Direct Routing

Costas Busch* Malik-Magdon Ismail† Marios Mavronicolas‡ Paul Spirakis§

July 11, 2003

Abstract

Direct routing is a special case of bufferless routing in which packets are not allowed to conflict with each other. The task is to compute the injection times of the packets so that they don't conflict. A well known lower bound on the routing time of any algorithm is $\Omega(C + D)$, where the *congestion* C is the maximum number of paths that use any edge, and the *dilation* D is the maximum length of any path. We study the extent to which direct routing algorithms can achieve this lower bound.

We present a simple *greedy* direct routing algorithm for arbitrary routing problems that has routing time $O(C \cdot D)$. We show that this routing time is worst case optimal. In particular, we construct a "hard" routing problem on the mesh, for which *any* direct routing algorithm has routing time $\Omega(C \cdot D)$, while $C + D = \Theta(\sqrt{C \cdot D})$.

We then consider many interesting routing problems on commonly used network topologies. We show that variants of the simple greedy algorithm achieve optimal routing time. The routing problems and corresponding routing times (rt) are:

i. Trees: arbitrary routing problems on arbitrary tree topologies ($rt = O(C + D)$).

ii. Mesh with n nodes: permutations ($rt = O(\sqrt{n})$); arbitrary one-bend paths ($rt = O(C + D)$).

iii. Butterfly with n inputs: random destinations and permutations ($rt = O(\lg n)$ w.h.p.).

iv. Hypercube with n nodes: random destinations and permutations ($rt = O(\lg n)$ w.h.p.).

*Computer Science Department, Rensselaer Polytechnic Institute, 110 8th Street, Troy, NY 12180, USA. Email: buschc@cs.rpi.edu

†Computer Science Department, Rensselaer Polytechnic Institute, 110 8th Street, Troy, NY 12180, USA. Email: magdon@cs.rpi.edu

‡Department of Computer Science, University of Cyprus, P. O. Box 20537, Nicosia CY-1678, Cyprus. Email: mavronic@ucy.ac.cy

§Department of Computer Engineering and Informatics, University of Patras, Rion, 265 00 Patras, Greece, & Computer Technology Institute, P. O. Box 1122, 261 10 Patras, Greece. Email: spirakis@cti.gr

1 Introduction

Packet routing is the general task of delivering a set of packets from their sources to their destinations. Here, we consider *direct routing*: packets follow specified paths; packets cannot be buffered at intermediate nodes along the path; and, packets are not allowed to conflict with each other. Thus, packets proceed "directly" from their source to their destination. Direct routing is especially appropriate when buffers are costly or unavailable (for example optical networks [20]), or collisions cannot be tolerated (for example power aware routing in sensor networks [8]) or packet delivery times must be guaranteed (for example real-time systems applications).

The task is to compute a valid *routing schedule*, which is a set of injection times such that if the packets enter the network at their prescribed times, then they will follow their prespecified paths without conflicts. The *routing time* is the time at which the last packet is absorbed at its destination. In general we would like to minimize the routing time. We assume that the nodes are synchronous, i.e. a global clock defines a discrete time. At most two packets may use a link per time step (at mose one in either direction of the link). We measure the efficiency of a direct routing algorithm with respect to the *congestion* C (the maximum number of packets that use an edge) and the *dilation* D (the maximum length of any path). A well known lower bound on the routing time of any routing algorithm (direct or not) is given by $\Omega(C+D)$. It is desirable to design routing algorithms with routing time close to this lower bound. For *store-and-forward* routing, in which nodes have buffers for storing packets in transit, there are routing algorithms with performance close to the lower bound [4, 12, 13, 15, 17, 19]. However, such algorithms are not applicable when no buffers are available.

Here, we provide a comprehensive study of direct routing, paying particular attention to its efficiency class with respect to C and D. In order to guarantee that the packets will be routed without conflicts, a direct routing algorithm must necessarily be *offline* – some centralized resource with complete information regarding the packet paths computes the packet injection times. Thus, another consideration is the *offline time* of a direct routing algorithm, i.e. the amount of time the centralized resource takes to compute the routing schedule.

Contributions. We present results for the general direct routing problem, as well specific classes of problems on many common network architectures. The architectures we consider are the *tree, mesh, butterfly* and *hypercube* (Figure 1).

All our direct routing algorithms are based on a simple *greedy* approach, which considers one packet at a time and assigns it the earliest available injection time. For an arbitrary routing problem, we show that the routing time of this greedy algorithm is $O(C \cdot D)$. We show that for direct routing, this is worst case optimal. More specifically, we expicitly construct a "hard" routing problem on the mesh network for which $C + D = \Theta(\sqrt{C \cdot D})$. We then show that *every* direct routing algorithm will have a routing time that is $\Omega(C \cdot D)$ for this routing problem. Since $C \cdot D$ is strictly worse that $C + D$ for this hard routing problem, we see that direct routing is significantly harder than routing with buffers, in the general case.

We show that variants of the greedy algorithm can be used to route *optimally* for many interesting network topologies. Thus, for these topologies, when it comes to offline routing, buffers offer no significant advantage and would be a wastage of resources. The routing times (rt) achieved are summarized below.

i. Trees: for arbitrary routing problems on arbitrary tree topologies, $rt \leq 2C + D - 2$.

ii. Mesh with n nodes: in the worst case, $rt = \Omega(C \cdot D)$ for *any* algorithm. For permutation routing, $rt \leq 4\sqrt{n} - 2 + D$ using the simple greedy algorithm – we obtain this upper bound using only one-bend paths (paths consisting of at most two line segments); in the worst case, $D = \Omega(\sqrt{n})$ so this is optimal. For arbitrary one-bend paths, $rt \leq 8C + 4D - 8$ using a variant of the greedy algorithm.

iii. Butterfly with n inputs and one packet per input: for random destinations, $\mathbf{P}\left[rt \leq \frac{5}{2}\lg n\right] > 1 - 3n^{-\frac{1}{2}}$. For permutation routing, $\mathbf{P}\left[rt \leq 5\lg n\right] > 1 - 6n^{-\frac{1}{2}}$. Since $D = \Omega(\lg n)$, these results are asymptotically optimal with high probability (w.h.p.).

iv. Hypercube with n nodes and one packet per node: for random destinations, $\mathbf{P}\left[rt \leq 7\lg n\right] > 1 - \frac{1}{32n}$. For permutation routing, $\mathbf{P}\left[rt \leq 14\lg n\right] > 1 - \frac{1}{16n}$. Since $D \geq \frac{1}{4}\lg n$ w.h.p, these results are asymptotically optimal w.h.p.

