WAOA 2004
(Submitted to *SIAM Journal on Computing)*

Universal Bufferless Routing

Costas Busch
Department of Computer Science,
Rensselaer Polytechnic Institute.
Email: buschc@cs.rpi.edu
Fax: +1-518-276-4033

Malik Magdon-Ismail
Department of Computer Science,
NY 12180, USA.
Email: magdon@cs.rpi.edu
Fax: +1-518-276-4033

Marios Mavronicolas
Department of Computer Science,
University of Cyprus.
Email: mavronic@ucy.ac.cy
Fax: +357-228-92701

August 23, 2004

Abstract

In a routing problem, a set of packets must be routed from their sources to their destinations along specified paths in a connected network. Given paths with congestion C and dilation D a lower bound on the routing time is $\Omega(C + D)$. The celebrated result of Leighton, Maggs and Rao (1988) established, non-constructively, the existence of a routing schedule which uses constant size buffers and routes the packets in optimal time $O(C + D)$. Since then, constructive algorithms, as well as generalizations to distributed, buffered routing schedules have been developed.

A long standing open problem is to give or show the existence of *bufferless* routing algorithms with optimal performance guarantees. This is the problem we address here. Our main result is a new *deterministic* technique that constructs a universal bufferless algorithm by emulating a universal buffered algorithm. The heart of the emulation is to replace packet buffering with packet circulation on regions of the network. The cost of the emulation on the routing time is proportional to the square of the node buffer size used by the buffered algorithm. We apply this emulation to a simple randomized universal buffered algorithm to obtain a *distributed*, universal bufferless algorithm with routing time the optimal routing time within a poly-logarithmic factor:

$$O\left((C + D) \cdot \log^3(n + N)\right),$$

where n is the size of the network and N is the number of packets.

1 Introduction

Packet routing has received a large amount of attention over the past decade on account of its importance to applications ranging from parallel and distributed algorithms to communication networks. The task is to deliver packets from their sources to their destinations along specified paths in a given network. A packet routing algorithm is *universal* if it can be applied to any routing problem on any network topology. For a given set of paths, the *routing time* is the time at which the last packet reaches its destination. Universal algorithms with optimal or near-optimal routing time are known if packets may be buffered along their paths, [21, 28, 29, 33, 36].

A long standing and important open problem is to give universal *bufferless* routing algorithms with near optimal performance guarantees. In this paper, we will present a distributed bufferless routing algorithm that is optimal up to poly-logarithmic factors. We introduce a new technique for developing bufferless algorithms based upon emulating buffered algorithms. Applying this technique to a simple randomized buffered protocol gives the advertised result.

Preliminaries. A routing problem $Q = (G, \Pi, P)$ on the graph G with n nodes consists of a set of N packets $\Pi = \{\pi_1, \pi_2, \ldots, \pi_N\}$ that are to be routed on their respective paths $P = \{p_1, p_2, \ldots, p_N\}$, where p_i is a path in G. We will represent paths either as a sequence of edges, or as a sequence of nodes, and the length of a path $|p|$ is the number of edges in the path. The *edge-congestion* C is the maximum number of packets that use an edge in G, the *node-congestion* $\overline{C}$ is the maximum number of packets that use a node in G, and the *dilation* D is the maximum path length in P.

We assume a synchronous routing model, in which time is divided into a sequence of discrete time steps. An edge may be traveresed by at most one packet in either direction during a time step. A well known lower bound on the routing time in this model is given by $\Omega(C + D)$, and so the optimal routing time $rt^* = \Omega(C + D)$. In a buffered algorithm, packets may either traverse edges or be buffered at a node. In a bufferless algorithm, a packet must traverse an available edge at every time step.

An Impossibility Result. If all packets must follow the paths specified in P, without collisions or buffering, then the only degree of freedom for a bufferless routing algorithm is the injection times of the packets. Such a routing paradigm is known as *direct routing*, [4, 19]. In this case, it is shown in [19] that there exist routing problems for which bufferless routing times better than a $\sqrt{N}$ factor from optimal are not possible. Thus, if the paths remain unchanged, then near-optimal universal bufferless algorithms do not exist (where near optimal means within poly-logarithmic factors from the lower bound $C + D$). Thus, to obtain near-optimal bufferless schedules, we must allow packets to deviate from their paths. However, we still measure performance with respect to C and D of the original paths. The justification of this is that if the paths P themselves are optimal, i.e., they minimize $C + D$, then we obtain bufferless routing times that are near-optimal for the given sources and destinations. In this paper we are not concerned of how the optimal paths are obtained, but rather how to send the packets to their destinations given the paths.

Contributions. Our main result is a deterministic technique for bufferless emulation of buffered algorithms. Given a near-optimal universal buffered algorithm that routes problems with simple paths, and uses buffers of size γ, we give a universal bufferless algorithm, which emulates the buffered algorithm. The cost of the emulation on the routing time is $O(\gamma^2 \cdot \log n)$.

We apply this emulation result to a simple randomized buffered algorithm that uses $O(\log(n + N))$ buffers to obtain a bufferless routing algorithm with routing time $O((C+D) \cdot \log^3(n+N))$ with high probability, which approximates within poly-logarithmic factors the optimal routing time for

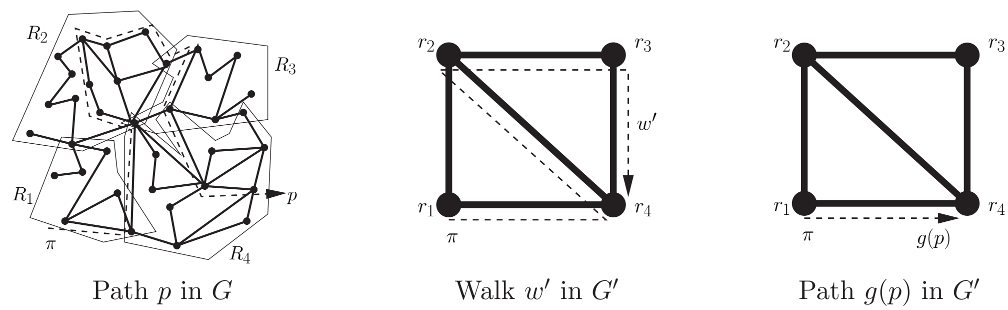


Figure 1: An example of a region graph.

the given paths. If all the nodes know the network topology, and the values of C and N, then the bufferless algorithm is distributed, i,e., routing decisions are made locally at each node.

