Randomized multi-packet routing on meshes

Michael Kaufmann, Jop F. Sibeyn

RUU-CS-91-48
December 1991

Utrecht University

Department of Computer Science

Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

Randomized multi-packet routing on meshes

Michael Kaufmann, Jop F. Sibeyn

Technical Report RUU-CS-91-48
December 1991

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands

ISSN: 0024-3275

Randomized multi-packet routing on meshes

Michael Kaufmann* Jop F. Sibeyn†

Abstract

We present algorithms for routing packets on a two-dimensional array of processors in the so-called k-k routing model. Each processor sends and receives exactly k packets. Using new techniques for the performance analysis we show that a simple randomized three phase algorithm performs optimally: For all $k \geq 8$, the k-k routing problem is solved with $k \cdot n/2 + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ routing steps and constant size queues with very high probability, which nearly matches the lower bound of $k \cdot n/2$ steps. For $k < 8$ we present refined algorithms which come close to optimal. In addition, we prove interesting results for cut-through routing: the same randomized algorithm routes packets consisting of k flits each with $k \cdot n/2 + n/k + 3/2 \cdot k + \mathcal{O}(k \cdot (n \cdot \log n)^{1/2})$ routing steps with very high probability. We can generalize the algorithm for routing on meshes of arbitrary dimension and nearly reach the trivial lower bounds for both classes of problems. For routing on meshes with wrap-around connections we present a four phase algorithm which can be generalized easily for routing on meshes with wrap-around connections of arbitrary dimension. The performance is close to optimal. E.g., the k-k routing problem takes $k \cdot n/4 + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ routing steps. Finally we analyse routing with locality.

1 Introduction

One of the main problems in the simulation of idealistic parallel computers by realistic ones is the problem of message routing through the sparse network of links which connect the different realistic processors among each other. We consider the case that the processors form an array and are connected by a two-dimensional mesh of communication links, which has size $n \times n$. The links (edges) are bidirectional, at most one packet of information can go along an edge in each direction in unit time. The model of communication we use is the MIMD-model. Each processor can communicate with all its neighbors in a single step, and the communication steps are synchronized. Each processor can send and receive at most one packet per edge and per step. It may store packets in a local queue.

A k-k routing problem is the problem of transporting k packets from each processor in the mesh to k not necessarily distinct destination processors. Each processor initially sends and finally receives k packets. In case $k = 1$, the problem is the permutation routing problem and has been attended most consideration in the past. The goal is to bound the number of steps as well as the maximal used size of the queues. For the cut-through routing however, we consider the k packets within the same processor as k-flits arising from the same big packet. The flits from the same packet behave like a worm. All flits follow the first one, known as the head, to the destination. A worm may never be cut off, i.e., at any given time, consecutive flits of a worm are at the same or adjacent processors. The goals remain the same.

Using sorting some algorithms solve the problem in $3 \cdot n + \mathcal{O}(\text{low order})$ steps with queue size 1. Krizanc, Rajasekaran and Tsantilas [2] gave a randomized algorithm which needs $2 \cdot n + \mathcal{O}(\log n)$ steps with constant

*Max-Planck-Institut für Informatik, 6600 Saarbrücken, Germany

†Department of Computer Science, University of Utrecht, P.O. Box 80.089, 3508 TB Utrecht, the Netherlands. Email: jopsi@ruuinf.cs.ruu.nl. The work of the author was financially supported by the Foundation for Computer Science (SION) of the Netherlands Organization for Scientific Research (NWO). This research was partially supported by the ESPRIT II Basic Research Actions Project of the EC under contract No 3075 (Project ALCOM).

queue size. An algorithm of the same flavor but deterministic [4] solves the problem in $2 \cdot n + \mathcal{O}(n/f(n))$ steps for arbitrary queue size $f(n), 1 \leq f(n) \leq n$. Finally, Leighton, Makedon and Tollis [8] presented an $2 \cdot n - 2$ algorithm with constant, but large queue size. Recently this algorithm has been improved by Rajasekaran and Overholt to one with queues of maximal size 58 [11]. Note that $2 \cdot n - 2$ is a natural lower bound for the number of steps.

The case $k \geq 2$ gained more attention recently. Here the trivial lower bound is $k \cdot n/2$, which arises when half of the packets have to change sides. Kunde and Tensi [5] achieved a bound of $5/4 \cdot k \cdot n + \mathcal{O}(k \cdot n/f(n))$ with queue size $\mathcal{O}(k \cdot f(n))$ using a deterministic algorithm. With randomization Rajasekaran and Raghavachari [12] could generalize [2] to get an algorithm with a $k \cdot n + o(k \cdot n)$ step bound and $\mathcal{O}(k)$ queue size. Recently Kunde [6] presented a deterministic algorithm which comes very close to the distance bound assuming that the k-k problem consists of k separated permutation problems which have to be known in advance. In that case and if k is divisible by 8 he achieved really nice bounds ($\lceil k/8 \rceil \cdot 2 \cdot n + \lceil k/4 \rceil n + O(k \cdot n^{2/3})$).

In this paper, we remove Kunde's assumptions and present randomized algorithms which solve the general k-k problem in $k \cdot n/2 + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ steps and $\mathcal{O}(k)$ queue size (means optimally) with very high probability, improving Kunde's bounds slightly. The idea is to randomize the packets within the column or row in which they reside. This disturbs all bad initial distributions to such an extent, that the remainder of the routing can be done fast. For $k \geq 8$, this gives the stated bound. Different algorithms which work successfully for smaller k are presented in [3]. For the case that the mesh has additional wrap-arounds we extend our techniques in a nontrivial way such that we get bounds of $k \cdot n/4 + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ which nearly matches the trivial lower bound of $k \cdot n/4$. Furthermore the same techniques provide very good bounds for multidimensional k-k routing.

For cut-through routing we considerably improve the known bounds for the number of steps, namely approximately by a factor of 2. Makedon and Simvonis [10] gave the first good bounds for cut-through routing, which is also discussed by Leighton [9]. Rajasekaran and Raghavachari [12] show that their analysis for the general k-k-routing problem also holds for the cut-through routing problem. Both use randomized algorithms. We make some very general considerations for one-dimensional routing assuming no specific conflict-resolution strategy but only that each processor should route one of the packets that are available. These considerations can be applied for any strategy, and especially for the rules of the cut-through routing. Using the algorithm for the general k-k-routing and the analysis for the "arbitrary" conflict-resolution strategy, we get an almost optimal bound of $k \cdot n/2 + n/k + 3/2 \cdot k + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ routing steps with very high probability for cut-through routing.

The remainder of this paper is organized as follows: After some definitions in Section 2, we introduce a number of methods that are generally used in mesh routing algorithms, and state the Chernoff bounds. Then, we will derive some important results on the routing in one-dimensional meshes. These results are used in the subsequent sections where we present our main results. In Section 6, a simple layered algorithm is discussed. It is appealing but inherently sub-optimal. This section is quite self-contained and it might be skipped by the reader who is interested only in the main results of the paper. In Section 7 we present our optimal algorithm for k-k and cut-through routing on meshes without wrap-around connections. The same is done for meshes with wrap-around connections in Section 8. In the final section Section 9 we shortly consider the problem of routing with locality.

2 Definitions and notation

The PUs of a d-dimensional $n \times \cdots \times n$ mesh are indexed by a d-tuple $(i_1, \ldots, i_d)$, $0 \leq i_j \leq n-1$. The PU with index $(i_1, \ldots, i_d)$ is denoted $P_{i_1, \ldots i_d}$.

With respect to the defined indexing, we can give two very useful permutations. These are the large shift LS: routing the packets residing in $P_{i_1, \ldots, i_d}$ to $P_{(i_1+n/2) \bmod n, \ldots, (i_d+n/2) \bmod n}$; and the point reflection PR:

routing the packets residing in $P_{i_1,\ldots,i_d}$ to $P_{n-i_1,\ldots,n-i_d}$. For our purposes the k fold versions of them, k-LS and k-PR are even more important. Under k-LS (k-PR) every PU wants to send k packets to the destination given by LS (PR). Often these permutations and variations of them give a first impression of the performance of a routing algorithm. We will often use the term k-permutation or just permutation for distributions of packets such that every PU sends and receives precisely k-packets. k-LS and k-PR are k-permutations. With this definition, the k-k routing problem is the problem of efficiently routing k-permutations. A k-permutation can be decomposed into k permutations (e.g., by solving an edge coloring problem in a regular bipartite graph of degree k). However, as the global information that would be necessary for such a decomposition is not available to the PUs, this fact cannot be used for an efficient routing algorithm.

For many problems it is hard to derive lower bounds. In the theory of routing algorithms, the situation is rather fortunate: There, we have at our disposal three standard methods for deriving lower bounds. They are

Distance bound The diameter of the mesh ($d \cdot (n-1)$, for a d-dimensional mesh without wrap-around connections; $d \cdot n/2$, for a d-dimensional mesh with wrap-around connections), is an obvious lower bound for routing algorithms. In the permutation routing problem on meshes without wrap-around connections this bound has been matched (Section 1). This bound imposes the heaviest restriction when the loading of the mesh is small.

Bisection bound Any connection between $P_{i_1,\ldots,n/2-1,\ldots,i_d}$ and $P_{i_1,\ldots,n/2,\ldots,i_d}$ can be used for the transmission of at most one packet during one routing step. Thus, if there are $k/2 \cdot n^d$ packets that have to go through these PUs in one-direction, then such a distribution of packets cannot be routed with less than $k \cdot n/2$ routing steps. This bound is most important in heavily loaded meshes without wrap-around connections. For k-k routing it has been matched under conditions (Section 1) and is almost matched by a randomized algorithm in Section 7.

Connection availability bound Any connection in the mesh can be used for the transmission of at most one packet during one routing step. There are $4 \cdot n^d$ connections in the mesh (in a mesh without wrap-around connections a bit less). Thus, a certain distribution of packets over the mesh with $D = \sum_{\text{packets}} \text{distance to go}$, cannot be routed with less than $D/(4 \cdot n^d)$ routing steps. This bound plays a role for the lower bound of algorithms on heavily loaded meshes with wrap-around connections. In Section 8 we show that it is almost matched by a randomized algorithm.

k-LS is a worst-case example for the bisection and the connection availability bound. k-PR is worst-case for the distance and bisection bound.

One of the most important algorithmic decisions is, how the PUs manage the packets they have: Which packets are sent on, and which packets are kept? It seems pointless to leave a connection idle, when there is a packet that has to go over it. In the routing algorithms of all papers we know of, including this paper, the PUs do what they can. The remaining question is, what the PUs do when there is more than one packet that has to go over a certain connection. That is, how are contentions for connections resolved? In this paper the situation is relatively easy because most algorithms are composed of separated phases of routing in one dimension only. So, there will be no packets that "go around the corner" adding to the packets already moving in a certain row or column of the mesh. For routing in a one-dimensional processor array (linear or circular), there are the following much used approaches:

farthest-destination-first strategy From among the packets that have to go over a certain connection, the packet is selected that has to go the farthest in that direction. This strategy is the most widely used. For short it is called "farthest-first strategy".

farthest-origin-first strategy From among the packets that have to go over a certain connection, the packet is selected whose origin lies the farthest away. Although this strategy is sometimes used, it can never

perform better than the farthest-destination-first strategy. It may have advantages in the analysis of an algorithm.

arbitrary strategy From among the packets that have to go over a certain connection, an adversary with total knowledge may select the most unfortunate packet. This is not a practical strategy, but it gives an upper bound on the routing time with any other strategy.

random strategy From among the packets that have to go over a certain connection, one packet is selected at random. This may be an interesting strategy in heavily loaded systems. Especially in a dynamic environment it guarantees that after some time any packet will reach its destination without attaching round numbers to the packets.

