Greedy $\tilde{O}(C + D)$ Hot-Potato Routing on Trees

Costas Busch* Malik Magdon-Ismail† Marios Mavronicolas‡ Roger Wattenhofer§

July 11, 2003

Abstract

In *hot-potato (deflection) routing,* nodes in the network have no buffers for packets in transit. A hot-potato *routing algorithm* is *greedy* if packets are advanced from their sources toward their destinations whenever possible. The *dilation* D is the longest distance a packet has to travel; the *congestion* C is the maximum number of packets that traverse any edge. The *routing time* of a routing-algorithm is the time for the last packet to reach its destination. A well known lower bound on the routing time is $\Omega(C + D)$. When is it possible to design routing algorithms whose routing times match this lower bound?

Here, we address this fundamental question within the context of hot-potato routing for the specific case of a *tree* with n nodes. In particular, we present two greedy, hot-potato routing algorithms:

i. A deterministic algorithm, which has a routing time of $O((\delta \cdot C + D) \lg n)$, where δ is the maximum node degree; thus, for bounded degree trees, the routing time becomes $O((C + D) \lg n)$.

ii. A randomized algorithm which has a routing time of $O((C + D) \lg^2 n)$ with high probability, for any node degree. Randomization is used for adjusting packet priorities.

Both algorithms are *online* and simple yet efficient. They are built upon the idea of using *safe deflections*, which are deflections whose net effect is to "recycle" edges from one path to another. These are the *first* known hot-potato routing algorithms (whether greedy or not) for trees whose routing time is within logarithmic factors of the $\Omega(C + D)$ lower bound.

*Computer Science Department, RPI, 110 8th Street, Troy, NY 12180, USA. Email: buschc@cs.rpi.edu

†Computer Science Department, RPI, 110 8th Street, Troy, NY 12180, USA. Email: magdon@cs.rpi.edu

‡Computer Science Department, University of Cyprus, P. O. Box 20537, Nicosia CY-1678, Cyprus. Email: mavronic@ucy.ac.cy

§Computer Science Department, ETH Zurich, 8092 Zurich, Switzerland. Email: wattenhofer@inf.ethz.ch

1 Introduction

Packet routing is the general task of delivering a set of packets from their sources to their destinations. *Hot-potato* (or *deflection*) routing is relevant in networks whose nodes cannot buffer packets in transit – any packet that arrives at a node must immediately be forwarded to another node at the next time step, as if it were a "hot potato". A *routing algorithm* (or *protocol*) specifies at every time step the actions that each node takes while routing the packets. The *routing time* of the algorithm is the time at which the last packet is delivered to its destination. It is generally desirable for a routing algorithm to deliver all the packets to their destinations as quickly as possible.

Hot-potato routing was introduced by Baran [4], and since then, hot-potato routing algorithms have been observed to work well in practice [5]. They have been used in parallel machines such as the HEP multiprocessor [31], the Connection machine [17], and the Caltech Mosaic C [30], as well as in high speed communication networks [22]. Hot-potato routing is especially relevant in optical networks where it is difficult to buffer messages [1, 16, 22, 33, 35].

Here, we consider *tree* networks (acyclic connected graphs) in which each edge is a bi-directional link. Trees are important because many real-life networks are built upon them (for example, hierarchical infrastructures), which explains the interest that this type of routing problem has generated in the literature (see, for example, [2, 3, 20, 25, 27, 29, 34]). Furthermore, as articulated by Leighton [20], a spanning tree can be used to route packets in an arbitrary network. We consider trees in which nodes are *synchronous*, namely, a global clock defines a discrete time. At each time step t, a node may receive packets, which it forwards to adjacent nodes according to the routing algorithm. These packets reach the adjacent nodes at the next time step $t + 1$. At each time step, a node is allowed to send at most one packet per link.*

We consider *one-to-many* routing problems, in which each node is the source of at most one packet; however, each node may be the destination of multiple packets. For a routing problem in which packet paths have already been specified, there are two parameters that are useful for evaluating the performance of a routing algorithm: the *dilation* D, which is the maximum length of any path, and the *congestion* C, which is the maximum number of paths that use any link (in either direction). Since at most one packet can traverse an edge in a given direction at each time step, a trivial lower bound on the time needed to route all the packets to their destinations is $\Omega(C + D)$. It is desirable to design routing algorithms with routing time close to this lower bound. For *store-and-forward* routing, in which nodes have buffers for storing packets in transit, there are routing algorithms with performance close to the lower bound [7, 19, 21, 24, 26, 28]. However, such algorithms are not applicable when buffers are not available.

Here, we will assume that the packets are routed along the unique shortest path from their source to their destination.† We consider *greedy* hot potato routing. A routing algorithm is greedy if a packet always follows its path whenever this is possible. In hot-potato routing, a problem occurs if two or more packets appear at the same node at the same time, and all these packets have the same link as the next edge in their path. This constitutes a *conflict* between the packets because only one of them can follow that particular link. Since nodes have no buffers, the other packets will have to follow different links that deviate from their paths. We say that these packets are *deflected*. In a greedy algorithm, a packet π can be deflected only when another packet makes progress along the link that π wished to follow. If the number of deflections that a packet undergoes is small, the routing time will be close to optimal. Due to deflections, the path ultimately followed by a packet may not be the packet's original path; however, it will still contain the original path. We note that our $\tilde{O}(C + D)$ bounds hold with respect to the C and D of the *original* shortest paths; hence they are at most polylogarithmic factors away from the true optimal.

Contributions. We present two greedy hot-potato routing algorithms on trees. These are the *first* known hot-potato algorithms for trees. Greedy algorithms tend to be simple and easy to implement efficiently (as are the ones we present). Further, when the network traffic is low, and there are not many conflicts, greedy algorithms will not unnecessarily delay packets.

