Store-and-Forward Multicast Routing on the Mesh*

Kieran T. Herley
(Corresponding Author)
Dept. of Computer Science
University College Cork
Cork, Ireland
k.herley@cs.ucc.ie
Tel: +353 21 4902134
Fax: +353 21 4274390

Andrea Pietracaprina
Dip. di Ingegneria
dell'Informazione
Università di Padova
Padova, Italy
andrea.pietracaprina@unipd.it

Geppino Pucci
Dip. di Ingegneria
dell'Informazione
Università di Padova
Padova, Italy
geppino.pucci@unipd.it

December, 2006

[1]A preliminary version of this paper appeared in Proceedings of the 13$^{\text{th}}$ Annual ACM Symposium on Parallel Algorithms and Architectures, Crete, Greece, 2001. This work was supported, in part, by MIUR under project *ALGO-NEXT*.

Abstract

We study the complexity of routing a set of messages with multiple destinations (*multicast routing*) on an n-node square mesh under the store-and-forward model. A standard argument proves that $\Omega\left(\sqrt{cn}\right)$ time is required to route n messages, where each message is generated by a distinct node and at most c messages are to be delivered to any individual node. The obvious approach of simply replicating each message into the appropriate number of unicast (single-destination) messages and routing these independently does not yield an optimal algorithm. We provide both randomized and deterministic algorithms for multicast routing, which use constant-size buffers at each node. The randomized algorithm attains optimal performance, while the deterministic algorithm is slower by a factor of $O\left(\log^2 n\right)$. We also describe an optimal deterministic algorithm that, however, requires large buffers of size $O\left(c\right)$.

1 Introduction

Routing primitives for collective communication are frequently used in parallel and distributed systems when a piece of information must be shared among several hosts. A typical example is *multicast*, which requires delivering a message from its source to a number of distinct destinations. Multicast communication patterns are the message-passing counterpart of the algorithmically useful "concurrent-read" primitives supported by shared-memory models such as the PRAM [7], and arise in several parallel applications (e.g., matrix computations such as LU decomposition, Gaussian elimination and network/circuit simulations), as well as in the implementation of barrier synchronization and cache-coherence protocols. Hence, the provision of an efficient multicast primitive facilitates the implementation of such applications on distributed-memory systems.

A number of earlier studies have explored the multicast problem under the wormhole-routing switching model. Some of these works address the problem of routing a single message to several destinations and develop suitable strategies for selecting the paths that connect the message's source to the destinations and for scheduling the actual delivery along the selected paths efficiently [12, 13, 17]. Others focus mainly on determining conditions for deadlock avoidance [1] or rely exclusively on experimental analysis of the performance of the multicast routing algorithms under certain traffic conditions [11, 8, 19].

To the best of our knowledge, no analytical results are available in the literature on the time complexity of multiple-message multicast problems, except for those that can be derived through a reduction to the unicast (one destination per message) setting, which is achieved by replicating each multicast message at the source into several unicast copies to

be sent independently to the various destinations.

This paper makes an initial contribution aimed at filling this gap. Specifically, we develop upper and lower bounds on the complexity of multicast routing on the mesh under the store-and-forward model. We represent an instance of the multicast routing problem on the n-node square mesh by an $\texttt{mcast}(n, d, c)$ *message set*, a set of n messages, each associated with at most $d < n$ destinations to which a copy of the message must be delivered, subject to the constraint that no node is the destination of more than $c < n$ messages overall. It is assumed that each node is the source of exactly one message. Parameters d and c will be referred to as the *degree* and *congestion* of the message set, respectively[1].

1.1 Background

When $c = 1$, we must also have $d = 1$ and the problem reduces to standard permutation routing, which can be solved in $O(\sqrt{n})$ time with constant-size queues [9, Sec.1.7]. In a slightly different scenario, where initially there are $m \leq n$ messages (at most one per node) and messages represent robots that cannot be replicated, it is shown in [3] that multicast routing with $c = 1$ takes $\Theta\left(\sqrt{dn}\right)$ time (note that d can be greater than 1 when $m < n$).

The case where $c > 1$ requires more sophisticated techniques employing the $(d, c)-$routing problem that involves the routing a set of n messages in which each node is the source of a most d (distinct) messages and the destination of at most c messages. By adapting the (d, c)-routing strategies presented in [18, 15] to work under the initial condition that each node carries a single multicast (*i.e.* multi-destination) message rather than d unicast (*i.e.* single-destination) messages, one can achieve $O\left(\sqrt{dcn}\right)$ routing time

[1] In the literature, this problem has often been referred to as as *one-to-many* routing [9].

with constant buffers, which, as we will see, is not optimal for multicast routing.

To see how (d, c)-routing differs from multicast routing and why we might entertain hopes of solutions to the latter more efficient than the $O\left(\sqrt{dcn}\right)$ time bound attainable through (d, c)-routing, note that although a multicast routing instance does require one copy of each message be delivered to that message's various destinations, it is not necessary that all of these copies be generated at the message's source s and transported separately across the network to their respective destinations. Indeed, if two destinations t' and t'' are close to one another but far from their source s, it could be more convenient to dispatch a single message copy to some node s', which then replicates the message and sends the copies onwards to t' and t''. If s' is "close to" t' and t'' but "far from" s this should result in significantly less network traffic than the traffic generated by routing two distinct copies of the message all the way from s to t' and t''. A careful exploitation of this splitting idea will yield running times for multicast routing significantly better than those achievable by reducing it to (d, c)-routing.

Finally, it is worth pondering the relationship between multicast routing and a form of unicast routing which allows multiple messages sharing a common destination s to be combined into a single message along their way to s [16, 10]. In fact, for every instance $\mathcal{M}$ of the multicast routing problem in which each message $x \in \mathcal{M}$ must be delivered from its source $s(x)$ to its various destinations $\{t_1(x), t_2(x), \ldots, t_k(x)\}$, there is a related instance $\mathcal{M}^R$ of unicast routing in which a group of messages generated at $\{t_1(x), t_2(x), \ldots, t_k(x)\}$ need to be combined and delivered to node $s(x)$. Now, a schedule $\mathcal{S}$ for instance $\mathcal{M}^R$ that specifies the paths taken by messages, the timing of their movements, and the combinations along those paths could provide a solution for the multicast instance $\mathcal{M}$, if the schedule were run in reverse (with combining operations in $\mathcal{S}$ replaced by splittings). However,

none of the known algorithms for unicast routing (with combining) provides the basis for an efficient multicast algorithm. For example, the work by [10] is largely concerned with scheduling the movement of packets along pre-selected paths; the path selection techniques employed (largely based on greedy path selection and random intermediate destinations) do not guarantee that the paths offer optimal performance when combining is allowed.

