Routing on Trees

Antonios Symvonis

Basser Department of Computer Science
University of Sydney
Sydney, N.S.W. 2006
Australia
symvonis@cs.su.oz.au

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. Moreover, we describe an on-line algorithm that completes the routing of any permutation in $n-1$ routing steps by using at each vertex v buffering area of size at most $d^2(v)$ where $d(v)$ is the degree of vertex v. Our results provide upper bounds on the number of routing steps required to route a permutation on an arbitrary connected graph G since the routing can be done by using only the edges of a spanning tree of G.

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 (queuesize), 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 is the exchange of the packets at the endpoints of an edge of graph G. The exchange of the 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. As a consequence, $rt(G) < 3n$ for any graph G of n vertices.

A lot of work has been devoted to the study of packet routing problems. As is natural, several routing models have been considered. However, most of the papers in the literature [2, 4, 5, 6, 7,

8, 10, 11, 12, 13, 15, 17, 18] consider the model in which, at any time step, all edges can carry a packet (bidirectional edges can carry one packet for each direction). Upfal [16] considered the model in which, at each step, each processor can either send or receive and only along one communication link. Meyer auf der Heide, Oesterdiekhoff and Wanka [9] considered the model in which each processor can receive at most one packet per step. Note that the above list of references is in no way complete.

The matching routing model suffers from some serious disadvantages when compared with the routing models in the papers mentioned above. It does not allow for more than one packet to be present in a vertex at any time, and, as a consequence, queues cannot be modeled. Moreover, it does not allow for the consumption of the packets when they arrive at their destination. A packet which has arrived at its destination, say vertex v, will be derouted away from it by another packet which has to cross v.

In this paper, we will consider the routing model in which 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 packets competes for the same communication link. A different way to describe this restriction is to say that with each communication link there is associated a buffer that can hold at most one packet. The routing schedule has to assure that this buffer is never overloaded. Since there is no chance that two packets will compete for the same edge, it is fair to say that during the routing queues are not created. We will refer to it as the *simplified routing model* and we will denote the routing number of graph G with respect to the simplified routing model by $rt'(G)$.

The rest of the paper is organized 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. We also describe an algorithm that produces a routing schedule that needs $O(n \log n)$ bits for its description in $O(n^3)$ time. However, that algorithm makes a slightly different assumption regarding the queue allowed at each vertex. In Section 4, an on-line algorithm that achieves the same number of routing steps is derived. In contrast with the first off-line algorithm of Section 3 that does not need any buffering area, the on-line algorithm uses at each vertex v buffering area of size at most $d^2(v)$ where $d(v)$ is the degree of vertex v. We conclude in Section 5 with further research that has to be done in this area.

2   Preliminaries

A *finite directed graph* $G = (V, E)$ is a structure which consists of a finite set of vertices V and a finite set of edges $E = \{e_1, e_2, \ldots, e_{|E|}\}$. Each edge is incident to the elements of an ordered pair of vertices (u, v). u is the *start-vertex* of the edge and v is its *end-vertex*. Occasionally, we refer to the vertex and the edge sets of graph G by $V(G)$ and $E(G)$, respectively.

Edges with the same start and end-vertices are called *self-loops*. We define the directed *self-loop augmented graph* $G^{SL} = (V, E')$ of $G = (V, E)$ to be the graph with $E' = E \cup \{e^v = (v, v) | v \in V\}$ (one self loop is added for each vertex in G provided that it does not already exist).

A *directed path* is a sequence of edges $e_1, e_2, \ldots$ such that the end-vertex of e_{i-1} is the start-vertex of e_i. A directed path with no repeated vertex is called a *simple* directed path. A directed path in which the start-vertex of its first edge is the same with the end-vertex of its last edge is called a *directed cycle*.

If we ignore the direction of the edges of a finite directed graph, we get a *finite undirected graph*. The notions of *path, simple path*, and *cycle* are defined similarly. A *tree* is an undirected graph with no cycles and exactly $|V| - 1$ edges.

