is westernmost in its row.

It can be shown that packets are always available for the exchange, that for all i and j with $j > i$, no N_j-packet is further east in its row than any N_i-packet in that row before step idn, and that the construction behaves in the same way as the algorithm does when run on the constructed permutation.

As in the previous analyses, we can find constants $1/(5 \cdot (k+1)) \leq c \leq 1/(4 \cdot (k+1))$ and $2/5 \leq d \leq 1/2$, for $n \geq 20 \cdot (k+1)$. We conclude that $\lfloor l \rfloor dn \geq \lfloor 2n/(9 \cdot (k+1)) \rfloor (2n/5)$, giving a bound of $\Omega(n^2/k)$.

Theorem 2.17 *For any deterministic farthest-first dimension order routing algorithm on the $n \times n$ mesh with queues of size $k \geq 1$, there exists a permutation that requires $\Omega(n^2/k)$ steps to deliver all of the packets in the permutation.*

Nearest-first Outqueue Policy. The nearest-first outqueue policy, where the next packet to be advanced in a dimension is the one that has the nearest to go in that dimension (ties are broken by FIFO ordering), is a plausible outqueue policy because it attempts to deliver messages that are near their destinations by giving priority to them[1]. To complete the description of the routing algorithm, let us suppose that all other aspects of the routing algorithm are destination-exchangeable. Call an algorithm that meets these restrictions a *nearest-first dimension order* algorithm. For the lower bound of nearest-first dimension order algorithms, we define $p = (k+1)cn + dn$. The initial arrangement of packets is the same as for the destination-exchangeable dimension order routers (Figure 2.4, left), and the definition of the N_i-column is as in that case as well.

The only exchange rule for the construction is as follows. For $i \geq 1$ and $j > i$, if an N_j-packet is scheduled by the outqueue policy of a node to enter the N_i-column

[1] This explanation of plausibility is not an exact characterization of the nearest-first algorithm. A packet might be moving east or west near its destination column and receive priority, yet is many nodes away vertically from its destination.

during steps 1 to idn, then exchange that packet with an N_{j-1}-packet in the $(i-1)$-box not scheduled to enter the N_i-column. Furthermore, the N_{j-1}-packet chosen for the exchange is one that is westernmost in its row. Note that if $j \geq i+2$, then the exchange rule could be applied to a packet twice in one step — once to exchange for an offending packet and once because the packet itself has become an offending packet due to the first exchange.

As in the farthest-first construction, it can be shown that packets are always available for the exchange, that for all i and j with $j > i$, no N_j-packet is further east in its row than any N_i-packet in that row before step idn, and that the construction behaves in the same way as the algorithm does when run on the constructed permutation. The analysis for the nearest-first algorithm is the same as for the destination-exchangeable algorithms.

Theorem 2.18 *For any deterministic nearest-first dimension order routing algorithm on the $n \times n$ mesh with queues of size $k \geq 1$, there exists a permutation that requires $\Omega(n^2/k)$ steps to deliver all of the packets in the permutation.*

An Upper Bound for Dimension Order Algorithms. We prove in Theorem 2.19 that the bound for destination-exchangeable dimension order routers given in Theorem 2.16 is tight.

Theorem 2.19 *There is a destination-exchangeable version of the dimension order routing algorithm that routes any permutation on the $n \times n$ mesh in time $O((n^2/k)+n)$, where $k \geq 1$ is the size of the queue.*

Proof: Assume that each node has four incoming queues (labelled North, South, East, and West) as defined in Section 2.4.1, each of size k. The outqueue policy of each node is that packets trying to go straight have priority, resolving ties using FIFO. The inqueue policy is that a packet is always admitted if there is space. Note that these policies are destination-exchangeable.

More precisely, the inqueue policy of North and South queues is always to accept an incoming packet. To see why such a queue has room to accept a packet, note the following. Packets moving straight along a column have priority over turning

packets. It is easy to prove by induction on the distance from the North (South) edge of the mesh that any North (South, respectively) queue will eject a packet in each step that it contains at least one packet, and so it can always accept one.

The inqueue policy of East and West queues is to accept an incoming packet if there are fewer than k packets in the queue at the beginning of the step and to refuse if there are exactly k packets in it at the beginning of the step.

For any fixed row i, define a *turning interval* to begin when an East or West queue at some column j in row i contains k packets, all of which want to turn into column j, and to end when the last of these k packets turns. There are at most n/k turning intervals for row i (since at most n packets ever use East or West queues in the row), so it suffices to show that the time from the beginning of one turning interval to the beginning of the next is $O(n)$. The turning interval itself can last at most n steps: because every North or South queue in column j with at least one packet transmits a packet every step, the n packets destined for column j can delay the k turning packets for at most n steps.

By Lemma 2.21 below, once the turning interval ends there can be at most $3n$ steps until either every packet is in its destination column or another turning interval begins.

Once all packets are in North or South queues in their destination columns, it takes only $2n$ steps before all are delivered. □

We now show that after at most $3(n-1)$ steps after the end of a turning interval, either all packets are in their destination column, or another turning interval has begun, i.e., that there is an East or West queue in some column j that contains k packets, all of which want to turn into column j. Without loss of generality, we can restrict our attention to eastbound packets.

Define a *live packet immediately after step t* to be a packet that has not reached its destination column by the end of step t. A packet is *delayed in step t* if it is live immediately after step $t-1$ and did not cross a link in step t. Also, if q and p are live packets, we will say that *q is east of p* if either q is in a node that is east of the node p is in, or p and q are in the same node (and hence in the same West queue) but q arrived in the node earlier than p did.

If p is a live packet immediately after step t, then let $p'(t)$ be the westernmost live packet immediately after step t that is east of p, if any. If p is a live packet immediately after step t and the distance between p and $p'(t)$ is d, then define

$$d_t(p, p'(t)) = \begin{cases} 2 & \text{, if } d = 0 \\ 1 & \text{, if } d = 1 \\ 0 & \text{, if } d \geq 2 \end{cases} .$$

If p is not live immediately after step t, or $p'(t)$ does not exist (i.e., p is the easternmost live packet in the row), then define $d_t(p, p'(t)) = 0$. The function d_t, loosely speaking, is a measure for how much the packet $p'(t)$ can delay packet p in the future.

Finally, at the beginning of the turning interval, number all eastbound packets (live or not) p_1, p_2, etc. from the east and define

$$\Phi_m(t) = \sum_{\substack{\text{packets } p_i \\ \text{immediately after step } t, \\ i \leq m}} d_t(p_i, p_i'(t)).$$

Note that if p_i is live immediately after step t, then $p'(t) = p_j$ for some $j < i$ (unless p_i is the easternmost live packet in the row). If we define $\Phi_0(t) = 0$, then for $m \geq 1$,

$$\Phi_m(t) = \Phi_{m-1}(t) + d_t(p_m, p_m'(t)).$$

If we can show that Φ_m is bounded, is monotonically decreasing in t, and always decreases if p_m is delayed, and show that p_m is never delayed when $\Phi_m = 0$, then we can bound the time it takes p_m to reach its column destination.

Lemma 2.20 *Let step 0 be the last step of a turning interval, and let step T be the first step of the next turning interval. The following statements hold:*

(1) *For all m, $\Phi_m(0) \leq 2(m-1)$,*

(2) *For all m and $0 \leq t < T$, $\Phi_m(t) \geq 0$,*

(3) *If $\Phi_m(t) = 0$ for some m and $0 \leq t < T$, then no live packet p_i, $i \leq m$, is delayed in step $t' > t$, where $t' < T$,*

(4) *For all m and $0 \leq t < T$, $\Phi_m(t+1) \leq \Phi_m(t)$, and*

(5) *For all m and $0 \leq t < T$, if p_m is delayed in step $t+1$, then $\Phi_m(t+1) < \Phi_m(t)$.*

Proof: Statements (1) and (2) follow easily from the definition of Φ_m.

To see that (3) is true, we need only observe that if $\Phi_m(t) = 0$, then all live packets east of and including p_m are at least two nodes away from each other. Thus, the inqueue policy of the next node east of a live packet will accept it. (At most $k-1$ packets can be in the next node, all waiting to turn, since a new turning interval has not begun.) By induction on t, (3) holds for all $t' > t$.

We now prove (4) and (5) simultaneously by induction on m. Throughout, we use the fact that if p_m is live immediately after steps t and $t+1$, then $p'_m(t+1)$ is always in a node at or east of the node $p'_m(t)$ is in; this implicitly uses the fact that the outqueue policy is FIFO among eastbound packets.

Basis : $m = 1$. $\Phi_1(t) = 0$ for all $0 \leq t < T$, and p_1 is never delayed.

Induction : Suppose the statements are true for all $m' < m (\leq n)$.

Case 1: p_m is not live immediately after step $t+1$ (and hence was not delayed in step $t+1$). Then by the definition of Φ_m and the induction hypothesis,

$$\Phi_m(t+1) = \Phi_{m-1}(t+1) \leq \Phi_{m-1}(t) \leq \Phi_m(t).$$

Case 2: p_m is live immediately after steps t and $t+1$.

Case 2a: p_m was not delayed during step $t+1$. Since p_m can move only one node in one step, then $d_{t+1}(p_m, p'_m(t+1)) \leq d_t(p_m, p'_m(t)) + 1$.

- If $d_{t+1}(p_m, p'_m(t+1)) = d_t(p_m, p'_m(t)) + 1$, then $p'_m(t) = p'_m(t+1)$ was delayed in step $t+1$. We know that $p'_m(t) = p_{m'}$ for some $m' < m$, and so by the induction hypothesis, $\Phi_{m'}(t+1) < \Phi_{m'}(t)$. But $\Phi_{m-1}(t) = \Phi_{m'}(t)$ and $\Phi_{m-1}(t+1) = \Phi_{m'}(t+1)$, because packets p_i, $m - 1 \geq i > m'$, are not

live immediately after steps t and $t+1$. Thus, $\Phi_{m-1}(t+1) = \Phi_{m'}(t+1) < \Phi_{m'}(t) = \Phi_{m-1}(t)$. We have

$$\begin{aligned}
\Phi_m(t+1) &= \Phi_{m-1}(t+1) + d_{t+1}(p_m, p'_m(t+1)) \\
&= \Phi_{m-1}(t+1) + d_t(p_m, p'_m(t)) + 1 \\
&\leq \Phi_{m-1}(t) + d_t(p_m, p'_m(t)) \\
&= \Phi_m(t).
\end{aligned}$$

- If $d_{t+1}(p_m, p'_m(t+1)) \leq d_t(p_m, p'_m(t))$, then

$$\begin{aligned}
\Phi_m(t+1) &= \Phi_{m-1}(t+1) + d_{t+1}(p_m, p'_m(t+1)) \\
&\leq \Phi_{m-1}(t+1) + d_t(p_m, p'_m(t)) \\
&\leq \Phi_{m-1}(t) + d_t(p_m, p'_m(t)) \\
&= \Phi_m(t).
\end{aligned}$$

CASE 2B: p_m was delayed during step $t+1$.