1.2 Our contributions

By combining simple diameter and bandwidth considerations we show that an arbitrary $\texttt{mcast}(n, d, c)$ message set requires $\Omega\left(\sqrt{cn}\right)$ time to be routed (Theorem 1).

If $O\left(c\right)$ messages can be stored at a node, we show that a suitable routing strategy (similar to the one adopted in [6] in the context of shared-memory simulations), which replicates messages only when they are sufficiently close to their destinations, exhibits optimal performance in the worst case (Theorem 2). For large values of c, this algorithm is impractical because of the large buffer size.

Achieving optimal routing time using constant-size buffers, which is highly desirable from a practical standpoint, is much harder. Under this limitation, we develop an optimal randomized algorithm, and a deterministic algorithm whose performance is an $O\left(\log^2 n\right)$ factor away from optimal (Theorems 3 and 4, respectively). Both algorithms rely on partitioning the messages into groups, so that messages with destinations within (suitably small) regions of the mesh are likely to be evenly distributed among the groups. Such a distribution is obtained in the probabilistic setting by randomly assigning messages to groups, and in the deterministic setting by employing a more complex scheme based on a replicated assignment of copies of messages to groups that is governed by highly expanding graphs.

It is interesting to note that, unlike the case of (d, c)-routing with unicast messages, the results proved in this paper show that the routing time for an arbitrary $\texttt{mcast}(n, d, c)$ message set is independent of parameter d. This is mainly explained by the observation that while an instance of (d, c)-routing may involve up to dn *distinct* messages that need to be transported individually across the width of the network, an $\texttt{mcast}(n, d, c)$ message set consists of only n messages (albeit with up to d destinations each) for which the splitting technique sketched in the Introduction can be exploited to reduce the amount of network activity entailed in its delivery.

The rest of the paper is organized as follows. Section 2 introduces our machine model and describes a number of basic primitives that will be employed by our algorithms. Section 3, presents the lower bound and the simple time-optimal deterministic algorithm that uses buffers of size $O(c)$. Sections 4 and 5 present, respectively, the randomized and deterministic algorithms with constant-size buffers. Our algorithms assume that the congestion of the message set c is given as an input parameter. In Section 6 we briefly discuss how this assumption can be removed without affecting the asymptotic running time. Finally, Section 7 offers some concluding remarks.

2 Technical setting

2.1 Machine model

Our reference machine is an n-node square two-dimensional mesh. We assume that every node of the mesh has an *input queue*, an *output queue* and a *buffer*. Each message is located initially in the input queue of its source node and remains there until it is injected into the network. The message travels wrapped in a *packet* consisting of a header, which encodes

the destination addresses, and a payload, which contains the actual message body. Once a packet reaches a destination node, the associated message is copied into the output queue of that node and the packet is removed from the network. While travelling, a packet can be held only in buffers. We will refer to buffer sizes in terms of the number of packets they can accommodate.

The mesh is synchronous with time divided into *steps*. We assume that in a single step either the transmission of a packet across a link or the creation of a copy of a packet/message can be accomplished. This assumption is realistic in a setting in which all messages have approximately the same size and account for a non-negligible fraction of the size of their respective packets, independently of the number of destinations. Moreover, we assume that in one step a node can execute a constant number of elementary operations on a packet, including inspecting its destination addresses. Although the time required by this latter activity may depend on d, we make the realistic assumption that this is dominated by the time for packet transmission, upon which our notion of time is based.

In the algorithms, we often make use of tessellations of the mesh into square submeshes of equal size. When the mesh is tessellated into t square submeshes of n/t nodes each, we will call each such submesh a *t-tile*. Throughout the paper, tessellations with different values of t are used, where t is a function of the parameters of the routing instance[2]. Moreover we always assume that the mesh nodes as well as the tiles of a given tessellation are numbered according to some natural ordering (e.g., row-major or snake-like) starting from 0.

Our algorithms often require all packets located at a particular submesh to visit all of

[2]For convenience, in the presentation and analysis of the algorithms floors and ceilings are omitted under an implicit assumption that the quantities involved are integral. This will not affect the asymptotics as we shall show.

the nodes in the submesh in turn, or requires groups of packets located in distinct tiles of a given tessellation to visit all of the tiles. These movements can easily be orchestrated in a Hamiltonian fashion thanks to the following straightforward property.

Fact 1 *There exists a one-to-one mapping of the nodes of an m-node ring into the nodes of an m-node mesh in such a way that each pair of adjacent ring nodes are mapped to mesh nodes that are at most a distance of two apart in the mesh.*

In the reminder of the paper, we will use the expression Hamiltonian cycle of a mesh/submesh in a loose way to refer to the ring embedding claimed in Fact 1.

2.2 Basic primitives

Our algorithms will make use of a number of basic primitives which are briefly described below. Consider n packets encapsulating the messages of an $\texttt{mcast}(n, d, c)$ message set and originating in distinct nodes. The following two propositions state useful well-known results regarding the mesh.

Proposition 1 ([9, Ch.1]) *Sorting a set of n packets according to some specified packet field can be accomplished in $O(\sqrt{n})$ time on the mesh. A prefix computation on n values can be completed in the same amount of time.*

Both algorithms assume that the inputs are initially distributed one per node.

Proposition 2 ([2]) *Any instance of (k, k)-*routing*, where each node is the source and the destination of at most k unicast messages can be performed in $\Theta(k\sqrt{n})$ time using buffers of size $O(k)$.*

We will use the term *permutation routing* when referring to instances of (k, k)-routing with $k = 1$.

