Minimal Adaptive Routing on the Mesh with Bounded Queue Size

Donald D. Chinn *
Tom Leighton †
Martin Tompa ‡

Technical Report #94–07–03

July 7, 1994

Department of Computer Science and Engineering, FR-35
University of Washington
Seattle, Washington, U.S.A. 98195

*dci@cs.washington.edu; Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195. This material is based upon work supported in part by the National Science Foundation under Grant MIP-9213469.

†ftl@math.mit.edu; Mathematics Department and Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139. This material is based upon work supported in part by AFOSR Grant F49620-92-J-0125 and ARPA Grants N00014-91-J-1698 and N00014-92-1799.

‡tompa@cs.washington.edu; Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195. This material is based upon work supported in part by the National Science Foundation under Grants CCR-9002891 and CCR-9301186.

Abstract

An adaptive routing algorithm is one in which the path a packet takes from its source to its destination may depend on other packets it encounters. Such algorithms potentially avoid network bottlenecks by routing packets around "hot spots." Minimal adaptive routing algorithms have the additional advantage that the path each packet takes is a shortest one.

For a large class of minimal adaptive routing algorithms, we present 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 more general routing problems, such as the h-h routing problem. It also extends to a large class of dimension order routing algorithms, yielding an $\Omega(n^2/k)$ time bound.

To complement these lower bounds, we present two upper bounds. One is an $O(n^2/k)$ time dimension order routing algorithm that matches the lower bound. The other is the first instance of a minimal adaptive routing algorithm that achieves $O(n)$ time with constant sized queues per node. We point out why the latter algorithm is outside the model of our lower bounds.

1. Introduction

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 bounded queues (which most use) in the worst case.

In theory, on the other hand, there are many fast algorithms for static routing problems on synchronous networks, 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.

For some time now, both sides 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, no dependence on destination other than preferred directions, and minimal (i.e., shortest) routes from source to destination. 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 size of each queue. We

also provide a simple algorithm that routes any permutation on the $n \times n$ mesh in $O(n^2/k)$ time.

The lower bound holds even if the algorithm is "adaptive", meaning that the choice of path for a packet may depend on the state of the network as the packet traverses the network and, in particular, which other packets it encounters. This is the first nontrivial lower bound for adaptive routing.

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 [2], the MP-1 from MasPar [21], the Paragon from Intel Scientific, the J-machine from MIT [23], the Touchstone DELTA from Intel [11], the DASH from Stanford [19], and the Mosaic from Cal Tech [26].

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. At the very least, a good routing algorithm should be able to route permutations efficiently.

This introduction concludes by surveying some of the known results for permutation routing.

1.1. Routing with Unbounded Queues

Borodin and Hopcroft [3] 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 (i.e., the path a packet takes depends only on its source and destination). Kaklamanis *et al.* [13] 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.

A simple oblivious way to route packets in mesh networks is to use *dimension order* paths. In order to reach its destination, a packet first travels along its row until it reaches its destination column. It then moves in that column until it reaches its destination row. Each move the packet makes reduces the distance to its destination by one. This scheme is both easy to implement in hardware and simple enough so that the implementation of the routing logic consumes a relatively small area on a chip, making it the algorithm of choice in practice.

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 [16, pages 159–162]). Unfortunately, this algorithm requires $\Theta(n)$ size queues at each node. (Leighton [17] 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 [14] 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 [20] 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 [15] 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 [18] and Rajasekaran and Overholt [24] 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 [10] 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 routing [1, 5, 8, 9, 12, 22], where at each step every node in the network must send all packets it received during the previous step. In these algorithms, packets again typically take nonminimal paths. Newman and Schuster [22]

give an algorithm that routes any permutation in $7n + o(n)$ steps, but the algorithm uses sorting. Bar-Noy *et al.* [1] 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 [10, 15, 16, 18, 22, 24] each violate either the assumption of bounded queues, or both the assumptions of minimal paths and using only preferred directions. In addition to our lower bounds, we present the first minimal adaptive routing algorithm that achieves $O(n)$ time with bounded queues. Our algorithm does exploit the full destination addresses (and, like the others, it does so in a complicated and possibly impractical way). Thus, it is impossible to eliminate the assumption of destination-exchangeability from our lower bound.

The remainder of the paper is organized as follows. Section 2 describes the model and defines terms necessary for the argument. Section 3 provides the construction of the permutation that achieves the lower bound. Section 4 proves the lower bound. Section 5 extends the argument to other routing problems. Section 6 describes the $O(n)$ time minimal adaptive algorithm. Section 7 gives conclusions and open problems.

