ELSEVIER

Information Processing Letters 57 (1996) 215-223

Information Processing Letters

Routing on trees

Antonios Symvonis [1]

Basser Department of Computer Science, University of Sydney, Sydney, N.S.W. 2006, Australia

Received 1 February 1995; revised 1 October 1995
Communicated by R.G. Dromey

Abstract

In this paper, we study the permutation packet routing problem on trees. We show that every permutation can be routed on a tree of n vertices in $n-1$ routing steps. We provide two algorithms that produce such routing schedules. The first algorithm builds in $O(n^2)$ time a schedule that needs $O(n^2)$ bits for its description while the second one produces in $O(n^3)$ time a schedule that can be described with $O(n \log n)$ bits.

Keywords: Algorithms; Off-line routing; Permutation; Routing number; Routing model; Trees

1. Introduction

The *permutation packet routing problem* on a connected undirected graph is the following: We are given a graph $G = (V, E)$ and a permutation π of the vertices of G. Every vertex v of G contains a packet destined for $\pi(v)$. Our task is to route all packets to their destinations.

During the routing, the movement of the packets follows a set of rules. These rules specify the *routing model*. Routing models might differ on the way edges are treated (unidirectional, bidirectional), the number of packets a vertex can receive (transmit) in a single step, the number of packets that are allowed to queue in a vertex (queue-size), etc. Usually, routing models are described informally.

Let $rt_M(G, \pi)$ be the number of steps required to route permutation π on graph G by using routing model M. The routing number of graph G with respect to routing model M, $rt_M(G)$, is defined to be

$$rt_M(G) = \max_{\pi} rt_M(G, \pi)$$

over all permutations π of the vertex set V of G.

The routing number of a graph was first defined by Alon, Chung and Graham in [1]. In their routing model, the only operation allowed during the routing was the exchange of packets at the endpoints of an edge of graph G. The exchange of packets at the endpoints of a set of disjoint edges (a matching on G) can occur in one routing step. We refer to this model as the *matching routing model* and, for a graph G, we refer to the routing number of G with respect to the matching routing model, simply as the routing number of G, denoted by $rt(G)$. In [1], it was shown that $rt(T) < 3n$ for any tree T of n vertices.

A lot of work has been devoted to the study of packet routing problems (see [4] and the references therein). As it is natural, several routing models have been considered. However, most of the papers in the literature consider model which are quite different than the unrealistic matching routing model.

In this paper, we will consider a routing model

[1] Email: symvonis@cs.su.oz.au.

which is closely related to the routing models used in practice. At any time step, all edges can carry at most one packet in each direction. Moreover, at each time step, at most $d(v)$ packets can be found in any vertex v of degree $d(v)$ and no pair of these packets competes for the same communication link. A different way to describe this restriction is to say that with each communication link there is an associated buffer that can hold at most one packet. The routing schedule has to assure that this buffer is never overloaded. In addition, we assume that each node can store the packet that originates from it (until its transmission) at no extra cost. This is a common assumption in packet routing algorithms. We will refer to this model as the *simplified routing model* and we will denote the routing number of graph G with respect to it by $rt'(G)$.

The only previous work on off-line packet routing which applies to a large class of interconnection networks is that of Annexstein and Baumslag [2]. They presented a method to solve the permutation offline packet routing problem on product graphs. They proved that if we know how to route off-line any permutation on graphs G and H (without using any queues) in at most $r(G)$ and $r(H)$ routing steps, respectively, then we can route off-line any permutation on the product graph $F = G \times H$ in

$$r(F) = 2 \cdot \min(r(G), r(H)) + \max(r(G), r(H))$$

steps (again, without creating any queues). The method of Annexstein and Baumslag cannot be used for routing permutations on trees since trees are not product graphs.

The rest of the paper is organised as follows: In Section 2, we give definitions for terms we use in the paper. In Section 3, we show that $rt'(T) < n$ for any tree $T = (V, E)$ of n vertices. We achieve this upper bound by demonstrating an algorithm that computes a routing schedule of at most $n - 1$ steps for any permutation over the vertex set of T. The routing schedule requires $O(n^2)$ bits for its description and is computed in $O(n^2)$ time. In Section 4, we describe an algorithm that produces a routing schedule that needs $O(n \log n)$ bits for its description in $O(n^3)$ time. We conclude in Section 5 with further research that has to be done in this area.