Consider a tessellation of the mesh into t-tiles, for some t. We define the *t-migration* primitive as follows. Let S be a set of packets, each with a number of destinations. The *t-migration* problem involves the selection, for each tile, of n/t packets with destinations in the tile (or all packets with destination in the tile if they are less than n/t), and the routing of a copy of each packet (*packet-copy*) to every tile for which the packet was selected. Each packet-copy is routed to some *intermediate destination* in its destination t-tile, so that each node receives at most one packet-copy (and each t-tile as a whole receives at most n/t packet-copies). The t-migration primitive can be executed as follows:

1. Execute t prefix computations, where for the i-th prefix each packet contributes a value 0 or 1 depending on whether it has destinations in the i-th tile (with tiles numbered in some natural fashion). At the end of the i-th prefix a packet will be *marked* as selected for the i-th tile if its prefix value is at most n/t. (Observe that the space needed for representing the marks assigned to a packet never exceeds the space used to represent its destinations.)
2. Let m_j be the number of marks assigned to the packet originating in node j. Create m_j copies of the packet (packet-copies) as follows.
   (a) Route the packet to the node with index $\sum_{\ell=0}^{j-1} m_\ell$, which can be determined through a prefix computation on the m_js using permutation routing. (We assume here that the nodes are numbered in some suitable snake-like order.)
   (b) The positioning of the packets along the sequence of nodes determined by the snake-like ordering, effectively partitions the sequence into *segments*, with each packet positioned at the beginning of "its" segment, which contains m_j consecutive nodes.
   (c) Broadcast a copy of each packet (packet-copy) to every node of its corresponding segment. Associate the packet-copies with the distinct tiles for which the packet was selected.
3. Sort the packet-copies according to their destination tiles and assign distinct ranks to packet-copies destined to the same tile, using a prefix computation. Then execute

a permutation routing to send each packet-copy to its associated tile, using its rank as destination within the tile.

Proposition 3 *The t-migration primitive can be performed in $O(t+\sqrt{n})$ time*

Proof: Step 1 can be executed by pipelining t prefix computations, for a total time of $O(t+\sqrt{n})$. All other steps require a number of prefix, sorting and permutation routing primitives, thus contributing an additional $O(\sqrt{n})$ term to the running time. Note that the broadcasts of the packets to all of the nodes of their corresponding segments, performed in Substep 2.(c), can be implemented as a single instance of a prefix computation using a suitable associative operator [9, Sec.1.2.1]. □

3 Preliminary results

The following theorem provides a lower bound on the worst-case performance of any multicast routing algorithm. The proof is based on a standard congestion argument, which is well known in the mesh-routing literature.

Theorem 1 *For any setting of parameters n, d, c with $1 \leq d, c < n$, there exists an* `mcast`(n, d, c) *message set such that the time required to deliver all of the messages to their destinations is $\Omega(\sqrt{cn})$.*

Proof: Consider a square submesh A of side $\max\{1, \lfloor\sqrt{n/(2c)}\rfloor\}$, and let B denote the rest of the mesh. Note that $|A| = \Theta(n/c)$ and $|B| \geq c|A|$. Consider an instance of `mcast`(n, d, c) where $c|A|$ messages originating at nodes of B have exactly one destination each in A. Clearly, $\Omega\left(c\sqrt{|A|}\right) = \Omega(\sqrt{cn})$ time is needed for all of these messages to enter A. □

Despite the fact that in a general instance of multicast the relative magnitude of d and c is arbitrary, our algorithms are designed to work for the case $d \leq c$. The following lemma shows that there is no loss of generality by restricting our attention to this case.

Lemma 1 *The routing of an* $\texttt{mcast}(n, d, c)$ *message set with* $d > c$ *can be reduced to the routing of at most two* $\texttt{mcast}(n, c, c)$ *message sets in* $O(\sqrt{n})$ *time using constant-size buffers.*

Proof: For $0 \leq i < n$, refer to the packet originating at node i as the i-th packet, and let $\ell_i = \lceil d_i/c \rceil$, where d_i denotes the number of destinations of this packet. Using the same technique as that employed in Step 2 of the t-migration algorithm, the reduction strategy creates a copy of the i-th packet in every node of index $(\sum_{j=0}^{i-1} \ell_j + k) \bmod n$, with $0 \leq k < \ell_i$, assigning (at most) c distinct destinations of the original packet to each copy. Observe that $\sum_{j=0}^{n-1} d_j \leq cn$, or otherwise there would be a node which is a destination for more than c messages. Thus

$$\sum_{j=0}^{n-1} \ell_j \leq \sum_{j=0}^{n-1} (1 + d_j/c) \leq 2n,$$

hence each node receives at most 2 copies of packets. The whole process can be accomplished in $O(\sqrt{n})$ time using constant space at each node, using prefix, segmented broadcast and routing primitives (see Propositions 1 and 3). □

We now present a simple deterministic algorithm for routing any $\texttt{mcast}(n, d, c)$ message set, with $d \leq c$, in optimal $O(\sqrt{cn})$ time, which, however, requires buffers of size $\Theta(c)$. The algorithm, described below, makes use of a series of $\log_4 c + 1$ nested tessellations of the mesh into tiles of increasing fineness, namely 4^i-tiles, for $0 \leq i \leq \log_4 c$. Note that for

$0 \leq i \leq \log_4 c$, every 4^i-tile consists of four quadrants which are 4^{i+1}-tiles. We refer to a node in a 4^{i+1}-tile and the corresponding nodes in the same relative positions in the other three 4^{i+1}-tiles within the same 4^i-tile as 4^i-*siblings*.

1. Replicate the entire set of packets in every c-tile as follows. For $0 \leq i < \log_4 c$ in turn, send copies of the 4^i packets located at each node to the 3 nodes that are its 4^i-siblings within the same 4^i-tile. After the last iteration each node contains $O(c)$ packets.
2. In parallel within each c-tile do the following:
   (a) For each packet retain only the addresses related to destinations within the tile, stripping out the other addresses. Partition the packets into classes determined by the number of their destinations within the tile. More precisely, let class C_j comprise all packets with between 2^j and $2^{j+1} - 1$ destinations within the tile, for $0 \leq j \leq \lfloor \log_2 d \rfloor$.
   (b) Execute $\lfloor \log_2 d \rfloor + 1$ prefix computations to rank separately the packets of each class, and use the ranks to assign a *intermediate node* to each packet, in order to spread the packets of each individual class evenly among the nodes of the tile. More precisely, a packet with rank r in class C_j will be assigned the $(r \bmod (n/c))$-th node of the tile as an intermediate node.
   (c) Route each packet to its intermediate node.
   (d) Within each node, for every packet received in the previous step, create as many copies as the packet has destinations within the tile, assigning a distinct destination to each copy. We will refer to the copies as *clones*.
   (e) Route each clone to its final destination.