- If $p'_m(t)$ was delayed in step $t+1$, then as in CASE 2A, we have $\Phi_{m-1}(t+1) < \Phi_{m-1}(t)$.

$$\begin{aligned}
\Phi_m(t+1) &= \Phi_{m-1}(t+1) + d_{t+1}(p_m, p'_m(t+1)) \\
&< \Phi_{m-1}(t) + d_{t+1}(p_m, p'_m(t+1)) \\
&\leq \Phi_{m-1}(t) + d_t(p_m, p'_m(t)) \\
&= \Phi_m(t).
\end{aligned}$$

- If $p'_m(t)$ was not delayed in step $t+1$, then $p'_m(t)$ had to be within one node of p_m immediately after step t. Thus,

$$\Phi_m(t+1) = \Phi_{m-1}(t+1) + d_{t+1}(p_m, p'_m(t+1))$$

$$\begin{aligned} &< \Phi_{m-1}(t+1) + d_t(p_m, p'_m(t)) \\ &\leq \Phi_{m-1}(t) + d_t(p_m, p'_m(t)) \\ &= \Phi_m(t). \end{aligned}$$

□

Lemma 2.21 *For any row, once a turning interval ends, there can be at most* $3(n-1)$ *steps until either every packet is in its destination column or another turning interval begins.*

Proof: By Lemma 2.20, the m-th packet from the east can be delayed at most $2(m-1)$ steps. The packet must travel at most $n-1$ nodes. Since $m \leq n$, the lemma follows. □

Chapter 3

A MINIMAL ADAPTIVE ALGORITHM USING CONSTANT SIZED QUEUES

We now present a deterministic, minimal adaptive routing algorithm for the $n \times n$ mesh that routes permutations in $O(n)$ time and uses constant size queues in each node. It uses the distance each packet has to travel in the vertical and horizontal dimensions to make routing decisions and is thus not destination-exchangeable. These same bounds were known for routing algorithms based on sorting [Kun88, LMT89, NS92, RO92], but those algorithms do not use minimal routes.

3.1 The Algorithm

Without loss of generality, we assume that we are routing just packets that need to move either northeast or directly north to get to their destination. The entire algorithm consists of sequential applications of this algorithm, corresponding to the four kinds of packets (NE, NW, SE, SW).

The algorithm operates on submeshes of the mesh and consists of an alternation between vertical phases, where packets move closer to their destinations in the vertical dimension, and horizontal phases, where packets move closer to their destinations in the horizontal dimension. As packets get closer to their destinations, the independent submeshes that can be handled in parallel get smaller and more numerous. We will assume for simplicity that n is a power of 3.

We start with the entire $n \times n$ mesh. In the Vertical Phase, a packet that is at least $\frac{1}{9}n$ rows away from its destination will end the phase at least $\frac{1}{27}n$ rows and at most $\frac{1}{9}n$ rows away from its destination. The Horizontal Phase gives the same guarantees in the horizontal direction.

Throughout the algorithm, we will assume the existence of a set of three tilings

of the $n \times n$ mesh, each with tiles of size $3h \times 3h$, such that any two nodes within distance h of each other vertically and horizontally are both within some tile of at least one of the tilings. Lemma 3.4 shows that such tilings exist.

We now use the three tilings of $\frac{1}{3}n \times \frac{1}{3}n$ ($h = \frac{1}{9}n$) in the next Vertical Phase and Horizontal Phase to route packets to within $\frac{1}{27}n$ rows and $\frac{1}{27}n$ columns of their destinations. A packet participates in the Vertical (Horizontal) Phase if it starts the phase at least $\frac{1}{27}n$ rows (columns, respectively) away from its destination.

We continue in this fashion, routing packets closer to their destinations using successively smaller tiles, until packets are close enough to their destinations to use a dimension order algorithm. The Vertical and Horizontal Phases guarantee that packets never move away from their destinations and that at most a constant number of packets reside in a node at any time.

for $j = 0, 1, 2, \ldots$

- (Base Case.) If $\frac{1}{3^j}n < 27$, then use the dimension order algorithm[1] on the entire $n \times n$ mesh and then exit the algorithm.

- Otherwise, consider the three tilings, where each tile is of size $\frac{1}{3^j}n \times \frac{1}{3^j}n$. For tiles on the edge of the mesh that may not be $\frac{1}{3^j}n \times \frac{1}{3^j}n$, extend the tile to a "virtual tile" of size $\frac{1}{3^j}n \times \frac{1}{3^j}n$, where no packet is moved outside the actual mesh.

  For a given tiling, the actions below are performed on each of the tiles in the tiling independently and in parallel. A packet participates in an action on a given tile only if its current location and destination are both within that tile. Perform the following Vertical Phase for each of the three tilings in succession, followed by the Horizontal Phase for each of the three tilings in succession. In the special case of $j = 0$, there is only one tiling consisting of one $n \times n$ tile.

---

[1] Exactly what dimension order algorithm used here is not important, as long as the outqueue policy always schedules some packet to an outlink if there is any packet that needs to use it, and the inqueue policy always accepts if there is space. The algorithm in the proof of Theorem 2.19 is sufficient.

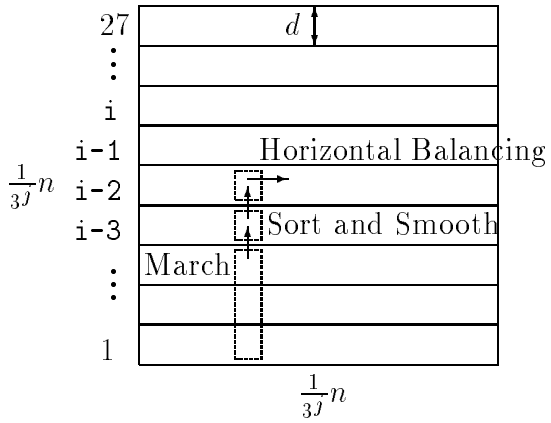

Figure 3.1: The Vertical Phase of the algorithm.

- **Vertical Phase**

  1. Divide each tile of the tiling into 27 horizontal strips of height $d = \frac{1}{27 \cdot 3^J}n$. Define an *active packet* to be one whose destination is in strip i and whose location at the beginning of this Vertical Phase is in one of strips $1, \ldots, i-3$ (i.e., it is at least three strips away from its destination; in particular, packets whose destinations are in strips 1, 2, or 3 are inactive). See Figure 3.1. For each of the following steps, every node knows how long it will take (see Lemmas 3.14, 3.15, and 3.16) and can delay that long before starting the next step.

  2. **March**. An active packet whose destination is in strip i moves to strip $i-3$ via only column edges. Each node in strip $i-3$ transmits packets whose destinations are in strip i as far north within strip $i-3$ as possible. When a node in strip $i-3$ contains $q = 408$ active packets destined for strip i, it refuses to receive any more such packets.

  3. **Sort and Smooth**. This step is performed in two sequential substeps, one for packets whose destinations are in strip i, where i is even, and one for packets whose destinations are in strip i, where i is odd.

     Move active packets from strip $i-3$ to strip $i-2$, using only column edges, in decreasing order according to the horizontal distance they need to go. When each node in the $i-2$ strip (in the column we are considering) has the same number of packets, a new "layer" of packets is added (see

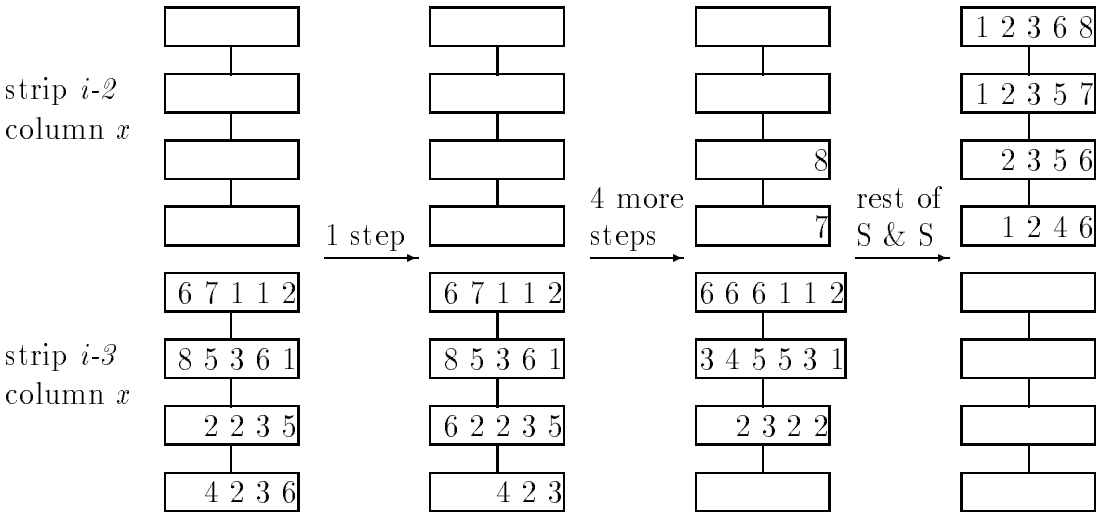

Figure 3.2: Sort and Smooth ($d = 4$). Each box represents a node. Each packet is represented by the horizontal distance to its destination.

Figure 3.2). This is implemented as follows. If a node in strip $i-3$ is the t-th from the southernmost node of the strip, then on steps t and after, the node will transmit north the active packet that has the farthest east to go. Each node in strip $i-2$ can count how many packets it has received to determine whether to hold an incoming packet or send it north: the t-th from the northernmost node of the strip holds every t-th packet it receives.

In Figure 3.2, the packet in the bottommost node that needs to travel six columns is the only packet that moves in the first step. In the fourth step of Sort and Smooth, the "8" moves into strip $i-2$. In the fifth step, the packet that has the next farthest to go (the "7") enters strip $i-2$. The packets will enter strip $i-2$ one at a time in decreasing horizontal distance to go, filling in one layer at a time in strip $i-2$.

4. **Horizontal Balancing**. Each node performs the following operation, called the *2-rule*: if the node has more than two active packets, then it transmits east the active packet that has the farthest east to go (ties broken arbitrarily).

- **Horizontal Phase**

Similar to steps 1 through 4 of the Vertical Phase. (Replace "height" by "width", "north" by "east", etc.)

We will now prove the correctness of the algorithm, place a bound on the queue size, and give the running time of the entire algorithm for permutation routing problems. All of the facts for the Vertical Phase proved in the following subsections easily translate to the corresponding facts for the Horizontal Phase.

3.2 Correctness

We now show that the routing algorithm presented in Section 3.1 is minimal adaptive and delivers all the packets in a permutation. We begin by showing that Horizontal Balancing does not cause any packet to "overshoot" its destination column.

For any column c, define a *($\leq c$)-packet* to be an active packet whose destination column is at or west of column c. Define a *($> c$)-packet* to be any other active packet (i.e., one whose destination is east of column c).

