Randomized Single-Target Hot-Potato Routing [*†]

Ishai Ben Aroya [‡] Ilan Newman [§] Assaf Schuster [¶]

Abstract

We present randomized hot-potato routing algorithms on d-dimensional meshes and on the n-dimensional hypercube. The algorithms are designed for routing many packets to a single destination, or a relatively small number of destinations. The important feature, which was not obtained previously, is that the algorithms utilize the higher in-degree of the nodes and are asymptotically optimal.

[*] A preliminary version was presented at the 3rd ISTCS 1995 (Israeli Symp. on the Theory of Computing Systems).

[†] This work was supported in part by the French-Israeli grant for cooperation in Computer Science, and by a grant from the Israeli Ministry of Science.

[‡] Department of Computer Science, Technion, Haifa, Israel 32000. E-mail: ishai@cs.technion.ac.il

[§] Mathematics and Computer Science, Haifa University, Haifa, Israel 31905. E-mail: ilan@mathcs.haifa.ac.il

[¶] Department of Computer Science, Technion, Haifa, Israel 32000. E-mail: assaf@cs.technion.ac.il

1 Introduction

In this work we consider a routing mode known as *hot-potato* or *deflection* routing [AS91, GG86, GH92, Haj91, Max89, Szy90, ZA91, NS89]. The important characteristic of algorithms which assume this mode is that they use no buffer space for storing delayed packets. Each packet, unless it has already reached its destination, must leave the processor at the step following its arrival. Packets may reach a processor from all its neighbors and then are redirected, each one on a different outgoing link. This may cause some packets to be "deflected" away from their preferred direction. Such an unfortunate situation cannot happen in the "store-and-forward" routing mode, in which a packet can be stored at a processor until it can be transmitted to its preferred direction.

Much work has recently focused on exact analysis and design of hot potato routing algorithms. One reason is that variants of hot potato routing are used by recent parallel machines such as the HEP multiprocessor [Smi81], and by high-speed communication networks [Max89]. In particular, hot potato routing is very important in fine-grain massively-parallel computers, such as the Caltech Mosaic C [Sei92]. For such machines, even the inclusion of a small sized storage buffer at each processor causes a substantial increase in the cost of the machine. Another domain in which deflection-type routing is highly desirable is optical networks [AS91, GG86, Szy90, ZA91]. In such networks, storage must take the electronic form. Thus, packets that should be stored must be converted from (and back to) the optical form. In the current state of technology, this conversion is very slow compared to optical transmission and switching rates.

The other reason for the growing interest in hot potato routing, which made this field the subject of several recent works is that the hot potato paradigm seems by its nature a 'distributed dynamic' process, where packets move at each step according to some simple set of *local* rules. As it turned out, this process proved to be hard to analyze, especially if the algorithm is somewhat 'simple', (we shall see along the sequel a possible definition of what 'simple' should be).

The first hot-potato algorithm was proposed by Baran [Bar64]. Borodin and Hopcroft suggested an algorithm for the hypercube [BH85]. Prager [Pra86] showed that the Borodin-Hopcroft algorithm terminates in n steps on the 2^n-nodes hypercube for a special class of permutations. Several subsequent works considered permutation routing, and randomized algorithms were proposed, so that the complexity for most permutations could be proven to be asymptotically optimal [FR92, KKR93] for d-dimensional meshes (or tori). A 'simple', deterministic, non optimal algorithm for 2-dimensional meshes was given by [BNRST93]. A *non simple* asymptotically optimal deterministic algorithm for permutation routing on meshes was given by [NS92]. The algorithm is based on sorting. An improvement in the leading constant was later obtained by [KLS94]. Very recently, a lower bound on routing permutations by a certain class of algorithms was given [BS94].

Work on general type of routing problems was done by [Haj91, BC91, BHS94, Fei94, BRS94, BTS95]. The goal in earlier works [Haj91, BC91], which was pursued in [BTS95, Fei94, BRS94], was to present a "simple" algorithm for hypercubes and meshes which routes k packets with any combination of origins and destinations, in $d_{max} + 2(k-1)$ steps, where d_{max} is the maximal source-to-destination distance. But what do we mean by a 'simple' algorithm? The permutation routing algorithms of [NS92] and later [KLS94] are based on sorting. In particular they work in phases, and at times packets would be sent away from their destinations in order to keep the correct structure. In a sense, they simulate "store and forward" routing algorithms in a hot potato manner. Apparently, this kind of algorithms does not satisfy the architects of machines which are based on hot potato routing.

Let us outline a few features of hot potato algorithms that are desirable and were mentioned in the literature.

