Deterministic Many-to-Many Hot Potato Routing

Allan Borodin [*] Yuval Rabani [†] Baruch Schieber [‡]

Abstract

We consider algorithms for many-to-many hot potato routing. In hot potato (deflection) routing a packet cannot be buffered, and is therefore always moving until it reaches its destination. We give optimal and nearly optimal deterministic algorithms for many-to-many packet routing in commonly occurring networks such as the hypercube, meshes and tori of various dimensions and sizes, trees and hypercubic networks such as the butterfly. All these algorithms are analyzed using a charging scheme that may be applicable to other algorithms as well. Moreover, all bounds hold in a dynamic setting in which packets can be injected at arbitrary times.

[*] Dept. of Computer Science, University of Toronto, Toronto, Canada. This work was done while this author visited the Departments of Computer Science at The Hebrew University, Jerusalem and The Weizmann Institute of Science, Rehovot, Israel. E-mail: `borodin@cs.toronto.edu`

[†] Most of this work was done while this author was at the Lab. for Computer Science, MIT, Cambridge, MA, supported by ARPA/Army contract DABT63-93-C-0038, and at the Dept. of Computer Science, University of Toronto, Toronto, Canada. Present address: Computer Science Dept., The Technion, Haifa 32000, Israel. Work at the Technion supported by a David and Ruth Moskowitz academic lecturship award, and by a grant from the fund for the promotion of sponsored research at the Technion. E-mail: `rabani@cs.technion.ac.il`, URL: `http://www.cs.technion.ac.il/~rabani/`

[‡] IBM T.J. Watson Research Center, Yorktown, NY. E-mail: `sbar@watson.ibm.com`

1 Introduction

This paper studies routing in a synchronous network, in which at most one packet can traverse any link in each time step. We consider a form of routing known as *hot potato routing* or *deflection routing* [1, 5, 8, 9, 10, 11, 12, 13, 15, 20, 21]. The striking feature of this form of routing is that unlike traditional store and forward packet routing, it involves no queues at intermediate nodes. Thus packets are always moving, giving rise to the term "hot potato." A packet attempts to travel "towards" its destination. However, due to congestion, two or more packets may arrive at a node and wish to exit it on the same outgoing link. In the event of such contention, the node forwards one of the packets along the preferred edge. The other packet(s) will be "deflected" away from the preferred direction, exiting the node along other link(s). In particular, some packets may temporarily move further away from their destinations. This paper focuses on the general problem of many-to-many routing where an arbitrary number of packets can originate at any network node and an arbitrary number of packets can be destined for any node. In the context of hot potato routing, it is assumed that the number of packets originating at a node does not exceed the degree of the node. Moreover, we will consider the routing to take place in the following dynamic context: an adversary constructs both a many-to-many routing problem with k packets and a set of injection times for each of the packets (subject to the constraint that packets are not injected so as to exceed the degree of the node). Note that since all algorithms considered here are deterministic, this is equivalent to the situation where an adversary injects packets as it observes the routing process. In this setting we will analyze the number of steps (following its injection) it takes any packet to arrive at its destination as a function of k and the initial origin to destination distance that the packet has to travel.

We give optimal and nearly optimal deterministic algorithms for many-to-many packet routing in commonly occurring networks such as the hypercube, meshes and tori of various dimensions and sizes, trees and hypercubic networks such as the butterfly. All these algorithms are analyzed using a charging scheme that may be applicable to other algorithms as well. Moreover, all bounds hold in the dynamic setting.

Three key reasons for favoring deflection routing are: (i) potential faster switching since no messages are buffered at intermediate nodes, (ii) deflections disperse the packets adaptively and may afford the possibility of avoiding "hot spots," and (iii) the elimination of queues can reduce the price of switching hardware. Because of these reasons deflection routing was used in parallel machines such as the HEP multiprocessor [17], as well as high-speed communications networks [15], especially in optical networks [1, 9, 15, 20], where buffering involves transforming the packets into electronic media.

Although it has been advocated for thirty years (see Baran [3]), few papers have attempted any precise analysis for hot potato schemes, while experimentally these algorithms seem to work exceptionally well [1, 9, 10, 13, 15, 16]. One of the reasons for this may be the fact that the (usually adaptive) dispersion of packets in hot-potato routing seems to make the analysis quite difficult. Feige and Raghavan [8] resurrected interest in this topic within the theoretical computer science community by analyzing average case and randomized hot potato algorithms for the torus and hypercube networks. They also called attention to an important paper by Hajek [11] concerning worst case hot potato routing for arbitrary (i.e., many-to-many) routing problems.

Mansour and Patt-Shamir [14] considered the general problem of many-to-many routing of k packets in a store and forward manner. They were able to show that *any* greedy store and forward algorithm that only uses shortest paths will route packets with arbitrary origins and destinations so that each packet p arrives in at most $\texttt{dist}\ (p) + k - 1$ steps, where $\texttt{dist}\ (p)$ is the length of the shortest path from the source of p to its destination. A greedy algorithm in this setting is an algorithm that would always forward a packet if there is an available link on some shortest path to the destination. One can note that achieving a multiplicative bound of $\texttt{dist}\ (p) \cdot k$ is rather obvious, but the additive bound they derive is non-trivial. This bound is also optimal in the sense that there are cases (for example, when all the packets have the same destination) when no better bound is possible. Their analysis is easily seen to hold in our dynamic setting.

Although independent and not motivated by the Mansour and Patt-Shamir [14] result, Hajek [11] gave a similar result for hot potato routing. In the case of store and forward routing, an edge conflict results in the buffering and thereby delaying of a packet by one step. In the case of hot potato routing on an undirected network an edge conflict may result in a deflection that may delay a packet by two steps. Hajek [11] (and later Brassil and Cruz [6]) considered two cases: (i) many-to-one routing in arbitrary networks (i.e., the special case when all packets have the same destination), and (ii) many-to-many routing in the hypercube network. For both cases Hajek was able to devise a routing algorithm that guarantees delivery in at most $\delta_N + 2(k-1)$ steps, where δ_N is the diameter of the network and k is the total number of packets. Moreover, this bound is achieved by *any* hot potato algorithm that gives priority to packets based on the present distance to their destination; i.e., packets with shortest distance to go have priority. Once again, we observe that achieving a bound of $\delta_N \cdot 2(k-1)$ is quite easy for such schemes but Hajek's proof is relatively subtle. In fact, it is not at all clear that Hajek's analysis can be applied in the dynamic setting. The analysis by Brassil and Cruz [6] can be used to obtain a $\texttt{dist}\ (p) + 2(k-1)$ bound for hypercube, improving upon the $\delta_N + 2(k-1)$ bound in Hajek [11].

