

Efficient Bufferless Routing on Leveled Networks

Costas Busch, Shailesh Kelkar, and Malik Magdon-Ismael

Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY, USA.
{buschc,kelkas,magdon}@cs.rpi.edu

Abstract. We give near optimal bufferless routing algorithms for *leveled networks*. N packets with preselected paths are given, and once injected, the packets may not be buffered while in transit to their destination. For the preselected paths, the *dilation* D is the maximum path length, and the *congestion* C is the maximum number of times an edge is used. We give two bufferless routing algorithms for leveled networks:

(i) a *centralized* algorithm with routing time $O((C + D) \log(DN))$;

(ii) a *distributed* algorithm with routing time $O((C + D) \log^2(DN))$.

The distributed algorithm uses a new technique, *reverse-simulation*, which is used to obtain a distributed emulation of the centralized algorithm. Since a well known lower bound on the routing time is $\Omega(C + D)$, our results are at most one or two logarithmic factors from optimal.

1 Introduction

We study bufferless routing on leveled networks, where packets cannot be stored at nodes while in transit to their destination. In particular we consider *hot-potato* (or *deflection*) routing [2], in which packets get “deflected” (like a “hot-potato”) if they cannot make progress toward their destination. Bufferless routing is appropriate when buffering is costly or impossible, for example in optical networks.

A leveled network with *depth* L has $L + 1$ *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). Many routing problems on multiprocessor networks can be represented as routing problems on leveled networks, for example routing problems on the Butterfly, the *Mesh* (Figure 1), shuffle-exchange networks, multidimensional arrays, the hypercube, fat-trees, de Bruijn networks, etc. (see [7, 14] for more details).

We assume a *synchronous* routing model in which at each discrete time step, a node forwards at most one packet down any link (two packets may use a link, one in each direction). We study *many-to-one batch routing problems*: we are given N packets with *preselected* paths; each node is the source of at most one packet, but may be the destination of many packets. 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.

The *routing time* is the time at which the last packet reaches its destination. For the preselected paths, the *congestion* C is the maximum number of packets

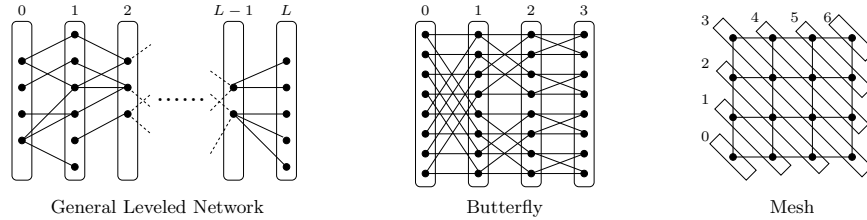


Fig. 1. Leveled networks

that traverse any edge, and the *dilation* D is the maximum path length. At most one packet can traverse any edge per time step, a lower bound on the routing time is $\Omega(C + D)$, and routing times close to this are optimal.

If two or more packets wish to follow the same link at the same time step, then a *conflict* occurs. Only one packet can follow this link, and the others must be *deflected* along alternative links (since there are no buffers). Deflections may change the preselected paths, however, we only consider bufferless algorithms in which the final path followed by the packets contains every edge in its preselected path. A routing time close to $(C + D)$ is still optimal with respect to any routing algorithm, buffered or not, even when we allow path deformation, provided that the final paths contain the preselected paths.

Our 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 at most a logarithmic factor from optimal. The second algorithm is distributed and has routing time $O(C \log^2(DN) + D \log(DN))$, a logarithmic factor worse than the centralized algorithm. In the distributed algorithm, all routing decisions are made locally. 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 efficiently emulates the centralized algorithm. The final paths used by the packets contain the preselected paths, which is useful to provide guarantees for the delivery time of the packets. Further, for the centralized algorithm, a packet never strays away from its preselected path.

