Dartmouth College
Dartmouth Digital Commons

Computer Science Technical Reports | Computer Science

12-2-1991

Optimal Algorithms for Multipacket Routing Problems on Rings

Fillia Makedon
Dartmouth College

Antonios Symvonis
University of Sydney

Follow this and additional works at: https://digitalcommons.dartmouth.edu/cs_tr

 Part of the Computer Sciences Commons

Dartmouth Digital Commons Citation

Makedon, Fillia and Symvonis, Antonios, "Optimal Algorithms for Multipacket Routing Problems on Rings" (1991). Computer Science Technical Report PCS-TR91-174. https://digitalcommons.dartmouth.edu/cs_tr/66

This Technical Report is brought to you for free and open access by the Computer Science at Dartmouth Digital Commons. It has been accepted for inclusion in Computer Science Technical Reports by an authorized administrator of Dartmouth Digital Commons. For more information, please contact dartmouthdigitalcommons@groups.dartmouth.edu.

OPTIMAL ALGORITHMS FOR MULTIPACKET ROUTING PROBLEMS ON RINGS

Fillia Makedon
Antonios Symvonis

Technical Report PCS-TR91-174

Dec 1991
Submitted to
the Journal of
Parallel & Distributed
Computing

Optimal Algorithms for Multipacket Routing Problems on Rings

Fillia Makedon* Antonios Symvonis†

December 2, 1991

Abstract

We study multipacket routing problems. We divide the multipacket routing problem into two classes, namely, distance limited and bisection limited routing problems. Then, we concentrate on rings of processors. We prove a new lower bound of $2n/3$ routing steps for the case of distance limited routing problems. We also give an algorithm that tightens this lower bound. For bisection limited problems the lower bound is $kn/4, k > 2$, where k is the number of packets per processor. The trivial algorithm needs in the worst case $k\lfloor n/2 \rfloor$ steps to terminate. An algorithm that completes the routing in $kn/4 + 2.5n$ routing steps is given. We define the class of pure routing algorithms and we demonstrate that new lower bounds hold if the routing is to be done by an algorithm in this class.

Basser Dept. of Computer Science TR424-1991

*Department of Mathematics and Computer Science, Dartmouth College, Hanover, NH03755
†Basser Department of Computer Science, University of Sydney, Sydney, N.S.W. 2006, Australia

1 Introduction

A great deal of work has been devoted to the study of the packet routing problem [1]-[16]. This is because the packet routing problem is closely related to parallel computation. Through the routing of messages (packets) we are able to emulate shared memory [14]. More generally, for a parallel computer to be computationally effective, it must be able to route messages from their origin processors to their destination processors fast and with small, preferably constant size queues. These queues are created while two or more packets are waiting to cross the same communication channel.

In this paper, we consider two types of packet routing problems, namely, *distance limited* and *bisection limited* routing problems, a distinction which is based on the number of packets each processor has to route. We concentrate on permutation problems on a ring of processors. The reason for doing so, is because, before trying to attack the problem for the more general case of r-dimensional meshes and tori, we must have a full understanding of the problem in its simplest form. We prove a new lower bound for distance limited problems on rings and we give an algorithm that matches the lower bound. For the case of bisection limited routing problems, we present an algorithm that routes the packets in near optimal time.

Surprisingly, not too much attention has be given to the packet routing problem on rings. According to our knowledge, it is the first time that an in depth investigation is attempted. The majority of the previous work on packet routing problems on multidimentional meshes focuses in the two dimensional case. Probabilistic [2, 10, 13, 15, 16] as well as deterministic [3, 4, 5, 6, 8, 9, 10, 12, 15] algorithms have been proposed. All of these algorithms try to minimize the number of routing steps required to complete the routing and the size of the extra memory used for queueing purposes. The only work on r-dimensional meshes, $r > 2$, is by Kunde [4, 5].

A *ring of n processors* is defined to be a graph $G = (V, E)$ where, $V = \{i | i = 0, 1, 2, ..., n-1\}$ and $E = \{(i, i-1 \bmod n) | i = 0, 1, \ldots, n-1\}$. Set V represents the processors while set E represents the communication links between them. At any step,

each processor can communicate with both of its neighbors.

We define the *distance along the shortest path* between processors $P = i$ and $Q = j$, denoted $D_s(P, Q)$, to be the minimum number of links (edges) that a packet has to traverse starting from processor P and destined for processor Q. Obviously, $D_s(P, Q) = D_s(Q, P)$. Formally, for processors $P = i$ and $Q = j$ we define $D_s(P, Q) = \min\{|j - i|, n - |j - i|\}$. The notation $[xy]_{dir}$ will be used to denote the distance from processor x to processor y in direction *dir*. *dir* can be clockwise (cw) or counter-clockwise (ccw). Observe that $[xy]_{cw} = n - [yx]_{cw} = n - [xy]_{ccw}$. For clarity reasons, in the rest of the paper processor i will be denoted by P_i. We also assume that the ring is lied down in such a way that the processor numbers increase in the clockwise direction.

In a *permutation routing problem* each processor has one packet to transmit to any other processor. At the end, each processor receives exactly one packet. In the *multipacket permutation problem* each processor has k packets all of which are destined for the same processor. At the end, each processor receives exactly k packets. Formally, a multipacket permutation problem R on a ring can be defined as a triple $< n, k, F >$ where n is the number of processors on the ring, k is the number of packets per processor, and F is a function $F_.: \{0, 1, \ldots, n-1\} \rightarrow \{0, 1, \ldots, n-1\}$ such that F defines a permutation. The multipacket permutation problem arises when a single packet in the permutation routing problem consists of k flits. Some work has already been done on square meshes for this case: Symvonis and Makedon [10, 15] treated the k flits as an unbreakable "snake", while Kunde and Tensi [6] routed the flits of a packet independently. More recently, Rajasekaran and Raghavachari [13] presented randomized algorithms using both approaches and Kunde [5] derived a new deterministic algorithm for the case where the packets are routed independently. Up to now no work has been done on rings.

The remainder of the paper is organized into sections as follows. In Section 2, we define the classes of distance limited and bisection limited routing problems. In Section 3, we concentrate on distance limited problems on rings of n processors. We present a lower bound of $2n/3$ steps, and we give an algorithm that matches that bound. In Section 4, we investigate the bisection limited problem on a ring of n processors. The known lower

bound for this problem is $kn/4$ routing steps. We present an algorithm that routes any problem in at most $kn/4 + 2.5n$ routing steps. In Section 5, we define the class of pure routing algorithms and we demonstrate that new lower bounds hold if the routing is to be done by an algorithm in this class. Finally, in Section 6, we discuss further work that has to be done in this area.

