# Direct Routing: Algorithms and Complexity[*]

Costas Busch[†]     Malik Magdon-Ismail[‡]     Marios Mavronicolas[§]

Paul Spirakis[¶]

May 19, 2005

## Abstract

*Direct routing* is the special case of *bufferless routing* where $N$ packets, once injected into the network, must be delivered to their destinations without collisions. We give a general treatment of three facets of direct routing:

(i) *Algorithms.* We present a polynomial time *greedy* direct algorithm which is worst-case optimal. We improve the bound of the greedy algorithm for special cases, by applying variants of the this algorithm to commonly used network topologies. In particular, we obtain *near-optimal* routing time for the *tree*, *mesh*, *butterfly* and *hypercube*.

(ii) *Complexity.* By a reduction from Vertex Coloring, we show that optimal Direct Routing is inapproximable, unless P=NP.

(iii) *Lower Bounds for Buffering.* We show that certain direct routing problems cannot be solved efficiently; in order to solve these problems, *any* routing algorithm needs buffers. We give non-trivial lower bounds on such buffering requirements for general routing algorithms.

# 1    Introduction

*Direct routing* is the special case of *bufferless routing* where $N$ packets need to be delivered from their sources to their destinations without colliding with each other, i.e., once injected, the packets proceed "directly" to their destination without being delayed (buffered) at intermediate nodes. Since direct routing is bufferless, packets spend the minimum possible time in the network given the paths they must follow – this is appealing in power/resource constrained environments (for example optical networks or sensor networks). From the point of view of quality of service, it is often desirable to provide a guarantee on the delivery time after injection, for example in streaming applications like audio and video. Direct routing can provide such guarantees.

A *path-routing problem* is specified by a set of packets with respective sources and destinations on a network (graph), together with a corresponding path for each packet. Given a path-routing problem, the task of a *direct scheduling algorithm* (or *direct algorithm* for short) is to compute the injection times of the packets. If the packets are injected at their specified times, they will follow their respective paths to their destinations without collisions. The objective is to minimize the *routing time rt*, which is the time at which the last packet is absorbed at its destination. We note that direct algorithms are inherently offline, since the computation of injection times requires knowledge about all packets in order to guarantee no collisions between the packets.

We assume a synchronous model for routing, in which during every time step, a packet may traverse one link. We measure the routing time 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). Denote by $rt^*$ the optimal routing time for a given path-routing problem. Since packets can traverse at most one link per time step, a well known lower bound on the routing time for *any* algorithm (direct or not) is $rt^* \geq \max\{C, D\} \geq \frac{1}{2}(C + D) = \Omega(C + D)$. It is generally desirable to design scheduling algorithms with routing times close to this lower bound.

Another kind of routing problem we consider is the *batch-routing problem*, in which we are given a set of packets with sources and destinations but the paths are not specified. We convert instances of batch-routing problems to instances of path-routing problems by finding appropriate paths for the packets. The congestion and dilation of the chosen paths affect the efficiency of the scheduling algorithm. In general we want to find paths which minimize $C$ and $D$. Let $C^*$ and $D^*$ be the congestion and dilation of the optimal paths (i.e. the paths that minimize $\max\{C, D\}$). Then, for any scheduling algorithm it holds that $rt^* = \Omega(C^* + D^*)$, and it is desirable to achieve routing times close to this lower bound.

Since the above lower bounds on the routing time for path or batch-routing problems

apply for *any* scheduling algorithm (direct or not), we are comparing the performance of our algorithms to optimal algorithms which may use buffers.

## 1.1 Contributions

We give a general analysis of three aspects of direct routing, namely efficient algorithms for direct routing; the computational complexity of direct routing; and, the connection between direct routing and buffering.

### 1.1.1 Algorithms

We first study path-routing problems in arbitrary networks. We give a greedy direct algorithm with routing time $O(C \cdot D)$. We show that this is worst-case optimal for direct algorithms. A natural question is whether one can improve this routing time bound for routing problems on particular network topologies. We study batch-routing problems on the tree, mesh, hypercube, and butterfly, for which we show that there exist paths for which simple variations of the greedy direct algorithm give routing times close to the optimal routing time $\Omega(C^* + D^*)$. Thus, in many cases, efficient routing can be achieved without the use of buffers.

Throughout, we will use the expression *with high probability (w.h.p.)* to denote a probability of the form $1 - O(n^{-\alpha})$, for some $\alpha > 0$, where $n$ is the size of the network.

*Arbitrary Networks:* We consider path-routing problems on arbitrary network topologies. We give a simple *greedy* direct algorithm which considers packets in some order, assigning the first available injection time to each packet. The greedy algorithm is really a family of algorithms, a particular realization of which depends on the particular ordering of the packets that is chosen. Different orderings can lead to different routing times, and we make use of this dependence to determine particular "good" orderings for specific networks such as the tree and the mesh (see below).

For *any* ordering of the packets, the greedy direct algorithm guarantees a routing time $rt \leq C \cdot D$. We show that this is worst-case optimal: there exist instances of path-routing problems for which no direct algorithm can achieve a better routing time than $C \cdot D$.

*Tree:* We study batch-routing problems on arbitrary trees. Given a set of packets with arbitrary sources and destinations, we convert the batch-routing problem to a path-routing problem by assigning the unique shortest paths to the packets. These paths are optimal in terms of congestion and dilation, since any other selection of paths must

2

include the shortest paths. We give a direct algorithm for the path-routing problem using the shortest paths, with routing time $rt \leq 2C^* + D^* - 2 < 3 \cdot rt^*$. The direct algorithm is obtained using the greedy algorithm with a particular ordering of the packets.

*Mesh:* We study batch-routing problems on the $d$-dimensional mesh with $n$ nodes. Given a set of packets with arbitrary sources and destinations, we convert the batch-routing problem to a path-routing problem, by using near-optimal paths with respect to the congestion and dilation (the paths are obtained using the construction described in [**?**]). Using these paths, we give a direct algorithm which, with high probability, has routing time
$$rt = O(d^2 C^* \log^2 n + d^2 D^*) = O(d^2 \cdot \log^2 n \cdot rt^*).$$

This result follows from a more general result we give for path routing problems on the mesh: if the paths contain at most $b$ "bends", i.e. at most $b$ dimension changes, then we give a direct algorithm with routing time $O(b \cdot C + D)$. This direct algorithm gives the near optimal routing time as advertised, because the paths constructed in [**?**] have $O(d \log n)$ bends with congestion $C$ within $d \log n$ of the optimal $C^*$, and dilation $D$ within $d^2$ of the optimal $D^*$.

We also study *permutation* batch-routing problems on the 2-dimensional mesh, in which each node is the source and destination of one packet. We show that if the packets follow one-bend shortest paths, then using the same direct algorithm, we obtain routing time a constant factor away from optimal.

*Butterfly:* We study permutation batch-routing problems on a butterfly with $n$ inputs. We first convert an input instance of the permutation to a path-routing problem using Valiant's method [28, 29]: we use two butterflies connected back to back, and each packet uses a path to an intermediate random node in the output of the first butterfly. The path selection guarantees that the congestion is $O(\lg n)$, with high probability. We then apply the direct schedule obtained by the greedy algorithm using an arbitrary ordering of the packets, which gives routing time $rt \leq 5 \lg n$, with high probability. This bound is within a constant factor from optimal, since $D^* = \Omega(\log n)$.

*Hypercube:* We study permutation batch-routing problems on a hypercube with $n$ nodes. We convert the permutation problem to a path-routing problem by selecting a random intermediate node for each packet. The paths from the source to the intermediate node are obtained by lexicographically bit-fixing the bits of the source to match those of the intermediate node. Then a similar approach is used to construct the path from the intermediate node to the destination. We then apply the greedy direct algorithm

3

with an arbitrary ordering of the packets to obtain the direct schedule. The resulting schedule has routing time $rt < 14 \lg n$, with high probability. This is a worst-case constant factor approximation to the optimal schedule, since there exist permutations for which $D^* = \Omega(\lg n)$.

### 1.1.2 Computational Complexity

We show that the problem of finding the optimal direct schedule for path-routing problems is NP-complete. Thus, there are instances of path-routing for which computing the optimal schedule is computationally hard. In order to obtain this result, we reduce the vertex coloring problem to path routing. In particular, given an instance of the vertex coloring problem, we construct, in polynomial time, an instance of a path-routing problem with dilation $D$, such that the vertex coloring has a solution with $k$ colors if and only if there exists a direct routing schedule with routing time $D + k - 1$. Thus, the reduction is *gap-preserving* [15], so direct routing is as hard to approximate as coloring.

### 1.1.3 Lower Bounds for Buffering

We construct path-routing problems for which every direct algorithm requires routing time $C \cdot D$, while $(C + D) = \Theta(\sqrt{C \cdot D}) = o(C \cdot D)$. If we were allowed to use buffers, there exist scheduling algorithms ([19, 22, 24]) that would give routing time close to the optimal $O(C + D)$. Thus, in order to solve the above routing problems efficiently, packets have to be buffered (we say that a packet is buffered if it remains in the buffer of a node for a time step, given that the packet was in the same node the previous time step). We study what the minimum buffering requirments would be in order to improve the routing time of the schedule. Specifically, we show that in order to obtain a routing time which is $O(C + D)$, a scheduling algorithm has to buffer packets $\Omega(N^{4/3})$ times in total. More generally, we show that to obtain a routing time within $N^\epsilon$ of optimal, a scheduling algorithm must buffer packets $\Omega(N^{(4-2\epsilon)/3})$ times.