Lemma 3.1 *For any column c, any row r, and any $s \geq 1$, immediately after Sort and Smooth, there are at most $2s$ ($\leq c$)-packets in the first s nodes of r that are west of and including column c.*

Proof: Suppose there were at least $2s+1$ ($\leq c$)-packets in the first s nodes west of and including column c at the end of Sort and Smooth. Let $x_1, x_2, \ldots, x_s$ be the respective numbers of ($\leq c$)-packets in those nodes.

Then there are at least $((x_1 - 1) + (x_2 - 1) + \ldots + (x_s - 1)) \cdot d$ ($\leq c$)-packets in the rectangle consisting of the first s columns of strip $i-2$ west of and including column c, each destined for nodes in the corresponding rectangle in strip i. This is because in order for a node ν to have x ($\leq c$)-packets at the end of Sort and Smooth, there must be at least $x - 1$ ($\leq c$)-packets in each node of ν's column within its strip. (See Figure 3.2.)

But $x_1 + x_2 + \ldots + x_s \geq 2s+1$. Thus, there are at least $(2s+1-s) \cdot d = (s+1) \cdot d$ packets destined for sd nodes, contradicting the fact that this is a permutation routing problem. □

For any row r and column c, we will say that *Condition C(r, c)* holds at the beginning of some step t, where the first step of Horizontal Balancing corresponds to $t = 1$, if for all $s \geq 1$ there are no more than $2s$ ($\leq c$)-packets in the first s nodes of row r west of and including column c.

Lemma 3.2 *For all $t \geq 0$, for all rows r and columns c, no ($\leq c$)-packet is transmitted east by the node in column c during step t of Horizontal Balancing, and Condition C(r, c) holds at the beginning of step $t + 1$.*

Proof: The proof is by induction on t.

Basis : $t = 0$. The first part of the lemma is vacuously true, since there is no step 0 of Horizontal Balancing. The second part of the lemma follows immediately from Lemma 3.1.

Induction : Fix an r and c. Assume the statement is true for $t - 1 \geq 0$ (in particular, Condition C(r, c) holds at the beginning of step t). We will prove the statement for t. During step t, no ($\leq c$)-packet is transmitted east by the node ν in column c, because if there were, then by the 2-rule ν would have had at least three ($\leq c$)-packets at the beginning of step t, violating Condition C(r, c) for $s = 1$ at step t. (Recall that the 2-rule prevents any node from transmitting a ($\leq c$)-packet in preference to a ($> c$)-packet.)

Now suppose that Condition C(r, c) does not hold for some s' at the beginning of step $t + 1$. Then the first s' nodes west of and including ν contained $2s'$ ($\leq c$)-packets at the beginning of step t and received a ($\leq c$)-packet from the $(s' + 1)$-st node during step t. But this means that the $(s' + 1)$-st node had at least three ($\leq c$)-packets at the beginning of step t. Thus, there were at least $2s' + 3$ ($\leq c$)-packets in the first $s' + 1$ nodes west of and including ν, violating Condition C(r, c) for $s = s' + 1$ at step t.

Since we chose r and c arbitrarily, we have proved Lemma 3.2 for all r and c, and hence for all packets. □

Lemma 3.3 *Suppose that, at the beginning of the Vertical Phase, every packet is within $27d$ rows of its destination row. Then at the end of the Vertical Phase,*

every active packet is at least $d+1$ and at most $3d-1$ rows away from its destination row. Every inactive packet is at most $3d-1$ rows away from its destination row.

Proof: Any active packet destined for the i-th strip will end the phase in the strip $i-2$. Thus, every active packet is at least $d+1$ and at most $3d-1$ rows away from its destination row. Any inactive packet is within three strips (i.e., at most $3d-1$ rows) of its destination row. □

The following tiling lemma is folklore:

Lemma 3.4 *There exist three tilings of the $n \times n$ mesh with tiles that are $9d \times 9d$ such that any two nodes within distance $3d$ in both the vertical and horizontal dimensions are contained in some tile of at least one of the tilings.*

Proof: Define the three tilings as follows. The north (and west) boundaries of the tiles in the first tiling are nodes in row (respectively, column) i, where $i \equiv 1 \pmod{9d}$. The tiles of the second tiling are displaced $3d$ rows east and $3d$ columns south from the tiles in the first tiling. The tiles in the third tiling are displaced $3d$ rows east and $3d$ columns south from the tiles in the second tiling. It is easy to see that any two nodes within $3d$ rows and $3d$ columns of each other must be contained in the same tile in one of the tilings, as follows.

It suffices to show that if two nodes within $3d$ rows and $3d$ columns are not in a tile of either the first or second tiling, then they must both be in a tile of the third tiling.

If the two nodes are not in a tile of the first tiling, then either

(1) There is an $i \geq 1$ such that one of the two nodes is west of column $9di+1$ and the other is at or east of that column, or

(2) There is an $i \geq 1$ such that one of the two nodes is north of row $9di+1$ and the other is at or south of that row.

Similarly, if the two nodes are not in a tile of the second tiling, then either

(3) There is a $j \geq 0$ such that one of the two nodes is west of column $9dj + 3d + 1$ and the other is at or east of that column, or

(4) There is a $j \geq 0$ such that one of the two nodes is north of row $9dj + 3d + 1$ and the other is at or south of that row.

Since the two nodes are within $3d$ rows and $3d$ columns of each other, (1) and (3) cannot both hold, and (2) and (4) cannot both hold.

If (1) holds, then both nodes are at or east of column $9d(i-1) + 6d + 1$ and west of column $9di + 3d + 1$. If (4) holds, then both nodes are at or south of row $9dj + 1$ and north of row $9dj + 6d + 1$. Thus, if (1) and (4) hold, then both nodes are contained in a tile of the third tiling. A similar argument holds if (2) and (3) are true. □

Theorem 3.5 *No packet makes a move that places it farther from its destination, and all packets eventually are delivered. That is, the algorithm above is minimal adaptive.*

Proof: During March and Sort and Smooth, packets move only towards their destination vertically. Lemma 3.2 ensures that a packet does not move away from its destination horizontally. Along with the analogous lemma for the Horizontal Phase, this shows that no packet makes a move that places it farther from its destination.

Using induction on j and Lemmas 3.3 and 3.4 (and the horizontal analogue of Lemma 3.3), we see that every packet is no more than $3 \cdot \frac{1}{27 \cdot 3^j} n$ rows and $3 \cdot \frac{1}{27 \cdot 3^j} n$ columns away from its destination after the j-th iteration.

Basis : $j = 0$. Every packet is within $3d = 3 \cdot \frac{1}{27 \cdot 3^0} n$ rows and $3d = 3 \cdot \frac{1}{27 \cdot 3^0} n$ columns of its destination at the end of the 0-th iteration by Lemma 3.3.

Induction : Suppose the claim is true for $j - 1 < \log_3 n - 3$. By the induction hypothesis, after the $(j-1)$-th iteration, all packets are within $3(\frac{1}{27 \cdot 3^{j-1}} n)$ rows and $3(\frac{1}{27 \cdot 3^{j-1}} n)$ columns of their destinations. By Lemma 3.4, the packet and its destination are both within some tile of one of the three tilings of the j-th iteration. (Note that the tiles are $\frac{1}{3^j} n \times \frac{1}{3^j} n = 9\frac{1}{27 \cdot 3^{j-1}} n \times 9\frac{1}{27 \cdot 3^{j-1}} n = 27(\frac{1}{27 \cdot 3^j}) n \times 27(\frac{1}{27 \cdot 3^j}) n$.)

Thus, by Lemma 3.3, every packet will end the j-th iteration at most $3(\frac{1}{27\cdot 3^j}n)$ rows and $3(\frac{1}{27\cdot 3^j}n)$ columns away from its destination.

This ends the proof by induction.

When $\frac{1}{3^j}n < 27$, then the dimension order algorithm is used. Thus, every packet is eventually delivered. □

3.3 Queue Size

We now show that during the algorithm, no more than a constant number of packets ever occupy a node at the same time. We do this by examining the queue size during and at the end of each step in a Vertical Phase. In what follows, let $q = 17\cdot(27-3) = 408$.

Lemma 3.6 *Suppose no node begins the March with more than 17 packets. No more than $q+1$ active packets ever occupy a node at the same time during the March. At the end of the March, no node contains more than q active packets.*

Proof: Consider any node ν in strip $i-3$. Let t be the step after which ν's north neighbor refuses to accept more packets destined for strip i (or $t=0$, if ν is the northernmost node in strip $i-3$). Until time t, ν has no more than 17 active packets, since it sends one north at each step.

After time t, whenever ν has a packet destined for a strip north of strip i, it transmits one such packet. It may accumulate an additional q packets that end the march at ν. Therefore, ν never has more than $q+1$ active packets at any time (q packets destined for strip i and one packet destined for a strip north of strip i). □

Lemma 3.7 *During Sort and Smooth, no more than $2q+1$ active packets ever occupy a node at the same time. At the end of Sort and Smooth, no node contains more than q active packets.*

Proof: Consider any strip i.

Each node in strip $i-3$ receives at most one packet before it starts transmitting packets northward. Since a node in strip $i-3$ always transmits once it starts

transmitting (and has at least one packet destined for strip i), then its queue will never hold more than $q+1$ active packets destined for strip i. If i is odd, it may contain an additional q active packets destined for strip $i-1$ that completed their Sort and Smooth in the even substep.

A node in strip $i-2$ will never contain more than q active packets destined for strip i. If i is even, it may contain an additional q active packets destined for strip $i+1$ that will move in the odd substep.

At the end of Sort and Smooth, each node in strip $i-2$ will contain no more than q active packets destined for strip i and no other active packets. □

Lemma 3.8 *If a node contains no more than $r > 2$ active packets at the beginning of Horizontal Balancing, then the node contains no more than r active packets during Horizontal Balancing. Also, if a node contains no more than two active packets at the beginning of Horizontal Balancing, then the node never contains more than three active packets during Horizontal Balancing.*

Proof: Because of the 2-rule, any node that has $r > 2$ active packets transmits one active packet to the east until it has two active packets. (It might later receive a packet, but then the 2-rule is in effect again.) Since the node is always transmitting when it has three or more active packets and can only receive at most one packet per step, then the number of packets in the node can never increase when it has three or more active packets. In particular, the node can never have more than r active packets.

A node that begins with two or fewer packets can receive packets until it has three packets, at which point it will start transmitting. As above, it can never hold more than three active packets. □

Lemma 3.9 *No more than two active packets end Horizontal Balancing in the same node.*

Proof: The 2-rule ensures this. □

In what follows, a Vertical Phase is divided into three subphases, one for each of the three tilings. Horizontal subphases are defined analogously. We will now bound

V1 V2 V3 H1 H2 H3 V1 V2 V3

Figure 3.3: Subphases of the algorithm. A packet can remain inactive for at most seven subphases.

the number of inactive packets that occupy a node at any given time by bounding the number of subphases a packet can remain inactive.