2. 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, 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.* [4]. For consistency with the existing literature, we use the word "queue" to denote a set of waiting packets. 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 the queue, to attempt to send out the node's outlinks. No more than one packet can be scheduled to each outlink. Examples of outqueue policies are FIFO or farthest-first [16, 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 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 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.

Call an algorithm that fits this description a *destination-exchangeable routing algorithm.* Note that the restriction to profitable outlinks is similar to the definition of a *nonpredictive* algorithm, given by Ranade [25], Leighton [16, page 556], and Maggs and Sitaraman [20]. 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

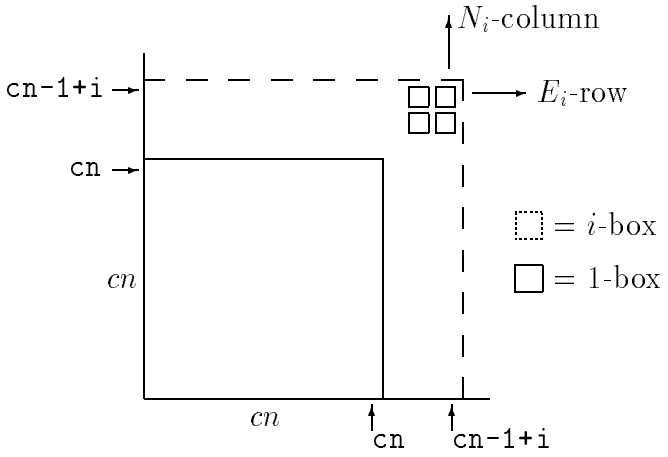

Figure 1: The $n \times n$ mesh.

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 [6] and Cypher and Gravano [7]. An example of a nonminimal destination-exchangeable algorithm is the $O(n^{3/2})$ hot potato algorithm of Bar-Noy *et al.* [1].

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 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.

3. The Construction

For any given destination-exchangeable 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 now present the construction.

1. Let $p = \lfloor (k+1)(cn + c^2 n) + dn \rfloor$, 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, provided $\lfloor (k+1)(cn + c^2 n) + dn \rfloor \leq (1-c)n - l$. (See Section 4.3.)

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 3 and 4 will show that the packets to be exchanged are always available.)

   - 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.

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.

4. 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 certain facts about the construction itself and then prove that when the routing algorithm is run on the constructed permutation, these facts also hold.

4.1. Properties of the Construction

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

Lemma 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: 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 per step.

Lemma 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.

Lemma 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.

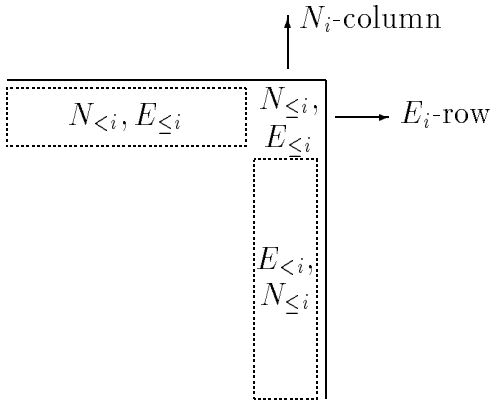

Figure 2: The i-box up to step idn.

Lemma 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 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 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 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 $0 \leq t \leq idn$.

Figure 2 illustrates the invariant of the construction. $N_{<i}, E_{\leq i}$ indicates that the node only contains N_j-packets, where $j < i$, or E_j-packets, where $j \leq i$ (and similarly for $E_{<i}, N_{\leq i}$ and $N_{\leq i}, E_{\leq i}$).

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 1 through 4 are vacuously true, since they do not apply when $t = 0$. Lemmas 5 through 8 are verified by inspection of the initial arrangement of the packets.

Induction : Assume Lemmas 5 through 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 1 follows from Lemmas 5 and 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 5 or 6.

Since no N_j-packet has left the $(i-2)$-box by the end of step $t-1$ by Lemma 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 follows from Lemmas 7 and 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 7). A similar argument holds for E_i-packets using Lemma 8.

Lemma 3 follows from Lemma 7 for step $t-1$, and Lemmas 1 and 2 for step t. The number of N_i-packets that begin the construction is $\lfloor (k+1)(cn+c^2n)+dn \rfloor$. From Lemmas 1 and 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 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 $i \leq l \leq c^2n$ for our choices of c and d (see Section 4.3). Thus, all x exchanges can be performed.