## 1.2 Related Work

The only previous work which specifically addresses direct routing is for permutation problems on trees [3, 27], where the authors obtain a routing time $O(n)$ for any tree with $n$ nodes. This result is worst-case optimal, since there are permutations on trees with $rt^* = \Omega(n)$. However there are many examples where their algorithms are sub-optimal, for example, for a permutation routing on a star network, in which case they obtain a routing time $\Omega(n)$. Our algorithm for trees is every-case optimal for *arbitrary* routing problems, in particular for a permutation routing on a star network, our algorithm gives routing time $O(1)$.

4

Cypher *et al.* [11] study an online version of direct routing in which a worm (packet of length $L$) can be re-transmitted if it is dropped (they also allow the links to have bandwidth $B \geq 1$). For the case corresponding to our work ($L = B = 1$), they give an algorithm with routing time $O((C + \log n) \cdot D)$. We give an off line algorithm with routing time $O(C \cdot D)$, show that this is worst case optimal, and that it is NP-hard to give a good approximation to the optimal direct routing time. We also obtain near-optimal routing time (with respect to buffered routing) for many interesting networks, for example the mesh.

Adler *et al.* [1] study a dual to the direct routing problem, which is time constrained routing, where the task is to schedule as many packets as possible within a given time frame. They show that the time constrained version of the problem is NP-complete, and also study approximation algorithms on linear networks, trees and meshes. They also discuss how much buffering could help in this setting.

Other models of bufferless routing are *matching routing* [2, 25, 30], where packets move to their destinations by swapping packets in adjacent nodes, and *hot-potato routing* [4, 5, 7, 8, 9, 13, 16, 21], in which packets follow links that bring them closer to the destination, and if they cannot move closer (due to collisions) they are deflected. A basic difference between those routing models and direct routing is that the packets don't follow some specific path. Optimal routing for given paths on arbitrary networks have been studied extensively in the context of store-and-forward algorithms (which are algorithms with buffers) [18, 19, 22, 24, 26].

**Paper Organization.** We give some necessary preliminaries in Section 2. Then, we present direct scheduling algorithms in Section 3 followed by the computational complexity of direct routing in Section 4, ending with lower bounds on buffering in Section 5. We conclude with some discussion of out results in Section 6.

# 2 Preliminaries

## 2.1 Problem Definitions

Consider a graph $G = (V, E)$ with $n \geq 1$ nodes. A path $p$ in $G$ is a sequence of nodes $p = (v_1, v_2, \ldots, v_k)$. The length of a path $p$, denoted $|p|$, is the number of edges in the path. For any edge $e = (v_i, v_j) \in p$, let $d_p(e)$, the depth of $e$ in $p$, denote the length of the path $(v_1, \ldots, v_i, v_j)$, which is the prefix of $p$ ending at $e$.

We consider routing problems where packets need to be delivered in the network. We model the graph so that the nodes are synchronous: time is discrete and all nodes take steps simultaneously. At each time step, at most one packet can follow a link in each direction; thus, at most two packets can follow a link at the same time, one packet at each direction.

A *path-routing problem* $(G, \Pi, \mathcal{P})$ is specified by a set of $N \geq 1$ packets $\Pi = \{\pi_i\}_{i=1}^N$; each packet has a pre-specified path $p_i = (s_i, \ldots, \delta_i) \in \mathcal{P}$, where $s_i$ is the source and $\delta_i$ the destination of the packet. The objective is to send the packets to their destinations by following the pre-specified paths.

Consider two packets $\pi_i$ and $\pi_j$. We say that their respective paths $p_i$ and $p_j$ *interfere* if they share an edge in the same direction; in this case we also say that the packets interfere. The packets *collide* if they appear in the same node at the same time, *and* the next edge in their paths is the same.

Given a path-routing problem, in direct routing, the packets are sent to their destinations with no collisions. The only parameter that needs to be computed is the injection times of the packets which will guarantee collision-free delivery of the packets. Thus, a set of injection times $\mathcal{T} = \{\tau_i\}_{i=1}^N$ specifies a valid *direct schedule* if each packet $\pi_i$ is injected at its corresponding time $\tau_i$ into its source $s_i$, then it will follow its path without collisions to its destination where it will be absorbed at time $t_i = \tau_i + |p_i|$.

For a path-routing problem $(G, \Pi, \mathcal{P})$, the task of a *direct scheduling algorithm* (or direct algorithm for short) is to compute a direct schedule $\mathcal{T}$. The *routing time* of the algorithm, denoted $rt(G, \Pi, \mathcal{P})$, is the maximum time at which a packet gets absorbed at its destination, $rt(G, \Pi, \mathcal{P}) = \max_i\{\tau_i + |p_i|\}$ (this is also the routing time of the direct schedule $\mathcal{T}$). The *offline time* of the algorithm, $ol(G, \Pi, \mathcal{P})$ is the number of operations used to compute the direct schedule $\mathcal{T}$.

We also study *batch-routing problems* in which the paths are not specified in the beginning. In particular, a batch routing problem $(G, \Pi, Q)$ specifies for each packet $\pi \in \Pi$, a pair of sources and destinations $(s_i, \delta_i) \in Q$. Interesting batch-routing problems are *permutation problems*, in which every node in $G$ is the source and destination of one packet, and *random destination* problems, in which each node is the source of one packet and the destination of a packet is chosen randomly in the network. In order to solve batch-routing problems, we first convert them to path-routing problems, by selecting appropriate paths for the packets, and then apply a direct algorithm.

## 2.2 Dependency Graphs

Consider a path-routing problem $(G, \Pi, \mathcal{P})$. The *dependency graph* $\mathcal{D}$ of the routing problem is a graph in which each packet $\pi_i \in \Pi$ corresponds to a unique node. We will use $\pi_i$ to refer to the corresponding node in $\mathcal{D}$. There is an edge between two packets in $\mathcal{D}$ if their paths interfere. An edge $(\pi_i, \pi_j)$ with $i < j$ in $\mathcal{D}$ has an associated set of weights $\mathcal{W}_{i,j}$: $w \in \mathcal{W}_{i,j}$ if and only if $\pi_i$ and $\pi_j$ interfere on some edge $e$ for which $d_{p_i}(e) - d_{p_j}(e) = w$. Thus, in a valid direct routing schedule with injection times $\tau_i, \tau_j$ for $\pi_i, \pi_j$, it must be that $\tau_j - \tau_i \notin \mathcal{W}_{i,j}$. An

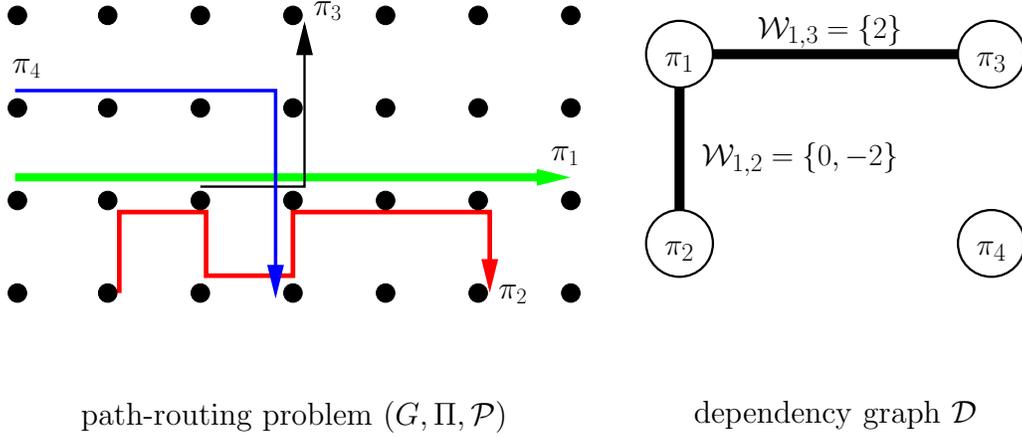illustration of a direct routing problem and its corresponding dependency graph are shown in Figure 1.



<center>path-routing problem $(G, \Pi, \mathcal{P})$           dependency graph $\mathcal{D}$</center>

Figure 1: An example direct routing problem and its dependency graph.

