Efficient Bufferless Routing on Leveled Networks

Costas Busch[*] Shailesh Kelkar[†] Malik Magdon-Ismail[‡]

August 23, 2004

Abstract

We study bufferless packet routing in which packets, once injected, cannot be buffered at nodes. Given a set of preselected packet paths, a well known lower bound on the routing time is $\Omega(C + D)$, where D is the *dilation* (maximum path length) and C the *congestion* (maximum number of times an edge is used). We show that for leveled networks, one can obtain bufferless routing that is at most one or two logarithmic factors from the lower bound:

i. We give a centralized bufferless algorithm with routing time $O(C \cdot \log(DN) + D)$, where N is the number of packets.

ii. We convert the centralized algorithm to a distributed bufferless algorithm with routing time $O(C \cdot \log^2(DN) + D \cdot \log(DN))$. The heart of our approach is a new technique, *reverse-simulation*, which constructs an efficient (at most logarithmic extra cost) distributed emulation of the centralized algorithm.

Our algorithms improve the best previously known result specialized for leveled networks by multiple logarithmic factors.

Keywords: Bufferless Routing; Hot-potato Routing; Congestion; Dilation; Leveled Networks.

1 Introduction

We consider bufferless routing problems, in which packets cannot be stored at nodes while in transit to their destination. In particular we consider *hot-potato*-style (or *deflection*) routing [3], in which if a packet cannot make progress toward its destination, it must be "deflected" away from its destination like a "hot potato". Hot-potato routing algorithms have been observed to work well in practice [4, 22, 29, 16, 28, 22], and are well-suited for optical networks where it is difficult or costly to buffer optical messages [1, 15, 22, 30, 31].

We study routing problems on leveled networks. A leveled network with *depth* L consists of $L+1$ *levels* of nodes, numbered 0 to L. Every node belongs to exactly one level, and the only edges are between nodes at consecutive levels (Figure 1). Leveled networks are interesting because many routing problems on multiprocessor network architectures can be represented as routing problems on leveled networks. Typical examples are routing problems on the N input *Butterfly* network and the *Mesh* network (Figure 1). Further, several routing problems on other multiprocessor

[*]Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180. Email: buschc@cs.rpi.edu. Fax: +1-518-276-4033. Contact author.

[†]Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180. Email: kelkas@cs.rpi.edu.

[‡]Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180. Email: magdon@cs.rpi.edu.

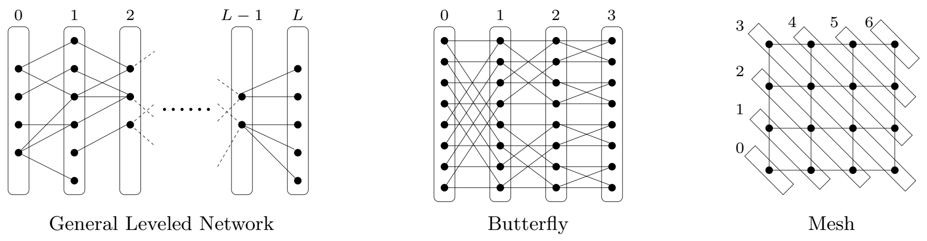

Figure 1: Leveled networks

architectures such as shuffle-exchange networks, multidimensional arrays, the hypercube, fat-trees, de Bruijn networks, and the multi-butterfly can also be viewed as routing problems on leveled networks (see [10, 19] for more details).

We assume a *synchronous* routing model in which time is discrete, and at each time step, a node receives packets, makes a routing decision, and then forwards the packets to adjacent nodes. At each time step, a node is allowed to send at most one packet per link. Note that at any time step at most two packets can traverse a link, one packet in each direction of the link.* We study *many-to-one batch routing problems* on leveled networks: we are given N packets where each node is the source of at most one packet, but may be the destination of many packets. Each packet has a *preselected path* from its source to its destination. The paths are selected before routing. Every preselected path is monotonic in the sense that every edge in a path connects a lower level node with a node in the next higher level, i.e., a path moves from left to right on the general leveled network depicted in Figure 1. Here we are only concerned with scheduling the packets given the paths, and not how to obtain the paths.

A routing algorithm specifies the schedule with which the packets move in the network. The *routing time* of an algorithm is the time needed until the last packet is delivered to its destination. Given the preselected paths, the routing time depends on the *congestion* C, the maximum number of packets that traverse any edge, and the *dilation* D, the maximum length of any path. Since at most one packet can traverse any edge at a time step, a trivial lower bound for the routing time of any routing algorithm is $\Omega(C + D)$. It is desirable to design routing algorithms with routing time close to this lower bound.

In hot-potato (bufferless) routing, an interesting situation occurs when two or more packets appear in the same node at the same time and all of them wish to follow the same link; note that there could be up to C such packets. This situation represents a *conflict* among the packets, because only one of them can follow the link successfully. Since there are no buffers at the nodes, the remaining packets must be sent on alternative links. We say that such packets are *deflected*. A consequence of deflections is that a packet may not always be able to remain on its preselected path, since when a packet is deflected is may not be possible to send it back on the edge that it traversed in the previous time step. We only consider bufferless algorithms in which the final path followed by the packets contains every edge in its preselected path. For such algorithms, the $\Omega(C + D)$ bound on the routing time is still valid. Thus, a routing time close to $C + D$ is optimal with respect to any routing algorithm, buffered or not, even when we allow path deformation, provided that the final paths must contain the preselected paths.

*Our leveled network model follows that in [20], the only difference being that we assume undirected edges, as opposed to directed edges in [20].