Recognizing that Hajek's analysis does not extend to the mesh, Ben-Dor, Halevi and Schuster [5] were the first to give many-to-many algorithms for the $n \times n$ two-dimensional mesh and also for higher dimensional meshes. For example, they give a somewhat complicated "greedy" algorithm for the $n \times n$ two-dimensional mesh that routes k packets within $O(n\sqrt{k})$ steps. The bound for a d-dimensional, n^d node mesh is $O(exp(d)n^{d-1}k^{1/d})$, where k is the total number of packets. When k is of the order of the size of the network these bounds coincide asymptotically with $O(\texttt{dist}\ (p) + k)$ bounds, but clearly are not as good when k is smaller.

Both the Mansour and Patt-Shamir [14] and the Hajek [11] results are derived by and can be interpreted as saying that contentions (deflections) can be charged in such a way that no packet delays another packet more than once. The implicit and still unrealized goal of Hajek was to find a simple (and we might add easy to implement) class of hot potato algorithms for which the bound $\texttt{dist}\ (p) + 2(k-1)$ holds for all undirected networks. (We remark that Hajek had a more general formulation applying to directed networks but we choose to restrict attention to undirected networks; in a directed network a deflection might cause a packet to be moved more than one edge away from its destination. The Mansour and Patt-Shamir result does apply to directed networks.) Hajek [11] gave an example showing that a natural candidate for such a class of algorithms (namely, algorithms that use the shortest distance to go priority rule) will not work for all networks.

Before listing our results in detail we discuss an important point in evaluating hot potato routing algorithms: their simplicity and "practical appeal". In order to capitalize on the potential of shorter switching time, as well as to further reduce the price of switching hardware, it is important that the routing choices made by a node at each step are simple. Ideally, the resources used by the routing algorithm — time, extra memory, randomness, communication — can be quantified precisely and optimized. A more practical approach, taken in essentially all of the routing literature, is to specify certain features of these algorithms that may imply their simplicity. We list some features from the least restrictive feature to the most restrictive. The first feature is *locality*: the assignments of packets at a node to outgoing links should depend only on data available at the node. Another feature is the *one-pass of packets* property defined by Hajek [11]: for each node and each time step, there is an ordering of the packets in the node according to some priority rules, so that a packet is deflected only if all outgoing links on shortest paths to its destination are assigned to packets before it in the ordering. Although not explicitly defined in [11] we assume that the priority of a packet has to depend only on its source, destination, its current node, and the link it arrived on. The most restrictive feature is the *one-pass of links* property defined by Feige and Raghavan [8]: for each node there is a fixed ordering of the incoming links, so that the incoming packets are always scanned and routed in this order. The last two properties can be extended to respective ℓ-pass properties. For example, in an *ℓ-pass of links* algorithm the links are scanned at most ℓ times and in each pass some packets get routed with all packets routed by the end of the ℓth pass. We assume that the only information obtained from previous passes is which outgoing links have already been assigned.

Adhering to these principles does not guarantee the intended simplicity. For instance, a node in a local algorithm may store the entire history of its previous decisions, and packets may contain the entire history of the nodes they passed on their way. However, typical local algorithms do not store information from round to to round, and typical packet headers do not contain significant data on its history. Similarly, a one-pass of links algorithm might perform very complicated calculations and store considerable data as it passes among the links. However, one usually perceives a one-pass of links algorithm as storing no additional data and performing $O(1)$ operations per link. A one-pass of packets algorithm seems to involve storing the priority for each packet and sorting according to the priority, so it would require more operations and additional storage compared with a simple one-pass of links algorithm.

Routing algorithms designed to perform well in the worst case should have "practical appeal". Again, it is hard (or even harder) to quantify this feature of the algorithm. We put some emphasis on "greedy" algorithms, agreeing with Ben-Dor, Halevi and Schuster [5] that such algorithms have the "practical appeal" desired by practitioners. Motivated by the discussion in Ben-Dor *et al.* [5] and similar to the definitions in Ben-Aroya, Eilam, and Schuster [4], and Feige [7], we distinguish between two types of greediness. An algorithm is *partially greedy* if it always assigns a packet an outgoing link on one of the shortest paths to its destination if such a link is available. Note that this property is local in the sense that an algorithm may assign a link e_1 to one packet and deflect another packet on link e_2, even if switching the links among the packets would avoid the deflection. An algorithm is *totally greedy* if it assigns packets at each node to outgoing links so as to minimize the number of deflections. Note that usually a "simple" algorithm would not possess the global property since this would require a maximum matching between packets and good links.

Results. In Section 2 we use a class of deflection arguments to give a general charging scheme that can be used for the analysis of hot potato routing algorithms. In this scheme we show how a certain property of the algorithm ensures that each deflection of a packet p would be charged to a different packet. We demonstrate the applicability of this scheme by showing simple algorithms achieving the $\texttt{dist}\ (p) + 2(k-1)$ bound for many-to-many routing in tree networks, the butterfly network, and lightly loaded two-dimensional meshes and tori. All these algorithms but the last satisfy the one-pass of links property. The algorithm for lightly loaded two-dimensional meshes and tori satisfies the two-pass of links property and one-pass of packets property. Feige [7] independently shows that for trees every totally greedy algorithm achieves the $\texttt{dist}\ (p) + 2(k-1)$ bound, and then shows how to use the result to achieve a bound of $2(\rho_N + k - 1)$ for any network N, where ρ_N is the radius of the network (i.e., $\min_v \max_u \texttt{dist}\ (u, v)$). A similar but somewhat weaker result of $2(\delta_N + k - 1)$ was independently given by Symvonis [18]. The $2(\rho_N + k - 1)$ bound is not necessarily achieved by a totally greedy algorithm and indeed Feige [7] shows that for some networks, totally greedy algorithms using simple priority rules to avoid livelock will result in significant (i.e., exponential in k) delay to a particular packet.