We say that two packets are *synchronized* if they are adjacent in $\mathcal{D}$ and 0 is in the weight set of the edge in $\mathcal{D}$ connecting the two packets. A clique $\mathcal{K}$ in $\mathcal{D}$ is *synchronized* if all the packets in $\mathcal{K}$ are synchronized, i.e., if 0 is in the weight set of every edge in $\mathcal{K}$. No pair in a synchronized clique can have the same injection time, as otherwise they would collide. Thus, the size of the maximum synchronized clique in $\mathcal{D}$ gives a lower bound on the routing time:

**Lemma 2.1 (Lower Bound on Routing Time)** *Let $\mathcal{K}$ be a maximum synchronized clique in the dependency graph $\mathcal{D}$. Then, for any direct algorithm, $rt(G, \Pi, \mathcal{P}) \geq |\mathcal{K}|$*

We define the *weight degree of an edge $e$* in $\mathcal{D}$, denoted $W(e)$, as the size of the edge's weight set. We define the *weight degree of a node $\pi$* in $\mathcal{D}$, denoted $W(\pi)$, as the sum of the weight degrees of all edges incident with $\pi$. We define the *weight of the dependency graph, $W(\mathcal{D})$*, as the sum of the weight degrees of all its edges, $W(\mathcal{D}) = \sum_{e \in E(\mathcal{D})} W(e)$. For the example in Figure 1, $W(D) = 3$.

# 3   Algorithms

Here we consider algorithms for direct routing. All the direct algorithms we give are based on a greedy algorithm which finds schedules for arbitrary path-routing problems in arbitrary networks. The greedy direct algorithm guarantees a routing time $O(C \cdot D)$. There exist path routing instances for which the direct routing time is $\Omega(C \cdot D)$, i.e., no direct algorithm can

<center>7</center>

perform better than the greedy algorithm, hence the greedy algorithm is worst case optimal. We give variations of the greedy direct algorithm which improve on the $O(C \cdot D)$ general bound for specific architectures, such as the tree, mesh, butterfly, and hypercube, as we discuss in the next subsections. The greedy algorithm is as follows:

1: // **Greedy Direct Algorithm:**
2: // **Input:** path-routing problem $(G, \Pi, \mathcal{P})$ with $N$ packets $\Pi = \{\pi_i\}_{i=1}^N$.
3: // **Output:** Set of injection times $\mathcal{T} = \{\tau_i\}_{i=1}^N$.
4: Let $\pi_1, \ldots, \pi_N$ be any arbitrarily chosen ordering of the packets.
5: **for** $i = 1$ *to* $N$ **do**
6:   Greedily assign the first available injection time $\tau_i$ to packet $\pi_i \in \Pi$, so that it does not collide with any packet already assigned an injection time.
7: **end for**

The greedy direct algorithm is really a family of algorithms, one for each specific ordering of the packets. By construction, no packet $\pi_j$ collides with any packet $\pi_i$ with $i < j$, and thus, by induction, the greedy algorithm produces a valid direct schedule. The routing time for the greedy algorithm will be denoted $rt_{Gr}(G, \Pi, \mathcal{P})$. Consider the dependency graph $\mathcal{D}$ for the routing problem $(G, \Pi, \mathcal{P})$. We can show that $\tau_i \leq W(\pi_i)$, where $W(\pi_i)$ is the weight degree of packet $\pi_i$, which implies:

**Lemma 3.1** $rt_{Gr}(G, \Pi, \mathcal{P}) \leq \max_i \{W(\pi_i) + |p_i|\}$.

**Proof:** We show that the injection times assigned by the greedy algorithm satisfy $\tau_i \leq W(\pi_i)$, from which the claim follows immediately. For packet $i$, we consider the path $p_i$ and the interval of times $[0, W(\pi_i)]$. Every time a packet $\sigma$, 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 $\pi_i$ to collide with $\sigma$ at the time $\sigma$ uses this edge. Since $W(\pi_i)$ is the number of times packets can collide with $\pi_i$, we remove at most $W(\pi_i)$ injection times from this set. As there are $W(\pi_i) + 1$ injection times in this set, it cannot be empty, so the greedy algorithm must assign an injection time to $\pi_i$ 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 in terms of $C$ and $D$. Since the congestion is $C$ and $|p_i| \leq D$ $\forall i$, a packet interferes with other packets at most $(C - 1) \cdot D$ times. Thus, $W(\pi_i) \leq (C - 1) \cdot D$, $\forall i$. Therefore, using Lemma 3.1 we obtain:

**Theorem 3.2 (Greedy Routing Time Bound)** $rt_{Gr}(G, \Pi, \mathcal{P}) \leq C \cdot D$.

We briefly discuss the offline 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 3.3** *The offline computation time of the greedy algorithm is $ol_{Gr}(G, \Pi, \mathcal{P}) = O(N \cdot C \cdot D)$*

The bound of Lemma 3.3 is tight, since in the worst case, each packet may have $C \cdot D$ interferences with other packets. In Theorem 5.1, we establish the existence of hard instances of direct routing, specifically instances where any direct algorithm requires $\Omega(C \cdot D)$ routing time. Thus, the general $O(C \cdot D)$ bound on the routing time of the greedy algorithm is worst-case optimal, to within constant factors. In the next subsections, we will show how the greedy algorithm can do better for batch-routing problems on specific network architectures by choosing appropriate paths and then using a more careful choice of the order in which packets are considered.

## 3.1 Trees

Here, we consider batch-routing problems on trees. Consider the batch-routing problem $(T, \Pi, Q)$, in which $T$ is a tree with $n$ nodes. We construct a path-routing problem $(T, \Pi, \mathcal{P})$ such that all the paths in $\mathcal{P}$ are shortest paths, for the given sources and destinations in $Q$. Shortest paths have optimal congestion on trees, given sources and destinations, since any other selection of paths must contain the shortest paths. Thus, if the shortest paths have congestion $C$ and dilation $D$, any other selection of paths must have at least as much congestion and dilation, which implies that $\Omega(C + D)$ is a lower bound for the routing time.

We now show that the greedy algorithm with a particular order in which the packets are considered gives an asymptotically optimal schedule. 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 $d_i$, with $d_{i_1} \leq d_{i_2} \leq \cdots \leq d_{i_N}$.

**Theorem 3.4 (Routing Time on Trees)** *Let $(T, \Pi, Q)$ be any batch-routing problem on the tree $T$. If the packets follow shortest paths $P$, then the routing time of the direct greedy algorithm using the distance-ordered packets is $rt(T, \Pi, \mathcal{P}) \leq 2C + D - 2$.*

**Proof:** We show that every injection time satisfies $\tau_i \leq 2C - 2$. When a packet $\pi_i$ with distance $d_i$ is considered, let $v_i$ be the closest node to $r$ on its path (see Figure 2). All packets that are already assigned times that could possibly collide with $\pi_i$ are those that use the two edges in $\pi_i$'s path incident with $v_i$ (for example packets $\pi', \pi''$ in Figure 2). Hence, there are
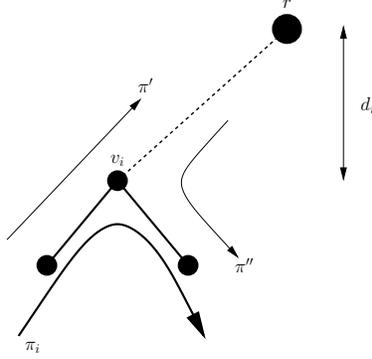
9

Figure 2: The path of a packet $\pi_i$ in the tree

at most $2C - 2$ such other packets. Since $\pi_i$ is assigned the smallest available injection time, it must therefore be assigned a time in $[0, 2C - 2]$. Since the path length of $\pi$ has length at most $D$, we obtain the desired result. ■

## 3.2   Mesh

A $d$-dimensional mesh network $M = M(m_1, m_2, \ldots, m_d)$ is a multi-dimensional grid of nodes with side length $m_i$ in dimension $i$. The number of nodes is $n = \prod_{i=1}^{d} m_i$, and we define $m = \sum_{i=1}^{d} m_i$. Every node is connected to up to $2d$ of its neighbors on the grid.

The hard instance of direct routing constructed in Theorem 5.1 is for the two dimensional mesh. In this problem, any direct algorithm would have routing time $\Omega(C \cdot D)$, and for this particular problem $C \cdot D$ is strictly worse than $C + D$ (see Theorem 5.1). Thus, with respect to worst case routing time on arbitrary inputs, the general greedy direct algorithm is asymptotically optimal for mesh networks. We discuss some important special cases where the situation is considerably better. In particular, we give a variation of the greedy direct routing algorithm which is analyzed in terms of the number of times that the packet paths "bend" on the mesh. We then apply this algorithm to the 2-dimensional mesh in order to obtain optimal permutation routing, and the $d$-dimensional mesh, in order to obtain near-optimal routing for arbitrary batch-routing problems.

### 3.2.1   Multi-bend Paths

Here, we analyze a variation of the greedy direct algorithm in terms of the number of times a packet bends in the network. Consider a path-routing problem $(G, \Pi, \mathcal{P})$. We first give an upper bound on the weight degree $W(\mathcal{D})$ of dependency graph $\mathcal{D}$ in terms of the number of bends in the paths. We then use the weight degree bound in order to obtain an upper

bound on the routing time of the algorithm.