Contributions. We present two new bufferless routing algorithms for many-to-one routing problems in leveled networks. The first algorithm is centralized, and has routing time $O(C \log(DN) + D)$, which is optimal when $C = O(D/\log(DN))$, and is at most a logarithmic factor from optimal in general. The algorithm is centralized in the sense that some node has complete information about the parameters of the routing problem and decides about the scheduling of all packets. The second algorithm is distributed, that is, all routing decisions are made locally at the nodes, and has routing time $O(C \log^2(DN) + D \log(DN))$ which is a logarithmic factor worse than the centralized algorithm. Both results hold with high probability (w.h.p.), i.e., with probability at least $1 - O(1/DN)$. The distributed algorithm relies on a new technique, *reverse-simulation*, which provides an efficient distributed emulation of the centralized algorithm.

The best known previous result for leveled networks achieved a routing time that is a $\log^9$-factor away from optimal [10], and so our new results represent a significant improvement. An important property of our algorithms is that the final paths used by the packets contain the original preselected paths. Further, for the centralized algorithm, a packet never strays away from its original preselected path. Thus, the performance of our algorithms is controlled by the quality of the preselected paths (C and D).

A high level description of our centralized algorithm is as follows. We first divide the network into *groups* of levels, so that each group consists of $2D$ levels. The effect of this division is that each packet path belongs to exactly one group. Packets on each group are routed independently. We now focus on one such group. We partition the group into areas of the network called *frames*. Each frame consists of roughly $\log(DN)$ levels. The purpose of the frames is that packets are sent to the destinations by following their paths from one frame to the next. In order to achieve this, we partition the packets in the group randomly and uniformly into roughly C disjoint packet sets; each packet set creates congestion at most $\log(DN)$ w.h.p.. We route packets of the same set in the same frame, and packets of different sets in different frames.

The packets of any particular set follow a schedule computed from a packet dependency graph. In the dependency graph, two packets share an edge if their paths in the frame conflict. We color the packets in the dependency graph and send them to the next frame according to their color, so that packets of different colors are sent in different rounds. Since packets of the same color don't conflict, we show that it takes $O(\log(DN))$ time to move all packets from one frame to the next. In order to get the time to deliver a packet from its source to its destination, we multiply this by the number of frames which is $O(D/\log(DN))$; thus, once the packet is injected it takes time $O(D)$ for the packet to be delivered. We show that the latest possible injection time for a packet is $O(C \log(DN))$ which leads to our routing time bound for the centralized algorithm.

To obtain the distributed algorithm, we color the dependency graph in a distributed way. To accomplish this, we use a randomized distributed coloring algorithm: packets randomly pick a color, and the packets attempt to route according to this coloring (i.e., they attempt to move to the next). If the result is successful, then the packets can move on; however, if the result is not successful (some packets conflict), then the packets trace their paths backwards (*reverse-simulation*) and the process repeats. We show that the added inefficiency of the distributed coloring is at most one extra logarithmic factor.

Related Work. Bufferless routing algorithms have been studied for specific network multiprocessor architectures such as the 2-dimensional mesh and torus [5, 9, 11, 14, 17], the d-dimensional mesh [5, 7], the hypercube [8, 14], trees [2, 13]. and vertex symmetric networks [23]. Multiprocessor architectures are extensively covered in [19]. Bhatt *et al.* [6] study hot-potato routing on leveled networks, but for different routing problems than the ones we consider here. The most related work to ours is [10], which is the result we improve by seven logarithmic factors; moreover, that result was expressed in terms of L instead of D which we consider here. A recent result in [12] shows that it is possible to obtain a general bufferless routing algorithm for arbitrary networks with routing time $O((C + D) \log^3(n + N))$, where n is the size of the network. However, that result doesn't take advantage of the special structure of leveled networks, which allows us to obtain smaller routing time. Further, our centralized algorithm is one of the few hot-potato routing algorithms that guarantee that the packets remain on their original preselected paths.

Leveled networks have also been studied in the context of store-and-forward routing (with buffers), by Leighton *et al.* [20], where they present an $O(C + L + \log N)$ randomized algorithm with constant size buffers. For store-and-forward routing, there has been a lot of research for obtaining optimal $O(C + D)$ algorithms for arbitrary networks [18, 21, 24, 26, 27].

Paper Outline. In Section 2 we give some necessary preliminaries. In Section 3 we present our centralized algorithm and in Section 4 the distributed algorithm.

2 Preliminaries

Here, we give some necessary preliminaries that will be used later in our algorithms. We will need a Chernoff-type tail inequality.

Lemma 2.1 (Chernoff bound, [25, Exercise 4.1]) *Let $\{X_i\}_{i=1}^n$ be independent Bernoulli random variables, with $Pr[X_i = 1] = p_i$. Let $X = \sum_{i=1}^n X_i$, and set $\mu = E[X] = \sum_{i=1}^n p_i$. For any $\delta > 2e$, $Pr[X > \delta\mu] < 2^{-\delta\mu}$.*

We now introduce packet paths, oscillations, frames, and the dependencies between packets.

Paths. Consider a packet π with preselected path p which has path length $|p|$. Its *current path* at time step t, denoted $p(t)$, is defined as follows. At time 0, the current path is the preselected path, $p(0) = p$. Suppose that at time t, packet π is in node v_i, with current path $p(t) = (v_i, v_{i+1}, \ldots, v_k)$. If at time t, packet π successfully follows the first edge (v_i, v_{i+1}) in $p(t)$ (the packet moves forward), then, at time $t + 1$, packet π appears in node v_{i+1} with current path $p(t + 1) = (v_{i+1}, \ldots, v_k)$. On the other hand, if at time $t + 1$ packet π is deflected toward a node v_j, then at time $t + 1$ it appears in node v_j with current path $p(t + 1) = (v_j, v_i, v_{i+1}, \ldots, v_k)$. If the packet moves forward, $|p(t + 1)| = |p(t)| - 1$ and if it is deflected, then $|p(t + 1)| = |p(t)| + 1$.