2 Two Types of Routing Problems

We obtain lower bounds on the number of steps required to solve a routing problem using two different arguments. The first one is a lower bound based on the maximum distance a packet has to travel (*distance bound*). The second one is based on the *bisection bound* of the network used. Then, the lower bound is max(*distance bound*, *bisection bound*). For the case of r-dimensional meshes of side-length n, the distance bound is $r(n-1)$ and the bisection bound is $nk/2$, where k is the number of packets each processor holds. Thus, the lower bound on the number of steps required to solve the multipacket permutation routing problem on the r-dimensional mesh is $\max\{r(n-1), nk/2\}$. Similarly, for the r-dimensional torus (an r-dimensional mesh with wrap-around connections), the lower bound is $\max\{r(n-1)/2, nk/4\}$.

It is clear from the above that we can divide the routing problems into two categories: the *bisection limited problems* and the *distance limited problems*. A problem is bisection-limited if the bisection lower bound is greater than the distance lower bound. Otherwise, we say that the problem is distance limited. For r-dimensional meshes and tori, we obtain that a problem is distance limited if the number of packets per processor is $k \leq 2r$. Otherwise it is bisection-limited.

It should be pointed out that the division of the routing problems into the two categories is based on worst case scenarios. Thus, there are problems that, according to the above distinction, are bisection limited, ($k > 2$ for rings), and still can be solved in less time than that indicated by the distance limit. One such trivial example is when all processors have packets destined for the processors immediately after them in the clockwise

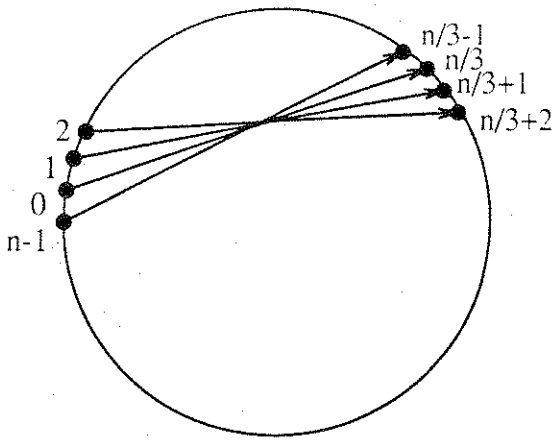


Figure 1: The instance of the permutation routing problem that requires $2n/3$ steps.

direction. Obviously, this routing problem can be solved in k steps.

In the rest of the paper we will concentrate on rings of processors. For a ring ($r = 1$) the problem is distance limited if $k \leq 2$ and bisection limited otherwise.

3 Distance Limited Routing Problem on a Ring

In what follows, we demonstrate a better lower bound for the case of distance limited routing problems on rings. The channel utilization of the network is taken into consideration in order to obtain the new lower bound. Then, we give an algorithm that matches the lower bound.

3.1 Lower Bound on a Ring of Processors

Let us assume that we have a ring of n processors, n is a multiple of 3, and we want to route a multipacket permutation on it. Each processor has two packets that will be routed independently. Consider the following situation: Initially, P_i contains 2 packets destined for $P_{(i+\frac{n}{3}) \bmod n}$, $0 \leq i < n$. (Figure 1).

Hence, in the counter-clockwise direction, each packet has to travel distance $2n/3$. If some packet decides to move in the counter-clockwise direction, then at least $2n/3$ steps are required, since the distance between origin and destination is exactly $2n/3$ in that direction. So, if we want to achieve a better routing time, we have to send all packets

in the clockwise direction. In this case, each of a total of $2n$ packets will travel for $n/3$ steps. Then, the total movement (number of wire crossings) is $2n^2/3$. Since all packets are moving in the same direction, only n communication links are used. Hence, this movement requires at least $2n/3$ routing steps to be accomplished. Thus, we have:

Theorem 1 *There is a distance limited permutation routing problem on a ring of n processors that requires $2n/3$ routing steps for its solution.*

3.2 An Algorithm that Tightens the Lower Bound

An algorithm that tightens the lower bound given in Theorem 1 is the following:

Algorithm *Route_on_a_Ring_1*

At step 1 Each processor determines the minimum distance its packets have to travel, say s, where $0 \leq s \leq \frac{n}{2}$. It also determines the direction in which the packets have to travel so that their distance to the destination is s. Let this direction be denoted by K (cw or ccw).

- If $s \leq \lceil \frac{n}{3} \rceil$, then both packets are send in direction K.
- If $\lceil \frac{n}{3} \rceil < s \leq \frac{n}{2}$, then the processor sends one packet in the cw direction and one in the ccw direction.

at step $t, t > 1$ Each processor transmits a packet toward its destination, if it has one.

A processor never changes the direction of a packet.

The packet that has to go further in a given direction has higher priority.

Lemma 1 *If Algorithm Route_on_a_Ring_1 is used for the routing of a distance limited permutation routing problem then, at any time t, any processor has at most 2, packets that want to move in the same direction.*

Proof The Lemma is obvious, since: i) each processor transmits a packet toward a given direction, if it has one, ii) at any step, it can receive at most one packet that must be

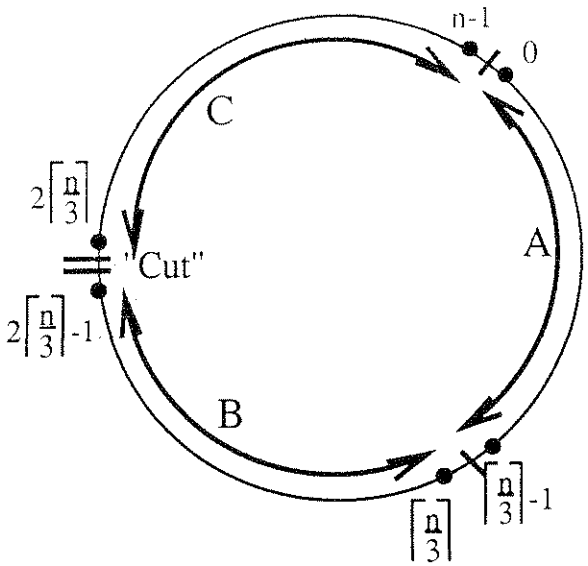


Figure 2: The ring is divided into three sections that are used in the proof of Lemma 2.

sent in that direction, and, iii) the initial load of each processor is at most 2 packets that want to travel in the same direction. ∎

Definition A *cut* c_i is defined to be the edge that connects $P_{(i-1) \bmod n}$ and P_i.