The main algorithmic results of this paper are in sections 3 and 4. In these sections we consider many-to-many routing algorithms for multi-dimensional meshes and tori. In Section 3 we give an algorithm that achieves the bound of $\texttt{dist}\ (p) + 4(k-1)$. Note that since the hypercube is a $\log N$-dimensional mesh this result applies also to the hypercube. The algorithm can be implemented to satisfy either the one-pass of packets property or the three-pass of links property. The algorithm has the additional feature that when applied to permutation routing in an $n \times n$ two-dimensional mesh it maintains the $O(n^{1.5})$ bound given in Bar-Noy *et al.* [2]. The advantage of our algorithm is that it also applies to many-to-many routing. We conjecture that the correct bound for this algorithm is the desired $\texttt{dist}\ (p) + 2(k-1)$. Feige [7] has analyzed a variant of our algorithm and shows that it achieves the bound $\texttt{dist}\ (p) + 2k$ for meshes and tori of all dimensions. It is also tempting to conjecture that these algorithms might be optimal (i.e., achieve worst case $O(n)$ for routing permutation) but Symvonis [19] has shown this to be false for dimensions bigger than two.

In section 4 we exhibit a more complicated, but partially greedy, algorithm that achieves the desired $\texttt{dist}\ (p) + 2(k-1)$ bound. We were not able to implement this algorithm in a bounded number of passes of packets (and thus also not in a bounded number of link–passes). A modification of this algorithm applied to the two-dimensional case achieves the desired $\texttt{dist}\ (p) + 2(k-1)$ bound and also maintains the $O(n^{1.5})$ bound given in [2] for permutation routing. For the two-dimensional case, Ben-Aroya, Eilam and Schuster [4] independently obtained an equivalent algorithm. As Feige [7] observes, the $\texttt{dist}\ (p) + 2(k-1)$ bound is somewhat arbitrary in that this bound is not a lower bound for all networks (e.g., a star network). For the mesh and, in particular, for the two-dimensional mesh, it is not known if the $\texttt{dist}\ (p) + 2(k-1)$ bound is necessary. If all packets are being routed to one destination then Ben-Aroya *et al.* prove that all packets can be routed in $\max_p \texttt{dist}\ (p) + k$, but it is not clear that the single destination case is a worst case.

2 The General Charging Scheme

In this section we present a general charging scheme for analyzing deflection routing algorithms. Consider a packet p that was deflected at time t_1 by packet p_1. (Packet p may be deflected by more than one packet in which case we are free to choose p_1 to ensure certain desirable properties.) Define a *deflection sequence* and a *deflection path* with respect to this deflection as follows. Follow packet p_1 starting at time t_1 either to its destination or up to time $t_2 > t_1$. Suppose that at time t_2 packets p_1 and p_2 meet. (Note that we are free to choose the time t_2 and the packet p_2.) Follow p_2 from time t_2 either to its destination or until some time $t_3 > t_2$, when it meets p_3. Then follow packet p_3. Continue in the same manner until a packet p_ℓ is followed to its destination. Define the sequence of packets: $p_1, p_2, \ldots, p_\ell$ as the deflection sequence of p at time t_1. Define the path that follows this sequence of packets from the point of deflection to the destination of p_ℓ to be the *deflection path.* The following claim is the key to our charging scheme.

Claim 1 *Suppose that for any deflection of packet p from node v to node u the shortest path from node 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.*

Proof. Suppose that p_ℓ is the last packet in more than one deflection sequence of packet p. Suppose that two such deflections occurred at nodes v_1 and v_1' at times t_1 and t_1' where $t_1 < t_1'$. In order for p_ℓ to be the last packet in both sequences both deflection paths must end at the destination of p_ℓ. Moreover, the length of the path from v_1' to this destination has to be $t_{\ell+1} - t_1'$ (where $t_{\ell+1}$ is the time when p_ℓ reached its destination). Consider the following path from u_1 (the node to which p was deflected from v_1) to the destination of p_ℓ. The first part of the path follows packet p from u_1 to v_1'. The last part follows the deflection path of p

at time t'_1. The length of this path is $t'_1 - (t_1 + 1) + t_{\ell+1} - t'_1 = t_{\ell+1} - t_1 - 1$ which is shorter than the deflection path from v_1 whose length is $t_{\ell+1} - t_1$; a contradiction. ■

This claim provides a design principle for routing algorithms, as summarized in the following corollary:

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

We note that this claim applies also to the dynamic case.

In the rest of this section we give some simple applications of the Claim 1 for three cases: trees, the butterfly network, and two-dimensional meshes.

First, consider any partially greedy algorithm on a tree. Suppose that a packet p is deflected from node v to u. The deflection path is given by following the unique packet p_1 that deflected p (there is only one edge adjacent to v along which p can make progress) either until p_1 reaches its destination or until it is deflected by p_2 in which case we continue by following p_2. Let p_ℓ is the last packet in the deflection sequence. It is easy to see that the deflection path is a shortest path from v to the destination of p_ℓ and that u is not in any such shortest path. Hence, the condition of Claim 1 holds.

Now, consider end to end routing on the undirected butterfly network (that is, all packets are injected at level 0 and all destinations are at the last level). We claim that any totally greedy algorithm guarantees arrival time of $\texttt{dist}\ (p) + 2(k-1)$, for every packet p. Again, we show that the condition of Claim 1 holds. Consider a packet p as it suffers a deflection at a node v in level i of the butterfly. We claim that the deflected packet p wanted to proceed to level $i+1$. This is because the only case in which p would want to go from level i to level $i-1$ is if it was previously deflected from a node u in level $i-1$ to level i. But in this case, since the algorithm is totally greedy, packet p would not be deflected, but go back from v to u in the next step. Consequently, we set the first packet in the deflection sequence to be the packet p_1 that advanced to level $i+1$ along the edge that p wanted to use. The deflection path follows the path taken by p_1, switching to p_2 that deflects p_1, if that ever happens, and so forth, until we reach p_ℓ that goes straight to its destination. The nodes along this path are located in monotonically increasing levels, and therefore the path is a shortest path. Note that because of the butterfly topology, no matter where packet p gets deflected it moves away from p_ℓ's destination. We remark that an analogous result applies to the directed butterfly network.