There are generalizations for routing on higher dimensional meshes, most notably the **farthest-total-first** strategy, giving priority to packets that have the largest total distance to go.

3 Routing algorithm techniques

Now we review some basic techniques developed for packet routing which will be used in the following. The first four techniques are used for 1-1 routing. The fifth technique is helpful for k-k routing, or routing in meshes of dimension larger than 2.

The "greedy" strategy. Each packet is routed in a first phase to its destination column and in the second phase to the destination row. Hence phase I consists only of horizontal movements and phase II only of vertical movements. Using Lemma 1 and the strategy that the packets with farthest destination should have highest priority, one can easily show, that the classical 1-1 routing problems can be solved in optimal time. Unfortunately, the packets from one row that want to go to the same column, accumulate between the two phases in one single queue, which may have size n. The algorithm performs not much better if the packets are allowed to move vertically as soon as they reached the destination column. Leighton [9] has shown that on the average this greedy strategy performs very well (queue size 5), since the extreme situation described above is very unlikely. Since the strategy is nice and easy to analyse, it serves in almost all packet routing algorithms as a basis.

Randomized vertical distribution of the packets within small horizontal strips. The idea to avoid the above bad situation is first to distribute the packets which want to go to the same column among several rows. This is usually done in a prephase. Since this new phase should not take too long, the packets are distributed within several horizontal strips of size ϵ, with ϵ a small fraction of n. Kunde [4] proposed to do the distribution by a vertical snake-like sorting according to the column destination; Krizanc, Rajasekaran and Tsantilas [2] distribute the packets in the same column randomly. Both approaches perform much better than the greedy strategy with respect to the queue size, but the number of steps is increased by $\mathcal{O}(\epsilon)$.

Superior/inferior packets. This difficulty is avoided by introducing the concept of superior and inferior packets. Only the packets near the corners of the mesh may travel far. Thus, no detour should be allowed for them. Therefore they are called superior packets and get a special treatment, namely they are routed in the randomization phase to the middle of the mesh where there is so much time slack that they can be routed there. This concept introduced by [13] and [8] use it to achieve an optimal running time of $2 \cdot n - 2$ and constant queue size. In the randomization step, the technique is not sufficient to achieve constant queue sizes. On the average the queue sizes are constant, but with high probability there is a queue with $\mathcal{O}(\log n)$ packets in it. A fourth technique solves this problem.

Smearing. At the end of the prephase and phase I the column is divided into blocks of size $\mathcal{O}(\log n)$; in each block of that size there are $\mathcal{O}(\log n)$ packets. Therefore the packets are redistributed within each block such that with an additional time amount of $\mathcal{O}(\log n)$ we achieve a final queue size of $\mathcal{O}(1)$. Note that using

this technique the algorithm becomes non-oblivious, which means that the routing of the packets does not only depends on their addresses [2].

Coalescing phases. Separating the phases of a routing algorithm has great advantages for its analysis. However, it is obvious that the performance will be at least as good, and in many cases better, when a packet that has finished its phase i proceeds with phase $i+1$ when the connection over which it has to go is available. Most common is to use a kind of lexicographical priority: A packet doing its phase i has absolute priority over a packet doing its phase j, when $i < j$. Among packets doing the same phase the normal priority rules are applied.

Coloring. The idea is to partition the problem into several subproblems which can be treated independently. Usually this is assured by the design of uniaxial algorithm, where at each single time step all packets are moved only on one axis, either horizontally or vertically. The idea is to split the set of packets into two halves, such that when the packets of the first part are moved horizontally, the other packets are moved vertically and vice versa. This concept is introduced by Kunde [7] for multidimensional routing and is also used by Rajasekaran/Raghavachari [12] for k-k routing in two-dimensional meshes. The latter partition the packets into two halves by randomly coloring the packets with two colors and show that the packets of different colors do not interfere although their algorithm is not uniaxial.

4 Chernoff bounds

In the analysis of the randomization technique we use the following fact, known as Chernoff bounds. These bounds provide close approximations to the probabilities in the tail ends of a binomial distribution.

Let $X = B(n,p)$ be the number of heads in n independent flips of a biased coin, the probability of a head in a single flip being p. Such an X has the binomial distribution. Using known results it is shown in [1] that

$$prob\{X \geq n \cdot p + h\} \leq e^{-h^2/(3 \cdot n \cdot p)}, \text{ for all } 0 < h < n \cdot p, \quad (1)$$

$$prob\{X \leq n \cdot p - h\} \leq e^{-h^2/(2 \cdot n \cdot p)}, \text{ for all } 0 < h < n \cdot p. \quad (2)$$

(1) and (2) give a bound on the probability of a large deviation of the number of heads from the expected number. The results of this paper will all hold "with very high probability". We formalize this notion:

Definition 1 *An event a happens with very high probability if $prob(a) \geq 1 - n^{-\alpha}$, for some constant $\alpha > 0$.*

Usually we will need that a polynomial number of independent binomial random variables, $X_1, \ldots, X_M$, $X_i = B(k \cdot n, p)$, $M \leq n^m$, for some constant m, are all bounded at the same time. (1) and (2) are so strong that this is no problem at all:

Lemma 1 *There is an $h = \mathcal{O}((p \cdot k \cdot n \cdot \log n)^{1/2})$, such that $p \cdot k \cdot n - h \leq X_i \leq p \cdot k \cdot n + h$, for all $1 \leq i \leq M$ at the same time, with very high probability.*

Proof: Let $h = (3 \cdot m \cdot p \cdot k \cdot n \cdot \log n)^{1/2}$. We show that the probability that one of the X_i exceeds the bound is very small: $P(\bigvee_{i=1}^{M}(X_i \leq p \cdot k \cdot n - h \text{ or } X_i \geq p \cdot k \cdot n + h)) \leq M \cdot (P(X_i \leq p \cdot k \cdot n - h) + P(X_i \geq p \cdot k \cdot n + h)) \leq e^{0.7 \cdot m \cdot \log n} \cdot 2 \cdot e^{-m \log n}$. □

We will use the Chernoff bounds in this form.

5 One-dimensional routing

With aid of Lemma 1 we will derive in this section some important results concerning the routing in a one-dimensional processor array. We consider the routing on a linear and on a circular processor array. Later on,

these results will be used for the analysis of the algorithms for routing on meshes without and with wrap-around connections respectively. On both types of one-dimensional arrays, we consider routing with the farthest-first and the arbitrary conflict resolution strategy. The results with the farthest-first strategy serve for the analysis of the k-k routing algorithms, those with the arbitrary conflict resolution strategy serve for the analysis of the cut-through routing algorithms. The proofs of some of the lemmas of this section are long and detailed. This is justified because they underly the main results of the whole paper.

5.1 Routing on a linear processor array

In this section we consider the routing on a linear processor array. First we will prove a lemma that precisely states the number of routing steps required when the farthest first strategy is used. From this we can derive a sharp result for the routing time of a randomization on a linear processor array. Hereafter, we do the same for the case that the arbitrary strategy is used.

5.1.1 Farthest-first strategy on a linear array

In order to prove that using the farthest-first conflict resolution strategy, randomizing k layers of packets in a linear processor array can be carried out in about $k \cdot n/4$ routing steps, we need a strong result as given in Lemma 2.

Define for a given distribution of packets over the PUs

$$h_r(i,j) = \#\{\text{ packets to be routed from PUs with index} \leq i \text{ to PUs with index} \geq j\} - 1,$$

and let T_r be the number of routing steps a distribution of packets requires for the routing to the right.

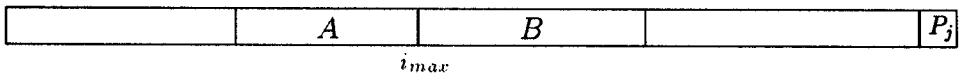
Lemma 2 $T_r = \max_{i<j}\{j - i + h_r(i,j)\}$.

Proof: Consider i_{max}, j_{max} such that $j_{max} - i_{max} + h_r(i_{max}, j_{max}) = \max_{i,j}\{j - i + h_r(i,j)\}$. A packet starting in a PU with index i_{max} or smaller has to go at least $j_{max} - i_{max}$ routing steps before it reaches $P_{j_{max}}$. At most one packet reaches $P_{j_{max}}$ from the left every routing step. This shows that $T_r \geq j_{max} - i_{max} + h_r(i_{max}, j_{max}) = \max_{i,j}\{j - i + h_r(i,j)\}$.

Now we prove the other side of the inequality. Consider the subproblem arising from the left part of the array of size j. Take all packets starting there into consideration. Assume that they end in P_j if they move further to the right in the original problem. We will show that

Claim: The last packet reaches P_j after $\max_{i,1\leq i\leq j-1}\{j - i + h_r(i,j)\}$ steps.

A packet with destination P_j can only be delayed by packets with the same destination. Therefore we need only to consider packets with destination P_j. We call them j-packets. Let i_{max} be such that $\{j - i_{max} + h_r(i_{max}, j)\} = \max_{i,1\leq i\leq j-1}\{j - i + h_r(i,j)\}$. Define $\mathcal{P}_l = \{P_1, \ldots, P_{i_{max}}\}$, $\mathcal{P}_r = \{P_{i_{max}+1}, \ldots, P_{j-1}\}$. Consider the following subdivision of the mesh:



A and B are sections of $\mathcal{P}_l$ and $\mathcal{P}_r$ respectively. The definition of i_{max} and j_{max} imply that, for all such sections A, B,

$$\#A_j \geq \#A, \tag{3}$$

$$\#B_j \leq \#B. \tag{4}$$

Here is B_j is the set of j-packets in B. (3) holds because otherwise i_{max} would have been chosen smaller. (4) holds because otherwise i_{max} would have been chosen larger; By (4) the j-packets that start in $\mathcal{P}_r$ can

move ahead of the j-packets coming from $\mathcal{P}_l$. This means that the j-packets from $\mathcal{P}_r$ reached P_j by step $j - i_{max} - 1$, and that these packets will not delay the j-packets from $\mathcal{P}_l$. By (3) the j-packets from $\mathcal{P}_l$ flow into P_j continuously from step $j - i_{max}$ on. Thus, the last of them reaches P_j by step $j - i_{max} + h_r(i_{max}, j)$, as stated. That the packets from $\mathcal{P}_l$ really flow into P_j without interruption, may require some explanation: These packets will not reach $\mathcal{P}_r$ faster if some of them are moved to the left. Now, we can wipe the j-packets to the left in a special way: For all i from i_{max} down to the smallest index of a PU that sends packets to P_j, all j-packets in P_i except for one, are moved to P_{i-1}. (3) assures that there is always a packet to leave behind. After the wiping all these packets lie as a long uninterrupted chain in $\mathcal{P}_l$, with possibly a pile at the left-most PU that sends a packet to P_j. The movement of the chain is not retarded and thus it moves every step one position to the right.

