Routing and Sorting on Circular Arrays

Jop F. Sibeyn *

Abstract

We analyze routing and sorting problems on circular processor arrays with bidirectional connections. We assume that initially and finally each PU holds $k \geq 1$ packets. On linear processor arrays the routing and sorting problem can easily be solved for any k, but for the circular array it is not obvious how to exploit the wrap-around connection.

We show that on an array with n PUs k-k routing, $k \geq 4$, can be performed optimally in $k \cdot n/4 + \sqrt{n}$ steps by a deterministical algorithm. For $k = 1$, the routing problem is trivial. For $k = 2$ and $k = 3$, we prove lower-bounds and show that these (almost) can be matched. A very simple algorithm has good performance for dynamic routing problems.

For the k-k sorting problem we use a powerful algorithm which also can be used for sorting on higher-dimensional tori and meshes. For the ring the routing time is $\max\{n, k \cdot n/4\} + \mathcal{O}((k \cdot n)^{2/3})$ steps. For large k we take the computation time into account and show that for $n = o(\log k)$ optimal speed-up can be achieved. For $k < 4$, we give specific results, which come close to the routing times.

1 Introduction

Machine Model. 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 connections which connect processing units, **PUs**, among each other. We consider this problem for the case that n PUs are connected such that they form a circular array. In the following such a processor array will be called **ring**. The PUs are indexed from 0 to $n-1$. Thus, the processor array can be depicted as in Figure 1.

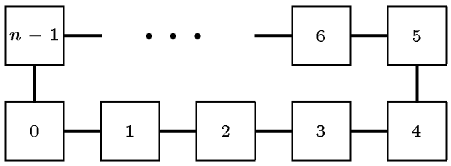


Figure 1: A ring with n indexed PUs.

Frequently we will refer to the PU with index i as P_i. The connection between P_i and P_{i+1} is denoted $(i, i+1)$. $(n-1, 0)$ is called the **wrap-around connection**. The one-dimensional network without wrap-around connection will be called **chain**.

The connections are bidirectional. In one routing step a PU can send one packet of bounded length to both its neighbors and also receive packets from both sides. The routing steps are synchronized. The PUs may store packets in a local queue. The model of computation is the MIMD model. A routing step takes time T_r, a computation step takes T_c. T_r is larger than T_c by a constant (typically about 50). Frequently only the number of routing steps will be stated.

Routing and Sorting. The basic communication problem is that of routing **permutations**: routing problems in which every PU holds a packet that must be routed to a destination PU such that every PU is the destination of precisely one packet. A generalization is the k-k **routing problem**, the problem of

*Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany. E-mail: jopsi@mpi-sb.mpg.de. This research was partially supported by EC Cooperative Action IC-1000 (project ALTEC: *Algorithms for Future Technologies*).

transporting k packets from each PU in the processor array to k, not necessarily distinct, destination PUs such that every PU finally receives k packets. Such a distribution of source/destination pairs is called a **k-k distribution**. Permutations are 1-1 distributions. $k > 1$ arises when the packets are too large to be transferred in a single routing step, or when every PU has many independent packets to send. This last situation occurs when on a network consisting of n PUs a network of $k \cdot n$ PUs is simulated. For the purpose of the routing each packet is equipped with a destination tag.

We also consider the **k-k sorting problem**. As in the k-k routing problem, each PU is origin and destination of k packets. Now however, the packets are not equipped with a *destination tag*, but with a *key* from a linearly ordered set on which they have to be sorted. This means that after the sorting, for any pair of packets p with key l and residing in P_i, and p' with key l' and residing in $P_{i'}$, we must have $l \leq l'$ iff $i \leq i'$. The k-k sorting problem arises in the same situations as the k-k routing problem.

The considered problems are related to the routing and sorting problems on mesh-connected processor arrays. In the literature most attention is given to two-dimensional meshes. The permutation routing problem has been solved optimally with deterministic algorithms requiring queues of constant length [9, 11, 1]. For the 1-1 sorting problem Kaklamanis and Krizanc presented [2] a randomized algorithm with near-optimal performance. By now, comparable performance has been achieved deterministically [6]. The k-k sorting and routing problems gained attention recently [12, 7, 3, 4, 8, 6].

The research on routing and sorting algorithms for the ring has contributed a lot to the insight that was necessary for developing the near-optimal deterministic sorting algorithms for meshes in [6]. For the future, it may be hoped that ideas that where developed for rings inspire progress for routing and sorting on meshes and tori for small k and for very large k. Apart from this, the problems for one-dimensional processor arrays have independent interest: one-dimensional architectures are very natural and can be produced easier than any other parallel architecture from very basic building blocks.

The k-k routing problem on chains is trivial: for all k optimal performance is achieved by routing packets to their destinations in a greedy way, giving priority to the packets that have to go farthest. For rings the k-k routing problem is non-trivial for $k > 1$. Makedon and Simvonis were the first to perform a serious study of this problem [10]. For the case that all packets sent by a PU have the same destination, they gave an algorithm which runs in $k \cdot n/4 + \mathcal{O}(n)$. Independently it was shown in [5] that the general k-k routing problem on a ring can be solved in $k \cdot n/4 + \sqrt{n}$ routing steps for all $k \geq 4$.

The k-k sorting problem can be solved on the chain almost optimally by applying ‘transposition sort’ (see Section 2.3) in $\max\{n, k \cdot n/2 \ + 1\}$ routing steps. Using a ring as if it were a chain would be a pity: on a ring we hope to solve any routing-like problem twice as fast as on a chain. A randomized near-optimal k-k sorting algorithm on a ring follows as a special case of the k-k sorting algorithm for meshes of arbitrary dimension in [16]. Later we showed in [15] that comparable performance can be achieved with a deterministic algorithm too.

Techniques and Results. This paper collects the key ideas and results from our papers [5] and [15]. New results are added, and the precision of some proves is enhanced. We prove lower bounds, and give algorithms for k-k routing and k-k sorting. Various ranges of values of k are considered: $k < 4$, $4 \leq k \leq n^2$, $k \geq n^2$.

In our routing algorithms the packets do not have intermediate destinations and are not reversed. Thus, the routing problem is reduced to **directing** the packets: determining for every packet the direction in which it will move. A packet is directed **towards** its destination if this direction gives the shortest way to its destination. Otherwise the packet is directed **away from** its destination. At first it seems difficult to perform the routing for $k > 1$ with less than $k \cdot n/2$ routing steps. The crucial idea is that not all packets have to be directed towards their destination. This mimics the effect of the randomization in the routing algorithm of [4]. Directing some packets away from their destination may bound the maximal number of packets that is transmitted over any connection.

Generally, it is not realistic to assume that the PUs dispose of global knowledge. An algorithm that requires that initially the PUs only know the destinations of the packets that reside in them, will be called a **local-knowledge algorithm**. This leaves open the possibility that information is gathered during a prephase (adaptive algorithms may even profit from information that becomes available during the routing). But, such a prephase might be undesirable or difficult to perform (e.g., in dynamic environments). An algorithm in which the decision how to route a packet p is solely based on information that is initially available in the PU where p resides, is called a **fully-local-knowledge algorithm**. For this class of algorithms we can prove higher lower bounds. The routing algorithms of this paper are deterministic, oblivious and queue optimal. Here **oblivious** means that the path of a packet is not-influenced by other packets after the packet is sent off.

There are several possible approaches for fully-local-knowledge algorithms. For general k. The fastest routing is obtained when the number of packets that a PU directs leftwards increases linearly with the sum of the distances that these packets have to move rightwards. In this way k-k distributions can be routed in $k \cdot n/4 + n/4$ steps. For large k this converges to a lower bound.

The $n/4$ term in the routing time of this algorithm is due to 'rounding errors' which are inherent to fully-local-knowledge algorithms. But, in a local-knowledge algorithm preprocessing can be performed in subchains of size $\sqrt{n}$, reducing the routing time to $k \cdot n/4 + \sqrt{n}$ for all $k \geq 4$. Modifying the algorithm slightly, 2-2 routing can be performed in $2/3 \cdot n + \sqrt{n}$, and 3-3 routing in $6/7 \cdot n + \sqrt{n}$. Except for $k = 3$ all these results approximate lower-bounds to within a lower order term.

Of even greater practical value is the algorithm we give for **dynamic routing problems**, problems in which packets are generated during the routing. The algorithm is simple. Packets are not hold back, they are sent away upon their generation. If it would be applied to k-k distributions, these would be routed in $k \cdot n/3 + n/3$ steps.

For the k-k sorting problem, we present an algorithm which runs in $\max\{n, k \cdot n/4\} \cdot T_r + \mathcal{O}(k^{2/3} \cdot n^{2/3} \cdot (T_r + \log k \cdot T_c))$ time, for all $4 \leq k \leq n^2$. This is achieved by alternating local sorting operations and 'scattering' operations. The algorithm is a deterministic refinement of randomized algorithms inspired by [2] and [16]. For $k \geq n^2$, the algorithm becomes extremely simple: the local sorting operations can be performed internally, or between neighbors. The time consumption is $k \cdot n/4 \cdot T_r + \mathcal{O}(k \cdot T_r + k \cdot \log k \cdot T_c)$. For $n = o(\log k)$, the routing time becomes negligible in comparison to the computation time: for these k optimal speed-up can be achieved over a sequential algorithm.

For k-k sorting with small k it is particularly difficult to exploit the wrap-around connection and to construct an algorithm which requires less than n steps. So far packets were not copied. If copying is allowed then the routing time can be reduced. For $k = 1$, we can use that many connections remain unused during the sorting. We do not loose time if copies of the packets are spread. If the PUs can hold at most c packets then the sorting is performed in $n/2 + n/(2 \cdot c)$ steps. For $k = 2$ we need $3/4 \cdot n$ steps, for $k = 3$ about $0.95 \cdot n$ steps. These results are the first to show that sorting on a ring can be performed with less than the trivial n steps.