Finally, consider the *lightly loaded* two-dimensional mesh or torus. (In the next section we generalize the following result to the d-dimensional mesh/torus.) We will say that on the $n_1 \times n_2$ mesh (torus), a routing problem is "lightly loaded" if initially there is at most one packet in any node in the boundary columns (in the case of the mesh) and at most two packets in all other nodes (so that on the first step all packets can traverse along their origin row if they so wish). A routing problem is *fully loaded* if the number of packets at any node is at most the degree of the node (i.e., the maximum load consistent with hot potato routing).

We consider the following algorithm for the lightly loaded case, which is a modification of the algorithm given in [2] for permutations. Packets traverse along their origin row attempting to enter their destination column in either direction but preferably in the correct direction. A packet already traveling in the correct direction on its destination column has the highest priority and will never be deflected again. Packets traveling in their destination column have priority over row packets attempting to enter the column. Packets traveling in a row towards their destination column have priority over packets wishing to reverse their direction on this row. Subject to these priority constraints a packet heading in the wrong direction (on either a row or column) will reverse direction whenever possible. In particular, two packets traveling in the wrong direction can meet at a node and both reverse directions.

Theorem 3 *For any two-dimensional mesh or torus and for any lightly loaded dynamic routing problem every packet p will be routed to its destination within at most* $\texttt{dist}\ (p)+2(k-1)$ *steps.*

Proof. In the lightly loaded case, a packet can only be deflected on its origin row (trying to move towards or enter its destination column) or on its destination column (trying to move towards its destination). Let us concentrate on p as it suffers a deflection at node v. If p is deflected on its destination column or as it tries to enter its destination column then at least one packet p_1 at node v is heading directly for its destination. Packet p_1 will not be deflected in the future. The deflection path we use in this case is p_1's path, which is clearly the shortest path to p_1's destination, and p gets deflected away from p_1's destination. If packet p is deflected on its origin row while trying to move towards its column by packet p_1, then p_1 is heading towards its destination column. We have two possibilities. The first possibility is that p_1 will enter its column in the correct direction. In this case we again use p_1's path as the deflection path. The second possibility is that p_1 is deflected by p_2 which is heading in the correct direction on this column. In this case our deflection path will switch to p_2. In either case, the deflection path is a shortest path and p gets deflected away from end of this shortest path. (In the case of the torus, p's distance to the other end of the deflection path may remain unchanged after the deflection.) Therefore, in all cases Claim 1 applies. ∎

Notice that, in particular, any instance of permutation routing is lightly loaded. The above argument shows that on the two-dimensional mesh or torus, our algorithm completes routing a permutation within $O(n^2)$ steps. However, the $O(n^{3/2})$ analysis that was given in [2] applies to our algorithm as well and improves upon the many-to-many analysis. Of course, our analysis applies to any many-to-many instance, not only permutations.

3 The dimension-by-dimension mesh algorithm

Consider the two-dimensional $n_1 \times n_2$ mesh/torus again. We modify the algorithm presented in the previous section to handle the fully loaded case. In this case some packets may not be able to enter their origin row and will be deflected into their origin column. These packets have the lowest priority and attempt to enter any row. These packets attempt to move towards their destination row, but they move into any receptive row, if possible. In this way a deflection (away from the destination row) is well defined. As soon as a packet manages to get into a row, it follows the algorithm for the lightly loaded case.

Theorem 4 *In any fully loaded dynamic routing problem , any packet p will reach its destination within at most* `dist` $(p) + 4(k-1)$ *time steps.*

Proof. In the fully loaded case, p might first be deflected onto its origin column. We will have to show how to account for all of p's deflections until it enters some row. (Once p enters a row, the charging for deflections proceeds exactly as in the lightly loaded case.) Let us say that while p is in its origin column it gets deflected at times $t_1, t_2, \ldots, t_\ell$ and that these deflections take place at rows $r_1, r_2, \ldots, r_\ell$. Since p could not enter row r_i at time t_i there must be some packet p_i which is heading in the correct direction along row r_i. Notice that the times t_i are all distinct, but the rows r_i and the packets p_i are not necessarily all distinct. We will say that p is deflected by p_i at time t_i but we will not necessarily charge p_i for this deflection. Note that p_i might be deflected by some packet q as it tries to enter its destination column and this same packet q might also deflect p_{i+1} and many other subsequent p_j's. Consider the sequence $p_1, p_2, \ldots, p_\ell$. It may be the case that a given packet appears many times on this sequence. We want to account for deflections occurring in a subsequence $p_i, \ldots, p_j$ with $p_i = p_j$ and then effectively remove this subsequence (excluding p_j). Continuing in this way, we will end up with a final sequence of deflections without any packet being repeated. At this point we can charge the remaining deflections to the packets occurring in this final sequence. (It is these remaining deflection charges which cannot be simply separated from the deflection charges that will be incurred once the packet enters a row and hence each packet may be charged twice.)

Let t_i be the first time that p is deflected by some p_i which will again deflect p while it is still in its origin column and let t_j be the last time that $p_j = p_i$ so deflects p. Obviously p_i is deflecting p at the same row $r_i = r_j$. It is also clear that there must be an 1-1 correspondence between the deflections of p occurring at times $t_i, t_{i+1}, \ldots. t_{j-1}$ and the deflections of p_i that are occurring in row r_i, from time $t_i + 1$ to time $t_j - 1$. (This is necessary because p and p_i meet again at time t_j.) Each of these deflections of p_i can be charged to a unique packet exactly as in the lightly loaded case. Indeed, these are the charges that are being incurred by p_i during this time period as it is trying to enter its destination column. We use the same charges for the corresponding deflections of p.

