Alcom-FT Technical Report Series
ALCOMFT-TR-03-133

An Analysis of Direct Routing: Why do we need Buffers?

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

November 5, 2003

Abstract

Direct routing is a special case of bufferless routing in which N packets must be routed without conflicts along specific paths. We show, by a reduction from vertex coloring, that direct routing in NP-complete, and that it is as hard to approximate as coloring. This result shows that buffering is necessary in order to tractably compute routing schedules. Further, we construct a hard routing problem for which *any* routing algorithm requires $\Omega(N^{(4-2\epsilon)/3})$ total packet buffering to approximate an optimal routing schedule with approximation ratio $O(N^{\epsilon})$, for any $0 \leq \epsilon < 1/2$.

We then study algorithms for efficient direct routing. We give a *greedy* algorithm for general direct routing problems with direct routing time that is worst case optimal. Finally, we consider many interesting routing problems on commonly used network topologies for which variants of the greedy algorithm achieve optimal or near optimal (within logarithmic factors) routing time. In particular, we study the Tree, the d-dimensional Mesh, the Butterfly and the Hypercube.

*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

In *direct routing* packets are routed along specific paths so that they do not conflict with each other; thus, packets are routed "directly" without being buffered at intermediate nodes. Direct routing is unavoidable when buffers are costly or unavailable (for example optical networks [27]), collisions cannot be tolerated (for example power aware routing in sensor networks [9]) or packet delivery times must be guaranteed (for example real-time systems applications).

We show that efficient direct routing is hard, in general. In order to make a direct routing problem tractable or to approximate the optimal routing schedule, a routing algorithm has to tolerate packet conflicts. This implies that packets need to be buffered at intermediate nodes. We show that there is a tradeoff between the amount of buffering required (number of times packets are delayed) and the efficiency of the routing algorithm. In particular, we show the following results.

i. By reduction from vertex coloring, we show that direct routing is NP-complete. The reduction is gap preserving with parameter $\rho = 1$, so direct routing is as hard to approximate as coloring is. We show that an approximate version of direct routing which allows packet conflicts is also NP-complete, which leads to a connection between the hardness of direct routing and the necessity of buffers. Using existing inaproximability results for coloring, we show that in order to be able to polynomially compute an $O(N^{1/2-\epsilon})$ approximation to an optimal routing schedule, $\Omega(N^{1/2-\epsilon})$ buffering of packets is required, for any $0 \leq \epsilon < 1/2$.

ii. We construct a "hard" routing problem for which *any* algorithm that has routing time within $O(N^{\epsilon})$ of optimal requires $\Omega(N^{(4-2\epsilon)/3})$ total packet buffering, where $0 \leq \epsilon < 1/2$.

We then study particular direct routing problems. 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 lower bound on the routing time of any routing algorithm (direct or not) is given by $\Omega(C + D)$. We give a polynomial time *greedy* direct routing algorithm with worst case routing time $O(C \cdot D)$. This greedy algorithm is worst case optimal as we construct a routing problem for which any direct routing algorithm has routing time $\Omega(C \cdot D)$, where $C \cdot D = \Omega((C + D)^2)$. Using variants of the greedy algorithm, we obtain optimal or near optimal direct routing schedules for the following network topologies (rt denotes the routing time):

i. *Tree:* for arbitrary routing problems on arbitrary trees $rt \leq 2C + D - 2$.

ii. *Multi-dimensional Mesh:* for routing problems in which packets bend in their paths at most b times $rt \leq 4(b+1)(C-1)+D$. We apply this result for permutation routing on the 2-dimensional $\sqrt{n} \times \sqrt{n}$ mesh where we use one-bend paths in order to obtain $rt = O(\sqrt{n})$, which is worst case optimal. On the d-dimensional mesh with n nodes, we obtain $rt = O(d^2 \cdot \log^2 n \cdot rt^*)$ with high probability (w.h.p.), by using paths with $\log n$ bends, where rt^* is the minimum possible routing time achievable for the given sources and destinations of the packets.

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 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.

Related Work. The only previous work known on direct routing is for trees. In particular, Symvonis [28] 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.