Lemma 2 *If Algorithm Route_on_a_Ring_1 is used for the routing of a distance limited permutation routing problem, then there are at most* $2\lceil n/3 \rceil$ *packets that want to cross any cut in the same direction.*

Proof Without loss of generality, we consider cut $c_{2\lceil \frac{n}{3} \rceil}$ and we examine packets moving only in the clockwise direction. The proof for the counter-clockwise movement is symmetric. Let $P_0, ..., P_{\lceil \frac{n}{3} \rceil - 1}$ constitute segment A of the ring, and $P_{\lceil \frac{n}{3} \rceil}, ..., P_{2\lceil \frac{n}{3} \rceil - 1}$ constitute segment B of the ring (Figure 2).

Initially, each processor in segment A has at most 1 packet that wants to move in the clockwise direction and also wants to cross the cut. Assume that the number of packets in segment A that want to cross the cut is m , $0 \leq m \leq \lceil n/3 \rceil$. Then, these packets must be destined for segment C, where $P_{2\lceil \frac{n}{3} \rceil}, ..., P_{n-1}$ constitute segment C. Since we are examining permutation routing, for any packet in segment A that wants to cross the cut in the clockwise direction, there must exist a unique destination in segment C. This means that, there exist m positions in segment C, none of which can be the destination

of a packet initially in segment B. Thus, in segment B, there must exist at most $\lceil \frac{n}{3} \rceil - m$ processors that have packets destined for segment C. In the worst case, all of these processors will send 2 packets toward the cut. The remaining m processors in segment B will send all together at most m packet toward the cut. So, the total number of packets which are initially in segment B and want to cross the cut is at most $2(\lceil n/3 \rceil - m) + m$. This implies that the total number of packets that want to cross the cut in the clockwise direction is at most $(2\lceil n/3 \rceil - m) + m = 2\lceil n/3 \rceil$. ∎

Theorem 2 *Using Algorithm Route_on_a_Ring_1, a distance limited permutation routing problem on a ring of n processors can be solved in $2\lceil n/3 \rceil$ steps.*

Proof In order to prove this, we need to show that all packets that want to cross any cut in a given direction, will do so after $2\lceil n/3 \rceil$ steps. Without loss of generality, let us consider cut $P_{2\lceil n/3 \rceil + 1}$ (the edge immediately after $P_{2\lceil \frac{n}{3} \rceil}$ in the clockwise direction). We simulate the routing process as follows: Assume that we have two tapes, tape A and B. Tape A has $2\lceil n/3 \rceil$ cells and can move to the right. Tape B has $\lceil n/3 \rceil$ cells, is not allowed to move, and is placed, initially, on top of the $\lceil n/3 \rceil$ rightmost cells of tape A. Each cell of a tape can hold at most one *pebble*. A pebble represents a packet that wants to cross the cut in the clockwise direction. If a pebble is placed on a cell of tape B, then a pebble must also exist on the underlying cell of tape A. We now observe that, initially, we have the following situation: If μ pebbles are on tape B, then at least μ cells in the $\lceil n/3 \rceil$ leftmost positions of tape A are empty. We associate the i^{th} pebble of tape B, $0 \leq i < 2\lceil n/3 \rceil$, where we count from left to right, with the i^{th} empty cell (*hole*) of tape A, where we count from right to left (Figure 3). This is a 1-1 relation.

Now, we start moving tape A to the right. In the worst case, each pebble in tape B will drop into its corresponding hole in tape A. Since the whole tape A will cross the cut after exactly $2\lceil n/3 \rceil$ steps, all pebbles cross the cut in at most $2\lceil n/3 \rceil$ steps. Thus, all packets that want to cross the cut can do so in $2\lceil n/3 \rceil$ steps. ∎

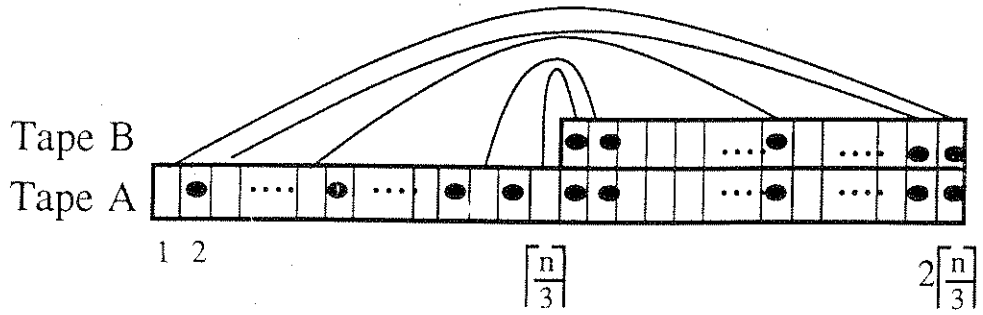


Figure 3: The simulation of the routing by two tapes as used in the proof of Theorem 2.

4 Bisection Limited Routing Problem on a Ring

In this section, we consider bisection limited problems on rings of n processors. Recall, from Section 2, that a problem is bisection limited for rings when k, the number of packets per processor, is greater than 2. The lower bound based on the bisection of the ring is $kn/4$. The trivial greedy algorithm that routes each packet along the shortest path to its destination takes, in the worst case, at least $k\lfloor n/2 \rfloor$ steps. To see that, consider the situation where the packets at P_i are destined for $P_{(i+\lfloor \frac{n}{2} \rfloor) \bmod n}$. If we consider any cut, then the packets from all the $\lfloor n/2 \rfloor$ processors behind it in the direction that the routing takes place want to cross it. There are $k\lfloor n/2 \rfloor$ such packets and they need at least $k\lfloor n/2 \rfloor$ steps to do so.

In the following section we describe an algorithm that completes the routing in the worst case after $kn/4 + 2.5n$ routing steps. In the special case where n and k are even the algorithm takes at most $kn/4 + 1.5n$ routing steps.