Both of our algorithms are *online*: at any time step, each node makes routing decisions based only on the packets it receives at that particular time step. We assume that each source node knows the path of the

*At any time step, at most two packets can traverse an edge in the tree, one packet along each direction of the edge.

†In a tree, any path between two nodes must contain the shortest path, so no other set of paths can have smaller congestion or dilation than the shortest paths.

packet it will inject into the network, the tree topology, and the congestion and the dilation of the routing problem; we emphasize, however, that it does not know the paths of other packets. The node then determines the time at which the packet will be injected. After the packet is injected, the packet is forwarded to its destination greedily. We give two hot-potato algorithms:

i. The algorithm Deterministic has routing time that is $O((\delta \cdot C + D)\lg n)$, where δ is the maximum node degree node in the tree. For bounded degree trees, the routing time is thus an $O((C + D)\lg n)$. All choices that a node makes in routing the packets can be done deterministically.

ii. The algorithm Randomized has routing time less than $\kappa(C + D)\lg^2 n$ with probability at least $1 - \frac{1}{n}$, where κ is a constant. Randomization is used when packets select priorities. These priorities are then used to resolve conflicts.

Note that for bounded-degree trees, the algorithm Deterministic guarantees a routing time that is within a logarithmic factor of optimal. The algorithm Randomized is only an additional logarithmic factor away from optimal; however, it remains so even for non-bounded degree trees.

Our algorithms are based on the idea of assigning *levels* to the nodes of the tree on the basis of *short-nodes*: a short-node r of a tree T with n nodes is a node such that if the tree were rooted at r, then each subtree contains at most $n/2$ nodes. Similarly, one can define short-nodes of r's subtrees, and so on. As we descend deeper into subtrees, the levels of the nodes increase. The level of a packet is the smallest level node that it crosses.

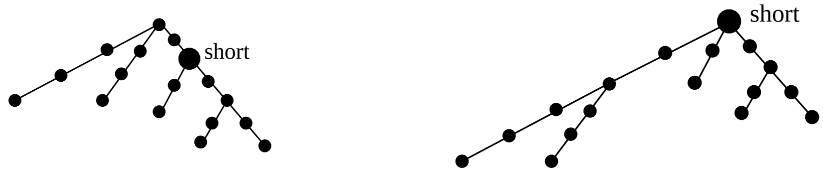
The general idea is that packets at different levels are routed in different phases. We show that there are at most $O(\lg n)$ such phases. In the algorithm Deterministic, each phase has a duration $O(C + D)$, while in the algorithm Randomized, in order to get a high probability result, we need to allow the phases to have duration $O((C+D)\lg n))$. Combining this with the bound on the number of phases then leads to our routing time bounds. The heart of both of our algorithms lies in the use of *safe* deflections, in which packets are only deflected onto edges used by other packets that moved forward in the previous time step. We observe that if all deflections are safe, then the congestion in the network can never increase as a result of deflections.

Related Work. Hot-potato routing algorithms have been extensively studied for various multiprocessor architectures such as the 2-dimensional mesh and torus [6, 11, 13, 15, 18], the d-dimensional mesh [6, 9], the hypercube [10, 15], vertex symmetric networks [23], and leveled networks [8, 12]. For more details about multiprocessor architectures we suggest [20]. There are no known, efficient hot-potato algorithms for arbitrary networks.

Various routing models for trees have been considered. *Matching routing* on trees is considered in [2, 27, 34]; here, at each time step, a set of edges with disjoint endpoints is chosen, and then the packets at the endpoints of each selected edge are exchanged. All of the results in matching routing consider permutation routing problems and provide algorithms with routing time $O(n)$, where n is the number of packets. In [3, 14, 32], the *direct routing* model is considered on trees; here, an injection time schedule is computed such that the packets follow their paths without conflicts. Direct routing algorithms are *offline*, some central node has global information about the routing problem and computes the injection times of the packets; in contrast, hot-potato routing is online and relies on deflections. In [3, 32] direct routing algorithms with routing time $O(n)$ are given. In [14], a direct routing algorithm (on trees) with optimal $O(C+D)$ routing time is presented. Roberts *et al.* [29] consider greedy hot-potato routing and show that there exist permutation problems such that any greedy hot-potato algorithm requires $\Omega(n)$ routing time.

To our knowledge, our algorithms are the *first* hot-potato routing algorithms which consider the general *congestion* + *dilation* routing problem on trees. The only other *congestion* + *dilation* hot-potato algorithms known are for other network topologies (leveled networks [12] and vertex symmetric networks [23]). For store-and-forward routing, there has been an extensive research on obtaining optimal $O(C + D)$ routing algorithms for arbitrary networks [19, 21, 24, 26, 28].

Paper Outline. We introduce trees and hot-potato routing in Sections 2 and 3, respectively. In Section 4, we present the algorithm Deterministic and its routing time analysis. In Section 5, we do so for the algorithm Randomized. We end with some concluding remarks in Section 6.



Original tree   Tree “rooted” at the short node

Figure 1: The short node

2 Trees

A tree $T = (V, E)$ is a connected acyclic graph with $|V| = n$ and $|E| = n - 1$. The *degree* of node v is the number of nodes adjacent to v. Let $v \in V$; then, T induces a subgraph on $V - \{v\}$ which consists of a number (possibly zero) of connected components. Each such connected component is a *subtree of v in T*.[‡] If v is adjacent to K nodes in T, then there are k disjoint subtrees $T_1, \ldots, T_k$ of v, one for each node $v_i \in T_i$ that is adjacent to v. The *distance* from v to u, is the number of edges in the (unique) shortest path from v to u.