The claim gives us the latest time for packets to reach their destination PU. Since it holds for arbitrary j, maximizing over j gives the total time bound. This is exactly the bound given in the lemma. □

Corollary 1 *When the packets are distributed over the linear processor array such that any subset of m PUs is sending and receiving at most $k \cdot m + \alpha$ packets, then using the farthest-first strategy, all packets will reach their destination after at most* $\max\{n-1, k \cdot n/2\} + \alpha$ *routing steps.*

Combining Lemma 1, Lemma 2 and its analogue for routing to the left, we get the important result on routing randomizations in a linear processor array:

Lemma 3 *Routing with the farthest-first conflict resolution strategy k packets from every PU of a linear processor array with n PUs to a random destination requires at most* $\max\{n, k \cdot n/4\} + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ *routing steps, with very high probability.*

Proof: Routing a randomization with every PU sending 4 packets takes more routing steps than when every PU would send only 1, 2 or 3 packets. So, we may assume $k \geq 4$. We consider $h_r(i,j)$ as defined before Lemma 2. By Lemma 1 $h_r(i,j) \leq k \cdot i \cdot (n-j)/n + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$. Thus, Lemma 2 gives for the routing time an upper bound of $\max_{i \leq j}\{(j-i) + h_r(i,j)\} \leq \max_{i \leq j}\{j - i + k \cdot i \cdot (n-j)/n\} + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$. Let $f_k(i,j) = j - i + k \cdot i \cdot (n-j)/n$. We have to find the maximum of f_k over all $0 \leq i \leq j \leq n$. Partial differentiation gives an extreme value $f_k(n/k, n - n/k) = n - n/k$. On the boundary there are two more extremal values: $f(0,n) = n$ and $f(n/2, n/2) = k \cdot n/4$. For $k \geq 4$ this last value is at least as big as the others. □

By this result we conclude that randomizations can be carried out very efficiently for all $k \geq 4$. For these k, the bisection is matched.

5.1.2 Arbitrary strategy on a linear array

Now, we will prove analogues of Lemma 2 and Lemma 3 for the case that the arbitrary conflict resolution strategy is used. With a minor modification, the so-derived results hold for cut-through routing as well.

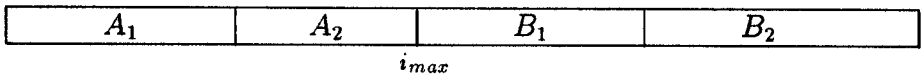
Analogously to $h_r(i,j)$ we define

$$h_r(i) = \#\{\text{ packets to be routed from PUs with index } \leq i \text{ to PUs with index } > i\} - 1.$$

Using this definition we can state the lemma on which the results for the cut-through routing will be based:

Lemma 4 *Routing a distribution of packets over a one-dimensional processor array takes at most* $\max_i\{h_r(i) + n - 1 - i\}$ *routing steps to the right.*

Proof: The proof resembles the proof of Lemma 2 but there are important differences as well. Let i_{max} be such that $h_r(i_{max}) - i_{max} = \max_i\{h_r(i) - i\}$. Define $\mathcal{P}_l$, $\mathcal{P}_r$ as in the proof of Lemma 2. Consider the following subdivision of the mesh:



A_1 and A_2 subdivide $\mathcal{P}_l$, B_1 and B_2 subdivide $\mathcal{P}_r$. The definition of i_{max} implies that for all such subdivisions into A_1, A_2, B_1, B_2,

$$\#\{A_2 \rightarrow \mathcal{P}_r\} \geq \#\{A_1 \rightarrow A_2\} + \#A_2, \tag{5}$$

$$\#\{\mathcal{P}_l \rightarrow B_1\} \geq \#\{B_1 \rightarrow B_2\} - \#B_1. \tag{6}$$

(5) holds because otherwise i_{max} would have been chosen smaller; (6) holds because otherwise i_{max} would have been chosen larger. In a special way, we will eliminate packets moving within $\mathcal{P}_l$ and $\mathcal{P}_r$ until for all such subdivisions

$$\#\{A_1 \rightarrow A_2\} = 0, \tag{7}$$

$$\#\{B_1 \rightarrow B_2\} \leq \#B_1. \tag{8}$$

This modifications will leave $\max_i\{h_r(i) - i\}$ and $h_r(i_{max})$ invariant, and do not decrease the required number of routing steps. Thus, when we show that the final distribution of packets can be routed with $h_r(i_{max})$ routing steps, this holds for the original distribution as well.

By (5) there is for any packet p moving to the right within $\mathcal{P}_l$ a packet q going from $\mathcal{P}_l$ to $\mathcal{P}_r$ starting to the right of the destination of p (or precisely at the destination of p). Without decreasing the number of routing steps, the pair of packets (p, q) can be replaced by a packet r, starting where p starts and ending where q ends. This has to be done with some care in order to leave $\max_i\{h_r(i) - i\}$ invariant (and preserve (5) as a consequence): There should be no packet p' with destination between the destination of p and the origin of q. In this way all packets moving within $\mathcal{P}_l$ can be eliminated one by one, realizing (7) for all choices of A_1 and A_2. Analogously, packets moving within $\mathcal{P}_r$ can be eliminated until (8) holds. For the now obtained distribution, it is easy to show that the lemma holds: (8) implies that the packets moving inside $\mathcal{P}_r$ move ahead of the packets coming from $\mathcal{P}_l$ and reach their destination fast. Thus, they will not delay the packets coming from $\mathcal{P}_l$. By (7) there are no packets moving inside $\mathcal{P}_l$ that can delay the packets moving from $\mathcal{P}_l$ to $\mathcal{P}_r$. Then, (5) implies that the packets from $\mathcal{P}_l$ are moving from step 1 on, and without delay, into $\mathcal{P}_r$. This means that the last of these packets enters $\mathcal{P}_r$ after $h_r(i_{max}) + 1$ steps. At worst this packet has to go to P_{n-1}. In that case it reaches its destination after another $n - 1 - i_{max} - 1$ steps.

When $h_r(i_{max}) + n - i_{max} < n - 1$, the packets moving within $\mathcal{P}_l$ should not be taken in arbitrary order. At any time the packet that starts the farthest to the right should be taken first. In this case, we should not establish (7), but stop the pairing when the remaining packets moving within $\mathcal{P}_l$ cannot interfere with the packets going to $\mathcal{P}_r$ anymore. □

The term $n - i$ in Lemma 4 could be replaced by $\max\{j|$ there is a packet that passes P_i going to $P_j\} - i$. After this refinement, the proof should be carried out with more care (the pairing within $\mathcal{P}_r$ should start with the packets going the least far to the right, and only for so far it is necessary).

Corollary 2 *When the packets are distributed over the linear processor array such that any subset of m PUs is sending and receiving at most $k \cdot m + \alpha$ packets, then using the arbitrary strategy, all packets will reach their destination after at most* $\max\{n - 1, k \cdot n/2 + n/2\} + \alpha$ *routing steps.*

Another consequence of Lemma 4 is the following analogue of Lemma 3 for routing randomizations when the arbitrary priority rule is used:

Lemma 5 *Routing with the arbitrary conflict resolution strategy k packets from every PU of a linear processor array with n PUs to a random destination requires at most* $\max\{n, k\cdot n/4+n/(4\cdot k)+n/2\}+\mathcal{O}((k\cdot n\cdot\log n)^{1/2})$ *routing steps, with very high probability.*

Proof: We may assume $k \geq 4$. By Lemma 1 $h_r(i) \leq k\cdot i\cdot(n-i)/n+\mathcal{O}((k\cdot n\cdot\log n)^{1/2})$. Thus, Lemma 4 gives for the routing time an upper bound of $\max_i\{n-i+h_r(i)\} \leq \max_i\{n-i+k\cdot i\cdot(n-i)/n+\mathcal{O}((k\cdot n\cdot\log n)^{1/2})\}$. Differentiation of $f_k : i \rightarrow n-i+k\cdot i\cdot(n-i)/n$, gives that f_k assumes its maximum value for $i_{max} = n/2\cdot(1-1/k)$. Substitution gives $f_k(i_{max}) = n\cdot k/4 + n/(4\cdot k) + n/2$. □

In comparison with the result of Lemma 3 we see an additional term of $n/(4\cdot k)+n/2$ in the routing time of a randomization. This is inevitable: if $k > 2$, then with high probability there is a packet that has to go a distance $n/2$, and whose first step can be postponed until all other packets have passed. On the other hand, this not-so-sharp result is already established for $k \geq 2$.

Cut-through routing imposes some specific restrictions, but when possible the farthest-first strategy could be applied. Thus, cut-through routing goes at least as fast as routing with the arbitrary strategy. So it seems that we could immediately apply the results of Corollary 2 and Lemma 5. This is true for Corollary 2, but with Lemma 5 we have to be careful: Cut-through routing implies also that all packets from a PU have the same destination. This means that when using Lemma 1 for estimating the maximal number of packets going to a subset of m PUs, we should use it with $k = 1$, and then multiply the result by k. This gives

Lemma 6 *Routing in cut-through mode k packets from every PU of a linear processor array with n PUs to a random destination requires at most* $\max\{n, k\cdot n/4+n/(4\cdot k)+n/2\}+\mathcal{O}(k\cdot(n\cdot\log n)^{1/2})$ *routing steps, with very high probability.*

5.2 Routing on a circular processor array

Now, we consider the routing on a one-dimensional processor array with wrap-around connections. Such a processor array is circular. If there are n PUs, no packet has distance larger than $n/2$ to its destination. We will derive in this section results that are very similar to those of Section 5.1. In the proofs of Section 5.1 it was convenient that the PU had fixed ends. On a circular processor array, the packets can go around. Especially when the arbitrary strategy is used this might give problems: A packet going through a PU P_i might be delayed by a packet that does not go through P_i, that in its turn was delayed by packets that passed through P_i earlier. Thus, it could be difficult to express the routing time as in Lemma 2 and Lemma 4.

5.2.1 Farthest-first strategy on a circular array

When the farthest-first conflict resolution strategy is used, the analysis of a result analogous to Lemma 2 is rather easy. We will use it later for the analysis of a k-k routing algorithm.

For a given distribution of packets over the PUs

$$h_r(i,j) = \#\{\text{ packets passing from left to right through both } i \text{ and } j\} - 1.$$

This is a slight modification of the earlier definition of $h(i,j)$ in Section 5.1.1. This definition leaves open the possibility that $i > j$ and that the packets go around before they reach P_j. Let T_r be the number of routing steps a distribution of packets requires for the routing to the right.