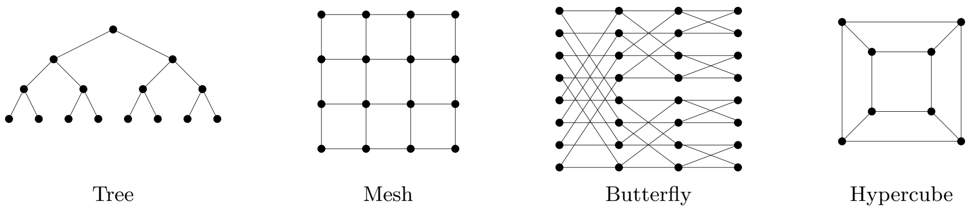


Figure 1: Interesting network topologies

In summary, we present a comprehensive treatment of the direct routing problem in terms of congestion and dilation. During the course of our analysis, we use techniques that may be of independent interest. We make extensive use of the *dependency graph* in which packets are nodes and two packets are adjacent if their paths may conflict. We link many properties of a direct routing problem to properties of the corresponding dependency graph. As a final note, we mention that the offline computation time of all our algorithms is polynomial in n (the nework size), N (the number of packets), C and D.

Related Work. The only previous work known on direct routing is for trees. In particular, Symvonis [21] and Alstrup et al. [2] study permutations on trees and give routing algorithms with routing time $O(n)$ for any tree with n nodes. This time is optimal in the worst-case, since there are trees with permutations that require $\Omega(n)$ routing time. However, there exist many permutation routing problems where $C + D = o(N)$, for which our tree algorithm is strictly better. For example, consider the tree with a root and $n-1$ children, and any permutation routing on this tree. Then $C = 1$ and $D = 2$, and for our algorithm, all packets are injected at time 0. Therefore, the routing time is 2 time steps, compared with the routing time $\Theta(n)$ of the algorithm in [2].

Other models of bufferless routing are *matching routing* and *hot-potato routing*. In the matching model, at each time step, a set of edges with disjoint end points is chosen and packets at the endpoints of each selected edge are exchanged. Matching algorithms have been studied for permutation problems on various network topologies, [1, 18, 24]. In hot-potato routing, when packets conflict they get deflected, i.e. use alternative paths to their destination. Hot-potato algorithms have been studied in [3, 5, 7, 9, 10] for various network topologies.

In terms of congestion and dilation, there has been extensive research for store-and-forward algorithms in attaining the $O(C + D)$ routing in arbitrary networks [12, 13, 15, 17, 19]. There are also hot-potato algorithms which come close to the lower bound $\Omega(C + D)$ for leveled networks [6] and vetrex symmetric networks [14].

Paper Outline. In the remainder of the paper, we proceed as follows. In the next section, we give useful definitions and preliminary results. In Section 3, we introduce the dependency graph and in section 4, we present the "hard" routing problem. In Section 5, we present the greedy algorithm. We then present results for trees, meshes, butterflies and hypercubes in Sections 6, 7, 8 and 9. We end with some concluding remarks in Section 10.

2 Preliminaries

Problem Definition. Let's first define the problem and introduce the definitions that we will subsequently need. We are given a graph $G = (V, E)$ with $n \geq 1$ nodes, and a set of packets $\Pi = \{\pi_i\}_{i=1}^{N}$. Each packet π_i is to be routed from its source, $\mathrm{s}(\pi_i) \in V$, to its destination, $\delta(\pi_i) \in V$, along a pre-specified path p_i. Thus, there are N packets to be routed on a graph with n nodes. For simplicity, we will consider acyclic packet paths (all our results hold also for paths with cycles).

Consider two paths $p_1 = (v_1, v_2, \ldots, v_k)$ and $p_2 = (u_1, u_2, \ldots, u_l)$. The paths p_1 and p_2 *collide* if they share an edge in the same direction, i.e., there is some pair of indices i, j such that $(v_i, v_{i+1}) = (u_j, u_{j+1})$, with $1 \leq i < k$ and $1 \leq j < l$ (we also say that π_1 and π_2 collide). Two packets *conflict* if they are routed in such a way that they appear in the same node at the same time, *and* the next edge in their paths is the *same*. The length of a path p, denoted $|p|$, is the number of edges in the path. The *distance* between two nodes, is the length of a shortest path that connects the two nodes.

The task is to determine the injection time τ_i for each packet $\pi_i \in \Pi$, such that if packet π_i is injected into node $\mathrm{s}(\pi_i)$ at time τ_i, it will follow the path to its destination $\delta(\pi_i)$ without any conflicts. A *direct routing problem* has the following components. *Input:* (G, Π, P), where G is a multi-graph, and the packets $\Pi = \{\pi_i\}_{i=1}^N$ have respective paths $P = \{p_i\}_{i=1}^N$. *Output:* The injection times $\mathcal{T} = \{\tau_i\}_{i=1}^N$, denoted a *routing schedule* for the routing problem. *Validity Requirement:* If each packet π_i is injected at its corresponding time τ_i into its source s_i, then it will follow a conflict-free path to its destination where it will be absorbed at time $t = \tau_i + |p_i|$. Such a routing schedule is called *valid*.

Consider a routing problem (G, Π, P). The *routing time* $rt(G, \Pi, P)$ is the maximum time at which a packet gets absorbed at its destination, $rt(G, \Pi, P) = \max_i\{\tau_i + |p_i|\}$. The *offline time*, $ol(G, \Pi, P)$ is the number of operations used to compute the routing schedule $\mathcal{T}$.

Let $d_p(e)$ denote the number of edges along a path p that need to be traversed in order to reach an edge e. Two packets π_1, π_2 are *synchronized* if their paths p_1, p_2 collide at some edge e and $d_{p_1}(e) = d_{p_2}(e)$. Thus, if injected at the same time, they will conflict. More generally, a set of packets is synchronized if every pair of packets in the set is synchronized. Let $\pi_1, \pi_2 \in \Pi$ be two packets with paths p_1, p_2 that collide at an edge e. If τ_1 and τ_2 are the injection times of π_1 and π_2, then in a valid direct routing schedule, it must be that $|\tau_1 - \tau_2 + d_{p_1}(e) - d_{p_2}(e)| > 0$.

Shortcut-Free Paths. Let $\mathcal{E}$ be the set of all edges used by the paths in P. A *spliced path* from node v_s to v_d of G, is a path $(v_s, w_1, w_2, \ldots, w_j, v_d)$ such that every edge along this path belongs to $\mathcal{E}$. In other words, a spliced path from v_s to v_d is made up of segments of the different paths in P spliced together. Let $p = (v_1, v_2, \ldots, v_k)$ be a path in P, and let $\mathcal{S}$ be the set of all spliced paths from v_1 to v_k using the paths in P. In particular, $p \in \mathcal{S}$. We say that path p is *shortcut-free* in P if the length of every path in $\mathcal{S}$ is at least the length of p, i.e., there is no shorter path from v_1 to v_k that can be obtained by using parts of the other paths. We say that the paths in P are *short-cut free* if every path $p \in P$ is shortcut-free in P.

