Lower Bounds for Hot-Potato Permutation Routing on Trees *

ALAN ROBERTS
The University of Sydney, Australia

ANTONIOS SYMVONIS
The University of Sydney, Australia

DAVID R. WOOD
The University of Sydney, Australia

Abstract

In this paper we consider hot-potato packet routing on trees. As a lower bound, for all sufficiently large n we construct a permutation routing problem on an n-node tree for which an oblivious greedy hot-potato algorithm requires at least $2n - o(n)$ time steps. This lower bound is also valid for the *minimum-distance* heuristic. Applying the *charging* argument of Borodin *et al.* [8] we establish that any greedy hot-potato algorithm routes a permutation on a tree within $2(n-1)$ steps.

Keywords

Hot-Potato, Packet Routing, Permutation, Tree, Lower Bound

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 packet origins and destinations 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 and destination of at most one packet the pattern is called a

* Supported by Australian Research Council Large Grant A49906214.
This is the corrected version of the paper in *Proc. 7th International Colloquium on Structural Information and Communication Complexity* (SIROCCO 2000), pp. 281-295, Carleton Scientific, 2000.

one-to-one pattern. 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" by the nodes of the network. At each routing step, every node decides which links to route the packets residing in it by, depending on local information only, usually consisting of the origin and destination nodes of the packets residing in it, and knowledge regarding the topology of the network. 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 permutation routing on trees under the *hot-potato* model. In a *hot-potato* (or *deflection*) routing algorithm there is no buffering of packets at nodes; i.e., each packet must traverse a link at every step until it reaches its destination. This approach, introduced some 35 years ago by Baran [3], has the advantage of potentially faster switching, while the elimination of buffering queues, which are used in store-and-forward algorithms, can reduce the cost of switching hardware. This is particularly important for optical networks (e.g., [1, 11]), where buffering involves transforming the packets into electronic form. In a hot-potato routing algorithm we must assume that for each node v, the number of packets which originate at v is at most the degree of v.

In this paper we concentrate on greedy 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*. If 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, we say that p is *deflected by* q. We formalise the notion of a greedy hot-potato algorithm as follows.

Definition 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. In the *minimum-distance* heuristic [14], a packet with minimum distance to its destination is chosen to advance, and in the *maximum-distance* heuristic, a packet with maximum distance to its destination is chosen to advance.

Hot-potato routing has been observed in a number of experiments to perform exceptionally well in practice (e.g., [1, 13]), however only recently has there been

any precise analysis of the performance of hot-potato algorithms [5, 6, 7, 8, 9, 10, 14, 15]. Lower bounds for hot-potato routing on meshes have been presented by Ben-Aroya *et al.* [4]. Borodin *et al.* [8] established an upper bound of $dist(p) + 2(k-1)$ for the number of steps required to route a packet p on a wide class of networks including trees, where $dist(p)$ 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 permutations 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 permutations. With these simpler cases, we might expect to gain tight bounds on the running time of a hot-potato algorithm.

Symvonis [16] 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.* [2] presented an $O(n \log n \log\log n)$ time 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. Again the routing is completed within $n-1$ steps.

This paper is organised as follows. In Section 2 we provide some simple lower bounds and observations which are used in the remainder of the paper. We also establish that any greedy hot-potato algorithm will route a permutation on an n-node tree within $2(n-1)$ steps. In Section 3 we present our main lower bound of $2n - o(n)$ for hot-potato routing on trees with an oblivious greedy algorithm, and with the minimum-distance heuristic. We conclude in Section 4 by discussing some open problems related to permutation routing on trees.

2 Preliminaries

We now make some elementary observations about hot-potato routing on trees.