Our sorting algorithms can be viewed as routing algorithms, interlaced with some short sorting phases. In practice such an approach could be superior over an odd-even transposition algorithm even on a chain: when using odd-even transposition sort, after every step a PU must insert the packets it received at the correct position among the packets it holds. In our algorithms, in most steps a PU only has to consider whether a packet must be routed on or that it has reached its destination.

Summary. We resume the most important results of this paper in Table 1 and Table 2. On the

k	flk			lk		gk	
1		1/2	1/2	1/2	1/2	1/2	1/2
2		41/58	3/4	2/3	2/3	2/3	2/3
3		1	1	7/9	6/7	3/4	6/7
≥ 4	even odd	$k/4 + 1/(4+4/k)$ $k/4 + 1/4$	$k/4 + 1/4$	$k/4$	$k/4$	$k/4$	$k/4$

Table 1: Lower and upper bounds for routing under various assumptions on the available knowledge, and for various values of k.

left side of every entry the lower bound is given, on the right side the number of steps taken by an algorithm. In all results a factor n is omited, and lower order terms are neglected. In Table 1 the three main columns correspond to fully-local-knowledge routing, local-knowledge routing and global-knowledge routing, respectively. In Table 2 we distinguish sorting without making copies and sorting in which it is allowed to make copies.

Contents. The remainder of this paper is organized as follows: we start with general notions and basic

k	no copies		copies	
1	2/3	1	1/2	1/2
2	5/7	1	2/3	3/4
3	7/9	1	7/9	0.95
≥ 4	$k/4$	$k/4$	$k/4$	$k/4$

Table 2: Results for sorting, assuming that copying is either forbidden or allowed.

results. In Section 3 we give lower bounds. Then we consider the k-k routing problem. Section 5 presents the sorting algorithms.

2 Preliminaries

2.1 Efficiency Criteria

For routing and sorting algorithms certain quality measures are generally accepted. An algorithm is considered efficient if it bounds the number of steps as well as the maximal used size of the queues. Deterministic algorithms are superior over randomized algorithms. Preferably the algorithms should run without making copies. On-line algorithms are considered much more interesting than off-line algorithms. Ideally routing algorithms should allow for dinamization and should be oblivious (for sorting algorithms this does not appear to make sense). In an oblivious algorithm the path of a packet can be stamped onto it at the start, and the packet does not have to be processed anymore during its routing.

Our main algorithms are near-optimal in the following sense:

Definition 1 *An algorithm is called* **near-optimal** *if (1) there is a function f such that for all (considered) n and an arbitrary input I of size n the time consumption for I, $T(I)$, satisfies $T(I) \leq f(n)+o(f(n))$; and (2) for all n there is an input I' for which $f(n)$ is a lower bound on the time consumption.*

2.2 Basic Routing Lemmas

We present basic lemmas for routing on a chain or a ring consisting of n PUs. When several packets are competing for the use of the same connection the packet that has to go farthest should get priority. This strategy is called the **farthest-first strategy**.

Let for a given distribution of packets $h_r(i,j)$ denote the number of packets that is passing from left to right through both i and j. For routing on a ring this leaves open the possibility that $i > j$ and that the packets go through $(n-1, 0)$ before they reach j. Let T_r be the number of steps a distribution of packets requires for routing the packets that travel rightwards. T_l is defined analogously. In [3] and [4], respectively, we proved

Lemma 1 *When the farthest-first strategy is used for routing a distribution of packets on a chain, then* $T_r = \max_{i<j}\{j - i + h_r(i,j)\} - 1$.

Lemma 2 *When the farthest-first strategy is used for routing a distribution of packets on a ring, then* $T_r = \max\{\ \max_{i<j}\{j-i+h_r(i,j)\} - 1,\ \max_{j<i}\{n+j-i+h_r(i,j)\} - 1\ \}$.

For T_l analogous lemmas hold.

In all cases were the sources and destinations are regularly distributed, these lemmas reduce to checking (1) the maximum distance a packet may have to travel; (2) the maximum number of packets a connection may have to transfer.

2.3 Routing on Chains

For chains both analysis and optimal routing algorithm are trivial. We will shortly show this.

The diameter of the chain with n PUs is $n-1$. This gives a first lower bound of $n-1$ for routing algorithms on the chain. In k-k distributions in which all packets residing in the left half of the array are sent to the right half, there are $k \cdot n/2$ packets that have to pass through the connection $(n/2-1, n/2)$. Thus

Lemma 3 *Routing on a chain with n PUs requires at least* $\max\{n-1, k\cdot n/2\}$ *routing steps.*

The **greedy algorithm** is the algorithm that routes all packets along the shortest path as soon as possible to their destination.

Lemma 4 *Using the farthest-first strategy, the greedy algorithm is optimal, for routing on a chain.*

Proof: For $k = 1$ this is obvious: no packet is delayed and the last packet reaches its destination after at most $n-1$ routing steps. Now consider the case $k > 1$. By Lemma 1 we have to determine $\max_{i<j}\{j - i + h_r(i,j)\}$. Initially, there reside at most $k\cdot(i+1)$ packets in the set of PUs with index i or smaller. There are at most $k\cdot(n-j)$ packets with destination in the set of PUs with index j or larger. Thus, $T_r \leq \max_{i<j}\{j - i + k\cdot\min\{i+1, n-j\}\}$. For $k > 1$, the maximal value is attained for $i = n/2 - 1$, $j = n/2$. □

2.4 Sorting on Chains

As a subroutine of some of the sorting algorithms for the ring we apply sorting on subchains.

For the 1-1 sorting problem on chains it is customary to use odd-even transposition sort. This means that a PU sends alternatingly a packet left- and rightwards; such that in one step all PUs with even index send in one direction and the PUs with odd index in the other. This method only works under the (unrealistical) assumption that the connections can compare packets and send the smallest of two passing packets leftwards and the largest rightwards. In that case odd-even transposition sort requires only $n-1$ steps and no queues at all. Under the same assumption, $k \geq 2$, $k\cdot n/2$ steps are sufficient for k-k sorting.

We assume that only PUs are able to compare the packets. In that case PUs must be able to hold at least two packets. The performance that is achieved with connections that act as comparators can be matched when a copy is retained for every packet that is sent. After each step one of the two identical packets is thrown away.

If one assumes that making a copy costs $\Omega(1)$ steps, then it is undesirable to make copies during every step. Taking some extra steps there is no need for copies. Special care should go to the last step. In [1] it is shown that

Lemma 5 *Without copying packets,* 1-1*-sorting can be performed in n steps. A PU holds at most two packets.*

We give an algorithm for $k \geq 2$. A PU, holding at least two packets, performs a **rightward transposition-sort step**, if it sends its packet with the largest key rightwards. A **leftward transposition-sort step** is defined analogously. A PU performs a **full transposition-sort step**, if it performs a leftward and a rightward transposition-sort step at the same time. The algorithm consists of transposition-sort steps:

1. All PUs perform $k\cdot n/2 - 1$ full transposition-sort steps.
2. All PUs perform a rightward transposition-sort step.
3. All PUs perform a leftward transposition-sort step.

The PUs at the ends, P_0 and P_{n-1} should be treated specially. P_0 participates only in the rightward steps, P_{n-1} only in the leftward steps. Using the 0-1 principle it is easy to prove that the algorithm is correct.

Lemma 6 *k-k sorting for $k \geq 2$ can be performed in $k\cdot n/2 + 1$ steps. This can be achieved without making copies of the packets. A PU holds at most $k+1$ packets.*

Modifying step 2 and step 3, the queue size can be minimalized:

Lemma 7 *k-k sorting for $k \geq 3$ can be performed in $k\cdot n/2 + 1$ steps and with maximal queue size k.*

Proof: Suppose that the packets in the PU are indexed in sorted order from 1 to k. Let $k \geq 3$. Now, in step 2, all P_i with $i \geq 1$ send in phase 2 their packets with index 2 to P_{i-1}. In step 3 these packets are returned again. □

3 Lower Bounds

Most bounds are given for the routing problem. For k-k sorting we can use that a sorting problem requires at least as many steps as the corresponding routing problem. In Section 3.5 specific bounds for the sorting problem are given.

3.1 General Bounds

For the k-k routing problem, the following 'standard' lower bounds are available:

Distance bound. A packet travels at most over distance 1 during a single routing step. Thus, a routing algorithm takes at least the diameter of the processor array, $n/2$ in our case.

Throughput bound. At most one packet can go over a connection during one routing step. Thus, a routing algorithm takes at least the maximum number of packets that has to pass through a single connection.

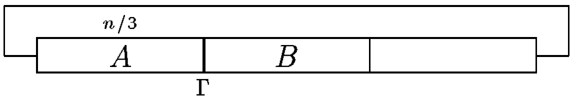
Connection availability bound. During a single step at most $2 \cdot n$ packets are moved. Thus, a routing algorithm takes at least the maximal sum of the moves of all packets divided by $2 \cdot n$ steps.

For $k = 1$, the distance bound gives the strongest result. For $k \geq 2$, the other bounds are stronger: if the $k \cdot n$ packets are arranged such that the packets from P_i have destination in $P_{i+n/2 \bmod n}$, then all packets have to move over a distance $n/2$ which takes at least $k \cdot n/4$ steps by the connection availability bound. The same result can also be obtained by considering that $k \cdot n/2$ packets have to move over the two connections $(0, n-1)$ and $(n/2-1, n/2)$.

Lemma 8 *Routing k-k distributions on a ring requires at least* $\max\{n/2, k \cdot n/4\}$ *routing steps.*

3.2 Global Knowledge

Consider the 2-2 distribution 2-$L_{n/3}$, the shift over $n/3$ under which a packet from P_i has destination in $P_{(i+n/3) \ mod \ n}$. 2-$L_{n/3}$ is such that in



all packets residing in a PU in A should move to a PU in B. If they are all routed along the shortest path, the connection Γ has to transfer $2/3 \cdot n$ packets, requiring $2/3 \cdot n$ routing steps at least. Γ may be relieved by routing a number of packets leftwards. However, the length of this way is $2/3 \cdot n$. Therefore, this does not decrease the routing time.