Lemma 7 $T_r = \max_{i,j}\{h_r(i,j) - (j-i) \bmod n\}$.

Proof: Let i_{max}, j_{max} be such that $h_r(i_{max}, j_{max}) + (j_{max} - i_{max}) \bmod n = \max_{i,j}\{h_r(i,j) + (j-i) \bmod n\}$. No packet starting in a PU with index i_{max} or smaller will reach $P_{j_{max}}$ before step $(j_{max} - i_{max}) \bmod n$. At most one packet reaches a PU from the left in any step. Thus T_r is at least as big as stated.

Now we check the other side of the inequality. Consider the packets moving to or through some PU P_j from left to right. These packets are called j-packets. We will show that

Claim: The last j-packet reaches P_j after $\max_i\{h_r(i,j) + (j-i) \bmod n\}$ steps.

A j-packet can only be delayed by other j-packets. Thus, for the proof of the claim we can neglect all other packets. Let i_{max} be such that $h_r(i_{max}, j) + (j - i_{max}) \bmod n = \max_i\{h_r(i,j) + (j-i) \bmod n\}$. Define $\mathcal{P}_l = \{P_{j+1}, \ldots, P_{i_{max}}\}$, $\mathcal{P}_r = \{P_{i_{max}+1}, \ldots, P_{j-1}\}$. One of these sets wraps around (unless $j = 1$ or n). For sections A and B of $\mathcal{P}_l$ and $\mathcal{P}_r$ as in the proof of Lemma 2 we have (3) and (4). Then, we can argue as we did in the proof of Lemma 2, that the j-packets from $\mathcal{P}_r$ move ahead of the j-packets from $\mathcal{P}_l$ to P_j, and that the j-packets from $\mathcal{P}_l$ move into P_j continuously from step $j - i_{max}$ on. □

Combining Lemma 1, Lemma 7 and its analogue for routing to the left, we get a result for randomized routing on a ring of PUs:

Lemma 8 *Routing with the farthest-first conflict resolution strategy k packets from every PU of a circular processor array with n PUs to a random destination requires at most* $\max\{n/2, k \cdot n/8\} + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ *routing steps, with very high probability.*

Proof: Increasing the number of packets to route will not decrease the routing time. Thus, we may assume that $k \geq 4$. We are going to use Lemma 7, so we have to show that $h_r(i,j)$ is small enough for arbitrary i and j. It is easy to check that for given i and j the expected number of packets going from left to right through both of them is $k \cdot \sum_{l=1}^{n/2-(j-i) \bmod n} l/n \simeq k \cdot (n/2 - (j-i) \bmod n)^2/(2 \cdot n)$. The maximal value is assumed when $j = i + 1$. For $k \geq 4$ this maximal value is smaller than $k \cdot n/8$. □

The result is most interesting for $k \geq 4$. For these values of k the connection-availability bound tells us that the result cannot be improved: On the average a packet will have to move a distance $n/4$. Thus, in total the $k \cdot n$ packets have to go about $D = k \cdot n^2/4$ steps. In one routing step $2 \cdot n$ steps can be made. Thus, even when *all* connections can be used continuously, routing randomizations will take on the average $k \cdot n/8$ routing steps. It is amazing that this can be achieved also as a worst-case bound (except for a small additional term). When we compare with Lemma 3 we find that adding the wrap-around connection has halved the routing time of a k-permutation.

5.2.2 Arbitrary strategy on a circular array

For the analysis of the cut-through routing algorithms of Section 8.2, we need strong results without assuming the farthest-first strategy. We will prove analogues of Lemma 7 and Lemma 8 for the case that upon a collision during a routing in a ring any packet may be selected for proceeding first. The result for the cut-through routing then follows immediately.

Analogously to $h_r(i,j)$ we define

$$h_r(i) = \#\{\text{packets to be routed out of } P_i \text{ to the left}\} - 1.$$

T_r is defined as before. Define i_{max} to be an index of a PU such that $h_r(i_{max}) = \max_i\{h_r(i)\}$. The delay can be expressed in terms of $h_r(i_{max})$:

Lemma 9 *After t steps, a packet is delayed in the PU where it resides at most $h_r(i_{max}) - t$ times.*

Proof: Consider a packet p in a PU P. There are two possibilities: (1) Until now P has been passing packets to the right continuously; (2) P has been idle during some earlier step. In the first case there are still at most $h_r(i_{max}) - t + 1$ packets that P has to pass to the right. In any of the following steps, P can pass p to the right

or another packet. After at most $h_r(i_{max}) - t + 1$ times passing other packets, it must pass p. In the second case, there where no packets in P that had to go to the right during some earlier step. Hereafter, packets will not be delayed in P anymore: They are sent on by P, the step after they are received. □

Corollary 3 *A packet is delayed at most $h_r(i_{max})$ times.*

Summing the maximal path length and the maximal delay, we find the the analogue of Lemma 7:

Lemma 10 $T_r \leq h_r(i_{max}) + n/2$.

Now, we can prove the analogue of Lemma 8 for cut-through routing:

Lemma 11 *Routing with the arbitrary conflict resolution strategy k packets from every PU of a circular processor array with n PUs to a random destination requires at most $k \cdot n/8 + n/2 + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$, routing steps, with very high probability.*

Proof: By Lemma 10 we only have to check that $\max_i\{h_r(i)\}$ is bounded. The expected value of $h_r(i)$ is $k \cdot \sum_{j=1}^{n/2} j/n \simeq k \cdot n/8$, for all i. Thus, by Lemma 1 $h_r(i) \leq k \cdot n/8 + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$, for all i at the same time, with very high probability. □

In comparison with the result of Lemma 8 we see an additional term of $n/2$ in the routing time of a randomization. As in the result of Lemma 5 this is inevitable. A nice point is that now there is no minimal value of k from where on the result becomes good. Later on this will mean that the results for routing in the cut-through model on the two-dimensional mesh with wrap-around connections will hold for any value of k.

Lemma 11 holds for the case that $k \cdot n$ packets are routed on a circular processor to a random destination independently of each other. Taking into account that a cut-through routing implies that all packets from a PU have the same destination (c.f. Lemma 6), we find

Lemma 12 *Routing in the cut-through mode one packet consisting of k flits each from every PU of a one-dimensional processor array with n PUs to a random destination requires at most $k \cdot n/8 + n/2 + \mathcal{O}(k \cdot (n \cdot \log n)^{1/2})$, routing steps, with very high probability.*

6 Using levels

In this section we present a rather simple algorithm that routes k-permutations (k even) with $(k+1) \cdot n/2 + k/8 \cdot \epsilon + k \cdot \mathcal{O}((n \cdot \log n)^{1/2})$ routing steps, with ϵ a small fraction of n. The packets are distributed over two colors and $k/2$ levels. The packets of both colors are routed by uniaxial algorithms which route one level of packets after the other. This algorithm is a generalization of the simplest algorithm of [2] and basically uses the techniques of greedy routing and randomized distribution of packets within small strips.

In the first stage of the algorithm the packets are given a level number and a color:

for every packet p **do**

1. **assign** randomly a number from $\{1, 2, \ldots, k/2\}$ to p;
2. **assign** randomly a color green or blue to p.

The packets with number l are called l-packets. They are in level l. The packets are green or blue. Eventually, a random selection of exactly $k/2$ packets could be colored green and blue. Then, each number from $1, \ldots, k/2$ could be attributed precisely once to packets of both colors. This has no special advantages. It is not good to assign level numbers and colors deterministically unless we know the k permutations that are hidden in a k-permutation. In the second stage of the algorithm, the packets are treated differently depending on their color. The routing of the green packets is given by (notation of algorithms in a pseudo-Algol fashion)

Proc LevelRoute(green);

for $l := 1$ **to** $k/2 + 1$ **do**

a. **randomize** each green l-packet along the column in a strip of height ϵ;

b. **route** each green l-packet along the row to the column of its destination,
route each green $l - 1$-packet along the column to the row of its destination.

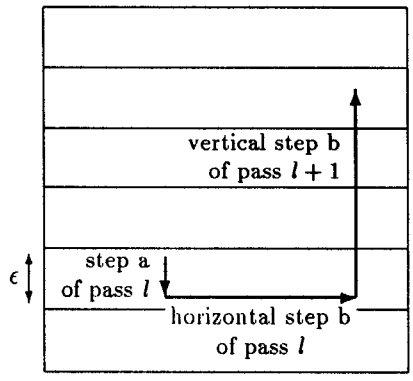


Figure 1: The routing of a green l-packet.

The blue packets are routed in parallel with the green packets, 90° out of phase with them. The routing of a green i-packet by LevelRoute is illustrated in figure 1. The choice of $\epsilon > 0$ is determined by a tradeoff between routing time and queue sizes.

With the results on one-dimensional routing we derived in Section 5 and Lemma 1, it is not difficult to demonstrate that each phase of LevelRoute can be performed sufficiently fast: Corollary 1 gives

Lemma 13 *Step a of every pass of the loop of LevelRoute takes at most* $\epsilon + \mathcal{O}((\epsilon \cdot \log n)^{1/2})$, *with very high probability.*

Slightly more difficult is

Lemma 14 *Step b of every pass of the loop of LevelRoute takes at most* $n + \mathcal{O}((n \cdot \log n)^{1/2})$, *with very high probability.*

Proof: Without loss of generality, we may concentrate on the routing along a certain row i of green l-packets in parallel with blue $l - 1$-packets, during steb b of pass l. The green packets are in row i as a result of the randomization in step a. Consider m positions of row i. There are $\epsilon \cdot m$ green l-packets that may be sent here, each with probability $1/\epsilon$. Lemma 1 gives that at most $m + \mathcal{O}((m \cdot \log n)^{1/2})$ of them are sent to these m positions. The blue packets are in row i because they have their destination here. They may be distributed very unevenly over the row, but at most $m + \mathcal{O}((n \cdot \log n)^{1/2})$ of them have destination in any subset of m PUs. The worst case is that the blue packets all start in the left most ϵ PUs and have to be spread out, and that the green packets all have to go to the right most ϵ PUs. Lemma 2 tells us that such a distribution can be routed with $n + \mathcal{O}((n \cdot \log n)^{1/2})$ routing steps. □

Lemma 13 and Lemma 14 together give

Corollary 4 *LevelRoute routes k-permutations with* $(k/2 + 1) \cdot (n + \epsilon + \mathcal{O}((n \cdot \log n)^{1/2}))$ *routing steps, for every even k, with very high probability.*

We can easily improve on this result for large k: Looking at Lemma 3 it is clear that 4 levels of packets can be randomized within ϵ steps. So, we should not do the randomization for every level separately, but perform

them all together at the start. Furthermore, during step b of the first pass of the loop of LevelRoute, no connection is used more than $n/2$ times. If k is large enough, this slack can be used for randomizing the packets of the highest $4 \cdot n/\epsilon$ levels. Now we have

Theorem 1 *The improved version of LevelRoute routes k-permutations with* $(k+1)/2 \cdot n + k/8 \cdot \epsilon + k \cdot \mathcal{O}((n \cdot \log n)^{1/2})$, *for even* k, $k \geq 4 \cdot n/\epsilon$, *with very high probability.*

6.1 Further improvements

In the result of Theorem 1 everything except the term $k \cdot n/2$ might be saved by better algorithms and analysis. In this section we discuss some obvious ideas for improvements. These ideas are:

- distinguishing superior and inferior regions in the mesh,
- performing the randomization of the i-packets, for all $i > 1$, during the routing of packets that are routed earlier.