4.1 The Algorithm

Before we proceed with the description of the algorithm, we need to give some definitions for certain variables we use: Let S_i denote the distance that the packets initially located at P_i have to travel along the shortest path to their destination. S_i can be written as $S_i = \lambda_i n$, $0 \leq \lambda_i \leq 1/2$, where λ_i is a coefficient used in our algorithm. Our algorithm routes a fraction of the packets which are initially at P_i along the shortest path to their destination, and routes the remaining packets along the longest path. The number of

packets that are routed along the shortest path and are originated at P_i is denoted by σ_i.

Algorithm *Route_on_a_Ring_2*

At step 1 $P_i, 0 \leq i < n$, determines the number of packets σ_i it will send along the shortest path using the following rule:

$\sigma_i = k - \lfloor \lambda_i k \rfloor$

It then adds σ_i out of the k packets to a queue associated with the link on the shortest path, and the remaining packets to the queue associated with the other link.

The packets at the front of the queues are transmitted.

at step $t, t > 1$ Each processor transmits a packet toward its destination, if it has one.

A processor never changes the direction of a packet.

The packets are transmitted using a FIFO policy .

Our efforts to analyze Algorithm *Route_on_a_Ring_2* in a way similar to the analysis of Algorithm *Route_on_a_Ring_1* were not successful. In particular, we were not able to introduce in a proof of that kind the fact that the routing problem is a permutation. So, we proceed with a totally different approach. Again, we consider the number of packets that cross any cut in any direction. But now, we prove that the given routing problem is "easier" to be solved than a special kind of routing problem for which we can make statements regarding its complexity. An upper bound obtained for this special problem will clearly be an upper bound on the initial (and "easier") routing problem. It is the first time that this method for proving upper bounds on routing algorithms is applied in such an extend. A first simple proof of that type was introduced by Symvonis and Makedon in [10, 15].

Definition A *half-ring* of a ring consisting of n processors, n is even, is any section of the ring consisting of exactly $n/2$ consecutive processors.

Note that a half-ring is well defined only on rings that consist of an even number of processor. It is undefined when there is an odd number of processors on the ring.

Definition Two half-rings of the same ring of processors are said to be *disjoint* if they share no common processor.

Obviously a ring of n processors, n is even, is made up by two disjoint half-rings and there n different decompositions of a ring into two disjoint half-rings.

Definition A multipacket routing problem on a ring of n processors, n is even, is said to be *symmetric with respect to cut* c_i if all the packets at the half-ring composed by $P_i, \ldots, P_{(i+\frac{n}{2}-1) \bmod n}$ are destined for the half-ring composed by $P_{(i+\frac{n}{2}) \bmod n}, \ldots, P_{(i-1) \bmod n}$ and vice versa. If a multipacket routing problem is not symmetric with respect to cut c_i then it is called *asymmetric* (with respect to cut c_i).

Again, a symmetric (asymmetric) multipacket routing problem with respect to cut c_i is well defined only for rings consisting of an even number of processors.

Lemma 3 *Assume a ring of n processors, n is even, any cut c_i, any direction, and an asymmetric multipacket permutation routing problem with respect to cut c_i of initial load of k packets per processor that is to be routed. Also assume that Algorithm Route_on_a_Ring_2 is used for the routing. Then, there exists a symmetric multipacket permutation problem with respect to cut c_i of initial load of $k+1$ packets per processor such that, if it is routed using Algorithm Route_on_a_Ring_2 and the extra packet is sent towards the cut in the direction under consideration, it will send at least as many packets to cross the cut in the given direction as the initial asymmetric routing problem. (Informally we can say that the new routing problem is "harder".)*

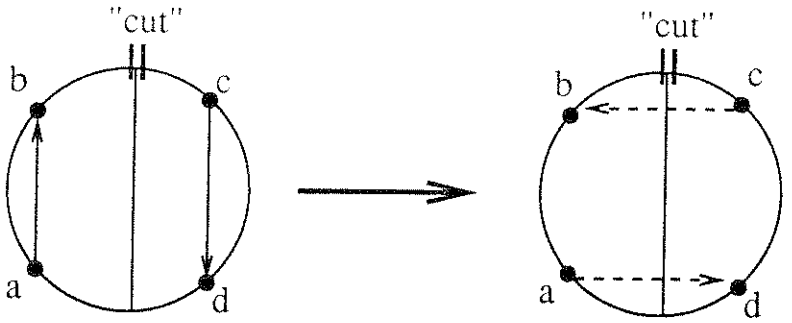
Proof Without loss of generality, we concentrate in the number of packets that cross the cut in the clockwise direction. A diameter that passes through the cut divides the ring into two disjoint half-rings. Half-ring A consists of $P_i, \ldots, P_{(i+\frac{n}{2}-1) \bmod n}$ and half-ring B consists of $P_{(i+\frac{n}{2}) \bmod n}, \ldots, P_{(i-1) \bmod n}$. We will show how to transform any asymmetric multipacket routing problem (with respect to cut c_i) of initial load of k packets to a symmetric problem (with respect to cut c_i) of initial load of $k+1$ packets. Furthermore,

the number of packets that cross the cut in the new routing problem is greater or equal with the number of packets that cross the cut if the original problem is routed. (Both problems are routed by Algorithm *Route_on_a_Ring_2*.)

First observe that if there is a processor in half-ring A that has packets destined for another processor in half-ring A, then, there exist a processor in half-ring B that has packets destined for a processor in half-ring B. The above observation follows from the pigeonhole principle.

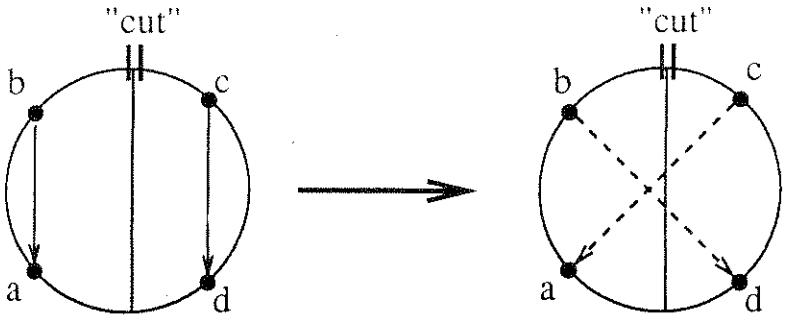
Our transformation consists by picking two processors, one in each half-ring, that have packets destined for the half-ring they belong. Then, the destinations will be switched. By performing the above transformation for at most $n/2$ times, we will get a symmetric multipacket routing problem (with respect to cut c_i). Now it remains to prove that if we load each processor at the new problem with one additional packet which is to be routed towards the cut, the new problem sends at least as many packets to cross the cut as the initial one (and thus, it is "harder"). We distinguish four cases.