Lemma 9 *Routing* 2-2 *distributions on a ring requires at least* $2/3 \cdot n$ *routing steps.*

3.3 Fully-Local Knowledge

Now we analyze lower bounds for deterministic fully-local-knowledge algorithms. Our main goal is to show that for this class of algorithms the lower bounds really become higher and to demonstrate some techniques that can be used for proving this. The maximum over all packet distributions of the required number of routing steps will be denoted H. More details can be found in [14].

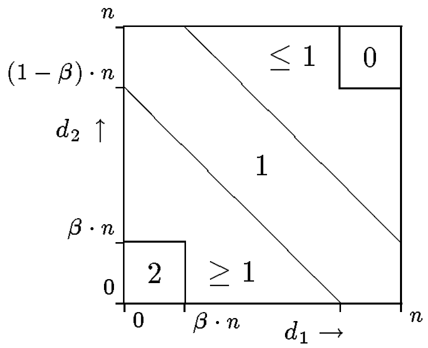
The ring is isotropic, so physical intuition tells us that the optimal direction of the packets is obtained when for even k all PUs apply the same rules. I.e., there is a global function $f : \{0, 1, \ldots, n-1\}^2 \to \{0, 1, \ldots, k\}$ which gives the number of packets that will be directed rightwards. For odd k the direction should be given by a function f_0 applied by all PUs with even index, and a function f_0 for the PUs with odd index. This intuition is hard to prove though. Therefore, we will call algorithms that are of this type **uniform**, and consider lower bounds for uniform algorithms.

The case $k = 2$. Consider a PU P holding packets p_1 and p_2 that have to travel d_1 respectively d_2 steps rightwards. In order to give minimal routing time the direction function $f : \{0, 1, \ldots, n-1\}^2 \to \{0, 1, 2\}$

must satisfy some obvious rules:

$$
\begin{aligned}
f(n-d_1, n-d_2) &= 2 - f(d_1, d_2), \\
f(d_1, d_2) &= f(d_2, d_1), \\
f(d_1 - 1, d_2) &\geq f(d_1, d_2).
\end{aligned}
$$

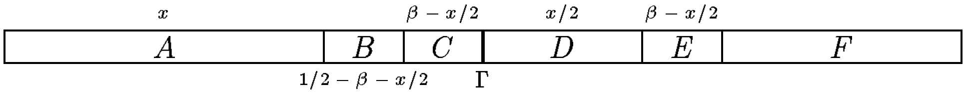
Furthermore, if the number of routing steps must be less than $(1-\beta)\cdot n$, then p_1 must be directed rightwards when $d_1 < \beta\cdot n$ and leftwards when $d_1 > (1-\beta)\cdot n$. For p_2 we can reason analogously. Also, we must have $f(d_1, d_2) \geq 1$, when $d_1 + d_2 < (1+\beta)\cdot n$: Suppose that for some d_1, d_2 with $d_1 + d_2 < (1+\beta)\cdot n$, both packets are routed leftwards. When all PUs had two packets like this, more than $(1-\beta)\cdot n$ packets would move leftwards through P. Using all these relations, we can draw the following picture, giving the possible values of f:



Exploiting the derived properties of f to construct a bad permutation, we find

Theorem 1 *Uniform deterministic fully-local-knowledge* 2-2 *routing requires at least* $41/58\cdot n \simeq 0.71\cdot n$ *steps.*

Proof: Consider the following subdivision of the ring

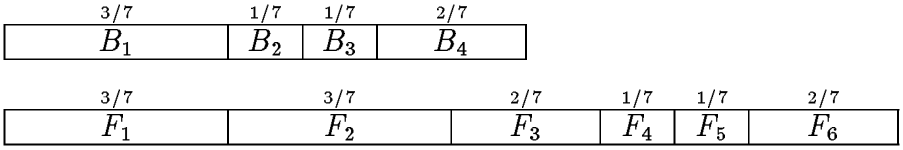


Indicated are the lengths of the sections as fractions of n. F is made as large as necessary, it may partially overlap with A. Γ indicates a connection. We now define a 2-2 distribution as follows: One packet from every PU in A has a PU in D as destination, the other packet is routed to a PU in F such that the sum of their distances, d_1, d_2, satisfies $d_1 + d_2 = (1+\beta)\cdot n$. Both packets from a PU in B have a PU in F as destination. One packet is routed to the first free location in F at distance d_1, the other packet is routed as far as possible but not farther than $(1+\beta)\cdot n - d_1$. The derived rules imply that from A and B at least one packet is routed through Γ. All packets from C are routed through Γ.

Thus, by counting the number of packets that must cross Γ we obtain: $H \geq (1/2 + 3/2\cdot\beta - x/2)\cdot n$. As was to be expected, H becomes larger when x is made smaller. However, x cannot be made too small, because we must be able to give the packets p_1, p_2 from any PU in B destinations in F such that $d_1 + d_2 \leq (1+\beta)\cdot n$. Not all destinations in F are available: two packets from PUs in A already go to every three consecutive PUs in F. Thus, only four packets from PUs in B can have destination in three consecutive PUs of F.

If the destinations of the packets are constructed as indicated above, then it is difficult to determine the minimal value of x. In the next paragraph we show that the packets in B can be given destinations such that $d_1 + d_2 \leq 2\cdot(\#B + \#C + \#D + \#E) + 3/7\cdot\#B$, for all PUs in B. Substituting the lengths of the sections and solving the equation $d_1 + d_2 \leq (1+\beta)\cdot n$, this gives $x \geq (3 + 8\cdot\beta)/31$. Substituting this in $H \geq \min\{1-\beta, 1/2 + 3/2\cdot\beta - x/2\}\cdot n$, we find $H \geq 41/58\cdot n$ for $\beta = 17/58$.

We turn to the construction of the destinations of the packets starting in B. Consider the following subdivisions of B and F:



The numbers indicate fractions of $\#B = (1/2 - \beta - x/2) \cdot n$. From every PU in B_1 one packet goes to every PU of F_1 and one packet to every PU in F_2. From every PU in B_2 one packet goes to every third PU in F_1 and one packet to every third PU in $F_5 \cup F_6$. From every PU in B_3 one packet goes to every third PU in F_2 and one packet to every third PU in $F_3 \cup F_4$. From every PU in B_4 one packet goes to every PU in F_3 and one packet to every PU in $F_4 \cup F_5$. It is easy to check that this just 'fits' and that for all PUs $d_1 + d_2 \leq 2 \cdot (\#B + \#C + \#D + \#E) + 3/7 \cdot \#B$. $\square$

The case $k = 3$. When two direction functions are used, f_0 for PUs with even index and f_1 for PUs with odd index, we can find an interesting result for the case $k = 3$. For $i \in \{0, 1\}$, $\bar{i}$ denotes $(i+1) \bmod 2$. The optimal f_0, f_1 must satisfy

$$f_i(d, d, d) = 3 - f_{\bar{i}}(n - d, n - d, n - d). \tag{1}$$

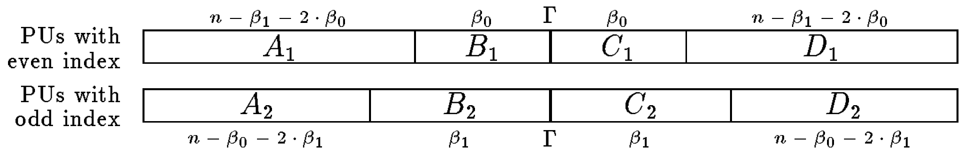
Let β_i, $i = 0, 1$, be the largest numbers such that $f_i(\beta_i, \beta_i, \beta_i) = 3$. With these β_i,

Lemma 10 *If $d < n - \beta_i$, then $f_{\bar{i}}(d, d, d) \geq 1$.*

Proof: We prove the lemma for $i = 0$. Suppose that for certain d $f_1(d, \ldots, d) = 0$. By (1), this is equivalent to $f_0(n - d, d, n - d) = 3$. By definition of β_0, this implies that $n - d \leq \beta_0$ and thus that $d \geq n - \beta_0$. $\square$

Lemma 11 *Uniform deterministic fully-local-knowledge* 3-3 *routing requires at least n steps.*

Proof: Exploiting Lemma 10 we can construct a distribution of packets requiring at least n routing steps. Consider



The packets from the PUs in A_1 are routed to PUs in D_1, the packets from the PUs in B_1 are routed to PUs in C_1. Each PU in A_1 routes at least 1 packet through Γ. From each PUs in B_1 3 packets are routed through Γ. The packets from the PUs with odd index are routed analogously. For this distribution, $H \geq (2 \cdot n - 3 \cdot \beta_0 - 3 \cdot \beta_1)/2 + (3 \cdot \beta_0 + 3 \cdot \beta_1)/2$. $\square$

General k. For general k we use that always an integral number of packets must be directed in both directions. Even and odd k are considered separately.

Lemma 12 *Uniform deterministic fully-local-knowledge k-k routing, k even, requires at least $k/4 \cdot (k + 2)/(k + 1) \cdot (n - 1)$ steps.*

Proof: Let $f : \{0, 1, \ldots, n-1\}^k \rightarrow \{0, 1, \ldots, k\}$ be the direction function. Let β_l, $0 \leq l \leq k$, be the largest numbers such that $f(\beta_l, \ldots, \beta_l) \geq l$ (if there is no such number then $\beta_l = 0$). For H we get

$$H \geq \max\{l \cdot \beta_l, (k - l + 1) \cdot (n - \beta_l - 1)\}.$$

The second bound follows from the number of packets that must be routed over a distance $n - \beta_l - 1$. Equating and solving, shows that in any case $H \geq (k - l + 1)/(k + 1) \cdot l \cdot (n - 1)$. $l = k/2$ gives the stated result. $\square$

For $k = 4$, Lemma 12 gives a bound of $6/5 \cdot (n-1)$ steps. For large k the bound converges to $k \cdot n/4 + n/4$.