Theorem 2 *The above algorithm correctly routes any* `mcast`(n, d, c) *message set, with* $d \leq c$*, in optimal* $\Theta(\sqrt{cn})$ *time, using buffers of size* $\Theta(c)$ *at each node.*

Proof: Correctness follows immediately by observing that the entire message set is replicated in every c-tile and within a tile a copy (clone) of a packet is dispatched to each of the packet's destinations in the tile.

For $0 \leq i < \log_4 c$, Iteration i of Step 1 consists of an instance of $(4^i, 3 \cdot 4^i)$-routing to be executed within a 4^i-tile since each node holds 4^i packets to be forwarded to its three

siblings. By Proposition 2 this takes $O\left(4^i\sqrt{n/4^i}\right)$ time. Since the running times of the $\log_4 c$ iterations form a geometric series, the overall running time for Step 1 is $O\left(\sqrt{cn}\right)$[3]. Step 2.(a) involves no packet movement. Step 2.(b) can be executed by pipelining the required $\lfloor \log_2 d \rfloor + 1$ prefix computations in time $O\left(\log_2 d + \sqrt{n/c}\right)$. The ranking ensures that when packets are sent to their intermediate nodes, each node receives

$$\sum_{j=0}^{\lfloor \log_2 d \rfloor} \lceil |C_j|/(n/c) \rceil \leq 1 + \log_2 d + c = O\left(c\right)$$

packets, since $\sum_{j=0}^{\lfloor \log_2 d \rfloor} |C_j| \leq n$ and $d \leq c$. Therefore Step 2.(c) requires $\left(O\left(c\right), O\left(c\right)\right)$-routing within each c-tile, which takes $O\left(\sqrt{cn}\right)$ time. As for Step 2.(d), we observe that since each packet in C_j contributes at least 2^j clones and each node will eventually receive at most c clones, we have that $\sum_{j=0}^{\lfloor \log_2 d \rfloor} 2^j |C_j| \leq c(n/c) = n$. By virtue of the redistribution, in Step 2.(d) each node will create

$$\sum_{j=0}^{\lfloor \log_2 d \rfloor} (2^{j+1} - 1) \left\lceil \frac{|C_j|}{n/c} \right\rceil = O\left(c\right)$$

clones in total. Hence the final routing in Step 2.(e) is again an instance of $\left(O\left(c\right), O\left(c\right)\right)$-routing within c-tiles, which takes $O\left(\sqrt{cn}\right)$ time. The theorem follows by adding up the contributions of all the steps and noticing that no node receives more that $O\left(c\right)$ packets at any time during the algorithm. □

[3]This description implicitly assumes that the $\sqrt{n/4^i}$ quantities are integral. It should be clear however that the techniques extend to arbitrary n at the cost of a small constant factor per iteration. Note the overall cost of Step 1 is formed by summing those of the various iterations and so that these small constant factors cannot accumulate in a bad way. Similar reasoning can be applied to all the other results in the paper that involve involve implicit assumptions of the integrality of the size of the submeshes involved.

4 Randomized Algorithm

We now present an optimal randomized algorithm for routing an arbitrary $\texttt{mcast}(n, d, c)$ message set with $d \leq c$ and constant buffer size.

The algorithm is organized in *phases*. In each phase the mesh is tessellated into tiles of a suitable size, and the packets with destinations that have not been visited in previous phases are randomly partitioned into a number of *groups* equal to the number of tiles. Each group of packets is first routed to a distinct tile and then circulated around the tiles (in a Hamiltonian fashion), allowing the packets in the group to attempt reaching their destinations within a tile as the group "visits" that tile. Most of these packet deliveries will succeed due to the fact that the random assignment of packets to groups ensures that "competitors" of a packet with a destination x in a tile C, meaning those other packets that also have to be delivered to destinations in a given neighbourhood of x (called a *block*), are evenly distributed among the various groups. Therefore, it is unlikely that the delivery of the packet to x will be frustrated by above-average contention due to such competitors. At the end of the phase, the total number of uncompleted deliveries will be substantially reduced.

Let us now describe the algorithm in detail. Without loss of generality, assume that $c \leq n/16$, since otherwise the strategy where all packets circulate along a Hamiltonian cycle of the mesh suffices to achieve optimal performance. (We assume for now that the value of c is known to the nodes. We discuss how to remove this assumption in Section 6.) There are $K + 1$ phases, numbered from 0 to K, where $K = \log_{16}(\lceil c \log^2 n/n \rceil)$. (Note that $K = 0$ for $c \leq n/\log^2 n$.) For $0 \leq i \leq K$, let $c_i = c/16^i$, and consider a tessellation of the mesh into c_i-tiles $\{T^i_j : 0 \leq j < c_i\}$, where the numbering identifies a Hamiltonian

cycle of the tiles.

In turn, consider a tessellation of each c_i-tile into $b_i = \sqrt{n/c_i}$ square subtiles of b_i nodes each, referred to as b_i-*blocks*. (See Fig. 1.) Note that since we assumed $c \leq n/16$, $b_i \geq 4$ for every $i \geq 0$.