Oscillations. Suppose packet π has current path $(v_i, v_{i+1}, \ldots, v_k)$. π *oscillates* on edge $e = (v_i, v_{i+1})$ if it moves back and forth on e: if at time t, π appears in v_i, then at time $t+1$, π appears in v_{i+1}, and at time $t + 2$ it is back in v_i, and so on. When a packet oscillates, the length of its current path increases and decreases by one each time. Oscillations are useful because they provide a way to "buffer" packets on edges instead of at nodes.

Frames. We partition the levels of the network into γ non overlapping *frames* $F_1, F_2, \ldots, F_\gamma$, each containing λ levels (except for the last frame, which may contain fewer). Frame F_i, $1 \le i < \gamma$, consists of the λ levels $(i-1)\lambda, \ldots, i\lambda-1$. Frame F_γ consists of the levels $(\gamma-1)\lambda, \ldots, L$. Note that $\gamma = \lceil (L+1)/\lambda \rceil$. We will pick $\lambda = 4\alpha \log(DN)$, where α is a parameter to be defined later; thus, the frames have logarithmic size. (We assume that $\log(DN)$ is an integer, if not we use $\lceil \log(DN) \rceil$.)

We refer to the levels that comprise frame F_i as the *inner-levels* of F_i, and we number them from 1 to λ. Thus, inner-level k of frame F_i corresponds to real level $(i-1)\lambda + (k-1)$, where $1 \leq k \leq \lambda$. The *odd inner-levels* are numbered $1, 3, \ldots, \lambda - 1$ (recall that λ is even). The inner level of an edge is the smaller of the inner-levels of the nodes it is incident with. Thus, corresponding to odd inner-levels are *odd inner-edges*, and similarly even inner-levels and even inner-edges.

Packet Sets and Dependency Graphs. We partition the set of packets Π into $s = 8\alpha e C$ sets, $\Pi_1, \Pi_2, \ldots, \Pi_s$. Each packet is placed into one of these sets uniformly at random. Thus, $\Pi = \bigcup_{i=1}^{s} \Pi_i$, and $\Pi_i \cap \Pi_j = \emptyset$ for $i \neq j$, so $|\Pi| = \sum_{i=1}^{s} |\Pi_i| = N$.

Consider the packets in Π_i, and two consecutive frames F_j and F_{j+1}. For each packet $\pi \in \Pi_i$ denote by q_π the sub-path of the preselected path that consists only of edges in F_j and F_{j+1}. We define the packet dependency graph $G_{(i,j)} = (V_{(i,j)}, E_{(i,j)})$ as follows. The nodes of $V_{(i,j)}$ correspond to the packets in Π_i, so $|V_{(i,j)}| = |\Pi_i|$. Let $\pi, \sigma \in \Pi_i$, then $(\pi, \sigma) \in E_{(i,j)}$ if and only if the paths q_π and q_σ share some edge in (F_j, F_{j+1}), i.e., if the paths collide.

The *degree* of a packet π in $G_{(i,j)}$, denoted $d_{(i,j)}(\pi)$, is the number of edges incident with π. The degree of $G_{(i,j)}$, denoted $d_{(i,j)}$, is the maximum degree of any packet in $V_{(i,j)}$. Let $d = \max_{\{i,j\}} d_{(i,j)}$, i.e., d is the maximum degree of any of the graphs $G_{(i,j)}$, for any i and j.

We show that d cannot be too big. In fact, a packet path collides with at most $2\lambda C$ other paths over two consecutive frames. Only approximately $2\lambda C/s = O(\lambda/\alpha)$ of these packets are in the same set, so we expect that $d = O(\lambda/\alpha)$. The next lemma formalizes this notion.

Lemma 2.2 *$d \leq \lambda/\alpha = 4\log(DN)$, with probability at least $1 - 1/DN$.*

Proof: Consider $d_{(i,j)}(\pi)$, for $\pi \in \Pi_i$. Note that $|q_\pi| \leq 2\lambda$. Let R denote the set of packets that collide with π on q_π, $|R| \leq |q_\pi| C \leq 2\lambda C$. Let $\sigma \in R$, then $Pr[\sigma \in \Pi_i] = \frac{1}{s}$. Therefore, $\mu = E[d_{(i,j)}(\pi)] = \sum_{\sigma \in R} Pr[\sigma \in \Pi_i] = \frac{|R|}{s} \leq \frac{\lambda}{4\alpha e}$. Since the events $\sigma \in \Pi_i$ are independent Bernoulli trials, we can apply Lemma 2.1 with $\delta = 2e + \frac{\lambda}{2\alpha\mu}$ to obtain

$$Pr[d_{(i,j)}(\pi) > \tfrac{\lambda}{\alpha}] < 2^{-(2e\mu + \frac{\lambda}{2\alpha})} \leq 2^{-\frac{\lambda}{2\alpha}}$$

d is the maximum degree over any node in any $G_{(i,j)}$. Since every packet has path length at most D, a packet appears as a node in at most D of the $G_{(i,j)}$'s. Thus, $\sum_{i,j} |V_{(i,j)}| \leq DN$. We now succesively apply the union bound to obtain the desired result:

$$Pr[d_{(i,j)} > \tfrac{\lambda}{\alpha}] = Pr[\max_{\pi \in V_{(i,j)}} d_{(i,j)}(\pi) > \tfrac{\lambda}{\alpha}] \leq |V_{(i,j)}| 2^{-\frac{\lambda}{2\alpha}};$$

$$Pr[d > \tfrac{\lambda}{\alpha}] = Pr[\max_{i \in [1,s], j \in [1,\gamma]} d_{(i,j)} > \tfrac{\lambda}{\alpha}] \leq \sum_{i,j} |V_{(i,j)}| 2^{-\frac{\lambda}{2\alpha}} \leq DN\ 2^{-\frac{\lambda}{2\alpha}}.$$