The main idea behind our algorithms is to look at the tree from the point of view of a short node (see Figure 1). A node v in the tree is *short* if every subtree of v contains at most $n/2$ nodes. At least one short node is guaranteed to exist; the algorithm Find-Short-Node (Algorithm 1), finds one in $O(n)$ time.

Algorithm: Find-Short-Node(tree T)
Input: A tree T with n nodes $v_1, \ldots, v_n$.
Output: A short node of T.
begin
    $r \leftarrow$ any arbitrary node of T;
    Let T^r be the rooted tree with root r. Using a standard preorder traversal on T^r, compute for every node v_i, the number of nodes in the subtree of T^r which is rooted at v_i;
    $X \leftarrow r$;
    while X *is not short* **do**
        Let T' be a subtree of X in T which contains more than $n/2$ nodes;
        Let X' be the node of T' which is adjacent to X (i.e., the “root” node of T');
        $X \leftarrow X'$;
    end
    return X;
end

Algorithm 1: Find-Short-Node

A tree T may have many short-nodes, however, algorithm Find-Short-Node returns a *unique* short-node, assuming that the start node r in the algorithm is chosen deterministically. So, from now on, we will consider the unique short-node that is computed by algorithm Find-Short-Node.

We now define (inductively) the *level* ℓ of a node, and the *inner-trees* of T as follows. The tree T is the only inner-tree at level $\ell = 0$. The only node at level $\ell = 0$ is the short node of T. Assume we have defined inner-trees up to level $\ell \geq 0$. Every connected component obtained from the inner-trees of level ℓ by removing the short nodes of these inner-trees at level ℓ is an inner-tree at level $\ell + 1$. The level $\ell + 1$ nodes are precisely the short nodes of the inner-trees at level $\ell + 1$.

It is clear that the above definition inductively defines the inner-trees at all levels; it correspondingly assigns a level to every node. The process is illustrated in Figure 2. We can easily construct an $O(n^2)$ procedure to determine the node levels and inner-trees of T at every level. Further, the following properties

[‡]Note that for unrooted trees which we consider here, a subtree of a node v originates from every adjacent node of v; in contrast, the convention for rooted trees is that a subtree of v is any tree rooted at a child of v.

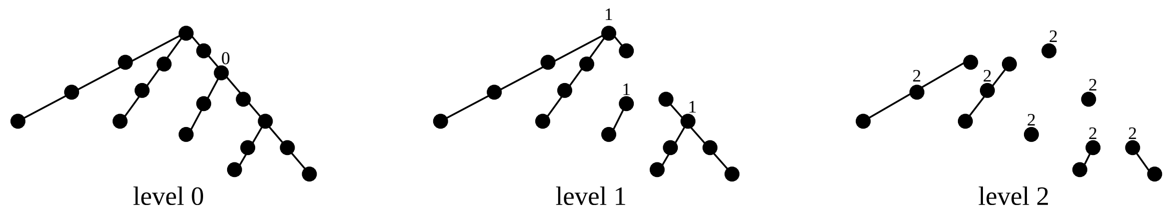


Figure 2: The process of constructing inner-trees at levels 0, 1 and 2

(which we state here without proof) hold: (i) every inner tree is a tree, (ii) the maximum level of any node and inner-tree is no more than $\lg n$, (iii) an inner-tree T' at level ℓ contains a unique node x at level ℓ, which is the short node of the inner-tree (we say that x is the inducing node of T'), (iv) any two inner-trees at the same level are disconnected, and (v) all nodes in a level-ℓ inner-tree other than the inducing node have a level that is smaller than ℓ.

3 Packets

Packet Paths. A *path* is any sequence of nodes $(v_1, v_2, \ldots, v_k)$. The length of the path is the number of edges in the path. The *original path* of a packet π is the shortest path from the source node of the packet to its destination node. Let ℓ be the minimum level of any node in the original path of π. Then, there is a unique node v with level ℓ in the path of π (since otherwise inner-trees of the same level would not be disconnected). Let T' be the inner-tree that v is inducing. The whole original path of π must be a subgraph of T' (from the definition of inner-trees). We say that the level of packet π is ℓ, and that the inner-tree of π is T'.

Assume now that packet π is injected into the network. At any time step t, the *current path* of a packet is the shortest path from the current node that the packet resides to its destination node. At the moment when the packet is injected, its current path is the same with the original path. While packet π is being routed to its destination, it may deviate from its original path due to deflections. However, the packet traverses at least once each edge of its original path before it reaches its destination.

We say that a packet moves *forward* if it follows the next link of its current path; otherwise, the packet is deflected. When the packet moves forward, its current path gets shorter by removing the edge that the packet follows. Any time that the packet is deflected, its current path grows by the edge on which the packet was deflected. Note that even with deflections, the current path of a packet is always the shortest path from the current node to the destination node.

Packet Routing and Deflections. In our algorithms, a packet remains in its source node until a particular time step, specified by the algorithms, at which the packet becomes *active*. When the packet becomes active, it is injected at the first time step that its first link on its original path is not used by any other packets that reside at the source node. We call such an injection a *canonical injection*.

After a packet is injected in the network, the packet moves forward to the destination. At each time step, each node in the network does the following: (i) the node receives packets from adjacent nodes, (ii) the node makes routing decisions, and (iii) according to these decisions, the node sends packets to adjacent nodes.

We say that two or more packets *meet* if they appear in the same node at the same time step. We say that two or more packets *conflict* if they appear in the same node at the same time and both wish to follow the same link forward. In a conflict, one of the packets will successfully follow the link, while the other packets must be deflected. In a greedy algorithm, a packet always attempts to follow a link forward unless it is deflected by another packet with which it conflicts for the same edge. The algorithms we consider here are greedy.