For any subset of packets $\Pi' \subseteq \Pi$, let $\mathcal{D}_{\Pi'}$ denote the subgraph of $\mathcal{D}$ induced by the set of packets $\Pi'$. (Note that $\mathcal{D} = \mathcal{D}_\Pi$.) Consider the path $p$ of a packet $\pi$. Let's assume that $p = (\ldots, v_i, v, v_j, \ldots)$, such that the edges $(v_i, v)$ and $(v, v_j)$ are in different dimensions. We say that the path of packet $\pi$ *bends* at node $v$, and that $v$ is an *internal bending node.* We define the source and destination nodes of a packet $\pi$ to be *external bending nodes.* The *segment* $p' = (u_i, \ldots, u_j)$ of a path $p$, is a subpath of $p$ in which only $u_i$ and $u_j$ are bending nodes. Consider two packets $\pi_1$ and $\pi_2$ whose respective paths $p_1$ and $p_2$ interfere at some edge $e$. Let $p'_1$ and $p'_2$ be the two respective segments of $p_1$ and $p_2$ which contain $e$. Let $p'$ be the longest subpath of $p'_1$ and $p'_2$ which is common to $p'_1$ and $p'_2$; clearly $e$ is an edge in $p'$. Let's assume that $p' = (v_i, \ldots, v_j)$. It must be that $v_i$ is a bending node of one of the two packets, and the same is true of $v_j$. Further, none of the other nodes in $p'$ are bending nodes of either of the two packets. We refer to such a path $p'$ as a *common subpath.* Note there could be many common subpaths for the packets $\pi_1$ and $\pi_2$, if they meet multiple times on their paths.

Since $p_1$ and $p_2$ interfere on $e$, the edge $h = (\pi_1, \pi_2)$ will be present in the dependency graph $\mathcal{D}$ with some weight $w \in \mathcal{W}_{1,2}$ representing this interference. Now consider some other edge $e' \neq e$ in $p'$. If the packets $\pi_1$ and $\pi_2$ collide at $e$, then they must collide at $e'$. This implies that the interference on edge $e'$ is represented in the dependency graph $\mathcal{D}$ with the same weight $w$ on the edge $h$. Similarly, all interferences of the two packets on their common subpath $p'$ are represented with the same weight $w$ on edge $h$. Thus, weight $w$ suffices to represent the interference of the two packets on the entire subpath $p'$. Therefore, a common subpath contributes at most one to the weight-number of $\mathcal{D}$, and in order to find an upper bound on $W(\mathcal{D})$, we only need to find an upper bound on the number of common subpaths. Using this observation we obtain the following bound:

**Lemma 3.5** *For any subset $\Pi' \subseteq \Pi$, $W(\mathcal{D}_{\Pi'}) \leq 2(b+1)|\Pi'|(C-1)$, where $b$ is the maximum number of internal bending nodes of any path in $\Pi'$.*

**Proof:** For each common subpath, one of the packets must bend at the beginning and one at the end nodes of the subpath. Thus, a packet contributes to the total number of subpaths only when it bends. Consider a packet $\pi$ which bends at a node $v$. Let $e_1$ and $e_2$ be the two edges of the path of $\pi$ adjacent to $v$. On $e_1$ the packet may meet with at most $C-1$ other packets. Thus, $e_1$ contributes at most $C-1$ to the number of common subpaths. Similarly, $e_2$ contributes at most $C-1$ to the number of common subpaths. Thus, each internal bend contributes at most $2C-2$ to the number of common subpaths, and each external bend $C-1$. Therefore, for the set of packets $\Pi'$, where the maximum number of internal bends is

$b$, the number of common subpaths is bounded by $2(b+1)|\Pi'|(C-1)$, which is also a bound on $W(\mathcal{D}_{\Pi'})$. ∎

Since the sum of the node weight degrees is $2W(\mathcal{D})$, we have that the average node weight degree of the dependency graph for *any* subset of the packets is upper bounded by $4(b+1)(C-1)$. We say that a graph $\mathcal{D}$ is *K-amortized* if the average weight degree for every subgraph is at most $K$.* Thus $\mathcal{D}$ is $4(b+1)C$-amortized. A *generalized coloring* of a graph with weights on each edge is a vertex coloring in which the difference between the colors of adjacent nodes cannot equal a weight. $K$-amortized graphs admit generalized colorings with $K+1$ colors. This is the content of the next lemma.

**Lemma 3.6** *Let $\mathcal{D}$ be a $K$-amortized graph. Then $\mathcal{D}$ has a valid $K+1$ generalized coloring.*

**Proof:** We use induction on $n$, the size of $G$. For $n=1$, the claim is trivial. Assume it is true for $n < r$ for $r > 1$, and consider $n = r$. Since the average node weight degree is at most $K$, there is a node $v$ with weight degree at most $K$. Consider the subgraph induced by $\mathcal{D} - v$. This subgraph is $K$-amortized, so suppose we have a valid $K+1$ generalized coloring of $\mathcal{D} - v$, which exists (by the induction hypothesis). Since $v$ has weight degree at most $K$, one of the $K+1$ colors can now be assigned to it to obtain a valid $K+1$ generalized coloring of $G$. ∎

A generalized coloring of the dependency graph gives a valid injection schedule with maximum injection time one less than the largest color, since with such an injection schedule no pair of packets is synchronized. Lemma 3.6 implies that the dependency graph $\mathcal{D}$ has a valid $4(b+1)(C-1)+1$ generalized coloring. Lemma 3.6 essentially determines the order in which the greedy algorithm considers the packets so as to ensure the desired routing time. Hence, we get the following result:

**Theorem 3.7 (Multi-bend Routing Time on Mesh)** *Let $(M, \Pi, \mathcal{P})$ be a path-routing problem on a mesh $M$ with congestion $C$ and dilation $D$. Suppose that each packet has at most $b$ internal bends. Then there is a direct schedule with routing time $rt(M, \Pi, \mathcal{P}) \leq 4(b+1)(C-1)+D$.*

Note that when $b = O(1)$ then the routing time of Theorem 3.7 is optimal. An implementation of the induction in the proof of Lemma 3.6 immediately leads to the following recursive procedure to compute the direct schedule in Theorem 3.7:

1: //**Recursive algorithm** Injection($\Pi$) **to compute direct schedule in mesh:**

---

*$K$-amortized graphs are similar to balanced graphs [6].

2: //**Input:** Set of packets $\Pi$, together with their paths.

3: //**Output:** injection times $\{\tau_i\}_{i=1}^N$, where $\tau_i$ is injection time for packet $\pi_i \in \Pi$.

4: **if** $|\Pi| = 1$ **then**

5:     Set $\tau_1 = 0$;

6: **else**

7:     Find $\pi_j \in \Pi$ such that $W(\pi_j) = \min_{\pi \in \Pi}\{W(\pi)\}$;

8:     Set $\Pi' = \Pi - \{\pi_j\}$;

9:     Call $\mathsf{Injection}(\Pi')$ to set the injection times of all packets in $\Pi'$.

10:     Set $\tau_j$ to the smallest available injection time so as not to collide with any packets already assigned injection times.

11: **end if**

### 3.2.2 Permutation Routing on Mesh

Consider a 2-dimensional $\sqrt{n} \times \sqrt{n}$ mesh. Take an arbitrary permutation batch-routing problem $(G, \Pi, Q)$. We solve the permutation problem by using shortest paths with at most one internal bend, such that the packet first moves in the row of its source until the column of the destination where it bends and then moves in the column toward the destination. Let $\mathcal{P}$ denote the set of paths that we obtained in this manner. Since at most $\sqrt{n}$ packets originate and have destination at each row, the congestion at each edge in the row is at most $O(\sqrt{n})$. Similarly for edges in rows. Applying Theorem 3.7, and the fact that $D = O(\sqrt{n})$, we get:

**Theorem 3.8 (Permutation Routing Time on Mesh)** *Let $(M, \Pi, Q)$ be a permutation routing problem on a 2-dimensional mesh. Then, there is a selection of paths $\mathcal{P}$ with a direct schedule that gives routing time $rt(M, \Pi, \mathcal{P}) = O(\sqrt{n})$.*

Note that that the routing time of Theorem 3.8 is worst case optimal for permutation routing on the mesh, since there exist permutations with $D = 2\sqrt{n} - 2$, in which a packet goes from one corner of the mesh to the opposite corner.

### 3.2.3 Near Optimal Direct Routing on Mesh

Here we consider again the $d$-dimensional mesh $M$, for arbitrary $d$. Suppose we are given an arbitrary batch-routing problem $(M, \Pi, Q)$. For any selection of of paths $\Pi$ for the sources and destinations in $Q$, with congestion $C$ and dilation $D$, it holds that $C + D = \Omega(C^* + D^*)$ (where $C^*$ and $D^*$ denote the smallest possible congestion and dilation of any path selection for the given sources and destinations $Q$). Thus, the routing time of any scheduling algorithm is $\Omega(C^* + D^*)$.

Maggs *et al.* [20], give a strategy to select paths in the mesh such that the congestion achieved by the paths is $C = O(dC^* \log n)$ with high probability. In that algorithm the dilation is $D = O(m \log n)$. Busch *et al.* [10] improve the dilation to be $D = O(d^2 D^*)$, while preserving the congestion bound. The paths obtained by the algorithms in [10, 20], are constructed from the concatenation of $O(\log n)$ dimension-by-dimension shortest paths between random nodes in the mesh. Since a dimension-by-dimension shortest path between two nodes on the mesh contains at most $d$ bends, the number of bends that a packet makes is $b = O(d \log n)$. We summarize these observations in the following lemma.