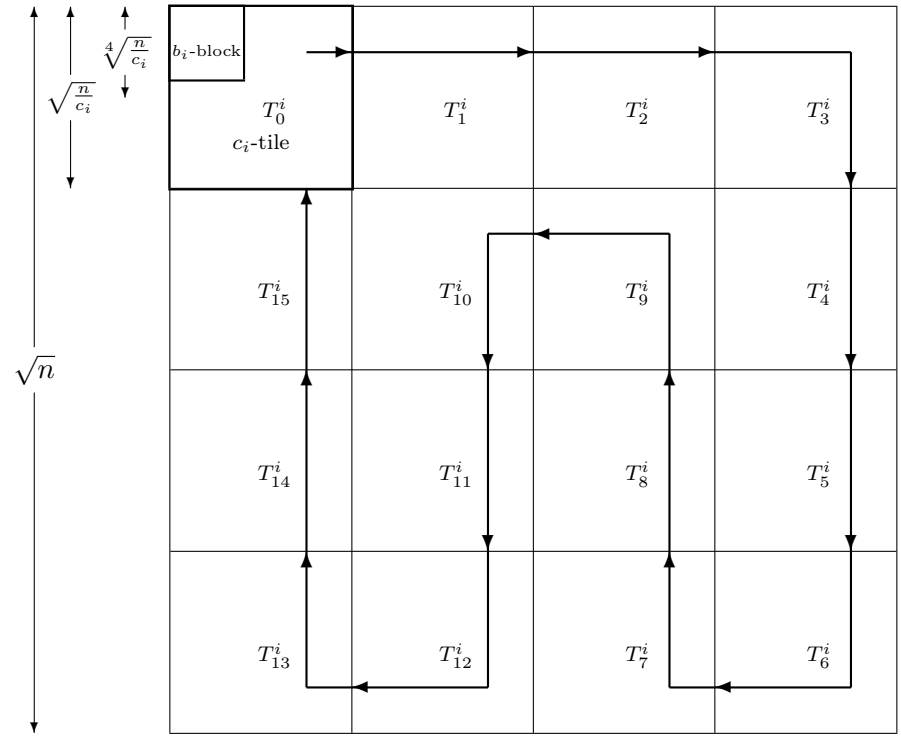

Figure 1: The double tessellation of the mesh employed in Phase i of the randomized algorithm. Observe that the numbering of the c_i-tiles follows the Hamiltonian cycle indicated by the solid path.

For $0 \leq i \leq K$, Phase i consists of the following steps.

1. Partition the packets into c_i *groups* $G_1, G_2, \ldots, G_{c_i}$ each containing at most $2n/c_i$ packets and distribute each group evenly among the nodes of a distinct c_i-tile as follows.

   (a) Partition the packets into c_i *tentative groups* $G'_0, G'_1, \ldots G'_{c_i-1}$, by assigning each packet to a group with uniform probability $1/c_i$.

   (b) Sort the packets by their tentative group index and rank the packets separately within each tentative group using a segmented prefix computation.

   (c) Partition the packets of each tentative group into segments of size n/c_i (allowing for one "incomplete" segment per group) using prefix computations. Observe that the total number of segments among all tentative groups is $\sum_{j=0}^{c_i-1} \lceil |G'_j|/(n/c_i) \rceil \leq 2c_i$, and let these segments be globally numbered from 0 onwards.

   (d) For $0 \leq j < c_i$, assign packets belonging to the two segments of indices j and $j + c_i$ to group G_j.

   (e) Sort the packets by their group index and rank them separately within each group using prefix computations. For each group G_j, use the ranks of the packets to assign intermediate nodes within tile T_j^i so that at most 2 packets are assigned to the same intermediate node.

   (f) For $0 \leq j < c_i$, send all packets in group G_j to their intermediate nodes in tile T_j^i using permutation routing.

2. In parallel for each cell T_j^i, repeat the following sequence of steps c_i times:

   (a) Let r be a suitable constant to be specified later. Run r instances of the t-migration primitive with $t = b_i$ so that each b_i-block of T_j receives at most rb_i copies of packets (packet-copies) that have destinations within that block and that these packet-copies are distributed evenly among the nodes of the block. The r instances of the primitive must be executed in such a way that a packet is selected no more than once for the same b_i-block. Also, retain the original copy (master copy) of each packet at the node where it was located at the beginning of this substep.

   (b) Within each b_i-block, circulate the packet-copies received in the previous substep along a Hamiltonian cycle of the block, to allow them to reach all of their destinations in the block.

   (c) Discard all packet-copies created in Substep 2.(a) and shift the retained master copies to the same relative positions in the next cell T_{j+1}^i mod c_i.

3. If $i = K$, repeat Step 2 while there are deliveries yet to be completed.

The correctness of the above algorithm is ensured by Step 3. In order to get a probabilistic bound on the running time, we need the following two lemmas.

Lemma 2 *For every $0 \leq i \leq K$, at the beginning of Phase i, the number of packets with unvisited destinations within any given b_i-block is at most $b_i c_i$, with high probability.*

Proof: We bound from above the probability that there exists a phase for which the stated property does not hold. Note that such a phase cannot be Phase 0, since there are at most $c = c_0$ packets destined to any mesh node in the message set. Suppose that the first phase for which the property does not hold is Phase $i+1$, with $i \geq 0$, and let B be a b_{i+1}-block such that at the beginning of the phase there are more than $b_{i+1}c_{i+1}$ packets with unvisited destinations in B. Then, it is easy to see that there must be at least one of the four b_i-blocks contained in B, say B', for which more than $b_i c_i/16$ packets with unvisited destinations in the block were not selected in Step 2.(a) of Phase i. This implies that in Phase i these packets belonged to groups where the maximum number of packets for B', namely rb_i, were selected. Since at the beginning of Phase i there were at most $b_i c_i$ packets in total with unvisited destinations in B', the unselected packets cannot belong to more than $b_i c_i / rb_i = c_i/r$ groups, hence to more than $c_i/(r/2)$ tentative groups. The probability of this event is bounded above by

$$\begin{aligned}
& \binom{b_i c_i}{b_i c_i/16} \binom{c_i}{c_i/(r/2)} \left(\frac{2}{r}\right)^{b_i c_i/16} \\
\leq & \left(\frac{e b_i c_i}{b_i c_i/16}\right)^{b_i c_i/16} \left(\frac{e c_i}{c_i/(r/2)}\right)^{c_i/(r/2)} \left(\frac{2}{r}\right)^{b_i c_i/16} \\
\leq & \left(16e \left(\frac{er}{2}\right)^{16/(b_i r/2)} \frac{2}{r}\right)^{b_i c_i/16}
\end{aligned}$$

$$\leq \left(\frac{4}{5}\right)^{\sqrt{nc_i}/16},$$

where the last inequality is obtained by using the fact that b_i is always at least 4 and by fixing $r \geq 150$. (No attempt is made here to optimize the constants.) Since the above probability is exponentially small in n, the lemma follows by applying the union bound over the at most polynomially many phases and blocks. □

Lemma 3 *In Phase K, all remaining deliveries are made in the first execution of Step 2, with high probability.*

Proof: Let B be an arbitrary b_K-block and G' an arbitrary tentative group formed in Step 1.(a). By Lemma 2, at the beginning of Phase K there are at most $b_K c_K$ packets with unvisited destinations in B, with high probability, and each such packet is assigned to G' independently with probability $1/c_K$. By Chernoff's bound [4] the probability that G' contains more than $rb_K/2$ packets with unvisited destinations in B is at most $\left(e^{\delta}/(1+\delta)^{1+\delta}\right)^{\mu}$ (where $\mu = b_K$ and $1+\delta = r/2$), which is less than $\left(\frac{2e}{r}\right)^{rb_K/2}$. Since $b_K = \Omega(\log n)$ by our choice of K, it is clear that by choosing a large enough value for constant r and by applying the union bound over all blocks and groups, the probability that there exists a tentative group with more than $rb_K/2$ packets with unvisited destinations in the same block can be made polynomially small. This implies that the routing is completed after the first execution of Step 2, with high probability. □

The algorithm may be implemented efficiently, as the following theorem demonstrates.