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

A *permutation packet routing problem* R is defined by a pair (G, π) where $G = (V, E)$ is

the directed graph that represents the network in which the routing will take place (vertices in V represent processors and edges in E represent unidirectional communication links) and π is the permutation to be routed. Formally, the set P of $|V|$ packets to be routed is defined by $P = \{p_1, p_2, \ldots, p_{|V|} \mid p_i = (i, \pi(i)), \ i, \pi(i) \in V, \ 1 \leq i \leq |V|\}$. A more general definition that incorporates the maximum allowed queuesize was given in [15]. Note that, even though the informal definition of most routing models involves undirected graphs with bidirectional communication links, a corresponding directed graph can be easily defined.

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. We do that in order to make it possible to incorporate self loops in the directed paths. 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

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

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."*

One important property that trees possess is that there is a unique simple path between any pair of vertices in the tree. 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.

3 The Routing Number of Trees with Respect to the Simplified Routing Model

In this section we will describe two algorithms that compute off-line solutions to the routing problem on trees (with respect to the simplified model) that need at most $n - 1$ routing steps. Note that, in the case that the tree is simply a chain of n nodes, this is optimal. The time complexity of the first algorithm will be $O(n^2)$ while for the second one it will be $O(n^3)$. However, the total space (in bits) required to report the off-line solution by the first algorithm is $O(n^2)$, in contrast to $O(n \log n)$ required by the second.

3.1 An $O(n^2)$ Off-line Routing Algorithm for Trees

Before we proceed with the description of the algorithm, we need to define some notation. Let $T = (V, E)$ be the tree in which the routing will take place. V is the vertex set of T and E its edge set. It holds that $|E| = |V| - 1$.

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

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

("con" stands for "consumed"). 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
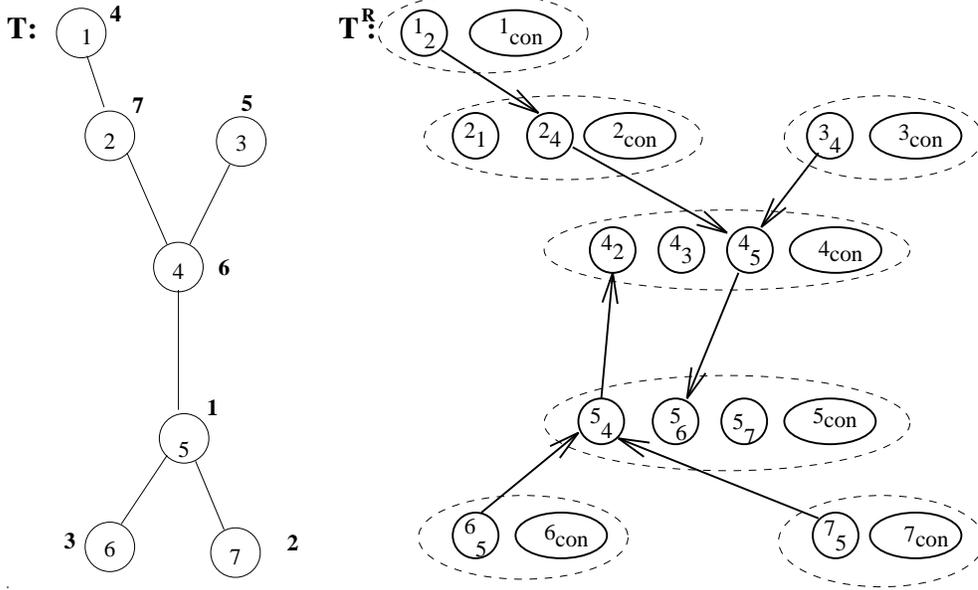
Figure 1: The numbers next to the nodes of the tree T represent the destination of the packet located in that node. Graph T^R is the routing graph which corresponds to tree T and the permutation to be routed.

happening on a tree, for each packet p, there is a unique simple (with no repeating vertices) path $path(current(p), dest(p))$ from $current(p)$ to $dest(p)$. We denote by $f(p)$ the first node 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 hasn't reached its destination after i routing steps (stages). The edge that corresponds to packet p, denoted by $edge(p)$, is defined as follows:

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

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

The following lemmata regarding the routing graph are useful for the design and the time analysis of the off-line routing algorithm.