Since $\lambda = 4\alpha \log(DN)$, the lemma follows. ■

3 Centralized Algorithm

Here, we give the centralized algorithm. Let $\gamma' = 2\lceil D/\lambda \rceil$. A *group* is a collection of γ' consecutive frames (at most $2D + 2\lambda$ levels). Let S_1, S_2 be two *sets* of groups given by $S_1 = \{[1,\gamma'], [\gamma'+1, 2\gamma'], \ldots\}$, and $S_2 = \{[\gamma'/2+1, 3\gamma'/2], [3\gamma'/2+1, 5\gamma'/2], \ldots\}$, where $[i,j]$ denotes the set of frames $F_i, \ldots, F_j$. We treat the last set of frames in S_1, S_2 as a group even though they may contain fewer than γ' frames. The groups in S_1 do not share any levels, and similarily for S_2. Note that the groups in S_2 are shifted by $\gamma'/2$ frames with respect to the groups in S_1. A packet belongs to S_1 (resp. S_2) if its path lies entirely within some group in S_1 (resp. S_2). Since each group contains at least $2D$ levels, every packet belongs to either S_1 or S_2 (or both). If it belongs to both, then we assign it to S_1.

We route the packets in two *sessions*. In the first session, we route the packets belonging to S_1. The second session begins after the first session ends, and in the second session, we route the packets belonging to S_2. Having two sessions contributes at most a factor of 2 to the routing time. In a particular session, because the groups are (framewise) disjoint, the packets in one group can be routed simultaneously with all the packets in another group without any possibility of interfering. Thus it suffices to describe the algorithm to route the packets. This algorithm runs simultaneously in all the groups of the first session, and then again in all the groups of the second session.

Let $\Pi_i(g, \mathsf{sess})$ denote the packets in Π_i that belong to group g in session sess. For example, $\Pi_i([1,\gamma'], 1)$ are the packets in Π_i whose paths lie entirely in the frames $[1,\gamma']$. Let $\Phi_1 = \cup_{i,g}\Pi_i(g,1)$ denote the packets that will be routed in the first session and $\Phi_2 = \cup_{i,g}\Pi_i(g,2)$ those that will be routed in the second session. An overview of the algorithm is given below.

```
Algorithm: Centralized Algorithm
Input: Routing problem with packets $\Pi$ on a graph $G$;
Output: A bufferless packet routing schedule with routing time $O(C\log(DN) + D)$;
begin
    $\alpha = 3$; $\lambda = 4\alpha\log(DN)$; $\gamma = \lceil (L+1)/\lambda \rceil$; $\gamma' = 2\lceil D/\lambda \rceil$; $s = 8\alpha e C$;
    $m = 2s + \gamma' - 1$; $\chi = d + 1$; $\tau = 2(\chi + \lambda - 1)$;
1   Partition $G$ into frames $F_1, \ldots, F_\gamma$ of width $\lambda$; Construct the group sets $S_1$ and $S_2$;
2   Partition $\Pi$ uniformly at random into sets $\Pi_1, \ldots, \Pi_s$;
3   Divide time into two sesssions each consisting of $m \cdot \tau$ time steps. The packets in $\Phi_1$ are
    routed in session 1, and then the packets in $\Phi_2$ are routed in session 2.
4   for session $\mathsf{sess} = 1, 2$ do
5       for each $g$ simultaneously do
6           Route_Group($\Pi_1(g,\mathsf{sess}), \Pi_2(g,\mathsf{sess}), \ldots, \Pi_s(g,\mathsf{sess})$)
end
```

We now describe algorithm $\mathsf{Route\_Group}(\Pi_1(g,\mathsf{sess}), \Pi_2(g,\mathsf{sess}), \ldots, \Pi_s(g,\mathsf{sess}))$. The input is a set of packets of a particular group. We will describe the algorithm for group $g = [1,\gamma']$ and $\mathsf{sess} = 1$, hence the packets are to be routed in the frames $F_1, \ldots, F_{\gamma'}$. The algorithm for other groups is identical excepting for a change in the the indices. Since we focus on a particular group in the first session, we will simplify the notation by dropping the g and sess dependence. Hence Π_i will denote $\Pi_i([1,\gamma'], 1)$. The algorithm consists of m phases, $\phi_1, \phi_2, \ldots, \phi_m$, each of duration τ time steps. To each packet set Π_i, we associate a wave which will route the packets in Π_i, as we describe next. An outline of the algorithm is sketched below.

```
Algorithm: Route_Group
Input: Routing problem with packets $\Pi(g)$ in group $g$ consisting of frames $F_1, \dots, F_{\gamma'}$;
Output: A bufferless packet routing schedule for the packets in the group;
begin
    Obtain $G_{(i,j)}$, compute $d$, and greedily color each $G_{(i,j)}$ with at most $\chi$ colors;
    Divide time into phases $\phi_1, \dots, \phi_m$, each phase consisting of $\tau$ time steps;
    Define waves $\omega_1, \dots, \omega_s$, where wave $\omega_i$ enters the network at phase $\phi_{2i-1}$;
    for each packet set $\Pi_i$ do
        Packets of set $\Pi_i$ follow wave $\omega_i$ as follows;
        for each phase $\phi$ in which wave $\omega_i$ points to frame $F_j$ do
            // Packets in $F_j$ will move to $F_{j+1}$;
            Initially, only packets of $\Pi_i$ oscillate in $F_j$, and $F_{j+1}$ is empty;
            Phase $\phi$ consists of time steps $t_1, t_2, \dots, t_\tau$;
            Define boats $b_1, \dots, b_\chi$, where boat $b_k$ enters the network at time $t_{4k-3}$;
            Packets of color $k$ follow boat $b_k$ to target inner-level $\ell_k = \lambda - (2k-1)$ in $F_{j+1}$, where
            they will oscillate until the next phase;
end
```