Theorem 3 *The above randomized algorithm routes any* `mcast`(n, d, c) *message set with $d \leq c$ in optimal time $O(\sqrt{cn})$ with high probability, using buffers of size $O(1)$ at each node.*

Proof: Consider Phase i. As indicated in the individual substeps, Step 1 can be implemented using a constant number of sorting, permutation routing and prefix primitives requiring $O(\sqrt{n})$ time. Within each c_i-tile, every iteration of Step 2 involves: a constant number of calls to the t-migration primitive, with $t = b_i$ (Substep 2.(a)); a complete tour of the nodes within each b_i-block along a Hamiltonian cycle (Substep 2.(b)), and the movement of the packets in the tile to the adjacent tile (Substep 2.(c)). By applying Proposition 3, each iteration takes $O\left(b_i + \sqrt{n/c_i}\right) = O\left(\sqrt{n/c_i}\right)$ time. Hence, Step 2 requires $O\left(c_i\sqrt{n/c_i}\right) = O\left(\sqrt{c_i n}\right)$ time altogether. The stated running time follows since, by Lemma 3, in Phase K Step 2 is executed only once, with high probability.

As for the buffer size, observe that Step 1 of the algorithm begins with one packet per node and ends with at most two per node and that the constituent substeps involving segmented prefix and sorting operations never load more than $O(1)$ packets per node. Step 2 begins with $O(1)$ packets per node; the migration primitive of Substep 2.(a) requires only $O(1)$ space per node and ensures that each node receives $O(1)$ packet-copies; the remaining substeps, Substeps 2.(b) and Substep 2.(c), similarly require only $O(1)$ space per node. Hence, buffers of constant size suffice to run the algorithm. □

5 Deterministic Algorithm

In this section we develop a deterministic algorithm that routes any $\texttt{mcast}(n, d, c)$ message set, with $d \leq c$, in time $O\left(\sqrt{cn}\log^2 n\right)$ using buffers of constant size. Throughout the section, we assume that $c \geq \log^4 n$, since otherwise the same upper bound on the routing time can be achieved by employing the known algorithms for (d, c)-routing with constant-sized buffers, as described in the Introduction.

The idea underlying the deterministic algorithm is somewhat similar to the one at the core of the randomized algorithm described in the previous section: divide the packets into groups, so that message deliveries to any given block within a single group create less congestion than in the original problem. However, we cannot rely on randomization to spread deliveries to a block evenly among the groups, hence we must resort to a more complex scheme based on a replicated assignment of copies of packets to groups, regulated by the highly expanding graphs defined below.

Definition 1 *For constant λ, $0 < \lambda < 1$ and integers r and c, with $1 \leq r \leq c \leq m$, a regular bipartite graph*[4] *$\mathcal{G} = (U, V; E)$ is an* (m, c, r, λ)-expander *if $|U| = m$, $|V| = c < m$, each node in U (resp., V) has degree r (resp., rm/c), and for every $S \subseteq U$ with $|S| \leq |V|/r$, the edges emanating from S are incident upon a set $\Gamma(S)$ of nodes of V of size $|\Gamma(S)| \geq \lambda |S| r$.*

The existence of such graphs can be proved using the probabilistic method, by resorting to the natural correspondence between the set of bipartite graphs $\mathcal{G} = (U, V; E)$ with $|U| = m$, $|V| = c \leq m$ and $\deg(u) = r$ for all $u \in U$, and the group Σ_{rm} of permutations of $(0, 1, \ldots, rm - 1)$. Under this correspondence, any permutation $\pi \in \Sigma_{rm}$ represents the graph $\mathcal{G} = (U, V; E)$, where node $u_i \in U$, for $0 \leq i < m$, is adjacent to the nodes of V of indices $\lfloor c\pi\left[ir + j\right]/(rm) \rfloor$, for $0 \leq j < r$. When $c = \Omega(\log m)$ and $r = \Theta(\log m)$, it can be shown that a random permutation in Σ_{rm} represents an (m, c, r, λ)-expander, for a suitable constant λ, with probability at least $1 - 1/\mathrm{poly}(c)$. (See [5, 14] for instances of such proofs.)

Given an $\texttt{mcast}(n, d, c)$ message set, an $(n/\log n, c, r, \lambda)$-expander $\mathcal{G} = (U, V; E)$ with

[4]For technical reasons, we allow the possibility of two or more edges linking the same pair of nodes. Hence our "graphs" are technically multi-graphs. Since this will not affect the subsequent discussion in any material way, for simplicity we will use the term graph in the sequel.

$r = \Theta(\log n)$ is employed in the deterministic algorithm as follows. Let $\pi \in \Sigma_{rn/\log n}$ denote the permutation representing $\mathcal{G}$ and assume that the entries of π are evenly distributed among the mesh nodes by storing entry $\pi[k]$ at the mesh node of index $k \bmod n$, which requires only constant space per node. (Note that the set of neighbours of set $S \subset U$ with size at most n/r may be probed in $O(\sqrt{n})$ time using permutation routing.) We partition the message set arbitrarily into $\log n$ *classes* of $n/\log n$ messages each. We consider a tessellation of the mesh into c-tiles $\{T_j : 0 \leq j < c\}$, each of size n/c, where the numbering identifies a Hamiltonian cycle of the tiles. In turn, each c-tile is partitioned into $b = \sqrt{n/c}$ square subtiles of b nodes each, referred to as *b-blocks*.

We route each class separately in a number of phases. Consider the routing of an arbitrary class. In the first phase, we execute the following steps.