Analogously to [2] we could hope to gain back the ϵ and $\mathcal{O}((n \cdot \log n)^{1/2})$ terms with the first technique. The second technique could be useful to save most of the randomization time. By cut arguments we will show that most of this is impossible.

6.1.1 Superior and inferior regions

In [2] and later papers [8] distinguishing superior and inferior packets turned out to be very useful for the reduction of the number of routing steps of the permutation routing problem. These results suggest that we might be able to perform every pass of the loop in LevelRoute with only $n + \mathcal{O}(\log n)$ routing steps. As there are $k/2 + 1$ passes, this would give a total routing time $k \cdot n/2 + n + k \cdot \mathcal{O}(\log n)$.

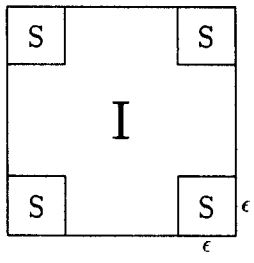


Figure 2: Location of superior (S) and inferior (I) regions.

Stage 1 of the algorithm is the same as in Section 6. Stage 2 is modified in order to safe the routing time that is lost during the randomization and to reduce the time lost due to delays. Superior packets are those that start in one of the four corner squares of size $\epsilon \times \epsilon$. The other packets are inferior. Superior and inferior regions are illustrated in figure 2. Inferior packets are routed by LevelRoute. Superior packets are randomized towards the center. This implies that the packets that have to go from corner to corner are moving towards their destination. These ideas are made precise by replacing step a and b of LevelRoute by

a′. route each inferior green l-packet along the column to a random row in its current ϵ-strip,
route each superior green l-packet along the column to a random row in the range $\epsilon + 1$, $\ldots, n/2$ (for the upper two superior regions the range $n - \epsilon, \ldots, n/2 + 1$);

b′. route each green l-packet along the row to the column of its destination,
route each green $l - 1$-packets along the column to the row of its destination.

In the remainder of this section we will show that steps a′ and step b′ of LevelRoute can be carried out with $n + \Theta(\epsilon^2)/n$ routing steps, and that this bound is sharp. It is routing the two colors at the same time that prevents us from achieving better. First we will show that with superior regions of size $\eta \times \eta$, there is a

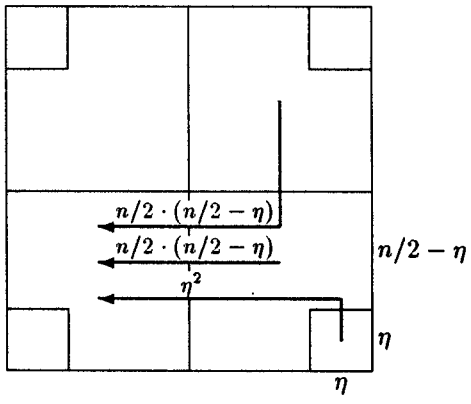


Figure 3: Routing k-PR takes at least $n + \eta^2/(n/2 - \eta)$ for every pass.

lower bound on the routing time of $n + 2 \cdot \eta^2/n$: Consider the routing of k-PR. During every pass of the loop of the modified LevelRoute (except for the first and last pass) a part of the central vertical cut of length $n/2 - \eta$ must throughput $n \cdot (n/2 - \eta) + \eta^2$ packets (figure 3). This will take at least $n + \eta^2/(n/2 - \eta) > n + 2 \cdot \eta^2/n$. η must be taken larger than ϵ. Thus, routing k-PR in this suggested way will take $k \cdot n/2 + n + k \cdot \epsilon^2/n$ routing steps.

Now, we will show that we can come close to this bound: With superior regions of size $\epsilon \times \epsilon$, we can achieve a routing time $k \cdot n/2 + n + k \cdot 2 \cdot \epsilon^2/n$. This follows directly from

Lemma 15 *If $\epsilon \leq n/6$, then step a′ and step b′ can be carried out with $n + 4 \cdot \epsilon^2/n$ routing steps.*

Proof: We analyse the routing to the left during some pass i, $1 < i \leq k/2$, while executing step a′ and step b′. There are ϵ superior packets that can be randomized to one of the $n/2 - \epsilon$ PUs higher up in the same column. On the average each of these PUs gets $\epsilon/(n/2 - \epsilon) \leq 3/n$ packets. Assume for a while that all the packets reach the position to which they randomize with 0 routing steps. The worst possible distribution of the packets moving to the left in a row between $\epsilon + 1$ and $n - \epsilon$ is

$n, \ldots, n - \epsilon + 1$	$n - \epsilon, \ldots, n/2 + 1$	$n/2, \ldots, \epsilon + 1$	$\epsilon, \ldots, 1$	
0	0	1	$\leftarrow 1$	blue packets
$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$	inferior green packets
0	0	0	$\leftarrow 3 \cdot \epsilon/n$	superior green packets

where we indicate the number of packets that can go from every PU to the left. By "$\leftarrow$" we mean that the packets may go to the left-most ϵ strip. The randomization may add $\mathcal{O}((n \cdot \log n)^{1/2})$ packets to all three types of packets. Using Lemma 2 we find that routing this distribution of packets takes at most $T_l = n + 3 \cdot \epsilon^2/n + \mathcal{O}((n \cdot \log n)^{1/2})$ steps. This implies that packets that have to move a distance d_l to the left are delayed at most $n - d_l + 3 \cdot \epsilon^2/n + \mathcal{O}((n \cdot \log n)^{1/2})$ times. In practice not all packets start their routing to the left at the same time: the superior green packets come too late. This however, will not delay one of these packets, or any of the other packets more than before. So, it is still true that no packet will spend more than T_l steps in its routing to the left. □

What is the importance of the reduction of a term $\Theta(k \cdot \epsilon)$ to a term $\Theta(k \cdot \epsilon^2/n)$ in the routing time? We could

already have obtained this lower routing time by randomizing in strips of size ϵ', with $\epsilon' = \epsilon^2/n$. However, this would give longer queues (for given ϵ the maximal queue length is $\mathcal{O}(n/\epsilon)$). Thus, the gain is a reduction of the queue lengths, without increasing the number of routing steps.

6.1.2 Saving the randomization time

We consider another approach to reduce the time that is lost by the randomization. For the randomization

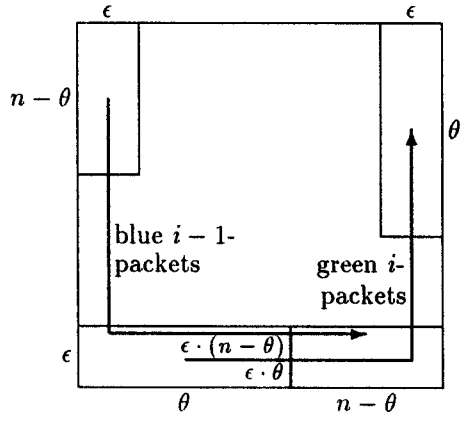


Figure 4: Routing k-PR leaves no room for randomization.

of the 1-packets we can use the superior/inferior idea; for $l > 1$, the l-packets might be randomized during the routing of packets of lower levels. As soon as the packets that still must be randomized find an empty slot in the packet stream they could drop in and flow along with the other packets to the position they had chosen. We will give an example that demonstrates that on some connections there are no empty places in the packet stream and that thus this approach cannot work: Consider the routing of the point reflection k-PR by LevelRoute. During pass i, $1 < i \leq k/2$, of the loop certain cuts of length ϵ have to put through $\epsilon \cdot n$ packets. One such a cut is depicted in figure 4. This shows that during the routing of step b of the loop of LevelRoute, there is no room at all to perform step a of the next pass of the loop.

Concluding, we can say that it seems difficult to modify a routing algorithm that distinguishes $k/2$ levels of packets to run much faster than we have done in Section 6.

7 Routing on meshes without wrap-around connections

The idea of introducing levels of packets, as was done in Section 6, gave good but sub-optimal results. This is not surprising after all: The full profit of having many packets to route can only be made if they are routed all together. That is what we will do in this section. In Section 7.1 we present the main algorithm for routing on meshes without wrap-around connections. We will show that it routes k-permutations with $k \cdot n/2 + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ routing steps with very high probability while maintaining queue lengths bounded by $\mathcal{O}(k)$. The analysis is simple and the algorithm seems practical. Essentially we present an uniaxial algorithm which routes k-permutations with $k \cdot n + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ steps. This algorithm consists of three phases, one for randomizing the packets, followed by two phases in which the packets are routed to their destination. Applying the coloring technique described in Section 3, the routing time is reduced to $k \cdot n/2 + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$. Only for $k < 8$, the results are not optimal. For these k better algorithms are discussed in [3]. In Section 7.2, the algorithm is considered for cut-through routing, and in Section 7.3 it is generalized for routing on meshes of arbitrary dimension.

7.1 k-k routing

In this section we present an optimal algorithm for routing k-permutations on two-dimensional meshes without wrap-around connections, and analyse it using the results of Section 5.1.1.

The first stage of the algorithm is to attribute a color to the packets:

for every packet p **do**
assign randomly a color green or blue to p.

In the second stage of the algorithm, the packets are treated differently depending on their color. The routing of the green packets is given by

Proc NoWrapRoute(green);
for every green packet p with destination $P_{i,j}$ **do**
1. randomize p along the column;
2. route p along the row to column j;
3. route p along the column to row i.

The blue packets are routed in parallel with the green packets, 90° out of phase with them. The routing of

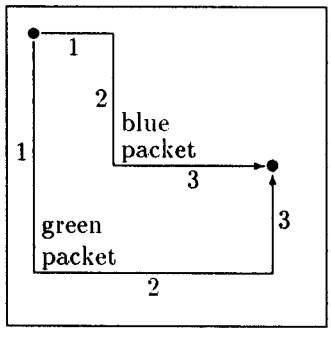


Figure 5: The routing of a green and a blue packet.

a green and a blue packet coming from the same PU and going to the same PU is illustrated in figure 5. NoWrapRoute can be seen as a modification of the basic algorithm of [2]. The only difference is that in NoWrapRoute the packets are randomized more thoroughly.

Now, we will analyse the routing time of NoWrapRoute. The discussion of the queue size is given in Section 7.5.