One example of shortcut-free paths are a collection of shortest paths between pairs of vertices on a graph. However, it is easy to see that shortcut-free paths are not restricted to shortest paths. Note the two paths in a set of short-cut free paths may collide at mupliple edges. If the paths are shortcut-free, in order to construct a valid routing schedule, two packets need only be desynchronized at the first (or any one) edge on which they collide:

Lemma 2.1 *Let Π be a set of packets with shortcut-free paths P. Suppose that $p_1, p_2 \in P$ collide, and let (v, w) be any particular edge at which the paths collide. Then the respective packets π_1 and π_2 do not conflict (on any edge) if and only if π_1 and π_2 reach v at different time steps.*

Proof: The only if direction is clear. We consider the if direction, i.e. suppose to the contrary that π_1 and π_2 reach v at different times but conflict at some other edge (v', w'). So both packets reach v' at the same time. If (v', w') is earlier than (v, w) in one of the paths then the packets take different times to reach v from v' hence one of the paths from v' to v is shorter contradicting the fact that the paths are shortcut-free. On the other hand, if (v', w') is later than (v, w) on one of the paths, then the packets take different times to get from v to v' which means that one of the paths from v to v' is shorter, contradicting the fact that the paths are shortcut-free. In either case we get a contradiction, so (v', w') cannot exist as claimed. ■

3 Dependency Graphs

Consider a routing problem (G, Π, P). The *dependency multi-graph* $\mathcal{D}$ of the routing problem is a graph such that each packet $\pi_i \in \Pi$ is a node in the graph. There is an edge between two packets in $\mathcal{D}$ for every edge on which their paths collide. Corresponding to the dependency multi-graph is the *dependency graph* $\overline{\mathcal{D}}$ in which all the edges between two vertices are collapsed into one edge. Thus, there is an edge between packets in $\overline{\mathcal{D}}$ if and only if their paths collide at least once.

A clique $\mathcal{K}$ in $\overline{\mathcal{D}}$ is synchronized if all the packets in $\mathcal{K}$ are synchronized. No pair in a synchronized clique can have the same injection time, as otherwise they would conflict. Thus, the size of the maximum synchronized clique in $\overline{\mathcal{D}}$ gives a lower bound on the routing time:

Lemma 3.1 *Let $\mathcal{K}$ be a maximum synchronized clique in the dependency graph $\overline{\mathcal{D}}$. Then, for any routing algorithm, $rt(G, \Pi, P) \geq |\mathcal{K}| + |p_{max}|$, where p_{max} is the maximum path length of the packets in $\mathcal{K}$ and $|\mathcal{K}|$ is the number of nodes in $\mathcal{K}$.*

The dependency graph also gives us a lower bound on the offline computation time. If two packets collide at an edge, then any algorithm must check that they are desynchronized at that edge. This implies that every edge in the dependency multi-graph graph $\mathcal{D}$ has to be examined at least once. We have:

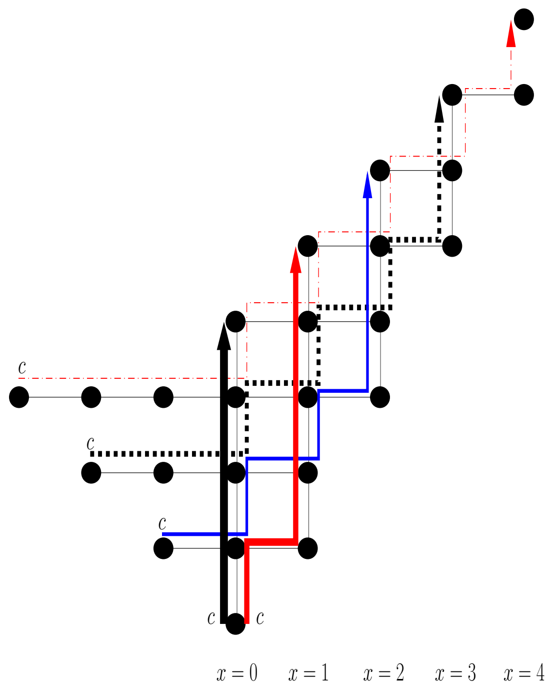
Lemma 3.2 *Any direct routing algorithm has offline computation time which is at least the number of edges in the dependency multi-graph $\mathcal{D}$.*

4 A Hard Routing Problem

Here, we show the existence of hard direct routing problems.

Theorem 4.1 *For every direct routing algorithm, there exist routing problems for which the routing time is $\Omega(C \cdot D)$, and $C + D = \Theta(\sqrt{C \cdot D})$.*

Proof: We give an explicit construction of a routing problem for which the number of packets is $N = \Theta(C \cdot D)$ and all the packets form a synchronized clique of size N in the dependency graph $\overline{\mathcal{D}}$. Then we can apply lemma 3.1 to obtain the $\Omega(C \cdot D)$ lower bound. The routing problem is illustrated in the figure on the right, which describes a routing problem on a mesh (for a description of the mesh network see Section 7). There are L levels in this routing problem. Each path in the figure (ending with an arrow) is actually the path for c packets. The anchor path is the vertical path of length L ($L = 4$ in the figure). Each path can be associated to a level, which denotes the x-coordinate at which the path moves vertically up after making its final left turn. Thus the anchor path is the level-0 path, which begins at coordinates $(0, 0)$ and ends at $(0, L)$. There is a path for every level in $[0, L]$, and so the total number of packets is $N = c(L + 1)$. The level-i path for $i > 0$ begins at $(1 - i, i - 1)$ and ends at $(i, L + i)$, and is constructed as follows. Beginning at $(1 - i, i - 1)$, the path moves right till $(0, i - 1)$, then alternating between up and right moves till it reaches level i at node $(i, 2i - 1)$ (i alternating up and right moves), at which point the path moves up to $(i, L + i)$.



We list some properties of this set of paths. Let $j > i \geq 0$. (i) The level-j path meets the level i path exactly once at the edge from $(i, i + j - 1)$ to $(i, i + j)$. Further, an edge is shared by at most 2 paths. (ii) Every packet is synchronized with every other packet, i.e., if packets π_1, π_2 follow paths p_1, p_2 which share an edge e then $d_{p_1}(e) = d_{p_2}(e)$: if the two paths are the same then there is nothing to show, otherwise, this follows from (i) and the fact that the level-i path is injected at $(1 - i, i - 1)$. Thus, if two packets are injected at the same time into two paths p_1, p_2, then they will conflict at e. (iii) The length of the level i path is $L + 2i$. The conjestion is $C = 2c$ and the dilation is $D = 3L$.