1. Create r *replicas* of each packet and assign them to r of c *groups* based on the expander $\mathcal{G} = (U, V; E)$ using ideas similar to those of Step 2 of the algorithm of Proposition 3 as follows. Let the i-th packet of the class correspond to node u_i of U, and the ℓ-th group G_ℓ correspond to node v_ℓ of V.
   (a) Using a prefix computation and permutation routing, spread the packets among the mesh nodes so that each packet is positioned at the beginning of a segment of r nodes and no node belongs to more than $O(1)$ segments.
   (b) Create the replicas of each packet, one for each node of its assigned segment, using a segmented broadcast.
   (c) Using permutation routing, route the j-th replica of the i-th packet to the node of index $((i-1)r + j) \bmod n$ (which stores the value $\pi[(i-1)r + j]$) so that the replica can be assigned to group G_ℓ with $\ell = \lfloor c\pi[(i-1)r + j] / (rn/\log n) \rfloor$. Note that each group formed in this fashion comprises $\Theta(n/c)$ replicas.
2. Sort the replicas by their group index and rank them separately within each group. For each group G_ℓ, use the ranks to assign intermediate nodes within tile T_ℓ so that at most $O(1)$ replicas are given the same intermediate node.
3. For $0 \leq \ell < c$, send all replicas in group G_ℓ to their intermediate nodes in tile T_ℓ using permutation routing.

4. In parallel for each tile T_ℓ, repeat the following steps c times:

   (a) Let $\alpha > 1/\lambda$ be a constant. Repeat the following three substeps r times.

      i. Perform α instances of the t-migration operation. This involves creating copies of replicas (hereinafter referred to as clones), each clone of replica x corresponding to some block for which x has unvisited destinations, and the subsequent selection and movement of clones to their intended blocks in such a way that at most αb clones are delivered to each block and each block node receives $O(1)$ of those clones. The various instances of the primitive must be executed making sure that a clone is selected no more than once for the same b-block.

      ii. Within each b-block, circulate the clones received in the previous substep along a Hamiltonian cycle of the block, to allow them to reach all of their destinations in the block.

      iii. Discard all clones created in Substep 4.(a).i, marking in each replica the destinations visited during Substep 4.(a).ii using sorting and prefix operations.

   (b) Shift the replicas to the same relative positions in the next tile $T_{\ell+1} \bmod c$.

5. Route all replicas back to the positions they occupied upon creation (i.e., at the end of Substep 1.(b) using permutation routing.

6. Coalesce the r replicas of each packet into a single packet, thus identifying all destinations that are still unvisited. (Note that a destination for a packet is unvisited if it was not marked as visited in one of the replicas during Substep 4.(a).iii.) This is similar to the replica-creation process of Step 1 and involves the same algorithmic ingredients.

The subsequent phases consist of repetitions of Steps 1–6, with the exception that in Substep 4.(a) we perform only one iteration rather than r, thus selecting αb packets for each block rather than αbr.

The next lemma provides a bound on the number of phases needed to complete all packet deliveries in a class. We say that a packet is *alive* for a b-block at the end of a phase if it contains unvisited destinations within that block.

Lemma 4 *For $i \geq 1$, at the end of the i-th phase, the number of packets alive for each block is at most $c/(rb^{i-1})$.*

Proof: Let us first prove the claim for the first phase. Suppose that at the end of Step 6 there are more than c/r packets alive for a given block B, and consider a subset of exactly c/r such packets. By the expansion properties of $\mathcal{G}$, in Step 1 the replicas of these packets were spread among at least λc groups. Since none of the clones of these replicas was selected for B in the appropriate iterations of Substep 4.(a) (i.e., when the respective groups were in the tile containing B), it must be the case that at least $\lambda c(\alpha br) > cbr$ clones of other packets with destinations in B were selected. This is a contradiction, since the overall number of clones with destinations in the block cannot exceed cbr.

A similar argument shows that in each of the remaining phases the total number of packets alive for any given block decreases by a factor b. □

Lemma 4 guarantees that $\log_b(bc/r) = O(\log n/\log(n/c)) = O(\log n)$ phases are sufficient to complete the routing of each class. We are now ready to prove the main result of this section.

Theorem 4 *The above deterministic algorithm routes any* `mcast`(n, d, c) *message set with* $d \leq c$ *in time* $O\left(\sqrt{cn}\log^2 n\right)$ *using buffers of size* $O(1)$ *at each node.*

Proof: Consider the first phase of the algorithm. Steps 1, 2, 3, 5, and 6 are all implemented using a constant number of calls to sorting, (segmented) prefix computations and permutation routing, for a total running time of $O(\sqrt{n})$ with constant size buffers. Within each c-tile, every iteration of Step 4 involves: $\Theta(r)$ invocations of the t-migration primitive with $t = b$ (Substep 4.(a).i); r complete tours of the nodes within each b-block along a Hamiltonian cycle (Substep 4.(a).ii); r sortings and segmented prefix computations in order to mark the visited destinations of each replica (Substep 4.(a).iii); and, finally, the movement of the replicas to the adjacent tile (Substep 4.(b)). Hence, by applying Proposition 3, the

overall time for Step 4 is $O\left(cr(b+\sqrt{n/c})\right) = O\left(r\sqrt{cn}\right)$, using buffers of constant size. In summary, the first phase can be completed in time $O\left(\sqrt{cn}\log n\right)$. For the subsequent phases, observe that we perform a single iteration in Step 4.(a), hence the overall running time of each such phase is $O\left(\sqrt{cn}\right)$. Since Lemma 4 guarantees that $O\left(\log n\right)$ phases suffice, it follows that the routing of a single class is performed in time $O\left(\sqrt{cn}\log n\right)$. By summing over the $\log n$ classes, we conclude that the entire message set is routed in time $O\left(\sqrt{cn}\log^2 n\right)$. □

In the above algorithm we made the implicit assumption that the permutation π representing the expander graph $\mathcal{G}$ is available at the mesh nodes. However, it would seem that a different expander is needed for every congestion value c, and therefore the time to generate the permutation should be included in the analysis. However, as an immediate corollary of the fact that a random permutation in $\Sigma_{rn/\log n}$ represents an $(n/\log n, c, r, \lambda)$-expander, for a suitable constant λ, with probability at least $1 - 1/\text{poly}(c)$, it follows that the same random permutation represents a family of graphs $\{\mathcal{G}_k = (U, V_k; E_k) \ : \ 0 \leq k \leq \log n - 4\log\log n\}$, where $\mathcal{G}_k$ is an $(n/\log n, c_k, r, \lambda)$-expander, with the c_ks forming a geometric series of values ranging from $\log^4 n$ to n, with probability at least $1 - 1/\text{poly}(\log n)$. Therefore, there exists a *single* permutation that can be used to route any message set irrespective of its congestion. One such permutation can be stored in a distributed fashion among the mesh nodes and can be probed in $O\left(\sqrt{n}\right)$ time.