We will now remove the subsequence $p_i, \ldots, p_{j-1}$. The remaining sequence may still have packets occurring more than once and we want to apply the same procedure to the sequence that remains. The only issue to worry about is that the charges will all be to distinct packets; that is, after we remove a subsequence having charged certain packets, the same packets cannot be charged again as we try to subsequently remove another subsequence. Suppose after removing $p_i, \ldots, p_{j-1}$, we next wish to remove $p_u, ..., p_{v-1}$. Clearly $u > j$ since p_j cannot equal to p_u as t_j is the time of the last occurrence of p_j in the deflection sequence. We claim that any packet receiving a charge for one of the deflections in the first subsequence completes too early to be "caught" by any deflection path that begins at time t_u or later. Consider the last time p_i is deflected before time t_j. As a result of this deflection p_i is either deflected away from its destination column, or remains in the same distance to its destination column. (The latter may happen only in a torus.) Trace p_i's path from this deflection until it last deflects p at time t_j, then trace p's path until it is first deflected by p_u at time t_u, then trace p_u's path until any deflection it suffers at its destination column. This path is at least one step behind any deflection path corresponding to deflections of p_i before time t_j. ■

We now wish to generalize this result to any number of dimensions of any size. (Clearly the two-dimensional result did not depend on the size of the mesh.) In particular, our generalization applies to the hypercube. We now describe this generalization to any number of dimensions and once again distinguish between the lightly loaded case (where at most two packets are injected at each node) and the fully loaded case. Say, we have a d-dimensional mesh and arbitrarily order the dimensions, calling them dimensions $1, 2, \ldots, d$.

The lightly loaded case: A packet traverses along the dimension 1 until it is able to enter dimension 2 in the proper 1 coordinate value and then traverses this dimension until it can enter dimension 3 in the proper 2 coordinate value, etc. Once a packet enters a given dimension it will stay in that dimension until it enters a higher dimension. More precisely, a packet now in dimension i wishing to next fix dimension j will enter the highest dimension h with $i \leq h \leq j$ that it can enter. If the packet cannot enter dimension j in the correct direction then it is said to be deflected. As before, once a packet enters dimension d it stays in this dimension until it reaches its destination. And as before, packets which are deflected always attempt to change direction as soon as possible.

Theorem 5 *The above algorithm guarantees delivery time of* `dist` $(p) + 2(k-1)$*, for every packet p in any lightly loaded dynamic routing problem on a d-dimensional mesh or torus. In particular, the bound holds for the n-dimensional hypercube with $N = 2^n$ nodes.*

Proof. The proof is a direct generalization of the proof for the two-dimensional case. Namely, when a packet p is deflected we trace a deflection path forward in time and dimensions. This path is a shortest path, and p is deflected away from its other end. ■

The fully loaded case: Consider the first step for a given packet p. Suppose that the first dimension which must be corrected is i. Packet p will attempt to enter the highest possible dimension ℓ with $\ell \leq i$. If p can enter such a dimension then it proceeds as in the lightly loaded case. Otherwise, p enters the lowest dimension h, with $h > i$. Thereafter p will attempt to enter a dimension less than or equal to i at which point it proceeds as in the lightly loaded case. While doing so the dimensions upon which p is traveling are monotonically non increasing. That is, p can be "forced" onto a lower dimension but never a higher dimension. (It has priority over packets wishing to enter h from a higher dimension, but not over packets already executing the lightly loaded case and wishing to enter h from a lower dimension.) Suppose that p is in dimension h while attempting to enter the lightly loaded case. If p's destination in the first $h-1$ dimensions is $x_1, \ldots, x_{h-1}$, then during the time period while p is in dimension h and is still attempting to enter the "lightly loaded case," it sets its "target" to be any node consistent with these coordinate values so that there is again a well defined notion of a deflection.

Theorem 6 *The above algorithm guarantees delivery time of* `dist` $(p) + 4(k-1)$*, for every packet p in any fully loaded dynamic routing problem on a d-dimensional mesh or torus. In particular, the bound holds for the n-dimensional hypercube with* $N = 2^n$ *nodes.*

Proof. Again, this is a direct generalization of the two-dimensional case. In particular, we must account for the deflections until a packet enters the lightly loaded stage. Any deflection of a packet p in this preliminary stage will be said to be deflected by a packet q traveling in the correct direction along dimension i, where i is the first dimension that must be corrected. Then, as in the two-dimensional case we will look at the deflection sequence and remove subsequences until there are no repetitions in the sequence. This can be done since once again the algorithm proceeds in such a manner that if packet p is twice deflected by packet q it must be the case that they intersect in the same place and hence again there is an 1-1 correspondence between the deflections of p and those of q between the two times q deflected p. ■

We show how to implement the algorithm for the lightly loaded case in two ways: (i) as a two-pass of links algorithm and (ii) as a one-pass of packets algorithm. Then, we show how to implement the algorithm for the fully-loaded case in two ways: (i) as a three-pass of links algorithm, and (ii) as a one-pass of packets algorithm.

The lightly loaded case: We first show the implementation as a two-pass of links algorithm. Consider a packet p entering a node v using the incoming edge in dimension i and direction ϕ (for $1 \leq i \leq d$, and $\phi \in \{+,-\}$). In the algorithm for the lightly loaded case there are two possibilities: (1) if packet p wishes to go out in the same dimension and the same direction, then it always succeeds in doing so. (2) if packet p wishes to go out on an outgoing link of dimension $h > i$ it succeeds only if no packet of higher dimension wishes to use the same outgoing edge. Each of these two possibilities is implemented in a separate pass.

For each node v, in the first pass we scan its incoming edges in any fixed order and assign the outgoing edges for all packets that wish to continue in their incoming direction; that is, if a packet p enters v in the ϕ direction of dimension i and wishes to continue in the same dimension and the same direction, then the outgoing edge of v in the ϕ direction of dimension i is assigned to p. In the second pass we scan the incoming edges in a fixed order from dimension d to dimension 1 (where in each dimension we first scan the incoming edge in the "+" direction and then the incoming edge in the "$-$" direction), and assign to each packet p its desired outgoing link unless this edge has been assigned already to a previous packet. In this case we assign to p an outgoing edge of the highest dimension amongst all the available outgoing edges whose dimension is not higher than the dimension of the desired link of p. It is easy to see that this scheme indeed implements the algorithm for the lightly loaded case.

This two-pass of links algorithm can be implemented as a one-pass of packets algorithm. The order of the packets would be first the packets that wish to continue in their incoming direction, and then the rest of the packets ordered according to the dimension of their incoming edges.

The fully loaded case: We implement the algorithm for the fully loaded case as a three-pass of links algorithm as follows. In the first two passes we consider only the packets currently following the algorithm of the lightly loaded stage and repeat the algorithm described above. In the third pass we scan the incoming edges in any order and assign the outgoing links for the rest of the packets (i.e., those that try to enter the lightly loaded stage). Each such packet enters the lightly loaded stage if it can do so using one of the available outgoing links.