The fundamental idea behind the centralized algorithm is to partition the network into *frames* each containing $O(\log(DN))$ levels. Packets are divided into $O(C)$ sets that move from frame to frame. The packets of a particular set are routed from frame to frame by coloring their dependency graph, which is a graph representing the conflicts between packet paths. It takes $O(\log(DN))$ time steps to move all the packets from one frame to the next, and since a packet traverses at most $O(D/\log(DN))$ frames to its destination, once a packet is injected, it is delivered in $O(D)$ time steps. The packet sets are injected sequentially, spaced by the time it takes to move packets from one frame to the next, so the last packet is injected at time $O(C \log(DN))$, resulting in a routing time $O(C \log(DN) + D)$.

Note that packets are not injected all simultaneously, but rather each packet is injected at an appropriate time after which it moves from frame to frame.

The main idea behind the distributed algorithm is to color the dependency graph in a distributed way: packets randomly select colors and if the coloring is valid, they will make it to their destination. If not, they use *reverse simulation* to trace their paths backwards, recompute colors, and the process repeats. The distributed coloring imposes an extra logarithmic factor in the routing time.

Related Work. Bufferless routing algorithms have been studied for various specific network topologies, [1, 3–8, 10–12, 15]. Most related to our work is [7], which gives a distributed algorithm for leveled networks with routing time $O((C + L) \log^9(LN))$. By using refined techniques, we improve this result by *seven* logarithmic factors, and obtain a result in terms of D , rather than L . A recent result in [9] gives 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. By taking advantage of the special structure of leveled networks, we can obtain a better routing time. Further, our centralized algorithm is one of the few hot-potato routing algorithms that keep the packets on their original preselected paths. Store-and-forward (buffered) routing algorithms exist with near optimal routing time for leveled as well as arbitrary networks (see for example [13]).

Paper Outline. We begin with some preliminaries (Section 2), followed by the centralized (Section 3) and the distributed algorithm (Section 4).

2 Preliminaries

We begin with some preliminaries regarding packet paths, oscillations, frames, and the dependencies between packets.

Paths. Every packet π has a preselected path p with path length $|p|$. Its *current path* at time step t , denoted $p(t)$, is defined as follows; we assume that the current path is maintained in the header of the packet, and is used for deciding where to send the packet at each time step. At time 0, $p(0) = p$, its preselected path. If at time t , packet π is in node v_i , with current path $p(t) = (v_i, v_{i+1}, \dots, v_k)$, and packet π successfully follows the first edge (v_i, v_{i+1}) (the packet moves forward), then, at time $t + 1$, packet π appears in node v_{i+1} with current path $p(t + 1) = (v_{i+1}, \dots, v_k)$. If, however, 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 $p(t + 1) = (v_j, v_i, v_{i+1}, \dots, v_k)$. 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 $(v_i, v_{i+1}, \dots, v_k)$. 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* $F_1, F_2, \dots, F_\gamma$, each containing λ levels (except for the last frame, which may

contain fewer). Frame F_i , $1 \leq i < \gamma$, consists of the λ levels $(i-1)\lambda, \dots, i\lambda-1$. Frame F_γ consists of the levels $(\gamma-1)\lambda, \dots, L$. Note that $\gamma = \lceil (L+1)/\lambda \rceil$. We will pick $\lambda = 4\alpha \log(DN)$, where α is an integer constant that will be defined later; thus, the frames have logarithmic size. (If $\log(DN)$ is not an integer, we use $\lceil \log(DN) \rceil$.)

We call the levels in frame F_i 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 $1, 3, \dots, \lambda-1$ (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 $s = 8\alpha eC$ sets, $\Pi_1, \Pi_2, \dots, \Pi_s$. 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 its 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 $d = O(\lambda/\alpha)$:

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

Groups. We partition the network into *groups*, such that each group is a collection of γ' consecutive frames, where $\gamma' = 2 \lceil D/\lambda \rceil$ (namely, the group consists of at most $2D + 2\lambda$ levels). We define two sets of groups. The first set of groups is $S_1 = \{g_1, g_2, \dots, g_{k_1}\}$, where group g_i consists of frames $F_{(i-1)\gamma'+1}, \dots, F_{i\gamma'}$. The group g_{k_1} consists of the rightmost frames in the network and may contain fewer than γ' frames. Note that the groups in S_1 do not share any levels. The second set of groups is $S_2 = \{h_1, h_2, \dots, h_{k_2}\}$, where group h_i consists of frames $F_{(i-1+1/2)\gamma'+1}, \dots, F_{(i+1/2)\gamma'}$. The group h_{k_2} , consists of the rightmost frames in the network and may contain fewer than γ' frames. 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 a group if its path lies entirely within the group. A packet belongs to at least one group (since its preselected path length is at most D and the groups have size $\geq 2D$). If the packet belongs to a group in S_1 , we assign it to S_1 , otherwise it belongs to a group in S_2 , and we assign it to S_2 . Note that a packet may belong to a group in S_1 and to a group in S_2 if its path is in the intersection of the two groups. In this case, it is assigned to S_1 . We denote by

$\Pi(S_i)$ the packets that belong to group S_i , and by $\Pi(x, S_j)$ the set of packets that belong to group x of S_j .

3 Centralized Algorithm

In the centralized algorithm, we route the packets in two consecutive *sessions*. First, we route the packets $\Pi(S_1)$ (belonging to groups in S_1), and then the packets $\Pi(S_2)$. Since the packets in $\Pi(S_2)$ are routed after the packets in $\Pi(S_1)$ have reached their destinations, they cannot possibly interfere with each other.

A particular session contains packets in various groups. Since a packet's path is contained in a single group, and since the groups are level-wise 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 in any one group. We will focus on the particular group $x = g_1$ of S_1 . The algorithm for other groups is identical. We will simplify the notation by dropping the x and S_j dependence. Hence, from now on, Π will denote $\Pi(g_1, S_1)$, and Π_i will denote $\Pi_i(g_1, S_1)$.

The session consists of m phases, each of duration τ time steps. Packets move on waves, from left to right, one frame per phase. Each packet set Π_i is associated with a particular wave, and each packet in Π_i uses this wave until it reaches its destination. Packets are assigned colors with respect to the dependency graph. Packets of the same color are routed together on a *boat* (level) in the wave. Different colors use different boats.

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 , and points to the next higher frame at each subsequent phase, so in phase ϕ_{i+k} , it points to frame F_{k+1} . Eventually, ω points to the last frame $F_{\gamma'}$, and then leaves the network (becomes NULL). There are s waves $\omega_1, \dots, \omega_s$ (equal to the number of packet sets). Wave ω_i enters the network at phase ϕ_{2i-1} . 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 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 then moves along its wave and is absorbed either when the wave contains its destination or its destination is one frame ahead of the wave. Note that waves are spaced 2 frames apart in order to avoid interference of packets in different waves while the waves move from frame to frame.

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_τ . 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 $\ell_1, \ell_2, \dots, \ell_{\chi_{(i,j)}}$ 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. do not 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 ever 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 $b_1, \dots, b_{\chi_{(i,j)}}$. 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 the term boat to refer to the inner-level it points to. Note that the last boat enters at time $t_{4\chi_{(i,j)}-3}$, 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 its 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 $e' = (v_{\ell-1}, v_\ell)$ 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

Since λ must be large enough to accommodate 2χ levels in F_{j+1} (at least χ odd target inner-levels), and $\chi \leq d + 1$, $\lambda = 4\alpha \log(DN) \geq 2(d + 1)$. From Lemma 1, $d \leq 4 \log(DN)$ w.h.p., so we can choose $\alpha = 3$. The routing time is $O(m \cdot \tau)$ (m

phases, each of duration τ). Using $m = O(s + \gamma') = O(C + D/\log(DN))$, and $\tau = O(\lambda + \chi) = O(\log(DN))$ w.h.p. (Lemma 1), we get

Theorem 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 when all nodes know C , D , and N (a commonly made assumption [7, 13]). Given C, D, N , nodes can compute $\lambda, \gamma', s, m, \tau$. (Nodes do not need information about the paths of packets other than the one they inject.)