Overview of the Approach. The main idea behind the bufferless emulation of a buffered algorithm is to use regions in the network in order to emulate buffer space. We decompose the graph into connected regions each containing approximately γ edges. The regions form a region graph, on which the nodes are regions. Now, a buffered algorithm executes as if the regions were the nodes. In the buffered algorithm, a packet is either buffered in a node (region) or "hops" from node to node. The path of each packet in the original graph is translated on a path on the region graph. The buffer needed at each node is at most γ. Figure 1 illustrates the general idea of decomposing the graph into regions and then mapping a packet's path to the graph in which every node corresponds to a region.

The buffered algorithm on the region graph is emulated by a bufferless algorithm on the original graph. If in the buffered algorithm a packet needs to be buffered in a node (region), then, in the emulation the packet "circulates" in the respective region by moving from one edge of the region to the next. A packet circulates until the buffered algorithm prescribes that the packet makes its next hop, in which case the packet moves to the respective adjacent region. Since the buffered algorithm requires γ buffer space per node (region), there is enough room to circulate all the packets in the γ edges of the region in a buffereless fashion.

Related Work. There are no known results for universal bufferless routing with near-optimal routing time guarantees. However, near-optimal bufferless routing has been obtained for specific bufferless routing models and architectures, which we summarize. In *hot-potato* routing, packets are deflected in a collision to available links [6]. Our model of bufferless routing is essentially the same with the hot-potato routing model, with the restriction that in collisions packets are deflected on particular available edges specified by the emulation (and not on any available edge as is done in typical hot-potato algorithms). Hot-potato routing algorithms have been extensively studied for a variety of architectures such as the mesh and torus [5, 7, 8, 12, 14, 16, 17, 18, 22, 23, 26, 27, 32, 39], hypercubes [11, 13, 22, 25, 35], trees [20, 37], vertex-symmetric networks [30], and leveled networks [10, 15]. Typically, by allowing packets to deviate from their paths slightly, one obtains routing times that are within poly-logarithmic factors of optimal. In *direct* routing, packets follow their paths without buffering and without any collisions, [4, 40, 19]. Busch *et al.* [19] give a comprehensive study of direct routing where they give a universal $O(C \cdot D)$ centralized algorithm, and near-optimal algorithms for the tree, mesh, butterfly, and hypercube. Wormhole routing is similar to direct routing, but here, packets occupy more than one edge [21, 24]. In [21] the authors give a randomized, universal distributed wormhole routing algorithm with routing time $O(L \cdot C \cdot D)$, where L is the length of the packet, which can be improved if the edges have higher bandwidths. A dual to direct routing is *time constrained routing*, where the task is to schedule as many packets as possible within a given time frame [1, 2]. In *matching routing*, packets are swapped at adjacent nodes, and permutation problems on trees have been studied in [3, 34, 41].

There are two variants of buffered algorithms. Those that use buffers on every edge (*edge-buffers*) and those that use buffers in every node (*node-buffers*). For non-bounded degree networks, these variants are distinct. The existence of optimal, universal *buffered* routing algorithms using constant size edge-buffers was first established by Leighton, Maggs and Rao [28]. Scheideler [38] showed that edge-buffers of size 2 are sufficient. Thereafter, the main focus has been on constructive algorithms with optimal, $O(C + D)$, routing time, [9, 29, 31, 33, 36]. These algorithms use large (proportional to the congestion C) buffers. Leighton *et al.* [28] improve this result, requiring only edge-buffers of size $O(\log ND)$ to obtain routing time $O(C + D \log ND)$. Cypher *at al.* [21] give an algorithm with edge-buffers of size $O(\log CD)$ and slightly better routing time. Our bufferless algorithm is based on emulating a universal buffered algorithm. However, the existing results, though powerful, do no suit our purpose because we need algorithms where the node-buffers are small (logarithmic), and so we offer a simple randomized algorithm that satisfies the conditions for bufferless emulation.

Paper Outline. We first discuss how to decompose a graph into connected regions of approximately a given size (Section 2). We then show how these regions are used for bufferless emulation of a buffered algorithm (Section 3). Finally we apply the emulation to a randomized buffered algorithm (Section 4) to obtain near-optimal universal bufferless routing (Section 5).

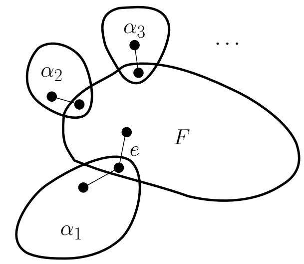
2 Regions

We first discuss how to decompose a connected graph G into connected components of approximately a specified size. Such a decomposition will be required by the bufferless emulation algorithm. Specifically, let $G = (V, E)$ be an undirected connected graph. Let F be a subset of the edges in E. The subgraph induced by F is the graph $H = (U, F)$, where U is the union of all vertices in V that are incident with edges in F. We say that the edge set F is *connected* if the induced subgraph H is connected. A *connected decomposition* of G is a partition of the edges in E into disjoint sets $E_1, E_2, \ldots, E_k$ such that $\cup_{i=1}^{k} E_i = E$ and every E_i is connected. We refer to the E_i's as the *connected edge sets* or *regions* in the decomposition, and denote the number of edges in E_i as the size of E_i, $|E_i|$. Notice that the subgraphs, $H_1 = (V_1, E_1), \ldots, H_k = (V_k, E_K)$ induced by the edge sets may have overlapping vertex sets. We say that E_i is *connected* to E_j if and only if $V_i \cap V_j \neq \emptyset$. Notice that if E_i is connected to E_j, then $E_i \cup E_j$ is a connected edge set.

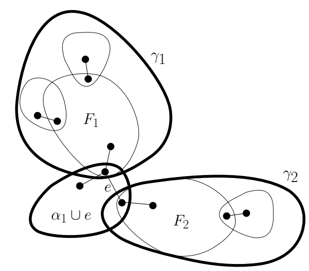
An $[\alpha, \beta]$-*partition* of G (if it exists) is a connected decomposition of G, $\{E_i, \ldots, E_k\}$, such that $\alpha \leq |E_i| \leq \beta$ for $i = 1, \ldots, k$. Notice that if $\alpha \approx \beta$, then an $[\alpha, \beta]$-partition decomposes G into connected edge sets of size approximately equal α. The main purpose here is to show that such approximate decompositions are possible for any connected graph. Our proof will be constructive, hence it can be directly converted to an algorithm. The following lemma will be instrumental in the proof. Essentially it states that a connected graph can be decomposed into two large connected edge sets. The proof is constructive.