This three-pass of links algorithm can also be implemented as a one-pass of packets algorithm. The order of the packets would be first the packets currently following the algorithm of the lightly loaded stage in the same order as above and then the rest of the packets.

Some aspects of these algorithms are not at all essential for establishing the stated bounds. For example, while a packet is attempting to enter the "lightly loaded stage," it does not matter if it sets its destination row as its "target." For the higher dimensional meshes we can keep a packet in row i while waiting to enter row j even if there is some row h with $i \leq h \leq j$ which is available. We have presented the algorithms so that they would be both simple to implement as well as hopefully working well "in practice." In particular, we suspect that these algorithms can be shown to have provably good (say, $O(n)$) performance for routing permutations.

4 A better algorithm for the multi-dimensional mesh

In this section we give another algorithm for the d-dimensional mesh. This algorithm will route any k packets so that the delivery time of each packet p is bounded by $\texttt{dist}\ (p)+2(k-1)$. Note that a hypercube is a $\log n$-dimensional mesh and thus our algorithm works for the hypercube. This algorithm when applied to the two-dimensional case can be slightly modified so it will still maintain the $O(n^{1.5})$ bound for routing a permutation, as the algorithm of [2]. For two dimensions this algorithm works also on tori. However, it does not seem to extend to work on any torus with at least three dimensions.

Our algorithm is motivated by the general charging scheme for analyzing deflection routing algorithms given in Section 2. Consider a packet p that is deflected at time t from node v to node u. W.l.o.g. assume that the coordinates of v are $(x_1, \ldots, x_r, \ldots, x_d)$ and the coordinates of u are $(x_1, \ldots, x_r - 1, \ldots, x_d)$.

We claim that in order for the condition of Claim 1 to hold it suffices to prove the existence of a deflection sequence $p_1, p_2, \ldots, p_\ell$ and a corresponding deflection path with the following properties:

1. Let v_ℓ be the destination of p_ℓ (the last packet in the sequence). The r-th coordinate of p_ℓ is at least x_r.
2. The deflection path from v to v_ℓ is a shortest path between these two nodes. Moreover, in this shortest path the coordinates of v are corrected in a cyclical order (if a correction is necessary) starting from coordinate r, moving to coordinate $r+1$, and so forth up to the coordinate $r-1$ to obtain v_ℓ.

To see this note that if the above two properties hold then the shortest path from u to v_ℓ is strictly longer than the deflection path from v to v_ℓ.

So, the problem at hand is how to define for every deflection the deflection sequence (and in particular the last packet in the sequence) that would satisfy these two properties. For this we need the following definitions.

Consider a packet p at an intermediate node v in time t. The *good directions* of p are all the directions along which p can advance from v towards its destination. Each such direction is denoted by a pair from $\{1, \ldots, d\} \times \{+, -\}$, where the first entry denotes the coordinate and the second entry denotes the good direction along the coordinate. Define the *interval* of p at v with respect to a good direction (r, ϕ) (where $\phi \in \{+, -\}$) to be the maximal cyclic (closed) interval that ends at $r-1$ and contains only coordinates that are already fixed; that is, if this interval is $[c, r-1]$, then for all $c \leq j \leq r-1$ (if $c \leq r-1$), or for all $1 \leq j \leq r-1$ and $c \leq j \leq d$ (if $c > r$), the current coordinate of p is also its destination coordinate and this is not the case for $c-1$. (The interval may be empty.)

At each time step of the algorithm each packet p that has not reached its destination yet is associated a *desired direction*. Intuitively, this direction is the good direction along which the packet p currently desires to advance. Initially, we make the desired direction of each packet to be the good direction along the minimum coordinate that it has to correct. Suppose that at time $t > 0$ packet p is at an intermediate node v and it arrived there along coordinate r. Then its desired direction at t is its good direction (r', ϕ), where r' is the closest coordinate greater or equal to r (again, we assume cyclical order) that is not fixed yet.

Let us now define the algorithm. At time t there are at most $2d$ active packets at a node $v = (x_1, \ldots, x_d)$ and the algorithm has to assign to each of these packets an outgoing direction. The assignment of the outgoing directions will obey the following two rules.

Rule 1. Suppose that (r, ϕ) is the desired direction of some packet p and that the packet q is assigned to this direction. Then, the interval of q with respect to (r, ϕ) is at least as long as the interval of p with respect to this direction.

Rule 2. Suppose that packet p is deflected along direction (r, ϕ). Let r' be the closest coordinate greater or equal to r (cyclically) such that (r', ϕ') is a good direction for p at v. Let q be the packet assigned to this direction. Then, the interval of q with respect to (r', ϕ') is at least as long as the interval of p with respect to this direction.

First, we give an algorithm that implements rules 1 and 2. Then, we prove that any algorithm that obeys these rules achieves delivery time of `dist` $(p) + 2(k-1)$ for each packet p.

Implementing rules 1 and 2 is done as follows. Each packet will have a list of good directions, beginning with the desired one. We do the assignment of packets to links in rounds. In each round, all packets compete. Each packet competes for the first possibility on its list. The priority of each packet is determined by the size of its interval with respect to this possibility, ties broken arbitrarily. Losing packets remove the head of their list. Winning packets keep on competing on the same directions in future rounds. A loser in one round may (by changing its list head and therefore its desired direction) cause a winner in the same round to become a loser in a subsequent round. The process ends when all losing packets have empty lists. Then, the final winners move on their winning direction and the rest of the packets are deflected arbitrarily. Since every round where someone loses reduces the sum of lengths of all lists, this is guaranteed to terminate.

Fix a coordinate and a direction. Note that the sequence of interval sizes associated with the packets that won in this coordinate/direction in the rounds of the competition is monotonically non-decreasing. Therefore, Rule 1 is implemented. Now, consider a deflected packet p. It lost the competition in all of its good directions. Hence, the interval of the current winner in each of these directions is no smaller than the interval of p w.r.t. this direction. This easily implies the implementation of Rule 2.

We give an alternative implementation that may be easier to implement. We first assign packets to all directions that are desired directions of at least one packet. Let (r_i, ϕ_i), for $i = 1, \ldots, s$, be the desired directions. Define the *primary candidate* of a desired direction (r_i, ϕ_i) to be a packet whose desired direction is (r_i, ϕ_i) and whose interval size with respect to this direction is maximum. Define the *desired interval size* of such a direction (r_i, ϕ_i) to be the interval size of its primary candidate with respect to this direction.