Observation 1 *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 (see Figure 1(a)). 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. □

Lemma 1 *There is a permutation routing problem on an n-node tree for which the minimum-distance heuristic takes $3n/2$ steps, and the maximum-distance heuris-*

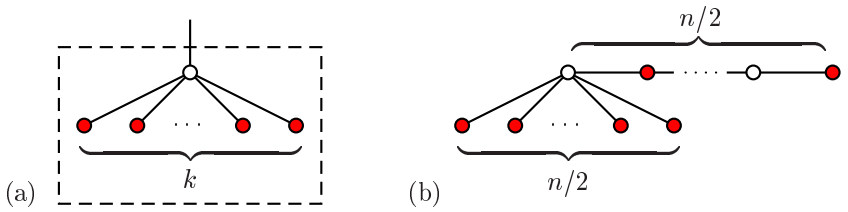

Figure 1: (a) The subtree T, (b) The tree B_n.

tics takes 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 1(b). 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 Observation 1 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, and 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. □

We now make an observation concerning hot-potato routing on bipartite networks (e.g., trees, meshes, hypercubes, etc.) which we shall exploit in the main lower bound and in the analysis of the greedy hot-potato algorithm on trees. Suppose the nodes are coloured black and white such that adjacent nodes receive different colours. We say packets which originate at nodes with the same colour have the same *parity*. Since in a hot-potato algorithm each packet moves at every step, a packet originating at a black/white node 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 *In a hot-potato routing algorithm on a bipartite network, conflicting packets must have the same parity.*

We now apply this observation in conjunction with the *charging* argument of Borodin *et al.* [8] to analyse the greedy hot-potato routing algorithm. To aid understanding we repeat the important details from this paper. Suppose p is a packet which is deflected at some time t_1 by the packet p_1. Follow packet p_1 until time t_2 where it reaches its destination or it is deflected by packet p_2, whichever happens first. In the latter case, follow packet p_2 until time t_3 where 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 at time t_{l+1}. The sequence of

packets $p_1, p_2, \ldots, p_l$ is defined to be the *deflection sequence* corresponding to the deflection of packet p at time t_1. 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 at time t_1.

Lemma 2 (Borodin *et al.* [8]) *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.*

Theorem 1 *A greedy hot-potato algorithm will route a permutation on an n-node tree within $2(n-1)$ steps.*

Proof. For an arbitrary packet p we denote by $\mathit{defl}(p)$ the number of times that p is deflected before reaching its destination, and by $\mathit{dist}(p)$ the distance from the origin of p to its destination. Then p will reach its destination in exactly $2 \cdot \mathit{defl}(p) + \mathit{dist}(p)$ steps. We now establish a bound on $\mathit{defl}(p)$. Assume without loss of generality that the origin of p is a white node. Clearly there are at least $\lceil \mathit{dist}(p)/2 \rceil$ black nodes in the tree. At each black node v there is a packet which originates at v which cannot deflect p (see Observation 2). According to Lemma 2, each deflection of p can be charged to a distinct packet. Hence

$$\mathit{defl}(p) \leq n - 1 - \lceil \mathit{dist}(p)/2 \rceil \quad .$$

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

$$2\left(n - 1 - \lceil \mathit{dist}(p)/2 \rceil\right) + \mathit{dist}(p) \leq 2(n-1) \quad . \qquad \square$$

Note that there is a well-known (non-greedy) hot-potato algorithm (see e.g., [12]) for many-to-many packet routing on an arbitrary network G, which in the case of trees, also attains an upper bound of $2(n-1)$. The directed graph with two symmetric arcs for every edge of G has an Eulerian tour of length $2|E(G)|$. By routing the packets by following such an Eulerian tour, each packet reaches its destination within $2|E(G)|$ steps. So this algorithm on an n-node tree terminates within $2(n-1)$ time steps.

3 The Main Lower Bound

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

The Tree Construction

The tree T_k ($k \geq 2$), illustrated in Figure 2 with black and white nodes, consists of:

- 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 node v_1 is considered to be the *root* node.
- Sets of black nodes A_i^L and A_i^R, $2 \leq i \leq k$, each with $4k$ nodes adjacent to one white node attached to the backbone via a 2-path at v_i^L and v_i^R respectively.
- Black nodes w_i^L and w_i^R, $2 \leq i \leq k$, each connected via a 3-path to the backbone at v_i^L and v_i^R respectively.
- Sets of black nodes B_i^L and B_i^R, $1 \leq i \leq k$, each with $4k^2$ nodes attached to one white node, which is connected to the backbone at u_i^L and u_i^R respectively.
- Black nodes x_i^L and x_i^R, $1 \leq i \leq k-1$, each connected to the same white node as the B_i^L and B_i^R nodes respectively.