Lemma 2.1 *Let $k > 1$. Any connected graph $G = (V, E)$ with $|E| > 3k - 3$ can be decomposed into two connected edge sets each of size at least k.*

Proof: Using a depth first search, determine a connected edge set F of size $2k-2$. Note that $|E-F| \geq k$. $E-F$ is composed of a number of connected edge sets $\alpha_1, \alpha_2, \alpha_3 \ldots$, each of which are not connected to any other, but all of which are connected to F. The situation is illustrated in the figure to the right. Let α_1 be the largest such satellite edge set of F. If $|\alpha_1| \geq k$ then α_1 and $F \cup \alpha_2 \cup \alpha_3 \cdots$ are both connected and have size $\geq k$. Suppose, on the other hand, that $|\alpha_1| \leq k-1$, in which case there is at least one other satellite. We then include the edge e as shown in the figure into α_1, removing it from F. If F remains a connected edge set, then we add to F one of the edges that connect it to one of its other satellites. It is also possible that one of its other satellites is now connected to α_1, in which case that can be merged with α_1, however this does not affect the argument. The end effect is that F now still has $2k-2$ edges, and the size of α_1, its largest satellite has strictly increased. We can repeat this process until either $|\alpha_1| \geq k$, and we are done, or in removing an edge from F and placing it in α_1, F becomes disconnected into exactly two edge sets F_1, F_2.



The situation is illustrated in the figure to the right. The edge e is therefore a bridge edge in F. We merge F_1 and F_2 with their respective satellites to get γ_1, γ_2 as illustrated. Note that since $|\alpha_1| \leq k$, $|\gamma_1| + |\gamma_2| \geq 2k-2$. If neither $|\gamma_1| \geq k$ nor $|\gamma_2| \geq k$, this means that $|\gamma_1| = |\gamma_2| = k-1$. In this case, merge γ_2 with e to form a connected edge set of size k. The remaining edges form a connected edge set of size at least $2k-2 \geq k$, so we are done. Thus, we suppose that one of γ_1 or γ_2 has size $\geq k$, and w.l.o.g., suppose that it is γ_1. Now consider $\alpha_1' = \gamma_2 \cup \alpha_1$. There are two cases: $|\alpha_1'| \geq k$, and we are done; or $|\alpha_1| < |\alpha_1'| < k$, in which case $|\gamma_1| \geq 2k-1$, in which case we can add edges from $F_1's$ satelites to F_1 to bring it up to the size $2k-2$. At this point, renaming $F_1 \to F$ and $\alpha_1' \to \alpha_1$, we will have recreated our original picture except that we have strictly increased the size of the largest satellite, and so once again we can continue adding edges to α_1. Thus we keep adding edges to α_1 until we get two connected edge sets both of size $\geq k$, concluding the proof. ■



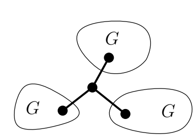
Theorem 2.2 (Existence of a $[k, 3k-3]$-partition) *Let $G = (V, E)$ be a connected graph. For any k, where $1 < k \leq |E|$, there exists a $[k, 3k-3]$-partition of G.*

Proof: If $k = 1$ the claim is obvious, so we will assume that $k \geq 2$. If $k \leq |E| \leq 3k-3$, then E itself is an $[k, 3k-3]$-partition so there is nothing to prove. We will now prove the claim by strong induction on $|E|$. The induction hypothesis is:

P(N) : There exists a $[k, 3k-3]$-partition for any $G = (V, E)$ whenever $|E| \in [k, N]$.

We claim that $P(N)$ is true for all N. We know that $P(3k-3)$ is true, so suppose that $P(N)$ is true for some $N \geq 3k-3$, and consider $P(N+1)$. Let $G = (V, E)$ be any graph with $|E| = N+1$. Since $|E| > 3k-3$, by Lemma 2.1, E can be decomposed into two connected edge sets E_1, E_2 with $k \leq |E_1| \leq |E_2| \leq N$. By the induction hypothesis, there exist $[k, 3k-3]$-partitions of E_1 and E_2. The union of these two partitions is a $[k, 3k-3]$-partition of E, concluding the proof. ■

The following example proves that the result of Theorem 2.2 is tight. For a given k, let G be any connected graph with $k-2$ edges, and connect 3 such graphs in a wheel configuration as shown on the right. It is easy to see that the only decomposition in which every edge set has $\geq k$ edges is the entire graph itself, which has $3k-3$ edges.



The proof in Theorem 2.2 is constructive, based upon the proof of Lemma 2.1, and one can show that the complexity of the algorithm to compute a decomposition is in $O(|E|^2)$.

2.1 Region Graph

Consider a connected graph $G = (V, E)$, with n nodes. Take an $[\alpha, \beta]$-partition of G, which gives regions (connected edge sets) $R_1, R_2, \ldots, R_k$. Let the subgraphs induced by these regions have vertex sets $U_1, U_2 \ldots, U_k$. The *region graph* $G' = (V', E')$, has a vertex set $V' = \{r_1, r_2, \ldots, r_k\}$ where each vertex r_i corresponds to the region R_i of G. Two vertices r_i, r_j are adjacent in G, i.e., $(r_i, r_j) \in E'$ if and only if $U_i \cap U_j \neq \emptyset$, i.e., the corresponding regions have intersecting vertex sets. An example of a region graph is given in Figure 1.

2.2 Routing Problems in Region Graph

Let $Q = (G, \Pi, P)$ denote a routing problem with edge-congestion C, node-congestion $\overline{C}$ and dilation D. Let $\{R_1, \ldots, R_k\}$ be an $[\alpha, \beta]$-partition of G. Every edge in G belongs to exactly one region. Let $G' = (V', E')$ be the corresponding region graph. The mapping $f : E \to V'$ is defined for every $e \in E$ by $f(e) = r_i$ if and only if $e \in R_i$. Consider a path $p \in P$, with $p = (e_1, e_2, \ldots, e_l)$. We define a function g which maps a path in G to a path in G' as follows. For any path $p = (e_1, e_2, \ldots, e_l)$ in G, consider the walk in G' given by $w' = (f(e_1), f(e_2), \ldots, f(e_l))$. $g(p)$ is the path obtained after removing all the cycles in w', $g(p) = (f(e_{i_1}), f(e_{i_2}), \ldots, f(e_{i_l}))$.

We now transform the routing problem Q on the original graph into a routing problem $Q' = (G', \Pi, P')$ on the region graph, in which the paths in G' are given by the transformed paths, $P' = \{p'_1, p'_2, \ldots, p'_N\}$ where $p'_i = g(p_i)$, $\forall p_i \in P$. Let C', $\overline{C'}$ and D' denote the edge-congestion, the node-congestion and the dilation of the paths in P'. For any routing problem, the edge-congestion is bounded by the node-congestion. A path uses node r_i only if it contains edges in R_i. By construction, $|R_i| \leq \beta$, so the number of edges in P that use R_i is at most βC, thus $\overline{C'} \leq \beta C$. Since $|g(p)| \leq |p|$ for any path p in G, we have the following lemma.