Since every pair of packets is synchronized, in the dependency graph $\overline{\mathcal{D}}$, the packets form a synchronized clique of size N. Thus, we can apply Lemma 3.1 to obtain $rt(G, \Pi, P) \geq N + D$. Since $N = c(L + 1) =$

$\frac{C}{2}(\frac{D}{3}+1)$, $rt(G,\Pi,P)=\Omega(C\cdot D)$. Choosing $c=\Theta(\sqrt{N})$ and $L=\Theta(\sqrt{N})$, we have that $C+D=\Theta(\sqrt{N})$ so $C+D=\Theta(\sqrt{C\cdot D})$, concluding the proof. ■

5 The Greedy Algorithm

We consider the following greedy algorithm.

```
// Greedy direct routing algorithm:
// Input: routing problem (G, Π, P) with N packets Π = {π_i}_{i=1}^N.
// Output: Set of injection times T = {τ_i}_{i=1}^N.
Let π_1, ..., π_N be an arbitrary ordering of the packets.
for i = 1 to N do
    Greedily assign the first available injection time τ_i to packet π_i ∈ Π so that it does not conflict with
    any packet already assigned an injection time.
end for
```

It is easy to show by induction, that no packet π_j conflicts with any packet π_i with $i<j$, and thus the greedy algorithm produces a valid routing schedule. The routing time for the greedy algorithm will be denoted $rt_{Gr}(G,\Pi,P)$. Let deg_i be the degree of packet π_i in the dependency multi-graph $\mathcal{D}$. Next we show that $\tau_i \le deg_i$ which implies the folowing lemma:

Lemma 5.1 $rt_{Gr}(G,\Pi,P)\le \max_i\{deg_i+|p_i|\}$.

Proof: We show that the injection times assigned by the greedy algorithm satisfy $\tau_i \le deg_i$, from which the claim follows immediately. For packet i, we consider the path p_i and the interval of times $[0, deg_i]$. Every time a packet σ, that has already been assigned an injection time, uses an edge on p_i, we remove the (at most one) injection time in this set that would cause π_i to conflict with σ at the time σ uses this edge. Since deg_i is the number of times packets collide with π_i, we remove at most deg_i injection times from this set. As there are deg_i+1 injection times in this set, it cannot be empty, so the greedy algorithm must assign an injection time to π_i that is in this set, as it assigns the smallest available injection time. ■

We now give an upper bound on the routing time of the greedy algorithm. Since the congestion is C and $|p_i|\le D\ \forall i$, a packet collides with other packets at most $(C-1)\cdot D$ times. Thus, $deg_i\le (C-1)\cdot D$, $\forall i$. Therefore, from Lemma 5.1 we obtain:

Corollary 5.2 *$rt_{Gr}(G,\Pi,P)\le C\cdot D$.*

The general $O(C\cdot D)$ bound on the routing time of the greedy algorithm is worst case optimal, within constant factors, since from Theorem 4.1, there exist worst-case routing problems with $\Omega(C\cdot D)$ routing time. In the next sections, we will show how the greedy algorithm can do better for particular routing problems. For those problems, we specify a particular order in which the greedy algorithm considers the packets.

Consider now shortcut-free paths. In Lemma 2.1, we showed that in order to obtain a valid routing schedule, the packets need only be desynchronized at one of the edges on which they collide, and the packets can be guaranteed to not conflict with each other for their entire paths. This leads to the following result. Let $\overline{deg}_i$ be the degree of packet π_i in the dependency graph $\overline{\mathcal{D}}$.

Lemma 5.3 *For a shortcut-free direct routing problem, $rt_{Gr}(G,\Pi,P)\le \max_i\{\overline{deg}_i+|p_i|\}$.*

Proof: We show that the injection times assigned by the greedy algorithm satisfy $\tau_i\le \overline{deg}_i$, from which the claim follows immediately. The proof is very similar to Lemma 5.1, so we will be brief. In the interval of times $[0,\overline{deg}_i]$, every packet already assigned an injection time removes at most one injection time in this set. The difference from Lemma 5.1 is in that case, each such packet could remove a time for every time it used the path. Since there are at most $\overline{deg}_i$ such packets, at least one such time must be available for assignment to this packet by the greedy algorithm. ■

The maximum degree of any node in the dependency graph $\overline{\mathcal{D}}$ is $N-1$, since a packet may collide with at most $N-1$ other packets. Therefore, from Lemma 5.3 we obtain the following corollary.

Corollary 5.4 *For a shortcut-free direct routing problem,* $rt_{Gr}(G, \Pi, P) \leq N + D - 1$.

There are routing problems for which the $N + D - 1$ bound for shortcut-free free paths is smaller than the general $C \cdot D$ bound of Corollary 5.2. For example, in the case where all paths are shortcut-free and the same we have $C = N$, and thus, $N + D$ is smaller than $C \cdot D = N \cdot D$.

Now we discuss the offline routing time of the greedy algorithm. Each time an edge on a packets path is used by some other packet, the greedy algorithm will need to desynchronize these packets if necessary. This will occur at most $C \cdot D$ times for a packet, hence,

Lemma 5.5 *The offline computation time of the greedy algorithm is* $ol_{Gr}(G, P, D) = O(N \cdot C \cdot D)$.

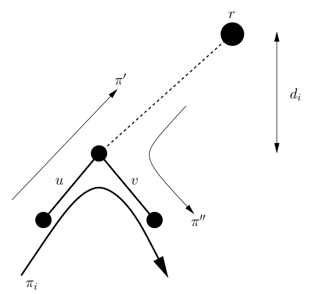
Thus, the offline time is polynomial. Further, since in the worst case, the size of the dependency multi-graph is $\Omega(N \cdot C \cdot D)$, from Lemma 3.2 we see that the greedy offline time is worst case optimal.

6 Trees

Consider the routing problems (T, Π, P), in which T is a tree with n nodes, and there are N packets $\Pi = \{\pi_i\}_{i=1}^{N}$ in which all the paths in P are shortest paths. An asymptotically optimal direct routing schedule can be computed efficiently. The algorithm is the greedy algorithm of Section 5 with the only difference being that we impose a particular order in which the packets are considered.

Let r be an arbitrary node of T. Let d_i be the closest distance that $\pi_i's$ path comes to r. The direct routing algorithm can now be simply stated as the greedy algorithm with the packets considered in sorted order, according to the distance d_i. In particular, Order the packets $\pi_{i_1}, \pi_{i_2}, \ldots, \pi_{i_N}$ so that $d_{i_1} \leq d_{i_2} \leq \cdots \leq d_{i_N}$. We show that this algorithm achieves $O(C + D)$ routing time.

Theorem 6.1 *Let* (T, Π, P) *be any routing problem on the tree* T*. Then the routing time of the greedy algorithm using the distance-ordered packets is* $rt(T, \Pi, P) \leq 2C + D - 2$.