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, [1, 25, 31]. In hot-potato routing, when packets conflict they get deflected, i.e. use alternative paths to their destination, [3, 4, 6, 8, 10, 15]. In [3], the authors study lower bounds for near-greedy hot-potato algorithms on the mesh.

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 [17, 19, 22, 24, 26]. There are also hot-potato algorithms which come close to the lower bound $\Omega(C + D)$ for leveled networks [7] and vetrex symmetric networks [21]. To our knowledge, our paper is the first that addresses the question of how much buffering is necessary in order to solve routing problems efficiently.

The remainder of the paper is organized as follows. First, we discuss how hard direct routing is with respect to offline computation and routing time, which leads to lower bounds on buffering. Next, we consider algorithms for direct routing. In particular, we give a general greedy algorithm for direct routing and applications of this algorithm to the Mesh, the Tree, the Butterfly, and the Hypercube. Finally, we end with some concluding remarks.

2 Preliminaries

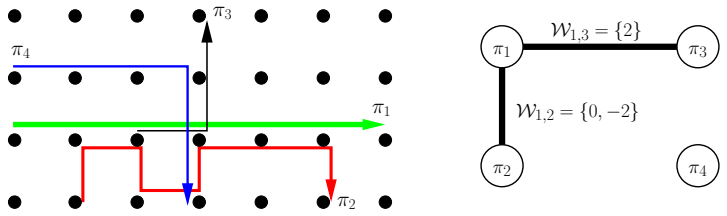
Problem Definition. Let's first define the problem and introduce the definitions that we will subsequently need. We are given a graph $G = (V, E)$ with $n \geq 1$ nodes, and a set of packets $\Pi = \{\pi_i\}_{i=1}^{N}$. Each packet π_i is to be routed from its source, $\mathrm{s}(\pi_i) \in V$, to its destination, $\delta(\pi_i) \in V$, along a pre-specified path p_i. The nodes in the graph 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 p is a sequence of vertices $p = (v_1, v_2, \ldots, v_k)$. Two paths p_1 and p_2 *collide* if they share an edge in the same direction. We also say that their respective packets 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. For any edge $e = (v_i, v_j) \in p$, let $d_p(e)$ denote the length of path $(v_1, \ldots, v_i, v_j)$. The *distance* between two nodes, is the length of a shortest path that connects the two nodes.

A *direct routing problem* has the following components. Input: $(G, \Pi, \mathcal{P})$, where G is a multigraph, and the packets $\Pi = \{\pi_i\}_{i=1}^{N}$ have respective paths $\mathcal{P} = \{p_i\}_{i=1}^{N}$. Output: The injection times $\mathcal{T} = \{\tau_i\}_{i=1}^{N}$, denoted a *routing schedule* for the routing problem. Validity: 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 $t_i = \tau_i + |p_i|$.

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

Dependency Graphs. Consider a routing problem $(G, \Pi, \mathcal{P})$. The *dependency graph* $\mathcal{D}$ of the routing problem is a graph in which each packet $\pi_i \in \Pi$ is a node. We will use π_i to refer to the corresponding node in $\mathcal{D}$. There is an edge between two packets in $\mathcal{D}$ if their paths collide. An edge (π_i, π_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 π_i and π_j collide 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 τ_i, τ_j for π_i, π_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.



routing problem, $(G, \Pi, \mathcal{P})$ dependency graph, $\mathcal{D}$

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

We say that two packets are *synchronized*, if the packets are adjacent in $\mathcal{D}$ with some edge e and 0 is in the weight set of e. 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 conflict. 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 routing algorithm, $rt(G, \Pi, \mathcal{P}) \geq |\mathcal{K}| + |p_{max}|$, where p_{max} is the maximum path length of the packets in $\mathcal{K}$.*

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* π in $\mathcal{D}$, denoted $W(\pi)$, as the sum of the weight degrees of all edges incident with π. 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 Hardness of Direct Routing

Here, we will discuss how hard direct routing is both from the offline computation point of view, and from the routing time point of view. We show that in both senses, direct routing is hard, which will imply the necessity of buffering.