In our algorithms, packets are deflected in a particular way, which allows the congestion of the edges not to increase. Consider a node v at time step t. Let S_f denote the set of packets which appear in v at time step t and moved forward at time step $t-1$ toward v. Let E_f be the edges that the packets of S_f followed at time $t-1$. Let π be a packet that is deflected away from v at time t. Then, π attempts to be deflected on an edge of E_f. If this is not possible, then it is deflected on any other edge adjacent to v. Thus, π is not deflected on E_f only if other packets will use all the edges of E_f. We call this process of deflecting packets *canonical deflection*.

If π successfully follows an edge of E_f, then we say that the deflection of π is *safe*. We will show that in our algorithms, the deflections are always safe. Safe deflections have the following effect. Let e be the edge of E_f that π will be deflected on. Let σ be the packet of S_f that followed e at time step $t-1$. Edge e is transferred from the current path of σ to the current path of π; thus, the edges "recycle" from one path to another path. Next, we show that it is always possible to have safe deflections when packet injections and deflections are canonical.

Lemma 3.1 *If packet injections and deflections are canonical, then packet deflections are also safe.*

Proof: Let v be some node, and S the set of packets that will be routed from v at time step t. We write $S = S_f \cup S_d \cup S_i$, where S_f, S_d and S_i are disjoint sets such that: S_f are those packets which moved forward at time step $t-1$, in order to appear in v at time step t; S_d are those packets that were deflected at time step $t-1$; S_i are those packets which are injected at time step t in node v. Let E_f and E_d denote the respective set of edges, adjacent to v, which the packets of S_f and S_d followed at time step $t-1$. Clearly, $|S_f| = |E_f|$ and $|S_d| = |E_d|$; furthermore, since $S_f \cap S_d = \emptyset$, it must be $E_f \cap E_d = \emptyset$. Let S' denote the set of packets of S that will be deflected. We only need to show that the packets of S' follow edges of E_f.

We can write $S_f = S_1 \cup S_2 \cup S_3 \cup S_4$, where S_1 are packets that will move forward on edges of E_f, S_2 are packets that will move forward on edges of E_d, and S_3 are packets that will move forward on edges not in $E_f \cup E_d$, and S_4 are packets that will be deflected; sets S_1, S_2, S_3, S_4 are disjoint. Furthermore, we can write $S_d = S_5 \cup S_6$, where S_5 are packets of S_d that will move forward on edges of E_d and S_6 are packets that will be deflected; sets S_5 and S_6 are disjoint. Clearly, $S' = S_4 \cup S_6$.

For every packet of S_f which moves forward on an edge of E_d, a packet of S_d must be deflected. This implies that $|S_2| = |S_6|$. Let A be the set of edges of E_f that are not used by packets of S_1; in other words, A is the set of edges of E_f on which safe deflections can occur. We have that $|A| = |S_f| - |S_1|$. We also have that $|S'| = |S_4| + |S_6| = |S_4| + |S_2|$. Equivalently, $|S'| = |S_f| - |S_1| - |S_3|$. It follows that $|S'| \leq |A|$. Subsequently, all packets can be deflected on edges of E_f. It follows that all deflections are safe, as needed. ■

Consider some edge e. The congestion of edge e, denoted C_e^t, is the number of current paths that go through edge e at the beginning of time step t. Let $C^t = \max_{e \in E} C_e^t$, namely, C^t denotes the network congestion at time t. Note that $C = C^0$. Safe deflections imply that for any edge e and any time step t, C_e^t is no more than C_e^0, since edges are transferred from one current path to another one due to deflections, and the number of original paths crossing e is C_e^0. Therefore, from Lemma 3.1 we obtain:

Lemma 3.2 *If packets injections and deflections are canonical, then $C^t \leq C$, for any $t \geq 0$.*

Deflection Sequences. In the analysis of our algorithm Deterministic, we use a technique developed by Borodin *et al.* [9, Section 2], called "general charging scheme", with which they analyze deflection routing algorithms. Below, we adapt the discussion from [9, Section 2] so that it is appropriate for trees. Consider a packet π that was deflected at time t_1 by packet π_1. Define a *deflection sequence* and a *deflection path* with respect to this deflection as follows. Follow packet π_1 starting at time t_1 either to its destination or up to time $t_2 > t_1$, when it is deflected for the first time after t_1 by some packet π_2. Follow π_2 from time t_2 either to its destination or until some other time $t_3 > t_2$, when π_2 is deflected for the first time after t_2 by some packet π_3. Then follow packet π_3. Continue in the same manner until a packet π_j is followed to its destination. Define the sequence of packets: $\pi_1, \pi_2, \ldots, \pi_j$ as the deflection sequence of π at time t_1. Define the path that follows this sequence of packets from the point of deflection to the destination of π_j to be the deflection path. (See Figure 3.)

Claim 3.3 [9] *Suppose that for any deflection of packet π from node v to node u, the shortest path from node u to the destination of π_j (the last path in the deflection sequence) is at least as long as the deflection path. Then, π_j cannot be the last packet in any other deflection sequence of packet π.*

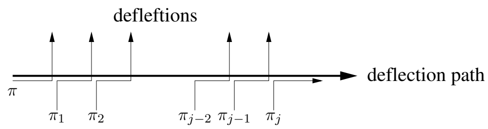


Figure 3: The deflection sequence $\pi_1, \pi_2, \ldots, \pi_j$.

Clearly, Claim 3.3 holds for greedy routing on trees. Claim 3.3 implies that we can "charge" the deflection of π to packet π_j, in the sense that when a packet is deflected another packet makes it to the destination. This implies the following corollary.