Proof: It suffices to show that the injection time for every packet computed in the algorithm satisfies $\tau_i \leq 2C - 2$. Consider packet π_i at depth d_i as illustrated in the figure. Every packet that has already been assigned an injection time must pass through a node with depth $\leq d_i$. If such a packet could possibly conflict with π_i then it follows a path similar to either π_i, π' or π'' as shown in the figure. In all cases, these packets must therefore use one of the edges u, v, and since the congestion is C, there can be at most $2C - 2$ such packets. Consider the set of possible injection times $[0, 2C - 2]$. Each such possibly conflicting packet will remove at most one injection time from this set, so this set will be non-empty after the times conflicting with these (at most) $2C - 2$ packets are removed. Since π_i is assigned the smallest injection time that guarantees no conflict with packets that have already been assigned times, it must therefore be assigned an injection time $\tau_i \leq 2C - 2$, concluding the proof. ■

It can be shown that the offline computation time of this algorithm is $ol(T, \Pi, P) = O((C + D)N)$ which is optimal for trees (within constant factors), provided that every node knows its depth. This time is better than the general upper bound of the greedy algorithm in Lemma 5.5, since each packet need only consider two edges of its path (and not on its entire path).

7 Mesh

A mesh network M with n nodes is a $\sqrt{n} \times \sqrt{n}$ grid of nodes. Every node is connected to up to 4 of its neighbors on the grid. Since the worst-case routing problem in Theorem 4.1 is on a mesh, Corollary 5.2 imlpies that the greedy routing algorithm achieves asymptotically optimal worst case routing time. We discuss some important special cases where the situation is considerably better.

An important class of routing problems on the mesh is *permutation* routing problems where each node is the source and destination of exactly one packet (the source and the destination are different for a packet). In other words, the destinations are a permutation of the source nodes. For our study, we consider *one-bend* packet paths. We say that a packet *bends* whenever it changes direction from a horizontal link to vertical link or vice versa. We consider one-bend paths, in which every packet bends at most once. One-bend paths are trivially short-cut free, since they are shortest paths. We will show that for permutation problems, the greedy algorithm routes worst case optimally using one-bend paths. We then extend the study of one-bend permutation problems to one-bend arbitrary routing problems where once again the greedy algorithm with a specific ordering for the packets has an optimal $O(C+D)$ time. We note that this result is only optimal under the constraint of one-bend paths. Unlike with trees, it may be possible to lower the congestion by considering different paths.

One-bend Permutations Problems. Consider a permutation routing problem (M, Π, P) on a mesh M with n nodes where packets have one-bend paths. We will show that the greedy routing algorithm of Section 5, gives routing time $O(\sqrt{n})$. There are permutation problems such that the packets have to traverse a distance $D = \Theta(\sqrt{n})$ links (regardless of the one-bend path constraint). For such problems, the greedy algorithm is optimal (within constant factors). In order to show this result, we first bound the number of packets that any packet can collide with, which then gives a bound on the degree $\overline{deg}_i$ of a packet in the dependency graph $\overline{\mathcal{D}}$.

Lemma 7.1 *Let (M, Π, P) be a permutation routing problem with one-bend paths on a mesh M with n nodes. Then, $\overline{deg}_i \leq 4\sqrt{n} - 2$.*

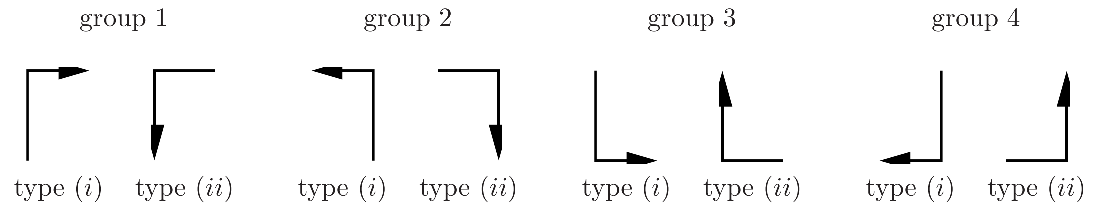
Proof: Consider the one-bend path of a packet π_i. The only other packets that can collide with π_i either have a source node or destination node on the vertical grid line of this path or the horizontal grid line of this path. There are $2\sqrt{n}$ such source nodes, and $2\sqrt{n}$ such destination nodes. Packet π_i uses one of these source nodes and one of the destination nodes. Since no two packets have the same source and no two packets have the same destination, there are $4\sqrt{n} - 2$ remaining nodes which means that at most $4\sqrt{n} - 2$ other packets can collide with this packet. ■

Since the paths are shortcut-free, and $D = O(\sqrt{n})$, using Lemma 7.1 and Lemma 5.3, we get the following theorem for the greedy algorithm:

Theorem 7.2 *Let (M, Π, P) be a permutation routing problem with one-bend paths on a mesh M with n nodes. Then, the greedy routing algorithm has a routing time $rt_{Gr}(M, \Pi, P) = O(\sqrt{n})$.*

Arbitrary Routing Problems. We will now prove a much stronger result regarding arbitrary one-bend paths, in which there is no restriction on the number of packets or on how the packets choose their sources and destinations. Namely, we will give an algorithm to direct route *any* collection of one-bend paths in $O(C+D)$ time (optimal routing time up to constant factors). Consider such a routing problem (M, Π, P), in which M has n nodes and the number of packets is N. We find an appropriate ordering of the packets so that the greedy algorithm finds the optimal routing schedule. The strategy is similar to the strategy used for the optimal tree direct routing algorithm in Section 6.

There are 8 types of one-bend paths which we group into four groups of 2 as shown below.



Notice that in any group above, a type (i) packet can never conflict with a type (ii) packet, since they are always moving in different directions. This observation gives the following lemma.

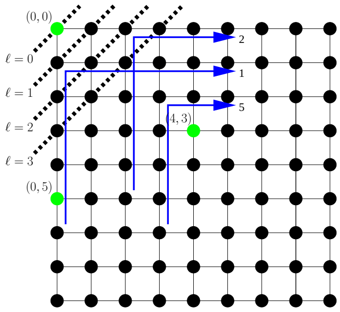
Lemma 7.3 *Let $\mathcal{T}_{(i)}$ be a set of injection times of a valid direct routing schedule for the packets of group 1(i) and let $\mathcal{T}_{(ii)}$ be a set of injection times of a valid direct routing schedule for the packets of group 1(ii). Then $\mathcal{T}_{(i)} \cup \mathcal{T}_{(ii)}$ is a valid direct routing schedule for the union of these packets. Similarily for the group 2, 3 and 4 packets.*

Lemma 7.3 allows us to determine injection times for the type (i) packets and type (ii) packets separately for each of the four groups. We can then route the four groups consecutively, in four separate phases. Suppose that we have determined the injection times of the four groups separately. Let τ_i denote the maximum injection time of packet in group i. Since the duration of phase i is at most $\tau_i + D$, we thus have the following lemma:

Lemma 7.4 *Suppose that injection times for the group* 1, 2, 3 *and* 4 *packets have been determined. Let τ_1, τ_2, τ_3, τ_4 respectively be the maximum injection time in each group. Then the packets can be delivered in time $rt(M, \Pi, P) \leq \tau_1 + \tau_2 + \tau_3 + \tau_4 + 4D$.*

Lemma 7.4 allows us to consider the groups of packets separately, and so from now on, we will focus on group 1 packets. Lemma 7.3 additionally allows to focus only on the type (i) packets. By symmetry (rotation or reflection of the mesh), we see that all the groups are essentially similar routing problems, hence the same algorithm can be used to determine route times for all the groups. We will thus describe the algorithm only for the group 1, type (i) packets.

