Lower Bounds for One-to-one Packet Routing on Trees using Hot-Potato Algorithms

ALAN ROBERTS[1], ANTONIOS SYMVONIS[2] AND DAVID R. WOOD[3]

[1] *University of Sydney, Sydney NSW 2006, Australia*
[2] *Department of Mathematics, University of Ioannina, 45110 Ioannina, Greece*
[3] *School of Computer Science, Carleton University, Ottawa, Ontario, K1S 5B6, Canada*
Email: alanr@zip.com.au, symvonis@cc.uoi.gr, davidw@csc.carleton.ca

In this paper, we consider hot-potato packet routing of one-to-one routing patterns on n-node trees. By applying a 'charging argument', we show that any greedy hot-potato algorithm routes a one-to-one routing pattern within $2(n - 1)$ steps. On the other hand, a trivial lower bound suggests that at least $3n/2$ steps are required by any oblivious greedy algorithm. As the main contribution of the paper, we tighten the $2(n - 1)$ upper bound by constructing (for all sufficiently large n) an elaborate one-to-one packet routing problem on an n-node tree for which an oblivious greedy hot-potato algorithm requires at least $2n - o(n)$ steps. This improved lower bound is also shown to be valid for the *minimum-distance* heuristic. For trees of maximum degree d, we establish a lower bound of $2((d - 3)/(d - 2))n - o(n)$ routing steps.

Received 4 July 2001; revised 11 December 2001

1. INTRODUCTION

In a packet routing problem we are given a synchronous network represented by a connected undirected graph and a set of packets distributed over the nodes of the graph. Each packet has an *origin* and a *destination* node and the aim is to route each packet to its destination in as few steps as possible, subject to each edge carrying at most one packet in each direction at each time step. The distribution of the origins and destinations of packets specifies the *routing pattern*. In a *many-to-many* pattern each node may be the origin and destination of more than one packet. If each node is the origin of at most one packet then the routing pattern is called *many-to-one*. In a *one-to-many* routing pattern each node is the destination of at most one packet. If each node is the origin and destination of at most one packet the pattern is called *one-to-one*. A one-to-one pattern with the same number of packets as nodes is called a *permutation*.

Packet routing algorithms fall into two main categories, namely *on-line* and *off-line* algorithms. In *on-line routing*, routing decisions are made in a distributed manner at each node of the network. At each routing step, every node decides by which links to route the packets residing in it, depending on local information only, usually consisting of the origin and destination nodes of the packets residing in it. (More complicated on-line schemes where 'local knowledge' incorporates information accumulated in the node since the beginning of the routing can also be defined.) In *off-line routing*, a *routing schedule* which dictates how each packet moves during each step of the routing is precomputed. A routing schedule can be thought of as a collection of paths, each path corresponding to a particular packet and describing the route that the packet follows from its origin to its destination node.

In this paper we examine on-line one-to-one packet routing on trees under the *hot-potato* model. In a *hot-potato* (or *deflection*) routing algorithm there is no buffering of packets at nodes; that is, each packet must traverse a link at every step until it reaches its destination. This approach, introduced some 35 years ago by Baran [1], has been observed in a number of experiments to perform exceptionally well in practice [2, 3, 4, 5, 6, 7] and has been used in parallel machines such as the HEP multiprocessor [8], the Connection Machine [9] and the Caltech mosaic C [10]. The elimination of buffering queues used in store-and-forward algorithms has the advantage of potentially faster switching, which is particularly important for optical networks [5, 11, 12, 13], where buffering involves transforming the packets into electronic form.

In this paper we concentrate on greedy on-line hot-potato routing algorithms which at each step attempt to advance each packet towards its destination. If, at some time step t, a packet p moves away from its destination then we say p is *deflected*; otherwise we say p is *advanced*. If p is deflected and there is a packet q which is at the same node as p before step t and q is assigned a link whose end-point is closer to the destination of p, then we say that p is *deflected by q*. We formalize the notion of a greedy hot-potato algorithm as follows.

DEFINITION 1.1. *A hot-potato routing algorithm is said to be* greedy *if, whenever a packet p is deflected, all the links which would advance p towards its destination are used by other advancing packets.*

If at some time step, there is a link at some node which advances more than one packet residing at this node towards their respective destinations, then we say the packets are in *conflict*. We consider three types of greedy algorithms which differ with regard to their methods for resolving conflicts. We say a greedy hot-potato routing algorithm is *oblivious* if, for each conflict, the packet to traverse the link is chosen arbitrarily from those packets which wish to do so. The *minimum-distance* heuristic, proposed in [14, 15], chooses a packet with minimum distance to its destination to advance and in the *maximum-distance* heuristic a packet with maximum distance to its destination is chosen to advance.

Only recently has there been any precise analysis of the performance of greedy hot-potato algorithms [16, 17, 18, 19, 20, 21, 22]. Non-greedy hot-potato algorithms have appeared in [16, 23, 24, 25, 26, 27, 28] and lower bounds for hot-potato routing on meshes have been presented by Ben-Aroya *et al.* [29]. An important result, developed independently by Borodin *et al.* [19] and Feige [25], establishes an upper bound of $\mathrm{dist}(p) + 2(k-1)$ on the number of steps used by a greedy hot-potato algorithm to route a packet p on a wide class of networks including trees, where $\mathrm{dist}(p)$ is the shortest distance from the origin to the destination of p and k is the number of packets participating in the routing. However this result is not tight for one-to-one packet routing and for trees. The gap between the known lower bounds, the experimental results and the recent upper bounds motivate our analysis of the performance of greedy hot-potato algorithms on trees and in particular for one-to-one routing patterns. With these simpler cases, we might expect to gain tight bounds on the running time of a hot-potato algorithm.

Borodin *et al.* [19] also introduce the notion of a *totally greedy* hot-potato algorithm (referred as *maximum advance* by Feige [25]) which, at each time step, minimizes the possible number of deflected packets at each node. This involves solving a maximum matching problem between packets and 'good' links. Since, for trees, there is exactly one link which advances a packet towards its destination, a greedy algorithm on a tree is necessarily totally greedy. Feige [25] also introduced the class of *minimum advance* hot-potato algorithms which always advance at least one packet (for every node) towards its destination. Feige and Krauthgamer [30] proved that minimum advance hot-potato algorithms never livelock on trees.

Symvonis [31] developed an $O(n^2)$ time algorithm for determining a routing schedule for off-line permutation routing on trees. The routing is completed within $n-1$ steps, which is clearly optimal. Alstrup *et al.* [32] develop an algorithm for the same problem which delays each packet at its origin for some amount of time and then moves the packet directly towards its destination. The schedule is computed in $O(n \log n \log\log n)$ time and again the routing is completed within $n-1$ steps.

Packet routing on trees has also been studied under the *matching model* [33, 34, 35]. Here each node holds exactly one packet and the only operation allowed is the exchange of the packets at the end-points of an edge. Zhang [35] described an algorithm in the matching model for permutation routing on an n-node tree within $3n/2 + O(\log n)$ steps. Pantziou *et al.* [34] established a close relationship between the matching and hot-potato routing models that allows the application of tools for the analysis of hot-potato algorithms to the matching model. In particular, they present an on-line algorithm for many-to-many routing on trees under the matching model, which routes k packets within $d(k-1) + d \cdot \mathit{dist}$ steps, where d is the maximum degree of the tree and dist is the maximum distance from the origin to the destination of a packet. Their off-line algorithm solves the same problem within $2(k-1) + \mathit{dist}$ steps.

Our results. Based on the 'charging argument' [19, 25] and by utilizing the fact that a tree is a bipartite network, we show that any greedy hot-potato algorithm routes a one-to-one routing pattern on an n-node tree within $2(n-1)$ steps. On the other hand, a straightforward lower bound suggests that there are one-to-one routing problems requiring at least $3n/2$ steps by an oblivious greedy hot-potato algorithm.

A natural question which arises is how to close the gap between the $2(n-1)$ upper bound and the $3n/2$ lower bound. The main contribution of the paper establishes that the upper bound is optimal (within lower order terms). More specifically, for sufficiently large n, we construct an elaborate one-to-one packet routing problem on an n-node tree for which an oblivious greedy hot-potato routing algorithm requires at least $2n - o(n)$ steps. We also show that the same lower bound applies for the minimum distance heuristic.

A possible criticism of the trees used in the development of the above lower bounds is that some nodes have high degree. We therefore establish a lower bound of $2((d-3)/(d-2))n - o(n)$ on the number of routing steps under the minimum-distance heuristic (and thus, for an oblivious greedy algorithm) applied to an infinite family of n-node trees with maximum degree d.

The paper is organized as follows. In Section 2, we provide some simple lower bounds and observations which are used in the remainder of the paper. We also show that any greedy hot-potato algorithm will route a one-to-one pattern on an n-node tree within $2(n-1)$ steps. Section 3 presents the lower bound of $2n-o(n)$ on the number of steps required by an oblivious greedy hot-potato algorithm to route a one-to-one pattern on an n-node tree. In Section 4, we establish the same lower bound for the minimum-distance heuristic. We also establish a lower bound of $2((d-3)/(d-2))n-o(n)$ on the number of routing steps under the minimum-distance heuristic applied to an infinite family of n-node trees with maximum degree d. We conclude in Section 5.