Lemma 2.3 (Congestion and dilation in the region graph) *$C' \leq \overline{C'} \leq \beta C$; $D' \leq D$.*

2.3 Euler Tours in Regions

We define Euler tours with respect to the directed representation $G^D = (V, E^D)$ of the undirected graph G: each (undirected) edge $(u, v) \in E$ is replaced by two directed edges $(u, v), (v, u) \in E^D$. Let R_i^D denote the region of G^D that corresponds to the region R_i in G. Since the in-degree equals the out-degree of every node in R_i^D, R_i^D has an Euler tour. Let $\psi_i = (v_1, v_2, \ldots, v_1)$ denote an Euler tour in R_i^D. Note that ψ_i is walk in R_i. We will refer to ψ_i as the "Euler tour" of R_i (an abuse of notation, since ψ_i is not an Euler tour of R_i). Note that for an $[\alpha, \beta]$-partition of G, every Euler tour ψ_i satisfies $2\alpha \leq |\psi_i| \leq 2\beta$.

3 Emulation

Let $G = (V, E)$ be a connected graph with n nodes and let $\{R_1, \ldots, R_k\}$ be an $[\alpha, \beta]$-partition of G with corresponding region graph $G' = (V', E')$. For routing problem $Q = (G, \Pi, P)$ in G, we obtain the corresponding routing problem $Q' = (G', \Pi, P')$ in G'. Let (s_i, d_i) denote the source and destination of each packet $\pi_i \in \Pi$, and let $S = \{(s_1, d_1), (s_2, d_2), \ldots, (s_N, d_N)\}$. Let $Q_s = (G, \Pi, S)$ denote the routing problem in G in which the packets need to be delivered from their sources to their destination, without necessarily following the paths in P.

The general idea behind our approach is to design a bufferless routing Algorithm B to solve the routing problem Q_s. The bufferless algorithm will depend on a buffered Algorithm A to solve the routing problem Q' in G'. The bufferless algorithm will then *emulate* the running of Algorithm A in G' to solve Q_s in G.

3.1 Buffered Routing in G' – Algorithm A

Our bufferless algorithm in G will emulate a buffered algorithm A in G'. Algorithm A solves routing problem Q' in G' and uses node-buffers of size at most γ to do so. We require algorithm A to receive at most γ packets at every time step. It is then possible to divide the execution of Algorithm A, into a sequence of *phases*, in which each phase has the following two properties:

(i) Each phase is a fixed time period consisting of at least one time step;

(ii) During each phase, each packet traverses at most one edge in G', and each node receives at most γ packets from adjacent nodes or through injection.

A trivial division of the execution of Algorithm A into phases that satisfies these two properties is to take each phase to be a single time step. In Section 4, we give a specific buffered Algorithm A_1 in which each phase contains $O(\log(n + N))$ time steps. During a single phase of Algorithm A, a packet π may perform one of four actions (in G'):

(i)	Remain in the buffer of its current node.	[**Buffering**]
(ii)	Move from its current node to a neighboring node.	[**Packet Transfer**]
(iii)	Be injected into the network at its source node.	[**Injection**]
(iv)	Move to and be absorbed in its destination node.	[**Absorbtion**]

3.2 Bufferless Routing in G – Algorithm B

Algorithm B emulates the phases of Algorithm A (which is faster than emulating the individual time steps of Algorithm A). Algorithm B emulates the buffering of packets and their transfer from node to node using an $[\alpha, \beta]$-partition of G, where $\alpha = 2\gamma$. (We assume that $2\gamma \leq |E|$ and by Theorem 2.2, we can set $\beta = 6\gamma - 3$.) In Algorithm A, when a packet is buffered in a node r_i of G', then Algorithm B emulates this by letting the packet circulate in the edges of region R_i in G. When in Algorithm A a packet is transferred from node r_i to node r_j of G', in Algorithm B the packet is transferred from region R_i to region R_j in G. Similarly, algorithm B handles the packet injection and absorbtion. Next we describe the emulation in more detail.

Phases and Rounds. Let Φ denote the number of phases in Algorithm A. In Algorithm B, time is divided into Φ phases. Each phase of B emulates a phase of A. In order to perform the emulation of a phase, Algorithm B further divides each phase into Σ *rounds*, where Σ is defined below. The duration of each round is $T_r = 4\beta^2 + 4\beta$ time steps. Thus the bufferless algorithm runs for $\Phi \cdot \Sigma \cdot T_r$ time steps in total.

For the duration of a round, a region is either in the *sending* or the *receiving* state – we say that the region is sending, or receiving. In the emulation, when a packet has to be transferred from one region to the next, the first region should be sending while the other receiving. We guarantee that for any pair of adjacent nodes there is a round in each phase in which one region is sending and the other is receiving (and vice-versa), as follows.

In order to determine if a region is sending or receiving, we first obtain a vertex coloring of G'. Let δ_i denote the color (non-negative integer in binary representation) assigned to node r_i in G' (which will also be the color of region R_i), and let δ denote the maximum color we obtain from the vertex coloring. Note that $\delta \leq n'$, where $n' = |V'| \leq |E|/\alpha$. Let σ denote the number of bits in δ,

$\sigma = \lceil \log \delta \rceil \le \lceil \log n' \rceil$. By pre-padding with zeros, we assume that every δ_i has σ bits. We define the *state parameter* $\mathbf{x}_i$ for region R_i to be the 2σ-bit integer $\bar{\delta}_i \delta_i$, where $\bar{\delta}_i$ is the binary complement of δ_i. We use the notation $\mathbf{x}_i(k)$ to denote the k-th bit of $\mathbf{x}_i$. We set $\Sigma = 2\sigma \le 2\lceil \log n' \rceil$, i.e., each phase in Algorithm B, consists of 2σ rounds, $\omega_1, \omega_2, \ldots, \omega_{2\sigma}$. During round ω_k, if $\mathbf{x}_i(k) = 0$ then region R_i is sending, otherwise, if $\mathbf{x}_i(k) = 1$, then region R_i is receiving. Our assignment of colors ensures that during every phase, a region can send or receive from each of its neighbors.