The setup is similar to the centralized algorithm: packets follow boats on waves to their destinations. The major difference with the centralized algorithm is that the new algorithm provides a distributed coloring of the dependency graphs $G_{(i,j)}$. The distributed coloring is accomplished with the method of reverse simulation that is described below.

4.1 Reverse Simulation

In the distributed algorithm, we define $\chi = 2\lambda/\alpha$ (with $\alpha = 12$) which will be an upper bound on the number of colors assigned to the packets. Packets of set Π_i follow wave ω_i . Suppose ω_i points to frame F_j . We define the initial and target levels in F_j, F_{j+1} as in the centralized algorithm. 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 will move to F_{j+1} , where they will oscillate at their target levels.

A phase is divided into ξ rounds r_1, \dots, r_ξ , each of length 2τ time steps, twice as long as a phase in the centralized algorithm. Each round has χ boats and target levels as in the centralized algorithm. At the beginning of round r_1 , each packet in A chooses a color 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. ($A = A_1 \cup A'_1$.)

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 reaches 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} 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 guaranteed 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

Since $\chi \geq 2d$ w.h.p, a packet picks a valid color with probability $\geq \frac{1}{2}$, thus only $O(\log(DN))$ rounds are needed for every packet to pick a valid color, adding an extra factor of $\log(DN)$ to the centralized routing time. (We omit the details.)

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

References

1. S. Alstrup, J. Holm, K. de Lichtenberg, and M. Thorup. Direct routing on trees. In *Proc. SODA*, pages 342–349, 1999.
2. P. Baran. On distributed communications networks. *IEEE Transactions on Communications*, pages 1–9, 1964.
3. A. Ben-Dor, S. Halevi, and A. Schuster. Potential function analysis of greedy hot-potato routing. *Theory of Computing Systems*, 31(1):41–61, Jan. / Feb 1998.
4. S. N. Bhatt, G. Bilardi, G. Pucci, A. G. Ranade, A. L. Rosenberg, and E. J. Schwabe. On bufferless routing of variable-length message in leveled networks. *IEEE Trans. Comput.*, 45:714–729, 1996.
5. A. Borodin, Y. Rabani, and B. Schieber. Deterministic many-to-many hot potato routing. *IEEE Tran. on Parallel and Dist. Sys.*, 8(6):587–596, June 1997.
6. A. Broder and E. Upfal. Dynamic deflection routing on arrays. In *Proc. STOC*, pages 348–358, May 1996.
7. C. Busch. \tilde{O} (congestion + dilation) hot-potato routing on leveled networks. *Theory Comput. Syst.*, 37(3):371–396, 2004.
8. C. Busch, M. Herlihy, and R. Wattenhofer. Hard-potato routing. In *Proc. STOC*, pages 278–285, May 2000.
9. C. Busch, M. Magdon-Ismail, and M. Mavronicolas. Universal bufferless routing. In *Proc. 2nd Workshop on Approximation and Online Algorithms (WAOA)*, pages 239–252, 2004.
10. C. Busch, M. Magdon-Ismail, M. Mavronicolas, and R. Wattenhofer. Near-optimal hot-potato routing on trees. In *Proc. Euro-Par*, pages 820–827, 2004.
11. U. Feige and P. Raghavan. Exact analysis of hot-potato routing. In *Proc. FOCS*, pages 553–562, 1992.
12. C. Kaklamanis, D. Krizanc, and S. Rao. Hot-potato routing on processor arrays. In *Proc. SPAA*, pages 273–282, 1993.
13. F. T. Leighton, B. M. Maggs, A. G. Ranade, and S. B. Rao. Randomized routing and sorting on fixed-connection networks. *J. Algorithms*, 17(1):157–205, 1994.
14. F. Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays - Trees - Hypercubes*. Morgan Kaufmann, San Mateo, 1992.
15. F. Meyer auf der Heide and C. Scheideler. Routing with bounded buffers and hot-potato routing in vertex-symmetric networks. In *Proc. ESA*, 1995.