For odd k we have the following analogue of Lemma 12:

Lemma 13 *Uniform deterministic fully-local-knowledge k-k routing, k odd, requires at least $k \cdot n/4 + n/4$ steps.*

Proof: let f_0 and $f-1$ be the direction functions for PUs with even and for PUs with odd index, respectively. Let β_{il}, $i = 0, 1$, $0 \leq l \leq k$, be the largest numbers such that $f_i(\beta_{i_1}, \ldots, \beta_{i_l}) \geq l$. For H we get

$$H \geq \max\{l \cdot (\beta_{0_l} + \beta_{1_l})/2, (k - l + 1) \cdot (2 \cdot n - \beta_{0_l} - \beta_{1_l})/2\}.$$

Equating and solving gives $H \geq (k - l + 1)/(k+1) \cdot l \cdot (n-1)$. $l = k/2 + 1/2$ gives the stated result. □

Lemma 13 gives a beautiful generalization of Lemma 11.

3.4 Local Knowledge

In Section 5 we will see that k-k distributions can be routed faster than given by Lemma 13 with a local-knowledge algorithm that starts by gathering some information. However, the information spreads slowly and must be encoded within a packet of standard length. Furthermore, for oblivious algorithms, the information cannot be obtained 'for free' by analysis of the packets that flow by. For them the gathering of information must be performed in a prephase. These considerations give that local-knowledge algorithms will always require some $\Omega(\sqrt{n})$ routing steps more than global knowledge algorithms.

For the case $k = 3$ we can show that sometimes local-knowledge algorithms require more steps than global-knowledge algorithms by a factor larger than one. In the proof we use an interesting joker-zone-like argument.

Lemma 14 *An oblivious local-knowledge algorithm for 3-3 routing requires at least $7/9 \cdot n \simeq 0.78 \cdot n$ routing steps.*

Proof: We will consider two distributions of packets that are the same in a large fraction of the ring but for which the packets should be directed differently. We demand that the routing time is bounded by $(1-x) \cdot n$. Consider

$1/2 - 3/2 \cdot x$	x	$1/2 - 3/2 \cdot x$	x	x
A	B	C	D	E

Γ

where the numbers indicate fractions of n. In the first distribution the packets from $A \cup B$ have to move as a block to $C \cup D$. If the routing time must be less than $(1-x) \cdot n$, then at least $1/3$ of the packets must be routed leftwards and thus through Γ. In the second distribution, the packets from A still have to go to C, but now the packets from E go to D. These packets all have to move through Γ. Some of the packets from A that were directed leftwards may be informed that now they should better move rightwards. As there are permutations that require $3/4 \cdot n$ routing steps with global knowledge, at most $(1 - x - 3/4) \cdot n$ routing steps may be spent on gathering information. Thus, at best the outmost $(1/4 - x) \cdot n$ PUs at both sides of A will not send packets through Γ anymore. The other $x/2 \cdot n$ PUs continue to send one packet each. Hence, at least $3^1\!/\!_2 \cdot x \cdot n$ will go through Γ. This gives the condition $3^1\!/\!_2 \cdot x \leq 1 - x$, which leads to the result. □

3.5 Lower Bounds for Sorting

We show that 1-1 sorting without making copies requires at least $2/3 \cdot n$ steps. 2-2 sorting requires at least $5/7 \cdot n$ steps. As in Section 3.4 we use joker arguments. For these arguments it is essential to consider sorting without copying. In fact, in Section 5.4 we show that by making copies 1-1 sorting can be performed faster than the here proven lower bounds.

Lemma 15 *If copying is not allowed, then 1-1 sorting takes at least $2/3 \cdot n$ steps.*

Proof: We give two packet distributions with help of the following subdivision of the ring:

$n/3$	$n/3$	1	$n/3-1$
A	$2 \cdots 2$	1	$2 \cdots 2$

The values 1 and 2 indicate the keys of the packets in these sections. In distribution $\mathcal{D}_0$ the packets in A have key 0, in distribution $\mathcal{D}_2$ these packets have key 2. Let p be the packet in $P_{2/3 \cdot n}$ with key 1.

In $\mathcal{D}_0$ p must move $n/3$ steps leftwards, in $\mathcal{D}_2$ p must move $n/3$ steps rightwards. If p does not move then it takes at least $n/3$ steps until it knows which of the two distributions it is in. If p moves m steps rightwards, then it may get to know already after $n/3 - m$ steps the distribution, but if this turns out to be $\mathcal{D}_0$, then it must move $n/3 + m$ steps leftwards to reach its destination. If p moves leftwards it needs even one step more. □

Lemma 16 *If copying is not allowed, then 2-2 sorting takes at least* $5/7 \cdot n \simeq 0.71 \cdot n$ *steps.*

Proof: We give two packet distributions with help of the following subdivision of the ring:

$3/7 \cdot n$	$2/7 \cdot n$	$2/7 \cdot n$
A	$2 \cdots 2$	C

In distribution $\mathcal{D}_0$ the packets in A have key 0, in distribution $\mathcal{D}_2$ these packets have key 2. In the second section the packets all have key 2. The packets in C have keys that gradually increase from left to right from 0.1 to 1.9.

In $\mathcal{D}_0$ all packets in C must move $2/7 \cdot n$ steps leftwards, in $\mathcal{D}_2$ they must move $2/7 \cdot n$ steps rightwards. If a packet moves along the wrong path towards its destination then the sorting takes at least $5/7 \cdot n$ steps. Thus, in a imaginary algorithm sorting with less than $5/7 \cdot n$ steps all packets must take the shortest path. However, it takes $2/7 \cdot n$ steps until it is know in $P = P_{5/7 \cdot n}$ which direction is the right one. Suppose that at that moment there are l packets from C to the left of P and r packets to the right. $l + r = 4/7 \cdot n$. Under $\mathcal{D}_0$ it takes at least r steps to finish the sorting, under $\mathcal{D}_2$ at least $l + 2/7 \cdot n$ (use Lemma 2). The best distribution is $l = n/7$, $r = 3/7 \cdot n$, but even then the sorting takes $5/7 \cdot n$ steps. □

For larger k, most notably $k = 3$, we could not show that sorting requires more steps than routing.

4 Routing Algorithms

On the chain the greedy algorithm performs optimal for all values of k (Lemma 11). For rings this is still true for $k = 1$: in this case the packets move without delay to their destination. However, for $k > 1$ the greedy algorithm may take twice as many routing steps as necessary. The worst-case example is the k-k distribution k-$L_{n/2}$ under which all packets residing in P_i must be routed to $P_{(i+n/2) \bmod n}$. The greedy algorithm routes all packets from left to right through $(n/2 - 1, n/2)$, and takes $k \cdot n/2$ routing steps. Routing half of the packets rightwards and the other half leftwards takes exactly $\max\{n/2, k \cdot n/4\}$. However, it is not a good idea to route always half of the packets in either direction: doing this, the permutation k-L_1, under which P_i routes its packets to $P_{(i+1) \bmod n}$, would require $k/2 \cdot (n-1)$ routing steps.

4.1 Fully-Local Knowledge

In oblivious routing algorithms the routing problem is reduced to the problem of directing the packets suitably. In this section we give a simple fully-local-knowledge algorithm which has near-optimal performance. In this algorithm the packets are directed in a weighted way: the farther the packets in a PU have to travel rightwards, the more packets are directed leftwards. This approach can be viewed as a deterministic version of the randomized algorithm in which a packet that has to travel d steps rightwards is directed leftwards with probability d/n.

For the direction all PUs perform the following algorithm:

1. Sort the packets on the distance d_i, $1 \leq i \leq k$, they have to go rightwards. Store them in $a_1, \ldots, a_k$.

2. Let $j = k - round(\sum_{i=1}^{k} d_i/n)$.

3. Direct $a_1, \ldots, a_j$ rightwards; direct $a_{j+1}, \ldots, a_k$ leftwards.

Call this algorithm FLKDIRECT, an acronym for *f*ully-*l*ocal-*k*nowledge *direct*. Notice that FLKDIRECT can be applied for all k. For $k = 1$, it directs all packets towards their destinations, for $k \geq 2$ the algorithm is non-trivial.

By its generality FLKDIRECT is a very attractive algorithm, by its simplicity it can be implemented easily. The analysis is a bit tricky though. Because the following lemma is of central importance, implying the near-optimality of FLKDIRECT, its long proof is rendered completely. The reader may want to skip at least part of it.

Lemma 17 *When a k-k distribution is directed with* FLKDIRECT*, then no connection has to transfer more than $k \cdot n/4 + n/4$ packets.*

Proof: Consider a k-k distribution. Let $a_{i,j}$ be the number of packets that has to go from P_i to P_j. By definition of a k-k distribution, $u_i = \sum_{j=0}^{n-1} a_{i,j} = k$, and $v_j = \sum_{j=0}^{n-1} a_{i,j} = k$, for all $0 \leq i, j \leq n-1$.

We analyze the number H of packets that has to go over $(n-1, 0)$. First we consider $\overline{H}$, the (not necessarily integral) number of packets that would go over $(0, n-1)$ if in step 2 the numbers would not be rounded. $\overline{H}$ also equals the expected number of packets that would go over $(n-1, 0)$ in the above sketched randomized algorithm. The packets coming from P_{n-1} contribute $(n-1) \cdot a_{n-1,0} + (n-2) \cdot a_{n-1,1} + \cdots + 2 \cdot a_{n-1,n-3} + a_{n-1,n-2}$ to $n \cdot \overline{H}$. Summing over all origins we find

$$n \cdot \overline{H} = \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} (i-j) \cdot a_{i,j}/n. \tag{2}$$

Let $-1/2 < \delta_i \leq 1/2$ be the effect of the rounding for H_i, the contribution of P_i to H. Then,

$$H = \overline{H} + \sum_{i=1}^{n-1} \delta_i. \tag{3}$$

Combining (2) and (3), we prove that $H \leq k \cdot n/4 + n/2$. Then we will refine the proof to obtain the stated result.