Lemma 3.1 *If R_i and R_j are adjacent, then during every phase ϕ, there is at least one round ω_s (ω_r) in which R_i is sending (receiving) and R_j is receiving (sending).*

Proof: Since R_i and R_j are adjacent, δ_i and δ_j must differ at some bit s, $0 \le s \le \sigma - 1$. Thus, rounds s and $s + \sigma$ satisfy the requirements, since $\overline{\mathbf{x}_i(s+\sigma)} = \mathbf{x}_i(s) = \overline{\mathbf{x}_j(s)} = \mathbf{x}_j(s+\sigma)$. ■

Packet Circulation. Packet circulation is a basic function for the emulation. During packet circulation, a packet π repeatedly follows the Euler tour of the region R_i that it is in: at each time step, packet π follows the next edge in the Euler tour; when π reaches the end of the Euler tour it continues from the beginning of the tour, and so on. At the time step in which packet π traverses an edge $e \in \psi_i$, we say that e is the *current* edge of π.

At each round of a phase, a region is either sending or receiving. The speed at which a packet circulates in its region depends on whether the region is sending or receiving. If the region is receiving, then the packet follows the Euler tour in the normal fashion.

If the region is sending, then the packet moves at an effectively slower speed as follows. At time step 0 (the beginning of the round), suppose that π is at node u with current edge $e = (u, v) \in \psi_i$. At time step 0, packet π follows its current edge (u, v) and at time step 1, π appears in node v. At time step 1, suppose that its new current edge in ψ_i is (v, w); the packet *does not* follow its new current edge in ψ_i, but instead it follows edge (v, u) from v back to u, and thus at time step 2, it appears back in node u. Thus after two time steps, the packet has effectively not moved. We call such an operation an *oscillation*, and we say that packet π oscillates on its current edge in the Euler path. The time period of the oscillation is 2 time steps, The packet continues in this fashion for subsequent time steps, so at even time steps $t = 2i$, it appears in node u, and at odd time steps $t = 2i + 1$ it appears in node v, for $i \ge 0$. The packet performs β such oscillations on its current edge e, and so after 2β time steps, the packet appears at u and follows edge e for the last time. At time step $T_s = 2\beta + 1$, the packet is now at v and at this point it stops oscillating on edge e and begins oscillating on its new current edge $(v, w) \in \psi_i$. Thus, after T_s time steps, the packet advances by one edge in the Euler path of ψ_i. Consequently, since $|\psi_i| \le 2\beta$, after $2\beta T_s = 4\beta^2 + 2\beta$ time steps, a packet circulating in region R_i has oscillated at least once on every edge of ψ_i.

Lemma 3.2 *After $4\beta^2 + 2\beta < T_r$ time steps, a packet circulating in a sending region R_i has oscillated at least once on every edge in ψ_i.*

Suppose that the directed edge $e = (u, v) \in \psi_i$, is an edge in the Euler path of a receiving region R_i. If at time step t, no packet has edge e as its current edge, then we say that e is *empty*. At each time step, we say that an empty edge is associated with an *empty slot*. Empty slots are similar to packets in that they too circulate – as the packets in a receiving region circulate (forwards) in ψ_i, the empty slots circulate in ψ_i at the same rate. They continue to circulate until some packet occupies the empty edge.

Emulation of Buffering. Suppose that packet π is buffered at node r_i of G' during the execution of phase ϕ of Algorithm A. Assume that in Algorithm B, packet π is in region R_i of G. Packet π will circulate in R_i through the entire phase ϕ.

Lemma 3.3 *If packet π is in R_i at the end of phase $\phi - 1$ of bufferless Algorithm B, and in phase ϕ of buffered Algorithm A it is buffered in node r_i, then in phase ϕ of bufferless Algorithm B, it can be buffered in region R_i using circulation.*

Emulation of Packet Transfer. Suppose that in phase ϕ of Algorithm A, packet π moves from node r_i to node r_j. Assume that at the beginning of phase ϕ in Algorithm B, packet π is in region R_i. During phase ϕ in Algorithm B, π will move from R_i to R_j as follows. Packet π will circulate in R_i until a round ω of ϕ in which R_i is sending and R_j is receiving (the existence of such a round is guaranteed by Lemma 3.1).

Since r_i and r_j are adjacent in G', there exists a node u which is common to R_i and R_j. Since node u is in R_i, there exists an edge $e_i = (u_i, u) \in \psi_i$ on the Euler path of R_i. Similarly, there exists an edge $e_j = (u, u_j) \in \psi_j$ on the Euler tour of R_j. During round ω, packet π circulates (in slow mode) in region R_i along the Euler tour ψ_i. At some particular slow time step τ of the round, the current edge of π will be e_i. During the course of its $T_s > \beta$ oscillations on edge e_i, the packet will appear at the common node u at the $\beta + 1$ times $\tau + 1, \tau + 3, \ldots, \tau + 2\beta + 1$. If at any of these times, the edge $e_j \in \psi_j$ is an empty slot, i.e., not the current edge of any packet circulating (in normal mode) in R_j, then π switches from oscillation on edge e_i, making e_j its new current edge. π now continues to circulate in R_j at normal speed. Note that π will have completed its circulation on edge e_i in at most $4\beta^2 + 2\beta$ time steps, thus π will enter R_j within the first $4\beta^2 + 2\beta$ time steps of round ω.

We now show that during round ω, for at least one of the time steps $\tau+1, \tau+3, \ldots, \tau+2\beta+1$, the edge $e_j \in \psi_j$ will be an empty slot. Remember that empty slots circulate in R_j at the rate of one edge per time-step. Thus, if an empty slot is not occupied by any packet during its circulation, then every edge in ψ_j will become an empty slot at least once during a consecutive 2β time steps. In particular, edge e_j will become an empty slot at least once in the time steps $\tau + 1, \tau + 2, \tau + 3, \ldots, \tau + 2\beta + 1$. A problem arises if e_j becomes empty at time $\tau + k$ where k is even, because then packet π will not be at node u, able to utilize this edge. This problem is solved if there is a second *consecutive* empty slot in R_j that will also not be occupied by any other packet during its circulation. This second empty slot must also appear at least once in the time steps $\tau + 1, \tau + 2, \tau + 3, \ldots, \tau + 2\beta + 1$, and since both these empty slots cannot appear at $\tau + k$ for k even, we are assured that π will be able to transfer into R_j.

From the previous phase, suppose that there are at most γ packets circulating in R_j. During the current phase, at most γ more packets will enter R_j, by definition of the buffered Algorithm A. In the worst case, all the $\gamma - 1$ packets other than π that will enter have already entered, and none of the packets that are to leave this region in this phase have left yet. In this case there are at most $2\gamma - 1$ packets that could be circulating in R_j during round ω. Since $\alpha = 2\gamma$ and there are at least $2\alpha = 4\gamma$ edges ψ_j, we conclude that there are at least $2\gamma + 1$ empty slots during round ω. By the pigeonhole principle, at least two of these empty slots must be consecutive, and we have the following lemma.