**Lemma 3.9 ([10, 20])** *For any batch routing problem $(M, \Pi, Q)$ on the d-dimensional mesh, there exists paths $\mathcal{P}$ that use at most $b = O(d \log n)$ bends which achieves near optimal congestion $C = O(dC^* \log n)$ and near optimal dilation $D = O(d^2 D^*)$.*

Lemma 3.9 combined with Theorem 3.7 give the following result:

**Theorem 3.10 (Near Optinal Batch Routing on the Mesh)** *For any batch-routing problem $(M, \Pi, Q)$ on the mesh, there exist packet paths $\mathcal{P}$ and a respective direct schedule with routing time $rt(M, \Pi, \mathcal{P}) = O(d^2 C^* \log^2 n + d^2 D^*)$, with high probability.*

Theorem 3.10 implies that there is a direct routing schedule with routing time which is within a factor $d^2 \log^2 n$ from the optimal.

## 3.3 Butterfly

We consider the $n$-input butterfly network $B$, where $n = 2^y$, $y \geq 0$ (see [17]). In the butterfly network $B$, 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 $l$th bit complemented. Note that there is a unique path from an input node to an output node and the length of the path is $\lg n + 1$.

We study permutation routing problems on the butterfly, in which each input node is the source of one packet, and each output node is the destination of one packet. In order to solve the permutation problems efficiently, we will use Valiant's scheme [28, 29] which uses two back to back butterflies where packets choose random intermediate nodes before reaching their destinations. Thus, we first study the random destination problem on a butterfly, which we then use to solve permutation routing problems.

### 3.3.1 Random Destinations on Butterfly

Consider a random destinations routing problem $(B, \Pi, Q)$ on a butterfly $B$, in which every input node is the source of one packet and the destination of each packet is chosen independently and uniformly at random among the output nodes of the butterfly. Since the path from a source to a destination is unique, we immediately obtain a path-routing problem $(B, \Pi, \mathcal{P})$, which we solve using the greedy direct algorithm.

A trivial lower bound on the routing time is $\lg n + 1$, the length of any path. We will show that the greedy direct algorithm 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 interferes with at most $\frac{3}{2} \lg n$ other packets w.h.p. This implies that the maximum weight degree in the dependency graph $\mathcal{D}$ is at most $\frac{3}{2} \lg n$ (since the paths are shortest paths, as explained below), and then using Lemma 3.1 we obtain the advertised bound.

Consider a packet $\pi_i \in \Pi$ with path $(v_0, v_1, \ldots, v_{\lg n})$. Let $m_k$, $k = 1, \ldots, \lg n - 1$, be the number of other packets that could possibly interfere with packet $\pi_i$, with the first interference edge being $(v_k, v_{k+1})$ (note that it is not possible to have interferences on edge $(v_0, v_1)$). For any one of the $m_k$ packets, let $q_k$ be the probability that it actually uses the edge $(v_k, v_{k+1})$. Note that all the $m_k$ packets independently choose their destinations, and we will see in the next lemma that $q_k$ does not depend on the specific packet, only on $k$. The following lemma follows from the properties of the butterfly network.

**Lemma 3.11** $m_0 = 0$, $m_k = 2^{k-1}$, and $q_k = 2^{-(k+1)}$, for $k = 1, \ldots, \lg n - 1$.

**Proof:** Clearly $m_0 = 0$. Let $\pi_j$ be some packet that interferes for the first time with $\pi_i$ on edge $(v_k, v_{k+1})$. Packet $\pi_j$ arrives at $v_k$ using edge $(w, v_k)$ with $w \neq v_{k-1}$. The number of input nodes that can reach $w$ is $2^{k-1}$, and $\pi_j$ could have originated from any of these nodes. Thus, $m_k = 2^{k-1}$.

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

Let $X_i$ be the number of different other packets that interfere with packet $\pi_i$. Let $x_j^{(i)}$ be the Bernoulli random variable that equals 1 if packet $\pi_j$ interferes with packet $\pi_i$, and let $q_j^{(i)} = \mathbf{P}[x_j^{(i)} = 1]$. Clearly, $X_i = \sum_j x_j^{(i)}$, and $x_j^{(i)}$ are independent for different $j$. Let $\mu = \mathbf{E}[X_i] = \sum_j q_j^{(i)}$. Using Lemma 3.11 we obtain

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

Thus, the expected number of packets that use packet $\pi_i$'s path is $\frac{1}{4}(\lg n - 1)$, independent of $\pi_i$ and the specific path used by packet $\pi_i$. Note that in the dependency graph $\mathcal{D}$, $X_i$ is equal to $W(\pi_i)$. This is a consequence of the fact that since the packet paths are the shortest, when two packets interfere then for every common edge of their paths the resulting weight on the dependency graph $\mathcal{D}$ is the same.

We will use the following version of the Chernoff bound to get a concentration result for $X_i$, which will give an upper bound of the routing time.

**Lemma 3.12 ([23])** *Let* $y_1, \ldots, y_m$ *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* $\alpha > 2e$, $\mathbf{P}[Y > \alpha\mu] < 2^{-\alpha\mu}$.

We now give the upper bound on the routing time for the butterfly when using the greedy direct algorithm.

**Theorem 3.13 (Random Destinations Routing Time on Butterfly)** *For a random-destinations routing problem* $(B, \Pi, Q)$, *on the n-input butterfly* $B$, *using the unique shortest paths* $\mathcal{P}$, *the routing time of the greedy direct algorithm satisfies* $\mathbf{P}\left[rt_{Gr}(B, \Pi, \mathcal{P}) \leq \frac{5}{2}\lg n\right] > 1 - 2\sqrt{2}n^{-\frac{1}{2}}$.

**Proof:** For packet $\pi_i$, define the event $E_i$ by $E_i = \{X_i > \frac{\alpha}{4}\lg n\}$ for some $\alpha > 2e$. Applying the Chernoff bound in Lemma 3.12 and using Equation (1), we get

$$\mathbf{P}[E_i] = \mathbf{P}\left[X_i > \frac{\alpha}{4}\lg n\right] \leq \mathbf{P}[X_i > \alpha\mu] < \frac{2^{\alpha/4}}{n^{\alpha/4}}.$$

The identity $\mathbf{P}[\max_i X_i \leq \alpha \lg n] = 1 - \mathbf{P}[\cup_i E_i]$ and the union bound then give

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

Since $\max_i W(\pi_i) = \max_i X_i$ and $D = \lg n$, we conclude by applying Lemma 3.1 with the choice $\alpha = 6 > 2e$. ∎

### 3.3.2 Permutations on Butterfly

It is known that in the butterfly there exist permutation routing problems with congestion at least $\Omega(\sqrt{n})$, i.e. some edges are hot-spots (see [23, Section 4.2]). In order to avoid hot-spots, Valiant [28, 29] proposed the following alternative scheme to route permutation routing problems in a butterfly-like network. Take two butterflies and connect them back to back, so that the outputs of the first butterfly are connected to the respective outputs of the the second butterfly. The permutation problem is for this butterfly network: each packet has

source on the input of the first butterfly and destination on the input of the second butterfly. The idea is to allow each packet to choose uniformly and at random an intermediate node on the output of the first butterfly. The path is then given by the source to the random intermediate node followed by the 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 3.13 twice to obtain:

**Theorem 3.14 (Permutation Routing Time on Butterfly)** *For a batch permutation routing problem $(B', \Pi, Q)$ on the $n$-input back to back butterfly $B'$, and using the paths $\mathcal{P}$ constructed using Valiant's scheme, the routing time of the greedy direct algorithm satisfies* $\mathbf{P}\left[rt_{Gr}(B', \Pi, \mathcal{P}) \leq 5 \lg n\right] > 1 - 4\sqrt{2}n^{-\frac{1}{2}}.$

## 3.4 Hypercube

We consider the $n$-hypercube network $H$ with $n = 2^y$ nodes, $y \geq 0$ (see [17]). In the hypercube network $H$ 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 $\lg n$. Note that the distance (shortest path) between any two nodes is $\leq \lg n$. Between a pair of nodes there exist many shortest paths.

We study permutations on the hypercube. In order to solve the permutations efficiently, we first send the packets to random intermediate nodes and then to the destination. Thus, we first study random destination problems.

### 3.4.1 Random Destinations on Hypercube

Consider a random destinations routing problem $(H, \Pi, Q)$, in which every node is the source of one packet, and each packet chooses its destination uniformly and at random. We will construct a path-routing problem with paths $\mathcal{P}$ such that the packets use (left-to-right) *bit-fixing paths* from their source to the destination as follows. Let $\pi$ be a packet which has to be routed from source $s(\pi)$ to destination $\delta(\pi)$; flip the leftmost bit at which the labels of $s(\pi)$ and $\delta(\pi)$ differ and send packet $\pi$ 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, for this selection of paths $D \leq \lg n$, since no more than $\lg n$ bits are flipped.