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

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

$$|V^R| = \sum_{v \in V} |S_v| = \sum_{v \in V} |\{v_u \mid u \in Neighbors(v, T)\} \cup v_{con}| =$$

$$= \sum_{v \in V} |\{v_u \mid u \in Neighbors(v, T)\}| + |V| = 2(|V| - 1) + |V| = 3|V| - 2$$

∎

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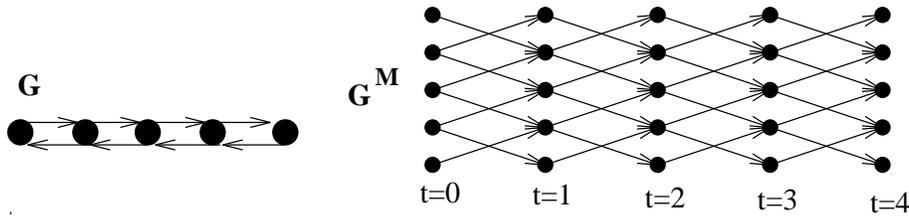ent(p) = current(q)$ and $f(p) = f(q)$. Then, the corresponding routing graph T^R consists of a collection of directed (toward the root) trees and a set of isolated vertices.*

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 exist, we conclude that each vertex of T^R has outdegree 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^1. 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 subpath $(u_v, v_u)(v_u, u_x)$. 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. ∎

Consider any rooted at vertex v tree T_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 Figure 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 corresponds to only one path (arbitrary chosen) connecting v (the root of T_v) with one of its leaves. We can choose the packets which will move during the next step (and also notify the one that will not) by a simple traversal of the tree (in the opposite direction from that indicated by the edges) in $O(|V(T_v)|)$.

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, π)*
/* π 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 haven't 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 structure that keeps track of the current position and the journey of each packet.

     (d) $i = i + 1$

Since at each iteration of the while-loop of Algorithm *Off_line_tree_routing_1(T, π)* at least one packet is advanced towards its destination and no packet is routed away from its destination, the total distance that all packet have to travel always reduces. This implies that after at most $O(n^2)$ steps the routing is completed. In the following lemma, we provide a better upper bound on the number of times the while-loop in algorithm *Off_line_tree_routing_1(T, π)* is executed.

---

[1] r might also stand for "con".

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

Lemma 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 $n - 1$ routing steps.*

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 doesn't 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 contains a packet that wants to move. Assume that packet p didn't 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 Figure 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. ∎

Corollary 1 *For any tree T of n vertices, $rt'(T) < n$.*

When we have to solve a routing problem on an arbitrary graph G instead of a tree, we can ignore all edges of G but those that form a spanning tree of G. We conclude that:

Corollary 2 *For any graph G of n vertices, $rt'(G) < n$.*

From Lemma 1 we know 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 Lemma 3 ensures that 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 where, "0" denotes that

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

the packet waits in a vertex while, "1" denotes that it moves. Since each packet will arrive at each destination after at most $n - 1$ steps, the array for each packet will have at most $n - 1$ entries. So, to report the solution of the routing problem we need space of size $O(n^2)$ bits. Notice that, the above analysis assumes that we know the path that each packet will follow. It is not difficult to precompute the paths in $O(n^2)$ time by using tree traversal methods.

The following theorem summarizes the results of this section.

Theorem 1 *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.*

3.2 A More Compact Solution

In this section, we present an algorithm based on the *multistage off-line routing method* introduced by Symvonis and Tidswell in [15]. Given a permutation π to be routed on tree T, we produce a routing schedule of at most $n - 1$ routing steps that can be described with $O(n \log n)$ bits. A small difference between the routing model assumed in this section and the one assumed in Algorithm *Off_line_tree_routing_1(T, π)* exists. In this section, we assume that the packet that originates at a given vertex can be stored at that vertex at no extra cost. This is equivalent to assuming a queue of size 1 packet at each vertex, but, only the packet initially stored at the vertex can use it.