Case 1



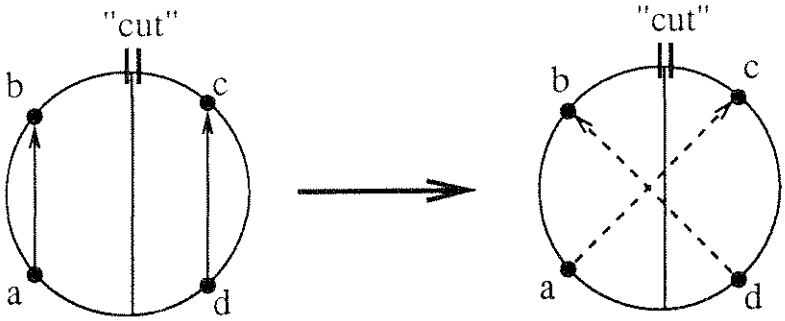
The packets at processors a and c are destined for processors b and d, respectively. After the switch of the destinations, the packets at processor a are destined for processor d and the packets at processor c are destined for processor b. Observe that before the switch no packet wants to cross the cut in the clockwise direction. We remind the reader that in this case as well as in the rest of this section, the routing is done by Algorithm *Route_on_a_Ring_2*. After the switch, a portion of the packets located at processor a might cross the cut in the clockwise direction. So, the new problem sends at least as many packets to cross the cut as the original one.

Case 2



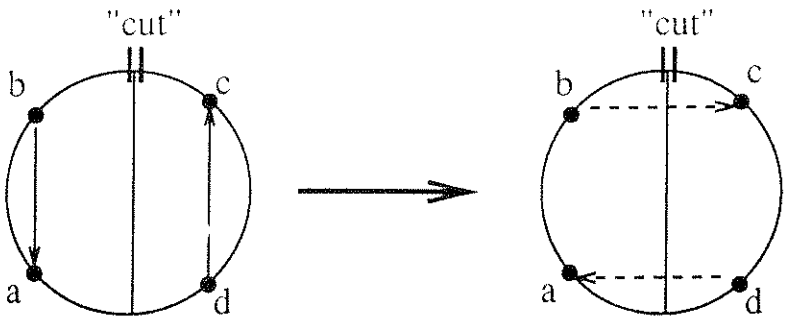
The packets at processors b and c are destined for processors a and d, respectively. After the switch of the destinations, the packets at processor b are destined for processor d and the packets at processor c are destined for processor a. Observe that before and after the switch only packets that are initially located at processor b will cross the cut in the clockwise direction. After the switch, at least the same number of packets will cross the cut in the clockwise direction since the distance the packets have to travel in that direction is reduced.

Case 3



The packets at processors a and d are destined for processors b and c, respectively. After the switch of the destinations, the packets at processor a are destined for processor c and the packets at processor d are destined for processor b. Before the switch only packets initially located at processor d will cross the cut in the clockwise direction while, after the switch, only packets located at processor a will do so. But the number of packets that cross the cut now, is greater or equal than before. This is because the distance the packets which cross the cut have to travel is reduced.

Case 4



The packets at processors b and d are destined for processors a and c, respectively. After the switch of the destinations, the packets at processor b are destined for processor c and the packets at processor d are destined for processor a. This is the most interesting case. Before the switch packets from both origins want to cross the cut while, after the switch, only packets initially located at processor b want to do so. Before the switch, processor b sends at most $(k/n)[ab]_{cw}$ packets towards the cut. Processor d sends at most $(k/n)[cd]_{cw}$ packets towards the cut. Thus, before the switch, at most $(k/n)([ab]_{cw} + [cd]_{cw})$ packets cross the cut in the clockwise direction. Consider now the number of packets (originated at processor b) that want to cross the cut after the switch. We have to distinguish two cases. In the first case the packets that cross the cut are routed along a shortest path. Then at least

$$k - \lfloor \frac{[bc]_{cw}}{n} k \rfloor \geq k - \frac{[bc]_{cw}}{n} k - 1 = \frac{k}{n}(n - [bc]_{cw}) - 1 = \frac{k}{n}[cb]_{cw} - 1$$

packets will cross the cut. In the case where the packets are routed along the longest path, the number of packets that cross the cut is at least $(k/n)[cb]_{cw}$. So, in any case, after the switch at least $(k/n)[cb]_{cw} - 1$ packets will cross the cut in the clockwise direction. But $[cb]_{cw} \geq [ab]_{cw} + [cd]_{cw}$. Thus, if we load processor b with one extra packet and we force it to cross the cut in the clockwise direction, the new problem sends at least as many packets to cross the cut as the original one.

The observation that all of the transformations needed to convert an asymmetric multipacket problem with respect to some cut c_i into a symmetric one with respect to the same cut might be of that described in case 4 proves the lemma. ∎

Lemma 4 *Assume a ring of n processors, n is even, any cut c_i, and any direction. Also assume that a symmetric multipacket permutation routing problem with respect to cut c_i of initial load of k packets per processor is to be routed. If Algorithm Route_on_a_Ring_2 is used for the routing, then the number of packets that cross the cut in any direction is at most equal to the number of packets that cross the cut if a new symmetric multipacket routing problem with respect to cut c_i of initial load of $k+2$ packets per processor is routed, where, in the new problem, the packets at P_j are destined for $P_{(j+\frac{n}{2})\mathrm{mod}n}$, $0 \leq j < n$, and the extra 2 packets are routed towards the cut in the given direction.*

Proof Without loss of generality we consider the packets that cross the cut in the clockwise direction. Only packets that are originated in the half-ring defined by $P_{(i+\frac{n}{2})\mathrm{mod}n}, \ldots, P_{(i-1)\mathrm{mod}n}$ will cross the cut in that direction. We can convert the initial symmetric problem to the one defined in the lemma by executing Algorithm *Symmetric_Conversion(i)*.