Lemma 3.10 *If a packet is active in some vertical subphase at iteration j of the algorithm, then it will be active in some horizontal subphase at iteration j or in some vertical subphase at iteration $j+1$.*

Proof: Let d be the height of a strip in iteration j. Since the size of a strip in iteration $j+1$ is $d/3$, then a packet at least $d+1$ rows away from its destination row at the end of the current iteration will be at least three strips away (vertically) at the beginning of the next iteration.

Since the packet is at least $d+1$ and at most $3d-1$ rows away from its destination row, then by Lemma 3.4 it will be an active packet in some tile of size $9d \times 9d$ in one of the tilings in the $(j+1)$-st iteration, if it did not move in an intervening horizontal subphase. (It could move vertically during Vertical Balancing of a horizontal subphase.) The $d+1$ and $3d-1$ distance bounds are guaranteed by Lemma 3.3. Thus, an active packet of the j-th iteration will move in one of the three vertical subphases in the $(j+1)$-st iteration if it was not active in the horizontal subphases of iteration j. □

Corollary 3.11 *Once a packet becomes active in some subphase, it can occupy space without moving (i.e., is inactive) in at most seven subphases between subphases in which it is active.*

Proof: Follows from Lemma 3.10 (and the corresponding horizontal lemma) and observing the sequence of subphases in the algorithm. See Figure 3.3. □

Corollary 3.12 *At the end of any vertical subphase (or horizontal subphase), no more than 17 packets occupy any node.*

Proof: From Lemma 3.9, at most two active packets from any subphase (vertical or horizontal) can occupy a node at the end of that subphase. From Corollary 3.11, only eight subphases' worth of active packets can occupy a node, plus the one packet that began in the node. Thus, at most 17 packets end any subphase in the same node. □

Lemma 3.13 *No more than* $2q + 18 = 834$ *packets ever occupy a node at the same time.*

Proof: The lemma follows from Lemmas 3.6, 3.7, 3.8, 3.9, and Corollary 3.12 by induction on the subphase number. Up to $2q + 1$ active packets and 17 inactive packets can occupy a node at the same time.

For the dimension order part of the algorithm (the base case), consider any node ν. By Lemma 3.3, no packet is farther than two rows and two columns from its destination at the beginning of this part of the algorithm. Thus, the only northeast bound packets that can enter ν are those whose destinations are within two rows north or two columns east of ν. There are nine such destinations and hence nine such packets because this is a permutation routing problem. This gives a bound of nine on the queue size during the dimension order part of the algorithm. □

3.4 Time Analysis

We now present the running time analysis of the algorithm by calculating the running time of each step of the Vertical Phase.

Lemma 3.14 *The March takes no more than* $qd - 1$ *steps.*

Proof: Assume for the sake of time analysis that during the March, whenever a node contains two or more packets that need to move northward, it prefers to send the one that was received from the south on the previous step. Otherwise, it makes an arbitrary choice. Note that because of this priority scheme in the March, once a packet starts moving, it continues to move uninterrupted until it reaches the node in which it will end the March.

By Corollary 3.12, at most 17 packets occupy a node at the beginning of the March. Suppose an active packet p is delayed by t steps. Since each delaying packet had to occupy a node at or south of p's node in the same column, then there are at least $(t-16)/17$ nodes south of it. The distance p travels in the March, then, is at most $d(27-3)-1-((t-16)/17)$.

The total number of steps before p reaches the node in which it ends the March is then at most $t+d(27-3)-1-((t-16)/17) = \frac{16}{17}t - \frac{1}{17} + d(27-3)$. This quantity is maximized when t is maximized. Since an active packet's destination is at least three strips away from its node at the beginning of the March, then t can be at most $17 \cdot d(27-3) - 1 = qd - 1$. Thus, the total number of steps before p reaches the node in which it ends the March is at most $\frac{16}{17}(qd-1) - \frac{1}{17} + \frac{1}{17}qd = qd - 1$. □

Lemma 3.15 *Sort and Smooth takes no more than* $2 \cdot ((d-1)+qd)$ *steps.*

Proof: Consider the even substep of Sort and Smooth. The analysis for the odd substep is identical.

Let P be the number of active packets in a column of strip $i-3$ destined for strip i, and let $P = sd + r$, where s and r are integers and $0 \leq r < d$.

It will take $d-1$ steps before the first packet moves from strip $i-3$ to $i-2$. The northernmost node of strip $i-3$ will then send a packet northward each step until there are no more packets to send, which will take $sd+r$ steps. Finally, the last packet to enter strip $i-2$ will move, uninterrupted, an additional $d-r$ nodes, if $r \geq 1$. If $r = 0$, then the packet will not have to move any further in strip $i-2$.

Thus, the even substep takes $(d-1)+(sd+r)+(d-r)$ steps if $r \geq 1$ and $(d-1)+(sd+r)$ steps if $r = 0$. If $P < qd$, then $sd < qd$, and so the substep takes no more than $(d-1)+(sd+r)+(d-r) \leq (d-1)+(s+1)d \leq (d-1)+qd$ steps. If $P = qd$, then $sd = qd$ and $r = 0$, and so the substep takes no more than $(d-1)+qd$ steps. □

Lemma 3.16 *Horizontal Balancing takes no more than* $3h-4$ *steps on an* $h \times h$ *tile.*

Proof: Any node with at least four packets at the end of step t of Horizontal Balancing had at least four packets at the beginning of each of steps $1, \ldots, t$ and therefore transmitted in steps $1, \ldots, t$. This is because if the node had three packets at some step $t' < t$ and four packets at step t, then it received a packet without sending one, violating the 2-rule.

Let M_t be the maximum, over all nodes in a single row r, of the number of packets in the node at the end of step t. There are at most $2h$ active packets in row r by Lemma 3.2. Thus, for all time steps t for which $M_t \geq 4$, $t + M_t \leq 2h$, since the node with M_t packets also transmitted t other packets in earlier steps. Thus, $M_{2h-3} \leq 3$.

Let t^* be the first step for which $M_{t^*} \leq 3$. Then at the end of step $t^* + i$, the leftmost i nodes of row r each have no more than two packets, for $i = 1, \ldots, h-1$. This is proved by induction on i.

Basis : $i = 1$. The leftmost node will have at most two packets after one step, since it obeys the 2-rule, it started step t^* with no more than three packets, and it did not receive any packets.

Induction : Assume the statement is true for $i = m - 1$. Then the leftmost $m - 1$ nodes each have at most two packets. Thus, the m-th node from the left will not receive a packet and will have at most two packets at the end of step $t^* + m$, since it obeys the 2-rule, and it had no more than three packets at the beginning of step $t^* + m$.

This ends the proof by induction.

Thus, after $t^* + h - 1$ steps, the leftmost $h - 1$ nodes have no more than two packets each. Also, we know that the rightmost node never has more than two packets (Lemma 3.2, where c is the rightmost node). Therefore, after at most $(2h - 3) + (h - 1) = 3h - 4$ steps, all nodes have no more than two packets. □

Lemma 3.17 *The dimension order part of the algorithm takes no more than 16 steps.*

Proof: Consider a packet p and the set of packets that can delay p in the dimension order part of the algorithm. We know that each packet is within two

rows and two columns of its destination (by Lemma 3.3, where $d = 1$). There are at most nine destination nodes other than p's destination node that could have a northeast bound packet π destined for it such that π takes a path that interferes (i.e., shares an outlink) with p's path.

More specifically, for each East outlink p uses, at most five other packets could also use it, and for each North outlink p uses, at most one other packet could also use it. Since nodes always transmit a packet along an outlink if there is one that needs to use that outlink, p need only wait six steps before being transmitted along an East outlink and two steps before being transmitted along a North outlink. Since p's destination is at most two rows and two columns away from where it started the dimension order part of the algorithm, then p will arrive at its destination in at most $2 \cdot 6 + 2 \cdot 2 = 16$ steps. □

Lemma 3.18 *The entire algorithm (including handling the four different types of packets) takes no more than* $4 \cdot 243n$ *steps.*

Proof: Let J be the number of iterations in the algorithm.

From Lemmas 3.14, 3.15, 3.16, and 3.17, the time to route just NE packets can be expressed by the following summation. (The factor of 6 is for the three tilings of each of the Vertical and Horizontal Phases. There is a factor of only 2 when $j = 0$.) The value d_j is the value of d during the j-th iteration of the algorithm.

$$\begin{aligned}
T(n) &\leq 2 \cdot ((qd_0 - 1) + 2 \cdot ((d_0 - 1) + qd_0) + (3n - 4)) + \\
&\quad \sum_{j=1}^{J} \left(6 \cdot \left((qd_j - 1) + 2 \cdot ((d_j - 1) + qd_j) + (3 \cdot \frac{1}{3^j} n - 4)\right)\right) + 16 \\
&= 2 \cdot (3qd_0 + 2d_0 - 7 + 3n) + 6 \cdot \sum_{j=1}^{J} \left(3qd_j + 2d_j - 7 + 3 \cdot \frac{1}{3^j} n\right) + 16 \\
&< 2 \cdot \left(\frac{1307}{27} n\right) + 6 \cdot \sum_{j=1}^{J} \left(\frac{1307}{27} \frac{1}{3^j} n\right) \\
&< \frac{2614}{27} n + \frac{2614}{9} n \cdot \sum_{j=1}^{\infty} \left(\frac{1}{3^j}\right)
\end{aligned}$$

The value of the last expression is less than $243n$. Since we have four different kinds of packets, we must multiply the bound by four, obtaining the upper bound on the running time of the entire algorithm. □

We now have shown that there is a minimal adaptive algorithm that runs in $O(n)$ time and uses $O(1)$ size queues in each node:

Theorem 3.19 *There exists a deterministic, minimal adaptive routing algorithm that routes any permutation in* $972n$ *steps and uses space for at most* 834 *packets in any node.*

Proof: The theorem follows from Lemmas 3.5, 3.13, and 3.18. □

We can improve the time bound by observing that at the beginning of the j-th iteration, for $j \geq 1$, active packets are within nine strips (that is, $\frac{1}{3 \cdot 3^j} n$ rows) of their destinations. We can now restate Lemmas 3.14 and 3.15 with $q = 17 \cdot (9-3) = 102$, reducing the time for iterations $j \geq 1$ by a factor of almost four. The new time bound is $564n$.

Note also that the queue size for iterations $j \geq 1$ is never more than $2q + 18 = 222$. (Lemmas 3.6, 3.7, and 3.13 need to be restated with the new value of q.)

Chapter 4

EXPERIMENTS ON DESTINATION-EXCHANGEABLE, NONMINIMAL ADAPTIVE ALGORITHMS

Generally speaking, it is difficult to obtain good analytic bounds on adaptive or randomized packet routing algorithms, because the interactions of packets when they are queued at a node are more difficult to predict in advance. Simulating these algorithms is often the only way to get information for how these algorithms behave.