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

The Routing Pattern

The routing of packets, illustrated in Figure 2 by directed arcs, is defined as follows.

- Packets originating in A_i^L are destined for the nodes in A_i^R, $2 \leq i \leq k$.
- Packets originating in A_i^R are destined for the nodes in A_i^L, $3 \leq i \leq k$.
- Packets originating in B_i^L are destined for the nodes of B_i^R, $1 \leq i \leq k$.
- Packets originating in B_i^R are destined for the nodes of B_{i-1}^L, $2 \leq i \leq k$.
- 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, $2 \leq i \leq k$.

Clearly there is at most one packet originating and destined for each node. A packet whose origin and destination is specified above is said to be *dedicated*. Since a partial permutation can always be extended to a full permutation, there is an assignment of destinations to the so-called *undedicated* packets to complete the permutation. A dedicated packet which originates in a black node in some subtree B_i^L is called at various times a B_i^L*-packet*, a B_i*-packet* and a *B-packet*, and similarly for packets originating in some B_i^R, A_i^L, A_i^R, w_i^L or w_i^R. We say a B_i^L-packet *departs* when it first passes v_i^L, and an A_i^L-packet *departs* when it first passes u_{i-1}^L, and similarly for the right-hand side.

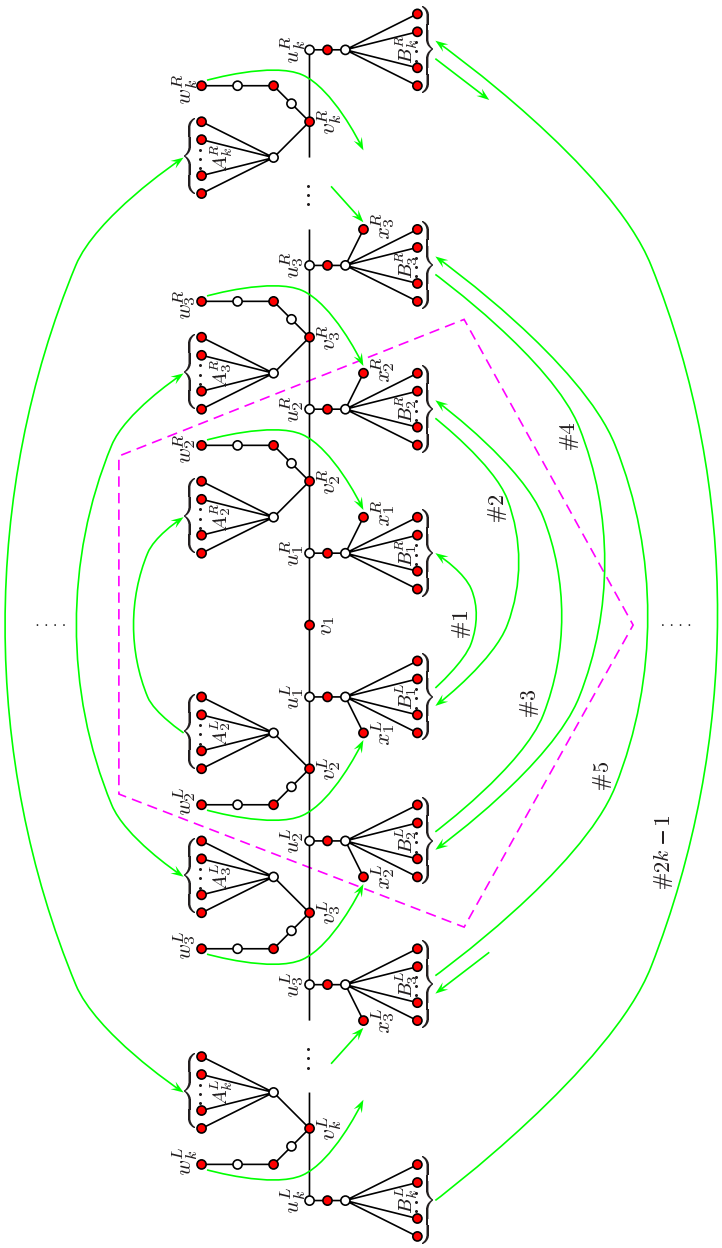

Figure 2: The tree T_k ($k \geq 2$) with routing pattern and phases indicated.