We will show that the direct greedy algorithm has routing time bounded by $7 \lg n$, w.h.p., which is optimal to within constant factors because it can be shown that $D \geq \frac{1}{4} \lg n$ w.h.p (using a simple Chernoff bounding argument). As with the butterfly analysis (Section 3.3),

let $X_i$ be the number of other different packets that packet $i$ interferes with. We will use the following result which is adapted from [23, Theorem 4.6]:

**Lemma 3.15 ([23])** $\mathbf{P}\left[\max_i X_i \leq 6 \lg n\right] > 1 - 1/(32n)$.

Thus, the maximum node weight degree in the dependency graph $\mathcal{D}$ is at most $6 \lg n$, with probability at least $1 - 1/(32n)$. Since $D \leq \lg n$, Lemma 3.1 implies that the routing time of the greedy algorithm is at most $7 \lg n$ w.h.p. We have the following theorem:

**Theorem 3.16 (Random Destinations Routing Time on Hypercube)** *For a random destination routing problem $(H, \Pi, Q)$ on the $n$-hypercube $H$, when choosing bit-fixing paths $\mathcal{P}$, the routing time of the greedy direct algorithm satisfies $\mathbf{P}\left[rt_{Gr}(H, \Pi, \mathcal{P}) \leq 7 \lg n\right] > 1 - 1/(32n)$.*

### 3.4.2 Permutations on Hypercube

It is known that on the $n$-hypercube there exist permutation routing problems with congestion at least $\Omega(\sqrt{n/\log n})$, i.e. some edges are hot-spots (see [23, Section 4.2]). In order to avoid hot-spots, we will use Valiant's scheme [28, 29]: for any permutation problem, we will construct paths $\mathcal{P}$ 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 the destination. This routing problem is the combination of two random destinations problem. Thus, we can apply Theorem 3.16 twice to obtain Theorem 3.17.

**Theorem 3.17 (Permutation Routing Time on Hypercube)** *For a permutation routing problem $(H, \Pi, Q)$ on the $n$-hypercube $H$, using Valiant's scheme with bit-fixing paths $\mathcal{P}$, the routing time of the greedy direct algorithm satisfies $\mathbf{P}\left[rt_{Gr}(H, \Pi, \mathcal{P}) < 14 \lg n\right] > 1 - 1/(16n)$.*

# 4 Computational Complexity of Direct Routing

In this section, we show that direct routing, and approximate versions of it, are NP-complete. First, we introduce the formal definition of the direct routing decision problem. In our reductions, we will use the well known NP-complete problem VERTEX COLOR, the vertex coloring problem [14], which asks whether a given graph $G$ is $\kappa$-colorable. The chromatic number, $\chi$ is the smallest $\kappa$ for which $G$ is $\kappa$-colorable. An algorithm approximates $\chi$ with approximation ratio $q$ if on any input $G$, the algorithm outputs $u$ such that $\chi \leq u$ and $u/\chi \leq q$. Typically, $q$ is expressed only as a function of the number of vertices in $G$.

It is known [12] that unless $\mathsf{P}=\mathsf{NP}^{\dagger}$, there does not exist a polynomial time algorithm to approximate $\chi$ with approximation ratio $N^{1/2-\epsilon}$ for any constant $\epsilon > 0$.

By polynomially reducing coloring to direct routing, we will obtain hardness and inapproximability results for direct routing. We now formally define a generalization of the direct routing decision problem which allows for collisions. We say that an injection schedule is a valid $K$-collision schedule if at most $K$ collisions occur during the course of the routing. (The number of collisions is the total number of times (over all pairs of packets) that a pair of packets interferes; thus, a pair of packets may contribute more than once to the total number of collisions.)

**Problem:** Approximate Direct Route

**Input:** A path-routing problem $(G, \Pi, \mathcal{P})$ and integers $T, K \geq 0$,

**Question:** Does there exist a valid $k$–collision direct routing schedule, for some $k \leq K$ with maximum injection time $\tau_{max} \leq T$?

The problem Direct Route is the restriction of Approximate Direct Route to instances where $K = 0$. Denoting the maximum injection time of a valid $K$-collision injection schedule by $T$, we define the *$K$-collision injection number* $\tau_K(G, \Pi, \mathcal{P})$ for a direct routing instance as the minimum value of $T$ for which a valid $K$-collision schedule exists. We say that a schedule approximates $\tau_K(G, \Pi, \mathcal{P})$ with ratio $q$ if it is a schedule with at most $K$ collisions and the maximum injection time for this schedule approximates $\tau_K(G, \Pi, \mathcal{P})$ with approximation ratio $q$.

We now show that direct routing is $\mathsf{NP}$-hard using a polynomial-time reduction from the vertex coloring problem.

**Theorem 4.1 (NP-Hardness)** Direct Route *is* $\mathsf{NP}$-*Hard.*

**Proof:** We show that here exists a polynomial time reduction from any instance $(G, \kappa)$ of Vertex Color to an instance $(G', \Pi, \mathcal{P}, T = \kappa - 1)$ of Direct Route.

Illustrated in Figure 3 is a path-routing problem for which there are $N$ packets, and all the packets form a synchronized clique of size $N$ in the dependency graph $\mathcal{D}$. There are $L$ levels in this routing problem. Each path in the figure (ending with an arrow) is the path for 1 packet. 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

---

$^{\dagger}$It is also known that if $\mathsf{NP} \nsubseteq \mathsf{ZPP}$ then $\chi$ is inapproximable to within $N^{1-\epsilon}$, however we cannot use this result as it requires both upper and lower bounds.
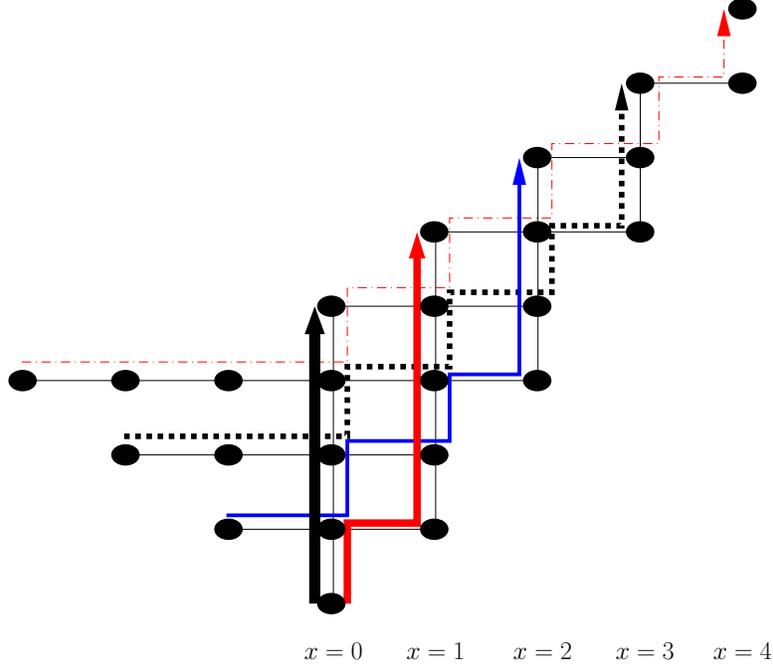
$x = 0 \quad x = 1 \quad x = 2 \quad x = 3 \quad x = 4$

Figure 3: A mesh routing problem

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 $N = L+1$. 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.

(i) Let $j > i \geq 0$. 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 $\pi_1, \pi_2$ follow paths $p_1, p_2$ which share an edge $e$ then $d_{p_1}(e) = d_{p_2}(e)$: 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 collide at $e$.

(iii) The length of the level $i$ path is $L + 2i$.

Since every pair of packets is synchronized, in the dependency graph $\mathcal{D}$, the packets form a synchronized clique of size $N$. Given an instance $I = (G, \kappa)$ of VERTEX COLOR, we now show how to reduce it to the corresponding instance $I' = (G', \Pi, \mathcal{P}, T = \kappa - 1)$ of DIRECT

20

ROUTE. Each node in $G$ corresponds to a packet in $\Pi$. The paths are initially as illustrated in the routing problem above with $L = N - 1$. We now show how to transform this routing problem so that the dependency graph $\mathcal{D}$ for the transformed routing problem is isomorphic to $G$. This is the instance $I'$ to which we reduce $I$.