However, the key fact about the lower bound of Chapter 2 is that it is constructive: given an algorithm in the class for which the lower bound applies, one can build a permutation that requires the algorithm $\Omega(n^2/k^2)$ time to route. It is plausible that the worst case permutations constructed for minimal adaptive algorithms are also bad for nonminimal algorithms. We can explore the difference between minimal adaptive algorithms and nonminimal adaptive algorithms in the worst case setting by running permutations constructed in the lower bound (hereafter called *CLT permutations*) on nonminimal adaptive algorithms.

Chaotic routing [BFS94, KS91, KS94] is a randomized, nonminimal adaptive algorithm that is competitive with state-of-the-art oblivious routers. Recall that a packet is said to be *derouted* if it makes a move that places it farther from its destination. In the Chaos algorithm, a node deroutes packets when it becomes congested, randomly picking which packet to deroute among the packets it contains. Areas of local congestion dissipate via this diffusion mechanism. More details about the algorithm will be given in Section 4.1.

The question we wish to answer is: How well does Chaos perform on worst case permutations? Our method was as follows. We first removed all of the derouting logic from the Chaos simulator, which also removed all sources of randomness in the algorithm. Call the resulting routing algorithm *minimal Chaos*. Minimal Chaos

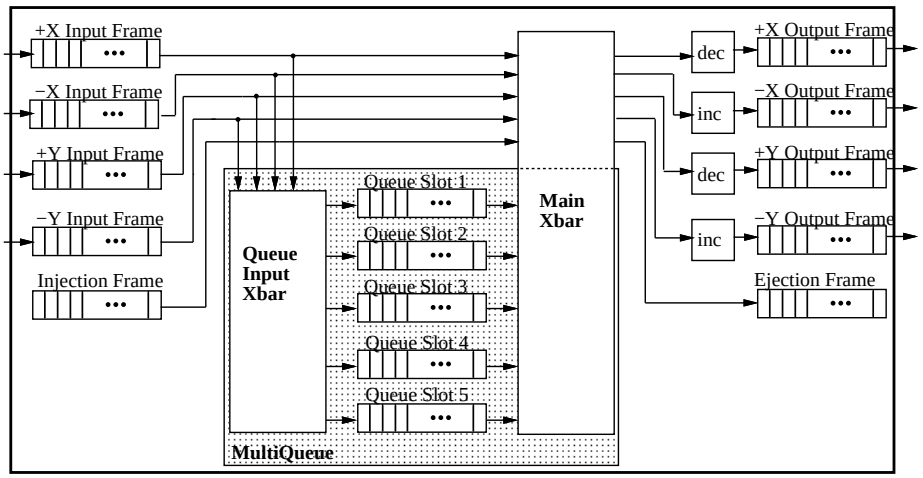

Figure 4.1: A Chaos router node.

is deterministic, destination-exchangeable, and minimal adaptive. We constructed the CLT permutation for minimal Chaos and then ran the unaltered Chaos on that permutation. The difference in running times of the two algorithms is a measure of how effective the derouting mechanism is in Chaos. By running the experiments over a range of mesh sizes, we observed that the running time of Chaos on the CLT permutations on the $n \times n$ mesh appears to be superlinear in n.

This chapter also describes a similar experiment for a randomized, destination-exchangeable hot potato algorithm. The running time of this algorithm also appears to be superlinear in n.

4.1 The Chaos Router

Figure 4.1 (from [BS92]) shows the block diagram of a node of the Chaos router. A more detailed description of its operation can be found elsewhere [Bol93, KS94], but we briefly describe it here.

A node consists of four *input frames* and four *output frames*, one for each of the four neighbors of a node. Input and output frames can be paired by the associated neighboring nodes. For example, in Figure 4.1, the $+X$ Input Frame would be paired with the $-X$ Output Frame. Each frame can hold exactly as many words as there are in one packet. A *word* can be defined as the amount of data that can

be transmitted between two adjacent nodes in one cycle. In addition to its four pairs of input and output frames, each node has a *central queue* (also called the *multiqueue*) that can hold up to five packets. There is also a mechanism to inject packets into a node and to deliver packets whose destination is that node.

The Chaos algorithm operates at each node as follows. The node examines the output frames in a specified order, and this is repeated indefinitely. If an output frame is *interesting*, the node must make a routing decision. If the multiqueue is not full, then an output frame is interesting if it does not contain a packet and either (1) any of the packets in the multiqueue or any of the packets in the input frames can profitably use the output frame (i.e., can move closer to its destination by crossing the associated wire), or (2) there is a packet in the output frame's paired input frame, and it has been there for longer than a threshold amount of time. In case (1), one of the packets is moved into the output frame; packets in the multiqueue have priority over packets in the input frames. Furthermore, if there is a packet waiting in the input frame of the same direction as the output frame, then it is moved into the multiqueue. In case (2), the packet is moved into the multiqueue.

If the multiqueue is full, then all empty output frames are interesting. In this case, if there is a packet in the multiqueue that can use the output frame profitably, then it is moved into the output frame. Otherwise, one of the five packets in the multiqueue is chosen randomly and moved into the output frame. As in the case where the multiqueue was not full, if there is a packet in the output frame's paired input frame, then it is moved into the multiqueue. No packet from an input frame is allowed to move directly into an output frame when the multiqueue is full.

The packet that is chosen randomly is derouted, since it could not profitably use the output frame. There is nothing to prevent a packet from being derouted frequently, resulting in a livelock situation. However, the Chaos algorithm is *probabilistically livelock-free*: the probability that a packet has not been delivered after T time tends to zero as T tends to infinity. (A proof of this can be found in [KS90] for the hypercube; the proof for the mesh is almost identical [Bol93].)

The Chaos router has a mechanism called *virtual cut-through*, which allows the head of a packet to move from frame to frame (either the input frames, the output

frames, or the central queue) without waiting for the tail to arrive in the same buffer space. The router also has a mechanism called *multiqueue bypass*, which allows a packet in an input frame to move directly to an output frame if no packet in the multiqueue can profitably use the output frame and the multiqueue is not full (case (1) above).

In a lightly loaded situation, a packet can take advantage of both virtual cut-through and multiqueue bypass. The header of a packet that enters an input frame at the beginning of cycle t can be in an output frame as early as the beginning of cycle $t + 3$. Since it takes one more cycle for the header to be transmitted to the next node, a header can reside in a node for as few as four cycles.

In heavily loaded situations, packets can enter an output frame at a rate of one every $s + 3$ cycles, where s is the size of a packet in words. The three extra cycles are needed to decide which packet is allowed to enter the output frame next.

In the experiments we describe in Section 4.2, all packets consisted of 10 words. Thus, a wire will transmit a packet every 13 cycles in heavily loaded situations, and a packet header can advance one node every four cycles in lightly loaded situations.

The minimal Chaos router works exactly as the Chaos router does, except that the derouting mechanism (and hence all sources of randomness in the algorithm) is disabled. If a packet is in an input frame, its associated output frame is empty, and the central queue is full, then no packet is derouted to make space. The incoming packet must wait until space is free in the central queue as packets are moved profitably through the output frames. It is not hard to see that minimal Chaos is in fact a deterministic, destination-exchangeable, minimal adaptive algorithm.

In a general routing situation, minimal Chaos would not be deadlock-free, but in the experiments we performed, all packets have destinations south and east of their starting node (and hence no packet needs to move north or west). Thus, no cyclic dependencies for resources can exist, and so the network never gets into a deadlocked situation.

Minimal Chaos is, in a sense, an idealized minimal adaptive algorithm, since it has no mechanism to prevent deadlock. Mechanisms to prevent deadlock, such as virtual channels [DA93] or that of Cypher and Gravano [CG92], complicate the logic needed to implement the algorithm.

4.2 The Experiments

We now describe the experiments we performed on the minimal Chaos and Chaos routers.

4.2.1 Experiment 1

Minimal Chaos falls into the class of algorithms in the lower bound of Chapter 2: it is deterministic and minimal adaptive, routing decisions are based only on local information, and the only parts of the destinations of packets used to make routing decisions are their profitable directions. In our first experiment, we built the CLT permutation for minimal Chaos. This permutation is constructed by injecting one packet into each of the nodes in a $cn \times cn$ corner of the mesh (where c is a constant[1]); these packets have destinations in one of several rows or columns just outside the corner (see Figure 4.2; note that, unlike in Chapter 2, packets start in the northwest corner of the mesh, rather than the southwest corner). A hot spot develops at the intersection of the destinations, since all packets in the permutation pass through that area.

The time it takes the last packet to reach its destination conceptually can be thought of as the sum of two components: the time it takes for the packet to escape the congestion at the hot spot and the time it takes the packet to reach its destination once it has escaped the hot spot. In the lower bound of Chapter 2, the *congestion component* on the $n \times n$ mesh grows quadratically with n for the last packet. The other component, what we will call the *distance component*, grows linearly with n.

In order to isolate the congestion component, we altered the CLT permutation so that the last twelve packets to be delivered were ones whose destinations were at the outer edge of the mesh. Running the altered permutation on minimal Chaos

[1] The constant c, about $1/17$, was calculated by letting $k = 8$ and using a finer analysis than that of Section 2.3.3, where we choose $c = 7/(12 \cdot (k+2))$. The choice of $k = 8$ uses the fact that of the 13 queue positions in a node on the south or east edge of the $cn \times cn$ corner, four of them are never used in the CLT permutation because they would send a packet in a nonprofitable direction, and one of them is never used during the relevant *idn* steps of the construction, by Lemmas 2.7 and 2.8.

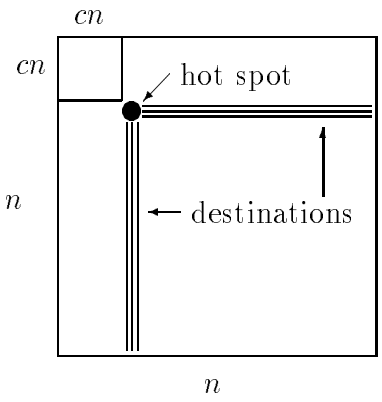

Figure 4.2: Generating the CLT permutation.

confirmed that the last packets to be delivered were near the edge of the mesh. Thus, we were able to ensure that the distance component grew linearly (roughly $4 \cdot (n - cn)$, since there is little congestion beyond the hot spot), allowing us to observe how the congestion component grew.

We then ran the Chaos algorithm on this permutation. The distinction between the congestion component and the distance component was not as clear in the Chaos case because there was no guarantee that the last packets to be delivered in the Chaos experiments were near the edge of the mesh. However, we discovered empirically that in fact, of the last packets to be delivered, at least one of them always had a destination near the edge of the mesh.

The results of the experiment are shown in Figure 4.3. The top curve represents the performance of minimal Chaos on the altered permutation, and the bottom curve represents the performance of Chaos on the same permutation. As predicted by the lower bound of Chapter 2, the congestion component of minimal Chaos grows quadratically with n and dominates the distance component. The results for Chaos, however, are inconclusive: it is difficult to estimate what the asymptotic behavior of the bottom curve of Figure 4.3 is.