The order and the assignment of the packets is determined by the following procedure. Given a prefix of the assignment of packets, the procedure is invoked to determine the next packet in the list and its assignment.

Step 1: Test if there is a direction (r_i, ϕ_i) such that there exists an unassigned packet that is not a primary candidate of any direction whose interval size with respect to (r_i, ϕ_i) is more than the size of its desired interval.

Step 2: If such a direction (r_i, ϕ_i) exists, let p be the unassigned packet that is not a primary candidate of (r_i, ϕ_i) and whose interval with respect to this direction is the longest. Add p to the ordered list and assign it to (r_i, ϕ_i).

Step 3: Otherwise, if there is an unassigned desired direction, add its primary candidate to the ordered list, and assign it to this direction.

Step 4: Otherwise (i.e., if all the desired directions are assigned packets), pick the unassigned packet with the longest interval with respect to an unassigned direction. Add this packet to the ordered list, and assign it to this direction.

Steps 2 and 3 guarantee that if (r, ϕ) is the desired direction of some packet p, then the interval of the packet assigned to (r, ϕ) with respect to this direction must be at least as long as the respective interval of p. Thus, they implement Rule 1. Step 4 guarantees that if a packet p is assigned to some direction, then the interval of any deflected packet with respect to this direction is no longer than the respective interval of p. This implies Rule 2.

Finally, consider the application of the algorithm to the two-dimensional case. This algorithm can be modified slightly so that it still maintains the $O(n^{1.5})$ bound for permutation routing. This modification is given by adding the following rule: when several packets are competing on the same good direction and none of them has priority according to Rules 1 and 2 give priority to the packet that is already traveling in the same direction (in case it is one of the competing packets).

Now, we prove that any algorithm that obeys Rules 1 and 2 achieves delivery time of $\texttt{dist}\ (p)+2(k-1)$ for each packet p. For this we show how to construct the deflection sequences and deflection paths that satisfy the two properties listed above. Consider a deflection path $p_1, \ldots, p_\ell$. We associate the time series $t_0 \leq t_1 \leq \cdots \leq t_\ell$ with this sequence. This time series has the following meaning: in the corresponding deflection path we follow the path taken by packet p_i from time t_{i-1} to time t_i, for $1 \leq i \leq \ell$. We say that p_i is the *active* packet of the sequence at time t, for all $t_{i-1} < t \leq t_i$. The deflection sequences are constructed dynamically. That is, at time step t all the deflection sequences are defined only up to the active packet at time t. (Note that if the last packet in a deflection sequence has arrived at its destination by time t, then this sequence is fully-defined at time t.)

At time t we have to augment two types of deflection sequences: one that corresponds to deflections at time t, and one that corresponds to past deflections. First, consider a deflection sequence corresponding to a present deflection. We have to define the first active packet in this sequence. W.l.o.g. suppose that packet p is deflected from node $v = (x_1, \ldots, x_r, \ldots, x_d)$ to node $u = (x_1, \ldots, x_r - 1, \ldots, x_d)$ (along direction $(r, -)$). Let c be the closest coordinate greater or equal to r (cyclically) such that (c, ϕ') is a good direction for p at v. The first active packet p_1 in the corresponding deflection sequence is the packet assigned to direction (c, ϕ'). W.l.o.g. assume that ϕ' is $+$ direction; i.e., p_1 advances to $(x_1, \ldots, x_c + 1, \ldots, x_d)$.

Lemma 7 *Destination coordinate c of both p and p_1 is at least x_c+1. If $c > r$, then destination coordinates r through $c-1$ of both p and p_1 are $x_r, ..., x_{c-1}$.*

Proof. The first claim follows since $(c, +)$ is a good direction for p, and since p_1 advances towards its destination coordinate c when it is assigned to direction $(c, +)$. The second claim follows from Rule 2, since if $c > r$, the interval of p with respect to c contains $[r, c-1]$, and thus the interval of p_1 with respect to c must contain $[r, c-1]$ as well. ■

Now, consider a deflection sequence corresponding to a past deflection. Suppose that p_k is the active packet in this sequence at time t. Let the desired direction of p_k at time t be (r, ϕ). The next active packet p_{k+1} in this sequence is the packet that is assigned the direction (r, ϕ).

(It may continue to be p_k.) We set the desired direction of p_{k+1} at time $t+1$ to be (c, ϕ'), where c be the closest coordinate greater or equal to r (cyclically) that is not fixed yet.

Lemma 8 *The interval of p_{k+1} with respect to its new desired direction at time $t+1$ contains the interval of p_k at time t with respect to its desired direction at this time.*

Proof. Let p_{k+1}'s interval with respect to (r, ϕ) at time t be $[c', r-1]$ (if the interval is empty, set $c' = r$). Then, p_{k+1}'s interval with respect to (c, ϕ') at time $t+1$ is $[c', c-1]$, since by definition of c, all the coordinates between r and $c-1$ are correct. So, the lemma clearly holds if $p_{k+1} = p_k$. If $p_{k+1} \neq p_k$, then, by Rule 1, since p_{k+1} is assigned to the desired direction of p_k, the interval of p_k with respect to its desired direction (r, ϕ) must be contained in $[c', c-1]$, and the lemma holds in this case as well. ■

Lemma 9 *If $r = c$ then $\phi' = \phi$.*

Proof. W.l.o.g. assume $\phi = +$. Let the rth coordinate of p_{k+1} (as well as p_k) at time t be x_r. By the first claim in Lemma 7, the rth destination coordinate of p_{k+1} is at least $x_r + 1$. Since p_{k+1} moves to $x_r + 1$ and r remains its desired direction, then the value of this destination coordinate is at least $x_r + 2$, which means that $\phi' = +$. ■

Consider a deflection sequence $p_1, \ldots, p_\ell$ and the corresponding time series $t_1, \ldots, t_\ell$ associated with a deflection of packet p from node $v = (x_1, \ldots, x_r, \ldots, x_d)$ to node $u = (x_1, \ldots, x_r - 1, \ldots, x_d)$. Using the above three lemmata, we show the following claims. (The analogous claims for a deflection to $(x_1, \ldots, x_r + 1, \ldots, x_d)$ can be deduced similarly.)