Lemma 4 follows from Lemma 8 for step $t-1$, Lemmas 1 and 2 for step t, and the number of E_i-packets that begin the construction, analogous to the proof of Lemma 3.

Lemma 5 follows from Lemmas 5 and 6 for step $t-1$ and Lemmas 3 and 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 5 or 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 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 4 ensure that we can perform an exchange, leaving the N_j-packet in the $(i-2)$-box.

Lemma 6 follows from Lemmas 5 and 6 for step $t-1$ and Lemmas 3 and 4 at step t, analogous to the proof of Lemma 5, except that EX1 and EX3 are the relevant exchange

rules.

Lemma 7 follows from Lemma 7 at step $t-1$ and Lemma 4 at step t. The proof is similar to that of Lemma 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 8 follows from Lemma 8 at step $t-1$ and Lemma 3 at step t. The proof is similar to that of Lemma 7, except that EX3 is the relevant exchange rule. □

Corollary 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 1 and 2. Then at least $\lfloor (k+1)(cn + c^2 n) \rfloor$ N_i-packets and $\lfloor (k+1)(cn + c^2 n) \rfloor$ E_i-packets remain in the i-box, and hence are undelivered, at time $\lfloor l \rfloor dn$. □

4.2. Properties of the Constructed Permutation

We must now show that the analogue of Corollary 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 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 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 10. □

Note that $(S_X)_X = S$ and $(S_X)_Y = S_{<X,Y>}$ 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 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 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.

All of the exchanges in $< X_{t+1}, \ldots, X_{\lfloor l \rfloor dn} >$ are of the type in the preconditions of Lemma 11, because for any pair of packets exchanged during any step of the construction,

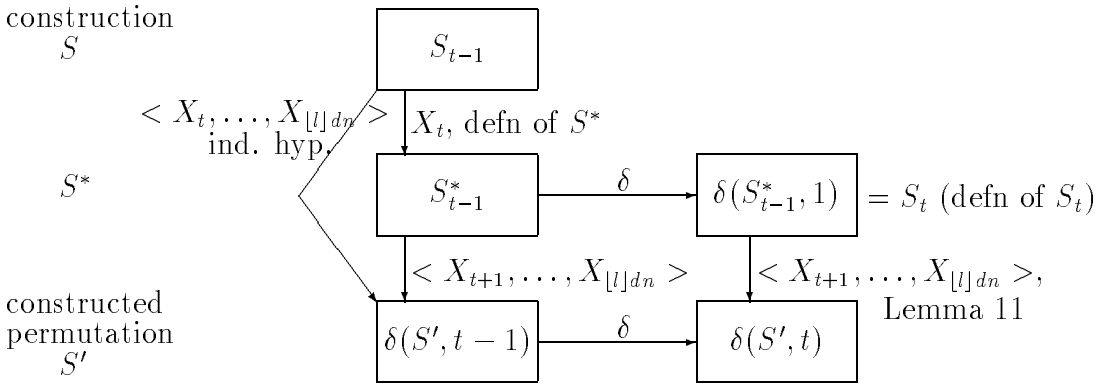

Figure 3: S_t, S^*_t, and $\delta(S', t)$ in Lemma 12.

each packet of the pair is in the $(i-1)$-box for some i and destined for nodes northeast of the $(i-1)$-box. Then by Lemma 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. 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 9), we have:

Theorem 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.

4.3. Choosing the Constants c and d

All that is left to complete the analysis is to find constants c and d satisfying 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 the proof of Lemmas 3 and 4).

The first constraint is

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

Rewritten, this becomes

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

This is satisfied, provided that

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

Choosing $c = 1/(2\cdot(k+2))$ and $d = 2/5$ satisfies Inequality (1) for $k \geq 1$. Taking derivatives of the left hand side of Inequality (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 (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 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:

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

We can now calculate the lower bound of Theorem 13 in terms of n and k by substituting our choices of c and d.

Theorem 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 13, the number of steps is at least

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

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

5. Extensions of the Lower Bound

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

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.

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.

Nonminimal extensions: Consider the class of destination-exchangeable algorithms where every packet is guaranteed never to move more than δ nodes beyond the rectangle consisting of those nodes in any of the shortest paths from the packet's source to its destination. The techniques of this paper can be used to obtain lower bounds for such nonminimal algorithms. Instead of choosing $p = (k+1)(cn+c^2n)+dn$, we choose $p = (\delta+1)\cdot((k+1)(cn+c^2n)+dn)$, since, for example, we must have enough N_i-packets to occupy not only the N_i-column south of the E_i-row, but also the δ columns to its east south of the E_i-row. This affects the invariant, the number of packets that escape those $\delta+1$ columns per step, and the constraint we must maintain when choosing the constants c and d. This yields a bound of $\Omega(n^2/(\delta+1)^3k^2)$.