2. Preliminaries

We assume that the reader is familiar with the standard graph theory terminology and notation (see [3]). Given an undirected graph G, we can transform it to a directed one by substituting each edge (u, v) in $E(G)$ by the pair of *anti-parallel* edges (u, v) and (v, u). We denote the graph produced by the above transformation by G^D. The set $Neighbors(v, G)$ is defined to be the set of vertices in G that can be reached from v by crossing just one edge. Formally,

$$Neighbors(v, G) = \{w \mid (v, w) \in E \text{ of } G\}.$$

We define the directed *self-loop augmented graph* $G^{SL} = (V, E')$ of $G = (V, E)$ to be the graph with $E' = E \cup \{e^v = (v, v) \mid v \in V\}$.

A *permutation packet routing problem* R is defined to be the pair (G, π) where $G = (V, E)$ is the directed graph that represents the network in which the routing will take place and π is the permutation to be routed. Formally, the set P of $|V|$ packets to be routed is defined by $P = \{p_1, p_2, \ldots, p_{|V|} \mid p_i = (i, \pi(i)), i, \pi(i) \in V, 1 \leqslant i \leqslant |V|\}$. A more general definition that incorporates the maximum allowed queue-size was given in [5]. Note that, even though the informal definition of most routing models involves an undirected graph G with bidirectional communication links, the corresponding directed graph G^D can be used in the formal definition of the routing problem.

An *off-line solution* (or *routing schedule*) of *length* L for the off-line packet routing problem $R = (G, \pi)$ is a set of directed paths $SOLUTION(R) = \{d_1, d_2, \ldots, d_{|V|}\}$ where d_i is the directed path corresponding to packet p_i. The paths are taken on graph G^{SL}, the self-loop augmented graph of G, instead of G. A self-loop from vertex v in the path of some packet indicates that the packet was not advanced at the corresponding routing step. Each directed path contains at most $L + 1$ vertices. For $i = 1, \ldots, |V|$ we have that

$$d_i = v_i^0 v_i^1 \ldots v_i^l, \quad 0 \leqslant l \leqslant L,$$

where, $v_i^0 = i$ and $v_i^l = \pi(i)$. In order to have a valid solution for our routing problem, the directed paths must satisfy the condition: "*At any routing step, each edge that corresponds to an unidirected communication link appears in at most one directed path.*" An off-line solution is said to be *uninterrupted* if no packet gets delayed once it starts its movement.

In order to describe the solution of a permutation routing problem we need to specify for each packet the path it follows during the routing. In the worst case, a solution of length L can be represented by a two dimensional matrix *SOLUTION* of $|V|$ rows (one for each packet) and $L+1$ columns (one for each routing step). $SOLUTION[p,t]$, $0, \leqslant t \leqslant L$, is the vertex in which packet p is after t routing steps. The space needed for reporting the solution is $O(|V| L \log |V|)$ bits. However, in cases in which there is a unique path between any pair of vertices of the underlying graph and derouting is not allowed, a path can be determined simply by knowing if at a given step the packet is advanced towards its destination. In this case space of $O(|V|L)$ bits is sufficient. Furthermore, if it also holds that the movement of the packets is uninterrupted, then only the step in which each packet starts its routing needs to be stored and, thus, $O(|V| \log L)$ bits are sufficient for reporting the routing schedule.

Given a tree T, we denote the unique path in T from vertex u to vertex v by $path(u,v)$. The number of edges in $path(u,v)$ is denoted by $path\_size(u,v)$. We assume that, if w is a vertex in $path(u,v)$, we can determine the vertex that is immediately after w in the path from u to v in constant time. It is easy to do so by using a $|V| \times |V|$ matrix N such that $N[u,v]$ contains the first vertex (not including u) of $path(u,v)$. Of course, some preprocessing is necessary to initialize matrix N and several algorithms for doing so are available. Obvious choices include shortest path algorithms and tree traversal techniques [3] which can optimally initialize matrix N in $O(|V|^2)$ time. In the rest of the paper, we assume that the information of matrix N is available to us.