We now describe the order in which the packets are to be considered by the greedy algorithm. Define a coordinate system for the Mesh with the origin, $\mathbf{O} = (0, 0)$, at the top left corner. The Mesh nodes all have coordinates (x, y), where $0 \leq x, y < \sqrt{n}$. The node (x, y) is x nodes right of the origin and y below it. The coordinates of some nodes indicated by lighter shading are illustrated in the figure to the right. We define levels in a network, starting with level zero, $\ell = 0$. The different levels are also illustrated in the figure to the right by the diagonal dotted lines. The level ℓ nodes are $(0, \ell), (1, \ell-1), (2, \ell-2), \ldots, (\ell, 0)$. The level of a packet is the level of the lowest level node that the packet passes through, i.e., the level of a packet is the closest distance the packet gets to the origin node. We are only considering paths of the form indicated in the figure to the right. Notice that for such paths, the level of the nodes traversed monotonically decreases by one each time to the minimum value and then monotonically increases. Some representative paths are illustrated, along with the level of the path.



The direct routing algorithm can now be simply stated as the greedy algorithm with the packets considered in sorted order, according to level. In particular, Let l_i be the level of packet π_i. Order the packets $\pi_{i_1}, \pi_{i_2}, \ldots, \pi_{i_N}$ so that $l_{i_1} \leq l_{i_2} \leq \cdots \leq l_{i_N}$. We will show the latest injection time is $O(C)$. The following lemma is the main result that will be needed.

Lemma 7.5 *Consider a packet π with level ℓ, that passes through node $(x, \ell-x)$, of level ℓ, where $0 \leq x \leq \ell$. Let e_1 be the edge that the packet uses to get to $(x, \ell - x)$, and let e_2 be the edge that the packet uses when it leaves $(x, \ell - x)$ (one of these edges may not exist). Any packet that has already been assigned an injection time that collides with this packet must use either e_1 or e_2.*

Proof: Let ℓ' be the level of any previously assigned packet π'. Then $\ell' \leq \ell$. Suppose that π' collides with π and suppose to the contrary that π' uses neither edges e_1 nor e_2. There are two cases. First suppose that π' collides with a horizontal edge of π. In this case since π' does not use e_2 and is a group 1 type (i) packet, the closest point of π' to $\mathbf{O}$ is $y, \ell - x$ where $y > x$, thus $\ell' > \ell$, contradicting the fact that $\ell' \leq \ell$. On the other hand, suppose that π' collides with a vertical edge of π. In this case, since π' does not use e_1, the closest point of π' to $\mathbf{O}$ is $(x, \ell - y)$ where $y < x$, once again contradicting the fact that $\ell' \leq \ell$, concluding the proof. ∎

Since the paths are shortcut free, Lemma 2.1 implies that when packet π is considered by the algorithm, in determining the injection time for π, we need only consider previously assigned packets π' that use at least one of the edges e_1 or e_2. It is sufficient to desynchronize π from π' at one of e_1 or e_2, which allows us to route the packets efficiently, as claimed in the following theorem.

Lemma 7.6 *Let τ_i be the injection times assigned by the greedy algorithm using the level-ordered packets. Then $\tau_i \le 2C - 2$.*

Proof: When the algorithm assigns an injection time to π_i there are at most $2C-2$ other packets that can use the edges e_1, e_2 as defined in Lemma 7.5. Consider the interval of injection times $[0, 2C-2]$. Each of these packets will remove at most one of these injection times, so after each of thes packets are considered, there must be at least one time left in this set for packet π_i. ∎

Combining Lemma 7.6 with Lemma 7.4, we get the following theorem.

Theorem 7.7 *Let (M, Π, P) be a one-bend routing problem on the mesh M. Then the routing time of the greedy algorithm using the level-ordered packets is $rt(M, \Pi, P) \le 8C + 4D - 8$.*

8 Butterfly

In a n-input butterfly network B, where n is a power of 2, each node has a distinct label $\langle l, r\rangle$, where l is its level and r is its row. The rows are labeled by $\lg n$-bit binary addresses. Nodes at level 0 are inputs (sources of packets) and nodes at level $\lg n$ are outputs (destinations of packets). Thus, an n-input butterfly has $n(\lg n + 1)$ nodes. For $l < \lg n$, a node labeled $\langle l, r\rangle$ is connected to nodes $\langle l+1, r\rangle$ and $\langle l+1, r^l\rangle$, where $r^{(l)}$ denotes r with the lth bit complemented. Note that there is a unique path from an input node to an output node.

We study random-destinations and permutation routing problems on the butterfly. In a random-destination problem, each input node is the source of exactly one packet, and the destination of each packet is chosen randomly and uniformly among the output nodes. In a permutation routing problem, each input node is the source of exactly one packet, and each output node is the destination of exactly one packet. We route permutations using Valiant's scheme [22, 23] where packets choose random intermediate nodes.

Consider a random destinations routing problem (B, Π, P). A trivial lower bound on the routing time is $\lg n$, the length of any path. We will show that the greedy algorithm of Section 5 gives routing time at most $\frac{5}{2}\lg n$ w.h.p., which is optimal up to a constant factor. In order to get this bound, we first show that any packet collides with at most $\frac{3}{2}\lg n$ other packets w.h.p. Thus, the maximum degree in the dependency graph $\overline{\mathcal{D}}$ is at most $\frac{3}{2}\lg n$. Since the paths are shortcut-free and $D \le \lg n$, using Lemma 5.3, we get the bound.

Consider a packet $\pi \in \Pi$ with path $v_0, v_1, \ldots, v_{\lg n}$. Let m_i, $i = 1, \ldots, \lg n - 1$, be the number of other packets that could possibly collide with packet π, with the first collision edge being (v_i, v_{i+1}) (note that it is not possible to have collisions on edge (v_0, v_1)). Let q_i be the probability that one of those m_i packets actually uses the edge (v_i, v_{i+1}). The following lemma follows from the properties of the butterfly network.