The Conflict Resolution Strategy

Recall that an oblivious greedy hot-potato algorithm resolves conflicts arbitrarily. Hence an adversary is free to substitute any conflict resolution strategy to produce a lower bound. The following strategy is designed so that, in general, those packets originating at nodes closer to the root (as drawn in Figure 2) have priority over packets originating at nodes further away, and that the w_i-packets block the B_i-packets from departing until the B_{i-1}-packets have been routed.

1. A dedicated packet has priority over an undedicated packet.
2. A B_i^R-packet has priority over the w_i^L-packet.
3. A B_i^L-packet has priority over the w_{i+1}^R-packet.
4. An A_i-packet has priority over a w_i-packet and over a B_i-packet.
5. The w_i^L-packet has priority over a B_i^L-packet.
6. The w_i^R-packet has priority over a B_i^R-packet.
7. An A_i-, B_i- or w_i-packet has priority over an A_{i+1}-, B_{i+1}- or w_{i+1}-packet.
8. We stipulate that a deflected packet returns to the node that it just came from.

It is tempting to define rules to govern potential conflicts between the remaining packets originating at black nodes, however in the following Lemma, we prove that such packets do not come into conflict.

Lemma 3 *If at some time point, for some j, $2 \leq j \leq k$, no B_j^L-packet has departed then for every i, $j < i \leq k$, no B_i^L-packet will have departed. Similarly, if no B_j^R-packet has departed then for every i, $j < i \leq k$, no B_i^R-packet will have departed.*

Proof. Whenever an A_{j+1}^L-packet reaches u_j^L there will be a B_j^L-packet in conflict with it, and by rule 7 the B_j^L-packet has priority, so no A_{j+1}^L-packet will pass u_j^L. Similarly, whenever a B_{j+1}^L-packet reaches v_{j+1}^L there will be an A_{j+1}^L-packet in conflict with it, and by rule 4 the A_{j+1}^L-packet has priority, so no B_{j+1}^L-packet will pass v_{j+1}^L. By induction, the result follows for the left-hand side. The proof for the right-hand side is identical. □

This result says that the B_i^L-packets are blocked behind v_i^L, at least until all the packets in B_{i-1}^L depart, and similarly for the right-hand side. We now state the main lower bound.

Theorem 2 *A greedy hot-potato routing algorithm applied to the above routing pattern on the tree T_k, with the above conflict resolution strategy, requires at least $2n_k - o(n_k)$ time 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 subtree B_i has $4k^2$ nodes and each subtree 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 Observation 1, 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.

We define *phase*-1 to be the time frame consisting of the start of the routing through to when the last B_1^L-packet is consumed. For all $j, 2 \leq j \leq k$, *phase*-$(2j-2)$ is the time frame starting when the first B_j^R-packet departs through to when the last B_j^R-packet is consumed. Similarly, *phase*-$(2j-1)$ commences when the first B_j^L-packet departs through to when the last B_j^L-packet is consumed. Phase-i is indicated by '#i' in Figure 2. Each phase is further sub-divided into time frames, as illustrated in Figure 3, defined by when the first packet departs, when the first packet is consumed, when the last packet departs, and when the last packet is consumed.