From the sum in (2) we can split off terms u_i and v_j such that $\sum_{i=1}^{n-1} \sum_{j=0}^{i-1} (i-j) \cdot a_{i,j} \leq \sum_{l=1}^{n/2} l \cdot u_{n/2-1+l} + \sum_{l=1}^{n/2-1} l \cdot v_{n/2+l}$. Once this is shown it follows that $n \cdot \overline{H} \leq k \cdot (\sum_{l=1}^{n/2} l + \sum_{l=0}^{n/2-1} l) = k \cdot n^2/4$. And so, by (3), $H \leq k \cdot n/4 + (n-1) \cdot 1/2$.

The estimate for (2) in terms of u_i and v_j can be proven in a standard way, using induction on n. For a more instructive proof we consider the case $n = 10$ with help of the following matrix A:

9	9	8	7	6	5	4	3	2	1	0
8	8	7	6	5	4	3	2	1	0	0
7	7	6	5	4	3	2	1	0	0	0
6	6	5	4	3	2	1	0	0	0	0
5	5	4	3	2	1	0	0	0	0	0
4	4	3	2	1	0	0	0	0	0	0
3	3	2	1	0	0	0	0	0	0	0
2	2	1	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6	7	8	9

$A_{i,j}$ is the coefficient of $a_{i,j}$ in (2). Subtracting one from all entries in a row or column corresponds to splitting of a term u_i or v_j, respectively. $A_{9,0}$ can be eliminated by taking 5 times u_9 and 4 times v_0. Then $A_{6,0}$ and $A_{7,1}$ are eliminated, and so on. For $n = 10$ this gives $n \cdot \overline{H} \leq (5 \cdot u_9 + 4 \cdot v_0) + (4 \cdot u_8 + 3 \cdot v_1) + (3 \cdot u_7 + 2 \cdot v_2) + (2 \cdot u_6 + v_5) + u_5$. For higher n the process of elimination goes analogously.

In order to show that actually $H \leq k \cdot n/4 + n/4$, we refine our estimates. Consider the matrix A' that is obtained after subtracting 5 from the entries in row 9 of A, 4 from the entries in column 0 of A, etc.:

9	0	0	0	0	0	−1	−2	−3	−4	−5
8	0	0	0	0	0	−1	−2	−3	−4	−4
7	0	0	0	0	0	−1	−2	−3	−3	−3
6	0	0	0	0	0	−1	−2	−2	−2	−2
5	0	0	0	0	0	−1	−1	−1	−1	−1
4	0	0	0	0	0	0	0	0	0	0
3	−1	−1	−1	−1	0	0	0	0	0	0
2	−2	−2	−2	−1	0	0	0	0	0	0
1	−3	−3	−2	−1	0	0	0	0	0	0
0	−4	−3	−2	−1	0	0	0	0	0	0
	0	1	2	3	4	5	6	7	8	9

The generalization of A' for larger n will be clear from the example. An accurate expression for H can be given in terms of the entries $A'_{i,j}$:

$$H = k \cdot n/4 - \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} A'_{i,j} \cdot a_{i,j}/n + \sum_{i=1}^{n-1} \delta_i .$$

A k-k distribution is given by 'placing' k packets in every row and column. Now suppose that we have a placement such that

- $\delta_i = 0$, for all $i < n/2$; $\delta_i = 1/2$, for all $i \geq n/2$.
- $a_{i,j} = 0$, for all $i, j < n/2$; $a_{i,j} = 0$, for all $i, j \geq n/2$.

For such a placement $H = k \cdot n/4 + n/4$. A placement with larger H value might be obtained only by increasing $\sum_{i=1}^{n-1} \delta_i$ beyond $n/4$. The packets can be rearranged such that indeed $\sum_{i=1}^{n-1} \delta_i > n/4$, but the induced increase of $\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} A'_{i,j} \cdot a_{i,j}/n$ is larger. $\Box$

Theorem 2 *A k-k distribution directed by* FLKDIRECT *can be routed in $k \cdot n/4 + n/4$ steps, for all $k \geq 2$.*

Proof: From Lemma 17 we know that no connection has to transfer more than $k \cdot n/4 + n/4$ packets. As no packet has to travel more than n steps, this proves the theorem for $k \geq 3$. Consider for $k = 2$ a PU P holding packets p_1 and p_2, going d_1 and d_2 steps rightwards. Suppose that p_1 is directed rightwards. Then $d_1 \leq d_2$ and $2 - (d_1 + d_2)/n \geq 1/2$. Hence, $2 - 2 \cdot d_1/n \geq 1/2$, and thus $d_1 \leq 3/4 \cdot n$. $\Box$

Lemma 18 *The bound given in Theorem 2 is sharp.*

Proof: For odd k, an example is provided by the k-k distribution k-$L_{n/2}$, under which all packets must be routed $n/2$ positions to the right. Every PU wants to direct $k/2$ packets rightwards. This may be rounded to $k/2 + 1/2$. Indirectly it also follows from Lemma 13.

For even k we first consider the case $k = 4$, and use the following subdivision of the ring:

$n/8$							
A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7

Γ

Suppose that all packets from A_0 are shifted to A_7, all packets from A_1 to A_6, all packets from A_2 to A_5, and all packets from A_3 to A_4. Let $\overline{H}_i$ be the not-rounded number of packets that P_i directs rightwards. Then $\overline{H}_i = 1/2$, for $0 \leq i < n/8$, ..., $\overline{H}_i = 7/2$ for $3/8 \cdot n \leq i < n/2$. After rounding it may happen that from A_i, $i = 0, 1, 2, 3$, $(i + 1) \cdot n/8$ packets are routed rightwards through Γ. Thus, in total $5/4 \cdot n$ packets may pass through Γ. The given example can be generalized easily for larger k. $\Box$

4.2 Large k

In FLKDIRECT, the packets in every PU are first sorted. For large k, the computation time , $\mathcal{O}(k \cdot \log k) \cdot T_c$, may become substantial. One might hope that the following simple algorithm performs well:

Algorithm ONEBYONEDIRECT;
$\delta := 0$;
for all packets p **do**
let d be the distance p has to travel rightwards;
if $\delta + 1 - d/n > 1/2$ **then** direct p rightwards, $\delta := \delta - d/n$
else direct p leftwards, $\delta := \delta + 1 - d/n$.

By the variable δ, $-1/2 < \delta \leq 1/2$, every PU keeps track of its history. Unfortunately, this algorithm performs poorly if all PUs get a sequence of packets with d values $n/2, n-1, 1, n-1, 1, \ldots$: the routing would take almost $k \cdot n/2$ steps.

For large k it is much better to divide the packets in each PU in subsets of n packets and to direct these subsets with FLKROUTE:

Theorem 3 *For arbitrary k, the k-k routing problem can be solved in $k \cdot n/4 \cdot T_r + \mathcal{O}(k \cdot \log n) \cdot T_c$ time, with a fully-local-knowledge algorithm.*

Proof: The computation time becomes $k/n \cdot \mathcal{O}(n \cdot \log n) \cdot T_c = \mathcal{O}(k \cdot \log n) \cdot T_c$. The routing time is bounded by $k/n \cdot (n^2/4 + n/4) \cdot T_r = (k \cdot n/4 + k/4) \cdot T_r = k \cdot n/4 \cdot T_r + \mathcal{O}(k) \cdot T_c$. □

Notice that a local-knowledge algorithm can not perform better: we did not even use all information that is available in the PUs themselves; sorting subsets of n packets gives a good balance between the loss of routing steps and the time required for the direction.

4.3 Dynamic Routing

Because in dynamic routing problems packets may be generated at irregular intervals, and because the routing is going on, fully-local-knowledge algorithms are the first to be considered for dynamic routing. When a packet is generated by a PU it should be sent away as soon as possible. Hence, a direction algorithm for dynamic routing must direct the packets one-by-one, and without knowing how many packets there are to come. The above algorithm ONEBYONEDIRECT would have been a good candidate for dynamic routing.

There is no canonical way to measure the performance of dynamic routing algorithms. To make a choice, we consider how much time it takes an algorithm to route k-k distributions. For the direction this means that a PU holding packets $p_1, \ldots, p_k$, must direct p_i independently of p_j for $j > i$, and independently of k: the future is unknown.

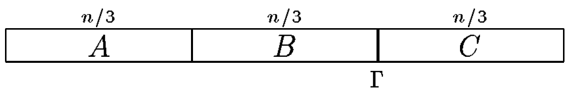
The following algorithm fits perfectly into the model, and has better performance than ONEBYONEDIRECT:

Algorithm DYNAMICDIRECT;
$\delta := 1$;
for all packets p **do**
let d be the distance p has to travel rightwards;
if $d \leq n/3$ or $d \geq 2/3 \cdot n$ **then** direct p towards its destination
else if $\delta = 1$ **then** direct p leftwards **else** direct p rightwards, $\delta := -\delta$.

The packets for which $n/3 < d < 2/3 \cdot n$ are directed alternatingly left- and rightwards. No packet has to travel more than $2/3 \cdot n$ steps.