3. Routing on trees

In this section, we show that $rt'(T) < n$. We prove this bound by exhibiting an algorithm that, given a tree $T = (V,E)$ of n vertices and a permutation π on T's vertex set, produces a routing schedule for the permutation problem (T^D, π) of length at most $n-1$. The routing schedule is produced in $O(n^2)$ time and requires $O(n^2)$ bits for its description.

3.1. The routing graph

For each vertex $v \in V$ we construct the set

$$S_v = \{v_u \mid u \in Neighbors(v,T)\} \cup \{v_{\text{con}}\}$$

("con" stands for "consume"). The set $V^R = \bigcup_{v \in V} S_v$ will be the vertex set of an auxiliary directed graph $T^R = (V^R, E^R)$ which we will use in the algorithm. We call T^R the *routing graph*. The edge set of the routing graph will be different at each stage of the off-line routing algorithm. We denote by $T_i^R = (V^R, E_i^R)$ the routing graph at stage i. E_i^R is the edge set of T_i^R.

During the course of the routing, we denote by $current(p)$ the current position of packet p while, by $orig(p)$ we denote its origin and by $dest(p)$ we denote its destination. Since the routing is happening on a tree, for each packet p, there is a unique simple path $path(current(p), dest(p))$ from $current(p)$ to $dest(p)$. We denote by $f(p)$ the first vertex on this path (not including $current(p)$) and by $s(p)$ the second one. In the case that $s(p)$ and/or $f(p)$ are not well defined ($path(current(p), dest(p))$ is too short for $s(p)$ and/or $f(p)$ to have meaning), we assume that they return the special value "con" (for "consumed").

To define graph $T_i^R = (V^R, E_i^R)$ we simply have to specify E_i^R since V^R is fixed. E_i^R contains at most $|V|$ edges, one for each packet that has not reached its destination after i routing steps (stages). The edge that corresponds to packet p, denoted by $edge(p)$, is defined as follows:

$$edge(p) = \begin{cases} (current(p)_{f(p)}, f(p)_{s(p)}) \\ \quad \text{if } f(p) \neq dest(p), \\ (current(p)_{f(p)}, f(p)_{\text{con}}) \\ \quad \text{otherwise.} \end{cases}$$

What we want to represent with each edge is the information that, if in this routing step packet p travels through edge $(current(p), f(p))$ of T, then in the next step it will compete for edge $(f(p), s(p))$ of T. An example of a routing graph is given in Fig. 1.

The following lemmata follow from the construction of the routing graph.

Lemma 1. *Let V^R be the vertex set of the routing graph constructed from tree $T = (V,E)$. Then, $|V^R| = 3|V| - 2$.*

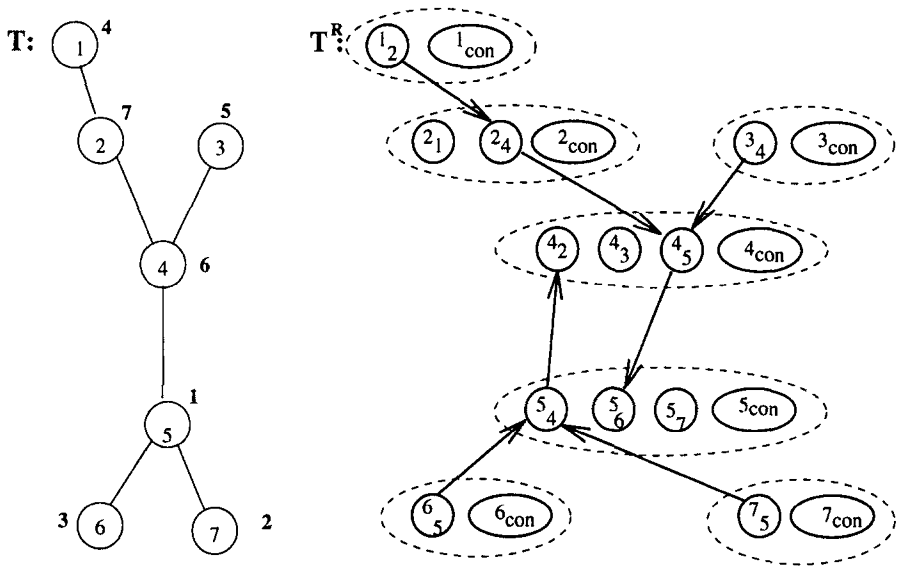

Fig. 1. The numbers next to the vertices of the tree T represent the destination of the packet located in that vertex. Graph T^R is the routing graph which corresponds to tree T and the permutation to be routed.