6 Dealing with unknown congestion values

The algorithms presented in the previous sections assume prior knowledge of the congestion value c, which is generally not the case in a practical context. As explained below, this

assumption can be removed for all of our algorithms without increasing the overall running time asymptotically. First, the average congestion $\bar{c}$ is determined as the total number of message deliveries divided by n. Then, the algorithm is run for geometrically increasing values of congestion, starting from $\bar{c}$. An execution for a given guessed congestion value c' is aborted when either the running time goes beyond a certain threshold proportional to the target time for the current guess, or the allotted buffer capacity (depending on the algorithm) is exceeded. Clearly, the algorithm terminates when the actual congestion value is guessed. Note that the sequence of running times of the various executions forms a geometrically increasing series dominated by its last term, hence it will be of the same order as the target time for the actual congestion value.

7 Conclusions

In this paper we presented several strategies for routing multicast message sets on meshes under the store-and-forward switching model. More specifically, we devised both randomized and deterministic time-optimal algorithms. However, while the former requires only constant-size buffers, the latter makes use of buffers whose size can be as large as the congestion of the message set. In order to perform the routing deterministically with constant buffers, we devised a more involved algorithm whose running time, however, is a polylogarithmic factor away from optimal. A further limitation of this algorithm is represented by its use of highly expanding graphs whose existence can be proved through the probabilistic method, although their explicit construction is notoriously hard.

The complexity gap between the randomized and deterministic cases is intriguing since no such gap exists for most other mesh problems (one notable exception being the PRAM

simulation problem as shown in [6]). Improving the running time of the deterministic algorithm to close this gap and avoiding the use of complex expanding graphs are interesting open problems. It is also interesting to extend our results to other topologies or to different routing models such as the robot model of [3] which prohibits replication and for which only the case of unit congestion has been considered so far.

8 Acknowledgments

The authors are very grateful to one anonymous referee for suggesting a number of helpful changes to improve the clarity of presentation of the results in this paper. We would like to dedicate this paper to our friend and colleague Jop Sibeyn and warmly salute his contribution to the state of knowledge in the field of mesh algorithms.

References

[1] E. Fleury and P. Fraigniaud. Strategies for multicasting in wormhole-routed meshes. *Journal of Parallel and Distributed Computing*, 53(1):26–62, 1998.

[2] M. Grammatikakis, D. Hsu, M. Kraetzel, and J. Sibeyn. Packet routing in fixed-connection networks: A survey. *Journal of Parallel and Distributed Computing*, 54(2):77–132, 1998.

[3] R. Grossi, A. Pietracaprina, and G. Pucci. Optimal deterministic protocols for mobile robots on a grid. *Information and Computation*, 173:132–142, 2002.

[4] T. Hagerup and C. Rüb. A guided tour of Chernoff bounds. *Information Processing Letters*, 33(6):305–308, 1990.

[5] K. Herley and G. Bilardi. Deterministic simulations of PRAMs on bounded-degree networks. *SIAM Journal on Computing*, 23(2):276–292, 1994.

[6] K. Herley, A. Pietracaprina, and G. Pucci. Implementing shared memory on mesh-connected computers and on the fat-tree. *Information and Computation*, 165(2):123–143, 2001.

[7] J. JáJá. *An Introduction to Parallel Algorithms*. Addison Wesley, Reading MA, 1992.

[8] R. Kesavan and D. Panda. Multiple multicast with minimized node contention on wormhole k-ary n-cube networks. *IEEE Trans. on Parallel and Distributed Systems*, 10(4):371–393, 1999.

[9] F. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays • Trees • Hypercubes*. Morgan Kaufmann, San Mateo, CA, 1992.

[10] F. Leighton, B. Maggs, A. Ranade, and S. Rao. Randomized routing and sorting on fixed-connection networks. *Journal of Algorithms*, 17(1):157–205, 1994.

[11] R. Libeskind-Hadas, D. Mazzoni, and R. Rajagopalan. Tree-based multicasting in wormhole-routed irregular topologies. In *Proc. of the 1st Merged IPPS/SPDP Symposium*, pages 244–249, March/April 1998.

[12] X. Lin and L. Ni. Deadlock-free multicast wormhole routing in multicomputer networks. In *Proc. of the 18th ACM Annual International Symposium on Computer Architecture*, pages 116–125, May 1991.

[13] X. Lin and L. Ni. Multicast communication in multicomputer networks. *IEEE Trans. on Parallel and Distributed Systems*, 4(10):1105–1117, 1993.

[14] A. Pietracaprina and G. Pucci. The complexity of deterministic PRAM simulation on Distributed Memory Machines. *Theory of Computing Systems*, 30(3):231–247, 1997.

[15] A. Pietracaprina and G. Pucci. Optimal many-to-one routing on the mesh with constant queues. *Information Processing Letters*, 96(1):24–29, 2005.

[16] A. Ranade. How to emulate shared memory. *Journal of Computer and System Sciences*, 42:307–326, 1991.

[17] D. Robinson, P. McKinley, and B. Cheng. Optimal multicast communication in wormhole-routed torus networks. *IEEE Trans. on Parallel and Distributed Systems*, 6(10):1029–1042, 1995.

[18] J.F. Sibeyn and M. Kaufmann. Deterministic 1-k routing on meshes, with application to hot-potato worm-hole routing. In *Proc. of the 11th Symp. on Theoretical Aspects of Computer Science*, LNCS 775, pages 237–248, February 1994.

[19] C. Wang, Y. Hou, and L. Hsu. Adaptive path-based multicast on wormhole-routed hypercubes. In *Proc. of the 8th Euro-Par*, LNCS 2400, pages 757–761, August 2002.