Claim 10 *Let $t_1 \leq t \leq t_\ell$. Consider the path given by following the active packets in the sequence up to time t. The r-th coordinate of every node in this path is at least x_r.*

Proof. As long as the deflection path proceeds along the rth coordinate, Lemmata 7 and 9 insure that the value of the rth coordinate is monotonically increasing. As soon as the deflection path uses another coordinate, Lemmata 7 and 8 insure that r is contained in the interval of the current packet in the deflection sequence with respect to its desired destination. Therefore, the value of the rth coordinate does not change along the rest of the path. ■

Claim 11 *For all $t_1 \leq t \leq t_\ell$, the path given by following the active packets in the sequence up to time t is a shortest path in which the coordinates are corrected in cyclical order.*

Proof. By Lemmata 7 and 8, the sequence of coordinates that the deflection path traverses is in cyclical order, i.e., it is bitonic, with one monotone segment starting at $c \geq r$, followed by another monotone segment starting at $c' \geq 1$ and ending at $c'' < r$. Lemmata 7 and 9 imply that in each portion of the path traversing a single coordinate, the value of that coordinate changes monotonically. ■

We conclude from these claims that following Rules 1 and 2 implies the conditions required in Claim 1, which in turn implies the claimed delivery time bounds.

5 Conclusions and open problems

The many-to-many routing problem studied here is symptomatic of the difficulties in analyzing hot potato algorithms. One intuitively believes that the "desired" $\texttt{dist}\ (p) + 2(k-1)$ bound (being as weak as it is for problems such as permutation routing) should hold for all networks via a "simple" and "practically appealing" algorithm. At the very least one suspects that for each network there should be an algorithm achieving the desired bound and the analysis should be reasonably direct using some easily constructed deflection path argument. But perhaps our intuition is wrong!

Obviously many open problems remain for the subject of hot potato routing. Here we just mention some of the problems most directly related to the results of this paper.

1. Is there an undirected network for which no deterministic hot potato algorithm can route every many-to-many instance so that packet p arrives within $\texttt{dist}\ (p) + 2(k-1)$ steps? Is it possible to establish an $O(\texttt{dist}\ (p) + k)$ bound for all networks?
2. What is the worst case many-to-many bound for Hajek's "shortest to go" priority methods when applied to the two-dimensional and higher dimensional meshes? Does every such scheme achieve the "desired" bound? Does any such scheme achieve the desired bound?
3. Can the bound given for the dimension by dimension algorithm be improved to the desired bound in the fully loaded case?
4. For any of the algorithms given in this paper what is the best bound for permutation routing?
5. Can one show that the "one-pass of links" property is a substantial restriction? That is, can lower bounds be proven in this model that do not hold in general for hot potato algorithms?
6. Is there an algorithm with a bounded number of packet/link–passes that achieves the $\texttt{dist}\ (p) + 2(k-1)$ bound for meshes and tori?

References

[1] A.S. Acampora and S.I.A. Shah. Multihop lightwave networks: a comparison of store-and-forward and hot-potato routing. In *Proc. IEEE INFOCOM*, pages 10–19. IEEE Computer Society Press, 1991.

[2] A. Bar-Noy, P. Raghavan, B. Schieber, and H. Tamaki. Fast deflection routing for packets and worms. In *Proc. 12th Symp. on Principles of Distributed Computing*, pages 75–86, August 1993.

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

[4] I. Ben-Aroya, T. Eilam, and A. Schuster. Greedy hot-potato routing on the two dimensional mesh, to appear in *Distributed computing*, 9(1), 1995.

[5] A. Ben-Dor, S. Halevi, and A. Schuster. Potential function analysis of greedy hot potato routing. In *Proc. 13th Symp. on Principles of Distributed Computing*, August 1994. to appear.

[6] J.T. Brassil and R.L. Cruz. Bounds on maximum delay in networks with deflection routing. In *Proc. 29th Allerton Conf. on Communication, Control and Computing*, pages 571–580, 1991.

[7] U. Feige. Observations on hot potato routing. In *Proc. 3rd Israel Symp. on the Theory of Computing and Systems*, pages 30–39, 1995.

[8] U. Feige and P. Raghavan. Exact analysis of Hot Potato routing. In *Proc. 33rd Symp. on Foundations of Computer Science*, pages 553–562, October 1992.

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

[10] A.G. Greenberg and B. Hajek. Deflection routing in hypercube networks. *IEEE Transactions on Communications*, 40(6):1070–1081, June 1992.

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

[12] C. Kaklamanis, D. Krizanc, and S. Rao. Hot potato routing on processor arrays. In *Proc. 5th Symp. on Parallel Algorithms and Architectures*, pages 273–282, 1993.

[13] D.H. Lawrie and D.A. Padua. Analysis of message switching with shuffle-exchanges in multiprocessors. In *Interconnection Networks.* IEEE Computer Society Press, 1984.

[14] Y. Mansour and B. Patt-Shamir. Greedy packet scheduling on shortest paths. In *Proc. 10th Symp. on Principles of Distributed Computing*, pages 165–175, August 1991.

[15] N.F. Maxemchuk. Comparison of deflection and store and forward techniques in the Manhattan street and shuffle exchange networks. In *Proc. IEEE INFOCOM*, pages 800–809. IEEE Computer Society Press, 1989.

[16] R. Prager. An algorithm for routing in hypercube networks. Master's thesis, University of Toronto, September 1986.

[17] B. Smith. Architecture and applications of the HEP multiprocessor computer system. In *Proc. 4th Symp. on Real Time Signal Processing*, pages 241–248. SPIE, 1981.

[18] A. Symvonis. A Note on Deflection Routing on Undirected Graphs. University of Sydney, Department of Computer Science TR 493, November 1994.

[19] A. Symvonis. Private Communications, November 1994.

[20] T. Szymanski. An analysis of “Hot Potato” routing in a fiber optic packet switched hypercube. In *Proc. IEEE INFOCOM*, pages 918–925. IEEE Computer Society Press, 1990.

[21] Z. Zhang and A.S. Acampora. Performance analysis of multihop lightwave networks with hot potato routing and distance age priorities. In *Proc. IEEE INFOCOM*, pages 1012–1021. IEEE Computer Society Press, 1991.