- Hajek [Haj91], and Feige and Raghavan [FR92], suggested variations of the *one-pass* property of algorithms: Packets that enter the node are considered one-by-one in a pre-specified order on the in-coming arcs. Each packet, in its turn, is assigned to an out-going arc which is not already taken by the previously considered packets. Thus the involved hardware scans the entrances in a single pass while assigning exits on-the-fly. The one-pass property was generalized by [BRS94] to k-*pass*, which is defined analogously.
- Ben-Dor, Halevi and Schuster [BHS94] defined *greedy* algorithms where a packet always advances towards its target unless all the "good" exits are already taken by other advancing packets. This definition was later extended to the *strongly greedy* concept [BTS95] or equivalently the *totally greedy* [BRS94] or the *maximal advancing* concept [Fei94], so that not only is the algorithm greedy, but also the number of deflected packets is minimized. Note that an implementation of a strongly greedy algorithm would involve solving a maximum bipartite matching problem at each node at each step, and thus seems "less simple" than the implementation of the greedy rule.

A difference in the applicability of algorithms that are greedy and algorithms of the type presented by [NS92] is the following: In [NS92], the algorithm complexity is guaranteed only for packets that are in the system at a fixed starting point t_0. If some packets pop up in the system a bit later, the whole algorithm will have to be delayed, or re-initiated after it ends for the first bunch of packets. Moreover, processors will know that the algorithm has terminated only after a fixed given time, even if the initial number of packets is small, or they all arrived long ago. In contrast, greedy algorithms are more 'distributed' in nature; as each node *locally* tries to minimize the distance of the packets it sees without keeping any obvious global structure, packets may enter the system at various steps without seemingly changing the situation. It is hoped that the right analysis of such algorithms will lead to the desirable situation of dealing with an 'on line' stream of entering packets. However, as arguments on the global situation are hard to pinpoint at that case, the analysis of such algorithms becomes harder and uses different machinery. We concentrate here on developing such algorithms for a restricted routing problem.

We focus mainly on the case of a single target, i.e., all packets are destined to the same place. As the degree of the n-hypercube with 2^n nodes is n, one might expect that during most steps, as many as n packets will enter the destination. As the hypercube is bipartite, if all packets start with even distance to the target then we may expect packets to arrive during even steps only. Thus, if there are k packets in the system (where k can be as big as $n2^{n-1}$), an immediate lower bound is $2k/n$. However, the bound of [Haj91, BC91] is worse by asymptotically a factor of n. Indeed our main result is a randomized algorithm that bridge this gap. We show that k packets can be routed to a single target in $\frac{2k}{n} + o(\frac{k}{n}) + \mathrm{poly(n)}$ steps with high probability. We then generalize this result in two ways. First we show, following [BC91, BTS95] that this can be extended for routing problems with more than one (but rather a small number of) destinations. Then we show that the algorithm can be generalized to d-dimensional meshes (with d not necessarily constant). Our algorithms are all greedy, (in fact strongly greedy) and have the two-pass property.

The main problem in single target routing is to make equal number of packets reach the destination via each entry. The difficulty with a deterministic approach is the "symmetry" that is inherent in the problem. We try to avoid this difficulty here by using randomization. However, a simple-minded oblivious random-choice does not suffice for our purposes: The initial distribution of the packets is not known in advance, hence choosing at random during steps in which the packet advances does not guarantee the goal. On the other hand, when the packet is deflected, it gets a chance to add a random "bad" dimension to the set of address bits which need to be switched. We use this choice to ensure that when the deflected packet starts advancing again it will take a path which will end up entering the destination via a random neighbor. We hope that the machinery that we develop here will be of use in subsequent research in this area.

The rest of this work is organized as follows. In Section 2 we describe the model and some preliminaries. In Section 3 we present the single-target algorithm for the hypercube. In Section 4 we describe the changes that are required for the hypercube algorithm to make it work for d-dimensional meshes too. In Section 5 we give some conclusions and open problems.

2 The Model and Preliminaries

We represent a network of processors by a graph in which each node corresponds to a processor and each arc represents a communication link. The networks we deal with are bipartite undirected graphs (the n-dimensional Boolean cube and d-dimensional meshes). Thus each edge is thought of as a bi-directional link in which at most one message can pass in each direction at any step. Each node has a unique *id* and each node knows the topology of the network.

A d-dimensional mesh of side-length N, is a graph with N^d nodes. Each node corresponds to a tuple $(x_0, \cdots, x_{d-1})$ where $0 \leq x_i \leq N-1$. Between $(x_0, \cdots, x_{d-1})$ and $(y_0, \cdots, y_{d-1})$ there is an edge if $x_i = y_i$ for all but a single index i', and $|x_{i'} - y_{i'}| = 1$.

The n-dimensional hypercube is a graph with 2^n nodes and $n2^{n-1}$ edges. Each node corresponds to a binary string of length n. Two nodes are connected by an edge if their corresponding strings differ in exactly one place. Every node in the hypercube has exactly n neighbors. We sometimes refer to edges by their dimension, that is, by the index of the place for which the two end points of the edge differ. The dimensions are indexed by $0, \cdots, n-1$.

The problem which is dealt with in this work is that of routing a set of packets in a synchronous network using the hot-potato style. A routing problem is a set of ordered pairs of nodes. Each such pair corresponds to an origin node at which a packet originates and a destination node to which the packet is sent. A routing problem is said to be of the *single-target* type if all packets are destined to the same node.