Algorithm *Symmetric_Conversion(cut)*

Begin

For $j = 0$ **to** $n-1$ **do**

Convert(cut,j)

End

Convert(cut,j)

Begin

- Locate the processor that has packets destined for $P_{(cut+j)\mathrm{mod}n}$. Let it be processor a.
- Let the destination of the packets initially at $P_{(cut+j+\frac{n}{2})\mathrm{mod}n}$ be processor b.

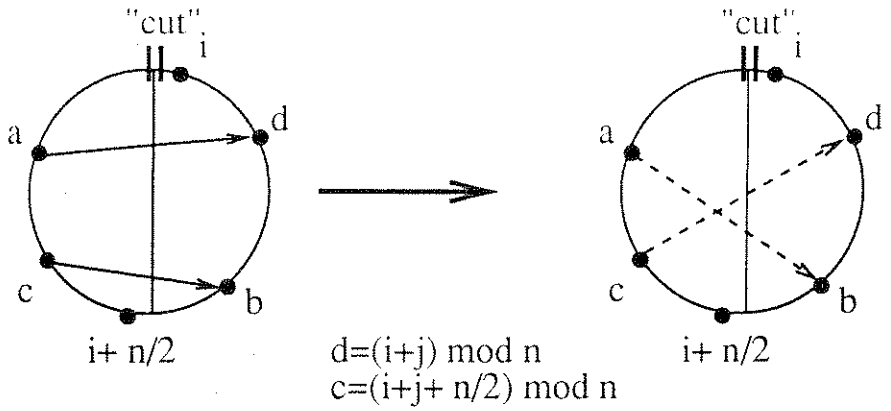


Figure 4: The transformation that occurs in a single iteration *Convert(cut,j)*.

- Swich the destinations of the packets such that:
  - The packets initially at $P_{(cut+j+\frac{n}{2}) \bmod n}$ are destined for $P_{(cut+j) \bmod n}$.
  - The packets initially at processor a are destined for processor b.

End

The transformation that occurs in a single execution of procedure *Convert(cut,j)* for $cut = i$, is illustrated in Figure 4.

We concentrate in the half-ring immediately before the cut in the clockwise direction. Consider iteration j, $0 \leq j < n/2$. Denote $P_{(i+j+\frac{n}{2}) \bmod n}$ by c and $P_{(i+j) \bmod n}$ by d. Observe that before the switch packets from processors a and c want to cross the cut in the clockwise direction. The packets originally at processor a will travel along their shortest path. So there are at most

$$k - \lfloor \frac{[ad]_{cw}}{n} k \rfloor \leq k - \frac{[ad]_{cw}}{n} k + 1 = \frac{k}{n}(n - [ad]_{cw}) + 1 = \frac{k}{n}[da]_{cw} + 1.$$

The packets originally at processor c will travel along their longest path. So there are at most

$$\lfloor \frac{[bc]_{cw}}{n} k \rfloor \leq \frac{[bc]_{cw}}{n} k.$$

Thus, before the switch at most $(k/n)([da]_{cw} + [bc]_{cw}) + 1$ packets will cross the cut in the clockwise direction.

After the switch, again, packets from both processors a and c will cross the cut. For the packets initially at processor a we have to consider two cases, namely, when they

travel along the shortest or the longest path. In the case where they travel along the shortest path, at least

$$k - \lfloor \frac{[ab]_{cw}}{n} k \rfloor \geq k - \frac{[ab]_{cw}}{n} k = \frac{k}{n}(n - [ab]_{cw}) = \frac{k}{n}[ba]_{cw}$$

packets will cross the cut. In the case where they travel along the longest path, at least

$$\lfloor \frac{[ba]_{cw}}{n} k \rfloor \geq \frac{[ba]_{cw}}{n} k - 1$$

packets will do so. Thus, in any case, at least $(k/n)[ba]_{cw} - 1$ packets originated at processor a will cross the cut. Let us now turn our attention to the packets originated at processor c. They have to travel exactly $n/2$ steps. We treat them as they were routed along the shortest path. Then, at least

$$k - \lfloor \frac{[cd]_{cw}}{n} k \rfloor \geq k - \frac{[cd]_{cw}}{n} k = \frac{k}{n}(n - [cd]_{cw}) = \frac{k}{n}[dc]_{cw}$$

packets originated at processor c will cross the cut. Thus, after the switch, a total of at least $(k/n)([ba]_{cw} + [dc]_{cw}) - 1$ packets will cross the cut. Recall that the number of packets that wanted to cross the cut before the switch was $(k/n)([da]_{cw} + [bc]_{cw}) + 1$. But,

$$([ba]_{cw} + [dc]_{cw}) = ([bc]_{cw} + [ca]_{cw} + [db]_{cw} + [bc]_{cw}) = ([da]_{cw} + [bc]_{cw})$$

Thus, if we load processor c with 2 extra packets and we route both of them towards the cut the new routing problem sends at least the same number of packet to cross the cut. ∎

Up to now, we have concentrated into the case where n, the number of processors on the ring, is even. In what follows, we describe how to augment a routing problem on a ring that consists of an odd number of processors to a routing problem on a ring that has one extra processor. This augmentation will never occur in practice. It is used only as a tool in proving statements regarding the number of packets that cross a cut in some direction if the number of processors in the initial problem is odd. For this reason, the augmentation will be defined in respect of a given cut in the initial ring and a given direction. Informally, the augmented routing problem with respect to cut c_i and direction *dir* is obtained as follows: A new processor is placed on the ring such that it becomes

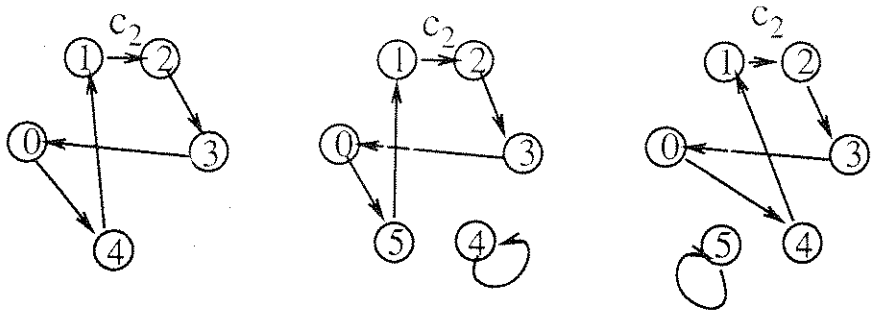


Figure 5: $R, R^R_{2,cw}, R^R_{2,ccw}$

the $\lceil n/2 \rceil^{th}$ processor after cut c_i in direction dir. The new processor is loaded with k packets that are destined for itself. Then the processors on the ring are renamed and the initial routing problem is updated so that the destinations reflect the new names. Figure 5 shows the results of the augmentation in a packet routing problem on a ring of 5 processors with respect to cut c_2 and both directions. Formally, we have the following definition:

Definition Given a multipacket permutation problem $R =< n, k, F >$, we define the *augmented multipacket permutation problem of R with respect to cut c_i and direction dir*, $dir = cw$ or ccw, to be:

$$R^R_{i,dir} = \begin{cases} R & \text{if } n \text{ is even} \\ < n+1, k, F' > & \text{if } n \text{ is odd} \end{cases}$$

where $F' : \{0, 1, \ldots, n\} \rightarrow \{0, 1, \ldots, n\}$ is defined as:

$$F'(j) = \begin{cases} (i + \lceil \frac{n}{2} \rceil + A(dir)) \bmod n & \text{if } j = (i + \lceil \frac{n}{2} \rceil + A(dir)) \bmod n \\ M(F(M^{-1}(j))) & \text{otherwise} \end{cases}$$

where

$$A(dir) = \begin{cases} -1 & \text{if } dir = cw \\ 0 & \text{if } dir = ccw \end{cases}$$

and $M : \{0, 1, \ldots, n-1\} \rightarrow \{0, 1, \ldots, n\}$ is an injective function that describes the renaming of the processors and is defined by:

$$M(j) = \begin{cases} j & \text{if } 0 \leq j < (j + \lceil \frac{n}{2} \rceil + A(dir) - 1) \bmod n \\ j+1 & \text{otherwise} \end{cases}$$

Lemma 5 *Assume a multipacket permutation routing problem* $R =< n, k, F >$. *If Algorithm Route_on_a_Ring_2 is used for the routing then the number of packets that cross any cut in any direction is at most:*

a.	$kn/4 + 1.5n$	*if* n *and* k *are even*
b.	$kn/4 + 2n$	*if* n *is even and* k *is odd*
c.	$kn/4 + 2n$	*if* n *is odd and* k *is even*
d.	$kn/4 + 2.5n$	*if* n *and* k *are odd*

Proof Without loss of generality consider cut c_i in the clockwise direction and the packets that cross it in that direction. First we prove cases a and b, i.e., when n is even. Lemmata 3 and 4 imply that a routing problem that is symmetric with respect to cut c_i and has initial load of $k + 3$ packets per processor, all destined for the processor located after $n/2$ positions in the clockwise direction, will send at least an equal number of packets to cross the cut c_i in the clockwise direction with any other multipacket permutation routing problem with initial load k packets per processor. (The 3 extra packets will be routed in the clockwise direction towards the cut.) For that "harder" problem, in the case where k is even, Algorithm *Route_on_a_Ring_2* will send exactly $kn/4$ packets to cross the cut. In the case where k is odd, at most $n/2$ extra packets will cross the cut. This is caused by the $\lfloor\ \rfloor$ operator that is involved in the computation of σ_i at step 1 of the algorithm. Now, by adding the extra flow created by the 3 extra packets that are routed towards the cut (exactly $1.5n$ extra packets will cross it) we get the quantities stated in cases a and b of the lemma.

In the case where n is odd, we consider the augmented problem $R^R_{i,cw}$ ($R^R_{i,ccw}$ if the ccw direction is considered). We make the following observation: If the augmented problem is routed using Algorithm *Route_on_a_Ring_2*, then the number of packets that cross cut c_i in the clockwise direction might be reduced by at most $n/2$. This is because of the renaming of the processors. The renaming causes the distances between processors to change. So, the number of packets that a processor sends towards the cut might change also. However, this number changes by at most 1 packet, and furthermore, it is reduced only at the processors in the half-ring immediately before the cut in the direction under consideration. Working in the same lines as before, in order to overcome the above

problem and guarantee that the augmented problem is at least as hard as the original, we load each processor in that half-ring with one extra packet that is routed towards the cut, i.e., the processors in the half-ring before the cut in the augmented problem have $k+1$ packets. Then, cases c and d are proven by Lemmata 3 and 4 and by arguing as in cases a and b for k even and odd, respectively. ∎

Theorem 3 *Using Algorithm Route_on_a_Ring_2, a bisection limited multipacket permutation problem* $R =< n, k, F >$ *on a ring of processors can be solved in*

a. $kn/4 + 1.5n$ *routing steps if* n *and* k *are even*
b. $kn/4 + 2n$ *routing steps if* n *is even and* k *is odd*
c. $kn/4 + 2n$ *routing steps if* n *is odd and* k *is even*
d. $kn/4 + 2.5n$ *routing steps if* n *and* k *are odd*

Proof The theorem is implied from Lemma 5. If at every step of the algorithm one packet wants to cross the cut, the theorem is obviously true. In the case where there is a period during the execution of the routing algorithm that no packet crosses the cut, say μ steps, then there are μ processors that do not have any packets that want to cross the cut. Thus, the number claimed at Lemma 5 was overestimated by at least μ. So, the theorem holds. ∎

5 Pure Routing Algorithms

One basic characteristic that both Algorithms *Route_on_a_Ring_1* and *Route_on_a_Ring_2* possess is that the decision of how many packets a processor will route towards a direction is independent of the position of the processor on the ring. Two processors that both have packets destined after distance d in the same direction will make exactly the same decision regarding the number of packets they route in the clockwise and the counter-clockwise direction. The above property defines a distinct class of routing algorithms.

Definition Assume an interconnection network N on which a packet routing problem is to be routed. A routing algorithm with the property that the routing decisions taken at any time t, by any processor P, are independent of the position of processor P on the network, is called a *pure routing algorithm.*

It should be mentioned that, not too many routing algorithms of those proposed in the literature for multidimensional meshes are pure algorithms. The algorithms that use sorting of subnetworks as a subroutine [3, 4, 6, 8, 10, 12] certainly do not belong in this class. The same holds for algorithms that treat corner packets differently [2, 8, 12, 13]. For 2-dimensional meshes, as pure algorithms can be considered the greedy routing algorithm that routes the packets first to the correct column and then tho the correct row, and the algorithm proposed by Symvonis and Makedon in [9] [15, Chapter 2].