Corollary 3.4 [9] *If the deflection sequence for each of the deflections incurred by a routing algorithm satisfies the conditions of Claim 3.3, then the arrival time of each packet is bounded by $dist(\pi) + 2(k-1)$, where $dist(\pi)$ is the length of the shortest path from the source of packet π to its destination and k is the number of packets.*

4 A Deterministic Algorithm

Here we present the algorithm Deterministic (Algorithm 2). Each node is the source of at most one packet. Let v be a node which is the source of a packet π. In this algorithm, v first computes the level of the packet. Then according to the packet level, node v makes π active at a particular time step. The packet then moves greedily in the network until it is absorbed at its destination.

Algorithm: Deterministic

Input: A tree T of maximum node degree δ; A set of packets Π with path congestion C and dilation D; Each node is the source of one packet; Each node knows T, C, D;

Do for each packet π of level ℓ:

begin

Packet π gets active at time $\tau \cdot \ell$, where $\tau = 2(\delta \cdot C - 1) + D$;

The injection and deflections of packet π are canonical;

Packet π moves greedily to its destination;

end

Algorithm 2: Deterministic

Lemma 3.1 implies that all deflections are safe. We continue with the routing time analysis of the algorithm. Let m be the maximum level in T (note that $m \leq \lg n$). We divide time into consecutive phases $\phi_0, \phi_1, \ldots, \phi_m$, such that each phase consists of τ time steps. Write $\Pi = \Pi_0, \Pi_1, \ldots, \Pi_m$, where Π_i are packets of level i. From the algorithm, the packets of set Π_i become active at the first time step of phase ϕ_i. We will show that all packets of level i are absorbed during phase ϕ_i. In particular, we will show that the following invariants hold, where $i \geq 0$:

P_i: all packets of $\Pi_0 \cup \Pi_1 \cup \cdots \cup \Pi_i$ are absorbed by the end of phase ϕ_i.

In order to show that the properties P_i are indeed invariants, we will first show that the following properties hold, where $i \geq 0$, and P_{-1} is taken to be true by default:

Q_i: if P_{i-1} holds, then all packets of Π_i are absorbed by the end of phase ϕ_i.

Now, we will consider a particular level $\ell \geq 0$ and phase ϕ_ℓ. Assume that $P_{\ell-1}$ holds (namely, all packets of $\Pi_0 \cup \Pi_1 \cup \cdots \cup \Pi_{\ell-1}$ have been absorbed by the end of phase $\phi_{\ell-1}$). We will show that Q_ℓ holds; namely, we will show that all packets of Π_ℓ will be absorbed by the end of phase ϕ_ℓ. Notice that in phase ϕ_ℓ the only packets injected are those of Π_ℓ. So, from now on, we will consider phase ϕ_ℓ and only the packets Π_ℓ. We will show that each packet remains inside its inner-tree for the duration of ϕ_ℓ. (Note that an inner-tree can be connected with another inner-tree of lower level.)

Lemma 4.1 *During phase ϕ_ℓ, each packet of Π_ℓ remains inside its inner-tree.*

Proof: Assume for contradiction that some packet of Π_ℓ leaves its inner-tree during phase ϕ_ℓ. Let π be the first packet which leaves its inner-tree, and let t be the time step at which this event occurs. That is, at time step t, packet π appears in a node v which is not in its inner-tree, and at time step $t-1$, packet π was in a node u in its inner-tree. Thus, in node u and time $t-1$, packet π is deflected, since the destination of π is in its inner-tree. Since deflections are safe, there must be another packet σ that moved forward from node v to node u at time step $t-2$. Since inner-trees of the same level are disjoint, we have that packet σ left its inner-tree before packet π, a contradiction. ■

From Lemma 4.1, it follows that only packets of the same inner-tree meet with each other; thus, only packets of the same inner-tree may conflict with each other. From now on, we will consider only packets of some particular inner-tree T' of level ℓ, such that T' is induced by some node r of level ℓ. Next, we show that every packet with inner-tree T' will be absorbed in phase ϕ_ℓ.

Corollary 3.4, applies to Algorithm Deterministic. For any packet π, we have that $\text{dist}(p) \leq D$. Moreover, at the beginning of phase ϕ_ℓ, the number of packets in inner-tree T' does not exceed $\delta \cdot C$, since: (i) the original path of each packet of T' goes through node r, (ii) the degree of r is at most δ, and (iii) each edge adjacent to r has congestion $C^{\tau \cdot \ell} \leq C$ (a consequence of Lemma 3.2). Further, no more packets can be added in T' during phase ϕ_ℓ. Thus, from Corollary 3.4, all packets in inner-tree T' will be absorbed within a period of time $2(\delta \cdot C - 1) + D = \tau$. Subsequently, all packets of inner-tree T' are absorbed by the end of phase ϕ_ℓ. This implies that all packets of Π_ℓ are absorbed by the end of phase ϕ_ℓ. Therefore, we have the following lemma:

Lemma 4.2 *Q_ℓ holds for all $\ell \geq 0$.*

From the definition of P_ℓ, we have that P_ℓ holds if $Q_1, \ldots, Q_\ell$ hold, which is true from Lemma 4.2. Therefore, we obtain the following result:

Lemma 4.3 *P_ℓ holds for all $\ell \geq 0$.*

Lemma 4.3 implies that P_m holds. The fact that P_m holds further implies that all packets will be absorbed by the end of phase ϕ_m. Since $m \leq \lg n$, all packets are absorbed by time step $\tau \cdot (m+1)$ which is at most $(2(\delta \cdot C - 1) + D)(\lg n + 1)$. We have:

Theorem 4.4 *The routing time of algorithm* Deterministic *is bounded by $O((\delta \cdot C + D) \lg n)$.*

From Theorem 4.4, we obtain the following corollary.