Lemma 2. *Assume a distribution of $|V|$ packets at the vertices of a tree $T = (V, E)$ which satisfies the requirement that no two packets compete for the same edge in a given direction, i.e., there does not exist a pair of packets p and q such that $current(p) = current(q)$ and $f(p) = f(q)$. Then, the corresponding routing graph T^R consists of a collection of directed (toward the root) trees and a set of isolated vertices.*

3.2. An $O(n^2)$ off-line routing algorithm for trees

Consider any tree T_v rooted at vertex v which is a (not strongly connected) component of a routing graph T^R (see for example the tree rooted at vertex 5_6 in Fig. 1). If at the first routing step we advance all the packets that correspond to the edges of the tree, then, at the second step, the edges of the original tree T that correspond to vertices of in-degree greater than 1, will be requested by more than one packet. (To see this, recall what is the information that a routing graph represents.) To avoid this situation, we will not advance all packets. We will advance the packets that correspond to only one path (arbitrary chosen) connecting v (the root of T_v) with one of the leaves of T_v. We can choose the packets which will move during the next step (and also notify the ones that will not) by a simple traversal of the tree (in the opposite direction from that indicated by the edges) in $O(|V(T_v)|)$ time.

We are now ready to present a high level description of the first off-line routing algorithm for trees.

```
Algorithm Off_line_tree_routing_1($T, \pi$)
/* $\pi$ is the permutation to be routed on tree $T$ */
1. $i = 0$    /* $i$ denotes the number of routing steps
   (stages) completed so far */
2. While there are still packets that have not reached
   their destination do
   (a) Based on the current position of the packets
       (after $i$ steps of routing) construct the routing
       graph $T_i^R$.
   (b) Choose, based on the trees that form $T_i^R$, the
       packets that will move in step $i + 1$.
   (c) Move the packets, i.e., update the data struc-
       ture that keeps track of the current position
       and the journey of each packet.
   (d) $i = i + 1$
```

Theorem 3. *Assume a tree T of n vertices and a permutation π on its vertex set that has to be routed. Then, Algorithm* Off_line_tree_routing_1(T, π) *produces an off-line routing solution of at most $n - 1$ steps which can be described with $O(n^2)$ bits, in $O(n^2)$ time. As a consequence, $rt'(T) < n$.*

Proof. First observe that Step 2(b) of Algorithm *Off_line_tree_routing_1*(T, π) is well defined. More specifically, it is not difficult to prove by induction that at the beginning of each iteration of the while-loop the invariant "*there does not exist a pair of packets* p *and* q *such that* $current(p) = current(q)$ *and* $f(p) = f(q)$" holds. Then, by Lemma 2 we conclude that T^{R} consists of a collection of directed (towards the root) rooted trees and isolated vertices.

Next, we prove that the produced schedule is of at most $n - 1$ routing steps. We do that by showing that any arbitrary packet p reaches its destination after $n-1$ steps. In each of the rooted trees in T^{R}, every vertex but the root can be considered to contain a packet that wants to move.[2] Assume that packet p did not move during some stage. For that stage, consider the path from $current(p)$ to the root v of the tree T_v that p belongs to, and the path in T_v of which the packets were advanced (see Fig. 2). Obviously, p does not belong in that path. p was not advanced because the decision taken at vertex $current(x)$ was to advance packet q instead of packet y, the ancestor of p. In this case, we say that *packet* q *delayed packet* p. Note that the above definition of *delay* allows for only one packet to delay packet p at each stage. The fact that derouting never happens, implies that (i) p cannot be delayed by the same packet twice, and (ii) the packets initially at the vertices of the path $path(orig(p), dest(p))$ cannot delay p. Thus, p can be delayed by at most

$$n - (path\_size(orig(p), dest(p)) + 1)$$

packets. So, p will reach its destination after at most

$$n - (path\_size(orig(p), dest(p)) + 1) + (path\_size(orig(p), dest(p))) = n - 1$$

routing steps.