Since the $O(n^{3/2})$ time hot potato algorithm of Bar-Noy *et al.* [1] is destination-exchangeable, the restriction in Theorem 14 of minimal routing cannot be eliminated entirely.

The Torus: The bound of Section 4.3 also holds asymptotically for the torus. The construction is simply 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.

***h*-*h* Routing Problems:** In h-h routing problems, each node is allowed to send up to h packets and receive up to h packets. The construction is modified to have h packets in each of the c^2n^2 nodes of the 1-box. As before, we define $p = \lfloor(k+1)(cn+c^2n)+dn\rfloor$, but we define $l = hc^2n^2/(2p)$. We again have three constraints similar to those of Section 4.3 to satisfy. The first constraint is

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

This constraint is satisfied for $c \le h/(3\cdot(k+1+h))$ and $d \le 5h/9$, if $(k+1)(c+c^2)+d \ge (h/2)+(1/n)$. Using techniques similar to those of Section 4.3, we choose $c \ge h/(4\cdot(k+1+h))$ and $d \ge 77h/144$ that satisfy all the constraints.

From this choice of c and d, we get

$$\lfloor l\rfloor dn \ge \left\lfloor\frac{h^2n}{26\cdot(k+1+h)^2}\right\rfloor\cdot\frac{77}{144}hn.$$

We conclude that $\lfloor l\rfloor dn = \Omega(h^3n^2/(k+h)^2)$.

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.

Dimension Order Routing: The arguments presented here also apply to dimension order routing. Because of the regularity in the paths, one can prove an $\Omega(n^2/k)$ bound for routing a worst case permutation in a destination-exchangeable dimension order router as follows.

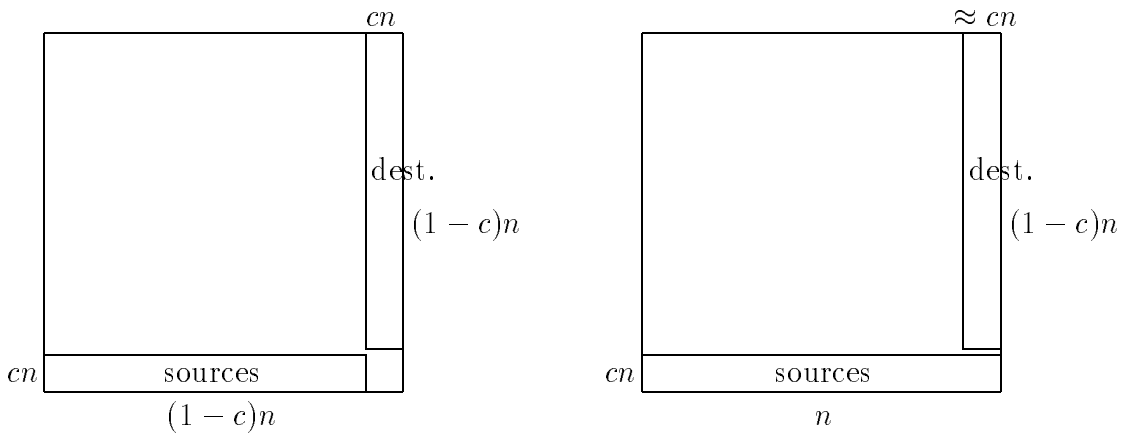

Figure 4: The construction for destination-exchangeable dimension order algorithms (left) and farthest-first algorithms (right).

The construction is similar to that of Section 3, 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 4). Define the N_i-column to be the $((1-c)n-1+i)$-th column of the mesh, and the i-box to be the set of nodes west of and including the N_i-column and south of and including row cn. We define $p = (k+1)cn + dn$ and $l = (1-c)cn^2/p$. The construction proceeds as before, but there is only one exchange rule: 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 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 4.3, we can find constants $2/(5\cdot(k+2)) \leq c \leq 1/(2\cdot(k+2))$ and $2/5 \leq d \leq 1/2$. We conclude that $\lfloor l \rfloor dn \geq \lfloor 3n/(8\cdot(k+2)) \rfloor (2n/5)$, giving a bound of $\Omega(n^2/k)$.