Lemma 3.4 *Suppose that in phase $\phi-1$ of bufferless Algorithm B, at most γ packets are circulating in region R_j, and that packet π is circulating in the adjacent region R_i. Suppose that in buffered Algorithm A, packet π moves from r_i to r_j in phase ϕ. Then during phase ϕ of bufferless Algorithm B, packet π can be transferred (using circulation) from region R_i to R_j.*

Emulation of Injection. Suppose that π is a packet that is to be injected into the network in Algorithm A. Let p be the path of π in G, and let e be the first edge in this path, and u the injection node. Suppose that $e \in R_i$ – note also that $u \in R_i$. In this case, π is injected into node r_i

in G'. Suppose that π is injected into r_i during phase ϕ of buffered Algorithm A. Then π will be injected into R_i in phase ϕ of bufferless Algorithm B during the last round in which R_i is receiving. After injection, it will circulate in R_i until the end of phase ϕ. Let $e = (u, v)$ be an edge on the Euler path ψ_i of R_i. We know that from the previous analysis of packet transfer that if R_i had at most γ packets circulating in phase $\phi - 1$, then e will be an empty slot at least $2\gamma + 1$ times during every receiving round. At the time that e becomes empty, π is injected into the network and e becomes its current edge. π then continues to circulate in R_i. Note that at least γ packets could be injected into R_i from the *same* injection node during a single receiving round.

Lemma 3.5 *Suppose that in phase $\phi - 1$ of bufferless Algorithm B, at most γ packets are circulating in region R_i. Suppose that packet π has first edge $e \in R_i$ and that during phase ϕ of buffered Algorithm A, packet π is injected into node r_i. Then during phase ϕ of bufferless Algorithm B, packet π can be injected into R_i. Further, at least γ packets can be injected into the same node during a single receiving round.*

Emulation of Absorbtion. Suppose that packet π moves from node r_i to its destination node r_j in phase ϕ in buffered Algorithm A. We use the packet transfer emulation to first move the packet from region R_i to R_j in phase ϕ. This takes at most $4\beta^2 + 2\beta$ time steps. Then the packet circulates in the receiving region at normal speed until it reaches its destination node, at which point it is absorbed. Since the packet completes the Euler tour for R_j in at most 2β time steps, the number of time steps to move and be absorbed is $4\beta^2 + 4\beta \leq T_r$, giving the following lemma.

Lemma 3.6 *Suppose that in phase $\phi - 1$ of bufferless Algorithm B, at most γ packets are circulating in region R_j, and that packet π is circulating in the adjacent region R_i. Suppose that in phase ϕ of buffered Algorithm A, packet π is absorbed in r_j. Then, during phase ϕ of bufferless Algorithm B, packet π can be absorbed at its destination node in region R_j.*

3.3 Analysis of Emulation by Bufferless Algorithm B

First, we prove that Algorithm B correctly emulates Algorithm A. We then analyse the routing time of Algorithm B in G in terms of the routing time of Algorithm A in G'.

Correctness. Assume that $\alpha = 2\gamma \leq |E|$ in order to guarantee the existence of the $[\alpha, \beta]$-partition. Algorithm B correctly emulates algorithm A if at the end of every phase ϕ:

i. In Algorithm A, packet π is in node r_i iff in Algorithm B it is circulating in region R_i

ii. In algorithm A packet π is injected (absorbed) at node r_i, if and only if in Algorithm B packet π is injected (absorbed) into region R_i.

We show by induction on ϕ that Algorithm B correctly emulates Algorithm A. Observe that when $\phi = 1$, Algorithm A can only inject packets into nodes. The conditions of Lemma 3.5 are satisfied, and since at most γ packets are injected into a node in G', Algorithm B can succesfully inject these packets into the corresponding regions. Suppose that Algorithm B correctly emulates Algorithm A up to phase $\phi_0 \geq 1$. At the end of phase ϕ_0, there are at most γ packets circulating in any region R_i since every packet π in node r_i in the execution of Algorithm A is in region R_i in the execution of Algorithm B. Thus, the conditions of Lemmas 3.3, 3.4, 3.5, and 3.6 are satisfied for every packet π. Every action that π could make in phase $\phi_0 + 1$ of Algorithm A can now be emulated in phase $\phi_0 + 1$ of Algorithm B. By induction, we have the following theorem.

Theorem 3.7 (Correctness of Emulation) *Algorithm B correctly emulates in G every phase in the execution of Algorithm A in G'. Each packet in Algorithm B follows a path from its source to destination, hence Algorithm B solves routing problem Q_s without buffers.*

Routing Time. Let $rt_B(Q_s)$ be the routing time for Algorithm B to solve routing problem Q_s. Let $\Phi_A(Q')$ be the number of phases used by Algorithm A to solve routing problem Q'. Since Algorithm B emulates Algorithm A phase for phase, the number of phases of algorithm B is also $\Phi_A(Q')$. The routing time is therefore given by $\Phi_A \cdot \Sigma \cdot T_r$. Since $T_r = 4\beta^2 + 4\beta$, $\beta = 6\gamma - 3$ and $\Sigma = 2\lceil \log \delta \rceil$, we obtain:

Theorem 3.8 (Bufferless Routing Time) $rt_B(Q_s) = \Theta(\Phi_A(Q') \cdot \gamma^2 \cdot \log \delta)$.

Since $\delta \leq |E|/\alpha = O(n^2)$, we have that $rt_B(Q_s) = O(\Phi_A(Q') \cdot \gamma^2 \cdot \log n)$

4 A Randomized Buffered Algorithm

We give a buffered algorithm that can be used to obtaining bufferless routing on arbitrary networks. Since the per-node buffer size enters into the routing time of the bufferless emulation, it is necessary to have buffered algorithms that limit the amount of per-node buffering. We refer to this algorithm as Algorithm A_1.

Algorithm A_1 is a randomized routing algorithm for routing porblems with simple paths, in arbitrary networks. Let $Q' = (G', \Pi, P')$ be a routing problem with acyclic paths P' on an arbitrary graph $G' = (V', E')$. Let $\overline{C'}$ be the node-congestion and and D' the dilation. Let N be the number of packets and n' the size of V'. Algorithm A_1 uses buffers of size $\gamma = 6\log(n' + 2N)$.

Algorithm 1 Buffered Algorithm A_1

Divide time into phases of length γ time steps.
for Each packet π **do**
    π selects uniformly at random an injection phase ϕ_π between phases 1 and $12\overline{C'}/\gamma$;
    Packet π is injected at the first time step of phase ϕ_π;
    Packet π follows its path at the speed of one edge per phase;

We will show that with high probability, Algorithm A_1 succesfully routes the packets, and at the same time satisfies the requirements in Section 3.1. For Algorithm A_1 and phase ϕ, define the following properties:

P1(ϕ): In phase ϕ every packet in the network succesfully traverses one edge in its path.