For the time analysis, we know from Lemma 1 that the number of vertices of the routing graph T^R that corresponds to a tree of n vertices is $3n - 2$. Based on this, it is not difficult to implement each iteration of the while-loop in $O(n)$ time. Since there will be at most $n - 1$ iterations, we conclude that Algorithm *Off_line_tree_routing_1*(T, π) produces an off-line solution in $O(n^2)$ time. At each routing step, each packet either advances towards its destination or waits at some vertex. So, the journey of each packet can be described by an array of 0/1 entries. So, to report the routing schedule we need space for $O(n^2)$ bits. □

[2] The packet that corresponds to vertex u_x in one of the rooted trees of T_{R} resides in vertex v of T.

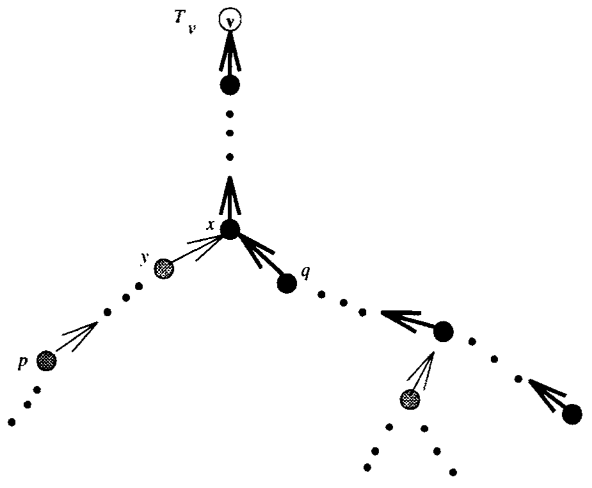

Fig. 2. Only the packets which reside in solid vertices are advanced.

4. A more compact routing schedule

In this section, we present an algorithm based on the *multistage off-line routing method* introduced by Symvonis and Tidswell in [5]. Given a permutation π to be routed on tree $T = (V, E)$, we produce a routing schedule of at most $n - 1$ routing steps that can be described with $O(n \log n)$ bits.

4.1. The multistage off-line routing method

Consider any routing problem $R = (G, \pi)$ where, $G = (V, E)$ is the directed graph and π is the permutation to be routed. Assume a trivial upper bound of B routing steps for the problem under consideration. (For the case of trees of n vertices, let $B = 2n$. Later on, we will see that it is sufficient to set $B = n$.)

We construct a multistage directed graph $G^{\mathrm{M}} = (V^{\mathrm{M}}, E^{\mathrm{M}})$, where $V^{\mathrm{M}} = \{(v,t) \mid v \in V \text{ and } 0 \leqslant t \leqslant B\}$ and $E^{\mathrm{M}} = \{((v,t),(w,t+1)) \mid w \in Neighbors(v,G) \text{ and } 0 \leqslant t < B\}$. The edges of E^{M} represent the communication that can take place between adjacent vertices of the interconnection network at any time. Fig. 3 shows the resulting graph when the interconnection network is a chain of 5 vertices. For permutations, an obvious lower bound of 4 routing steps that is based on a distance argument applies. For a graph $G = (V, E)$, the multistage graph G^{M} has $O(|V|B)$ vertices and $O(|E|B)$ edges.

Let $tower(G^{M}, v)$ be the set of vertices of the multistage graph G^{M} that corresponds to vertex v in G. Formally, $tower(G^{M}, v) = \{(v,t) \mid v \in V, (v,t) \in V^{M}, 0 \leqslant t \leqslant B\}$. The following theorem (proven in [5]) allows us to approach the off-line packet routing problem from a different point of view.