For h-h routing with destination-exchangeable dimension order routers, the analysis shows that $\lfloor l \rfloor dn \geq \lfloor 4hn/(15\cdot(k+1+h)) \rfloor (2hn/5)$, giving a bound of $\Omega(h^2n^2/(k+h))$.

The lower bound 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. In this case the lower bound holds even though this algorithm does make use of the entire destination address, and hence is not destination-exchangeable.

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 4).

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 $i \geq 1$, $j > i$, 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 is not hard to see that packets are always available for the exchange, that for $j > i$, no N_j-packet is further east in its row than any N_i-packet in that row, 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 $2/5 \leq d \leq 1/2$. We conclude that $\lfloor l \rfloor dn \geq \lfloor 2n/(9 \cdot (k+1)) \rfloor (2n/5)$, giving a bound of $\Omega(n^2/k)$.

We prove that the bound for destination-exchangeable dimension order routers is tight in Theorem 15.

Theorem 15: 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 is the size of the queue.

Proof: Assume that each node has four incoming queues (labelled North, South, East, and West), 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.

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 (respectively, South) 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, 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.

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. Suppose that $3n$ steps after the

end of a turning interval, another turning interval has not begun, i.e., that there is never an East or West queue in some column j that contains k packets, all of which want to turn into column j. Then it is easy to prove by induction that after $2n$ steps, every packet is either in its column or crossing a link per step in its row. Since a packet travels at most distance n in a row, any packet not in its destination column will reach it within n more steps.

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

6. An $O(n)$-Time, $O(1)$-Space Minimal Adaptive Algorithm

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 [15, 18, 22, 24], but those algorithms do not use minimal routes.

The algorithm 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.

6.1. 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).

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 19 shows that such tilings exist. We will assume for simplicity that n is a power of 3.

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

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

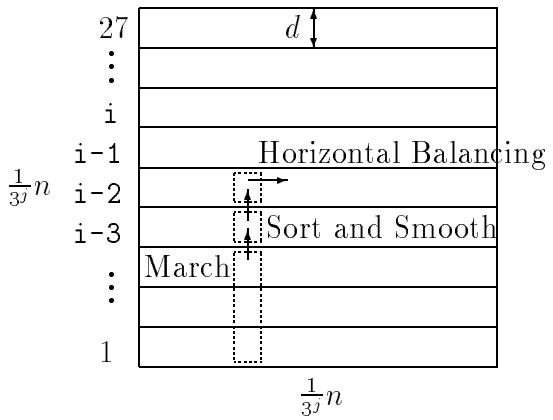

Figure 5: The Vertical Phase of the algorithm.

- 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 only participates in an action on a given tile 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.

- **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 not active). See Figure 5. For each of the following steps, every node knows how long it will take (see Lemmas 29, 30, and 31) 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 the strip 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

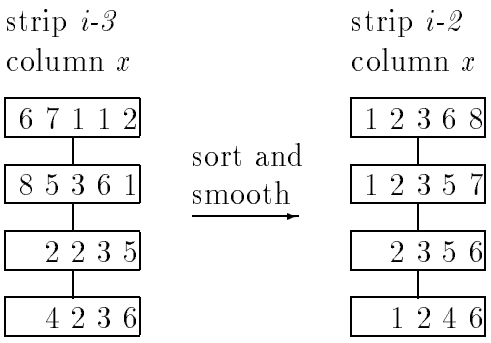

Figure 6: Sort and Smooth ($d = 4$). Each packet is represented by the horizontal distance to its destination.

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 6). 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.

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.

6.2. Correctness

We now show that the routing algorithm presented in Section 6.1 is minimal adaptive, and that it 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 16: 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 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 6.)

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. □

Consider any particular row r and destination column c. Suppose at the beginning of some step t, where the first step of Horizontal Balancing corresponds to $t = 1$, that for all $s \geq 1$ there are no more than $2s$ ($\leq c$)-packets in the first s nodes west of and including column c. Call this Condition C.

Lemma 17: For all $t \geq 0$, no ($\leq c$)-packet is transmitted east by the node in column c during step t of Horizontal Balancing, and Condition C holds at the beginning of step $t+1$.

Proof: The proof is by induction on t.

Basis : $t = 0$. This follows immediately from Lemma 16.

Induction : Assume the statement is true for $t - 1 \geq 0$ (in particular, Condition 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 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 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 for $s = s'+1$ at step t.