A hot potato routing algorithm is a synchronous algorithm, namely, all packets are sent in discrete steps. A packet that is sent using an arc at step t is guaranteed to reach the node at the other side of the arc before the beginning of step $t+1$. In hot potato routing nodes do not have buffers for waiting packets, a node must send each packet in the step following its arrival (if it is not the destination of the packet). In particular, this means that at the starting point, a node of degree d may have at most d packets to send. Each node in the network performs the following operations at every step:

- Receives the packets that were sent to it in the previous steps on entering arcs.
- Performs a local calculation.
- According to the result of the calculation, the node sends out all the packets on the outgoing arcs except for the packets which are destined to it. At most one packet can be sent on each outgoing arc.

2.1 Greedy Hot-Potato Routing

We say that e is a *good arc* for a packet p if e is on a shortest path from the current node to the destination. In that case we also say that the direction of e is a *good direction* for p and if e is along the i-th dimension then i is a *good dimension* for p. Conversely, if traversing e will increase the distance of p to its destination we say that e is a *bad arc* for p which goes in a *bad direction*, and that i is a *bad dimension* for p.

We say that a packet p *advances* in step t if it gets closer to its destination at that step, i.e., it moves on a good arc. Otherwise, we say that p is *being deflected.*

The algorithms presented in this work are *greedy* in the sense that a packet always tries to advance. Following [BHS94] we make the following definitions.

Definition 2.1 *A hot-potato routing algorithm is said to be* greedy *if whenever a packet p exits the current node using a bad arc, all its good arcs must be used by other advancing packets.*

We say that an algorithm is strongly greedy *when, in addition to being greedy, it also maximizes the number of advancing packets from every node. We say that the algorithm is* weakly greedy *when, from every non-empty node, at least one packet advances.*

Our aim is to design an algorithm that is greedy and that will route k packets to a single target, in time that is close to $2k/d$, where d is the degree of the network. Since all packets have the same destination, all packets at a given node are identical with respect to the algorithm. Since the algorithm is greedy, it sends as many as possible of these 'identical' packets towards the target.

2.2 Preliminaries

We distinguish between packets with odd distance to the destination and those for which this distance is even. As the networks we deal with are bipartite, packets from the two classes never meet at the same node. Thus, from now on, unless otherwise stated, we restrict our discussion to the packets that their initial distances are even.

Definition 2.2 *The even distance packets destined to a node z of a network are m-*dense around *z if there exists an integer $j > 0$ such that for all $1 \leq i \leq j$ there are at least m packets at distance $2i$ from z and there are no packets at a distance greater than $2(j+1)$.*

When m is not important, we sometimes simply say that the packets are *dense around z*, omitting m. For greedy algorithms, if the packets are dense, then after two steps a packet will arrive at the destination, and the network remains at least 1-dense, provided that there are still packets left in the network. In other words, as long as packets are dense, at least one packet arrives on every second step.

Let d_{max} be the maximal source-to-destination distance of any packet in the system.

Lemma 2.3 *For any network with bi-directional links, and* any *greedy algorithm, if all packets are destined to a single node z, then after d_{max} steps the packets will be 1-dense around z.*

Proof: Consider a packet p_0 during the first d_{max} steps. Assume that p_0 did not arrive to its destination. Then, it must have been deflected at least once during the d_{max} steps. By simple induction it can be proved that from the step after the last deflection till the d_{max}-th step, there is always a packet which is two steps closer to the destination than p_0. ∎

Theorem 2.4 *[Haj91]. For* any *network with bi-directional links, and* any *greedy algorithm, if all k packets are destined to a single node z, then after $2(k-1)+d_{max}$ steps, all packets will reach the destination.*

This theorem was given originally by Hajek [Haj91]. The notion of greedy routing makes it easy to prove in our framework.

Proof: Consider a packet that is initially the closest to the destination and which defeats all other packets in any conflict (it is easy to see that such a packet exists). Then this packet arrives to the destination by step d_{max}. By Lemma 2.3, after d_{max} steps, all other packets will be 1-dense around the destination and thus, from that time on, at least one packet will arrive at every second step. ▮

The theorem above ensures essentially that after d_{max} steps, at least one packet will reach the destination at every other step. However, we want to achieve considerably more. For instance, for the n-degree hypercube, one expects that as many as n packets will reach the destination at every other step (after possibly a short transition time).

3 Routing on the Hypercube

Suppose k packets are destined to a single node of the n-dimensional hypercube. Then at least k/n steps are required to move all the packets to that node, because at most n packets may enter it at any step. If all packets originate at nodes of even distance to the target then it requires at least $2k/n$ steps. In this section we present an algorithm which obtains this bound up to an additive quantity of $o(k/n) + \text{poly(n)}$, with high probability.

3.1 Cyclic Advancing and its Properties

For a node z we denote by $G(z)$ the set of indices in which it differs from the destination node (i.e., the set of good directions). For a node z and a dimension i we define the next dimension in *the cyclic order* to be

$$n(i,z) = (i + \min\{j \geq 1 : (i + j \bmod n) \in G(z)\}) \bmod n$$

For instance, if $z = (0,1,0,1,0)$ and the destination is $(0,0,0,0,0)$, then $n(i,z) = \begin{cases} 1 & \text{if } \mathrm{i} = 0,3,4 \\ 3 & \text{if } \mathrm{i} = 1,2 \end{cases}$.