Theorem 4. *Let $R = (G, \pi)$ be a routing problem. R has an uninterrupted solution of length L if and only if for each packet p there exists a path from a vertex $(orig(p), t)$ in $tower(G^{M}, orig(p))$ to vertex $(dest(p), t')$ in $tower(G^{M}, dest(p))$, $t' = t + distance(orig(p), dest(p)) \leqslant L$ and all such paths are mutually edge disjoint.*

```
Algorithm $Off\_line\_tree\_routing\_2(T, \pi)$
/* $\pi$ is the permutation to be routed on tree $T$ */
1. Let $P$ denote the set of packets to be routed. For
   each packet $p \in P$ let $START\_ROUTING(p) = 0$.
2. Order the packets in $P$. The way we order the pack-
   ets will be specified later.[3]
3. While $P \neq \emptyset$ do
   (a) Remove the next packet from $P$ (with respect
       to $P$'s ordering).
       Let it be packet $p$.
   (b) $start = 0$
   (c) While there does not exist a path in $G^M$
       from $(orig(p), start)$ to $(dest(p), start +$
       $path\_size(orig(p), dest(p)))$
       do $start = start + 1$
   (d) Update $G^M$ by removing from it the edges of
       the path from $(orig(p), start)$ to $(dest(p),$
       $start + path\_size(orig(p), dest(p)))$
   (e) $START\_ROUTING(p) = start$
```

The routing schedule that Algorithm *Off_line_tree_routing_2*(T, π) produces has the property that when a packet starts its movement it is never delayed at any intermediate vertex. Because of that property, the schedule can be fully described by simply stating the step in which each packet starts its movement towards its destination. To do this, we need $O(n \log n)$ bits.

The following lemma can be used to provide a first upper bound on the time complexity of Algorithm *Off_line_tree_routing_2*(T, π).

Lemma 5. *Assume a tree T of n vertices and a permutation π on its vertex set that has to be routed. Then, for an arbitrary ordering of the packets, Algorithm* Off_line_tree_routing_2(T, π) *produces a routing schedule in which all packets start their routing within n steps and, as a consequence, in the worst case a routing schedule of length at most $2n - 1$ is produced.*

Proof. Assume that we try to assign a path in the multistage graph G^{M} for packet p and we fail. This means that, for some *start*, there exist some edges in the unique path from vertex $(orig(p), start)$ to vertex $(dest(p), start{+}path\_size(orig(p), dest(p)))$ that are missing. Let e be the first missing edge and let q be the packet that used it. We say that *packet q delayed packet p*.[4] The way we assign paths in Algorithm *Off_line_tree_routing_2*(T, π) guarantees that when a packet starts moving, it is never delayed again. This, in turn, implies that packet p cannot be delayed by packet q more than once. Thus, within the first n tries ($start = 0 \ldots n-1$) we are guaranteed to find a path. Given the fact that the maximum distance a packet has to travel is at most $n - 1$, we conclude that the produced routing schedule is of length at most $2n - 1$.

Lemma 5 ensures us that, in the worst case, we might try n different paths for each packet. Since each path is at most $n-1$ edges long, we might need $O(n^2)$ time to compute the path of a single packet. Thus, $O(n^3)$ is an upper bound on the time complexity of Algorithm *Off_line_tree_routing_2*(T, π) if an arbitrary ordering is assumed.

4.2. An ordering that results to an optimal schedule

By being more careful in the order we route the packets, we can obtain a routing schedule of length

[3] Different orderings result to routing schedules of different lengths.

[4] Note that this definition of *delay* is different from that used in the proof of Theorem 3.

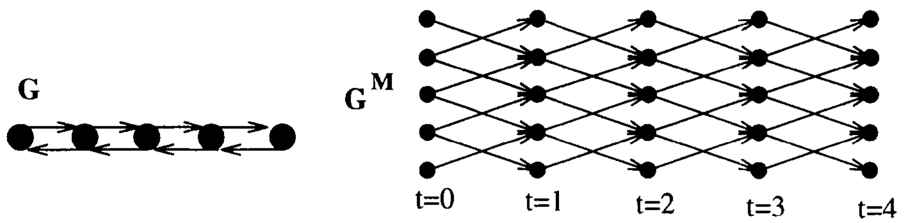

Fig. 3. A chain of 5 vertices and its corresponding multistage graph.

at most $n-1$.