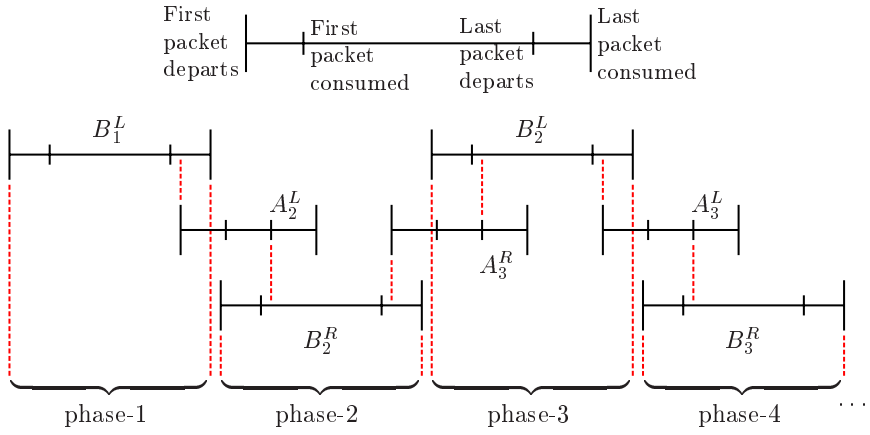

Figure 3: The time line for the routing.

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 is consumed before the last A_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 is consumed before the last A_{j+1}^R-packet departs.

The Induction Basis: Consider the case of $j = 2$. The tree T_2 is inside the dashed line in Figure 2.

Consider the movement of dedicated packets in the left-hand side of T_2 after three steps. (We can ignore the undedicated packets since they will lose in all conflicts and will return to the nodes that they came from.)

B_1^L-packets will move to u_1^L, A_2^L-packets will move down to the white node, the w_2^L-packet will move down to the white node, and the B_2^L-packets will move to u_2^L. After the second step, one B_1^L-packet will be at v_1, while one A_2^L-packet, one B_2^L-packet and the w_2^L-packet will be in conflict at v_2^L. (The remaining B_1^L, A_2^L and B_2^L-packets will have been deflected back to their origins.) By rule 4, the A_2^L-packet has priority over the other packets in this conflict, so it proceeds to u_1^L in the third step, where it will in turn be in conflict with a B_1^L-packet. By rule 7, B_1^L-packets have priority over A_2^L-packets, so no A_2^L-packet can depart until all of the B_1^L-packets have passed u_1^L. This implies in turn that the w_2^L- and B_2^L-packets will not pass v_2^L until all of the A_2^L-packets have departed.

Now consider the movement of dedicated packets in the right-hand side of T_2 after two steps. One B_2^R-packet will be in conflict at v_2^R with the w_2^R-packet. By rule 6, the w_2^R-packet has priority, so it will advance to u_1^R on the third step. At the same time the first B_1^L-packet to depart will arrive at u_1^R and will be in conflict with the w_2^R-packet. By rule 3, the B_1^L-packet has priority, so it will enter B_1^R first and will be consumed, followed by the remainder of the B_1^L-packets.

So during phase-1, i.e., while the B_1^L-packets move to B_1^R, all other packets cannot depart. Once the last B_1^L-packet passes u_1^L, the A_2^L-packets will be free to move along the backbone from left to right. Similarly, once this last packet from B_1^L is consumed, 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 move along the backbone from right to left. 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 5, the first B_2^R-packet to depart will reach u_1^L before the last A_2^L-packet has departed, hence induction hypothesis (2) is satisfied for $j = 2$. Once the last A_2^L-packet has passed u_1^L, the w_2^L-packet will still not be able to enter B_1^L as, by rule 2, 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 (i.e., 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. The distance from u_2^R to A_3^L (the destination of A_3^R-packets) is 9, so less than $2k$ of the packets from A_3^R will have departed when the last B_2^R-packet is consumed (i.e., the end of phase-2). As described above this initiates the start of 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 more than $2k$ remaining packets in

A_3^R, the first packet of B_2^L is consumed before the last A_3^R-packet passes 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 to depart is consumed 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 does not depart, which, by rule 6, in turn blocks the B_j^R-packets from departing. Once phase-$(2j-3)$ is completed, the w_j^R-packet moves into B_{j-1}^R, and B_j^R-packets are free to move across the backbone, thus beginning phase-$(2j-2)$. Hence induction hypothesis (1) holds for $j = i$.