2. PRELIMINARIES

We firstly make an observation concerning hot-potato routing on bipartite networks (for example, trees, meshes, hypercubes, etc.) which we shall exploit in our lower bounds and in the analysis of greedy hot-potato algorithms on trees. Suppose the nodes are coloured black and white such that adjacent nodes receive different colours. We associate with each packet the colour of the node where it originates and say that packets with the same colour have the same *parity*. Since in a hot-potato algorithm each packet moves at every step, a black/white packet will be at a white/black node after an odd number of steps and at a black/white node after an even number of steps. Hence we have the following observation.

OBSERVATION 2.1. In a hot-potato routing algorithm on a bipartite network, conflicting packets have the same parity.

2.1. Introductory lower bounds

We now establish lower bounds for the number of routing steps required for a hot-potato algorithm to move packets out of certain subtrees within a larger tree. These introductory results are used to prove our main lower bounds in Sections 3 and 4. Consider the subtree shown in Figure 1 consisting of k leaves adjacent to a single node, with a packet at each leaf destined for some node outside of the subtree.

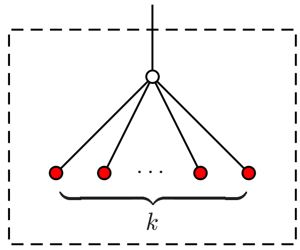

FIGURE 1. The subtree T with $k+1$ nodes and k leaves.

LEMMA 2.2. *Suppose the packets $p_1, p_2, \ldots, p_k$ are at the leaves of a subtree T with $k+1$ nodes and each packet p_i, $1 \leq i \leq k$, has a destination outside of T. Any hot-potato algorithm will take at least $2k$ steps for $p_1, p_2, \ldots, p_k$ to leave T.*

Proof. We proceed by induction on k. For $k = 1$ the sole packet will move to the non-leaf node in the first step and out of T in the second step. Assume the result holds for $k - 1$ packets. In the first step all of $p_1, p_2, \ldots, p_k$ will move to the non-leaf node and in the second step all but one of these packets will be deflected back to the leaf nodes. By induction, for the remaining $k - 1$ packets to leave T requires $2(k - 1)$ steps, so for $p_1, p_2, \ldots, p_k$ to leave T requires $2(k - 1) + 2 = 2k$ steps. □

We now use Lemma 2.2 to deduce the following lower bound.

LEMMA 2.3. *There is a permutation routing problem on an n-node tree for which the minimum-distance heuristic will take $3n/2$ steps and the maximum-distance heuristic will take n steps.*

Proof. Consider the tree B_n with $n/2$ nodes forming a path and $n/2$ leaves attached to one end of the path, as illustrated in Figure 2.

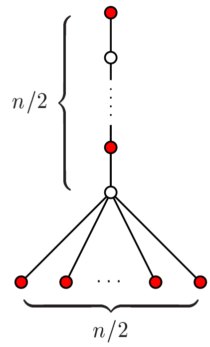

FIGURE 2. The tree B_n.

We define a permutation routing problem on B_n as follows. The packets which originate in the path have destinations in the leaves and the packets which originate in the leaves have destinations in the path. By Lemma 2.2 it will take $2(n/2) = n$ steps for all the packets in the leaves to enter the path. Under the minimum-distance heuristic the packet destined for the end of the path will be the last packet to enter the path and will take a further $n/2$ steps to complete the routing, hence a total of $3n/2$ steps. For the maximum-distance heuristic this packet will enter the path first and the total time will be n. □

By definition, the decisions of an oblivious algorithm regarding the packets which are advanced out of all those in conflict with each other are made arbitrarily. When proving lower bounds, this allows us to assume that an oblivious algorithm will make all the 'bad choices'. Since an oblivious algorithm can make exactly the same routing decisions as the minimum-distance heuristic, the lower bound for the minimum-distance heuristic implies a lower bound for an oblivious algorithm. Thus, we have the following.

COROLLARY 2.4. *There is a permutation routing problem on an n-node tree for which an oblivious greedy algorithm will take $3n/2$ steps.*

We now examine the performance of the hot-potato algorithm on complete d-ary trees. Consider a complete d-ary subtree of height h, with a packet at each node destined for some node outside of the subtree, as illustrated in Figure 3.

LEMMA 2.5. *Suppose a tree contains an n-node complete d-ary subtree ($d \geq 2$) of height h with a packet at each node of the subtree destined for some node outside of the subtree. Then the number of steps for a hot-potato routing algorithm to move all of the packets to outside of the subtree is*

$$2\left(\frac{dn}{d+1}\right) \text{ if } h \text{ is odd, and, } 2\left(\frac{dn+1}{d+1}\right) \text{ if } h \text{ is even.}$$

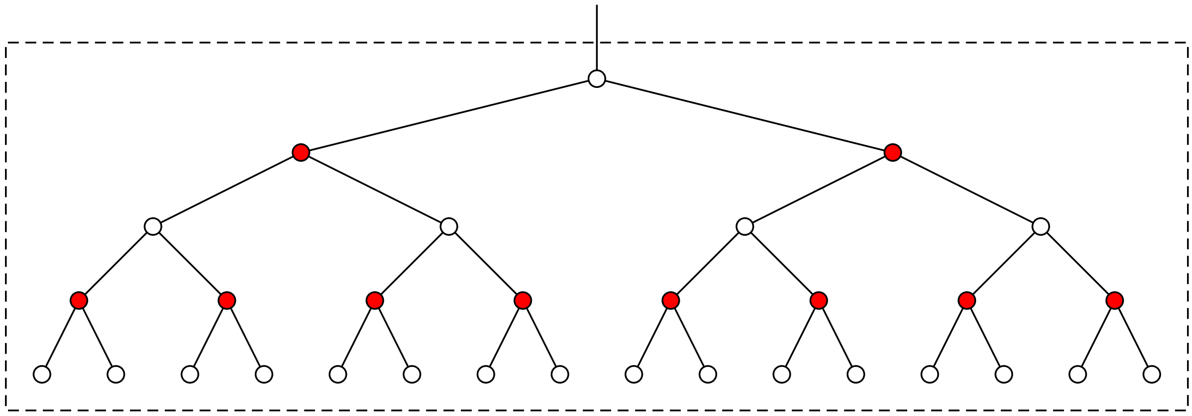

FIGURE 3. A complete binary subtree of height 4.

Proof. If the root node is coloured white (respectively, black) then after an odd number of steps a black (white) packet will be at the root and after an even number of steps a white (black) packet will be at the root. Thus on alternate steps black/white packets depart the subtree. Suppose without loss of generality that the leaves of the subtree are coloured white. Then there will be more white packets than black packets. After all the black packets have departed from the tree (along with an equal number of white packets) the remaining white packets will depart from the tree on alternate steps. Thus the total number of steps is twice the number of white packets; that is, twice the number of white nodes.

The number of nodes in the complete d-ary tree of height h is

$$n = \sum_{i=0}^{h} d^i = \frac{d^{h+1} - 1}{d - 1}. \tag{1}$$

Suppose h is even. The number of white nodes is

$$\sum_{\substack{i=0 \\ \text{even}}}^{h} d^i = \sum_{i=0}^{h/2} (d^2)^i,$$

which is the number of nodes in the complete d^2-ary tree of height $h/2$, which by (1) evaluates to

$$\begin{aligned}\frac{(d^2)^{h/2+1} - 1}{d^2 - 1} &= \frac{d^{h+2} - 1}{d^2 - 1} \\ &= \frac{d(d^{h+1} - 1) + (d - 1)}{(d - 1)(d + 1)} \\ &= \frac{dn + 1}{d + 1}.\end{aligned}$$

The result follows for even h. Now suppose h is odd. The number of white nodes is

$$\sum_{\substack{i=1 \\ \text{odd}}}^{h} d^i = d \sum_{\substack{i=0 \\ \text{even}}}^{h-1} d^i = d \sum_{i=0}^{(h-1)/2} (d^2)^i,$$

which by (1) is

$$\begin{aligned}d\left(\frac{(d^2)^{(h-1)/2+1} - 1}{d^2 - 1}\right) &= \frac{d}{d+1}\left(\frac{d^{h+1} - 1}{d - 1}\right) \\ &= \frac{dn}{d+1}.\end{aligned}$$

The result follows for odd h and hence for all h. □

Note that, in a complete d-ary tree, the majority of the nodes are leaves. We therefore can obtain a lower bound on the number of routing steps even if all the packets originate at the leaves of the subtree.

LEMMA 2.6. *Suppose a tree contains an n-node complete d-ary subtree and each leaf of this tree contains one packet whose destination is outside of the subtree. Then a hot-potato routing algorithm will take at least*

$$2\left(\frac{(d-1)n + 1}{d}\right)$$

steps to move all of the packets to outside of the subtree.