Lemma 8.1 *$m_0 = 0$, $m_i = 2^{i-1}$, and $q_i = 2^{-(i+1)}$, for $i = 1, \ldots, \lg n - 1$.*

Proof: Clearly $m_0 = 0$. Let σ be some packet $\sigma \ne \pi$ that collides for the first time with π on edge (v_i, v_{i+1}). Packet σ arrives at v_i using edge (w, v_i) with $w \ne v_{i-1}$. The number of input nodes that can reach w is 2^{i-1}, and σ could have originated from any of these nodes. Thus, $m_i = 2^{i-1}$.

To obtain q_i, we observe that from v_{i+1}, packet σ can reach $M = 2^{\lg n - (i+1)}$ destination nodes. Since the only way to get to these nodes is using the edge (v_i, v_{i+1}), and since the destination nodes are chosen randomly with uniform probability, the probability that packet σ uses this edge is $q_i = M/n = 2^{-(i+1)}$. ∎

Let X_i be the number of different other packets that collide with packet i. Let $x_j^{(i)}$ be the Bernoulli random variable that equals 1 if packet j collides with packet i, and let $q_j^{(i)} = \mathbf{P}[x_j^{(i)} = 1]$. $X_i = \sum_j x_j^{(i)}$, and $x_j^{(i)}$ are independent for different j. Let $\mu = \mathbf{E}[X_i] = \sum_j q_j^{(i)}$. Using Lemma 8.1 we obtain:

$$\sum_j q_j^{(i)} = \sum_{k=1}^{\lg n - 1} m_k q_k = \frac{1}{4}(\lg n - 1).$$

Thus, the expected number of packets that use packet i's path is $\frac{1}{4}(\lg n - 1)$, independent of i, or the specific path used by packet i. Note that in the dependency graph $\overline{\mathcal{D}}$, X_i is equal to $\overline{deg}_i$. We will use the following version of the Chernoff bound to get a concentration result for X_i:

Lemma 8.2 ([16]) *Let $y_1, \ldots, y_n$ be independent binomial random variables, with $\mathbf{P}[y_i = 1] = b_i$ for $i \in [1, m]$, where $0 < b_i < 1$. Let $Y = \sum_{i=1}^m y_i$, $\mu = \sum_{i=1}^m b_i$. Then, for any $\alpha > 2e$, $\mathbf{P}[Y > \alpha\mu] < 2^{-\alpha\mu}$.*

Define the event E_i by $E_i = \{X_i > \alpha \lg n\}$ for some $\alpha > 2e$. Applying the Chernoff bound in Lemma 8.2, we get $\mathbf{P}\left[X_i > \frac{\alpha}{4}\lg n\right] \le \mathbf{P}[X_i > \alpha\mu] < \frac{2^{\alpha/4}}{n^{\alpha/4}}$. The identity $\mathbf{P}[\max_i X_i \le \alpha \lg n] = 1 - \mathbf{P}[\cup_i E_i]$ and the union bound then give

$$\mathbf{P}\left[\max_i X_i \le \frac{\alpha}{4}\lg n\right] > 1 - n \cdot \frac{2^{\alpha/4}}{n^{\alpha/4}} = 1 - \frac{2^{\alpha/4}}{n^{\alpha/4-1}}$$

Taking $\alpha = 6$, since $\max_i \overline{deg}_i = \max_i X_i$, and $D = \lg n$, Lemma 5.3 then gives the following theorem:

Theorem 8.3 *For random destination routing problem (B, Π, P) on the n-input butterfly B, the routing time of the greedy algorithm satisfies $\mathbf{P}\left[rt_{Gr}(B, \Pi, P) \le \frac{5}{2}\lg n\right] > 1 - 2\sqrt{2}n^{-\frac{1}{2}}$.*

It is known that there exist permutation routing problems with congestion at least $\Omega(\sqrt{n})$, i.e. some edges are hot-spots (see [16, Section 4.2]). In order to avoid hot-spots, Valiant [22, 23] proposed the following alternative scheme to route permutation routing problems in a butterfly-like network. Take two butterflies and connect them so that the outputs of the first butterfly are are the inputs to the second butterfly. The permutation problem is for this "joint" butterfly network: each packet has source on the input of the first butterfly and destination on the output of the second butterfly. The routing idea is to allow each packet to choose uniformly at random an intermediate node on the output of the first butterfly. The path is then given by source to random intermediate node followed by intermediate node to destination. Such a routing scheme avoids hot-spots – the permutation problem is now equivalent to two random destinations problems. Thus, we can apply Theorem 8.3 twice to obtain the following theorem:

Theorem 8.4 *For a permutation routing problem (B, Π, P) on the n-input double-butterfly B using Valiant's scheme, the routing time of the greedy algorithm satisfies $\mathbf{P}[rt_{Gr}(B, \Pi, P) \le 5\lg n] > 1 - 4\sqrt{2}n^{-\frac{1}{2}}$.*

9 Hypercube

In the n-hypercube network H with n nodes, where n is a power of 2, each node v_i has a distinct $\lg n$-bit binary label $\langle i_1, i_2, \ldots, i_{\lg n}\rangle \in \{0,1\}^{\lg n}$. There is a link between two nodes v_i and v_j if and only if their respective labels $\langle i_1, i_2, \ldots, i_{\lg n}\rangle$ and $\langle j_1, j_2, \ldots, j_{\lg n}\rangle$ differ in exactly one position. Thus, the degree of every node is $\lg n$.

We study random-destinations and permutation routing problems on the hypercube. In a random-destinations problem, each node is the source of exactly one packet, and the destination of each packet is chosen randomly and uniformly among the nodes of the hypercube. In a permutation routing problem, each node in the hypercube is the source and destination of exactly one packet. For permutation routing, we use Valiant's scheme of routing to random intermediate nodes first, as with the double-Butterfly.

We use (left-to-right) *bit-fixing* to determine the paths given the sources and destinations: let π be a packet which has to be routed from source $\mathrm{s}(\pi)$ to destination $\delta(\pi)$; flip the leftmost bit at which the labels of $\mathrm{s}(\pi)$ and $\delta(\pi)$ differ and send packet π along the edge that leads to the resulting node v; now repeat this process with v and $\delta(\pi)$, continuing until the path has reached $\delta(\pi)$. Note that bit-fixing paths are shortest paths, since the number of bits flipped is minimum. Further, $D \le \lg n$, since no more than n bits are flipped.

Consider a random destinations routing problem (H, Π, P) with bit-fixing paths P. We will show that the greedy algorithm of Section 5, has routing time bounded by $7\lg n$, w.h.p., which is optimal to within constant factors because it can be shown that $D \ge \frac{1}{4}\lg n$ w.h.p (using a simple Chernoff bounding argument). As with the Butterfly analysis, let X_i be the number of other different packets that packet i collides with. We will use the following result which is adapted from [16, Theorem 4.6]:

Lemma 9.1 ([16]) $\mathbf{P}[\max_i X_i \le 6\lg n] > 1 - 1/(32n)$.