When the last B_{j-1}^L-packet departs, the A_j^L-packets are free to cross the backbone. Since the distance from A_j^L to A_j^R is at most $4k$, by parity (see Observation 2) at most $2k$ packets from A_j^L will have departed when the last packet from B_{j-1}^L is consumed. At least another $2k$ packets leave A_j^L while the B_j^R-packets move across the backbone at the start of phase-$(2j-2)$, so when the first packet from B_j^R is consumed, A_j^L-packets still block the w_j^L-packet from passing v_j^L. Hence induction hypothesis (2) holds $j = i$.

Once the last B_j^R-packet is consumed (i.e., the end of phase-$(2j-2)$), the w_k^L-packet moves down to its destination in B_{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 a packet in A_{j+1}^R has to travel is less than $4k$, by parity (see Observation 2) at most $2k$ of the packets from A_{j+1}^R will have departed when the last B_j^R-packet is consumed (i.e., 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 packets in A_{j+1}^R, the first packet of B_j^L is consumed before the last A_{j+1}^R-packet passes u_j^R. 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 Observation 1, 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 $2n_k - o(n_k)$. □

Corollary 1 *For every $n \geq n_2$, there exists an instance of the permutation routing problem on an n-node tree such that the oblivious greedy hot-potato routing algorithm requires at least $2n - o(n)$ steps.*

Proof. 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 backbone. Construct a routing pattern as described above for T_k and extend this to a complete permutation. By Theorem 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 $2n_k - o(n_k) = 2n - o(n)$. □

3.1 Lower Bound for the Minimum-Distance Heuristic

In this section we prove a lower bound of $2n - o(n)$ for the minimum-distance heuristic. 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.

Firstly, we construct a tree T'_k from the tree T_k by a local replacement technique illustrated in Figure 4. In particular, for each i, $1 \leq i < k$, a path of length two is inserted between u_i^L and the white node connecting to the nodes in B_i^L, and the node x_i^L is placed at the end of a path of length $4i-2$. A similar local replacement is used for the right-hand side.

To obtain a lower bound for the minimum-distance heuristic, it is important to specify the destination of all packets in the permutation, instead of simply extending a partial permutation to a full permutation, as otherwise the introduced packets may interfere with the desired routing. We employ the same routing pattern used in the previous section for the A-, B- and w-packets. The routing of other packets is illustrated in Figure 4 by a directed arc from the origin to the destination of each packet. These additional packets will be immediately consumed or will be consumed after the first step of the routing. Also, we specify that the B_1^R-packets are destined for the B_k^L-nodes, and the A_2^R-packets are destined for the A_2^L-nodes, thus specifying a destination for all packets in the routing.

Theorem 3 *For sufficiently large n, there exists a permutation routing problem on an n-node tree, such that the minimum-distance heuristic requires at least $2n - o(n)$ steps.*

Proof. We now show that for each conflict occurring in the tree T'_k, the minimum-distance heuristic gives the same priority as the conflict resolution strategy employed for the oblivious algorithm in the previous section. We denote the shortest distance between two nodes s, t by $dist(s,t)$. Note that $dist(u_i^{L/R}, v_1) = 2i-1$, and $dist(v_i^{L/R}, v_1) = 2i-2$. As we shall prove, a number of conflicts will arise between packets which have equal distance to their respective destinations. In this case, the same conflict resolution strategy is employed as was used by the oblivious algorithm in the previous section.