3.1 Waves

A *wave* ω is a pointer to a frame. Initially the wave is NULL. The wave enters the network (points to frame F_1) at some phase ϕ_i. At each subsequent phase the wave points to the next higher frame, so in phase ϕ_{i+k}, it points to frame F_{k+1}. Eventually, ω points to the last frame $F_{\gamma'}$, after which it leaves the network and becomes NULL. There are s waves $\omega_1, \omega_2, \dots, \omega_s$ (as many waves as there are packet sets). Wave ω_i enters the network at phase ϕ_{2i-1}. Note that waves are spaced 2 frames apart, which will be useful for moving packets (see below). The last wave ω_s enters in phase ϕ_{2s-1} and after γ' phases, it has left the network, so the number of phases is $m = 2s + \gamma' - 1$. We use the wave to also denote to the frame it points to.

The purpose of wave ω_i is to route the packets in set Π_i along with it, as it moves from lower to higher levels. Packet $\pi \in \Pi_i$ is injected when wave ω_i contains π's source. The packet is absorbed either when the wave contains its destination or its destination is one frame ahead of the wave.

At the beginning of each phase, packets appear inside their respective waves, and frames between waves are empty of packets; this property is essential for moving packets along their waves. Consider a phase ϕ during which wave ω_i points to frame F_j. At the beginning of ϕ, F_j contains only packets from Π_i, and F_{j+1} is empty of packets. By the end of phase ϕ, the packets in F_j will move from frame F_j to frame F_{j+1}. Thus, at the beginning of the next phase, all these packets are still in the wave ω_i, and frame F_j is empty (which allows packets of Π_{i+1} to move along wave ω_{i+1}). We continue by describing in detail how the packets of Π_i move from F_j to F_{j+1} during phase ϕ.

3.2 Initial and Target Levels

Suppose that phase ϕ consists of time steps $t_1, t_2, \dots, t_\tau$. At the beginning of phase ϕ, the packets of Π_i that are already in wave ω_i are oscillating on odd inner-edges of F_j. Suppose $\pi \in \Pi_i$ is oscillating on odd inner-edge $e = (v_\ell, v_{\ell+1})$ of F_j, where the inner level of v_ℓ is ℓ (which is odd). The packet oscillates on e so that at odd time steps $t_1, t_3, \dots$, packet π appears in v_ℓ. We say that π *oscillates* at inner-level ℓ, which is the *initial* inner-level of π in phase ϕ.

Now suppose that the current path of π at its initial inner-level ℓ is a sub-path of its preselected path. During phase ϕ, packet π will follow its current path until it reaches a *target* inner-level ℓ' in F_{j+1}, where it will oscillate for the remainder of the phase. At its target level, π's current path will remain a sub-path of its preselected path. The target level will become the new initial level at the next phase, when the wave ω_i points to F_{j+1}.

We define $\chi_{(i,j)}$ different target inner-levels $\ell_1, \ell_2, \ldots, \ell_{\chi_{(i,j)}}$ in F_{j+1}, where ℓ_k is inner-level $\lambda - (2k-1)$ in F_{j+1}. (Note that target inner-levels are odd, because λ is even.) The parameter $\chi_{(i,j)}$ is the chromatic number of the dependency graph $G_{(i,j)}$. Since $d_{(i,j)} \leq d$, a trivial polynomial time coloring algorithm using $d+1$ colors shows that $\chi_{(i,j)} \leq \chi = d+1$. Each packet in Π_i is thus assigned a color between 1 and $\chi_{(i,j)}$. Denote by $\Pi_i(k)$ the respective subset of Π_i with color k. Packets in $\Pi_i(k)$ have target level ℓ_k. Note that in the above discussion we assume that $j < \gamma'$. If $j = \gamma'$ then all the target inner-levels are set to real level $2D-1$, which are still in F_j. By construction, the paths of packets of same color are *conflict-free*, i.e. don't share any edge, and thus can be routed together in "boats" (see below). Further, the fact that the last frame extends beyond level $2D$ does not cause a problem because no packet will every need to move into that region, as it will be absorbed before that.

3.3 Boats

A *boat* b is a pointer to a level. We have $\chi_{(i,j)}$ boats $b_1, \ldots, b_{\chi_{(i,j)}}$. Initially, b_k is NULL. At time step t_{4k-3}, boat b_k points to the first inner-level of F_j (the boat enters the wave). At each subsequent step, the boat points to the next higher inner-level, so that at time step t_{4k-3+l} it points to inner-level $l+1$. After the boat reaches the last inner-level of F_j it continues to the inner-levels of F_{j+1} until the boat reaches the target level ℓ_k of F_{j+1}, after which b_k becomes NULL again. Note that boats are spaced 4 levels away from each other, which will be important when an oscillating packet needs to be deflected (see below). When the context is clear, we use boat to refer to the inner-level it points to. Note that the last boat enters at $t_{4\chi_{(i,j)}-3}$, and takes $2\lambda - 2\chi_{(i,j)} + 1$ steps to leave the wave, so the number of time steps per phase is $\tau = 2(\lambda + \max_{i,j} \chi_{(i,j)} - 1) \leq 2(\lambda + \chi - 1)$.