3.1 Decision Problems

Here we introduce the formal definition of the direct routing decision problem. In our reductions, we will use the wll known NP-complete problem Vertex Color, the vertex coloring problem [12], which asks whether a given graph G is κ-colorable. The chromatic number, $\chi(G)$ is the smallest κ for which G is κ-colorable. An algorithm *approximates* $\chi(G)$ *with approximation ratio* $q(G)$ if

on any input G, the algorithm outputs $u(G)$ such that $\chi(G) \leq u(G)$ and $u(G)/\chi(G) \leq q(G)$. Typically, $q(G)$ is expressed only as a function of N, the number of vertices in G. Combining the reduction in [11] with the inapproximability result for the independence number in [13], it is known that unless P=NP, there does not exist a polynomial time algorithm to approximate $\chi(G)$ 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 (a collision is counted whenever two packets conflict on an edge).

Problem 1 (Approximate Direct Route)

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

Question: *Does there exist a valid* k*–collision direct routing schedule* $\mathcal{T}$ *for some* $k \leq K$ *and 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 problem 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.

3.2 Computational Complexity of Direct Routing

We show that direct routing is NP complete using a polynomial-time reduction from coloring.

Theorem 3.1 (Direct Route **is NP Hard**) *There exists a polynomial time reduction from any instance* (G, κ) *of* Vertex Color *to an instance* $(G', \Pi, \mathcal{P}, T = \kappa - 1)$ *of* Direct Route.

Proof Sketch. We will use the direct routing problem illustrated to the right, for which the dependency graph is a synchronized clique. Each path is associated to a level, which denotes the x-coordinate at which the path moves vertically up after making its final left turn. There is a path for every level in $[0, L]$, and 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 $(i, L+i)$. Every packet is synchronized with every other packet, and meets every other packet exactly once.

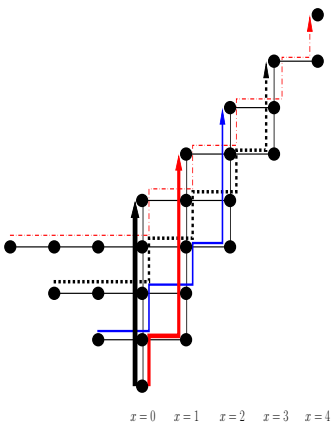


Figure 2: A mesh routing problem.

Given an instance $I = (G, K)$ of Vertex Color, we reduce it to an instance $I' = (G', \Pi, \mathcal{P}, T = K - 1)$ of Direct Route. Each node in G corresponds to a packet in Π. The paths are initially as illustrated in the routing problem above with $L = N - 1$. The transformed problem will have dependency graph $\mathcal{D}$ that is isomorphic to G, thus a coloring of G will imply a schedule for $\mathcal{D}$ and vice versa.

If (u, v) is not an edge in G, then we remove that edge in the dependency graph by altering the path of u and v without affecting their relationship to any other paths. After this construction, the resulting dependency graph is isomorphic to G. The construction of I' is polynomial in N. ■

DIRECT ROUTE is in NP, since, given a direct routing schedule, by traversing every pair of packets and checking for collision we can determine if it is valid and if the minimum injection time is $\leq T$, and so DIRECT ROUTE is NP complete. Further, we see that the reduction is gap preserving with gap preserving parameter $\rho = 1$ [14].

Theorem 3.2 (Inapproximability of Collision Injection Number) *A polynomial time algorithm that approximates $\tau_K(G, \Pi, \mathcal{P})$ with ratio r for an arbitrary direct 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 instance of direct routing in the proof of Theorem 3.1. Let K be the number of collisions allowed and suppose we have a schedule that approximates $\tau_K(G', \Pi, \mathcal{P})$ with ratio r. Let T the maximum injection time. For collision i, pick one of the packets involved and assign it injection time $T + i$. Now none of the packets collide, as they are all synchronized, so we have a valid direct routing schedule with maximum injection time $T + K$. So, $\tau_0(G', \Pi, \mathcal{P}) \leq T + K$, and since $\chi(G) = \tau_0(G', \Pi, \mathcal{P}) + 1$, we have that $\chi(G) \leq T + K + 1$. if $T/\tau_K(G', \Pi, \mathcal{P}) \leq r$ then $T/\tau_0(G', \Pi, \mathcal{P}) \leq r$ and so $(T + K + 1)/\chi(G) \leq r + K + 1$. The approximation is polynomial time since the reduction is polynomial time. ■