Proof. Since the leaves have the same parity, only on alternate steps can packets originating at leaves be at the root. Hence the number of steps is at least twice the number of leaves. The number of leaves in a d-ary tree of height h is

$$\begin{aligned}d^h &= \frac{d^{h+1} - 1}{d} + \frac{1}{d} \\ &= \left(\frac{d-1}{d}\right)\left(\frac{d^{h+1} - 1}{d - 1}\right) + \frac{1}{d} \\ &= \frac{(d-1)n + 1}{d}.\end{aligned}$$

The result follows. □

2.2. Algorithms

We now apply Observation 2.1 in conjunction with the *charging* argument (as described by Borodin *et al.* [19]) to provide an upper bound on the number of routing steps of a greedy hot-potato routing algorithm on a tree. To aid understanding we repeat the important details from that paper. Suppose p is a packet which is deflected by the packet p_1. Follow packet p_1 until it reaches its destination or it is deflected by packet p_2, whichever happens first. In the latter case, follow packet p_2 until it reaches its destination or it is deflected by packet p_3 and so on. We continue in this manner until we follow a packet p_l which reaches its destination. The sequence of packets $p_1, p_2, \ldots, p_l$ is defined to be the *deflection sequence* corresponding to the original deflection of packet p. The path (starting from the deflection node and ending at the destination of p_l) which is defined by the deflection sequence is said to be the *deflection path* corresponding to the deflection of packet p. Note that, for a particular packet p, we can define as many deflection sequences (paths) as the number of deflections p suffers during the course of its routing.

LEMMA 2.7. (Borodin *et al.* [19]) *Suppose that for any deflection of a packet p from node v to node u the shortest path from u to the destination of p_l (the last packet in the deflection sequence) is at least as long as the deflection path. Then, p_l cannot be the last packet in any other deflection sequence of packet p. Consequently we can 'charge' the deflection to packet p_l.*

This result is useful in the analysis of greedy hot-potato algorithms, as we now demonstrate in the case of trees.

THEOREM 2.8. *A greedy hot-potato algorithm will route a one-to-one pattern on an n-node tree within $2(n-1)$ steps.*

Proof. For an arbitrary packet p we denote by $\mathrm{defl}(p)$ the number of times that p is deflected before reaching its destination and by $\mathrm{dist}(p)$ the distance from the origin of p to its destination. Clearly p will reach its destination in exactly $2 \cdot \mathrm{defl}(p) + \mathrm{dist}(p)$ steps.

We now establish an upper bound on $\mathrm{defl}(p)$. Let p be a fixed packet originating at a node v, which we assume without loss of generality to be coloured white. According to Definition 1.1, in any deflection of p to a node u, the shortest path from u to the destination of the last packet in the corresponding deflection sequence is at least as long as the deflection path. Therefore, by Lemma 2.7, each deflection of p can be charged to a distinct packet.

Clearly there are at least $\lceil \mathrm{dist}(p)/2 \rceil$ black nodes in the tree and thus there are at most $n - 1 - \lceil \mathrm{dist}(p)/2 \rceil$ white nodes in the tree besides v. By Observation 2.1, only packets which originate at white nodes can deflect p. Hence

$$\mathrm{defl}(p) \leq n - 1 - \left\lceil \frac{\mathrm{dist}(p)}{2} \right\rceil .$$

Thus the number of steps for p to reach its destination is at most

$$2\left(n - 1 - \left\lceil \frac{\mathrm{dist}(p)}{2} \right\rceil\right) + \mathrm{dist}(p) \leq 2(n-1).$$

□

Note that there is a well-known (non-greedy) hot-potato algorithm (see [16]) for many-to-many packet routing on an arbitrary network which, in the case of trees, also attains an upper bound of $2(n-1)$. For an arbitrary interconnection network represented by a graph G, construct the directed graph G' with node set $V(G') = V(G)$ and arc set $A(G') = \{\overrightarrow{vw}, \overrightarrow{wv} : vw \in E(G)\}$. Every node of G' has equal in-degree and out-degree, so G' has an Eulerian tour (see, for example, [36]). Route the packets by following the Eulerian tour, assigning at most one packet to each outgoing arc. Once a packet reaches its destination it is consumed. The tour has length $2|E(G)|$, so the maximum number of time steps for a packet to reach its destination is $2|E(G)|$. Hence this algorithm on an n-node tree terminates within $2(n-1)$ time steps.

Consider a many-to-many routing pattern defined on an arbitrary n-node tree such that for every vertex v the number of packets originating at v is the degree of v and all packets are destined for some leaf node s. At most one packet can be consumed at each step and, since there are $2n-3$ packets not originating at s, at least $2n-3$ steps are needed by any routing algorithm. Hence the above bound for many-to-many packet routing on trees is tight (up to the additive constant).

3. LOWER BOUND FOR AN OBLIVIOUS ALGORITHM

We now describe a one-to-one packet routing problem on a tree with n nodes which will provide a lower bound of $2n - o(n)$ for the number of routing steps. The problem is described by (a) the tree, (b) the routing pattern and (c) a conflict resolution strategy.

The tree used in the lower bound proof consists of several small subtrees which are attached to the nodes of a backbone (see Figure 4). A routing pattern consists of the specification of the destination of each packet. For the purposes of the lower bound proof, for some packets it is only necessary to specify the subtree that contains their destination nodes, while, for others, the specification of the precise destination node is required (packet destinations are indicated by arrows in Figure 4).

In order to prove a lower bound we need to specify a 'bad' conflict resolution strategy that results in long routing times. Given that in an oblivious algorithm conflicts are resolved in an arbitrary fashion, the algorithm is then free to choose this 'bad' resolution strategy as its way of resolving conflicts.

Proving that the routing will terminate after at least $2n - o(n)$ steps requires a lot of technical detail. The main idea is to divide the routing into time-disjoint phases and to inductively show that at the end of each phase there is a class of packets that have not passed a certain backbone node on their trip toward their destination. The detailed proof consisting of the tree construction, the routing pattern, the conflict resolution strategy and the analysis of the routing is given in the subsections that follow.

3.1. The tree construction

The tree T_k ($k \geq 2$), illustrated in Figure 4 with nodes coloured black and white, is defined as follows.

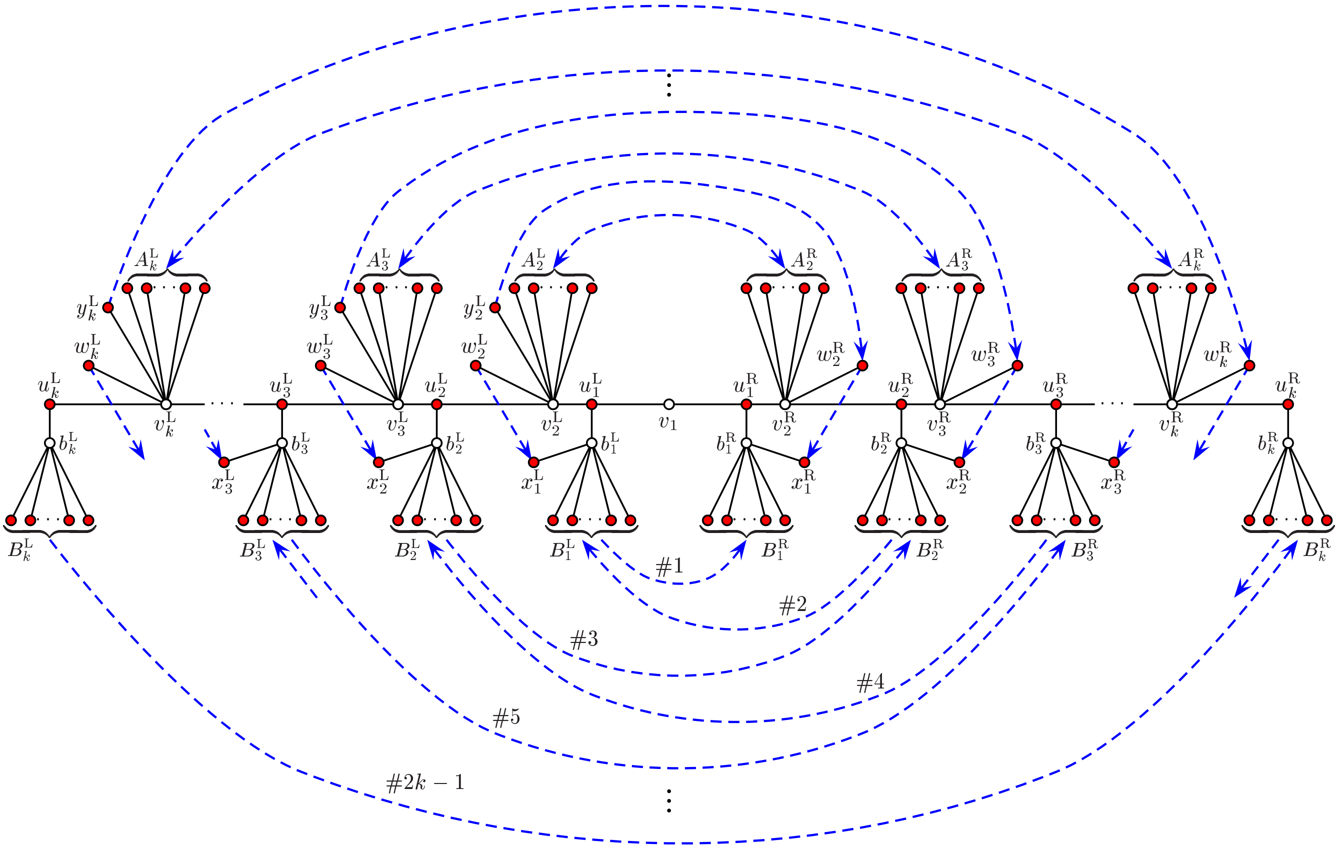

FIGURE 4. The tree T_k $(k \geq 2)$ with routing pattern and phases indicated.