The packets of $\Pi_i(k)$ will use boat b_k to move to their target level ℓ_k in F_{j+1}. Suppose $\pi \in \Pi_i(k)$ is oscillating with initial level ℓ at the beginning of phase ϕ. Packet π will continue to oscillate until its boat b_k is at inner-level ℓ, at which time packet π will "catch its boat" and move along with it. While on its boat b_k, π follows its current path until it reaches its target inner-level ℓ_k in F_{j+1}. If, during this trip, π passes through its target node it is absorbed; otherwise π reaches its target inner-level ℓ_k at which it will oscillate for the remainder of the phase. Note that b_k passes through odd inner-levels (in particular π's initial level) at odd time steps, so π is at its initial level when b_k passes through it.

Packet Injection. A packet $\pi \in \Pi_i(k)$ with source node in frame F_j, is injected into the network when its boat b_k passes through the source node. π then moves along with b_k, following its current path, until it reaches its target level ℓ_k. While packets move along their boats they may conflict with other packets; we now describe how to handle such conflicts.

3.4 Packet Conflicts

Suppose $\pi \in \Pi_i(k)$ is on its boat b_k, progressing along its current path to its target level ℓ_k. π cannot conflict with another packet of $\Pi_i(k)$ because their current paths are conflict-free ($\Pi_i(k)$ is an independent set in $G_{(i,j)}$). Earlier boats $b_{k'}$ with $k' < k$ are ahead of b_k, so π cannot conflict with packets in $\Pi_i(k')$. π can only conflict with packets in $\Pi_i(k'')$ for $k'' > k$, which are oscillating in F_j. In such a conflict, the oscillating packet is deflected (i.e., oscillating packets have lower priority than packets on boats). We show below that this does not disrupt the algorithm.

Suppose π deflects packet $\sigma \in \Pi_i(k'')$ which oscillates on edge $e = (v_\ell, v_{\ell+1})$ (ℓ is σ's inner-level in F_j). Packet π deflects σ at the (odd) time step t_k at which π passes through ℓ. Assume that σ followed edge $e' = (v_{l-1}, v_l)$ to reach v_ℓ. We deflect σ along edge e' to inner-level $\ell - 1$, (so that at time step t_{k+1}, σ appears in v_l). Note that this is always possible because no other packet oscillating at v_ℓ arrived there using edge e', because the packets that are oscillating at v_ℓ all followed the same boat, and hence had edge disjoint paths. Note also that a packet oscillating on the first inner-level may be deflected into the previous frame F_{j-1} by an injected packet, but this causes no problem. Packet σ now follows edge e' to appear back in v_l at the (odd) time step t_{k+2}. This is possible because at time step t_{k+1} there is no boat passing through inner-level $\ell - 1$ (boat b_{k+1} is two levels away), and thus σ cannot be deflected further. When packet σ is back at inner-level ℓ, it continues to oscillate in ℓ. Therefore, σ is always at level ℓ at odd time steps, and thus it can move with boat $b_{k'}$, when it passes through ℓ. Clearly, deflected packets remain on their path.

3.5 Routing Time

The only parameter that remains to be specified is α. This parameter determines the frame size λ which must be large enough to accommodate 2χ levels (at least χ odd target inner-levels) in F_{j+1}. Since $\chi \leq d + 1$, it suffices that $\lambda = 4\alpha \log(DN) \geq 2(d + 1)$. From Lemma 2.2, $d \leq 4 \log(DN)$ w.h.p., so ignoring the trivial case $D = N = 1$, $\alpha = 3$ will do.

The routing time is at most $m \cdot \tau$ (m phases, each of duration τ). Since $m = O(s + \gamma') = O(C + D/\log(DN))$, and $\tau = O(\lambda + \chi) = O(\log(DN))$ w.h.p. (Lemma 2.2), we obtain:

Theorem 3.1 *The routing time of the centralized algorithm is $O(C \log(DN) + D)$, with probability at least $1 - 1/DN$.*

4 Distributed Algorithm

We show how to make the centralized algorithm (Section 3) distributed. We assume that all nodes know the parameters C, D, and N. (This assumption is common to routing algorithms [10, 18, 24].) Given C, D, N, every node can compute $\lambda, \gamma', s, m, \chi, \tau$. Note, that nodes do not need to know the paths of the other packets, they only need to know the path of the packet they inject.

The setup of the distributed algorithm is similar to the centralized algorithm: packets follow boats on waves to reach their destinations. The entire centralized algorithm would work verbatim if not for the coloring of the dependency graphs $G_{(i,j)}$, which is the main centralized computation (since nodes need to know all the packet paths). Thus, we need a distributed coloring algorithm, which will compute a coloring as the packets follow the waves. We introduce the notion of reverse simulation to accomplish this.

4.1 Reverse Simulation

Let $\chi = 2\lambda/\alpha$ (λ/α is an upper bound on d, w.h.p., by Lemma 2.2). During phase ϕ, suppose that wave ω_i point to frame F_j. Packets of set Π_i follow wave ω_i. In F_j and F_{j+1} we define the initial and target levels as in the centralized algorithm. Consider the set of packets $A \subseteq \Pi_i$ which are oscillating at their initial inner levels in frame F_j, at the beginning of the phase. These packets will move to F_{j+1}, where they will oscillate in their target levels.