P2(ϕ): No more than γ packets are buffered at any node during phase ϕ.

P3(ϕ): No more than $\gamma/2$ packets arrive at any node during phase ϕ.

P4(ϕ): No more than $\gamma/2$ packets remain at any node at the end of phase ϕ.

Note that *P3* is stronger than we need. We introduce property *P4* for technical convenience. Since the maximum injection phase is $12\overline{C'}/\gamma$ and the maximum path length is D', we have:

Lemma 4.1 *If P1-P4 holds for $12\overline{C'}/\gamma + D'$ phases, then $\Phi_{A_1}(Q') \leq 12\overline{C'}/\gamma + D'$, and Algorithm A_1 is a valid algorithm for bufferless emulation.*

Let $\mathbf{P}[\phi_0]$ be the probability that properties *P1-P4* hold for all phases $\phi \leq \phi_0$. $\mathbf{P}[0] = 1$ by default. We now give a lower bound for $\mathbf{P}[\phi_0 + 1]$ in terms of $\mathbf{P}[\phi_0]$.

Lemma 4.2 *If P1-P4(ϕ_0) are true and P3($\phi_0 + 1$) is true, then P1-P4($\phi_0 + 1$) are true.*

Proof: If no more than $\gamma/2$ packets arrive at a node during phase $\phi_0 + 1$, then since *P4*(ϕ_0) is true, there are at most γ packets in the node during any time step of phase $\phi_0 + 1$, therefore *P2*($\phi_0 + 1$) is true. In the worst case all the at most $\gamma/2$ packets in the node at the end of phase ϕ_0 may leave sequentially on a single edge, requiring at most $\gamma/2$ time steps, which is less than the duration of the phase, so *P1*($\phi_0 + 1$) is true. The packets remaining in the node at the end of phase $\phi_0 + 1$ are only those that entered, which is at most $\gamma/2$ packets, thus *P4*($\phi_0 + 1$) is true. ∎

By induction, we obtain the following corollary.

Corollary 4.3 *P1-P4(ϕ) are true for all $\phi \leq \phi_0$ if and only if* P3*(ϕ) is true for all $\phi \leq \phi_0$.*

Thus, $\mathbf{P}[\phi_0+1] = \mathbf{P}[\{P1\text{-}P4(\phi) \text{ are true for } \phi \leq \phi_0\} \wedge \{P3(\phi_0+1) \text{ is true}\}]$. Noting that $\mathbf{P}[A \wedge B] = 1 - \mathbf{P}[\sim A \vee \sim B] \geq 1 - \mathbf{P}[\sim A] - \mathbf{P}[\sim B]$,

$$\mathbf{P}[\phi_0 + 1] \geq \mathbf{P}[\phi_0] - \mathbf{P}[\{P3(\phi_0 + 1) \text{ is false}\}]. \tag{1}$$

Consider a node v, and phase $\phi_0 + 1$. Let q_π be the probability that packet π arrives at node v during phase $\phi_0 + 1$ which can happen only if it is injected at a particular phase. Since the probility that it is injected at that particular phase is $\gamma/12\overline{C'}$, we conclude that $q_\pi \leq \gamma/12\overline{C'}$ if π uses node v (at most $\overline{C'}$ such packets), and 0 otherwise. Let $X_i(v) = 1$ if packet π_i appears at node v at phase $\phi_0 + 1$. $X_i(v)$ are independent random variables, whose sum is the number of packets that appear in node v at phase $\phi_0 + 1$. Let $X(v) = \sum_i X_i(v)$. $\mathbf{E}[X(v)] = \sum_i q_{\pi_i} \leq \overline{C'} \cdot \gamma/12\overline{C'} = \gamma/12$, so applying a version of the Chernoff bound gives

$$\mathbf{P}[X(v) > \gamma/2] < 2^{-\gamma/2}.$$

Applying the union bound now gives that $\mathbf{P}[\max_v X(v) > \gamma/2] < n' 2^{-\gamma/2}$, giving

Lemma 4.4 $\mathbf{P}[\{P3(\phi_0 + 1)$ *is false*$\}] < n' 2^{-\gamma/2}$.

Using (1), Lemma 4.4 and the fact that $\mathbf{P}[0] = 1$, we get the following result by induction.

Lemma 4.5 $\mathbf{P}[\phi_0] \geq 1 - \phi_0 n' 2^{-\gamma/2}$.

Since $n' < n' + 2N$, $12\overline{C'}/\gamma + D' < n' + 2N$ (because $\overline{C'} \leq N$ and $D' \leq n'$), and $2^{-\gamma/2} = (n' + 2N)^{-3}$, by setting $\phi_0 = \Phi_{A_1}(Q')$ in Lemma 4.5, and using Lemma 4.1, we obtain the following result.

Theorem 4.6 (Routing time of Algorithm A_1) *With probability at least $1 - O(1/(n' + 2N))$, Algorithm A_1 solves routing problem Q' in at most $12\overline{C'}/\gamma + D'$ phases, satisfying P1-P4 in each phase. The node-buffer size required is $\gamma = 6\log(n' + 2N)$.*

5 A Universal Bufferless Routing Algorithm

We use buffered Algorithm A_1 to construct bufferless Algorithm B_1 for arbitrary networks. Algorithm B_1 emulates Algorithm A_1. The buffer size used by algorithm A_1 is $\gamma = 6\log(n' + 2N)$. Since $n' \leq |E|/\alpha$, in order to guarantee the existence of an $[\alpha, \beta]$-partition, we assume that $\alpha \leq |E|$. Since $\alpha = 2\gamma$, we assume that $12\log(|E|/\alpha + 2N) \leq |E|$. It is sufficient that $2N \leq 2^{|E|/12} - |E|$.

Suppose $2N \leq 2^{|E|/12} - |E|$. Since $n' \leq n^2/2$, $\gamma \leq 6\log(n^2/2 + 2N)$, independent of G'. Combining Theorems 3.8 and 4.6, and the fact that in the emulation, $\Phi_{B_1}(Q_s) = \Phi_{A_1}(Q')$, we obtain that $rt_{B_1}(Q_s) = O((12\overline{C'}/\gamma + D') \cdot \log\delta \cdot \log^2(n + 2N))$. Using Lemma 2.3 and the facts that $\beta \leq 6\gamma$ and $\delta \leq n' = O(n^2)$, we obtain that $rt_{B_1}(Q_s) = O((C+D) \cdot \log n \cdot \log^2(n+N))$, with probability at least $1 - O(1/(n'+N))$.