Corollary 4.5 *If δ is bounded by a constant, then the routing time of algorithm* Deterministic *is bounded by $O((C + D) \lg n)$.*

5 A Randomized Algorithm

Here, we present the algorithm Randomized (Algorithm 3). The difference between Randomized and Deterministic is that the packets have now priorities. There are two levels of priority: low and high. At any time step, a packet is in one of these two priorities. The meaning of the priority is that in conflicts packets of high priority win over packets of low priority. Conflicts between packets of the same priority are resolved arbitrarily. Initially, when a packet becomes active, it is in the low priority. The packet has a chance to change its priority when conflicts occur. In a deflection, no matter what the previous priority was, the packet sets its priority to high with probability p (where p is specified in the algorithm), and sets its priority to low with probability $1 - p$. In the analysis, we will show that a packet in high priority has a good chance to reach its destination node without deflections.

Lemma 3.1 implies that all deflections are safe. We proceed with the routing time analysis of the algorithm. Let m be the maximum level in T (note that $m \leq \lg n$). We divide time into consecutive phases $\phi_0, \ldots, \phi_m$, and the packets into different sets $\Pi_0, \ldots, \Pi_m$, as we did in Section 4. We also consider

Algorithm: Randomized

Input: A tree T; A set of packets Π with path congestion C and dilation D; Each node is the source of one packet; Each node knows T, C, D;

Do for each packet π of level ℓ:

begin

Packet π gets active at time step $\tau \cdot \ell$, where $\tau = 16 \cdot (C + D) \cdot (2 \lg n + \lg \lg 2n) + 3D + 1$;

The injection and deflections of packet π are canonical;

Packet π moves greedily to its destination;

When packet π becomes active it has has low priority;

If π is deflected at time step t, then the next time step $t + 1$, the priority of π becomes high with probability $p = 1/(4(C+D))$, and low with probability $1-p$ (no matter what the previous priority was). The packet preserves the new priority until the next deflection;

end

Algorithm 3: Randomized

the properties P_i and Q_i, $0 \leq i \leq m$, defined in Section 4. We will show that properties P_i hold with high probability. In order to do this, we will first show that Q_i holds for any particular $i \geq 0$ with high probability.

Now, we consider a particular level $\ell \geq 0$ and phase ϕ_ℓ. Let $t_1, t_2, \ldots, t_\tau$ denote the time steps of phase ϕ_ℓ. Assume that $P_{\ell-1}$ holds (namely, all packets of $\Pi_0 \cup \Pi_1 \cup \cdots \cup \Pi_{\ell-1}$ have been absorbed by the end of phase $\phi_{\ell-1}$). We will show that Q_ℓ holds with high probability; namely, we will show that all packets of Π_ℓ will be absorbed by the end of phase ϕ_ℓ with high probability. Notice that in phase ϕ_ℓ the only packets injected are those of ϕ_ℓ. So, we will consider only the packets Π_ℓ. Notice that Lemma 4.1 holds. Thus, from now on, we will consider only packets of some particular inner-tree T' of level ℓ, such that T' is induced by some node r at level ℓ. We will show that every packet with inner-tree T' will be absorbed in phase ϕ_ℓ, with high probability. Let $T_1, T_2, \ldots, T_w$ denote the subtrees of r in T'. We first show some interesting properties about these subtrees.

Lemma 5.1 *Consider any subtree T_j, $1 \leq j \leq w$. The number of packets with destinations in T_j is at most C.*

Proof: Let e denote the edge that connects T_j with node r. At time step t_1, the packets that have destinations in T_j all have edge e in their original paths. Since the congestion upper bound is C, we have that the number of these packets is at most C. ■

Lemma 5.2 *Consider any time step t_i, $1 \leq i \leq \tau$, and any subtree T_j, $1 \leq j \leq w$. The number of packets that appear in T_j at time step t_i is at most C.*

Proof: Let A denote the set of packets with sources in T_j and B the set of packets with destinations in T_j. Let e be the edge that connects tree T_j with r. It must be that $|A| + |B| \leq C$, since all the packets in A and B have edge e on their original path, and the congestion does not exceed C.

Let X_i denote the set of packets which appear in T_j at time step t_i. We can write $X_i = Y_i \cup Z_i$, where Y_i are packets with destinations outside T_j, and Z_i are packets with destinations in T_j. We know that $Y_1 = A$. For $i > 1$, we can write $|Y_i| = |A| + a - b$, where a is the number of packets which entered T_j, and b is the number of packets which left T_j, between time steps t_1 and t_i, and all these packets have destinations outside T_j. Consider a packet π with destination outside T_j, which enters T_j in time step t_i (i.e. packet π traverses e at time step t_{i-1}). It must be that packet π has entered the network due to a deflection. Since deflections are safe, it must be that another packet $\sigma \in Y_{i-2}$ followed edge e forward at time step t_{i-2} (i.e. packet σ has its destination outside T_j). Thus, for any packet similar to π that enters T_j, there is another similar to σ the leaves T_j. This implies that $a \leq b$. Therefore, $|Y_i| \leq |A|$. Moreover, we know that $Z_i \subseteq B$. Which implies that $|X_i| = |Y_i| + |Z_i| \leq |A| + |B| \leq C$, as needed. ■

We define the *depth* of a node v, as the distance of the node from node r.

Lemma 5.3 *Consider any time step t_i, $1 \leq i \leq \tau$, and any subtree T_j, $1 \leq j \leq w$. At time step t_i, packets in subtree T_j appear in depth smaller or equal to D.*

Proof: Actually, we will show a stronger result: at time step t_i packets in subtree T_j appear in depth smaller or equal to D, and packets in level D are in *isolation*, that is, no more than one packet appears in the same node at depth D.