Letting $K = O(N^{1/2-\epsilon})$, since χ is hard to approximate with ratio $O(N^{1/2-\epsilon})$, we have

Corollary 3.3 (Inapproximability of Scheduling) *Unless* P=NP*, for $K = O(N^{1/2-\epsilon})$, there is no polynomial time algorithm to determine a valid K-collision direct routing schedule that approximates $\tau_K(G, \Pi, \mathcal{P})$ with ratio $O(N^{1/2-\epsilon})$ for any $\epsilon > 0$.*

Suppose a buffered routing algorithm for an arbitrary routing problem with given paths computes a routing schedule with routing time optimal up to a factor of $O(N^{1/2-\epsilon})$, i.e., $rt = O((C+D)N^{1/2-\epsilon})$. Suppose that the total number of times packets are buffered is B. Such a buffered schedule is equivalent to a direct routing schedule in which B collisions are allowed, and hence is a valid B-collision direct routing schedule for the packets. This schedule will therefore be an approximation to τ_B with ratio $O(N^{1/2-\epsilon})$. Using Theorem 3.3, we therefore have:

Theorem 3.4 ($\Omega(N^{1/2-\epsilon})$ Buffering Requirement for Tractable Buffered Routing) *A polynomial time algorithm with routing time at most a factor $O(N^{1/2-\epsilon})$ from optimal buffers packets $\Omega(N^{1/2-\delta})$ times in the worst case, for any $\delta > 0$.*

This means that in order to be able to *compute* a buffered schedule efficiently, we have a lower bound on the necessary buffering. This is for *any* algorithm. The specific form of the bounds comes from the specific bounds on inapproximability of coloring.

3.3 Routing Time of Direct Routing

We construct a "hard" routing problems 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 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 3.5 (Hard Routing Problem) *For every direct routing algorithm, there exist routing problems for which the routing time is* $\Omega(C \cdot D) = \Omega((C + D)^2)$

Proof: We construct a routing problem with $N = \Theta(C \cdot D)$ and for which the dependency graph $\mathcal{D}$ is a synchronized clique. The paths are as in Figure 2, and the description of the routing problem is in the proof of Theorem 3.1. The only difference is that c packets use each path. The conjestion is $C = 2c$ and the dilation is $D = 3L$. Since every pair of packets is synchronized, Lemma 2.1 implies that $rt(G, \Pi, \mathcal{P}) \geq N + D$. 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 $C + D = \Theta(\sqrt{C \cdot D})$. ■

Many Packets Are Buffered at Least Once. Let problem A denote the routing problem in in the proof of Theorem 3.5. Suppose that we have the option to buffer some of the packets. 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 3.6 *In routing problem A, if* $T \leq \alpha$*, then at least* $N - \alpha$ *packets are buffered at least once.*

Proof: If β packets are not buffered at all, then they form a synchronized clique, hence $T \geq \beta$. Therefore $\alpha \geq \beta$, and since $N - \beta$ packets are buffered at least once, the proof is complete. ■

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 3.6, the number of packets buffered is $\Omega(N)$:

Corollary 3.7 *There exists a routing problem for which any algorithm will buffer* $\Omega(N)$ *packets at least once in order to achieve asymptotically optimal routing time.*

Many Packets Are Buffered Many Times. We construct routing problem B which forces packets to be buffered multiple times. In routing problem A normalize the packets so that all packet lengths are $3L$; in order to do so add edges at the end of the paths when necessary. We then construct a routing problem B by concatenating k identical copies of routing problem A as described next. Let A_i denote the ith copy of A, $1 \leq i \leq k$. In problem B a packet traverses each routing problem A one after the other in sequence. In problem B, the path of a packet π is the concatenation of the paths in each copy of 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, $C = 2c$, $D = 3kL$ 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, for which we give the proof in an appendix.