Currently the dependency graph is $K_N$, an $N$-clique. We need to remove some of the edges to get $G$. If there is no edge between two nodes $u, v$ in $G$, this means that the corresponding packets must not collide. Suppose that the two packets $u, v$ collide at edge $e = (x, y)$ in $G'$. We augment $G'$ by adding two nodes $x'$ and $y'$ corresponding to $x$ and $y$ respectively. In $G'$, $x'$ (resp. $y'$) is adjacent to the same nodes as $x$ (resp. $y$) is. Suppose that the path of $u$ was $(\ldots, a, x, y, b, \ldots)$. We alter the $u$'s path to $(\ldots, a, x', y', b, \ldots)$, and do not change $v$'s path. The following figure illustrates the construction,
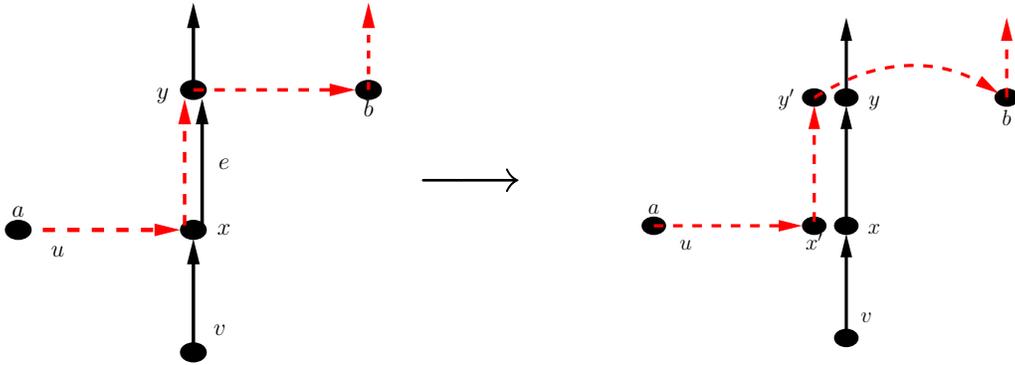


Notice that the paths corresponding to $u$ and $v$ no longer interfere. Further, the lengths of $u$ and $v$ and their relationships with any other paths have not been altered in any way. We perform this construction in every edge $(u, v)$ in $G$ that does not exist. The resulting dependency graph for the paths in $G'$ is isomorphic to $G$, and every edge in the dependency graph is synchronized.

Since every two packets that interfere in this routing problem are synchronized, they cannot be assigned the same injection time in any valid schedule. Interpreting the injection time of a packet as the color of that packet, we see that any valid direct routing schedule induces a valid coloring of $\mathcal{D}$. Since $\mathcal{D}$ is is isomorphic to $G$, this will also induce a valid coloring of $G$. Further, a valid coloring of $G$ and hence of $\mathcal{D}$ will induce a valid set of injection times since no two packets that interfere (and hence are adjacent in $\mathcal{D}$) will have the same injection time. Thus the answer to instance $I$ of VERTEX COLOR is true if and only if the answer to instance $I'$ of DIRECT ROUTE is true. The proof is concluded by noting that the construction of $I'$ is clearly polynomial in $N$. ∎

DIRECT ROUTE is in NP, as one can check the validity and routing time of a direct routing schedule, by traversing every pair of packets, so DIRECT ROUTE is NP complete.

Further, we see that the reduction in Theorem 4.1 is gap preserving with gap preserving parameter $\rho = 1$ [15].

**Theorem 4.2 (Inapproximability of Collision Injection Number)** *A polynomial time algorithm that approximates $\tau_K(G, \Pi, \mathcal{P})$ with ratio $r$, for an arbitrary path-routing problem, yields a polynomial time algorithm that approximates the chromatic number of an arbitrary graph with ratio $r + K + 1$. In particular, choosing $K = O(r)$ preserves the approximation ratio.*

**Proof:** Consider the same instance of direct routing as in the proof of Theorem 4.1 (see also Figure 3). Let $K$ be the number of collisions allowed and suppose that a polynomial time algorithm exists to approximate $\tau_K(G', \Pi, \mathcal{P})$.

We will prove a stronger result than required for the theorem. Suppose that $\ell \leq \tau_K(G', \Pi, \mathcal{P}) \leq u$. The approximation ratio is $r$ if $u/\tau_K(G', \Pi, \mathcal{P}) \leq r$, and we say the approximation is to within $q$ if $u/\ell \leq q$. We will show that $\ell_\chi \leq \chi \leq u_\chi$, where $\ell_\chi = \ell + 1$ and $u_\chi = u + K + 1$. From these bounds, we will obtain the theorem.

The lower bound follows immediately because $\ell \leq \tau_K(G', \Pi, \mathcal{P}) \leq \tau(G', \Pi, \mathcal{P}) = \chi - 1$. To prove the upper bound, suppose that $\tau_K(G', \Pi, \mathcal{P}) \leq u$, so a $K$-collision direct routing schedule with maximum injection time $u$ exists. For each collision, we can arbitrarily pick one of the packets involved in the collision (there can be at most two packets involved in a collision) and place it in a set $\mathcal{C}$, if the packet is not already in the set. The packets in $\Pi - \mathcal{C}$ therefore do not collide. We can assign the times $u + 1, u + 2, \ldots, u + |\mathcal{C}|$ to the packets in $\mathcal{C}$, and since all the packets are synchronized, none of the packets now collide, so we now have a valid direct routing schedule with maximum injection time $u' \leq u + |\mathcal{C}| \leq u + K$, since $|\mathcal{C}| \leq K$. Thus, $\chi = \tau(G', \Pi, \mathcal{P}) + 1 \leq u + K + 1 = u_\chi$.

Note that since $u/\tau_K(G', \Pi, \mathcal{P}) \leq r$, and

$$\frac{u}{\tau_K(G', \Pi, \mathcal{P})} \geq \frac{u}{\tau(G', \Pi, \mathcal{P})} \geq \frac{u}{(\tau(G', \Pi, \mathcal{P}) + 1)} = \frac{u}{\chi},$$

we have that $u/\chi \leq r$, and so $u_\chi/\chi \leq r + K + 1$, since $\chi \geq 1$. Further, $u_\chi/\ell_\chi \leq u/\ell + K + 1 = q + K + 1$.

Thus, an approximation for $\tau_K(G', \Pi, \mathcal{P})$ to within $q$ or with ratio $r$ yields an approximation for $\chi$ to within $q + K + 1$ or with ratio $r + K + 1$,, respectively. The approximation is polynomial time since the reduction is polynomial time. ∎

The previous theorem states that any approximation scheme for direct routing yields an approximation scheme for vertex coloring. Since it is known that vertex coloring is inapproximable, any inapproximability result for vertex coloring implies a corresponding inapproximability result for direct scheduling. In particular we will use the known result

that $\chi$ is inapproximable with ratio $N^{1/2-\epsilon}$ unless $\mathsf{P} = \mathsf{NP}$. Choosing $K = O(N^{1/2-\epsilon})$, we have:

**Corollary 4.3 (Inapproximability of Direct Scheduling)** *Unless* $\mathsf{P}=\mathsf{NP}$, *for* $K = O(N^{1/2-\epsilon})$, *there is no polynomial time algorithm to determine a valid $K$-collision direct schedule that approximates $\tau_K(G, \Pi, \mathcal{P})$ with ratio $O(N^{1/2-\epsilon})$ for any $\epsilon > 0$.*

In words, even if we allow $\Omega(N^{1/2})$ collisions, one cannot efficiently compute an approximately direct schedule with minimum injection time close to the minimum possible (with approximation ratio $O(N^{1/2})$).

# 5 Lower Bounds for Buffering

In direct routing the nodes have no buffers to store packets in transit. However, in store-and-forward routing, the nodes do have buffers. The buffers help to improve the routing time. Here we study the question of how much the buffers help in reducing the routing time.

We say that a packet is *buffered* if it stays at the buffer of a node for a time step, while at the previous time step the packet was also in the same node. We say that a packet is *buffered k times* it is buffered for $k$ time steps (the time steps may not be consecutive and the packet may be buffered at different nodes). The *amount of buffering* of a routing algorithm is the total sum of the number of times that packets are buffered (summed over all the packets).

We show that there is a tradeoff between the amount of buffering and the routing time. The more the buffering the better the routing time. The worst routing time is when the routing is direct. In order to obtain our results, we first give a "hard" routing problem for which any direct routing algorithm requires a large routing time $\Omega(C \cdot D)$ (compared to the optimal $O(C + D)$). Then, we show that in order to drop the routing time to the optimal, many packets in the network need to be buffered at least once. Finally, we strengthen our result and show that many packets need to be buffered many times.

## 5.1 Packets Buffered Once

We construct a "hard" path-routing problem for which *any* direct routing algorithm has routing time $rt = \Omega(C \cdot D) = \Omega((C + D)^2)$, which is asymptotically worse than optimal. We then analyze the amount of buffering that would be required to attain near optimal routing time, which results in a lower bound on the amount of buffering needed by *any* store-and-forward algorithm.