The experiment described above began with each node examining the south direction. In an asynchronous environment, the directions nodes are examining when packets are injected are unpredictable. We ran the Chaos algorithm on the CLT permutation with each node initially examining a random direction in order

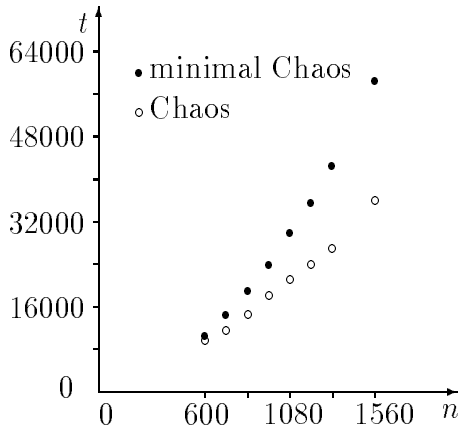

Figure 4.3: Results of Experiment 1. The top curve (filled circles) represents minimal Chaos, and the bottom curve (open circles) represents Chaos.

to isolate the effect of the initial state of the nodes. Chaos delivered the packets faster in the random initial state experiments than in the "all examining south" experiments, which is to be expected, since the constructed permutation was built assuming the "all examining south" initial state. However, in all of the five problem sizes for which we did this "random" experiment, Chaos performed no better than 4% better than the results in Figure 4.3.

The Chaos simulator has an animation package that allows us to observe the behavior of the algorithms by coloring nodes according to how many packets there are in them (see Figure 4.4). Using the animations, we observed that beyond some small distance from the hot spot, packets advance virtually uninterrupted (i.e., at roughly one node every four cycles). This provided the motivation for Experiment 2, which concentrated on the congestion component of delivery time.

4.2.2 Experiment 2

The mesh sizes in Experiment 1 were large enough so that the machines we used to simulate the Chaos algorithm on them had just enough memory to do so. However, from the animations of Experiment 1, we discovered not only that packets advance virtually uninterrupted after they escape the hot spot, but also that packets destined for the same row or column "fall into line," so that, informally speaking, nothing interesting happens beyond the hot spot. This observation allowed us to simulate

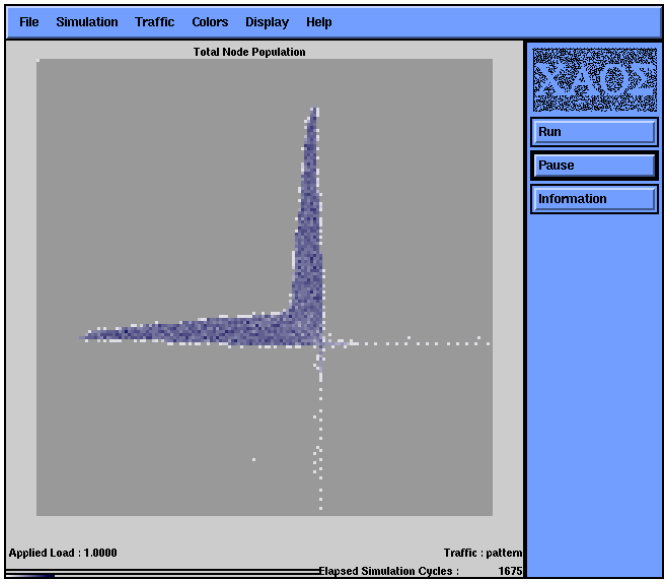

Figure 4.4: A snapshot of the 150×150 northwest corner of the 1560×1560 mesh. The snapshot is of the CLT permutation running on Chaos. Dark areas are heavily congested, whereas white nodes have just one packet in them.

just a small portion of the mesh (i.e., the relevant corner of the mesh) and still observe the same behavior near the hot spot. Since we were simulating a much smaller part of the mesh, we had enough memory in our machines to simulate much larger meshes in Experiment 2 than in Experiment 1. Figure 4.4 shows a typical state of the network in the 150×150 corner of the 1560×1560 mesh running the CLT permutation on Chaos.

Experiment 2, then, is identical to Experiment 1, except that only a $2cn \times 2cn$ part of the mesh was simulated. When a packet reaches the edge of the submesh, it immediately is removed from the network. Everything else about the experiment is the same: we built the CLT permutation on minimal Chaos with southward initial states and then ran the permutation on Chaos. Figure 4.5 shows the results of Experiment 2, which were consistent with the results in Experiment 1.

The curve for Chaos closely fits the curve defined by $4cn + 0.439n^{1.509}$, also plotted in Figure 4.5. We arrived at this function by first subtracting an estimate of the distance component, $4 \cdot cn$, from the observed data. The congestion component was estimated to be the number of cycles observed minus this estimate of the

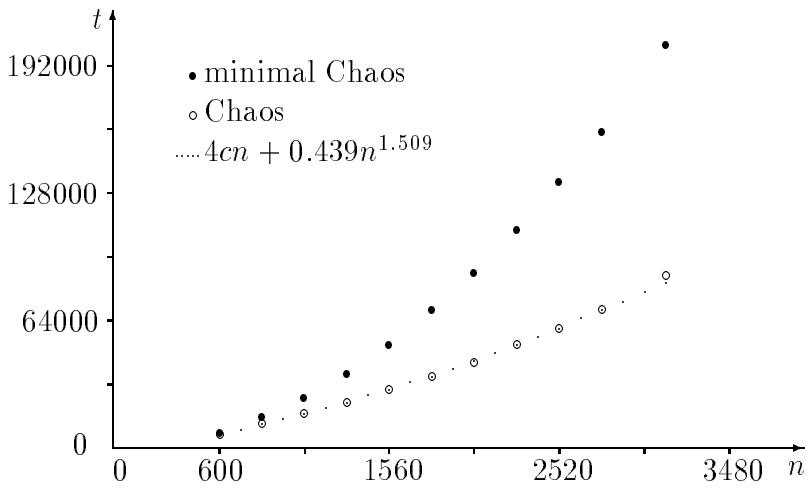

Figure 4.5: Results of Experiment 2. The top curve (filled circles) represents minimal Chaos, and the bottom curve (open circles) represents Chaos. If there were no congestion, the data would fit the curve $8cn$: for $n = 3120$, all packets would have escaped the $2cn \times 2cn$ box in 1456 cycles.

distance component. If the congestion component grows proportionally to n^x, then the ratio of the congestion components for two different problem sizes is related to the ratio of the problem sizes as follows:

$$\frac{\text{congestion component on the } m \times m \text{ mesh}}{\text{congestion component on the } n \times n \text{ mesh}} = \left(\frac{m}{n}\right)^x \tag{4.1}$$

Using linear regression on the logarithms of the mesh sizes and the estimated congestion components, we were able to approximate the congestion component by a polynomial of the form an^x. The resulting curve was $4cn + 0.439n^{1.509}$. The mean squared error[2] of the curve to the data is 1,096,855. If we perform linear regression on the data (not on the logarithms of the data), we obtain the curve $30.6n - 16308$, whose mean squared error is 9,855,884. When $n = 0$, this curve evaluates to a negative number, corresponding to a negative running time. Since the polynomial of higher degree fits the data better than the linear curve and the linear curve does not make sense at small values of n, we can conclude that it is unlikely that the algorithm's behavior is linear in n.

[2] If there are p data points $(x_1, y_1), \ldots, (x_p, y_p)$, then the *mean squared error* of a curve $f(x)$ to those data points is $\sum_{i=1}^{p} (y_i - f(x_i))^2/p$.

As in Experiment 1, we ran Chaos on the CLT permutation with each node in a random initial state. For all of the six mesh sizes on which we did the experiment with random initial states, Chaos performed no better than 10% better than the results in Figure 4.5. It is not surprising that the percentage difference is greater than in Experiment 1, since the distance component, which is not affected by random initial state, is much greater in Experiment 1 than in Experiment 2.

The CLT permutation arranges packets so that in minimal Chaos, packets destined for, say, the fourth row south of the hot spot are delayed by packets destined for the third, second, and first rows south of the hot spot (see Lemma 2.1). Similarly, packets destined for the fifth row are delayed by the fourth, third, etc. However, when we run Chaos on these permutations, the animation reveals that packets destined for the first row south of the hot spot are delayed by the second, third, etc. rows of packets. That is, although the worst case permutation for minimal Chaos causes poor behavior in Chaos, it does not do so in the same way as in the lower bound of Chapter 2.

Tables 4.1 and 4.2 in Section 4.5 give all of the data of the experiments in this chapter (except for the random initial state experiments).

4.2.3 Experiment 3

The results of Experiments 1 and 2 suggest that introducing random and nonminimal behavior into the routing algorithm allows a packet to move into its row or column before it would in the lower bound result of Chapter 2. Since a node using the Chaos algorithm introduces both randomness and nonminimality only when its central queue is full, it is plausible that reducing the queue size will increase the amount of randomness and nonminimality in the algorithm and thus deliver the packets in the CLT permutation more quickly.

Experiment 3 tested this hypothesis by reducing the central queue size of the nodes to two packets. The method for building the permutation and computing the curve that closely fits the data was the same as in Experiment 2. We used the same value of cn as in Experiment 2 in order to allow a direct comparison with the results of Experiment 2. Note that the result of Chapter 2 allows us to pick a larger value of c for this k, since the multiqueue is smaller. The results of Experiment 3

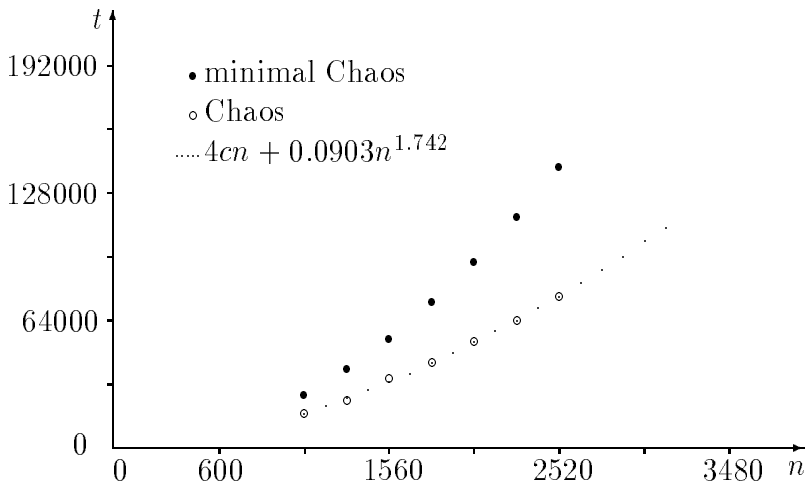

Figure 4.6: Results of Experiment 3 (multiqueue size = 2). The top curve (filled circles) represents minimal Chaos, and the bottom curve (open circles) represents Chaos. As in Experiment 2, if there were no congestion, the data would fit the curve $8cn$: for $n = 3120$, all packets would have escaped the $2cn \times 2cn$ box in 1456 cycles.

are shown in Figure 4.6.