Lemma 3.8 *For routing problem B, if* $T \leq \alpha$*, then at least* $N - j\alpha$ *packets each need to be buffered, at least* j *times, in* $A_1, \ldots, A_j$*, where* $1 \leq j \leq k$*.*

From Lemma 3.8, at least $N - k\alpha$ packets are each buffered at least k times in routing problem B. Suppose we wish to obtain an $O(N^\epsilon)$ approximation to an optimal schedule, for $0 \leq \epsilon < \frac{1}{2}$. In this case, $\alpha \leq \lambda(C + D)N^\epsilon$ for some $\lambda > 0$. Now, we choose $c = N^{(2-\epsilon)/3}$, $L = N^{(1+\epsilon)/3} - 1$ and $k = AN^{(1-2\epsilon)/3}$ with A chosen so that $\lambda A(2 + 3A) < 1$. We then have that $N - k\alpha \geq (1 - \lambda A(2 + 3A))N + o(N)$, i.e., $N - k\alpha = \Omega(N)$. By Lemma 3.8, $\Omega(N)$ packets are buffered at least k times. We therefore have the following result, which applies to *any* routing algorithm:

Theorem 3.9 (Buffering-Routing Time Tradeoff) *There exists a routing problem which requires* $\Omega(N^{(4-2\epsilon)/3})$ *total buffering to obtain a routing time that is a factor* $O(N^\epsilon)$ *from optimal.*

4 Algorithms For Direct Routing

Here we consider algorithms for direct routing. The algorithms we consider are variations of the following greedy algorithm, which we apply to the Tree, the Mesh, the butterfly and the Hypercube.

1: // **Greedy direct routing algorithm:**
2: // **Input:** 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 an arbitrary ordering of the packets.
5: **for** $i = 1$ *to* N **do**
6: Greedily assign the first available injection time τ_i to packet $\pi_i \in \Pi$ so that it does not conflict with any packet already assigned an injection time.
7: **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 $rt_{Gr}(G, \Pi, \mathcal{P})$. 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 π_i, which implies the following lemma:

Lemma 4.1 $rt_{Gr}(G, \Pi, \mathcal{P}) \leq \max_i\{W(\pi_i) + |p_i|\}$.

We now give an upper bound on the routing time of the greedy algorithm. Since the congestion is C and $|p_i| \leq D \ \forall i$, a packet collides with other packets at most $(C-1) \cdot D$ times. Thus, $W(\pi_i) \leq (C-1) \cdot D$, $\forall i$. Therefore, using Lemma 4.1 we obtain the following theorem.

Theorem 4.2 (Greedy Routing Time Bound) $rt_{Gr}(G, \Pi, \mathcal{P}) \leq C \cdot D$.

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 3.5, 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 by using a more careful choise of the order in which packets are considered.

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, the offline computation time of the greedy algorithm is $ol_{Gr}(G, \Pi, \mathcal{P}) = O(N \cdot C \cdot D)$, which is polynomial. This bound is tight since, in the worst case, each packet may have $C \cdot D$ collisions with other packets.

4.1 Trees

Consider the routing problem $(T, \Pi, \mathcal{P})$, in which T is a tree with n nodes, and all the paths in $\mathcal{P}$ are shortest paths. 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 4.3 *Let $(T, \Pi, \mathcal{P})$ 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: We show that every injection time satisfies $\tau_i \leq 2C - 2$. When packet π_i with distance d_i is considered, let v_i be the closest node to r on its path. All packets that are already assigned times that could possibly conflict with π_i are those that use the two edges in π_i's path incident with v_i, hence there are at most $2C - 2$ such packets. Since π_i is assigned the smallest available injection time, it must therefore be assigned a time in $[0, 2C - 2]$. ■

4.2 Multi-Dimensional 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 define $m = \sum_{i=1}^{d} m_i$. Every node is connected to up to $2d$ of its neighbors on the grid. Corollary 4.2 implies that the greedy routing algorithm achieves asymptotically optimal worst case routing time in the mesh. 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 on given arbitrary sources and destinations.

4.2.1 An Algorithm for Multi-bend paths

Here, we give a variation of the greedy direct routing algorithm which we analyze in terms of times the packet bend in the network. Consider a routing problem (G, Π, P). We first give an upper bound on the weight degree $W(\mathcal{D})$ of dependency graph $\mathcal{D}$ in terms of bends of 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 Π'. (Note that $\mathcal{D} = \mathcal{D}_{\Pi}$.) Consider the path p of a packet π. 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 π *bends* at node v, and that v is an *internal bending node*. We define the source and destination nodes of a packet π to be *external bending nodes*. The *segment* $p' = (v_i, \ldots, v_j)$ of a path p, is a subpath of p in which only v_i and v_j are bending nodes. Consider two packets π_1 and π_2 whose respective paths p_1 and p_2 collide 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 π_1 and π_2, if they meet multiple times on their paths.