We will compute an ordering for the packets such that any packet p that has to travel distance $path\_size(orig(p), dest(p))$ to its destination, will fail to find a path in the multistage graph at most $n-1-path\_size(orig(p), dest(p))$ times. Consider any packet p, the path $path(orig(p), dest(p))$ and the set of packets $I(p)$ that are initially in vertices of this path. Formally, $I(p) = \{q \mid orig(q) \in path(orig(p), dest(p))\}$. Out of the packets in set $I(p)$, only those that during their travel use some edges of the directed path $path(orig(p), dest(p))$ can delay packet p. Denote the set of these packets by $D(p)$ (see Fig. 4). However, a packet $q \in D(p)$ can delay packet p only if a path is assigned to q prior to p. Thus, our strategy will be to construct an ordering that places p earlier in the order than any packet of $D(p)$.

What follows is a high level description of an algorithm that determines an ordering of the packets. Each packet p has associated with it two variables, namely, $crossings[p]$ and $order[p]$. The packets will be inserted in a priority queue Q in which priorities are assigned based on the $crossings[\ ]$ values of the packets. The priority queue supports the standard operations $delete\_min(Q$: Priority queue) and $update(p$: Packet, v : Value, Q : Priority queue).

```
Algorithm Order(T, π)
/* π is the permutation to be routed on tree T */
1. For each packet p do
   (a) Let $e = (orig(p), f(p))$ be the first (directed) edge of T that p has to cross.
   (b) $crossings[p]$ = the number of packets that cross edge e during their routing.
   (c) $Insert(p, Q)$
2. $i = 1$
3. While $Q \neq \emptyset$ do
   (a) $p = delete\_min(Q)$
   (b) $order(p) = i$
   (c) For each packet $q \in D(p)$ do
         $update(q, crossings(q) - 1, Q)$
   (d) $i = i + 1$
```

Lemma 6. *Assume a tree T of n vertices and a permutation π on its vertex set that has to be routed. Then, when using the ordering determined by Algorithm* Order(T, π), *Algorithm* Off_line_tree_routing_2(T, π) *produces a routing schedule of at most $n-1$ steps. Moreover, the ordering is computed in $O(n^2)$ time.*

Proof. To prove that Algorithm $Order(T, \pi)$ produces the required ordering, we have to verify that the invariant "*there is a packet p in the priority queue with $crossings[p] = 0$*" always holds at the beginning of each iteration of the while-loop of Step 3. This can be easily proved by induction.

The fact that when we assign a value to $order[p]$, for every packet p, $crossings[p]$ is equal to 0, guarantees that an $order[\ ]$ value has not been assigned yet to any of the packets in $D(p)$. Thus, these packets cannot delay packet p.

Consider now the time complexity of Algorithm $Order(T, \pi)$. By traversing the paths along which the packets travel, we can compute for each (directed) edge the number of packets that cross it. Thus, Step 1(b) takes $O(n^2)$ time. In Step 3, $O(n^2)$ *updates* can happen. Since the update operations are actually "*decrease by* 1" operations, an implementation (array of linked lists) for which Step 3 requires $O(n^2)$ time is possible. □

The following theorem summarises the results of this section.

Theorem 7. *Assume a tree T of n vertices and a per-*

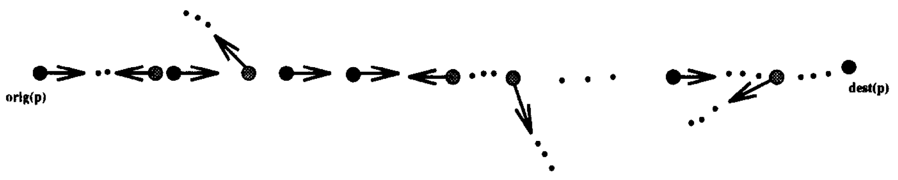

Fig. 4. Only the solid packets can delay packet p.

mutation π on its vertex set that has to be routed. Then, when using the ordering determined by Algorithm Order(T,π), *Algorithm* Off_line_tree_routing_2(T,π) *produces in* $O(n^3)$ *time an off-line routing solution of at most $n-1$ steps which can be described with* $O(n\log n)$ *bits.*

5. Further work

An interesting problem we are currently working on is to design algorithms with performance close to the *actual lower bound* for the permutation on hand. Such lower bounds can be obtained based on combinations of distance and bisection arguments or, by using the multistage routing method introduced in [5] and the relation between the routing problem and the multicommodity flow problem.