Since we chose r and c arbitrarily, we have proved Lemma 17 statement for all r and c, and hence for all packets. □

Lemma 18: 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 19: 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 and $3d$ columns from the tiles in the first tiling. The tiles in the third tiling are displaced $3d$ rows and $3d$ columns 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. □

Theorem 20: 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 17 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 18 and 19 (and the horizontal analogue of Lemma 18), 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. When $\frac{1}{3^j} n < 27$, then the dimension order algorithm is used. Thus, every packet is eventually delivered. □

6.3. 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 21: 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. Also, 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 $q + 1$ packets at any time. □

Lemma 22: 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 23: 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 24: 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.

Lemma 25: 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. 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. (It could move vertically during Vertical Balancing of a horizontal subphase.)

The fact that the packet is at least $d+1$ and at most $3d-1$ rows away from its destination row guarantees (by Lemma 19) that 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. The $d + 1$ and $3d - 1$ distance bounds are guaranteed by Lemma 18. □

Corollary 26: Once a packet becomes active in some subphase, it can occupy space without moving in at most seven subphases between subphases in which it is active.

Proof: Follows from Lemma 25 (and the corresponding horizontal lemma) and observing the sequence of subphases in the algorithm. See Figure 7. □

Corollary 27: At the end of any vertical subphase (or horizontal subphase), no more than 17 packets occupy any node.

V1 V2 V3 H1 H2 H3 V1 V2 V3

Figure 7: Subphases of the algorithm. A packet can remain inactive for at most seven subphases.

Proof: From Lemma 24, at most two active packets from any subphase (vertical or horizontal) can occupy a node at the end of that subphase. From Corollary 26, 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 28: No more than $2q + 18 = 834$ packets ever occupy a node at the same time.

Proof: The lemma follows from Lemmas 21, 22, 23, 24, and Corollary 27 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 18, 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. □

6.4. 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 29: 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 27, 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 30: 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.

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 31: 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 time step $t' < t$, 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 17. 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 17, 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 32: The dimension order part of the algorithm takes no more than 14 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 18, where $d = 1$). There are at most 10 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. Thus, p can be delayed by at most 10 steps. Since p's destination is at most four nodes away from where it started the dimension order part of the algorithm, then p will arrive at its destination in at most 14 steps. □

Lemma 33: 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 29, 30, 31, and 32, 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.

$$\begin{aligned}
T(n) &\leq 2 \cdot ((qd_0 - 1) + 2 \cdot ((d_0 - 1) + qd_0) + (3n - 4)) + \\
&\quad \sum_{j=1}^{J} \left(6 \cdot \left((qd_j - 1) + 2 \cdot ((d_j - 1) + qd_j) + (3 \cdot \frac{1}{3^j} n - 4)\right)\right) + 14 \\
&= 2 \cdot (3qd_0 + 2d_0 - 7 + 3n) + 6 \cdot \sum_{j=1}^{J} \left(3qd_j + 2d_j - 7 + 3 \cdot \frac{1}{3^j} n\right) + 14 \\
&< 2 \cdot \left(\frac{1307}{27} n\right) + 6 \cdot \sum_{j=1}^{J} \left(\frac{1307}{27} \frac{1}{3^j} n\right)
\end{aligned}$$

$$< \frac{2614}{27}n + \frac{2614}{9}n \cdot \sum_{j=1}^{\infty}\left(\frac{1}{3^j}\right)$$

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 34: 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 20, 28, and 33. □

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 9 strips (that is, $\frac{1}{3 \cdot 3^j}n$ rows) of their destinations. We can now restate Lemmas 29 and 30 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 21, 22, and 28 would need to be restated with the new value of q.)

7. Conclusions and Open Problems

The main conclusion of this paper is that if one wants to route all permutations in $o(n^2/k^2)$ time on the $n \times n$ mesh with queues of size at most k, then one 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.

Some open problems include:

- Is there a matching $O(n^2/k^2)$ bound for destination-exchangeable, minimal adaptive algorithms on the mesh?
- Is there a practical routing algorithm that routes arbitrary permutations in $O(n)$ time? By "practical", we mean that the constant in the $O(n)$ time bound is small, the queue size is bounded by a small constant, the queueing discipline is simple and fast, and the algorithm extends to the asynchronous and dynamic settings.

Acknowledgement

We are grateful to Allan Borodin, Greg Plaxton, Larry Ruzzo, Larry Snyder, and Torsten Suel for helpful discussions.

References

[1] 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.

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

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

[12] 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.

[13] 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.

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

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

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

[23] 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.

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

[25] 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.

[26] 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.