- T_k contains a path called the *backbone* consisting of the $4k - 1$ nodes

$$(u_k^L, v_k^L, u_{k-1}^L, v_{k-1}^L, \ldots, u_2^L, v_2^L, u_1^L,$$
$$v_1, u_1^R, v_2^R, u_2^R, v_3^R, u_3^R, \ldots, v_k^R, u_k^R),$$

  where 'L' and 'R' refer to the left- and right-hand sides of the tree, respectively. The u_i-nodes are coloured black, the v_i-nodes are coloured white and node v_1 is considered to be the *root* of the tree.
- For each i, $2 \leq i \leq k$, T_k contains a set A_i^L of $4k$ black nodes and black nodes y_i^L and w_i^L, all adjacent to v_i^L; and a set A_i^R of $4k$ black nodes and a black node w_i^R, all adjacent to v_i^R.
- For each i, $1 \leq i \leq k$, T_k contains a set B_i^L of $4k^2$ black nodes each adjacent to a white node b_i^L which is adjacent to u_i^L; and a set B_i^R of $4k^2$ black nodes each adjacent to a white node b_i^R which is adjacent to u_i^R.
- For each i, $1 \leq i \leq k-1$, T_k contains a black node x_i^L adjacent to b_i^L and a black node x_i^R adjacent to b_i^R.

Clearly the number of nodes in T_k, denoted by n_k, is $8k^3 + O(k^2)$.

3.2. The routing pattern

We define the routing of packets as follows, as illustrated in Figure 4 by directed arcs.

- The packets originating in A_i^L are destined for the nodes in A_i^R $(2 \leq i \leq k)$.
- The packets originating in A_i^R are destined for the nodes in A_i^L $(2 \leq i \leq k)$.
- The packet originating at y_i^L is destined for the node w_i^R $(2 \leq i \leq k)$.
- The packet originating at w_i^R is destined for the node x_{i-1}^R $(2 \leq i \leq k)$.
- The packet originating at w_i^L is destined for the node x_{i-1}^L $(2 \leq i \leq k)$.
- The packets originating in B_i^R are destined for the nodes of B_{i-1}^L $(2 \leq i \leq k)$.
- The packets originating in B_i^L are destined for the nodes of B_i^R $(1 \leq i \leq k)$.

Since there is at most one packet originating and destined for each node, we have a one-to-one routing pattern. A packet which originates in some node in B_i^L is called at various times a B_i^L-packet, a B_i-packet, a B-packet and an i-packet, and similarly for packets originating in some B_i^R, A_i^L, A_i^R, y_i^L, w_i^L or w_i^R.

We say a B_i^L-packet *departs* when it first advances past v_i^L, an A_i^L-packet or a y_i^L-packet *departs* when it first advances past u_{i-1}^L, a B_i^R-packet *departs* when it first advances past v_i^R and an A_i^R-packet *departs* when it first advances past u_{i-1}^R. Note that we have defined departure nodes for all packets except the w_i-packets; these packets will firstly be blocked behind v_i and then will be blocked behind u_{i-1}. In some sense the w_i-packets depart twice.

We say a B_i^{L}-packet *arrives* when it first advances past u_i^{R}, a B_i^{R}-packet *arrives* when it first advances past u_{i-1}^{L}, an A_i^{L}-packet or a y_i^{L}-packet *arrives* when it first advances past v_i^{R} and an A_i^{R}-packet *arrives* when it first advances past v_i^{L}. Generally speaking, a packet arrives when it first leaves the backbone after crossing the root. Note that this is not necessarily the time step when the packet is consumed, although as we shall prove a packet will always be consumed shortly after it arrives.

3.3. The conflict resolution strategy

Recall that an oblivious greedy hot-potato algorithm resolves conflicts arbitrarily. Hence an adversary is free to substitute any strategies for resolving conflicts and for deflecting packets to produce a lower bound. The following scheme is designed so that, in general, those packets originating at nodes closer to the root (as drawn in Figure 4) have priority over packets originating at nodes further away.

1. For each i and j, $1 \leq i < j \leq k$, an i-packet has priority over a j-packet.
2. For each i, $2 \leq i \leq k$, amongst the i-packets the A_i-packets have highest priority, followed in decreasing order of priority by the y_i^{L}-packet, the w_i^{R}-packet, the B_i^{R}-packets, the w_i^{L}-packet and the B_i^{L}-packets.
3. Wherever possible, a deflected packet returns to the node where it came from in the previous step.

3.4. Analysis

We now establish some introductory results concerning the behaviour of the above-defined packet routing problem on T_k.

LEMMA 3.1. *For all i*, $2 \leq i \leq k$,

(a) The w_i^{L}-packet cannot advance past v_i^{L} until the y_i^{L}-packet has advanced past v_i^{L}.
(b) No B_i^{L}-packet can depart until the w_i^{L}-packet has advanced past v_i^{L}.

Proof. (a) Suppose the y_i^{L}-packet has not advanced beyond v_i^{L}. Since no packet is destined for y_i^{L}, up to this point the y_i^{L}-packet will have been deflected back to y_i^{L} in any conflict at v_i^{L}. Therefore, whenever the w_i^{L}-packet is at v_i^{L}, the y_i^{L}-packet will be in conflict with it. Since the y_i^{L}-packet has priority over the w_i^{L}-packet, the w_i^{L}-packet will not have advanced beyond v_i^{L}.

(b) Suppose the w_i^{L}-packet has not advanced beyond v_i^{L} (which is the departure-node for B_i^{L}-packets). Since no packet is destined for w_i^{L}, up to this point the w_i^{L}-packet will have been deflected back to w_i^{L} in any conflict at v_i^{L}. Therefore, whenever a B_i^{L}-packet is at v_i^{L}, the w_i^{L}-packet will be in conflict with it. Since the w_i^{L}-packet has priority over a B_i^{L}-packet, no B_i^{L}-packet will have advanced beyond v_i^{L}. □

We now prove our main lower bound for the tree T_k.

THEOREM 3.2. *A greedy hot-potato routing algorithm applied to the above routing pattern on the tree T_k, with the above strategy for resolving conflicts and deflecting packets, takes at least $2n_k - o(n_k)$ routing steps.*

Proof. We establish this result by defining phases for the routing corresponding to the movement of each set of B_i-packets. Since each B_i has $4k^2$ nodes and each A_i has only $4k$ nodes, the most significant part of the routing is the time taken to route the B-packets. We then show that these phases are disjoint. Applying Lemma 2.2, we conclude that each phase corresponding to the routing of a set of B_i-packets takes twice as many steps as there are nodes in B_i. The role of the A-packets is to 'fill-up' the backbone during the transition between phases.

For all j, $1 \leq j \leq k$, we define phase-$(2j-1)$ to be the time frame starting when the first B_j^{L}-packet departs through to when the last B_j^{L}-packet arrives. For all j, $2 \leq j \leq k$, phase-$(2j-2)$ commences when the first B_j^{R}-packet departs through to when the last B_j^{R}-packet arrives. Phase-i is indicated by '#i' in Figure 4.

Each phase is further subdivided into time frames, as illustrated in Figure 5, defined by when the first packet departs, when the first packet arrives, when the last packet departs and when the last packet arrives.

We proceed by induction on $j = 2, 3, \ldots, k$ with the following *induction hypothesis*.

1. Phase-$(2j-3)$ is completed before the start of phase-$(2j-2)$.
2. In phase-$(2j-2)$, the first B_j^{R}-packet arrives before the y_j^{L}-packet departs.
3. Phase-$(2j-2)$ is completed before the start of phase-$(2j-1)$.
4. In phase-$(2j-1)$, the first B_j^{L}-packet arrives before the last A_{j+1}^{R}-packet departs.