Thus, the maximum degree in the dependency graph $\overline{\mathcal{D}}$ is at most $6\lg n$, with probability at least $1-1/(32n)$. Since $D \le \ln n$, Lemma 5.3 implies that the routing time of the greedy algorithm is at most $7\lg n$ w.h.p. We have the following theorem:

Theorem 9.2 *For a random destination routing problem (H, Π, P) on the n-hypercube H with bit-fixing paths. The routing time of the greedy algorithm satisfies* $\mathbf{P}\left[rt_{Gr}(B, \Pi, P) \leq 7 \lg n\right] > 1 - 1/(32n)$.

It is known that on the n-hypercube, there exist permutation routing problems with congestion at least $\Omega(\sqrt{n/logn})$, i.e. some edges are hot-spots (see [16, Section 4.2]). In order to avoid hot-spots, we will use Valiant's scheme [22, 23]: for any permutation problem, we will construct paths P' by first taking bit-fixing paths from a source to a random uniformly picked intermediate node, followed by bit-fixing paths from the intermediate node to a destination. This routing problem is the combination of two random destinations problem. Thus, we can apply Theorem 9.2 twice to obtain the following theorem,

Theorem 9.3 *For a permutation routing problem (H, Π, P') on the n-hypercube H which uses Valiant's scheme with bit-fixing paths, the routing time of the greedy algorithm satisfies* $\mathbf{P}\left[rt_{Gr}(B', \Pi, P') < 14 \lg n\right] > 1 - 1/(16n)$.

10 Discussion

We have given a comprehensive analysis of direct routing in general, and on particular network topologies. In all cases we find that a simple greedy scheduling strategy is optimal or near optimal. For the tree and mesh, our results can accomodate an arbitrary set of packets. For the butterfly and hypercube, we focused on the lightly loaded (one packet per input node) case. It would be useful to have worst case results for the fully-loaded ($\lg n$ packets per node) cases. In this case, we conjecture that direct routing on the fully loaded Butterfly is hard, namely that there is a problem with congestion $\lg n$ for which any direct routing algorithm would require $\Omega(\lg^2 n)$ routing time. The intuition for our conjecture can be found in Koch's result, [11], where he proves that the throughput of the lightly loaded butterfly with random destinations is $O(n/\lg n)$ w.h.p., hence the fully loaded butterfly with random destinations should require $\Omega(\lg^2 n)$ phases in which about $n/\lg n$ packets can be sent. The proof, however, has been elusive.

One important implication of our results is that on many interesting network topologies, buffers are not necessary for optimal offline routing. Further, packets can be guaranteed not to conflict, and delivery times can be ensured.

References

[1] N. Alon, F.R.K. Chung, and R.L.Graham. Routing permutations on graphs via matching. *SIAM Journal on Discrete Mathematics*, 7(3):513–530, 1994.

[2] Stephen Alstrup, Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Direct routing on trees. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 98)*, pages 342–349, 1998.

[3] A. Ben-Dor, S. Halevi, and A. Schuster. Potential function analysis of greedy hot-potato routing. *Theory of Computing Systems*, 31(1):41–61, January/February 1998.

[4] Petra Berenbrink and Christian Scheideler. Locally efficient on-line strategies for routing packets along fixed paths. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 112–121, N.Y., January 17–19 1999. ACM-SIAM.

[5] A. Broder and E. Upfal. Dynamic deflection routing on arrays. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 348–358, May 1996.

[6] C. Busch. Õ(Congestion + Dilation) hot-potato routing on leveled networks. In *Proceedings of the Fourteenth ACM Symposium on Parallel Algorithms and Architectures*, pages 20–29, August 2002.

[7] C. Busch, M. Herlihy, and R. Wattenhofer. Hard-potato routing. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pages 278–285, May 2000.

[8] Deborah Estrin, Ramesh Govindan, John Heidemann, and Satish Kumar. Scalable coordination in sensor networks. In *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 99)*, pages 263–270, NY, August 15-20 1999. ACM Press.

[9] U. Feige and P. Raghavan. Exact analysis of hot-potato routing. In IEEE, editor, *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 553–562, Pittsburgh, PN, October 1992.

[10] Ch. Kaklamanis, D. Krizanc, and S. Rao. Hot-potato routing on processor arrays. In *Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 273–282, Velen, Germany, June 30–July 2, 1993.

[11] Richard Koch. Increasing the size of a network by a constant factor can increase performance by more than a constant factor. *SIAM Journal of Computing*, 21(5):801–823, 1992.

[12] F. T. Leighton, B. M. Maggs, and S. B. Rao. Packet routing and job-scheduling in $O(congestion + dilation)$ steps. *Combinatorica*, 14:167–186, 1994.

[13] Tom Leighton, Bruce Maggs, and Andrea W. Richa. Fast algorithms for finding O(congestion + dilation) packet routing schedules. *Combinatorica*, 19:375–401, 1999.

[14] Friedhelm Meyer auf der Heide and Christian Scheideler. Routing with bounded buffers and hot-potato routing in vertex-symmetric networks. In Paul G. Spirakis, editor, *Proceedings of the Third Annual European Symposium on Algorithms*, volume 979 of *LNCS*, pages 341–354, Corfu, Greece, 25–27 September 1995.

[15] Friedhelm Meyer auf der Heide and Berthold Vöcking. Shortest-path routing in arbitrary networks. *Journal of Algorithms*, 31(1):105–131, April 1999.

[16] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge, UK, 2000.

[17] Rafail Ostrovsky and Yuval Rabani. Universal $O(\text{congestion+dilation+}\log^{1+\varepsilon} N)$ local control packet switching algorithms. In *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing*, pages 644–653, New York, May 1997.

[18] Grammati E. Pantziou, Alan Roberts, and Antonios Symvonis. Many-to-many routing on trees via matchings. *Theoretical Computer Science*, 185(2):347–377, 1997.

[19] Yuval Rabani and Éva Tardos. Distributed packet switching in arbitrary networks. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 366–375, Philadelphia, Pennsylvania, 22–24 May 1996.

[20] Rajiv Ramaswami and Kumar N. Sivarajan. *Optical Networks, a Practical Perspective*. Morgan Kaufmann, 1998.

[21] A. Symvonis. Routing on trees. *Information Processing Letters*, 57(4):215–223, 1996.

[22] L. G. Valiant. A scheme for fast parallel communication. *SIAM Journal on Computing*, 11:350–361, 1982.

[23] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, pages 263–277, May 1981.

[24] L. Zhang. Optimal bounds for matching routing on trees. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 445–453, 1997.