Theorem 4 *A k-k distribution directed by* DYNAMICDIRECT *can be routed in $k \cdot n/3 + n/3$ steps.*

Proof: We use the standard argument that in some form or other is used in all routing time analyses of this paper. In this case we show that at most $k \cdot n/3$ packets have to go over a connection. Consider the following subdivision of the ring:



Let $a_{A,C}$ be the number of packets going from A to C. $a_{B,C}$ and $a_{B,A}$ are defined analogously. The number of packets moving through Γ equals $H = a_{A,C}/2 + a_{B,C} + a_{B,A}/2 = (a_{A,C} + a_{B,C})/2 + (a_{B,C} + a_{B,A})/2$. Using $a_{A,C} + a_{B,C} \leq k \cdot n/3$ and $a_{B,C} + a_{B,A} \leq k \cdot n/3$, we find $H \leq k \cdot n/3$. By the rounding, every of the $2/3 \cdot n$ PUs that may send packets over Γ can send half a packet more. □

How good is $k \cdot n/3$? Trying examples one feels that this might be optimal. However, we did not succeed in proving any non-trivial lower bound for routing k-k distributions with a dynamic algorithm.

4.4 Local Knowledge

We turn back to routing k-k distributions. The term $n/4$ in Theorem 2 is a result of the rounding. It can be reduced by dividing the ring in subchains consisting of $\sqrt{n}$ adjacent PUs each. Within these subchains the rounding can be performed such that the sum of the numbers of packets that are directed rightwards is at most $1/2$ away from the sum of the numbers of packets the PUs in this subchain want to directed rightwards. Call the modified algorithm LKDIRECT

Theorem 5 *A k-k distribution directed by* LKDIRECT *can be routed in $k \cdot n/4 + \sqrt{n}$ steps, for all $k \geq 4$.*

Proof: The rounding can be performed in $\sqrt{n}/2$ steps. After this the term $n/4$ is reduced to $\sqrt{n}/4$. □

For $k = 2$ and $k = 3$ the above approach does not give useful results: packets may be sent almost around the ring. Let S_i be the (not necessarily integral) number of packets that P_i directs rightwards.

The Case $k = 2$. For $k = 2$, S_i equals the number of packets that have to travel less than $n/3$ steps rightwards + half the number of packets that have to travel between $n/3$ and $2 \cdot n/3$ steps rightwards. This is an alternative formulation of the rule of DYNAMICDIRECT. No packet has to travel more than $2/3 \cdot n$ steps. The rounding of the S_i can be performed again in subchains of $\sqrt{n}$ PUs. We get

Theorem 6 *2-2 routing can be performed in $2/3 \cdot n + \sqrt{n}$ steps, by a local-knowledge algorithm.*

Proof: As in the proof of Theorem 4, we can show that $H \leq 2/3 \cdot n$. By the rounding in subchains the additional term is only $\sqrt{n}$. □

The Case $k = 3$. For $k = 3$, $7/9 \cdot n$ is a lower bound for routing with local knowledge (see Lemma 14). This bound we cannot match, but we can come quite close to it.

Consider P_i. Let a_j be the number of packets in P_i that have to travel between $(j-1)/7 \cdot n$ and $j/7 \cdot n$, $j = 1, 2, \ldots, 7$. Then $S_i = 1/6 \cdot (6 \cdot a_1 + 5 \cdot a_1 + \cdots + a_6)$. The rounding of the S_i is performed in subchain of length $\sqrt{n}$.

Theorem 7 *3-3 routing can be performed in $6/7 \cdot n + \sqrt{n}$ steps, by a local-knowledge algorithm.*

Proof: The proof is analogous to the proof of Theorem 6. Using a subdivision in 7 sections it can be shown that for a k-k distribution $H \leq k \cdot n/(7 \cdot 6) \cdot 12 = 2/7 \cdot k \cdot n$. For $k = 3$ this gives $H \leq 6/7 \cdot n$. □

4.5 Concluding Remarks on Routing

Makedon and Simvonis [10] considered the case that all k packets from a PU have the same destination. This is a special case of ours. Their main result is an algorithm that requires $k \cdot n/4 + 5/2 \cdot n$ routing steps. They conjecture that $k \cdot n/4 + 3/2 \cdot n$ is a lower bound. Our results show that this conjecture does not hold, even the fully-local-knowledge algorithm requires less steps.

Comparing with the lower bound of Lemma 13 we see that fully-local-knowledge routing can be performed near-optimally for all odd k and for $k = 2$. For even $k \neq 2$ the routing time, as expressed by Theorem 2, converges to the lower bound of Lemma 12 for k going to infinity. Actually we believe that also for even k, $k \cdot n/4 + n/4$ is a lower bound for fully-local-knowledge routing, but we were not able to prove this convincingly. For local-knowledge routing our algorithm is near optimal for all $k \neq 3$. This follows by comparing the results of Lemma 9 and Lemma 8 with the results of Theorem 5 and Theorem 6, respectively.

1-1 routing and 2-2 routing with local knowledge take $n/2$ and $2/3 \cdot n + \sqrt{n}$ steps, respectively. Lemma 15 and Lemma 16, give lower bounds for the corresponding *sorting* problems: $2/3 \cdot n$ steps for 1-1 sorting and $5/7 \cdot n$ steps for 2-2 sorting. We see that here sorting takes more steps than routing by a factor larger than one. We do not know other examples of this on mesh-like architectures.

Global knowledge is not worth a lot: Except for $k = 3$, the lower bounds for routing with global knowledge are matched by local-knowledge algorithms.

In [5] two more algorithms for directing the packets were given. In the first algorithm, every PU directs at least $\alpha \cdot k$ packets in either direction. In the other algorithm all packets that have to travel less than $\beta \cdot n$ or more than $(1 - \beta) \cdot n$ steps rightwards are directed towards their destination. A detailed description and thorough analysis of these algorithms can be found in [14]. Though the ideas are natural and interesting, the results of the previous two sections show that FLKDIRECT and LKDIRECT are superior.

5 Sorting Algorithms

After the extensive analysis of the routing problem in Section 4, we now turn to k-k sorting. We start with a randomized algorithm. Then we show how the features of this algorithm can be maintained in a deterministic algorithm. At the end of this section we consider k-k sorting for very large and for the smallest k.

5.1 Randomized Sorting

In [16] it is shown that k-k sorting can be performed with $k \cdot n/4 + o(k \cdot n)$ steps on meshes of arbitrary dimension d for all $k \geq 4 \cdot d$. The algorithm is randomized. Particularly this implies that by a randomized algorithm near-optimal sorting on the ring is possible for all $k \geq 4$.

The central ideas of the algorithm are the following:

1. Route all packets to a randomly selected intermediate destination.

2. Divide the ring in subchains of appropriate size $m = o(n)$. Sort the packets within the subchains.

3. Let $\lfloor r_p \cdot n/(k \cdot m) \rfloor$ be the preliminary destination for a packet p with rank r_p within its subchain.

4. Route the packets to their preliminary destinations.

5. Bring the packets to their final destination by performing transposition-sort steps.

Call this algorithm RANDSORT. Originally the algorithm used an involved construction with presplitters and splitters from which the preliminary destination could be estimated. However, after a complete randomization of the packets there is no need for all this: we can just as well use the packets themselves.

Theorem 8 RANDSORT *performs k-k sorting on a ring in* $\max\{n, k \cdot n/4\} + o(k \cdot n)$ *steps.*

Proof: We indicate the main points. More details can be found in [16]. With Lemma 2 it can be shown that step 1 and step 4 each can be performed in $k \cdot n/8 + o(k \cdot n)$ steps. This is achieved by sending the packets with probability $1/2$ in either direction along the ring. How far a packet can be away from its destination after step 4? Consider a packet p with rank $r + 1$. There are r packets with key smaller than p. These packets are routed with probability m/n to the subchain to which p is randomized independently of each other. Hence, we may use Chernoff bounds (see [3]) to conclude that at most $r \cdot m/n + \mathcal{O}((r \cdot m/n \cdot \log n)^{1/2})$ of them are routed to the subchain of p. Hence, the maximum for the estimate of the rank of p is $r + \mathcal{O}((r \cdot n/m \cdot \log n)^{1/2})$. For $r = \Omega(k \cdot n)$ this is worst. Then we find an 'error' $\mathcal{O}((k \cdot n^2/m \cdot \log n)^{1/2})$. If $m = k^{1/3} \cdot \log^{1/3} n \cdot n^{2/3}$, then the preliminary destination of a packet lies at most a constant number of subchains away from its final destination. □

5.2 Deterministic Sorting

We present a near-optimal *deterministic* sorting algorithm, which can be viewed as a deterministic version of RANDSORT.

The randomization in RANDSORT serves several purposes. It is used for

- Making available locally information from which the preliminary destinations can be determined.
- Redistributing the packets in $k \cdot n/8 + o(k \cdot n)$ steps such that they can be routed from their intermediate destinations to their preliminary destinations with only $k \cdot n/8 + o(k \cdot n)$ steps.

Both effects can be realized by scattering the packets: dividing the ring in subchains, and handing out the packets of the subchains in a fair way over the subchains. This idea goes back on [13]. For efficiently routing the packets to their preliminary destination we could use the deterministical algorithm of [5]. However, the routing of the packets is very regular, and the packets can be routed simply along the shortest path to their destinations. These ideas give us the following algorithm:

1. Divide the ring in $k^{1/3} \cdot n^{1/3}$ subchains consisting of $n^{2/3}/k^{1/3}$ adjacent PUs each. Sort the packets in every subchain.

2. Send the packet with rank $i + j \cdot k^{1/3} \cdot n^{1/3}$, within subchain m, $0 \leq i, j, m \leq k^{1/3} \cdot n^{1/3} - 1$, to PU $\lfloor (m + j \cdot k^{1/3} \cdot n^{1/3})/k \rfloor$ in subchain i.

3. Sort the packets in every subchain.

4. Send the packet with rank i, $0 \leq i \leq k^{2/3} \cdot n^{2/3} - 1$ in subchain m, $0 \leq m \leq k^{1/3} \cdot n^{1/3} - 1$, to PU $\lfloor (i \cdot k^{1/3} \cdot n^{1/3} + m)/k \rfloor$.

5. Sort all pairs of subchains $(0,1), (2,3), \ldots$; sort all pairs of subchains $(1,2), (3,4), \ldots$.

Call this algorithm DETSORT.