The induction basis. Let $j = 2$. Consider the left-hand side of T_k after the first step. For all $i \geq 1$, the B_i^{L}-packets will be at b_i^{L}, and the A_{i+1}^{L}-packets, the y_{i+1}^{L}-packet and the w_{i+1}^{L}-packet will have moved down to v_{i+1}^{L}. The A_{i+1}^{L}-packets have priority over the y_{i+1}^{L}-packet and the w_{i+1}^{L}-packet, so one of the A_{i+1}^{L}-packets will advance in the second step to u_i^{L}, while the remaining A_{i+1}^{L}-packets, the y_{i+1}^{L}-packet and the w_{i+1}^{L}-packet will be deflected back to their respective origins. Also at the second step, one of the B_i^{L}-packets will advance to u_i^{L} and the remainder will be deflected back to their respective origins. Hence there is one A_{i+1}^{L}-packet and one B_i^{L}-packet in conflict at u_i^{L} after two steps. The B_i^{L}-packet has priority over the A_{i+1}^{L}-packet so it will be advanced in the next

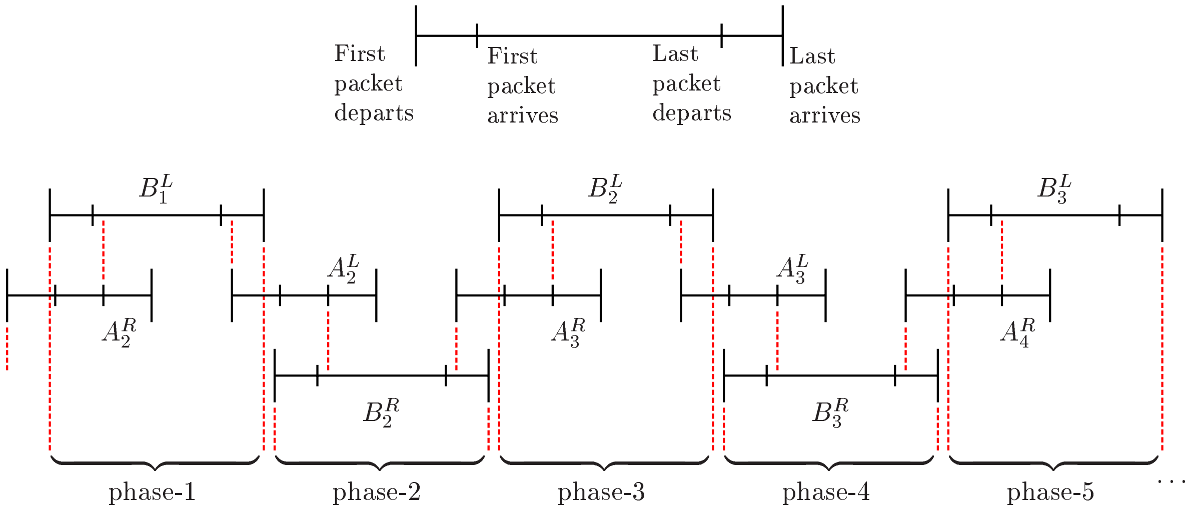

FIGURE 5. The time line for the routing.

step to v_i^{L} and the A_{i+1}^{L}-packet will be deflected back to v_{i+1}^{L}. Hence the B_1^{L}-packets are free to move across to the right-hand side two edges apart, while, for all $i \geq 2$, the A_i^{L}-packets are blocked behind u_{i-1}^{L}, the y_i^{L}-packet and the w_i^{L}-packet are blocked behind v_i^{L} and the B_i^{L}-packets are blocked behind v_i^{L} (see Lemma 3.1(b)).

Now, during the first two steps, the movement of packets in the right-hand side of T_k mirrors the movement of packets in the left-hand side (except that there are no B_1^{R}-packets and no y^{R}-packets). That is, for $i \geq 2$, one A_{i+1}^{R}-packet and one B_i^{R}-packet will be in conflict at u_i^{R} after two steps. As was the case on the left-hand side, the B_i^{R}-packet will be advanced in the next step to v_i^{R} and the A_{i+1}^{R}-packet will be deflected back to v_{i+1}^{R}. Since there are no B_1^{R}-packets, the A_2^{R}-packets will be free to move across to the left-hand side two edges apart.

After four steps, the leading B_1^{L}-packet will be at u_1^{R} and the leading A_2^{R}-packet will be at u_1^{L}. At the same time at u_1^{L}, there will also be a B_1^{L}-packet and an A_2^{L}-packet. The B_1^{L}-packet and the A_2^{L}-packet both wish to advance to v_1, while the A_2^{R}-packet wishes to advance to v_2^{L}. Since the A_2^{R}-packet is not in conflict with any other packets it will advance to v_2^{L} and, as discussed above, the B_1^{L}-packet will advance and the A_2^{L}-packet will be deflected. Now because the A_2^{R}-packet will be advancing to v_2^{L}, the A_2^{L}-packet must be deflected down to b_1^{L}. In the following step, this A_2^{L}-packet will be further deflected down to a B_1^{L} node (or to x_1^{L}) by an advancing B_1^{L}-packet and the leading A_2^{R}-packet will advance to its destination in A_2^{L}. This process continues, so that, for each A_2^{R}-packet, one A_2^{L}-packet is deflected down to B_1^{L}. (The A_2^{L}-packets can be thought to be 'making room' in A_2^{L} for the arriving packets.)

Now consider when the last A_2^{R}-packet departs. The w_2^{R}-packet will move to v_2^{R} where it will no longer be in conflict with an A_2^{R}-packet and hence will advance to u_1^{R} on the next step. Here it will be in conflict with B_1^{L}-packets moving down into B_1^{R}. Since B_1^{L}-packets have priority over the w_2^{R}-packet, the w_2^{R}-packet will be deflected back to v_2^{R}. Therefore, during phase-1 (that is, while B_1^{L}-packets move into B_1^{R}) the w_2^{R}-packet will be blocked at u_1^{R} until all of the B_1^{L}-packets have advanced past u_1^{R}.

Now consider when the last A_2^{R}-packet reaches its destination in A_2^{L}. Since there is the same number of A_2^{R}-packets as A_2^{L}-packets, all of the A_2^{L}-packets will have been deflected down into B_1^{L}. The y_2^{L}-packet will be at y_2^{L} and the w_2^{L}-packet will be at w_2^{L}. In the next step, the y_2^{L}-packet and the w_2^{L}-packet will both advance to v_2^{L}, where they will be in conflict. The y_2^{L}-packet has priority over the w_2^{L}-packet, so it will advance to u_1^{L} and the w_2^{L}-packet will be deflected back to B_2^{L} on the next step. At u_1^{L}, the y_2^{L}-packet will be in conflict with a departing B_1^{L}-packet. The B_1^{L}-packet has priority in this conflict, so the y_2^{L}-packet will be blocked behind u_1^{L} at least for the remainder of phase-1.

We have shown that during phase-1 (that is, while the B_1^{L}-packets move across the backbone) all other packets cannot depart. After the last B_1^{L}-packet departs, the A_2^{L}-packets will be free to depart, followed by the y_2^{L}-packet. Once this last B_1^{L}-packet arrives, thus marking the end of phase-1, the w_2^{R}-packet will move to x_1^{R} and will be consumed, thus freeing the B_2^{R}-packets to depart. This initiates the start of phase-2. Thus induction hypothesis (1) is satisfied for $j = 2$.

Since there are $4k \geq 8$ packets in A_2^{L} and the distance from u_1^{L} to A_2^{R} (the destination of A_2^{L}-packets) is 4, the first B_2^{R}-packet to depart will reach u_1^{L} before the y_2^{L}-packet has departed, hence induction hypothesis (2) is satisfied for $j = 2$. Once the y_2^{L}-packet has passed u_1^{L}, the w_2^{L}-packet will still not be able to pass u_1^{L} as the B_2^{R}-packets have priority over the w_2^{L}-packet in a conflict at u_1^{L}. Only once all of the B_2^{R}-packets have been consumed (that is, the end of phase-2) will the w_2^{L}-packet be free to move into x_1^{L}. The packets in B_2^{L} are now free to move along the backbone from left to right, thus marking the beginning of phase-3, so induction hypothesis (3) holds for $j = 2$.

The A_3^{R}-packets start to depart once the last B_2^{R}-packet has departed. Now the distance from u_2^{R} to u_1^{L} (the arrival node for B_2^{R}-packets) is $4 \leq 2k$ and A_3^{R}-packets move across the backbone two edges apart. Hence less than $2k$ of the packets from A_3^{R} will have departed when the last B_2^{R}-packet arrives (that is, the end of phase-2). In phase-3, while packets in B_2^{L} move left-to-right along the backbone, A_3^{R}-packets continue to move in the opposite direction. Since there are at least $2k$ remaining packets in A_3^{R}, the first packet of B_2^{L} arrives (at u_2^{R}) before the last A_3^{R}-packet departs (from u_2^{R}) . Hence induction hypothesis (4) holds for $j = 2$.

The induction step: We now show that the induction hypothesis holds for $j = i$ assuming that it holds for $j = i - 1$. By induction hypothesis (4) for $j = i - 1$, the first B_{j-1}^{L}-packet arrives before the last A_j^{R}-packet departs. Hence while B_{j-1}^{L}-packets move into B_{j-1}^{R} (phase-$(2j - 3)$), the w_j^{R}-packet is blocked behind u_{j-1}^{R} (since the B_{j-1}^{L}-packet has higher priority to the w_j^{R}-packet), which in turn blocks the B_j^{R}-packets from departing (since the w_j^{R}-packet has higher priority than a B_j^{R}-packet). Once phase-$(2j-3)$ is completed, the w_j^{R}-packet moves past u_{j-1}^{R} and down into x_{j-1}^{R} and thus B_j^{R}-packets are free to move across the backbone. This marks the beginning of phase-$(2j - 2)$. Hence induction hypothesis (1) holds for $j = i$.