Definition 3.1 *An algorithm is called* cyclic advancing *if whenever a packet p enters a node z via a good arc i (i.e., p was not deflected in the previous step), then during the next step a packet will advance from z via the arc $n(i,z)$.*

We view the *cyclic advancing* condition as if the same packet p which enters z via arc i is supposed to be sent out via $n(i,z)$. However, when several packets enter the node it is possible that another packet will be sent via $n(i,z)$, rather than p. If all packets are destined to the same node, the only property which distinguish between them is the arc through which they entered. The packet which advances using $n(i,z)$ "replaces" p in the analysis.

Definition 3.2 *Let z be a vertex and e a good arc entering z. The path that a packet is supposed to take from z with respect to e is the path that a packet would take according to the algorithm assuming that it entered z through e and it is the only packet in the system.*

Observation 3.3 *Let $\mathcal{A}$ be a cyclic advancing single-target algorithm. Consider an instance of the execution of $\mathcal{A}$. Assume that in that instance a packet p advanced to z at time $t-1$ and let $(e_1, \cdots, e_m)$ be the path which p is supposed to take from z (with respect to the arc it used to enter z). Then for any $0 \leq i \leq m-1$ a packet will be sent on e_i at time $t+i$.*

The degree of the destination node is n. We will fully utilize this fact if, starting from a certain step, n packets will reach the destination at every even step. When several packets arrive to a neighbor of the destination at a certain step, only one of them will reach the destination at the next step. We thus should try to make n packets reach n different neighbors of the destination node at every even step. We do that by associating a color in the range $\{0, \cdots, n-1\}$ with every packet. This coloring is for analysis purposes only.

Definition 3.4 *The* color of a packet *p is the index $c(p)$ of the last bit which p would flip on its supposed path to the destination.*

Note that for a cyclic advancing algorithm and a single target, this definition sets the color of the packet according to the initial choice of advancing direction. In a multi-packet system this color may change as a result of the packet not following the cyclic advancing rule. However, in such a case another packet, which is identical to the first from the algorithm viewpoint, maintains this color by advancing according to the cyclic advancing rule. We summarize the discussion on the relation between $c(p)$ and the arc which p is supposed to take at the next step in the following observation.

Observation 3.5 *Suppose a packet p at a node z with good arcs along dimensions $d_0 < d_1 < \cdots < d_i$ is colored with $c(p) = d_j$. If the algorithm is cyclic advancing then a packet will exit z during the next step along $d_{(j+1)\mathrm{mod}(i+1)}$. That packet will have the color $c(p)$ when it leaves.*

Proof: $d_{(j+1)\mathrm{mod}(i+1)}$ is the arc that p is supposed to take according to the cyclic advancing rule. Now the observation is implied by Observation 3.3 and the definition of $c(p)$. ∎

Consider a set of packets P of equal distance to the destination. The following claim shows that there is a relation between the number of packets in P which arrive to the destination with no deflection, and the number of different colors in P.

Claim 3.6 *Consider an execution of a greedy and cyclic advancing algorithm $\mathcal{A}$. Let P be a set of m packets that are at distance r at time t and suppose that they have distinct colors, that is, the set of colors they have is of size m. Then for each of the following steps $l = 1, ..., r$, a set of m packets will be at distance $r - l$ from the destination. Moreover, these m packets will have the same set of m colors.*

Proof: By induction on r. For $r = 1$, if there are m packets of distance 1 having different colors, then they must be at different nodes. Hence, all packets will enter the destination at the next step.

For $r > 1$ consider a site that contains k of these packets with k different colors. By Observation 3.5, at the next step k packets with the same k colors leave this site and are at distance $r - 1$. As this is true for each of the sites containing the packets with distinct m colors, there will be m packets with the same set of m different colors at distance $r - 1$ and by the induction hypothesis we are done. ∎

In particular, in the above setting, at least m packets will arrive to the destination at step $t + r$.

In order to make use of the properties of cyclic advancing algorithms we should try to maximize the number of different colors of packets which have the same distance to the destination. Since advancing packets keep their colors, the coloring of packets is done in the first step, and during steps in which the packet is deflected or following a deflection step. We will use random coloring to ensure a high variety of colors.

3.2 Routing to a Single Target

We now describe the algorithm for routing to a single target on the n-dimensional hypercube. Each node knows the target, hence its distance and the good and bad directions. Each node keeps an out-queue on each arc incident to it. The algorithm is described as the collection of operations performed by a node z of distance r from the destination.

Algorithm HYPERCUBE_SINGLE_TARGET.