**Theorem 5.1 (Hard Routing Problem)** *There exists a path-routing problem for which every direct algorithm has routing time $\Omega(C \cdot D) = \Omega((C + D)^2)$*

**Proof:** We construct a path-routing problem for which the dependency graph is a synchronized clique. The paths are as in Figure 3, and the description of the routing problem is given in the proof of Theorem 4.1. The only difference is that $c$ packets use each path. The congestion is $C = 2c$ and the dilation is $D = 3L$. Since every pair of packets is synchronized, Lemma 2.1 implies that $rt(G, \Pi, \mathcal{P}) \geq N$. Since $N = c(L+1) = \frac{C}{2}(\frac{D}{3} + 1)$, $rt(G, \Pi, \mathcal{P}) = \Omega(C \cdot D)$. Choosing $c = \Theta(\sqrt{N})$ and $L = \Theta(\sqrt{N})$, we have that $C + D = \Theta(\sqrt{N})$ so $C + D = \Theta(\sqrt{C \cdot D})$. ∎

Let problem $A$ denote the routing problem in the proof of Theorem 5.1. We would like to determine how much buffering is necessary in order to decrease the routing time for routing problem $A$. Let $T$ be the maximum injection time (so the routing time is bounded by $T+D$). We give a lower bound on the number of packets that need to be buffered at least once:

**Lemma 5.2** *In routing problem $A$, if $T \leq \alpha$, then at least $N - \alpha$ packets are buffered at least once.*

**Proof:** If $\beta$ packets are not buffered at all, then they form a synchronized clique, hence $T \geq \beta$ (Lemma 2.1), i.e., $\alpha \geq T \geq \beta$. To conclude, note that $N - \beta \geq N - \alpha$ packets are buffered at least once. ∎

If the routing time is $O(C + D)$, then $\alpha = O(C + D)$. Choosing $c$ and $L$ to be $\Theta(N^{1/2})$, we have that $\alpha = O(N^{1/2})$, and so from Lemma 5.2, the number of packets buffered is at least $N - \alpha = \Omega(N)$:

**Corollary 5.3** *There exists a routing problem for which any algorithm will buffer $\Omega(N)$ packets at least once to achieve asymptotically optimal routing time.*

## 5.2 Packets Buffered Many Times

We now construct a path-routing problem $B$ which forces packets to be buffered multiple times. In routing problem $A$ normalize the lengths of the packets so that all packet path lengths are $3L$; in order to do so add edges at the end of the paths when necessary. Denote by $A'$ the respective network. We then construct a routing problem $B$ by concatenating $k$ identical copies of $A'$, denoted $A'_1, \ldots, A'_k$. In problem $B$ a packet $\pi$ traverses each $A'_i$ one after the other in sequence, so that the path of a packet $\pi$ is the concatenation of the paths in each copy of $A'$ (the path in each $A'_i$ is the normalized path of a routing problem $A$). When a packet exits $A'_i$, it enters $A'_{i+1}$ at exactly the same level as in $A'_i$. In routing problem $B$, we set $C = 2c$, $D = 3kL$ and $N = c(L+1) = \Theta(C \cdot D/k)$. The packets in $B$ are synchronized in each $A'_i$, which leads to the following result.

**Lemma 5.4** *For path-routing problem $B$, if $T \leq \alpha$, then at least $N - j\alpha$ packets each need to be buffered, at least $j$ times, in the concatenation of $A'_1, \ldots, A'_j$, where $1 \leq j \leq k$.*

**Proof:** We prove the claim by induction on $j$. For $j = 1$ the claim follows immediately from Lemma 5.2. Let's assume that the claim holds for all $j < r$, where $r > 1$; we will prove that the claim holds for $j = r$.

Let $x_i$ denote the packets which are buffered $i$ times in $A'_1, \ldots, A'_{r-1}$, where $i \geq 0$. Clearly, $N = \sum_{i \geq 0} |x_i|$. From those packets, the only packets that may be buffered less than $r$ times in $A'_1, \ldots, A'_r$, are packets from $x_0, \ldots, x_{r-1}$. Let $X = \sum_{i=0}^{r-2} |x_i|$; from the induction hypothesis, we have that $|x_{r-1}| \geq N - (r-1)\alpha$, and since $N \geq X + |x_{r-1}|$, we conclude that $X \leq (r-1)\alpha$. Let $Y$ be the packets of $x_{r-1}$ which will not be buffered in $A'_r$. Since all packets in $Y$ are buffered exactly the same number of times before entering $A'_r$, these packets are all synchronized in $A'_r$, and so must be injected at different times (Lemma 2.1). Thus, $T \geq Y$, and so $\alpha \geq T \geq Y$. Therefore the number of packets buffered less than $r$ times in $A'_1, \ldots, A'_r$ is at most $X + Y \leq (r-1)\alpha + \alpha = r\alpha$. Consequently, at least $N - r\alpha$ packets are buffered at least $r$ times in $A'_1, \ldots, A'_r$. ∎

By constructing $B$ appropriately, we strengthen Corollary 5.3, and we obtain a tradeoff between routing time and buffering for *any* routing algorithm (direct or not):

**Theorem 5.5 (Buffering-Routing Time Tradeoff)** *There exists a path-routing problem which, for any $0 \leq \epsilon < \frac{1}{2}$, requires $\Omega(N^{(4-2\epsilon)/3})$ buffering in order to obtain a routing time that is a factor $O(N^\epsilon)$ from optimal.*

**Proof:** From Lemma 5.4, at least $N - k\alpha$ packets are each buffered at least $k$ times in routing problem $B$. Suppose we wish to obtain a routing time that is within $O(N^\epsilon)$ of optimality. Let $0 \leq \epsilon < \frac{1}{2}$. In this case, $T \leq \alpha = \lambda(C + D)N^\epsilon$ for some $\lambda > 0$. In routing problem $B$, we select the parameters $c, L, k$ as follows: $c = N^{(2-\epsilon)/3}$; $L = N^{(1+\epsilon)/3} - 1$; and, $k = \gamma N^{(1-2\epsilon)/3}$, where $\gamma$ is a constant chosen small enough so that $\lambda\gamma(2 + 3\gamma) < 1$ (for simplicity, assume that $c, L, k$ are integers). After some elementary algebra, we have that

$$N - k\alpha = (1 - \lambda\gamma(2 + 3\gamma))N - 3\lambda\gamma^2 N^{(2-\epsilon)/3},$$

i.e., $N - k\alpha = \Omega(N)$. Since at least $N - k\alpha$ packets are buffered at least $k$ times (Lemma 5.4), the total amount of buffering is at least $k(N - k\alpha) = \Omega(N^{(4-2\epsilon)/3})$. ∎

Setting $\epsilon = \Omega(1/\log N)$, we obtain a lower bound on the buffering required by any algorithm that guarantees an asymptotically optimal routing time of $O(C + D)$:

25

**Corollary 5.6 (Lower Bound on Buffering)** *There exist path-routing problems for which any routing algorithm with asymptotically optimal routing time requires* $\Omega(N^{4/3})$ *buffering.*

Note that asymptotically optimal algorithms exist for store-and-forward routing [19, 22, 24]. Corollary 5.6 gives a lower bound on how much buffering is necessary on these algorithms.

# 6 Conclusion

We have given a comprehensive analysis of direct routing. We studied the efficiency of the algorithms in terms of the congestion $C$ and the dilation $D$. We gave a simple greedy algorithm which is asymptotically optimal in the worst case. We then demonstrated that on many common topologies, direct routing is efficient. In particular, we considered the tree, $d$-dimensional mesh, butterfly and hypercube. One important implication of our results is that on particular network topologies and for certain batch routing problems, direct routing achieves routing times close to the routing times of store-and-forward algorithms.

We then continued to present hardness results for direct routing which show that direct routing is as hard as vertex coloring. Thus, in order to compute efficient direct routing schedules in polynomial time, we need to allow packets to collide.

Finally, we considered the connection between the efficiency of direct routing and buffering requirements of store-and-forward routing algorithms. We showed the existence of a hard routing problem in which *any* buffered routing algorithm requires $\Omega(N^{4/3})$ total packet buffering to obtain a routing time which is a constant factor from the optimal routing time.

# References

[1] M. Adler, S. Khanna, R. Rajaraman, and A. Rosen. Time-constrained scheduling of weighted packets on trees and meshes. In *Proceedings of 11th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 1999.

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

[3] S. Alstrup, J. Holm, K. de Lichtenberg, and M. Thorup. Direct routing on trees. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 98)*, pages 342–349, 1998.

[4] I. Ben-Aroya, D. D. Chinn, and A. Schuster. A lower bound for nearly minimal adaptive and hot potato algorithms. *Algorithmica*, 21(4):347–376, August 1998.

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

[6] B. Bollobás. *Random Graphs, Second Edition*. Cambridge University Press, new york edition, 2001.

[7] 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.

[8] 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, Aug. 2002.

[9] 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.

[10] C. Busch, M. Magdon-Ismail, and J. Xi. Optimal oblivious path selection on the mesh. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2005)*, Denver, Colorado, USA, April 2005.

[11] R. Cypher, F. M. auf der Heide, C. Scheideler, and B. Vöcking. Universal algorithms for store-and-forward and wormhole routing. pages 356–365, 1996.

[12] U. Feige and J. Kilian. Zero knowledge and the chromatic number. In *IEEE Conference on Computational Complexity*, pages 278–287, 1996.

[13] 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, Oct. 1992.

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

[15] D. S. Hochbaum. *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, New York, 1997.

[16] C. 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.

[17] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays - Trees - Hypercubes*. Morgan Kaufmann, San Mateo, 1992.

[18] 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.

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

[20] B. M. Maggs, F. M. auf der Heide, B. Vocking, and M. Westermann. Exploiting locality for data management in systems of limited bandwidth. In *IEEE Symposium on Foundations of Computer Science*, pages 284–293, 1997.

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

[22] F. Meyer auf der Heide and B. Vöcking. Shortest-path routing in arbitrary networks. *Journal of Algorithms*, 31(1):105–131, Apr. 1999.

[23] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge, UK, 2000.

[24] R. Ostrovsky and Y. Rabani. Universal $O(congestion+dilation+\log^{1+\varepsilon} N)$ 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.

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

[26] Y. Rabani and É. 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.

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

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

[29] 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.

[30] 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.