Since p_1 and p_2 collide 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 collision. Weight w suffices to represent the collision of the two packets on the entire subpath p'. Therefore, a common subpath contributes at most one to the weight-number of $\mathcal{D}$. Let $A_{\mathcal{P}}$ denote the number of common subpaths. We have that $W(\mathcal{D}) \leq A_{\mathcal{P}}$. Therefore, 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.

For each common subpath, one of the packets bends at the begining and end nodes of the subpath. Thus, a packet contributes to the number of total subpaths only when it bends. Consider a packet π which bends at a node v. Let e_1 and e_2 be the two edges of the path of π 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 contribues 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 Π', where the maximum number of internal bends is b, $A_{\mathcal{P}} \leq 2(b+1)|\Pi'|(C-1)$. Therefore,

Lemma 4.4 *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 Π'.*

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. K-amortized graphs are similar to balanced graphs [5]. Thus $\mathcal{D}$ is $4(b+1)C$-amortized. A *generalized coloring* of a graph with weights on each edge is a coloring in which the difference between the colors of adjacend nodes cannot equal a weight. K-Amortized graphs admit generalized colorings with $K+1$ colors. This is the content of the next lemma.

Lemma 4.5 (Efficient Coloring of Amortized Graphs) *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 order 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 $\leq K$, there is a node v with weight degree $\leq K$. 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 sycnhronized. Lemma 4.5 implies that the dependency graph $\mathcal{D}$ has a valid $4(b+1)(C-1)+1$ generalized coloring. Lemma 4.5 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 4.6 (Multi-bend Direct Routing Time) *Let $(M,\Pi,\mathcal{P})$ be a direct 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 routing schedule with routing time $rt \leq 4(b+1)(C-1)+D$.*

Permutation Routing on the 2-Dimensional Mesh. Consider a $\sqrt{n}\times\sqrt{n}$ Mesh. In a permutation routing problem every node is the source and destination of exaclty one packet. We solve permutation routing problems by using paths with one internal bend. Let e be a column edge in the up direction. Since at most $\sqrt{n}$ packet 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 columns. Applying Theorem 4.6, and the fact that $D=O(\sqrt{n})$, we then get that $rt=O(\sqrt{n})$, which is worst case optimal for permutation routing on the mesh.

Near Optimal Direct Routing on the Mesh. Maggs *et al.* [20, Section 3] give a strategy to select paths in the mesh M for a routing problem with arbitrary sources and destinations. The congestion achieved by the paths is within a logarithmic factor from the optimal, i.e., $C=O(dC^*\log n)$ w.h.p., where C^* is the minimum congestion possible for the given sources and dsetinations. Following the construction in [20], it can be shown that the packet paths are constructed from $O(\log n)$

shortest paths between random nodes in the Mesh. Hence, the number of bends b that a packet makes is $b = O(d \log n)$, and $D = O(m \log n)$, where m is the sum of the side lengths. We can thus use Theorem 4.6 to obtain a direct routing schedule with the following properties:

Theorem 4.7 *For any routing problem (M, Π) with given sources and destinations, there exists a direct routing schedule with routing time $rt = O(d^2 C^* \log^2 n + m \log n)$, w.h.p..*

Let D^* denote the maximum length of the shortest paths between sources and destinations for the packets in Π. D^* is the minimum possible dilation. Let rt^* denote the optimal routing time (direct or not). For any set of paths, $C + D = \Omega(C^* + D^*)$, and so the optimal routing time rt^* is also $\Omega(C^* + D^*)$. If $D^* = \Omega(m/(d^2 \log n))$, then $rt^* = \Omega(C^* + m/(d^2 \log n))$, and Theorem 4.7 implies the following result:

Corollary 4.8 *If $D^* = \Omega(m/(d^2 \log n))$, then there exists a direct routing schedule with $rt = O(rt^* d^2 \log^2 n)$, w.h.p..*

4.3 Butterfly

We consider the n-input butterfly network B, where $n = 2^k$, [18]. There is a unique path from an input node to an output node of length $\lg n + 1$. Assume that every input node is the source of one packet and the destinations are randomly chosen.