Consider now the case when $2N \geq 2^{|E|/12} - |E|$. We can send the N packets of routing problem Q_s on G to their destinations one after the other. Each packet takes time $O(D)$ to be delivered to its destination, and thus the total routing time to send all the packets is $O(DN)$. Clearly, $C \geq N/|E|$, and thus $C \geq (2^{|E|/12} - |E|)/2|E|$. Since $|E| = O(\log(N))$ and $D \leq |E|$, the routing time is $ND \leq CD|E| = O(C \log^2(N))$. This simple algorithm can easily be converted to a distributed algorithm with the same routing time. In this case, each node assigns to the packets it injects priorities according to the node's id and the order that the packet is injected. Thus, each packet in the network has a distinct priority. Packets are injected whenever there are free links and in conflicts higher priority packets win. If a packet is unable to remain on its original path, it follows any shortest path to its destination, and it is possible to do so after all other packets with lower priority have been delivered to their destinations.

Combining the above results for both cases of the number of packets, we obtain:

Theorem 5.1 $rt_{B_1}(Q_s) = O((C+D) \cdot \log n \cdot \log^2(n+N))$*, with probability at least* $1 - O(1/(n'+N))$.

6 Discussion

We have presented a distributed algorithm for routing packets in bufferless networks. Our algorithm is based on the emulation of algorithms with buffers. We partition the original graph into regions, and construct a respective region graph. Each region serves the purpose of a buffer. We then consider an algorithm with buffers on the region graph, and emulate this algorithm by circulating the packets in the regions, and thus avoiding the need of buffers. With this technique, the resulting routing time of our algorithm is $O\left((C+D) \cdot \log^3(n+N)\right)$, which is poly-logarithmic factors away from the optimal for the given paths.

For any set of packets there is an optimal selection of paths which minimizes $C + D$. Denote by $C^* + D^*$, the sum of congestion and dilation of the optimal paths. Given the optimal paths, our algorithm sends the packets to their destinations in time within poly-logarithmic factors from optimal, that is, $O\left((C^* + D^*) \cdot \log^3(n+N)\right)$. For a set of packets we can define the *bufferless set ratio* as the ratio between the smallest possible routing time of an algorithm without buffers and the smallest possible routing time of an algorithm with buffers. The *bufferless ratio* is the maximum set ratio of any packet set on any graph. Our result shows that the bufferless ratio is at most $\log^3(n+N)$. A interesting problem is to determine whether the bufferless ratio is even smaller than this.

References

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

[2] Micah Adler, Arnold L. Rosenberg, Ramesh K. Sitaraman, and Walter Unger. Scheduling time-constrained communication in linear networks. In *Proceedings of 10th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 1998.

[3] 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.

[4] 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.

[5] A. Bar-Noy, P. Raghavan, B. Schieber, and H. Tamaki. Fast deflection routing for packets and worms. In *Proceedings of the Twelth Annual ACM Symposium on Principles of Distributed Computing*, pages 75–86, Ithaca, New York, USA, August 1993.

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

[7] I. Ben-Aroya, T. Eilam, and A. Schuster. Greedy hot-potato routing on the two-dimensional mesh. *Distributed Computing*, 9(1):3–19, 1995.

[8] 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.

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

[10] 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.

[11] A. Borodin and J. E. Hopcroft. Routing, merging, and sorting on parallel models of computation. *Journal of Computer and System Sciences*, 30(1):130–145, February 1985.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] C. Busch, M. Herlihy, and R. Wattenhofer. Randomized greedy hot-potato routing. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 458–466, January 2000.

[18] C. Busch, M. Herlihy, and R. Wattenhofer. Routing without flow control. In *Proceedings of the Thirteenth ACM Symposium on Parallel Algorithms and Architectures*, pages 11–20, July 2001.

[19] Costas Busch, Malik Magdon-Ismail, Marios Mavronicolas, and Paul Spirakis. Direct routing. In *Proceedings of the 12th Annual European Symposium on Algorithms ESA 2004*, Bergen, Norway, September 2004. To appear.

[20] Costas Busch, Malik Magdon-Ismail, Marios Mavronicolas, and Roger Wattenhofer. Greedy $\tilde{O}(c+d)$ hot-potato routing on trees. In *Proceedings of the 10th International Conference on Parallel and Distributed Computing (Euro-par)*, September 2004. To appear.

[21] Robert Cypher, Friedhelm Meyer auf der Heide, Christian Scheideler, and Berthold Vöcking. Universal algorithms for store-and-forward and wormhole routing. In *Proceedings of the 28th ACM Symp. on Theory of Computing (STOC)*, pages 356–365, 1996.

[22] 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.

[23] Uriel Feige. Nonmonotonic phenomena in packet routing. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 583–591, New York, May 1–4 1999. ACM Press.

[24] Ronald I. Greenberg and Hyeong-Cheol Oh. Universal wormhole routing. *IEEE Transactions on Parallel and Distributed Systems*, 8(3):254–262, 1997.

[25] B. Hajek. Bounds on evacutation time for deflection routing. *Distributed Computing*, 1:1–6, 1991.

[26] 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.

[27] M. Kaufmann, H. Lauer, and H. Schroder. Fast deterministic hot-potato routing on meshes. In Springer-Verlag, editor, *Proceedings of the 5th International Symposium on Algorithms and Computation (ISAAC), LNCS*, volume 834, pages 333–341, 1994.

[28] 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. (preliminary version appears in FOCS 1988).

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

[30] 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.

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

[32] I. Newman and A. Schuster. Hot-potato algorithms for permutation routing. *IEEE Transactions on Parallel and Distributed Systems*, 6(11):1168–1176, November 1995.

[33] 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.

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

[35] R. Prager. An algorithm for routing in hypercube networks. Master's thesis, University of Toronto, Computer Science Department, 1986.

[36] 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.

[37] Alan Roberts, Antonios Symvonis, and David R. Wood. Lower bounds for hot-potato permutation routing on trees. In M. Flammini, E. Nardelli, G. Proietti, and P. Spirakis, editors, *Proceedings of the 7th Int. Coll. Structural Information and Communication Complexity, SIROCCO*, pages 281–295. Carleton Scientific, 20–22 June 2000.

[38] C. Scheideler. *Universal Routing Strategies for Interconnection Networks*, volume 1390 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 1998.

[39] P. Spirakis and V. Triantafillou. Pure greedy hot-potato routing in the 2-D mesh with random destinations. *Parallel Processing Letters*, 7(3):249–258, September 1997.

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

[41] 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.