Before we proceed with the algorithm, some definitions are necessary. Consider any routing problem $R = (G, \pi)$ where, $G = (V, E)$ is the directed graph (with no self-loops) that represents the interconnection network in which the routing takes place and π is the permutation to be routed . Assume a trivial upper bound of B routing steps for the problem under consideration (for the cases of trees or general graphs of n vertices, we set $B = 2n$).

We construct a multistage directed graph $G^M = (V^M, E^M)$ as follows:

$$V^M = \{(v, t) | \ v \in V \text{ and } 0 \le t \le B\}$$

and

$$E^M = \{((v, t), (w, t + 1)) \ | \ w \in neighbors(v, G) \text{ and } 0 \le t < B\}$$

The edges of E^M represent the communication that can take place between adjacent vertices of the interconnection network at any time.

Figure 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. Note that, for a graph $G = (V, E)$ the multistage graph has $O(|V|^2)$ vertices and $O(|VE|)$ edges.

Let $tower(G^M, v)$ be the set of vertices of graph G^M (the constructed multistage graph) that corresponds to vertex v in G. Formally,

$$tower(G^M, v) = \{(v, t) | \ v \in V, (v, t) \in V^M, 0 \le t \le B\}.$$

Theorem 2 *Let $R = (G, \pi)$ be a routing problem. R has a solution of length L if and only if for each packet p there exist a path from a vertex $(orig(p), t)$ in $tower(G^M, orig(p))$ to vertex $(dest(p), t')$ in $tower(G^M, dest(p))$, $t \le t' \le L$ and all such paths are mutually edge disjoint.*

Proof See [15] ∎

Theorem 2 states that the problem of obtaining an off-line solution to a routing problem reduces to the problem of finding edge disjoint paths in the directed multigraph G^M. This allows us to approach the off-line packet routing problem from a different point of view.

We are now ready to present a high level description of our algorithm.

Algorithm *Off_line_tree_routing_2(T, π)*
/* π 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 packets will be specified later.[2]

3. $P \neq \emptyset$ **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 node. 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 4 *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*[3]. 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 ($start = 0 \cdots n - 1$) 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 4 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.

---

[2] Different orderings result to routing schedules of different lengths.
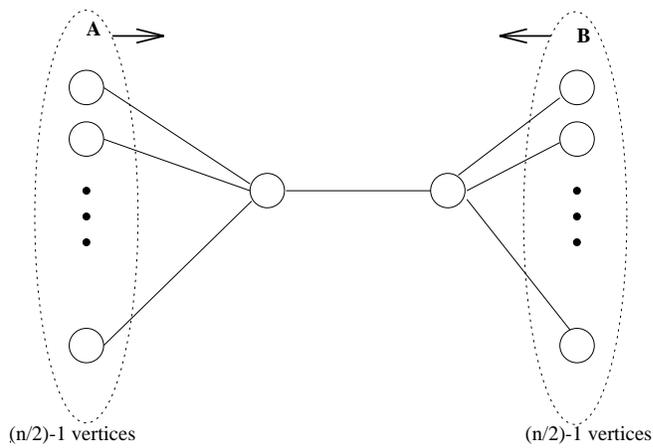[3] Note that this definition of *delay* is different from that used in the proof of Lemma 3.

Figure 4: An example in which the queues created during the execution of the trivial greedy on-line routing algorithm can grow as large as $\frac{n}{2} - 2$ packets.

The fact that the packets were routed in an arbitrary order, resulted in a routing schedule of at most $2n - 1$ packets in the worst case. The following lemma shows that by being more careful in the order we route the packets we can get a significantly better solution.

Lemma 5 *Assume a tree T of n vertices and a permutation π on its vertex set that has to be routed. Then, there exists an ordering of the packets such that, Algorithm* Off_line_tree_routing_2(T, π) *produces a routing schedule of at most $n - 1$ steps. Moreover, this ordering can be computed in $O(n^2)$ amortized time.*

Proof See [14] ∎

The following theorem summarizes the results of this section.