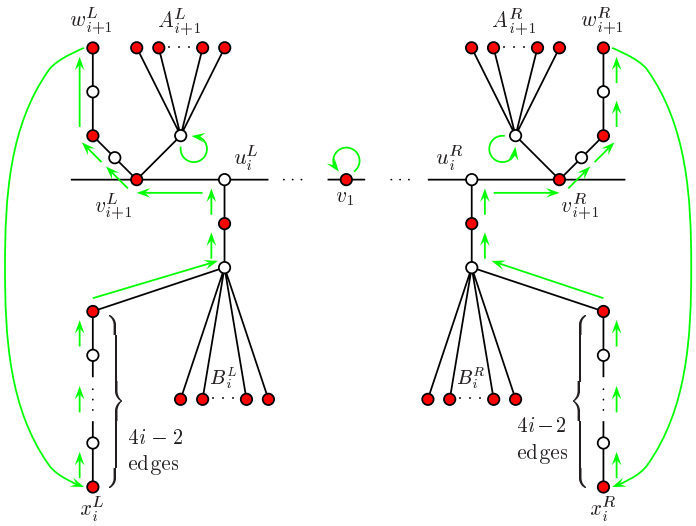

Figure 4: Modifying the tree T_k to produce T'_k.

Consider a conflict at u^L_i between a B^L_i-packet (destined for a B^R_i-node) and an A^L_{i+1}-packet (destined for an A^R_{i+1}-node). The distances in T'_k for both of these packets to their respective destinations is equal to $dist(u^L_i, v_1) + dist(v_1, u^R_i) + 3 = (2i-1) + (2i-1) + 3 = 4i+1$, so in this conflict we apply the same conflict resolution strategy as was employed for T_k.

Consider a conflict at v^L_i between a B^L_i-packet (destined for a B^R_i-node), an A^L_i-packet (destined for an A^R_i-node), and the w^L_i-packet (destined for the x^L_{i-1}-node). The distance for the B^L_i-packet is $dist(v^L_i, v_1) + dist(v_1, u^R_i) + 3 = (2i-2) + (2i-1) + 3 = 4i$, the distance for the A^L_i-packet is $dist(v^L_i, v_1) + dist(v_1, v^R_i) + 2 = (2i-2) + (2i-2) + 2 = 4i-2$, and the distance for w^L_i-packet is $dist(v^L_i, x^L_{i-1}) = 4(i-1) - 2 + 4 = 4i-2$. Hence, using the minimum-distance heuristic, the A^L_i-packet and the x^L_i-packet has priority over the B^L_i-packet, which was the case in conflict resolution strategy employed in the previous section.

Consider a conflict at u^R_i between a B^R_i-packet (destined for a B^L_{i-1}-node) and an A^R_{i+1}-packet (destined for an A^L_{i+1}-node). The distance for the B^R_i-packet is $dist(u^R_i, v_1) + dist(v_1, u^L_{i-1}) + 3 = (2i-1) + 2(i-1) - 1 + 3 = 4i-1$, and the distance for the A^R_{i+1}-packet is $dist(u^R_i, v_1) + dist(v_1, v^L_{i+1}) + 2 = (2i-1) + 2(i+1) - 2 + 2 = 4i+1$. Hence, using the minimum-distance heuristic, the A^R_{i+1}-packet has priority over the B^R_i-packet, which was the case in conflict resolution strategy employed in the previous section.

Consider a conflict at v^R_i between a B^R_i-packet (destined for a B^L_{i-1}-node), an A^R_i-packet (destined for an A^L_i-node), and the w^R_i-packet (destined for the x^R_{i-1}-

node). The distance for the B_i^R-packet is $dist(v_i^R, v_1) + dist(v_1, u_{i-1}^L) + 3 = (2i - 2) + 2(i-1) - 1 + 3 = 4i - 2$, the distance for the A_i^R-packet is $dist(v_i^R, v_1) + dist(v_1, v_i^L) + 2 = (2i-2) + (2i-2) + 2 = 4i - 2$, and the distance for the w_i^R-packet is $dist(v_1^R, x_{i-1}^R) = 4 + 4(i-1) - 2 = 4i - 2$. Since all the respective distance for the conflicting packets are equal the same conflict resolution strategy employed in the previous section is used by the minimum-distance heuristic.