Lemma 16 *Phase 1 of NoWrapRoute can be carried out with* $\max\{n, k \cdot n/8\} + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ *routing steps, with very high probability.*

Proof: For the number $h_u(i,j)$ of green packets in a certain column that have to go from row i or lower to row j or higher, Lemma 1 gives $h_u(i,j) \leq k/2 \cdot i \cdot (n-j)/n + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$. The proof is completed using Lemma 2, analogously to the proof of Lemma 3. □

Lemma 17 *Phase 2 of NoWrapRoute can be carried out with* $\max\{n, k \cdot n/4\} + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ *routing steps, with very high probability.*

Proof: Consider the green packets in any subset $\mathcal{S}$, of m PUs of an arbitrary row of the mesh. These packets are there as a result of the randomization in phase 1. Any of the at most $k/2 \cdot n \cdot m + \mathcal{O}((k \cdot m \cdot n \cdot \log n)^{1/2})$ green

packets in the columns corresponding to $\mathcal{S}$ has probability $1/n$ to be randomized to a PU of $\mathcal{S}$. Lemma 1 then gives that there are at most $k/2 \cdot m + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ green packets in $\mathcal{S}$. Substituting now in Corollary 1 gives the result. □

Lemma 18 *Phase 3 of NoWrapRoute can be carried out with* $\max\{n, k \cdot n/8\} + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ *routing steps, with very high probability.*

Proof: The routing in phase 3 is reversing the randomization of phase 1. This takes just as many steps. More formally, this can be proved using Lemma 2. □

Lemma 19 *The routing of the green and blue packets can be performed independently.*

Proof: Routing only the green or blue packets gives uniaxial algorithms, with phases of the same length. So the routing of green packets interferes in no way with the routing of blue packets. □

Summing the routing time of the three phases we get our main result

Theorem 2 *NoWrapRoute routes k-permutations on the two-dimensional mesh without wrap-around connections with* $\max\{3 \cdot n, 2 \cdot n + k \cdot n/4, k \cdot n/2\} + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ *routing steps, with very high probability.*

The maximum can be split out as follows (leaving away the term $\mathcal{O}((k \cdot n \cdot \log n)^{1/2})$):

	number of routing steps
$k \leq 4$	$3 \cdot n$
$4 \leq k \leq 8$	$2 \cdot n + k \cdot n/4$
$8 \leq k$	$k \cdot n/2$

For all $k \geq 8$, the given result matches the bisection bound except for the small additional term. In [3] we improve on this for $k < 8$.

It seems peculiar that we can match the bisection bound notwithstanding the routing steps that are "lost" during the randomization. By the randomization packets may go uselessly over the already heavily loaded central cuts. Fortunately, packets only pass the central cuts uselessly when these cuts where not fully loaded. E.g., when randomizing packets that have to be routed by k-LS, no packet will pass a central cut twice.

7.2 Cut-through routing

In this section we consider the performance of NoWrapRoute when the arbitrary strategy is employed. In the analysis we use results from Section 5.1.2. From this result, we can immediately obtain the routing time of NoWrapRoute for routing packets consisting of k flits each in the cut-through mode.

Lemma 20 *Phase 1 and 3 of NoWrapRoute take at most* $\max\{n, k \cdot n/8 + n/(2 \cdot k) + n/2\} + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ *routing steps each, with very high probability.*

Proof: Analogously to the proof of Lemma 16 and Lemma 18, using Lemma 5. □

Lemma 21 *Phase 2 of NoWrapRoute takes at most* $\max\{n, k \cdot n/4 + n/2\} + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ *routing steps, with very high probability.*

Proof: Analogously to the proof of Lemma 17, using Corollary 2. □

Summing gives

Theorem 3 *Employing the arbitrary queuing strategy, NoWrapRoute routes k-permutations on the two-dimensional mesh without wrap-around connections with* $\max\{3 \cdot n, k \cdot n/2 + n/k + 3/2 \cdot n\} + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ *routing steps, with very high probability.*

Corollary 5 *NoWrapRoute performs cut-through routing on the two-dimensional mesh without wrap-around connections of packets consisting of k flits each, with at most* $\max\{3 \cdot n, k \cdot n/2 + n/k + 3/2 \cdot n\} + \mathcal{O}(k \cdot (n \cdot \log n)^{1/2})$ *routing steps, with very high probability.*

The results of Theorem 3 and Corollary 5 are close to optimal for all $k \geq 2$. As the case $k = 1$ has no practical value for the cut-through routing, this effectively means that the cut-through routing problem is solved.

7.3 Higher dimensions

In this section we will consider the routing of k-permutations on meshes without wrap-around connections of arbitrary dimension d. As the diameter of an d-dimensional mesh with n^d PUs is $d \cdot (n-1)$ one might expect that the routing time would grow with d. This is not the case when k is sufficiently large: we will be able to show that if $k \geq 4 \cdot d$, k-permutations can be routed with, $k \cdot n/2 + \mathcal{O}(d \cdot (k \cdot n \cdot \log n)^{1/2})$ routing steps. This almost matches the bisection bound. This result is achieved by generalizing NoWrapRoute to an algorithm with $2 \cdot d - 1$ phases.

The algorithm starts by coloring the packets randomly:

for every packet p **do**
  attribute randomly a color c_i, $1 \leq i \leq d$ to p;

It is not important whether all packets in a single PU get different colors or that the colors are attributed completely randomly. After the coloring, the packets of color c_l, $1 \leq l \leq d$, are routed by

Proc NoWrapRoute(d, l);
{ All indices are cyclical in the range $1, \ldots, d$. }
for every packet p of color l **do**
  for $j := l$ **to** $l + d - 2$ randomize p along axis l;
  route p to its position on axis $l + d - 1$;
  for $j := l$ **to** $l + d - 2$ **do**
    route p to its position on axis j.

Analogously to Lemma 16, Lemma 17 and Lemma 18, we have

Lemma 22 *Phase* $1, \ldots, d-1$ *and* $d+1, \ldots, 2 \cdot d$ *of NoWrapRoute(d) can be carried out with* $\max\{n, k \cdot n/(4 \cdot d)\} + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ *routing steps, with very high probability.*

Lemma 23 *Phase d of NoWrapRoute(d) can be carried out with* $\max\{n, k \cdot n/(2 \cdot d)\} + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ *routing steps, with very high probability.*

In total,

Theorem 4 *NoWrapRoute(d) routes k-permutations on the d-dimensional mesh without wrap-around connections with* $\max\{(2 \cdot d - 1) \cdot n, k \cdot n/2\} + \mathcal{O}(d \cdot (k \cdot n \cdot \log n)^{1/2})$ *routing steps, with very high probability.*

For cut-through routing we get

Theorem 5 *NoWrapRoute(d) performs cut-through routing on the d-dimensional mesh without wrap-around connections of packets consisting of k flits each, with at most* $\max\{(2 \cdot d - 1) \cdot n, k \cdot n/2 + (d-1) \cdot n/k + (d - 1/2) \cdot n\} + \mathcal{O}(d \cdot k \cdot (n \cdot \log n)^{1/2})$ *routing steps, with very high probability.*

The result of Theorem 4 matches the bisection bound for $k \geq 4 \cdot d$. The result of Theorem 5 is close to optimal for $k \geq 2 \cdot d - 2$.

7.4 Average-case performance

NoWrapRoute(d) performs extremely good for routing k-permutations when only the worst-case routing time is considered. How about the average-case?

For routing random permutations, the randomization phases of NoWrapRoute(d) can be left away. This shows that on the average (for sufficiently large k), permutations can be routed with $k \cdot n/4 + \mathcal{O}(d \cdot (k \cdot n \cdot \log n)^{1/2})$ routing steps on meshes without wrap-around connections. Using NoWrapRoute(d) for random permutations will take $k \cdot n/2 - k \cdot n/(4 \cdot d) + \mathcal{O}(d \cdot (k \cdot n \cdot \log n)^{1/2})$ routing steps (for random permutations all $2 \cdot d - 1$ phases take $k \cdot n/(4 \cdot d) + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ routing steps). It is an interesting question whether the average-case routing time can be reduced without deteriorating the worst-case performance.

7.5 The size of the queues

The expected number of packets that resides in a PU is bounded by $k + \mathcal{O}(1)$. For $k \geq n$, the Chernoff bounds give that then the maximal number of packets in any PU is bounded by $k + o(k)$. For small k we cannot bound the number of packets in a single PU. But the Chernoff bounds give that the number of packets in any m PUs is bounded by $k \cdot m + \mathcal{O}((k \cdot m \cdot \log n)^{1/2})$. Employing the technique of Rajasekaran and Tsantilas [13] the packets in m adjacent PUs can be smeared out. This takes $\mathcal{O}(m)$ steps. So, with an additional $\mathcal{O}(m)$ steps the queue sizes can be bounded to $k + \mathcal{O}((k \cdot \log n/m)^{1/2})$. Taking $m = 1$ gives the above stated result. If $k \leq n^\alpha$, for some $\alpha < 1$, we can take $m = k \cdot \log n$. Resuming,

Theorem 6 *If* $k \leq n^\alpha$, *for some* $\alpha < 1$, *then the queue size can be bounded to* $k + \mathcal{O}(1)$, *with very high probability, at the expense of* $\mathcal{O}(n^\alpha \cdot \log n)$ *extra routing steps. For larger k the queue size is bounded by* $k + o(k)$.

Note that $\mathcal{O}(n^\alpha \cdot \log n) = o(n)$, under the given condition on k. Theorem 6 is general and in the same way the queue size of all algorithms in this paper can be bounded.

8 Routing on meshes with wrap-around connections

Previously we considered routing algorithms for meshes without wrap-around connections. For the permutation routing problem ($k = 1$), this is the most studied mesh. Primarily this is the case because for the mesh without wrap-around connections one can prove much nicer results than for the mesh with wrap-around connections. The lower bound for the permutation routing problem with wrap-around connections is $n - 1$. So far, no algorithm comes close to this (it is easy to prove that the basic algorithm of [2] routes permutations with $3/2 \cdot n + o(n)$ routing steps but we do not know of anything better). Directly using NoWrapRoute gives something comparable: k-permutations are routed with $3/8 \cdot k \cdot n + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ routing steps. This is not very satisfactory, one hopes that with wrap-around connections, a permutation is routed twice as fast as without. We will show that an alternative four-phase algorithm for routing k-permutations is optimal on both kinds of meshes. For meshes with wrap-around connections, this means a routing time $k \cdot n/4 + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$, matching the connection-availability bound.

In the first two phases of the algorithm, the packets are randomized entirely. In the last two phases the packets are routed to their destination. It is amazing that such an algorithm can perform optimally. At first sight, it seems that a lot of packets are sent in a wrong direction during the randomization, thereby wasting connection availability. Fortunately, this is only true when the permutation itself makes sub-maximal use of the connections. The argument is analogous to the argument in Section 7.1 that the heavily loaded central cuts did not exclude optimal performance of NoWrapRoute.

8.1 k-k routing

The algorithm is a four-phase variant of the earlier algorithm NoWrapRoute. The essential difference is that now the packets are randomized in both directions instead of only one direction. We will show that this gives optimal routing on meshes with and without wrap-around connections. While the algorithm NoWrapRoute can be seen as a modification of the basic algorithm of [2], the here presented algorithm more resembles the algorithm of Valiant & Brebner [14] for routing on the hypercube.