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_2(T, π) *produces an off-line routing solution of at most $n - 1$ steps which can be described with $O(n \log n)$ bits, in $O(n^3)$ time. The routing model used assumes that each processor can store its original packet at no extra cost.*

4   An On-line Routing Algorithm

In this section, we present an on-line algorithm that completes the routing of a permutation on a tree of n vertices in at most $n - 1$ steps. Denote by $d(v)$ the degree of vertex v. In contrast with the algorithms presented in the previous section, a buffering area of size $d^2(v)$ packets is required in each vertex v of the tree. Our method requires that additional information of $d(v)$ bits is attached to each packet. For this reason, our method is suitable for trees in which the maximum degree is bounded from above by a constant.

Consider first the trivial greedy on-line algorithm in which, at each time step, each processor tries to advance as many packets as possible towards their destination, while, the remaining packets are queued in the processor. It is not difficult to see that this simple algorithm will complete the routing in $n - 1$ steps. At the same time, the queue at some processor might become as large as $\frac{n}{2} - 2$. This can happen in the tree of Figure 4 when all the packets initially in the set (of processors) A are destined for the set (of processors) B and vice versa.

We will modify the above greedy algorithm and make it use queues of size at most $d^2(v)$ packets in any vertex v. For trees of maximum degree bounded by a constant, this yields an algorithm that uses constant size queues. A similar method was used by Symvonis and Makedon in [8] where an asymptotically optimal constant queue routing algorithm was developed for the *many-to-one*

packet routing problem on 2-dimensional meshes. In each vertex v, we associate with each of the $d(v)$ outgoing communication links (edges) a queue of size $d(v)$. Consider any edge (v, u). At most $d(v) - 1$ neighbors of v can send packets that want to cross edge (v, u) (remember that we never deroute a packet). So, in order to avoid overloading the queue associated with edge (v, u) we allow the neighbors of v to send packets that want to cross that edge only when the associated queue contains at most 1 packet.[4] We must inform the neighbors of v when the queue associated with (v, u) can receive packets. We do this by having at each step neighboring vertices exchanging information about the status of their queues. Vertex v has to transmit $d(v)$ bits of information to each of its neighbors. A "0" ("1") bit disables (enables) the transmission of packets that want to cross a specific edge. This information has to be transmitted even when no packet has to cross the edge that connects two neighbors. In general, the messages exchanged between vertices consist of two fields; the original packet and the information regarding the status of the queues.

We now give a high level description of the on-line algorithm.

Algorithm $On\_line\_tree\_routing(T, \pi)$
/* π is the permutation to be routed on tree T */

For each processor, repeat until the routing of all packets is completed

1. From the information received in the previous step regarding the queues of its neighbors, the processor decides which packets to transmit.

2. By checking the number of the remaining packets in each of its queues, the processor determines which of its queues can receive packets during the next step. Based on that information, a string of d bits is created[5].

3. The processor transmits the packets it selected in Step 1 concatenated with the binary string created in Step 2.

Theorem 4 *Assume a tree T of n vertices and a permutation π on its vertex set that has to be routed. Then, Algorithm* On_line_tree_routing(T, π) *completes the routing in at most $n - 1$ steps and by using queues of size $d^2(v)$ at each vertex v.*

Proof The statement regarding the number of steps can be proved in a way similar to that of Lemma 3. The statement regarding the queuesize is obvious. ∎

By performing the routing along the edges of a spanning tree of a graph G, we prove that:

Corollary 3 *Assume a graph G of n vertices and maximum degree d, and a permutation π on its vertex set that has to be routed. Then, π can be on-line routed in at most $n - 1$ steps and by using queue of size at most d^2 packets.*

The fact that any spanning tree of graph G can be used for the routing of a permutation on G, suggests the problem of finding a spanning tree in which the maximum degree is minimized. Unfortunately, this problem is NP-complete as it can be shown by an elementary reduction from the *hamiltonian path* problem [3].