We have to settle correctness and time consumption of DETSORT. Clearly, by Lemma 6,

Lemma 19 *Step 1 and step 3 each take less than* $k^{2/3} \cdot n^{2/3}$ *steps.*

An important observation in the analysis of the time consumption of step 2 and step 4 is that from every subchain precisely $k^{1/3} \cdot n^{1/3}$ packets are routed to any subchain.

Lemma 20 *Step 2 and step 4 each take less than* $\max\{n/2, k \cdot n/8\} + k^{2/3} \cdot n^{2/3}$ *steps.*

Proof: The smallest routing time is obtained by routing the packets along the shortest paths to their destinations using the farthest first strategy. In order to facilitate the analysis, we can first rearrange the packets within the subchains such that the packets that have to go farthest stand ahead. Then the departure times of the packets can be fixed such that no conflicts over the use of a connection arise. Packets have to travel at most over a distance $n/2$. The number of packets that has to go over a connection between two subchains is $k^{1/3} \cdot n^{1/3} \cdot \sum_{i=0}^{k^{1/3} \cdot n^{1/3}/2-1} (i + 1/2) = k^{1/3} \cdot n^{1/3} \cdot (k^{1/3} \cdot n^{1/3}/2)^2/2 = k \cdot n/8$. After the rearrangement of the packets the number of packets that has to go over a connection within a subchain is smaller. $\square$

Lemma 21 *Step 5 takes less than* $3 \cdot k^{2/3} \cdot n^{2/3}$ *steps.*

The packets that originated in subchain i that reside in subchain i' after step 2 were precisely $k^{1/3} \cdot n^{1/3}$ apart in subchain i. Hence, after the sorting of step 3, a packet p in subchain i' can find from its rank r_i among the packets that originated in subchain i an estimate of its rank within subchain i. This estimate is accurate up to $k^{1/3} \cdot n^{1/3}$. Thus p can estimate its global rank up to $k^{2/3} \cdot n^{2/3}$, which is precisely the number of packets in a subchain. It is not necessary to know where the packets originated: the total estimate is obtained by multiplying the number of packets in subchain i' that are smaller than p by $k^{1/3} \cdot n^{1/3}$. This shows that step 5 brings all packets to their destinations and that the algorithm is correct. Summing the time consumptions of the steps, we find

Theorem 9 DETSORT *performs k-k sorting on the ring in* $\max\{n, k \cdot n/4\} + \mathcal{O}(k^{2/3} \cdot n^{2/3})$ *steps. Only during* $\mathcal{O}(k^{2/3} \cdot n^{2/3})$ *steps the algorithm runs in a mode in which packets have to be compared, the other steps the algorithm runs in a routing mode.*

We conclude that deterministic near-optimal sorting is possible for all $k \geq 4$.

Until now we neglected the time for the computations within the PUs. Any step involving comparison can be performed in $T_r + \mathcal{O}(\log k) \cdot T_c$, by using priority queues for the k elements a PU holds. Now we can reformulate Theorem 9:

Corollary 1 DETSORT *performs k-k sorting on the ring in* $k \cdot n/4 \ \cdot T_r + \mathcal{O}(k^{2/3} \cdot n^{2/3} \cdot (T_r + \log k \cdot T_c))$ *time, for all* $4 \leq k \leq n^2$.

5.3 Large k

In real-live we may find k which are very large in comparison to n. For example, we want to sort a million entries on a ring consisting of 16 PUs. In this section we consider the cases $k \geq n^2$ and $k > 2^n$.

The Case $k \geq n^2$. If $k \geq n^2$, then the size of the subchains in DETSORT becomes less than 1 PU. This does not make the algorithm incorrect (though the computation time in Corollary 1 must be adapted), but it is easier to replace subchains by PUs in this case. The algorithm becomes

1. Sort the packets in every PU.
2. Send the packets with rank i, $0 \leq i \leq k-1$, to PU $i \bmod n$.
3. Sort the packets in every PU.
4. Send the packets with rank i, $0 \leq i \leq k$, to PU $\lfloor i \cdot n/k \rfloor$.
5. Sort all pairs of PUs $(P_0, P_1), (P_2, P_3), \ldots$; sort all pairs of PUs $(P_1, P_2), (P_3, P_4), \ldots$.

Call this algorithm HIGHKSORT. The correctness and time analysis of HIGHKSORT is analogous to the analysis of DETSORT. We get

Theorem 10 *For $k \geq n^2$, HIGHKSORT performs k-k sorting in $(k \cdot n/4 + \mathcal{O}(k)) \cdot T_r + \mathcal{O}(k \cdot \log k) \cdot T_c$ time.*

The Case $k > 2^n$. If $n = o(\log k)$, then the first term of the time in Theorem 10 becomes negligible: with n PUs HIGHKSORT sorts $k \cdot n$ keys in $\mathcal{O}(k \cdot \log k)$ time. A single RAM would have used $\Omega(k \cdot n \cdot \log(k \cdot n))$ time. Hence, the speed-up is $\Theta(n)$. In the remainder of this section we show that the speed-up is not just $\Theta(n)$, but even $n - o(n)$. The idea is that after the sorting of step 1, we can apply merging in step 3 and step 5.

Suppose that the best RAM algorithm for sorting m keys requires $\alpha \cdot m \cdot \log m \cdot T_c$ time.

Lemma 22 *HIGHKSORT can be implemented to run in $\alpha \cdot k \cdot \log k \cdot T_c + \mathcal{O}(k \cdot n) \cdot T_r$ time.*

Proof: In step 1 the optimal sorting algorithm is applied, taking $\alpha \cdot k \cdot \log k \cdot T_c$ time. The routing of step 2 and step 4 take $\mathcal{O}(k \cdot n) \cdot T_r$ time. In step 2 a PU receives from every PU k/n packets. They were sorted and it is easy to store them again in sorted order. Hence, step 3 can be implemented as merging n sorted sequences of length k/n to one sorted sequence of length k. Because two sorted sequences of length m each can be merged in $\mathcal{O}(m) \cdot T_c$ time, this can be done in $\mathcal{O}(k \cdot \log n) \cdot T_c$ time. Step 5 can be performed in the same time order by first forming sorted sequences within the PUs and then merging these sequences with the neighbors. □

This gives

Theorem 11 *Applying HIGHKSORT on a ring consisting of n PUs for sorting N keys gives a speed-up of $n - \mathcal{O}(n^2/\log N)$.*

Proof: Suppose that the term $\mathcal{O}(k \cdot n) \cdot T_r$ in Lemma 22 is at most $\beta \cdot k \cdot n \cdot T_c$. Then, the performance ratio is at least $\alpha \cdot N \cdot \log N/(\alpha \cdot k \cdot \log k + \beta \cdot k \cdot n) > n/(1 + \beta/\alpha \cdot n/\log N) \simeq n \cdot (1 - \beta/\alpha \cdot n/\log N)$. □

Corollary 2 *For $n = o(\log N)$ the speed-up of HIGHKSORT is $n - o(n)$.*

We have, so to say, a near-optimal speed-up.

5.4 Small k

DETSORT performs almost as good as we could hope for all $k \geq 4$, but for $k < 4$ we would like to find algorithms that require less than n steps. In this section we first give an easy algorithm for $k = 1$ and $k = 2$. Then we give a somewhat more involved algorithm for $k = 3$.

The Cases $k = 1$ and $k = 2$. For $k = 1$ we can use an algorithm that is very similar to DETROUTE. The idea is that it takes $n/2$ steps to spread information about all packets over the ring, as is done in step 2. During these $n/2$ steps, the connections are not fully used: we can send a copy of every packet to each half of the ring. This means that in step 4 the packets have to travel at most $n/4$ steps instead of $n/2$. We give the details:

1. Divide the ring in $n^{1/3}$ subchains consisting of $n^{2/3}$ adjacent PUs each. Sort the packets in every subchain.

2. Send the packet p with rank $i + j \cdot n^{1/3}$, within subchain m, $0 \leq i, j, m \leq n^{1/3} - 1$, to PU $\lfloor (m + j \cdot n^{1/3})/k \rfloor$ in subchain i, and send a copy of p to PU $\lfloor (m + j \cdot n^{1/3})/k \rfloor$ in subchain $(i + n^{1/3}/2) \bmod n^{1/3}$.

3. Sort the packets in every subchain.

4. Let p have rank i, $0 \leq i \leq 2 \cdot n^{2/3} - 1$ in subchain m, $0 \leq m \leq n^{1/3} - 1$. The preliminary destination of p is $r_p = i \cdot n^{1/3}/2 + m/ \bmod (n^{1/3}/2)$. If p is farther than $n/4$ positions away from its preliminary destination, then discard p, else route p to its preliminary destination.

5. Sort all pairs of subchains $(0, 1), (2, 3), \ldots$; sort all pairs of subchains $(1, 2), (3, 4), \ldots$.

Call this algorithm 11SORT

In step 2 each subchain receives $2 \cdot n^{1/3}$ regularly interspaced packets from every subchain. If a packet p is routed in step 2 to some PU P_i, then its copy p' is routed to $P_{(i+n/2) \mathrm{mod} n}$. The packets in the subchains in which p and p' reside are identical. Hence, after step 3 p and p' still reside precisely $n/2$ positions apart, have the same rank, and get in step 4 the same preliminary destination. So, precisely one of them has distance $n/4$ or less to its preliminary destination. The preliminary destination is accurate up to $n^{2/3}/2$ positions. This settles the correctness of 11SORT.