0. At the first step, packets start advancing arbitrarily according to the greedy rule.

1. Add each packet which advanced *into z via arc i to the out-queue of arc $n(i, z)$.*

2. For each packet which was deflected *into z via arc i,*

- *(a) with probability $\frac{r-1}{n}$ choose uniformly at random any good arc other than $n(i, z)$, and add the packet to its out-queue.*
- *(b) with probability $\frac{n-r+1}{n}$ add the packet to the out-queue of arc $n(i, z)$.*

3. If there are arcs with two or more packets in their out-queue move the extra packets arbitrarily to good arcs with empty out-queues (if there are any).

4. As long that there are still arcs with two or more packets in the out-queue, move the extra packets to the out-queues of bad arcs which are chosen at random, i.e., in an arbitrary order, for each such packet, choose uniformly at random a bad dimension with an empty out-queue and move this packet to the queue.

For the sake of the analysis, the color that is associated to a packet is the color it would have if it would exit according to its choice at Step 2, namely the color it would take if it were the only packet in the node. Note that if a packet is associated with a color, then there is a packet which actually advances from this node and has this color.

It is easy to see that Algorithm HYPERCUBE_SINGLE_TARGET is both greedy and cyclic advancing. There is no restriction on the initial number of packets at each node, as long that this number is no more than the node out-degree, i.e., n.

We will show that the number of different colors of packets which are at the same distance to destination, and which were deflected at the previous step, is large. In fact, we will show that for a sufficiently large number of such packets, with high probability there will be exactly n different colors.

Claim 3.7 *Assume that a packet p is to be deflected from a node at distance $r-1$, to one of its neighbors at distance r. Then the color it will be associated with at this neighbor after Step 2 in Algorithm* HYPERCUBE_SINGLE_TARGET *is random with probability $\frac{1}{n}$ for each of the n possible colors.*

Proof: Assume that p was deflected from a node that is at distance $r-1$. There are $n-r+1$ dimensions along which p could be deflected. For each of these dimensions, the probability that p would choose it as its color at the next step is $\frac{1}{n-r+1}$, times the probability that once p is deflected it will choose the same dimension, which is $\frac{n-r+1}{n}$. Thus we get a probability of $\frac{1}{n-r+1} \cdot \frac{n-r+1}{n} = \frac{1}{n}$. For any of the other $r-1$ dimensions the probability that it will be chosen is $\frac{r-1}{n} \cdot \frac{1}{r-1} = \frac{1}{n}$. ∎

Suppose that the algorithm could choose independently for every deflected packet a random color out of the set $\{0, \cdots, n-1\}$. The question "how many choices should be made until we get n different colors?" is known as the "coupon collectors problem". It can be analyzed in various ways. We analyze it here in a slightly more general form, so to be suitable for our needs in section 4 too. Assume we choose out of n colors the color i with probability $p_i \geq p$ and assume that we repeat this process, independently for m choices. We have

Claim 3.8 *The probability that in m choices there will be a color that is not chosen is at most $n(1-p)^m$.*

Proof: The probability that a certain color will not be chosen is at most $(1-p)^m$, summing up for the n colors the claim follows. ∎

Note that in collecting coupons, the choices are taken randomly and independently. In Algorithm HYPERCUBE_SINGLE_TARGET there is a dependency. The color of each previously deflected packet is determined at Step 2 of the algorithm, and is indeed independent for different packets, and for each packet it is $\frac{1}{n}$ for each of the colors. However, this choice may change at steps 3 or 4 of the algorithm. The color is finally determined by the actual arc through which the packet is sent (if the packet advances), or it is irrelevant if the packet will again be deflected. In particular, two packets at the same node may not choose the same color. However, it is quite clear that the dependency above is in our favor. Formally, assume that the colors were chosen sequentially in an arbitrary order. Let X_i' denote the number of colors after i trials in the random process defined by the algorithm and let X_i be the corresponding number in the coupon collector process. Then for any number t, $\Pr(X_i' < t) \leq \Pr(X_i < t)$. This is because initially each color is being chosen with probability p_i for both processes. In the algorithm this color may later be changed if it was chosen by another packet in the same node. In this case the choice is repeated but is not counted. Thus for any instance of the process defined by the algorithm there is a corresponding instance in the coupon collector process for which for each step i the number of colors collected is at most the same as in the algorithm's.

In our case the bound p is $\frac{1}{n}$, and we get,

Lemma 3.9 *If at the beginning of step t there are at least $m+n$ packets at distance r, then with probability at least $1 - ne^{-\frac{m}{n}}$ during step $t+r+1$ exactly n packets will enter the destination.*

Proof: During step $t+r-1$ at most n packets enter the destination. Thus, at least m packets will be deflected at least once during the time interval $[t, t+r-1]$. For each such packet, consider the first deflection during that time interval. In each deflection, the deflected packet chooses a new color, and at least m such choices are made by different packets. As we have explained $\Pr(X_i' < n) \leq \Pr(X_i < n)$, where X_i and X_i' are as before. By claim 3.8 with $p = \frac{1}{n}$ the latter probability is bounded by $\Pr(X_i < n) \leq n(1-\frac{1}{n})^m \leq ne^{-\frac{m}{n}}$. Thus with probability $1 - ne^{-\frac{m}{n}}$ we have n colors among these packets. We may apply Claim 3.6 for each of these deflection steps, each with its own set of colors, which sum up to n. Since by Claim 3.6 colors are preserved, the lemma follows. ∎

We split the packets to *levels* according to their distance to the destination node.