5   Conclusions - Further work

In this paper, motivated by the work of Alon, Chung and Graham [1], we considered the routing number of a tree T of n vertices with respect to the simplified routing model, denoted by $rt'(T)$. We

---

[4] Observe that, if we enable the transmission only when the queue associated with (v, u) is empty (and thus reduce the queue size by 1 packet per edge), the number of the required routing steps might increase.
[5] d is the maximum degree of the tree. Since the processor might have less than d neighbors, some of the bits might carry useless information.

proved that $rt'(T) \leq n-1$ by developing two algorithms that compute routing schedules of different qualities. This result is asymptotically optimal since for a chain of n vertices there exist a class of permutations that require $n-1$ steps for their routing.

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 [15] and the relation between the routing problem and the multicommodity flow problem.

References

[1] A. Alon, F.R.K. Chung and R.L. Graham, "Routing Permutations on Graphs via Matchings", in the Proceedings of the 25th Annual ACM Symposium on the Theory of Computing, San Diego, California, May 1993, pp. 583-591. To appear in the SIAM Journal on Discrete Mathematics.

[2] A. Borodin and J.E. Hopcroft, "Routing, Merging , and Sorting on Parallel Models of Computation", *Journal of Computer and System Sciences*, Vol. 30, 1985, pp. 130-145.

[3] M.R. Garey and D.S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-Completeness*, W.H Freeman and Company, San Francisco, 1979.

[4] C. Kaklamanis, Danny Krinzanc and Satish Rao, "Simple Path Selection for Optimal Routing on Processor Arrays", in the proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), San Diego, California, June 1992, pp. 23-30.

[5] M. Kaufmann, S. Rajasekaran and J.F. Sibeyn, "Matching the Bisection Bound for Routing and Sorting on the Mesh", in the proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), San Diego, California, June 1992, pp. 31-40.

[6] F.T. Leithon, B. Maggs and S. Rao, "Universal Packet Routing Algorithms", in the proceedings of the 29th Annual Symposium of Foundations of Computer Science, October 1988, pp. 256-271.

[7] F.T. Leighton, F. Makedon and I.G. Tollis, "A $2n-2$ Algorithm for Routing in an $n \times n$ Array With Constant Size Queues", in the proceedings of the 1st Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), June 1989, pp. 328-335.

[8] F. Makedon and A. Symvonis, "Optimal Algorithms for the Many-to-One Routing Problem on 2-Dimensional Meshes", to appear in the *Journal of Microprocessors and Microsystems*. An extended abstract appeared in the Proceedings of the 19th Annual ACM Computer Science Conference, San Antonio, Texas, March 1991, pp. 280-288.

[9] F. Meyer auf de Heide, B. Oesterdiekhoff and R. Wanka, "Strongly Adaptive Token Distribution". To appear in the proceedings of ICALP 1993.

[10] I. Parberry, "An Optimal Time Bound for Oblivious Routing", *Algorithmica*, 1990, 5, pp. 243-250.

[11] D. Peleg and E. Upfal, "The Generalized Packet Routing Problem", *Theoretical Computer Science*, 53, 1987, pp. 281-293.

[12] D. Peleg and E. Upfal, "The Token Distribution Problem", *SIAM Journal on Computing,* Vol. 18, No. 2, 1989, pp. 229-243.

[13] S. Rajasekaran and R. Overholt, "Constant Queue Routing on a Mesh", *Journal of Parallel and Distributed Computing*, Vol. 15, 1992, pp. 160-166.

[14] A. Symvonis, "Routing on Trees" Technical Report 471, Department of Computer Science, University of Sydney, September 1993.

[15] A. Symvonis and J. Tidswell, "A New approach to Off-Line Packet Routing. Case Study: 2-Dimensional Meshes", Proceedings of the 1992 DAGS/PC Symposium, Dartmouth Institute for Advanced Graduate Studies in Parallel Computation, June 1992, Hanover, NH, USA, pp. 84-93.

[16] E. Upfal, "Efficient Schemes for Parallel Communication", in ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, August 1982, pp. 55-59.

[17] L.G. Valiant, "A Scheme for Fast Parallel Communication", *SIAM Journal on Computing,* Vol. 11, No. 2, 1982, pp. 350-361.

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