Lemma 23 11SORT *performs* 1-1 *sorting in* $3/4 \cdot n + \mathcal{O}(n^{2/3})$ *steps.*

Proof: We only have to show that step 4 can be performed in $n/4$ steps. This is easy: no packets travels farther than $n/4$ steps, no connection transfers more than $n/8$ packets (every second packet within a distance $n/4$ has to pass through a connection). □

Of course we do not have to limit the algorithm to making one copy of a packet. If the PUs may hold up to c packets, then c copies can be made of every packet. In that case one of the copies is at most $n/(2 \cdot c)$ positions away from its destination in step 4. Thus, we get

Theorem 12 *If the PUs have storage capacity* $c \geq 4$*, then* 1-1 *sorting can be performed in* $n/2 + n/(2 \cdot c) + \mathcal{O}((c \cdot n)^{2/3})$ *steps.*

Proof: In step 2 one instance of each packet is routed rightwards, one instance of it is routed leftwards. Copies are dropped of when the packets reach a destination. The size of the subchains should be taken $n^{2/3}/c^{1/3}$. Then the local operations take $\mathcal{O}((c \cdot n)^{2/3})$ steps. According to Lemma 7 the queue size is bounded to c. □

Theorem 12 shows that for $c = \omega(1)$ (larger order than constant) 1-1 sorting can be performed with $n/2 + o(n)$ steps. This illustrates that one has to be very precise about the model when trying to prove lower bounds for the sorting problem.

With minimal modifications to step 2 and step 4, 11SORT can also be used for 2-2 sorting. We get

Theorem 13 2-2 *sorting can be performed in* $3/4 \cdot n + \mathcal{O}(n^{2/3})$ *steps.*

For $k = 2$ it is not possible to use more than two instances of packets: the time required for step 2 would increase faster than the time for step 4 would decrease. $3/4 \cdot n$ is not bad compared to the lower bound of $2/3 \cdot n$ for 2-2 *routing* given in Lemma 9.

The Case $k = 3$**.** For $k = 3$ we cannot apply a step like step 2 of 11SORT: such a step would take $3/4 \cdot n$ steps. The algorithm is obtained by modifying step 2, step 3 and step 4 of DETSORT.

2'. If we apply step 2, then every connection has to transfer $3/8 \cdot n$ packets in $n/2$ steps. This shows that there is some slack. We cannot send two instance of each packet $n/2$ apart, but a reduction of the routing time of step 4 by $\alpha \cdot n$ can be obtained when for each packet two instances reside $2 \cdot \alpha \cdot n$ apart for some $\alpha > 0$. We clarify the idea by considering a packet p residing in P_0 which would be routed in step 2 of DETSORT to P_i. Now one instance of p goes to $P^{(1)} = P_{i - \alpha \cdot n}$ and one instance to $P^{(2)} = P_{i + \alpha \cdot n}$ (all indices modulo n). If $\alpha \cdot n < i \leq n/2 - \alpha$, then p is routed to $P^{(1)}$. Here a copy of p is made and sent on to $P^{(2)}$. This combining trick is only possible when the shortest path to $P^{(2)}$ goes through $P^{(1)}$ or vice versa.

3′. After step 2′, every subchain has received a complete picture of the packets that virtually reside in a subchain $\alpha \cdot n$ positions to the right and of those $\alpha \cdot n$ positions to the left. These packets are sorted separately, as in step 3.

4′. Now the preliminary destination of each packet is computed exactly as in step 4. The instance of a packet that is farthest from its preliminary destination is discarded, the other instance moves to its preliminary destination along the shortest path.

Call this algorithm DETSORT′. The correctness of DETSORT′ follows from the correctness of DETSORT. The only point to consider is the time required for step 2′ and step 4′ as a function of α. We get

Theorem 14 *Taking* $\alpha = (1 - \sqrt{2/3})/4 \simeq 0.046$*,* DETSORT′ *performs* 3-3 *sort in* $(1 - \alpha) \cdot n + \mathcal{O}(n^{2/3})$ *steps.*

Proof: In step 2′ no packet has to move further than $n/2$ steps. What is the maximum number of packets a connection may have to transfer? Consider a connection Γ which transfers packets from left to right. From the subchain immediately to the left of Γ for a fraction $1/2 + 2 \cdot \alpha$ of the packets an instance is sent through Γ: for $(1/2 + 2 \cdot \alpha) \cdot n$ destinations one of the instances of a packet has to move through Γ. The fraction of the packets for which an instance is sent through Γ decreases linearly with the distance of a subchain to Γ. From the subchain at distance $(1/2 - 2 \cdot \alpha) \cdot n$ from Γ, the fraction equals $4 \cdot \alpha$. From there the fraction decreases linearly to 0 for the subchain at distance $n/2$. Computing the number of packets moving through Γ (accurate up to $\mathcal{O}(n^{2/3})$), gives $3 \cdot n \cdot (1/8 + \alpha - 2 \cdot \alpha^2)$. For $\alpha = (1 - \sqrt{2/3})/4$ this equals $n/2$. In step 4′ packets have to move at most over $(1/2 - \alpha) \cdot n$ steps, and connections surely do not have to transfer more than this many packets. □

5.5 Concluding Remarks on Sorting

We have given a k-k sorting algorithm which works without making copies. The algorithm is near-optimal for all $k \geq 4$. For $k < 4$, the sorting time can be reduced if it is allowed to make copies. For $k = 1$, near optimal performance can be reached when the PUs can hold $\omega(1)$ packets. For $k = 2$ or 3, one copy is the most that can be used effectively. For very large k, we gave an algorithm with speed-up $n - o(n)$.

6 Conclusion

We considered k-k routing and k-k sorting on rings, and found deterministical algorithms with near-optimal performance, $k \cdot n/4 + o(k \cdot n)$, for all $k \geq 4$. This means that k-k sorting is not substantially harder than k-k routing. Yet the algorithms are completely different and there is also a small difference in performance. In this respect the ring distinguishes itself from higher dimensional meshes and tori: there the best k-k routing algorithm known is actually a k-k sorting algorithm [6]. For $k = 1$, 2 and 3, we gave alternative algorithms taking less than n steps, coming close to derived lower bounds. An algorithm that can be applied to dynamic routing problems routes k-k distributions in $k \cdot n/3 + n/3$ steps.

Future research might show that the lower bounds for *uniform* fully-local-knowledge algorithms generally hold for fully-local-knowledge algorithms. Secondly, it is interesting to consider whether with global knowledge 3-3 routing can be performed in $3/4 \cdot n$ steps: this would show that sometimes global knowledge *can* speed-up routing on rings. Of greater practical importance is a more thorough analysis of the dynamic routing problem. It is not certain that the given algorithm is optimal: non-trivial lower bounds fail. For sorting without copies it is desirable to bring lower- and upper bounds closer together for $k < 4$. For applications of the ideas of this paper to other networks, we suggest that it might be possible to reduce the k-k routing time on meshes and tori below the sorting time. Finally we hope that the observation that the sorting algorithm requires only *one* local sorting operation, can be used to extend the class of networks to which the underlying sorting algorithm can be applied.

References

[1] Chlebus, B.S., M. Kaufmann, J.F. Sibeyn, 'Deterministic Permutation Routing on Meshes,' *Proc. 5th Symp. on Parallel and Distributed Proc.*, IEEE, 1993, to appear.

[2] Kaklamanis, C., D. Krizanc, 'Optimal Sorting on Mesh-Connected Processor Arrays,' *Proc. 4th Symposium on Parallel Algorithms and Architectures*, pp. 50-59, ACM, 1992.

[3] Kaufmann, M., S. Rajasekaran, J.F. Sibeyn, 'Matching the Bisection Bound for Routing and Sorting on the Mesh,' *Proc. 4th Symposium on Parallel Algorithms and Architectures*, pp. 31-40, ACM, 1992.

[4] Kaufmann, M., J.F. Sibeyn, 'Optimal Multi-Packet Routing on the Torus', *Proc. 3rd Scandinavian Workshop on Algorithm Theory*, pp. 118-129, 1992.

[5] Kaufmann, M., J.F. Sibeyn, 'Deterministic Routing on Circular Arrays,' *Proc. 4th Symposium on Parallel and Distributed Processing*, IEEE, pp. 376-383, 1992.

[6] Kaufmann, M., J.F. Sibeyn, T. Suel, 'Derandomizing Routing and Sorting Algorithms for Meshes,' *submitted to SODA '94*, 1993.

[7] Kunde, M., 'Concentrated Regular Data Streams on Grids: Sorting and Routing Near to the Bisection Bound', *Proc 31th Symposium on Foundations of Computer Science*, pp. 141-150, IEEE, 1991.

[8] Kunde, M., 'Block Gossiping on Grids and Tori: Deterministic Sorting and Routing Match the Bisection Bound,' *Proc. European Symp. on Algorithms*, 1993, to appear.

[9] Leighton, T., F. Makedon, Y. Tollis, 'A $2n-2$ Step Algorithm for Routing in an $n \times n$ Array with Constant Size Queues,' *Proc. Symposium on Parallel Algorithms and Architectures*, pp. 328-335, ACM, 1989.

[10] Makedon, F., A. Simvonis, 'Multipacket Routing on Rings,' *Proc. 1st Intern. ACPC Conference*, LNCS, 591, pp. 226-237, 1991.

[11] Rajasekaran, S., R. Overholt, 'Constant Queue Routing on a Mesh,' *Journal of Parallel and Distributed Computing*, pp. 160-166, June 1992.

[12] Rajasekaran, S., M. Raghavachari, 'Optimal Randomized Algorithms for Multipacket and Cut Through Routing on the Mesh', *Proc. 3rd Symposium on Parallel and Distributed Processing*, IEEE, 1991.

[13] Schnorr, C.P., A. Shamir, 'An Optimal Sorting Algorithm for Mesh Connected Computers,' *Proc. 18th Symposium on Theory of Computing*, pp. 255-263, ACM, 1986.

[14] Sibeyn, J.F., *Algorithms for Routing on Meshes*, Ph. D. Thesis, Universiteit Utrecht, Utrecht, 1992.

[15] Sibeyn, J.F., 'Deterministic Sorting on Circular Arrays,' *Proc. CSN'93*, SION, Amsterdam, 1993, to appear.

[16] Sibeyn, J.F., M. Kaufmann, 'Randomized k-k Sorting on Meshes,' draft, 1992.