After the last B_{j-1}^{L}-packet departs, the A_j^{L}-packets move across the backbone two edges apart. Since the distance from v_{j-1}^{L} (the departure node for B_{j-1}^{L}-packets) to u_{j-1}^{R} (the arrival node for B_{j-1}^{L}-packets) is at most $4k$ and the A_j^{L}-packets move across the backbone two edges apart, at most $2k$ packets from A_j^{L} will have departed when the last B_{j-1}^{L}-packet arrives. There are at least another $2k$ A_j^{L}-packets which begin to depart while the B_j^{R}-packets move across the backbone at the start of phase-$(2j - 2)$. Since the distance from v_{j-1}^{R} (the departure node for B_j^{R}-packets) to u_{j-1}^{L} (the arrival node for B_j^{R}-packets) is at most $4k$ and the B_j^{R}-packets move across the backbone two edges apart, when the first B_j^{R}-packet arrives, the y_j^{L}-packet will not have departed. By Lemma 3.1(a) the y_j^{L}-packet still blocks the w_j^{L}-packet from advancing past v_j^{L}. Hence induction hypothesis (2) holds for $j = i$.

Once the first B_j^{R}-packet arrives and throughout phase-$(2j - 2)$, the w_j^{L}-packet is still blocked behind u_{j-1}^{L} since the B_j^{R}-packets have priority over the w_j^{L}-packet. After the last B_j^{R}-packet arrives (that is, the end of phase-$(2j - 2)$), the w_j^{L}-packet moves down to its destination in x_{j-1}^{L} and B_j^{L}-packets are free to move across the backbone, thus beginning phase-$(2j - 1)$. Hence induction hypothesis (3) holds for $j = i$.

The A_{j+1}^{R}-packets start to depart once the last B_j^{R}-packet has departed. Since the distance from v_{j+1}^{R} (the departure node for A_{j+1}^{R}-packets) to v_{j+1}^{L} (the arrival node for A_{j+1}^{R}-packets) is at most $4k$ and the A_{j+1}^{R}-packets move across the backbone two edges apart, at most $2k$ of the A_{j+1}^{R}-packets will have departed when the last B_j^{R}-packet arrives (that is, the end of phase-$(2j - 2)$). This initiates the start of phase-$(2j - 1)$. While packets in B_j^{L} move left-to-right along the backbone, A_{j+1}^{R}-packets continue to move in the opposite direction. Since there are at least $2k$ remaining A_{j+1}^{R}-packets, the first B_j^{L}-packet arrives before the last A_{j+1}^{R}-packet departs. Hence induction hypothesis (4) holds for $j = i$.

By the induction principle, the induction hypothesis holds for all $j \leq k$. In the phase corresponding to the routing of say B_i^{L}-packets, by Lemma 2.2, at least twice as many steps are needed for the B_i^{L}-packets to depart as there are packets in B_i^{L}. Similarly for a set of B_i^{R}-packets. Hence each phase takes at least $2(4k^2) = 8k^2$ steps. Since there are $2k - 1$ phases, the total number of steps is at least $16k^3 - O(k^2)$. Since the number of nodes $n_k = 8k^3 + O(k^2)$, the total number of steps is at least $2n - O(k^2) = 2n_k - o(n_k)$. □

COROLLARY 3.3. *For every $n \geq n_2$, there exists a one-to-one routing pattern on an n-node tree such that an oblivious greedy hot-potato algorithm requires at least $2n-o(n)$ steps.*

Proof. Given n, choose k such that $n_k \leq n < n_{k+1}$. Consider the n-node tree constructed from T_k by appending a path of $n - n_k$ nodes to the end of the backbone. With the same routing pattern described above for T_k, by Theorem 3.2, at least $2n_k - o(n_k)$ steps are needed to complete the routing on this tree. Since $n < n_{k+1} = 8(k+1)^3 + O((k+1)^2) = 8k^3 + O(k^2)$ we have $n - n_k \leq O(k^2)$ and hence the number of steps required $2n_k - o(n_k) = 2n - o(n)$. □

Note that the size of the $o(n)$ term in our lower bound of $2n - o(n)$ can be reduced by having $4k^c$ nodes in B_i^{L} and B_i^{R} for some constant $c \geq 2$. The number of nodes is then $n_k = 8k^{c+1} + O(k^2)$ and the time taken is at least $16k^{c+1} = 2n_k - O(k^2) = 2n_k - O((n_k)^{2/(c+1)})$. For large c this lower bound tends to $2n_k$.

4. LOWER BOUNDS FOR THE MINIMUM-DISTANCE HEURISTIC

We now prove a lower bound of $2n - o(n)$ on the number of routing steps for the minimum-distance heuristic applied to n-node trees. To do so, we modify the construction described in the previous section so that essentially the same routing occurs when conflicts are resolved using the minimum-distance heuristic. Of course, a lower bound for the minimum-distance heuristic implies a lower bound for an oblivious algorithm. We describe separate lower bounds for ease of presentation.

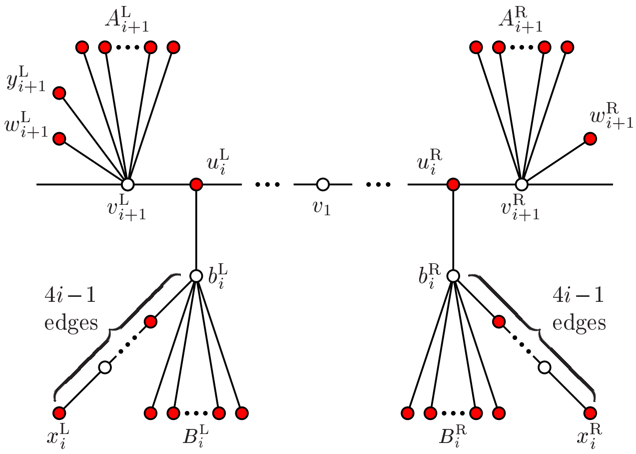

FIGURE 6. The tree T'_k.

We construct a tree T'_k from the tree T_k by a local replacement technique illustrated in Figure 6. In particular, for each i, $1 \le i \le k-1$, the edge from b^{L}_i to x^{L}_i is replaced by a path with $4i-1$ edges and similarly for the right-hand side. The same routing pattern used in the previous section is used for T'_k.

THEOREM 4.1. *For sufficiently large n, there exists a one-to-one routing pattern on an n-node tree, such that the minimum-distance heuristic requires at least $2n - o(n)$ steps.*

It is convenient to defer the proof of this result until later. We now establish lower bounds for the number of steps used by the minimum-distance heuristic applied to trees of bounded degree. To do so we replace the A_i and B_i sets of nodes used in the construction of T_k by complete d-ary trees for some constant d. In particular, we define a tree $T_{d,\alpha}$ which is parameterized by an even integer $d \ge 2$ and odd integer $\alpha \ge 3$. Let $k = d^\alpha/4$. Clearly $k \ge 2$ is an integer. k represents the number of times the basic construction is repeated along the backbone, as was the case previously. As shown in Figure 7, $T_{d,\alpha}$ is defined as follows.

- The backbone and the $u_i b_i$ edges are the same as in T_k.
- For each i, $1 \le i \le k$, there is a complete d-ary tree of height $2\alpha - 1$ rooted at b^{L}_i; the leaves of this tree are the nodes B^{L}_i. From b^{L}_i to x^{L}_i there is a path with $4i + 2\alpha - 3$ edges.
- For each i, $1 \le i \le k$, there is a complete d-ary tree of height $2\alpha - 1$ rooted at b^{R}_i; the leaves of this tree are the nodes B^{R}_i. From b^{R}_i to x^{R}_i there is a path with $4i + 2\alpha - 3$ edges.
- For each i, $2 \le i \le k$, there is a path with $\alpha - 1$ edges from v^{L}_i to a new node a^{L}_i and there is a complete d-ary tree of height α rooted at a^{L}_i; the leaves of this tree are the nodes A^{L}_i. From v^{L}_i there are paths each with $2\alpha - 1$ edges to w^{L}_i and to y^{L}_i.
- For each i, $2 \le i \le k$, there is a path with $\alpha - 1$ edges from v^{R}_i to a new node a^{R}_i and there is a complete d-ary tree of height α rooted at a^{R}_i; the leaves of this tree are the nodes A^{R}_i. From v^{R}_i there is a path with $2\alpha - 1$ edges to w^{R}_i.

We define the origin and destination of packets to be the same as with the tree T_k (and T'_k). Note that initially packets are only at the leaves of the complete d-ary subtrees in $T_{d,\alpha}$.