As in the centralized algorithm, packets will use boats to move to their target levels. There are χ boats. In order to follow the boats in a collision-free manner, the packets need to be colored so that packets of same color have conflict-free paths. In order to obtain the colors, the packets will simulate a distributed coloring algorithm, which consists of several rounds. In the first round each packet chooses a color randomly and uniformly among χ colors. Packets can then be divided into two disjoint sets: those that obtained a valid color (one that is different from the color of each of their neighbors in $G_{(i,j)}$); those that obtained an invalid color (one that coincides with the color of at least one neighbor). Packets now attempt to reach their target inner-levels. Packets assigned invalid colors will detect this when they collide with non-oscillating packets, and will attempt to correct this in the next round. This process continues until all packets have obtained valid colors. We now give the details.

A phase is divided into ξ rounds $r_1, r_2, \ldots, r_\xi$, and each round consists of 2τ time steps, which is twice the duration of a phase in the centralized algorithm; this is because packets will need to attempt to reach their target level, and return to their initial level in each round. Each round has χ boats and target levels (similar to the centralized algorithm). At the beginning of round r_1, each packet in A chooses a color uniformly and randomly among χ colors. Let A_1 be the set of packets with a valid color, and A'_1 the packets with an invalid color. Note that $A = A_1 \cup A'_1$.

During round r_1, *all* packets in A will follow their respective boats. The packets in A_1 will not be deflected, and they follow their respective boats to successfully reach their target levels where they will oscillate for the rest of the round. Some packets, $A''_1 \subseteq A'_1$, will collide with non-oscillating packets as they follow their boats. Such packets can mark themselves as members of A'_1. These packets need to choose new colors and try again. At the end of round r_1, *all* packets in A return to their initial level (see below). In round r_2, packets in set A''_1 choose a new color, and a subset $A'_2 \subseteq A''_1$ will still have an invalid color. A subset $A''_2 \subseteq A'_2$ will collide with non-oscillating packets, and will need to choose new colors in the next round. Continuing in this way, in round k, the packets in A''_{k-1} choose new colors, and those in $A'_k \subseteq A''_{k-1}$ still do not have a valid color. Of these packets, A''_k will collide with non-oscillating packets. We will show A'_ξ is empty w.h.p, i.e., all packets have a valid color by the last round. Thus, in the last round, all the packets reach their target inner-levels, where they will oscillate till the next phase. We give the details below.

We define 4 levels of priority, $0, 1, 2, 3$. When two or more packets collide, the packet with highest priority always wins, and ties are broken randomly. A packet which successfully reachs its target level in round k (without being deflected by non-oscillating packets) keeps its color in all subsequent rounds and attains priority 3 for the remainder of the phase, whenever it is not oscillating. An oscillating packet has priority 1. A packet that chooses a new color in a round attains priority 2 for the round. If, during the round, it collides with any priority 2 or 3 packet, it immediately attains priority 0 for the remainder of the round, and will select a new color in the next round. Such priority 0 packets do not "distract" other forward going packets, and they follow arbitrary paths, due to deflections, for the remainder of the round.

At the end of a round, all packets in A (with valid or invalid coloring) need to appear back at their initial levels. Let t be the time step that the last boat in the round leaves the network. After time t, all packets follow, in reverse, the path that they followed from the beginning of the round. Thus, by the end of the round, they appear at their initial level where they oscillate until the next round. The path reversal is accommodated by having the nodes store all their computations from the beginning of the round up to time t. After time t, the nodes simply do the reverse computations, since routing is a reversible operation. (This is why we need the round to be twice as long as τ.)

4.2 Packet Injections

So far we considered only the oscillating packets in Π_i, that already appear in F_j at the beginning of phase ϕ. We also need to consider the set of packets $B \subseteq \Pi_i$ that will be injected in F_j during ϕ. Packets of B can be further partitioned into two sets: B_1, which are the packets of B whose source are at odd inner-levels of F_j, and B_2, which have sources at even inner-levels of F_j. Packets of B_1 and B_2 are treated separately so that they can not interfere with each other.

We divide phase ϕ into three sub-phases ϕ_A, ϕ_{B_1}, and ϕ_{B_2} in which we send the packets of the respective sets A, B_1 and B_2 to F_{j+1}. Each sub-phase consists of ξ rounds. We also divide the frame F_{j+1} into three disjoint regions F_A, F_{B_1}, and F_{B_2}, each consisting of 2χ inner-levels and containing χ target levels. Region F_A occupies the upper one-third (right) inner-levels of F_{j+1}, F_{B_1} the middle one-third inner-levels, and F_{B_2} at the lower (left) one-third inner-levels. Packets of set A, B_1 and B_2, have their target levels in F_A, F_{B_1} and F_{B_2}, respectively.

During phase ϕ_A the packets of set A will move to region F_A, using the algorithm we described in Section 4.1. During ϕ_{B_1}, the packets of B_1 are injected into the network, and then they move to their target levels in region F_{B_1} using the reverse simulation technique that was used for packets in set A. The initial levels of the packets in B_1 are the inner-levels of their sources, and the packets are injected at the beginning of phase ϕ_{B_1}. Since a node injects at most 1 packet, the packets are guranteed to be able to oscillate on their initial inner-levels during the reverse simulation. At the beginning of phase ϕ_{B_2}, the packets of set B_2 are injected into the network. Those packets will move to their target levels in region F_{B_2} during phase ϕ_{B_2} using the reverse simulation technique that was used for packets in set A. Those packets will also oscillate on their initial inner-levels, which are even (as opposed to packets in A and B_1 which have odd initial inner-levels). In order to handle the even levels, during this phase the boats enter the frame F_j from inner-level 2.