Definition 3.10 *A node z belongs to level i if it is at distance i from the destination node. We say that level $i+1$ is higher than level i, etc. A level will be called empty if there is no packet in any of its nodes. A level will be called heavy during a certain step if there are at least $3n\ln n + n$ packets in this level at the beginning of that step.*

Recall that we consider only levels containing packets with even initial distance to the destination.

The following observation holds for any network with bi-directional communication links and any greedy single-target routing algorithm.

Observation 3.11

- *If x packets are deflected from level d to level $d+1$ during step t, then at least x packets will move from level $d+1$ to level d during step $t+1$. The reason is that for every deflected packet which arrives at a node the node has a good arc.*
- *If at the beginning of step t all levels above the r'th level are empty, then at the end of step $t+1$ all levels above the r'th level are empty.*
- *If at the beginning of step t there are x packets at levels higher than r then after step $t+1$ there are at most x packets at levels higher than r.*

Lemma 3.12 *Let l_0 be the highest (even) heavy level at step 0 (i.e., prior to step 1) and let t_0 be the first step after which there are no more than $n(3n\ln n+n)$ packets. Then at least n packets reach the destination during each even step j, $l_0+2 \leq j \leq t_0$, with probability at least $1-n^{-2}$.*

Proof: In what follows, all references are to even levels. Lemma 3.9 proves that during step l_0+2, n packets arrive with the desired probability. If at the beginning of step 3 level l_0 is still heavy, then Lemma 3.9 promises that with the desired probability n packets will arrive to the destination during step l_0+4. In this way we apply Lemma 3.9 until the first step $2j'$ after which level l_0 is not heavy anymore. There are two cases to consider: Either there are less than $n(3n\ln n+n)$ active packets, and we are done (with $t_0 \leq 2j'$), or that there is a heavy level which is lower than l_0 (By Observation 3.11 the number of packets in levels higher than l_0 does not increase, and this number was below the average of $3n\ln n+n$ packets per level). Let l_1 be the highest heavy level after step $2j'$, which is lower than l_0. If after step $2j'+l_0-l_1$ level l_1 is heavy, then Lemma 3.9 guarantees the arrival of n packets during step $2j'+l_0+2$ (which is the first even step for which such arrival was not guaranteed by the lemma when applied to level l_0). On the other hand, if l_1 is not heavy after step $2j'+l_0-l_1$ then we choose l_2 to be the highest heavy level after step $2j'+l_0-l_1$ which is lower than l_1, and repeat the above arguments. Finally, if level $l_k=2$ is not heavy after step $2j'+l_0-2$, then the total number of active packets in the system at that time is not higher than $n(3n\ln n+n)$, and we may set $t_0=2j'+l_0-2$. ∎

Finally, we are in the position to prove our main theorem.

Theorem 3.13 *Let $\alpha(n)=o(n)$ any function. Then Algorithm* HYPERCUBE_SINGLE_TARGET *routes k packets to a single target on an n-dimensional hypercube in no more than $(1+\frac{\alpha(n)}{n})\frac{2k}{n}+O(n^2\log n)$ steps with probability at least $1-e^{-\frac{\alpha(n)^2k}{8n^3}}$.*

We remark here, that the complexity of the algorithm is asymptotically optimal, up to an additive quantity that is independent of k. Namely, the ratio between the complexity of the algorithm and the trivial lower bound approaches 1 as k grows. In particular, for $\alpha(n)=\theta(\sqrt{n/\log n})$ Algorithm HYPERCUBE_SINGLE_TARGET becomes nearly n times more efficient than what is given by Hajek's analysis for $n^2\log n \ll k$.

Proof: Let l_0 be the highest (even) heavy level prior to step 1. Lemma 3.12 guarantees that starting from step l_0+2 (where $l_0 \leq n$) and until step t_0 after which there are at most $n(3n\ln n+n)=$

$n^2(3\ln n + 1)$ active packets in the system, n packets arrive to the destination at every even step with probability at least $1 - n^{-2}$. If $t_0 < n$ then after step n the rest of the active packets will be dense around the target, hence by Theorem 2.4, by the end of step $n + 2n(3n\ln n + n)$ all packets will arrive. If $t_0 \geq n$ then by Lemma 2.3 at that time the packets are dense around the target, so again by Theorem 2.4 at the end of step $t_0 + 2n(3n\ln n + n)$ all packets will have arrived. Thus, assume from now on that $t_0 \geq n$.