It is easily seen that the colouring of the nodes shown in Figure 7 is consistent; in particular, the A_i, B_i, y_i, w_i and x_i nodes are coloured black.

The number of nodes in the d-ary subtree of height α is

$$\frac{d^{\alpha+1}-1}{d-1} = \frac{4dk-1}{d-1}.$$

There are $d^\alpha = 4k$ leaves in the d-ary subtree of height α. Hence there are $4k$ nodes in each A_i, which is the same as in T_k (and in T'_k). The number of nodes in the d-ary subtree of height $2\alpha - 1$ is

$$\frac{d^{2\alpha}-1}{d-1} = \frac{(d^\alpha)^2-1}{d-1} = \frac{16k^2-1}{d-1}.$$

There are $d^{2\alpha-1} = (d^\alpha)^2/d = 16k^2/d$ leaves in the d-ary subtree of height $2\alpha - 1$. Hence there are $16k^2/d$ nodes in each B_i. The number of nodes in $T_{d,\alpha}$, denoted by $n_{d,\alpha}$, is

$$\begin{aligned} n_{d,\alpha} &= 2k\left(\frac{16k^2-1}{d-1}\right) + (2k-2)\left(\frac{4kd-1}{d-1}\right) + O(k^2) \\ &= \frac{32k^3}{d-1} + O\left(\frac{dk^2}{d-1}\right). \end{aligned} \tag{2}$$

THEOREM 4.2. *For every even constant $d \ge 2$ and odd $\alpha \ge 3$ the number of steps for the minimum-distance heuristic to route the above one-to-one routing pattern on the tree $T_{d,\alpha}$ is at least*

$$2\left(\frac{d-1}{d}\right)n_{d,\alpha} - o(n_{d,\alpha}).$$

Proof. Observe that the distances from B^{L}_i to u^{L}_i and from A^{L}_{i+1} to u^{L}_i are both 2α (and similarly on the right-hand side). Therefore, after 2α steps, the same pattern of conflicts will be initiated on $T_{d,\alpha}$ as on T_k. We now show that, for each conflict occurring in the tree T_k, the minimum-distance heuristic applied to $T_{d,\alpha}$ gives the same priority as the conflict resolution strategy employed for an oblivious algorithm applied to T_k.

For each conflict which occurs in T_k, Table 1 shows the node where the conflict occurs, the node the packets wish to move to, the destination of the packets involved and the distances to the respective destinations of the packets. We order the packets by non-decreasing distance, and, where the distances are equal, the conflict resolution strategy employed for an oblivious algorithm on T_k is employed again. It is easily verified that in each conflict the resulting priorities correspond precisely to the priorities which were specified under an oblivious algorithm. Therefore essentially the same routing of packets will occur on T_k

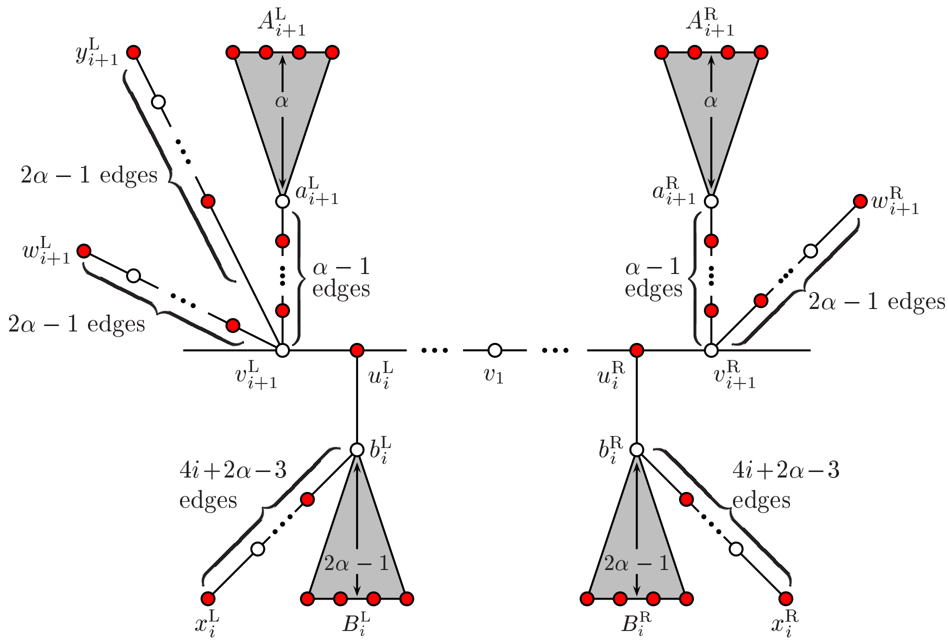

FIGURE 7. The tree $T_{d,\alpha}$.

TABLE 1. Distances to the destinations of packets in conflicts.

Conflict at	Move to	Packet	Destination	Distance	Priority
u_i^{L}	v_i^{L}	B_i^{L}	B_i^{R}	$4i+2\alpha-2$	First
		A_{i+1}^{L}	A_{i+1}^{R}	$4i+2\alpha-2$	Second
u_i^{L}	b_i^{L}	B_{i+1}^{R}	B_i^{L}	2α	First (since $i \geq 1$)
		w_{i+1}^{L}	x_i^{L}	$4i+2\alpha-2$	Second
u_i^{R}	b_i^{R}	B_i^{L}	B_i^{R}	2α	First (since $i \geq 1$)
		w_{i+1}^{R}	x_i^{R}	$4i+2\alpha-2$	Second
u_i^{R}	v_i^{R}	B_i^{R}	B_{i-1}^{L}	$4i+2\alpha-4$	First
		A_{i+1}^{R}	A_{i+1}^{L}	$4i+2\alpha-2$	Second
v_i^{L}	u_{i-1}^{L}	A_i^{L}	A_i^{R}	$4i+2\alpha-5$	First
		y_i^{L}	w_i^{R}	$4i+2\alpha-5$	Second
		w_i^{L}	x_{i-1}^{L}	$4i+2\alpha-5$	Third
		B_i^{L}	B_i^{R}	$4i+2\alpha-3$	Fourth
v_i^{R}	u_{i-1}^{R}	A_i^{R}	A_i^{L}	$4i+2\alpha-5$	First
		w_i^{R}	x_{i-1}^{R}	$4i+2\alpha-5$	Second
		B_i^{R}	B_{i-1}^{L}	$4i+2\alpha-5$	Third

under the specified conflict resolution strategy, as on $T_{d,\alpha}$ with the minimum-distance heuristic. In particular, the phases, as defined on T_k, will be disjoint.

As in Lemma 2.6 the time taken for each set of B_i-packets to depart is twice the number of B_i-packets; that is, each phase takes at least $2(16k^2/d) = 32k^2/d$ steps. The number of phases is $2k-1$, so the total number of routing steps is at least

$$\begin{aligned}\frac{(2k-1)32k^2}{d} &= \frac{64k^3}{d} - O\left(\frac{k^2}{d}\right)\\ &= \left(\frac{d-1}{d}\right)\left(\frac{64k^3}{d-1}\right) - O\left(\frac{k^2}{d}\right).\end{aligned}$$

By (2),

$$\frac{64k^3}{d-1} = 2n_{d,\alpha} - O\left(\frac{dk^2}{d-1}\right),$$

thus the total number of routing steps is at least

$$\frac{d-1}{d}\left(2n_{d,\alpha} - O\left(\frac{dk^2}{d-1}\right)\right) - O\left(\frac{k^2}{d}\right)$$
$$= 2\left(\frac{d-1}{d}\right)n_{d,\alpha} - O(k^2).$$

We now show that $k^2 = o(n_{d,\alpha})$. By (2)

$$\frac{k^2}{n_{d,\alpha}} < \frac{k^2(d-1)}{32k^3} < \frac{d}{32k} = \left(\frac{d}{32}\right)\left(\frac{4}{d^\alpha}\right) = \frac{d^{1-\alpha}}{8},$$

which tends to zero as $\alpha \to \infty$ (for constant d). Hence $k^2 = o(n_{d,\alpha})$ and the total number of routing steps is at least

$$2\left(\frac{d-1}{d}\right)n_{d,\alpha} - o(n_{d,\alpha}). \qquad \square$$

Since the maximum degree of $T_{d,\alpha}$ is $d+2$, we have the following result.

COROLLARY 4.3. *For all $d \geq 4$, there exists an infinite family of one-to-one packet routing problems on n-node trees with maximum degree d for which the minimum-distance heuristic takes at least $2((d-3)/(d-2))n - o(n)$ steps.*

We now show that Theorem 4.1 follows from Theorem 4.2.