Initially the packets are colored randomly. Thereafter, the green packets are routed by

Proc WrapRoute(green);
for every green packet p with destination $P_{i,j}$ **do**
1. randomize p along the row;
2. randomize p along the column;
3. route p along the row to column j;
4. route p along the column to row i.

The blue packets are routed in parallel with the green packets, 90° out of phase with them.

The randomization really randomizes:

Lemma 24 *After phase 1 and 2 a packet has equal probability to be in any of the n^2 PUs.*

Proof: Consider a packet residing in $P_{i,j}$. This packet is after phase 1 and 2 in $P_{i',j'}$ iff it was randomized in phase 1 to $P_{i,j'}$ and in phase 2 from there to $P_{i',j'}$. The probability that this happens is $1/n^2$ independent of i', j'. □

Once we know that the packets are entirely randomly distributed, phase 3 is just an inverse of a randomization and can be performed with the same number of routing steps. Likewise for phase 4. Now, we can use Lemma 8 for all four phases:

Theorem 7 *WrapRoute routes k-permutations on the two-dimensional mesh with wrap-around connections with* $\max\{2 \cdot n, k \cdot n/4\} + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ *routing steps, with very high probability.*

For $k \geq 8$ this result matches the connection-availability bound. The algorithm is also suited for routing on meshes without wrap-around connections: Using Lemma 3, we get

Theorem 8 *WrapRoute routes k-permutations on the two-dimensional mesh without wrap-around connections with* $\max\{4 \cdot n, k \cdot n/2\} + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ *routing steps, with very high probability.*

For $k \geq 8$ this result matches the bisection bound. For routing on meshes without wrap-around connections, NoWrapRoute has small advantages in comparison with WrapRoute: NoWrapRoute is already optimal for $k \geq 6$ [3]. Furthermore, the average case routing time of WrapRoute is even higher than that of NoWrapRoute: $k \cdot n/2 + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ instead of $3 \cdot k \cdot n/8 + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$.

8.2 Cut-through routing

In this section we will show that using WrapRoute on meshes with wrap-around connections with the arbitrary conflict resolution strategy, routes k-permutations with at most $k \cdot n/4 + 2 \cdot n + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ routing steps with very high probability. For routing packets consisting of k flits each in the cut-through model, this gives the same result (except for a slightly higher additional term).

Using Lemma 11 gives

Theorem 9 *NoWrapRoute with arbitrary queuing strategy on a mesh with wrap-around connections, takes at most* $k \cdot n/4 + 2 \cdot n + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ *routing steps, with very high probability.*

Note that this theorem holds for all values of k. For the cut-through routing we must use Lemma 12. This gives

Theorem 10 *Using NoWrapRoute for routing packets consisting of k flits each in the cut-through model on a mesh with wrap-around connections, takes at most* $k \cdot n/4 + 2 \cdot n + \mathcal{O}(k \cdot (n \cdot \log n)^{1/2})$ *routing steps, with very high probability.*

8.3 Higher dimensions

We considered k-k and cut-through routing on meshes with wrap-around connections of dimension 1 and 2. In this section we will generalize the four-phase algorithm of Section 8.1 to a $2 \cdot d$ phase algorithm for routing on meshes with wrap-around connections of arbitrary dimension d. When k is not too small, we can match the connection-availability bound.

Initially, the packets are colored randomly as in Section 7.3. After this, the packets are randomized and routed. The packets with color l, $1 \leq l \leq d$, by

```
    Proc WrapRoute(d, l);
{ All indices are cyclical in the range 1, ..., d. }
for every packet p of color l do
  for j := l to l + d − 1 do
    route p to a random destination along axis j;
  for j := l to l + d − 1 do
    route p to the correct position on axis j;
```

The packets of different colors move along different axis at all times. After Randomize every packet has equal probability to be in any PU. This is demonstrated analogously to Lemma 24. In total there are $2 \cdot d$ routing phases. Each of them takes as much time as a randomization on a k/d loaded circular one-dimensional processor array. Thus, by Lemma 8

Lemma 25 *On a mesh with wrap-around connections, all phases of WrapRoute(d) can be carried out with* $\max\{n, k \cdot n/(4 \cdot d)\} + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ *routing steps, with very high probability.*

In total,

Theorem 11 *WrapRoute(d) routes k-permutations on the d-dimensional mesh with wrap-around connections with* $\max\{2 \cdot d \cdot n, k \cdot n/4\} + \mathcal{O}(d \cdot (k \cdot n \cdot \log n)^{1/2})$ *routing steps, with very high probability.*

It turns out, that apart from a lower order term, the routing time for k-permutations is independent of the dimension of the mesh as long as there are enough packets to assure a good loading. When there are less packets the diameter of the mesh, $d \cdot (n-1)$, will play a role. The result of Theorem 11 matches the bisection bound for $k \geq 4 \cdot d$.

For cut-through routing we get

Theorem 12 *WrapRoute performs cut-through routing on the d-dimensional mesh with wrap-around connections of packets consisting of k flits each, with at most $k \cdot n/2 + d \cdot n + \mathcal{O}(d \cdot k \cdot (n \cdot \log n)^{1/2})$ routing steps, with very high probability.*

This is close to optimal for all $k \geq 2$.

9 Routing with respect to locality

In this section we will describe how our algorithm works in an environment, where each packet only has to travel a short distance. In [13] similar analysis can be found, but the results are worse and not as general as ours. We describe only the two-dimensional case, generalizations can be done in the same way as described above. We consider two cases: In the first case, which is easier to analyse but which is not directly applicable to practice, we assume that the maximum distance L any packet has to travel in one directions is known to all PUs. We say the problem has locality L. In the second case we assume that the PUs have only local information. The lower bound for routing with locality L is higher than might be expected: Consider a permutation under which all packets (except possibly some packets close to a border of the mesh) have to move exactly L steps to the right and L steps up. In total the $k \cdot n^2$ packets have to move $2 \cdot L \cdot k \cdot n^2$ steps. Every PU has four connections over which packets could be transmitted, but only two of them are useful for bringing packets closer to their destination. Thus, this routing will take at least $k \cdot L$ routing steps.

9.1 Locality L

In this section we analyse k-k routing when the maximal distance L a packet has to go in one direction is known to all PUs. We consider a three-phase algorithm resembling the basic algorithm of [2]. We will show that with queues of maximal length $k + \mathcal{O}(1)$ k-permutations can be routed with $17/16 \cdot k \cdot L$ routing steps on meshes with wrap-around connections and with $9/8 \cdot k \cdot L$ routing steps on meshes without wrap-around connections. This result is pretty close to the optimal routing time of $k \cdot L$.

We use a modification of the basic algorithm of [2].

Proc LocalRoute(L);

1. **for** every packet p in $P_{i,j}$ **do**
   generate a random number $x_p \in \{L/2, \ldots, L/2\}$;
   route p to row $i + x_p$;
3. **for** every packet p with destination $P_{i,j}$ **do**
   route p to column j;
2. **for** every packet p with destination $P_{i,j}$ **do**
   route p to row i;

LocalRoute is uniaxial. Later on, we will use coloring to reduce the number of routing steps.

We analyse the algorithm phase by phase for a PU that does not lie within distance L from a boundary of the mesh (for a mesh with wrap-around connections this is the case for all PUs). The analysis is based on Lemma 2 and Lemma 7. We will only determine the maximal distance over which a packet has to move during a phase and the maximal number of packets a connection has to transmit. As we saw before, when the origins or the destinations of the packets are uniformly distributed, there are these two facts that determine the routing time.

Lemma 26 *Phase 1 takes at most* $\max\{L/2, k/8 \cdot L\} + \mathcal{O}((k \cdot L \cdot \log n)^{1/2})$ *routing steps.*

Proof: In phase 1 a packet moves over distance $L/2$ at most. A packet residing in $P_{i,j-j'}$ is randomized to a PU lying higher than $P_{i,j}$ with probability $(L/2 - j')/L$. This gives $k/L \cdot \sum_{j'=0}^{L/2}(L/2 - j') \simeq k \cdot L/8$ for the

expected number of packets that are routed through $P_{i,j}$ from below. □

After phase 1 each packet resides in a random row within distance $L/2$ from the row where it started. The packets are still well-distributed: Any subset of m PUs contains at most $k \cdot m + \mathcal{O}((k \cdot m \cdot \log n)^{1/2})$ packets. Hereby,

Lemma 27 *During phase 2 no connection has to transmit more than* $k \cdot L + \mathcal{O}((k \cdot L \cdot \log n)^{1/2})$ *packets.*

Proof: In phase 2 no packet is moving over more than n steps. So, at most the packets residing in L PUs can pass in one direction through any PU. □

More appealing randomization techniques, requiring less routing steps, do not have this essential property. Lemma 27, and the fact that the maximum distance packets go is L, give

Lemma 28 *Phase 2 takes at most* $k \cdot L + \mathcal{O}((k \cdot L \cdot \log n)^{1/2})$ *routing steps.*

The analysis of LocalRoute is completed with

Lemma 29 *Phase 3 takes at most* $\max\{3/2 \cdot L, k \cdot L\} + \mathcal{O}((k \cdot L \cdot \log n)^{1/2})$ *routing steps.*

Proof: In phase 3, a packet moves over distance $3/2 \cdot L$ at most (when it had to go L steps and was randomized in phase 1 to the extreme of the strip of size L around its origin). Consider the packets that have to pass through a PU $P_{i,j}$ from below. These packets must have their destination in one of the PUs $P_{i+i',j}$, for some $1 \leq i' \leq L$. There are $k \cdot L$ such packets. □

Introducing two colors and routing the green and blue packets 90° degrees out of phase, we get

Theorem 13 *k-k routing with locality L can be performed on a mesh with wrap-around connections with* $\max\{5/2 \cdot L, k \cdot L\} + \max\{L/2, k \cdot L/16\} + \mathcal{O}((k \cdot L \cdot \log n)^{1/2})$ *routing steps, with very high probability.*

Corollary 6 *For* $k \geq 8$*, a k-k routing problem with locality L can be performed on a mesh with wrap-around connections with* $17/16 \cdot k \cdot L + \mathcal{O}((k \cdot L \cdot \log n)^{1/2})$ *routing steps, with very high probability.*

The queues are bounded by a smearing technique to $k + \mathcal{O}(1)$ (see Section 3).

On a mesh without wrap-around connections the borders give some complications. In this case, the best thing to do is to apply NoWrapRoute′ with $\epsilon = L$. The phases now take (before coloring and neglecting the additional term) $k \cdot L/4$, $k \cdot L$ and $k \cdot L$ routing steps, respectively. Summing, we find

Theorem 14 *A k-k routing problem with locality L can be performed on a mesh without wrap-around connections with* $\max\{L, k \cdot L/8\} + \max\{2 \cdot L, k \cdot L\} + \mathcal{O}((k \cdot L \cdot \log n)^{1/2})$ *routing steps, with very high probability.*

Accepting larger queues, the routing time can be reduced. If the initial randomization is done in strips of size ϵ, the queues can become $k \cdot L/\epsilon$ long, while the routing time (for both types of meshes) becomes $\max\{L, k \cdot \epsilon/8\} + \max\{2 \cdot L, k \cdot L\} + \mathcal{O}((k \cdot L \cdot \log n)^{1/2})$. The results of this section are only interesting when $k \cdot L = \Omega(\log n)$. For smaller $k \cdot L$ it is not difficult to give a deterministic algorithm that performs the routing with $\mathcal{O}(k \cdot L)$ routing steps while maintaining short queues (e.g., move the packets to the right block of size $L \cdot L$; and route them ito their destination by sorting).