As in Experiment 2, the time needed for Chaos to move all packets outside the $2cn \times 2cn$ corner can be closely approximated by $4cn + 0.0903n^{1.742}$. The mean squared error of this curve to the data is 369,154. If we perform linear regression on the data (not on the logarithms of the data), then we obtain the curve $40.5n - 28724$. The mean squared error of this linear curve to the data is 2,388,953. These results are evidence that when the central queue size was five, the randomness of the Chaos algorithm was more effective.

As in Experiments 1 and 2, we ran the CLT permutations with each node in a random initial state. Chaos performed no better than 3% better than the results of Figure 4.6. This difference is small because the algorithm is using more randomness than in Experiments 1 and 2, so the initial random state has less impact on delivery time than in the previous experiments.

4.3 A Greedy Hot Potato Algorithm

Hot potato or *deflection* routing [BNRST93, BDHS93, FR92, Haj91, KKR93], where a node must send on the next step any packets it receives in the current step, offers the possibility of simple logic and simple algorithms. *Greedy* hot potato routing [BDHS93], where packets use profitable outlinks whenever they are available, might be a nonminimal adaptive solution to route arbitrary permutations in time linear in n on the $n \times n$ mesh. Makedon and Symvonis [MS93] give an algorithm that is based on odd-even transposition and behaves much like a hot potato algorithm. Their algorithm uses a small amount of buffer space and is simple.

In hot potato algorithms, packets are deflected immediately when there is congestion, and so it is plausible that the packets of the CLT permutation will be spread out more quickly. Experiment 4 is a test of this hypothesis on a randomized hot potato algorithm that greedily assigns packets to outlinks based only on the profitable directions of the packets. At each step, each node randomly chooses whether to consider outlinks in the order North, South, East, West, or West, East, South, North. For each outlink considered, a randomly chosen packet that can use the outlink profitably, if any, is scheduled to that outlink. After this greedy scheduling, any unscheduled packets are assigned to available outlinks in an arbitrary way. Let us call our algorithm *GreedyHP*. GreedyHP is destination-exchangeable, and it is intended to approximate the hot potato algorithm (suggested by Borodin and Hopcroft [BH85]) that for each node randomly picks a scheduling of the outlinks that maximizes the number of packets that advance. This latter algorithm is currently impractical because computing a random maximal matching is expensive, either in time or in hardware complexity.

Unlike the Chaos algorithm, GreedyHP has no obvious deterministic, minimal counterpart. In order to construct a permutation like the CLT permutation, we ran GreedyHP using one seed of our pseudorandom number generator with all packets destined for the node in the southeast corner of the $2cn \times 2cn$ box. As packets enter the rows (columns) just outside the $cn \times cn$ box, we altered their destinations to be the node at the east (south, respectively) edge of that row (column, respectively). For example, the first $(1-c)n$ packets to enter column $cn+1$ have as their destination the node in column $cn+1$ and row $2cn$.

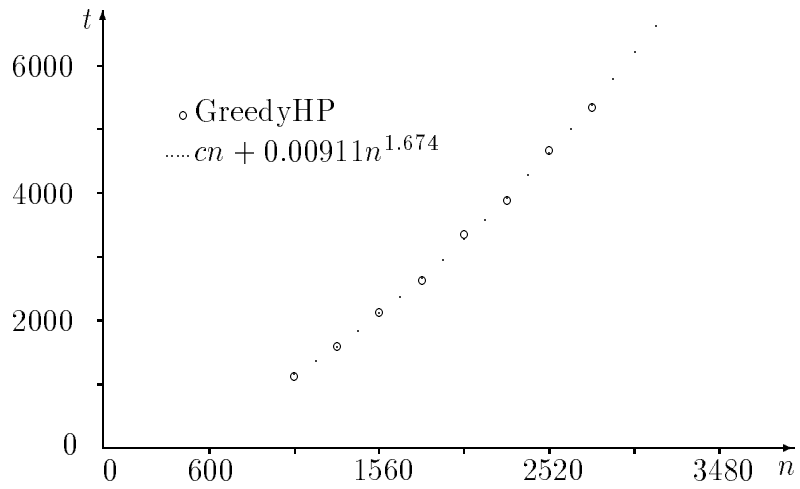

Figure 4.7: Results of Experiment 4. The curve (open circles) represents GreedyHP. If there were no congestion, the data would fit the curve $2cn$: for $n = 3120$, all packets would have escaped the $2cn \times 2cn$ box in 384 cycles.

Since this permutation is constructed given a particular pseudorandom number sequence, we then ran GreedyHP on the permutation with three other seeds to the generator. The results of these three runs were averaged and fitted to a curve of the form $cn + an^x$ (see Figure 4.7). The distance component was estimated to be cn. (The packet routing simulator used in this experiment was different from the Chaos simulator and used 1-word packets.) We subtracted cn from the value of the observed data and fitted it to a curve of the form an^x, as we did in Experiment 2. The resulting curve is $cn + 0.0911n^{1.674}$. The mean squared error of the curve to the data is 1249. A linear curve fit of the data yields $2.45n - 1734$, whose mean squared error is 10137.

From the results of this experiment, we can conclude that although randomness and nonminimality help destination-exchangeable algorithms "break" the lower bound of Chapter 2 (at least on these constructed permutations), they still exhibit behavior that is superlinear in n. Observing the animations of GreedyHP, we see that although packets are spread out by smaller queues, packets that deviate from the destination row or column immediately try to return to that row or column. That is, packets still interfere with other packets at or near the hot spot.

4.4 Summary and Discussion

Our experiments in this chapter have examined the performance of the Chaos algorithm and a plausibly practical hot potato algorithm. Both algorithms appear to have asymptotic running times that are superlinear in n. The experiments presented here do not preclude the existence of permutations that cause quadratic behavior for Chaos or GreedyHP, although this seems unlikely to us after observing the way these algorithms route packets.

The exponents of n that we calculated in Experiments 2, 3, and 4 should not be interpreted as exact characterizations of the asymptotic behavior of the algorithms for the following reasons. The calculations of the exponent were made on the assumption that the polynomial that approximates the observed data is of the form an^x, which ignores the possibility of lower order terms that might influence the running times. For example, in Experiment 2, if we fit a curve of the form an^x to the first six data points, we get $0.548n^{1.478}$, whereas if we fit to the last six data points, we get $0.157n^{1.641}$. This suggests that there are lower order terms, and so the exponents may be higher than the ones we computed. Another possibility is that the mesh sizes that we used were too small for the algorithms to exhibit their asymptotic behavior.

Another reason that the curves computed for Experiments 3 and 4 may be inaccurate is that the numbers of packets used in those experiments were chosen to be those of Experiment 2 to allow direct comparison. However, for Experiment 3, the lower bound of Chapter 2 says that we can choose c to be much greater. For Experiment 4, we have no theorem that prescribes how to construct a bad permutation. In short, the three experiments at best are guesses for how to build a family of bad permutations for these nonminimal adaptive algorithms, and direct comparison of the computed curves (e.g., making a conclusion about the effect of queue size based on the results of Experiments 2 and 3) is probably unsound.

However, we know that for both Chaos and GreedyHP, packets interfere with each other near the hot spot, and so it is plausible that if we were to continue the experiments on larger mesh sizes, then we would observe superlinear behavior, and possibly behavior of the curves computed in Experiments 2, 3, and 4 (or worse).

4.5 Experimental Data

This section contains all of the data of the Experiments 1 through 4 (except for the random initial state experiments).

Table 4.1: Data for Experiments 1 and 2. The difference in the observed data from Experiment 1 and Experiment 2 on the same problem sizes is due to the difference in the distance components of the two experiments. It is approximately $4(n - cn)$ cycles in Experiment 1, whereas it is approximately $4cn$ cycles in Experiment 2.

		Experiment 1		Experiment 2	
n	cn	min. Chaos	Chaos	min. Chaos	Chaos
600	35	10529	9382	8318	7075
720	42	14382	11699		
840	49	18815	14423	15765	11588
960	56	24091	17786		
1080	63	29792	21015	25505	17112
1200	70	35740	24227		
1320	77	42554	27152	37626	22529
1560	91	58262	36068	52430	29815
1800	105			68696	36051
2040	119			88162	43260
2280	133			109210	51428
2520	147			133277	60357
2760	161			158961	69456
3120	182			202982	86537

Table 4.2: Data for Experiments 3 and 4. The data for Experiment 4 is the average of three separate runs.

		Experiment 3		Experiment 4
n	cn	min. Chaos	Chaos	GreedyHP
1080	63	26655	17666	1141
1320	77	39538	24340	1597
1560	91	55108	34285	2130
1800	105	72711	42322	2640
2040	119	93424	53869	3343
2280	133	116026	63844	3893
2520	147	141706	76088	4658
2760	161			5331

Chapter 5

CONCLUSIONS

Our search for a simple routing algorithm on the $n \times n$ mesh that routes arbitrary permutations in $O(n)$ time has led to the discovery that no such "simple" (i.e., deterministic, destination-exchangeable, and minimal adaptive with bounded queues) algorithm exists. That is, the results of Chapter 2 tell us that if we want to route arbitrary permutations in $o(n^2/k^2)+O(n)$ time on the $n \times n$ mesh with queues of size at most k, then our algorithm must either: (1) incorporate the destination addresses (rather than just profitable outlinks) of packets in routing decisions, (2) use a routing algorithm that allows packets to take paths other than their minimal ones, or (3) incorporate randomness in routing decisions.

An immediate open problem is to show that there is a matching upper bound to the main result of Chapter 2:

Open Problem 5.1 *Show that there exists a deterministic, destination-exchangeable, minimal adaptive algorithm on the $n \times n$ mesh with queues of size k that routes arbitrary permutations in $O((n^2/k^2) + n)$ steps.*

The result of Chapter 3 tells us that if we do allow the full destination addresses of packets to be used in routing decisions, then we can find an algorithm (albeit complicated and probably impractical) that routes arbitrary permutations in $O(n)$ time.

The experiments of Chapter 4 suggest that two randomized, destination-exchangeable, nonminimal adaptive algorithms (one of them built into a routing chip, the other a plausibly practical algorithm) do not have behavior linear in n.

Conjecture 5.2 *There is no destination-exchangeable algorithm with bounded queues that can route arbitrary permutations in $O(n)$ time (expected, for randomized algorithms) on the $n \times n$ mesh.*

Krizanc, Rajasekaran, and Tsantilas [KRT88] give an algorithm that is almost a counterexample to the conjecture above. Their algorithm is a simple variant of Valiant and Brebner's randomized routing result on the hypercube [VB81], where a packet is first sent to a random intermediate node using the dimension order algorithm, and then sent from the intermediate node to its destination. In the Krizanc *et al.* result, a packet is sent to a random nearby node in the same column as its source and then is sent from that node to its destination via the dimension order algorithm. They prove an $O(n)$ running time with queues of size $O(\log n)$ with high probability. However, their result does not contradict the conjecture, because they allow unbounded queues. We know that if we have bounded queues, congestion builds in ways qualitatively different from congestion in networks with unbounded queues. (For example, the dimension order algorithm with the farthest-first outqueue policy and unbounded queues routes arbitrary permutations in $2n-2$ steps, whereas with bounded queues, it takes $\Omega(n^2/k)$ steps.)