After three steps of the routing, an A_2^R-packet (destined for A_2^L) and a B_1^R-packet (destined for B_k^L) will be in conflict at u_1^R. Clearly the A_2^R-packet has less distance to its destination so it will advance, followed by all of the A_2^R-packets. These packets will reach their destination during the first phase of the routing. In any conflict at u_1^R between a B_1^R-packet and a packet p passing through u_1^R, the distance to the destination of p will be less than the distance to B_k^L, so the B_1^R-packets will only pass u_1^R in a 'gap' in the movement of packets from right to left along the backbone. (Such a gap will occur whenever a w_i^R-packet crosses the backbone). Regardless, a B_1^R-packet travelling from right to left along the backbone will not affect the routing of any other packets.

No other conflicts occur in the routing on T_k or in T_k' so essentially the same routing of packets will occur on T_k under the specified conflict resolution strategy, as on T_k' with the minimum-distance heuristic. In particular, the phases, as defined in the previous section, will be disjoint. It is easily seen that the tree T_k' still has $8k^3 + O(k^2)$ nodes, so if T_k' has n_k' nodes, by Theorem 2, at least $2n_k' - o(n_k')$ steps are required to route the specified permutation.

For an arbitrary $n \geq n_2'$, as in Corollary 1, we choose k such that $n_k' \leq n < n_{k+1}'$, and add a path with $n - n_k'$ nodes to the end of the backbone of T_k'. We route the packets at these nodes to themselves. Applying the same argument as in Corollary 1 it follows that the the minimum-distance heuristic requires at least $2n - o(n)$ steps to route the permutation. □

4 Conclusion and Open Problems

In this paper we have established a tight bound of $2n - o(n)$ for the number of steps required for permutation routing on trees using an oblivious hot-potato algorithm and 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 permutation routing on trees.

Acknowledgements

The authors thank Michael Houle for useful suggestions and discussions.

References

[1] A. S. Acampora and S.I.A. Shah. Multihop lightwave networks: A comparison of store-and-forward and hot-potato routing. *IEEE Trans. Comm.*, 40(6):1082–1090, 1992.

[2] S. Alstrup, J. Holm, K. de Lichtenberg, and M. Thorup. Direct routing on trees (extended abstract). In *Proc. 9th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA'98)*, pages 342–349, 1998.

[3] P. Baran. On distributed communication networks. *IEEE Trans. Commun. Systems*, CS-12:1–9, 1964.

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

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

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

[7] A. Ben-Dor, S. Halevi, and A. Schuster. Potential function analysis of greedy hot-potato routing. *Theory Comput. Syst.*, 31(1):41–61, 1998.

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

[9] C. Busch, M. Herlihy, and R. Wattenhofer. Hard-potato routing. In *Proc. 32nd Annual ACM Symp. on the Theory of Computing (STOC'00)*, pages 278-285, 2000.

[10] C. Busch, M. Herlihy, and R. Wattenhofer. Randomized greedy hot-potato routing. In *Proc. 11th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA'00)*, to appear.

[11] J. R. Feehrer and L. H. Ramfelt. Packet synchronization for synchronous optical deflection-routed interconnection networks:. *IEEE Trans. Parallel Distri. Systems*, 7(6):605–611, 1996.

[12] U. Feige and P. Raghavan. Exact analysis of hot-potato routing. In *Proc. 33rd Annual Symp. on the Foundations of Computer Science (FOCS'92)*, pages 553–562, 1992.

[13] A. G. Greenberg and J. Goodman. Sharp approximate models of deflection routing in mesh networks. *IEEE Trans. Comm.*, 41(1):210–223, 1993.

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

[15] F. Meyer auf der Heide and M. Westermann. Hot-potato routing on multi-dimensional tori. In *Proc. Graph-theoretic Concepts in Computer Science: 21st Int'l Workshop (WG'95)*, Lecture Notes Comput. Sci. 1017, pages 209–221. Springer, 1995.

[16] A. Symvonis. Routing on trees. *Inform. Process. Lett.*, 57(4):215–223, 1996.

Alan Roberts, Antonios Symvonis and David R. Wood are with the Basser Department of Computer Science, The University of Sydney, NSW 2006, Australia. E-mail: alanr@zip.com.au, {symvonis,davidw}@cs.usyd.edu.au