What is left to show is that with high probability $t_0 - l_0 \leq (1 + \frac{\alpha(n)}{n})2\frac{k}{n}$. We are guaranteed by Lemma 3.12 that n packets will arrive to the destination during each individual even step, with probability at least $1 - n^{-2}$. We call such an event, in which n packets arrive to the destination a *success*. The successes for different steps are independent, hence $t_0 - l_0$ is at most twice the number of trials to obtain k/n successes. Namely, we have $r = \frac{k}{n}(1 + \frac{\alpha(n)}{n})$ Bernoulli trials with failure probability $\frac{1}{n^2}$ and we want to calculate the probability to have at least $\frac{k}{n}$ successes. We use the following Chernoff bound [Che52]

$$\Pr(S \leq (1-\delta)m) \leq e^{-\delta^2 m/2}$$

where S is the number of successes in r trials, and $m = r(1 - \frac{1}{n^2})$ is the expectation.

Now, for $\delta = \frac{\alpha(n)}{2n}$ we have that $(1-\delta)m \geq \frac{k}{n}$ for large enough n, thus we get $\Pr(S \leq \frac{k}{n}) \leq \Pr(S \leq (1-\delta)m) \leq e^{-\delta^2 m/2} \leq e^{-\frac{\alpha(n)^2 k}{8n^3}}$. We conclude that for $t_0 - l_0 = (1 + \frac{\alpha(n)}{n})2\frac{k}{n}$ we have k/n successes with probability at least $1 - e^{-\frac{\alpha(n)^2 k}{8n^3}}$. ∎

3.3 Multi-Target Routing on the Hypercube

Suppose a batch of k packets is to be delivered to an ordered set of M targets. We use here a simple version of the idea of Brassil and Cruz [BC91] to apply a single target algorithm to multi-target problems. The idea is to define priority between packets according to the order of the targets. Packets destined to earlier targets in the order get higher priority. Unfortunately, the resulting algorithm is only weakly greedy.

The algorithm is as follows. Let k packets be destined for each one of M targets. We begin by ordering the M destinations in an arbitrary fashion and assume that this ordering is known to each node. Let k_j, $j = 1, 2, \cdots, M$ be the number of packets initially admitted and destined for the jth destination, denoted z_j, so $k = \sum_{j=1}^{M} k_j$. We refer to packets destined to node z_j as *class j* packets.

Algorithm HYPERCUBE_MULTI_TARGET.

Let each node of the hypercube order arriving packets by class from the lowest to the highest numbered class. Starting from the lowest numbered class, packets are assigned to a preferred exit, if available, according to Algorithm HYPERCUBE_SINGLE_TARGET.

The algorithm may not follow the greedy principle when packets from a higher priority class must be deflected in directions which are good for a lower priority class. Notice, however, that it is weakly greedy. As the following theorem shows, it has a good termination time under certain conditions.

Theorem 3.14 *Suppose a batch of k packets is to be delivered to a set of M targets. Let $\alpha(n) = o(n)$ be any function. Then Algorithm* HYPERCUBE_MULTI_TARGET *delivers the packets in at most $(1 + \frac{\alpha(n)}{n})\frac{2k}{n} + O(Mn^2 \log n)$ steps with probability at least $1 - Me^{-\frac{\alpha(n)^2 k}{8n^3}}$.*

Proof: The proof is simply by iterating the result of Theorem 3.13 M times, each time for the next lower priority class. ∎

Similar to Theorem 3.13, Theorem 3.14 is useful only when there are many more packets than targets. Algorithm HYPERCUBE_MULTI_TARGET becomes n times more efficient than Hajek's result when $Mn^2 \log n \ll k$.

4 Routing on Meshes of High Dimensions

In this section we show how to modify the algorithm that was developed for the hypercube in Section 3 in order to generalize it for d-dimensional meshes. The algorithm is d times faster than the best known upper bound, coming close to the lower bound of $\lceil 2k/d \rceil$ steps.

The idea here is the same as in the hypercube case. We wish to assign colors to packets in order to get a claim similar to Claim 3.6. Then once we are able to ensure that we keep a large variety of colors, we are guaranteed that a large number of packets will enter the target at each (second) step and thus achieve the accelerated time.

We first define the concept of *cyclic advancing* for a d-dimensional mesh. The main difference in the definition is that for each vertex, there are usually two directions in each dimension (unless the vertex is on the boundary of the mesh). At most one of the directions may be good, the others are bad. Another difference is that the mesh is not a regular graph. The degree of a node may be between $2d$ for an inner node, to d for a corner node. Still as explained there are at most d good directions, and at most one per dimension, thus the good directions may be referred by their dimensions. Let $G(z)$ denote the set of indices of the good dimensions in a node z. As in the cube we define the *desired direction* for a packet that advanced in dimension i in the previous step to be

$$n(i, z) = (i + \min\{j \geq 0 : (i + j \bmod d) \in G(z)\}) \bmod d$$

Note that, in contrast to the cube, it might be that $n(i, z) = i$. This will happen if the i'th coordinate of the target is not equal to the i'th coordinate of z.

Definition 4.1 *A hot potato algorithm for the d-dimensional mesh is called* cyclic advancing *if when a packet p enters a node z via a good dimension i (i.e., if p was not deflected at the previous step), then a packet will advance from z using the dimension $n(i, z)$ during the next step.*

As in the cube, we define the *color* of a packet to be the dimension of the last arc on its supposed path. With these definitions one may verify easily that Observations 3.3, 3.5, 3.11 and Claim 3.6 hold for the case of the mesh too.