Proof of Theorem 4.1. It is easily verified that all distances on T'_k are the same as on $T_{d,\alpha}$ with $\alpha = 1$. Hence Table 1 with $\alpha = 1$ describes the distances to the destinations of packets involved in each conflict on T'_k. Therefore the same routing of packets will occur on T'_k with the minimum-distance heuristic as on T_k under the specified conflict resolution strategy. In particular, the phases, as defined for an oblivious algorithm on T_k, will be disjoint. It is easily seen that the tree T'_k has $8k^3 + O(k^2)$ nodes, so if T'_k has n'_k nodes, by Theorem 3.2, at least $2n'_k - o(n'_k)$ steps are required to route the specified pattern. For an arbitrary $n \geq n'_2$, as in Corollary 3.3, we choose the maximum k such that $n \geq n'_k$ and add a path with $n - n'_k$ nodes to the end of the backbone of T'_k. Applying the same argument as in Corollary 3.3 it follows that the minimum-distance heuristic requires at least $2n - o(n)$ steps to route the pattern. $\square$

5. CONCLUSION

In this paper we have established a tight bound of $2n - o(n)$ for the number of steps required for one-to-one packet routing on trees using an oblivious hot-potato routing algorithm and using the minimum-distance heuristic. For trees of maximum degree d we have shown a lower bound of $2((d-3)/(d-2))n - o(n)$ using the minimum-distance heuristic. For the maximum-distance heuristic we have a lower bound of n and an upper bound of $2(n-1)$. It is an open problem to close this gap in the bounds on the performance of the maximum-distance heuristic for one-to-one packet routing on trees.

ACKNOWLEDGEMENTS

The authors thank Michael Houle and Stefan Dobrev for helpful discussions.

The work presented in this paper was supported by the Australian Research Council Large Grant A49906214.

REFERENCES

[1] Baran, P. (1964) On distributed communication networks. *IEEE Trans. Commun. Syst.*, **12**, 1–9.

[2] Lawrie, D. M. and Padua, D. A. (1984) Analysis of message switching with shuffle-exchanges in multiprocessors. *IEEE Tutorial: Interconnection Networks*, pp. 341–348. IEEE Computer Society Press, Los Alamitos, CA.

[3] Borodin, A. and Hopcroft, J. E. (1985) Routing, merging, and sorting on parallel models of computation. *J. Comput. Syst. Sci.*, **30**, 130–145.

[4] Maxemchuk, N. F. (1989) Comparison of deflection and store-and-forward techniques in the Manhattan street and shuffle-exchange networks. In *Proc. IEEE INFOCOM '89*, Ottawa, Ontario, Canada, April 23–27, pp. 800–809. IEEE Computer Society Press, Los Alamitos, CA.

[5] Acampora, A. S. and Shah, S. I. A. (1992) Multihop lightwave networks; a comparison of store-and-forward and hot-potato routing. *IEEE Trans. Commun.*, **40**, 1082–1090.

[6] Greenberg, A. G. and Hajek, B. (1992) Deflection routing in hypercube networks. *IEEE Trans. Commun.*, **40**, 1070–1081.

[7] Greenberg, A. G. and Goodman, J. (1993) Sharp approximate models of deflection routing in mesh networks. *IEEE Trans. Commun.*, **41**, 210–223.

[8] Smith, B. J. (1981) Architecture and applications of the HEP multiprocessor computer. *Soc. Photocopti. Instrum. Eng.*, **298**, 241–248.

[9] Hillis, W. D. (1985) *The Connection Machine*. MIT Press, Cambridge, MA.

[10] Seitz, C. L. (1992) Mosaic C: an experimental, fine-grain multicomputer. In *Proc. Int. Conf. Celebrating the 25th Anniversary of INRIA*, Paris, France, December. *Lecture Notes in Computer Science*, **653**, 69–85. Springer, New York.

[11] Bononi, A., Forghieri, F. and Prucnal, P. R. (1993) Analysis of one-buffer deflection routing in ultra-fast optical mesh networks. In *Proc. IEEE INFOCOM '93*, San Francisco, CA, March 28–April 1, pp. 303–311. IEEE Computer Society Press, Los Alamitos, CA.

[12] Bononi, A. and Prucnal, P. R. (1994) New structures of the optical node in multihop transparent optical networks with deflection routing. In *Proc. IEEE INFOCOM '94*, Toronto, Ontario, Canada, June 12–16, pp. 415–422. IEEE Computer Society Press, Los Alamitos, CA.

[13] Feehrer, J. R. and Ramfelt, L. H. (1996) Packet synchronization for synchronous optical deflection-routed interconnection networks. *IEEE Trans. Parallel Distrib. Syst.*, **7**, 605–611.

[14] Hajek, B. (1991) Bounds on evacuation time for deflection routing. *Distrib. Comput.*, **5**, 1–6.

[15] Zhang, Z. and Acampora, A. S. (1991) Performance analysis of multihop lightwave networks with hot potato routing and distance-age-priorities. In *Proc. IEEE INFOCOM '89*, Bal Harbour, FL, April 7–11, pp. 1012–1021. IEEE Computer Society Press, Los Alamitos, CA.

[16] Feige, U. and Raghavan, P. (1992) Exact analysis of hot-potato routing. In *Proc. 33rd Ann. Symp. Foundations of Computer Science (FOCS'92)*, Pittsburgh, PA, October 24–27, pp. 553–562. IEEE Computer Society Press, Los Alamitos, CA.

[17] Ben-Aroya, I., Eilam, T. and Schuster, A. (1995) Greedy hot-potato routing on the two-dimensional mesh. *Distrib. Comput.*, **9**, 3–19.

[18] Ben-Aroya, I., Newman, I. and Schuster, A. (1997) Randomized single-target hot-potato routing. *J. Algorithms*, **23**, 101–120.

[19] Borodin, A., Rabani, Y. and Schieber, B. (1997) Deterministic many-to-many hot potato routing. *IEEE Trans. Parallel Distrib. Syst.*, **8**, 587–596.

[20] Ben-Dor, A., Halevi, S. and Schuster, A. (1998) Potential function analysis of greedy hot-potato routing. *Theory Comput. Syst.*, **31**, 41–61.

[21] Busch, C., Herlihy, M. and Wattenhofer, R. (2000) Randomized greedy hot-potato routing. In *Proc. 11th Ann. ACM-SIAM Symposium on Discrete Algorithms (SODA'00)*, San Francisco, CA, January 9–11, pp. 458–466. ACM/SIAM.

[22] Busch, C., Herlihy, M. and Wattenhofer, R. (2000) Hard-potato routing. In *Proc. 32nd Ann. ACM Symp. on the Theory of Computing (STOC'00)*, Portland, OR, May 21–23, pp. 278–285. ACM.

[23] Kaufmann, M., Lauer, H. and Schroeder, H. (1994) Fast deterministic hot-potato routing on processor arrays. In *Proc. 5th Int. Symp. on Algorithms and Computation (ISAAC'94)*, Beijing, August 25–27. *Lecture Notes in Computer Science*, **834**, 333–341. Springer, New York.

[24] Brassil, J. T. and Cruz, R. L. (1995) Bounds on maximum delay in networks with deflection routing. *IEEE Trans. Parallel Distrib. Sys.*, **6**, 724–732.

[25] Feige, U. (1995) Observations on hot potato routing. In *Proc. 3rd Israel Symp. on the Theory of Computing and Systems (Tel Aviv)*, pp. 30–39. IEEE Computer Society Press, Los Alamitos, CA.

[26] Meyer auf der Heide, F. and Westermann, M. (1995) Hot-potato routing on multi-dimensional tori. In Nagl, M. (ed.), *Proc. Graph-theoretic Concepts in Computer Science: 21st Int. Workshop (WG'95)*, Aachen, Germany, June 20–22. *Lecture Notes in Computer Science*, **1017**, 209–221. Springer, New York.

[27] Newman, I. and Schuster, A. (1995) Hot-potato algorithms for permutation routing. *IEEE Trans. Parallel Distrib. Syst.*, **6**, 1168–1176.

[28] Naor, J., Orda, A. and Rom, R. (1998) Scheduled hot-potato routing. *J. Graph Algor. Appl.*, **2**, 1–20.

[29] Ben-Aroya, I., Chinn, D. D. and Schuster, A. (1998) A lower bound for nearly minimal adaptive and hot potato algorithms. *Algorithmica*, **21**, 347–376.

[30] Feige, U. and Krauthgamer, R. (2000) Networks on which hot-potato routing does not livelock. *Distrib. Comput.*, **13**, 53–58.

[31] Symvonis, A. (1996) Routing on trees. *Inform. Process. Lett.*, **57**, 215–223.

[32] Alstrup, S., Holm, J., de Lichtenberg, K. and Thorup, M. (1998) Direct routing on trees. In *Proc. 9th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'98)*, San Francisco, CA, January 25–27, pp. 342–349.

[33] Alon, N., Chung, F. R. K. and Graham, R. L. (1994) Routing permutations on graphs via matchings. *SIAM J. Discrete Math.*, **7**, 513–530.

[34] Pantziou, G. E., Roberts, A. and Symvonis, A. (1997) Many-to-many routing on trees via matchings. *Theoret. Comput. Sci.*, **185**, 347–377.

[35] Zhang, L. (1999) Optimal bounds for matching routing on trees. *SIAM J. Discrete Math.*, **12**, 64–77.

[36] Even, S. (1979) *Graph Algorithms*. Computer Science Press, MD.