We prove the claim by induction on i. For the basis case, $i = 1$, the claim holds trivially true, since every node is the source of one packet and when the packets are injected they are in isolation at time step t_1; moreover, the original path dilation does not exceed D. Let's assume that the claim is true for any time step t_i, where $1 \leq i < k \leq \tau$. We will show that the claim is true for time step t_k. Note that the destination of any packet is at depth at most D (since the length of the original paths are at most D and all these paths cross node r). From the induction hypothesis, at time step t_{k-1}, all packets appear in depth D or smaller. Consider the packets at depth D and at time step t_{k-1} (note that these packets are in isolation, from the induction hypothesis). It must be that these packets wish to move at depth $D-1$, since none of them have reached their destinations, and all of them have their destinations in depth D or higher. All these packets successfully follow the links toward level $D-1$, and appear in depth $D-1$ at time step t_k. Therefore, at time step t_k, no packet will appear at depth higher than D. Moreover, at time step t_k the packets that appear in depth D can be only packets which appear in depth $D-1$ at time step t_{k-1} (since, from the induction hypothesis, there are no packets in depth $D+1$ at step t_{k-1}). These packets will appear in isolation in depth D at time step $t-1$, since each of these packets follows a different edge leading to depth D. Thus the claim holds for time step t_k, as needed. ∎

Let $R = [t_a, t_b]$, where $1 \leq a \leq b \leq \tau$, denote a time period containing time steps $t_a, t_{a+1}, \ldots, t_b$.

Lemma 5.4 *Consider a time period $R = [t_a, t_b]$, $1 \leq a \leq b \leq \tau$, and a tree T_j, $1 \leq j \leq w$. The number of different packets that appear in T_j during period R are at most $C + b - a$.*

Proof: From Lemma 5.2, we know that the number of packets that appear in T_j at time step t_a are at most C. At any subsequent time step, at most one new packet enters subtree T_j, which implies that during period R, the number of different packets that appear in T_j is at most $C + b - a$. ∎

We can bound the number of different packets that π may conflict with in a period as follows:

Lemma 5.5 *Consider a time period $R = [t_a, t_b]$, $1 \leq a \leq b \leq \tau$, in which a packet π is not deflected. During period R packet π may have conflicted with at most $2C + b - a$ different packets.*

Proof: Assume that at time step t_a, packet π is in subtree T_j and wishes to move to subtree T_k, where its destination resides, so that $k \neq j$. (If π has destination node r, or at time step t_a is either in r or T_k, then the analysis is similar.) Assume that packet π resides in subtree T_j for period $R' = [t_a, t_c]$, where $1 \leq a \leq c < b$. In order for π to conflict with some packet σ in T_j, it must be that packet σ resides in T_j during period R'. From Lemma 5.4, the number of packets similar to σ is at most $C + c - a \leq C + b - a$.

In time period $[t_{c+1}, t_b]$, packet π follows a path that includes the node r and a path in the subtree T_k. At the nodes of this path, packet π may conflict only with packets that have destinations in T_j. From Lemma 5.1, the number of these packets is at most C. Therefore, the total number of different packets that π may conflict with during period R is at most $2C + b - a$. ∎

Consider a time period $R = [t_a, t_b]$ in which packet π is not deflected. From Lemma 5.5, it follows that during period R, packet π may conflict with at most $2C + b - a$ packets. Let σ be any such packet. It is easy to see that σ will conflict at most once with π during period R (otherwise, packet π and σ would meet at two nodes at two different time steps during R, and this would imply that there are two different paths connecting the two nodes, which is impossible). Using this observation, we now prove:

Lemma 5.6 *Consider a time step t_i, where $1 \leq i \leq \tau - 2D$, at which packet π is in high priority. The probability that packet π reaches its destination in subsequent time steps without deflections is at least $1/2$.*

Proof: Let v be the node in which packet π resides at time step t_i. From Lemma 5.3, v appears in depth at most D. Note that the destination of v is a node which is at depth at most D (since the original paths have length at most D and cross node r). Hence, at time step t_i the current path of node v has length at most $2D$. So, consider time period $R = [t_i, t_{i+2D-1}]$. If during R packet π is not deflected (including time step t_{i+2D-1}), then it successfully reaches its destination node.

Since packet π is in high priority, it can be deflected only by other packets of high priority. We know that any other packet σ has only one chance to deflect packet π. This chance is given to packet σ with probability at most p: first packet σ gets deflected, then increases its priority with probability p, and then it is in a collision course with packet π. From Lemma 5.5, we have that the number of packets in similar situation to that of σ is at most $2C + i + 2D - 1 - i = 2C + 2D - 1 \leq 2(C + D)$. Therefore, packet π will be deflected by any of these packets with probability at most $2(C + D)p = 2(C + D)/(4(C + D)) = 1/2$. Thus, with probability at least $1/2$, no packet will deflect packet π. ■

Using Lemma 5.6, we obtain:

Lemma 5.7 *It a packet π gets deflected at time step t_i, $1 \leq i \leq \tau - 2D - 1$, then the probability that in subsequent time steps packet π reaches the destination node without deflections is at least $p/2$.*

Proof: After the packet is deflected at time step t_i, it becomes a high priority packet at time step t_{i+1} with probability p. From Lemma 5.6, we know that packet π is not deflected until it reaches its destination with probability at least $1/2$. Thus, after the deflection, packet π has a chance to reach its destination without deflections and in high priority with probability at least $p/2$. ■

From Lemma 5.7, we have that every time a packet is deflected, it has a chance to increase its priority and reach its destination without deflections. We next estimate how many times a packet gets deflected in a particular time period.

Lemma 5.8 *Consider a packet π which is in the network for the entire time period $R = [t_1, t_x]$, where $D \leq x \leq \tau$. Packet π gets deflected at least $(x - D)/2$ times in period R.*

Proof: Let a denote the number of times that π moves forward and b the number of times it is deflected, up to (and including) time step t_{x-1}. We have that $a + b = x - 1$. Every time that the packet moves forward its distance to the destination decreases, while every time it moves backward the distance increases. Let d_i denote the distance of π from its destination at time step t_i. We have that $d_x = d_1 - a + b$. Equivalently, $d_x = d_1 - x + 2b + 1$, which implies: $b = (d_x - d_0 + x - 1)/2$. We know that $d_x \geq 1$ (since π is in the network at time step t_x), and that $d_1 \leq D$, since in the original path of π the distance from its destination is at most D. Thus, $b \geq (x - D)/2$. ■

Next we compute the probability that packet π reaches its destination in phase ϕ_ℓ.