The results of Chapter 4 suggest that for practical routers, there is a connection between the worst case permutations for minimal adaptive algorithms and non-minimal adaptive algorithms. We emphasize "practical" because the algorithms must be simple, so that they can be built with a small amount of hardware. In particular, these algorithms will have to make local decisions. In lightly-loaded situations, a practical nonminimal adaptive algorithm will behave like a minimal adaptive algorithm, because in such situations there is no reason to deroute. Given that assumption, then only when congestion is heavy can a nonminimal algorithm behave differently from a minimal algorithm. But by then, it perhaps is too late to try to relieve the congestion.

Open Problem 5.3 *Show that there is no deterministic routing algorithm on the $n \times n$ mesh with constant sized queues that routes arbitrary permutations in $O(n)$ time and is practical. By a "practical" algorithm, we mean one that uses local decisions, that extends to the asynchronous and dynamic settings, whose hardware implementation would be small and fast, and whose constant in the running time is small.*

On the other hand, we may already know of practical algorithms that can route arbitrary permutations in $O(n)$ time. For example, despite the evidence in Chap-

ter 4, the Chaos algorithm or GreedyHP may be such an algorithm. Currently, it is difficult to analyze practical or plausibly practical nonminimal algorithms such as these.

Another possible avenue for future research is to devise algorithms and prove analytic bounds on the time it takes to deliver packets from a different input class. For example, it may be that from a practical point of view we are not interested in all permutations, but in some subset of permutations that is a more realistic benchmark (i.e., a subset that approximates inputs that arise in practice). An even more difficult open problem is to define an appropriate metric for the dynamic routing scenario, where packets are continuously injected into the network, and devise algorithms. Once again, we may already know of such algorithms but have not yet been able to prove good upper bounds on their performance.

If we believe that in the future the routing algorithm of a multiprocessor system will play a more significant role in its performance, then answering these questions will be one of the first steps to realizing the full potential of such systems.

Bibliography

[BAS94] I. Ben-Aroya and A. Schuster. A CLT-type lower bound for hot-potato permutation routing. Technical Report LPCR #9405, CS Department, Technion, May 1994.

[Bat80] K. E. Batcher. Design of a massively parallel processor. *IEEE Transactions on Computers*, 29(9):836–840, September 1980.

[BC91] J. T. Brassil and R. L. Cruz. Bounds on maximum delay in networks with deflection routing. In *29th Annual Allerton Conference on Communication, Control, and Computing*, pages 571–580, 1991.

[BDHS93] A. Ben-Dor, S. Halevi, and A. Schuster. On greedy hot-potatoe routing. Technical Report PCL Report #9204, CS Department, Technion, January 1993.

[BFS94] K. Bolding, M. Fulgham, and L. Snyder. The case for chaotic adaptive routing. Technical Report TR 94-02-04, University of Washington Department of Computer Science and Engineering, March 1994.

[BH85] A. Borodin and J. E. Hopcroft. Routing, merging, and sorting on parallel models of computation. *Journal of Computer and System Sciences*, 30:130–145, 1985.

[BNRST93] A. Bar-Noy, P. Raghavan, B. Schieber, and H. Tamaki. Fast deflection routing for packets and worms. In *Proceedings of the Twelfth Annual ACM Symposium on Principles of Distributed Computing*, pages 75–86, 1993.

[Bol93] K. Bolding. *Chaotic Routing: Design and Implementation of an Adaptive Multicomputer Network Router.* PhD thesis, University of Washington, Seattle, WA, July 1993.

[BRSU93] A. Borodin, P. Raghavan, B. Schieber, and E. Upfal. How much can hardware help routing? In *Proceedings of the Twenty Fifth Annual ACM Symposium on Theory of Computing*, pages 573–582, May 1993.

[BS92] K. Bolding and L. Snyder. Mesh and torus chaotic routing. In *Advanced Research in VLSI and Parallel Systems: Proceedings of the 1992 Brown/MIT Conference*, pages 333–347, March 1992.

[CG92] R. Cypher and L. Gravano. Adaptive, deadlock-free packet routing in torus networks with minimal storage. In *1992 International Conference on Parallel Processing*, pages 204–211, 1992.

[Chi] D. D. Chinn. The performance of minimal adaptive algorithms on worst case permutations. To appear in *Parallel Computer Routing and Communication Workshop*.

[CK92] A. Chien and J. H. Kim. Planar-adaptive routing: Low-cost adaptive networks for multiprocessors. In *Proceedings of the 19th International Symposium on Computer Architecture*, pages 268–277, 1992.

[CLT94] D. D. Chinn, T. Leighton, and M. Tompa. Minimal adaptive routing on the mesh with bounded queue size. In *Proceedings of the 1994 ACM Symposium on Parallel Algorithms and Architectures*, Cape May, NJ, June 1994.

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

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

[FR92] U. Feige and P. Raghavan. Exact analysis of hot-potato routing. In *Proceedings 33rd Annual Symposium on Foundations of Computer Science*, pages 553–562, Pittsburgh, PA, October 1992.

[Haj91] B. Hajek. Bounds for evacuation time for deflection routing. *Distributed Computing*, 5:1–6, 1991.

[HS90] T. Han and D. Stanat. "Move and smooth" routing algorithms on mesh-connected computers. In *28th Annual Allerton Conference on Communication, Control, and Computing*, pages 236–245, 1990.

[Int91] Intel. A Touchstone DELTA system description. Technical report, Intel, Portland, OR, 1991.

[KKR93] C. Kaklamanis, D. Krizanc, and S. Rao. Hot-potato routing on processor arrays. In *Proceedings of the 1993 ACM Symposium on Parallel Algorithms and Architectures*, pages 273–282, June 1993.

[KKT90] C. Kaklamanis, D. Krizanc, and T. Tsantilas. Tight bounds for oblivious routing in the hypercube. In *Proceedings of the 1990 ACM Symposium on Parallel Algorithms and Architectures*, pages 31–36, June 1990.

[Kri91] D. Krizanc. Oblivious routing with limited buffer capacity. *Journal of Computer and System Sciences*, 43:317–327, 1991.

[KRT88] D. Krizanc, S. Rajasekaran, and T. Tsantilas. Optimal routing for mesh-connected processor arrays. In *3rd Aegean Workshop on Computing (AWOC)*, volume 319 of *Lecture Notes in Computer Science*, pages 411–422. Springer-Verlag, 1988.

[KS90] S. Konstantinidou and L. Snyder. The chaos router: A practical application of randomization in network routing. In *Proceedings of the 1990 ACM Symposium on Parallel Algorithms and Architectures*, pages 21–30, June 1990.

[KS91] S. Konstantinidou and L. Snyder. Chaos router: Architecture and performance. In *Proceedings of the 18th International Symposium on Computer Architecture*, pages 212–221, May 1991.

[KS94] S. Konstantinidou and L. Snyder. The Chaos Router. *IEEE Transactions on Computers*, 43(12):1386–1397, December 1994.

[Kun88] M. Kunde. Routing and sorting on mesh-connected arrays. In *3rd Aegean Workshop on Computing (AWOC)*, volume 319 of *Lecture Notes in Computer Science*, pages 423–433. Springer-Verlag, 1988.

[Lei] T. Leighton. Personal communication.

[Lei90] T. Leighton. Average case analysis of greedy routing algorithms on arrays. In *Proceedings of the 1990 ACM Symposium on Parallel Algorithms and Architectures*, pages 2–10, July 1990.

[Lei92] F. Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann, 1992.

[LLJ+92] D. Lenoski, J. Laudon, T. Joe, D. Nakahira, L. Stevens, A. Gupta, and J. Hennessy. The DASH prototype: Implementation and performance. In *Proc. 19th Annual Symposium on Computer Architecture*, pages 92–103, June 1992.

[LMT89] T. Leighton, F. Makedon, and I. Tollis. A $2n - 2$ step algorithm for routing in an $n \times n$ array with constant size queues. In *Proceedings of the 1989 ACM Symposium on Parallel Algorithms and Architectures*, pages 328–335, July 1989.

[MP-87] MP-1 family data-parallel computers. Technical report, MasPar Computer Corporation, 749 North Mary Ave., Sunnyvale, CA., 1987.

[MS92] B. Maggs and R. Sitaraman. Simple algorithms for routing on butterfly networks with bounded queues. In *Proceedings of the Twenty Fourth Annual ACM Symposium on Theory of Computing*, pages 150–161, May 1992.

[MS93] F. Makedon and A. Symvonis. An efficient hueristic for permutation on meshes with low buffer requirements. *IEEE Transactions on Parallel and Distributed Systems*, 4(3):270–6, March 1993.

[ND90] M. Noakes and W. Dally. System design of the J-Machine. In *Proceedings of the 6th MIT Conference on Advanced Research in VLSI*, pages 179–194, 1990.

[NS89] J. Y. Ngai and C. L. Seitz. A framework for adaptive routing in multicomputer networks. In *Proceedings of the Symposium of Parallel Algorithms and Architectures*, pages 1–9. ACM, 1989.

[NS91] J. Y. Ngai and C. L. Seitz. A framework for adaptive routing in multicomputer networks. *Computer Architecture News*, 19(1):6–14, March 1991.

[NS92] I. Newman and A. Schuster. Hot-potato algorithms for permutation routing. Technical Report PCL Report #9201, CS Department, Technion, November 1992.

[Ran87] A. Ranade. Equivalence of message scheduling algorithms for parallel communication. Technical Report YALEU/DCS/TR-511, Department of Computer Science, Yale University, New Haven, CT, 1987.

[RO92] S. Rajasekaran and R. Overholt. Constant queue routing on a mesh. *Journal of Parallel and Distributed Computing*, 15(2):160–166, June 1992.

[SBSS93] C. Sietz, N. Boden, J. Seizovic, and W. Su. The design of the Caltech Mosaic C multicomputer. In *Proceedings of the Symposium on Integrated Systems*, pages 1–22, 1993.

[SWG92] J. Singh, W.-D. Weber, and A. Gupta. SPLASH: Stanford Parallel Applications for Shared Memory. *Computer Architecture News*, pages 5–44, March 1992.

[VB81] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Conference Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, pages 263–277, Milwaukee, WI, May 1981.

Vita

Donald Dotway Chinn was born in New Brunswick, New Jersey in 1967. He received his Bachelor of Arts in Computer Science with High Distinction from the University of California, Berkeley in May, 1988. He received his Master of Science in Computer Science in August, 1991, and his Doctor of Philosophy in Computer Science in March, 1995, both at the University of Washington.