For packet π_i, we consider the bernouilli random variables x_j which are one if packet π_j collides with π_i. Then the degree of π_i in the dependency graph, $X_i = \sum_j x_j$. We show that $E[X_i] = \frac{1}{4}(\lg n - 1)$, and since the x_j are independent, we use the Chernoff bound to get a concentration result on X_i. Thus, we show that w.h.p, $\max_i X_i = O(\lg n)$. Since the injection time assigned by the greedy algorithm to any packet is at most its degree in the dependency graph, we show that the routing time of the greedy algorithm for a random destination problem on the Butterfly satisfies $\mathbf{P}\left[rt_{Gr}(B, \Pi, \mathcal{P}) \leq \frac{5}{2} \lg n\right] > 1 - 2\sqrt{2} n^{-\frac{1}{2}}$. (the details are given in an appendix).

Valiant [29, 30] proposed permutation routing on butterfly-like networks by connecting two butterflies, with the outputs of one as the inputs to the other. The permutation is routed by first sending the packets to random destionations on the outputs of the first butterfly. This approach avoids hot-spots and converts the permutation problem to two random destinations problems. Thus, we can apply the result for random destinations twice to obtain the following theorem:

Theorem 4.9 *For permutation routing on the double-butterfly with random intermediate destinations, the routing time of the greedy algorithm satisfies $\mathbf{P}\left[rt_{Gr} \leq 5 \lg n\right] > 1 - 4\sqrt{2} n^{-\frac{1}{2}}$.*

4.4 Hypercube

We consider the n-hypercube network H with $n = 2^k$ nodes, [18]. The distance between any two nodes is $\leq \lg n$. Assume that each node is the source of one packet, and that the destinations are random. We use (left-to-right) *bit-fixing* to determine the paths given the sources and destinations (see appendix). The paths are shortest paths, and $D \leq \lg n$,

As with the Butterfly analysis, let X_i be the number of other different packets that packet i collides with. In [23, Theorem 4.6], it is shown that $\mathbf{P}\left[\max_i X_i \leq 6 \lg n\right] > 1 - 1/(32n)$. Thus, the maximum degree in the dependency graph $\mathcal{D}$ is at most $6 \lg n$ w.h.p. which implies that the routing time of the greedy algorithm is at most $7 \lg n$ w.h.p., $\mathbf{P}\left[rt_{Gr} \leq 7 \lg n\right] > 1 - 1/(32n)$. Once again, using random intermediate destinations, we can route any permutation on the hypercube by combining two random destinations problems:

Theorem 4.10 *For a permutation routing using random intermediate destinations and bit-fixing paths, the routing time of the greedy algorithm satisfies $\mathbf{P}\left[rt_{Gr} < 14 \lg n\right] > 1 - 1/(16n)$.*

5 Discussion

We have given a comprehensive analysis of direct routing. We presented hardness results of 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 conflict. We also show the existence of a hard routing problem in which *any* routing algorithm requires $\Omega(N^{(4-2\epsilon)/3})$ total packet buffering to approximate an optimal routing schedule with approximation ratio $O(N^{\epsilon})$, for any $0 \leq \epsilon < 1/2$. Thus, packet buffering is essential in order to obtain efficient routing schedules in a general setting.

We then examined direct routing algorithms. In particular, we gave a general greedy routing algorithm which is worst case optimal. We studied the efficiency of the algorithms in terms of the congestion C and the dilation D. Using variations of the greedy algorithm we solve optimaly (near optimaly) direct routing problems on the Tree, Multi-dimensional Mesh, Butterfly, and Hypercube. One important implication of our results is that on particular network topologies and for particular routing problems, direct routing achieves routing times close to the routing times obtained by store-and-forward algorithms.

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] 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.

[4] 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.

[5] Béla Bollobás. *Random Graphs*. Cambridge University Press, New York, second edition, 2001.

[6] 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.

[7] C. Busch. $\tilde{O}$(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.

[8] 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.

[9] 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.

[10] 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.

[11] M. Furer. Improved hardness results for approximating the chromatic number. In *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 414–421, 1995.