Lemma 5.9 *Packet π reaches its destination in phase ϕ_ℓ with probability at least $1 - 1/(n^2 \lg 2n)$.*

Proof: Consider the time period $R = [t_1, t_{\tau-2D-1}]$. Any deflection in that time period might increase the priority of packet π and then π may reach its destination without deflections. From Lemma 5.8, we have that π is deflected at least $x = (\tau - 2D - 1 - D)/2 = 8(C + D)(2 \lg n + \lg \lg 2n)$ times in period R. From Lemma 5.7, it follows that every time the packet is deflected in period R it has a chance to reach its destination with probability at least $p/2$. In other words, packet π fails to reach its destination in a deflection with probability at most $1 - p/2$. Therefore, π fails to reach its destination after x deflections with probability at most $(1 - p/2)^x$.[§] So,

$$\left(1 - \frac{p}{2}\right)^x = \left(1 - \frac{1}{8(C + D)}\right)^{8(C+D)(2\lg n + \lg\lg 2n)} \leq \frac{1}{e^{2\lg n + \lg\lg 2n}} = \frac{1}{n^2 \lg 2n}.$$

Thus, packet π reaches its destination in phase ϕ_ℓ with probability at least $1 - 1/(n^2 \lg 2n)$. ■

[§]Note that each deflection is treated as an independent event for reaching the destination node. We can do this because we have computed the $p/2$ lower bound for this probability for the worst possible scenario for each deflection. The consideration of the dependencies between deflections cannot possibly decrease the $p/2$ lower bound for each deflection.

Now, we consider all packets Π_ℓ in phase ϕ_ℓ.

Lemma 5.10 *The probability that all packets from the set* Π_ℓ *reach their destinations in phase* ϕ_ℓ *is at least* $1 - 1/(n \lg 2n)$.

Proof: From Lemma 5.9, any particular packet of Π_ℓ reaches its destination with probability at least $1 - 1/(n^2 \lg 2n)$. Thus, a packet will not reach its destination with probability at most $1/(n^2 \lg 2n)$. The number of packets in Π_ℓ is at most n (each node in the network injects at most one packet). By the union bound, the probability that one of these packets does not make it to the destination in phase ϕ_ℓ is at most $n \cdot 1/(n^2 \lg n) = 1/(n \lg 2n)$. Subsequently, all the packets make it to the destination with probability at least $1 - 1/(n \lg 2n)$, as needed. ■

From Lemma 5.10 we obtain the following corollary:

Corollary 5.11 *If* $P_{\ell-1}$ *holds, then* Q_ℓ *holds with probability at least* $1 - 1/(n \lg 2n)$*, for any particular* ℓ*,* $0 \le \ell \le m$.

We are now ready to show that properties P_ℓ hold with high probability:

Lemma 5.12 P_ℓ *holds with probability at least* $1 - (\ell + 1)/(n \lg 2n)$*, for any particular* ℓ*,* $0 \le \ell \le m$.

Proof: We will actually estimate the upper bound on the probability that P_ℓ fails. We have that P_ℓ fails if any of the $Q_0, Q_1, \ldots, Q_\ell$ fails. From Corollary 5.11, each of these properties fails with probability at most $1/(n \lg 2n)$. Thus, the probability that P_ℓ fails is at most $(\ell + 1)/(n \lg 2n)$. Therefore, P_ℓ holds with probability at least $1 - (\ell + 1)/(n \lg 2n)$. ■

From Lemma 5.12 and the fact that $m \le \lg n$, we obtain the following corollary:

Corollary 5.13 P_m *holds with probability at least* $1 - 1/n$.

Since $m \le \lg n$ and $\tau = O((C + D) \lg n)$, Corollary 5.13 implies that with probability at least $1 - 1/n$, all packets are absorbed by time step $\tau \cdot (m + 1) \le \kappa(C + D) \lg^2 n$, for some constant $\kappa \approx 33$. Thus we have:

Theorem 5.14 *With probability at least* $1-1/n$*, the routing time of* Randomized *is bounded by* $\kappa(C+D) \lg^2 n$*, for some constant* $\kappa > 0$.

6 Conclusions

We gave two hot-potato routing algorithms for trees. The deterministic algorithm is appropriate for trees whose degree is bounded by a constant and achieves routing time $O((C+D) \log n)$. The randomized algorithm is appropriate for arbitrary trees and achieves routing time $O((C + D) \log^2 n)$ with high probability. These are the *first* hot-potato algorithms known (greedy or non-greedy) whose routing time is within logarithmic factors from the $\Omega(C + D)$ lower bound. It still remains to close the gap between the $\Omega(C + D)$ lower bound and our upper bounds for trees.

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] 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.

[3] 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.

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

[5] 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 EUROPAR Conference*, pages 877–881. LNCS, 2000.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] C. Busch, M. Magdon-Ismail, M. Mavronicolas, and P. Spirakis. Direct routing. Submitted for publication to the ACM-SIAM Symposium on Discrete Algorithms (SODA'04).

[15] 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.

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

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

[18] 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.

[19] 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.

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

[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] M. Mitzenmacher. Constant time per edge is optimal on rooted tree networks. In *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 162–169, 1996.

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

[28] 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.

[29] 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.

[30] 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.

[31] 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.

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

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

[34] 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.

[35] 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.