Appendix. Proof Of Lemmata 1 and 2

Lemma 8. *Let V^{R} be the vertex set of the routing graph constructed from tree $T=(V,E)$. Then, $|V^{\mathrm{R}}| = 3|V|-2$.*

Proof. Observe that for each edge (u,v) of tree T, we create 2 vertices, namely u_v and v_u, in the routing graph. Thus,

$$\sum_{v\in V}|\{v_u \mid u\in Neighbors(v,T)\}| = 2|E| = 2(|V|-1).$$

Then, the number of vertices of V^R is

$$\begin{aligned}|V^R| &= \sum_{v\in V}|S_v| \\ &= \sum_{v\in V}|\{v_u \mid u\in Neighbors(v,T)\}\cup v_{\mathrm{con}}| \\ &= \sum_{v\in V}|\{v_u \mid u\in Neighbors(v,T)\}| + |V| \\ &= 2(|V|-1)+|V| = 3|V|-2\end{aligned}$$

Lemma 9. *Assume a distribution of $|V|$ packets at the vertices of a tree $T=(V,E)$ which satisfies the requirement that no two packets compete for the same edge in a given direction, i.e., there does not exist a pair of packets p and q such that $current(p)=current(q)$ and $f(p)=f(q)$. Then, the corresponding routing graph T^{R} consists of a collection of directed (toward the root) trees and a set of isolated vertices.*

Proof. We prove the lemma by showing that (i) all vertices of the routing graph T^{R} have out-degree at most 1, and (ii) T^{R} does not contain a directed cycle.

Assume that there is a vertex of T^{R}, say u_v, that has out-degree greater than 1. Then, from the way T^{R} is constructed, it is implied that there are at least two packets p and q such that $current(p)=current(q)=u$ and $f(p)=f(q)=v$. Since the lemma states that no pair of such packets exists, we conclude that each vertex of T^{R} has out-degree at most 1.

Next we show that T^{R} does not contain a directed cycle. Notice that, each vertex of T^{R} is of the form (u_v,v_r) for some vertices u, v, and r of V (r might also stand for con). The existence of edge (u_v,v_r) in E^{R} implies that (u,v) is an edge of the original tree T. This allows us to project any path of T^{R} to a path in T. We refer to it as the *projected path.* Now, assume that T^{R} contains a directed cycle. Then, since T is a tree the projected cycle must contain a vertex v such that, for some $u\in V$, (u,v) and (v,u) are consecutive edges of the projected cycle. This, in turn, implies that the directed cycle of T^{R} contains the sub-path $(u_v,v_u)(v_u,u_x)$. However, the first edge of the path cannot belong in the routing graph since the existence of edge (u_v,v_u) in T^{R} implies that a packet from vertex u will move to vertex v and then back to u, i.e., it is derouted. □

References

[1] A. Alon, F.R.K. Chung and R.L. Graham, Routing permutations on graphs via matchings, in: *Proc. 25th Ann. ACM Symp. on the Theory of Computing*, San Diego, CA (1993) 583–591; also: *SIAM J. Discrete Math.*, to appear.

[2] F. Annexstein and M. Baumslag, A unified approach to off-line permutation routing on parallel networks, *Math. Systems Theory* **24** (1991) 233–251.

[3] S. Even, *Graph Algorithms* (Computer Science Press, Rockville, MD, 1979).

[4] F.T. Leighton, Methods for message routing in parallel machines, in: *Proc. 24th Ann. ACM Symp. on the Theory of Computing*, Victoria, BC, Canada (1992) 77–96.

[5] A. Symvonis and J. Tidswell, A new approach to off-line packet routing. Case study: 2-dimensional meshes, in: *Proc. the 1992 DAGS/PC Symp.*, Dartmouth Institute for Advanced Graduate Studies in Parallel Computation Hanover, NH (1992) 84–93.