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

[13] J. Hastad. Clique is hard to approximate within $n^{1-\epsilon}$. In *Proceedings of the 37th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 627–636, 1996.

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

[15] 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.

[16] 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.

[17] 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.

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

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

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

[21] 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.

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

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

[24] 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.

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

[26] 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.

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

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

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

[30] 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.

[31] 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.

A Hardness of Direct routing

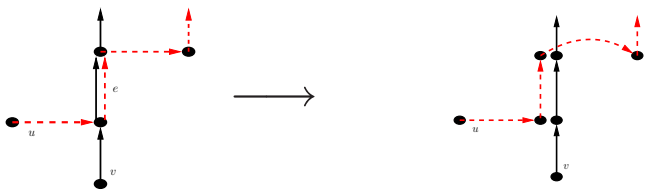
A.1 Computational Complexity of Direct Routing

Proof of Theorem 3.1. We will explicitly demonstrate the reduction from coloring. Illustrated on Figure 2 is a 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 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. 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 $d_{p_1}(e) = d_{p_2}(e)$: 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 $L + 2i$.

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 $I = (G, K)$ of Vertex Color, we now show how to reduce it to the corresponding instance $I' = (G', \Pi, \mathcal{P}, T = K - 1)$ of Direct Route. Each node in G corresponds to a packet in Π. 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. We thus alter the two paths corresponding to u, v at their intersection edge e as follows,



Notice that the paths corresponding to u, v no longer collide. Further, the lengths of u and v and their relationships with any other paths have not been altered in any way. The resulting dependency graph is isomorphic to G.

Since every two packets that collide 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 collide (and hence are adjacent in $\mathcal{D}$) will have the same injection time. Thus the answer to instance

I of VERTEX COLORis true if and only if the answer to I' DIRECT ROUTEis true. The proof is concluded by noting that the construction of I' is clearly polynomial in N. ■

A.2 Routing Time of Direct Routing

Proof of Lemma 3.8: We prove the claim by induction on j. For $j = 1$ the claim follows immediately from Lemma 3.6. 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 $X \leq (r-1)\alpha$. Let Y be the packets of x_{r-1} which will not be buffered in A_r. Then, the Y packets are all synchronized in A_r, and so must be injected at different times (Lemma 2.1). Thus, $T \geq Y$, and so $\alpha \geq Y$. Thereforem 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 $A_1, \ldots, A_r$. ■

B Algorithms for Direct Routing

Proof of Lemma 4.1: 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 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 $W(\pi_i)$ is the number of times packets can conflict with π_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 π_i that is in this set, as it assigns the smallest available injection time. ■

B.1 Buttrefly

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 4 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 node weight degree in the dependency graph $\mathcal{D}$ is at most $\frac{3}{2} \lg n$. Since the paths are shortcut-free and $D \leq \lg n$, using Lemma 4.1, 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 B.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 $\sigma \neq \pi$ 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 $w \neq v_{i-1}$. 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 B.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 $\mathcal{D}$, X_i is equal to $W(\pi_i)$. We will use the following version of the Chernoff bound to get a concentration result for X_i:

Lemma B.2 ([23]) *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 B.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 $\max_i W(\pi_i) = \max_i X_i$, and $D = \lg n$, Lemma 4.1 then gives the following theorem:

Theorem B.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 [23, Section 4.2]). In order to avoid hot-spots, Valiant [29, 30] 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 B.3 twice to obtain Theorem 4.9.

B.2 Hypercube

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 $s(\pi)$ to destination $\delta(\pi)$; flip the leftmost bit at which the labels of $s(\pi)$ 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, $D \leq \lg n$, 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 4, 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, 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 [23, Theorem 4.6]:

Lemma B.4 ([23]) $\mathbf{P}[\max_i X_i \leq 6 \lg n] > 1 - 1/(32n)$.

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 4.1 implies that the routing time of the greedy algorithm is at most $7 \lg n$ w.h.p. We have the following theorem:

Theorem B.5 *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* $\mathbf{P}[rt_{Gr}(B, \Pi, P) \leq 7 \lg n] > 1 - 1/(32n)$.

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 [29, 30]: 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 B.5 twice to obtain Theorem 4.10.