9.2 Local information only

Now, we consider the case that the locality value L is not known, but that each PU only knows the destination of its own packets. For ease we assume that the mesh has wrap-around connections.

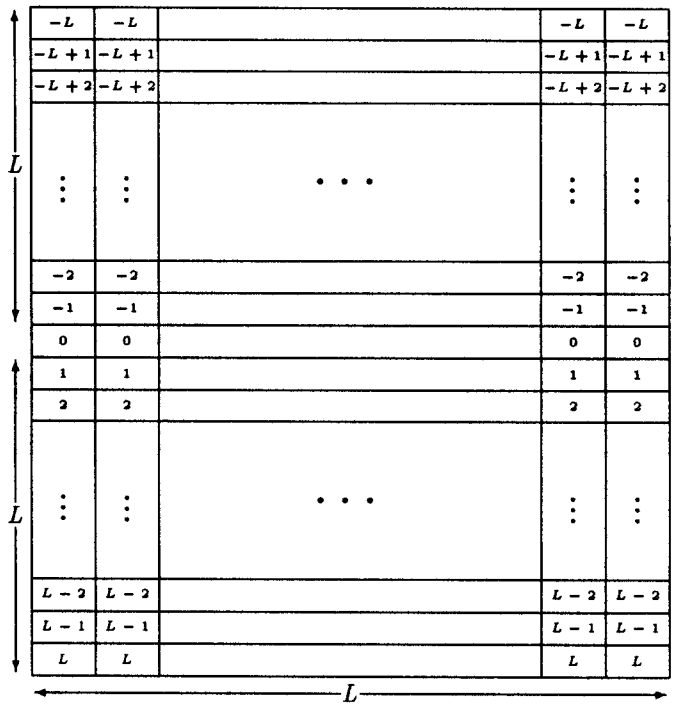


Figure 6: A distribution of packets that may lead to a long phase 2.

For phase 2 and phase 3 it is not essential to know L. An estimate of L is only necessary for knowing how large the strips must be in which the randomization of phase 1 is carried out. A first idea is to use the range between the origin and the destination of a packet as an estimate for L. Using this approach, there are distributions of packets such that after phase 1 all the PUs of a row contain an expected number of $\Omega(\log L)$ packets, an example is given in Figure 6. Here the numbers indicate the distance the packets residing in a PU have to move up or down. All these packets have to move L positions to the right. When all PUs of some row contain $\Omega(\log L)$ packets that have to move L positions to the right, phase 2 will take $\Omega(k \cdot L \cdot \log L)$ steps. This is much too long. We must assure that after phase 1 Lemma 27 holds. Any approach that does not randomize all packets in strips of the same size ϵ will give comparable problems. So, the PUs of a column must agree upon a common value for ϵ. Which common value for ϵ? Because initially the packets are perfectly distributed over the mesh, we could take $\epsilon = 1$. After this, Lemma 27 holds. But then all packets from a row may come together in one PU during phase 2. For the routing time this does not matter, but such queues are unacceptable. Therefore, ϵ should be (a fraction of) the maximal horizontal move. Obviously, this ϵ can be found and distributed during a preprocessing phase at the expense of n routing steps. In the remainder we will consider an interesting approach that, with slightly longer queues, requires almost no extra routing time.

It is not necessary that we know the precise value of L. A good estimate of it can be just as useful for assuring that the queues will not get too long after phase 2: E.g., suppose that a PU can hold at most $2 \cdot k + \mathcal{O}(1)$ packets. Then, we can take any $\epsilon \geq L/2$. Thus, when L is unknown, we only have to calculate its value within a factor two. This requires the transmission of much less packets than the calculation of the exact value of L. The connections that are not used for transmitting these information packets can be used for a preliminary randomization of the packets. In more detail, we use the following algorithm:

Proc NoLRoute(green);

1. a. **for** every PU $P_{i,j}$ **do**

$L_{i,j}$:= the largest move in one direction a packet residing in $P_{i,j}$ has to make;

route a packet with $L_{i,j}$ in all four directions;
if a packet with $L' \geq 2 \cdot L_{i,j}$ reaches $P_{i,j}$
then $L_{i,j} := L'$; route a packet with L' in all four directions
else eliminate the packet;

1. b. **for** every green packet p in $P_{i,j}$ **do**
generate a random number $x_p \in \{-8 \cdot n/k, \ldots, 8 \cdot n/k\}$;
route a packet (p, i, x_p) to row $i + x_p$;

1. c. **for** every PU $P_{i,j}$ **do**
$L_{i,j} := 2^{\lfloor \log L_{i,j} \rfloor}$;

1. d. **for** every PU $P_{i,j}$ **do**
if $L_{i,j}/2 > 8 \cdot n/k$ **then** { the randomization is not yet completed }
for every green packet (p, i', x_p) residing in $P_{i,j}$ **do**
Route p to row $i' + x_p \cdot L_{i,j} \cdot k/(16 \cdot n)$;

2. **for** every green packet p **do**
route p to the column of its destination;

3. **for** every green packet p **do**
route p to the row of its destination;

NoLRoute is a modification of LocalRoute. The blue packets are routed 90° degrees out of phase with the green packets. The packets of step 1.a are called information packets. No PU will transmit more than $\log n$ information packets over a single connection. Step 1.a and 1.b are coalesced with priority for the information packets. Step 1.b randomizes the packets in strips of size $8 \cdot n/k$. Because every PU holds $k/2$ green packets on the average, this can be done with $n + \mathcal{O}((n \cdot \log n)^{1/2})$ routing steps. Thus, step 1.a and step 1.b can be routed together with $n + \mathcal{O}((n \cdot \log n)^{1/2})$ routing steps. Step 1.c is executed after step n. After step 1.c all PUs will have the same estimate L' satisfying $L/2 < L' \leq L$. If the randomization of step 1.b was good enough, the packets can be routed to their destination immediately (phase 2 and phase 3), otherwise some additional randomization has to be performed first (step 1.d). All together we have

Theorem 15 *A k-k routing problem with unknown locality L can be performed on a mesh with wrap-around connections with* $\max\{n, k \cdot L/16\} + k \cdot L + \mathcal{O}(((n + k \cdot L) \cdot \log n)^{1/2})$ *routing steps, with very high probability.*

For sufficiently large k this result is the same as the result of Theorem 13. Routing more information packets, the value of L can be determined more precisely. Theorem 15 will remain true as long as the number of information packets transmitted over a connection remains bounded by $\mathcal{O}((n \cdot \log n)^{1/2})$. Possibly, the information necessary for the spread of L could be attached to the packets that are randomized. This reduces the number of packets and allows to determine L exactly. However, this might involve a lot of overhead for composing and decomposing packets, because the packets that are randomized only go a short way.

10 Conclusion

In this paper we presented randomized algorithms for routing k-permutations. We achieved optimality for all $k \geq 8$. For the cut-through routing problem, for routing with wrap-arounds, and for routing in higher dimensions we proved surprisingly sharp results as well.

Acknowledgement

It was Sanguthevar Rajasekaran who introduced us to the problem. During stimulating discussions in Saarbrücken with the three of us, the ideas of Section 6 where born. We also want to thank Marinus Veldhorst for his contribution to the precision of many proofs and the consistency of the notation.

References

[1] Hagerup, T., and Rüb, C. 'An efficient guided tour of Chernoff bounds,' Techn. Rep. A 88/03, Universität des Saarlandes, 1988. Also, Inf. Proc. Lett. 33, 305-308, 1990.

[2] Krizanc, D., Rajasekaran, S., and Tsantilas, T. 'Optimal Routing Algorithms for Mesh-Connected Processor Arrays,' Proc. VLSI Algorithms and Architectures: AWOC '88. Springer-Verlag Lecture Notes in Computer Science #319, pp. 411-22, 1988. To appear in Algorithmica.

[3] Kaufmann M., Sibeyn J.F., 'Multipacket Routing in Lightly Loaded Processor Arrays', draft (1991).

[4] Kunde, M. 'Routing and Sorting on Mesh-Connected Processor Arrays,' Proc. VLSI Algorithms and Architectures: AWOC '88. Springer-Verlag Lecture Notes in Computer Science #319, Springer-Verlag, pp. 423-33, 1988.

[5] Kunde, M., and Tensi, T. 'Multi-Packet Routing on Mesh Connected Arrays,' Proc. ACM Symposium on Parallel Algorithms and Architectures, SPAA 89, ACM Press, pp. 336-343, 1989.

[6] Kunde, M., 'Concentrated Regular Data Streams on Grids: Sorting and Routing Near to the Bisection Bound,' Proc. IEEE Symposium on Foundations of Computer Science, pp. 141-150, 1991.

[7] Kunde, M. 'Balanced Routing: Towards the Distance Bound on Grids,' Proc. 1991 ACM Symposium on Parallel Algorithms and Architectures, SPAA 91, ACM Press, pp. 260-271, 1991.

[8] Leighton, T., Makedon, F., and Tollis, I.G. 'A $2n - 2$ Step Algorithm for Routing in an $n \times n$ Array With Constant Size Queues,' Proc. ACM Symposium on Parallel Algorithms and Architectures, SPAA 89, ACM Press, pp. 328-35, 1989.

[9] Leighton, T., 'Average Case Analysis of Greedy Routing Algorithms on Arrays,' Proc. ACM Symposium on Parallel Algorithms and Architectures, SPAA 90, ACM Press, pp. 2-10, 1990.

[10] Makedon, F., Simvonis, A. 'On bit Serial packet routing for the mesh and the torus.' Proc. third Symposium on Frontiers of Massively Parallel Computation, pp. 294-302, 1990.

[11] Rajasekaran, S., and Overholt, R. 'Constant Queue Routing on a Mesh,' Proc. Symposium on Theoretical Aspects of Computer Science, Springer-Verlag Lecture Notes in Computer Science #480, pp. 444-455, 1990. To appear in JPDC.

[12] Rajasekaran, S., and Raghavachari, M. 'Optimal Randomized Algorithms for Multipacket and Cut Through Routing on the Mesh,' to be presented at the 3rd Symposium on Parallel and Distributed Computing, 1991.

[13] Rajasekaran, S., and Tsantilas, T. 'An Optimal Randomized Routing Algorithm for the Mesh and A Class of Efficient Mesh-like outing Networks,' Proc. 7th Conference on Foundations of Software Technology and Theoretical Computer Science, Springer-Verlag Lecture Notes in Computer Science #287, pp. 226-241, 1987.

[14] Valiant, L., and Brebner, G.J., 'Universal schemes for parallel communication,' Proc. 13th ACM Symp. on Theory of Comp., pp 263-277, 1981.