The following theorem demonstrates that different lower bounds can be obtained for pure routing algorithms. This is done by proving an $(n+1)k/4$ lower bound for the bisection limited permutation problem on a ring of n processors. Recall that the general lower bound for this problem is $kn/4$.

Theorem 4 *There is a bisection limited permutation routing problem on a ring of n processors that requires for its solution by any pure routing algorithm at least $(n+1)k/4$ routing steps.*

Proof Consider the situation where k, the number of packets per processor, is odd, the ring consists of an even number of processors , and the packets at P_i are destined for $P_{(i+\frac{n}{2}) \bmod n}$, $0 \leq i < n$. Let $k = 2\mu + 1$. Since we assume a pure routing algorithm, all processors must send the same number of packets along the clockwise direction (and thus, along the counter-clockwise direction). There are k different decisions that the algorithm can make. In decision d, $1 \leq d \leq k$, d packets are routed along the clockwise direction and $k - d$ are routed along the counter-clockwise direction. If decision d was taken, the routing will be completed after at least $(n/2)\max(d, k-d)$ steps. Then, the lower bound is the minimum number of routing steps needed to complete the routing if any of the d decisions was taken. Thus, any pure algorithm needs in the worst case at least

$$\frac{n}{2} \min_{1 \leq d \leq k} (\max(d, k-d)) = \frac{n}{2} \lceil \frac{k}{2} \rceil = \frac{n}{2}(\mu + 1) = \frac{n}{2}\mu + \frac{n}{2} = (2\mu + 1)\frac{n}{4} + \frac{n}{4} = \frac{(k+1)n}{4}$$

∎

6 Conclusions - Further work

In this paper, we have examined multipacket routing problems on rings. We divided these problems into two categories, distance limited and bisection limited routing problems. We presented a new lower bound for the case of distance limited problems and we gave an algorithm that tightens the lower bound. For the case of bisection limited problems, we presented an algorithm that solves any problem within $kn/4 + 2.5n$ routing steps. Thus, we have succeed to present an algorithm that approximates within an additive factor the number of routing steps required in the worst case. The class of the pure routing algorithms was defined, and it was demonstrated that better lower bounds can be obtained for this class of routing algorithms.

The results presented in this paper for the case of bisection limited permutation problems were at most $2.5n$ routing steps away from the lower bound. An interesting problem is to design an algorithm that matches the lower bound. We anticipate that this algorithm will not fall into the class of pure routing algorithms. Another problem that deserves attention is to derive lower bounds for pure routing algorithms for multidimensional meshes and tori. In this case it would be very interested if we had a lower bound that is a function of the maximum number of packets that can be accomodated at the queue of any processor.

References

[1] B. Aielo, F.T. Leighton, B. Maggs, M. Newman, "Fast Algorithms for Bit-Serial Routing on a Hypercube", Proceedings of the 2^{nd} Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '90, July 2-6, 1990, Crete, Greece.

[2] D. Krizanc, S. Rajasekaran, Th. Tsantilas, "Optimal Routing Algorithms for Mesh-Connected Processor Arrays", VLSI Algorithms and Architectures (AWOC'88), J. Reif, editor, Lecture Notes in Computer Science 319, 1988, pp. 411-422.

[3] M. Kunde, "Routing and Sorting on Mesh-Connected Arrays", VLSI Algorithms and Architectures (AWOC'88), J. Reif, editor, Lecture Notes in Computer Science 319, 1988, pp. 423-433.

[4] M. Kunde, "Balanced routing: Towards the Distance Bound on Grids", Proceedings of the 3^{rd} Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA'91, 1991.

[5] M. Kunde, "Concentrated Regular data Streams on Grids: Sorting and Routing Near to the Bisection Bound", Proceedings of the 32^{nd} IEEE Symposium on Foundation of Computer Science, 1991.

[6] M. Kunde,T. Tensi, "Multi-Packet Routing on Mesh Connected Arrays", Proceedings of ACM Symposium on Parallel Algorithms and Architectures, SPAA'89, June 1989, pp. 336-343.

[7] F.T. Leighton, "Average Case Analysis of Greedy Routing Algorithms on Arrays", Proceedings of the 2^{nd} Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '90, July 2-6, 1990, Crete, Greece.

[8] F.T. Leighton, F. Makedon, I.G. Tollis, "A 2n-2 Algorithm for Routing in an $n \times n$ Array With Constant Size Queues", Proceedings of ACM Symposium on Parallel Algorithms and Architectures, SPAA'89, June 1989, pp. 328-335.

[9] F. Makedon, A. Simvonis, "An Efficient Heuristic for Permutation Packet Routing on Meshes with Low Buffer Requirements", to appear in IEEE Transactions on Parallel and Distributed Systems.

[10] F. Makedon, A. Simvonis, "On Bit Serial Packet Routing for the Mesh and the Torus", Proceedings of the 3^{rd} Symposium on the Frontiers of Massively Parallel Computation, October 8-10 1990, pp. 294-302.

[11] J.Y. Ngai,C.L. Seitz, "A Framework for Adaptive Routing in Multicomputer Networks", Proceedings of ACM Symposium on Parallel Algorithms and Architectures, SPAA'89, June 1989, pp. 1-9.

[12] S. Rajasekaran, R. Overholt, "Constant Queue Routing on a Mesh", to appear in the Journal of Parallel and Distributed Computing.

[13] S. Rajasekaran, M. Raghavachari, "Optimal Randomized Algorithms for Multipacket and Wormhole Routing on the Mesh", Proceedings of the 3^{rd} IEEE Symposium on Parallel and Distributed Processing, Dallas, 1991.

[14] A.G. Ranade, "How to Emulate Shared Memory", Proceedings of the 28^{th} IEEE Symposium on Foundation of Computer Science, 1987, pp. 185-194.

[15] Simvonis A., " Packet Routing Problems on Mesh Connected Machines and High Resolution Layouts", Ph.D. Thesis, University of Texas at Dallas, 1991.

[16] L.G. Valiant, G.J. Brebner, "Universal Schemes for Parallel Communication", Proceedings of the 13^{th} Annual ACM Symposium on the Theory of Computing, May 1981, pp. 263-277.