4.3 Routing Time

First we determine an acceptable value of parameter α. A frame needs 2χ inner-levels for each of F_A, F_{B_1}, and F_{B_2}, so $\lambda \geq 6 \cdot \chi$ Since $\chi = 2\lambda/\alpha$, we obtain $\alpha \geq 12$, so $\alpha = 12$ will do.

We now choose the parameter ξ so as to ensure that the routing is succesful w.h.p.. Assume that $\chi \geq 2d$. Then, in a round, a packet picks a valid color with probability at least $\frac{1}{2}$. We denote a phase in which a packet eventually picks a valid color in one of the rounds as a succesful phase for the packet. If $\xi = 2\log(LN)$, then a particular phase will be unsuccesful for a particular packet with probability $2^{-2\log(DN)} = 1/(DN)^2$. The distributed algorithm will be succesful if every phase that a packet participates in is succesful. A packet participates in at most D/λ phases (as its path length is at most D). A packet is unsuccesful if one of its phases is succesful, therefore, by the union bound, a packet will be unsuccesful with probability at most $D/\lambda(DN)^2$. The distributed algorithm will be unsuccesful if one of the packets is unsuccesful, so applying the union bound again, the probability of failure is at most $DN/\lambda(DN)^2 < 1/DN$. Finally, by the union bound and Lemma 2.2, the probability that $\chi < 2d$ or some packet fails in some phase is at most $2/DN$.

There are m phases, and each phase has 3ξ rounds (since each sub-phase to route A, B_1, B_2 consists of ξ rounds), each of length 2τ time steps. Therefore, the routing time is $O(m\xi\tau)$. Since $\tau = O(\log(DN))$, $m = O(C + D/\log(DN))$, and $\xi = O(\log(DN))$, we obtain:

Theorem 4.1 *The routing time of the distributed algorithm is $O(C \log^2(DN) + D \log(DN))$, with probability $1 - O(1/DN)$.*

References

[1] A. S. Acampora and S. I. A. Shah. Multihop lightwave networks: a comparison of store-and-forward and hot-potato routing. In *Proc. IEEE INFOCOM*, pages 10–19, 1991.

[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*, pages 342–349, 1999.

[3] P. Baran. On distributed communications networks. *IEEE Transactions on Communications*, pages 1–9, 1964.

[4] Constantinos Bartzis, Ioannis Caragiannis, Christos Kaklamanis, and Ioannis Vergados. Experimental evaluation of hot-potato routing algorithms on 2-dimensional processor arrays. In *EUROPAR: Parallel Processing, 6th International EURO-PAR Conference*, pages 877–881. LNCS, 2000.

[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, January/February 1998.

[6] Sandeep N. Bhatt, Gianfranco Bilardi, Geppino Pucci, Abhiram G. Ranade, Arnold L. Rosenberg, and Eric J. Schwabe. On bufferless routing of variable-length message in leveled networks. *IEEE Trans. Comput.*, 45:714–729, 1996.

[7] A. Borodin, Y. Rabani, and B. Schieber. Deterministic many-to-many hot potato routing. *IEEE Transactions on Parallel and Distributed Systems*, 8(6):587–596, June 1997.

[8] J. T. Brassil and R. L. Cruz. Bounds on maximum delay in networks with deflection routing. *IEEE Transactions on Parallel and Distributed Systems*, 6(7):724–732, July 1995.

[9] 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.

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

[11] 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.

[12] Costas Busch, Malik Magdon-Ismail, and Marios Mavronicolas. Universal bufferless routing. In *Proceedings of the 2nd Workshop on Approximation and Online Algorithms (WAOA)*, September 2004. To appear.

[13] Costas Busch, Malik Magdon-Ismail, Marios Mavronicolas, and Roger Wattenhofer. Near-optimal hot-potato routing on trees. In *Proceedings of the 10th International Conference on Parallel and Distributed Computing (Euro-par)*, August-September 2004. To appear.

[14] 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.

[15] A. G. Greenberg and J. Goodman. Sharp approximate models of deflection routing. *IEEE Transactions on Communications*, 41(1):210–223, January 1993.

[16] W. D. Hillis. *The Connection Machine.* MIT press, 1985.

[17] 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.

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

[20] Frank Thomson Leighton, Bruce M. Maggs, Abhiram G. Ranade, and Satish B. Rao. Randomized routing and sorting on fixed-connection networks. *J. Algorithms*, 17(1):157–205, 1994.

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

[22] N. F. Maxemchuk. Comparison of deflection and store and forward techniuques in the Manhattan street and shuffle exchange networks. In *Proc. IEEE INFOCOM*, pages 800–809, 1989.

[23] 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.

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

[25] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[26] Rafail Ostrovsky and Yuval Rabani. Universal $O(\text{congestion}+\text{dilation}+\log^{1+\varepsilon} N)$ 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.

[27] 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.

[28] C. L. Seitz. The caltech mosaic C: An experimental, fine-grain multicomputer. In *Proceedings of the 4th Symp. on Parallel Algorithms and Architectures*, June 1992. Keynote Speech.

[29] B. Smith. Architecture and applications of the HEP multiprocessor computer system. In *Proceedings of the 4th Symp. Real Time Signal Processing IV*, pages 241–248. SPIE, 1981.

[30] T. Szymanski. An analysis of “hot potato” routing in a fiber optic packet switched hypercube. In *Proc. IEEE INFOCOM*, pages 918–925, 1990.

[31] Z. Zhang and A. S. Acampora. Performance analysis of multihop lightwave networks with hot potato routing and distance age priorities. In *Proc. IEEE INFOCOM*, pages 1012–1021, 1991.