Let $gd(z)$ be the number of good dimensions of a node z, i.e., the number of dimensions in which the node has a good arc. Let $bd(z)$ be the number of bad dimensions of the node z, i.e., the number of dimensions in which the node has a bad arc. Since a dimension may contain both a good and a bad arc, for every node z we have $d \leq gd(z) + bd(z) \leq 2d$.

We now describe the algorithm for routing to a single target on the d-dimensional mesh. Each node knows the distance and the good and bad directions of its neighbors. Each node keeps an out-queue of packets on each of its out-going arcs. The algorithm is described as the collection of operations performed by a node z.

Algorithm MESH_SINGLE_TARGET.

0. At the first step, packets start advancing arbitrarily according to the greedy rule.

1. Add each packet which advanced *into the node through an arc in dimension i to the out-queue of the good arc in the dimension $n(i, z)$.*

2. For each packet which was deflected *by a node z_s into the node z through an arc in dimension i, toss a biased coin and do the following:*

a. With probability $\frac{d-bd(z_s)}{d}$ choose uniformly at random a dimension j which is not a bad dimension in node z_s (in particular i is not chosen) and add the packet to the out-queue of the (good) arc in dimension j.

b. With probability $\frac{bd(z_s)}{d}$ add the packet to the out-queue of the arc through which it arrived, i.e., try to send it on the good arc in dimension i, going "back" to z_s.

Note that in general, not every dimension may be chosen by z. In particular, if $bd(z_s) = d$ then the packet is "returned" to z_s with probability 1.

3. If there are good arcs with empty out-queues and there are good arcs with more than a single packet in the out-queue, move the extra packets from the latter to the former.

4. If there are (good) arcs left with more than a single packet in the out-queue, then deflect the extra packets by assigning them randomly to out-queues of bad arcs. More precisely, this is done as follows: One by one for each packet, choose uniformly at random a dimension which has a bad arc having an empty out-queue, and add the packet to the out-queue of (any) such arc in the chosen dimension (Thus, for the next packet, the random choice may be taken out of a smaller set of dimensions, unless the chosen dimension contains two bad arcs with empty out-queues).

First we note that the algorithm is valid, namely all probabilities are in the interval $[0, 1]$ and can be computed locally by each node. Moreover, the algorithm is greedy and cyclic advancing, and there is no limitation on the initial number of packets in a node, except for the out-degree d. Here too, we show that all colors are chosen by the packets that are deflected with high enough probability.

Claim 4.2 *A packet that is deflected chooses a color at step 2 of the algorithm above with probability at least $\frac{1}{d}$ for every color which is not yet chosen for another packet.*

Proof: When the algorithm deflects a packet from the node z_s to the node z it may be viewed as if it chooses randomly out of the $bd(z_s)$ bad dimensions of z_s (a bad dimension having two bad arcs does not get a higher chance than a one having a single bad arc). The case that some bad directions (or dimensions) may already be taken is to our favor, since it increases the probability to choose the others. Thus, at the deflecting node z_s, each bad dimension is chosen with probability at least $\frac{1}{bd(z_s)}$. Then, when the packet reaches z, the elected dimension becomes the color of the packet with probability $\frac{bd(z_s)}{d}$, hence the a-priori probability to elect it is at least $\frac{1}{d}$.

If there exist dimensions which are not bad in z_s, they may not be chosen by it for deflecting packets, hence they are chosen by the receiving node, z. In this case there are $d - bd(z_s) > 0$ such good dimensions, each chosen with equal probability at the receiving node z. Thus, each dimension which is not bad in z_s is chosen by z with probability $\frac{d-bd(z_s)}{d} \cdot \frac{1}{d-bd(z_s)} = \frac{1}{d}$. ∎

Theorem 4.3 *Algorithm* MESH_SINGLE_TARGET *routes k packets on the d-dimensional mesh with side-length N to a single target in at most* $(1+\frac{\alpha(d)}{d})\frac{2k}{d}+O(Nd^2\log d)$ *with probability which is at least* $1-e^{-\frac{\alpha(d)^2 k}{8d^3}}$.

According to the theorem, when $Nd^2\log d \ll k$ Algorithm MESH_SINGLE_TARGET becomes nearly d times faster than the algorithms in [BC91, Fei94, BRS94].

Proof: The proof follows the line of the proof of Theorem 3.13. We already noticed that observations 3.3, 3.5, 3.11 and Claim 3.6 hold for this case too. By Claim 4.2 we get the following lemma, (which is the adaptation of Lemma 3.9 to our case)

Lemma 4.4 *If at the beginning of step t there are at least $m+2d$ packets at distance r, then at time $t+r+2$ at least d packets will enter the destination, with probability at least* $1-de^{-\frac{m}{d}}$.

Proof:(of the lemma) The proof is exactly as the proof of Lemma 3.9 where we replace the probability $\frac{1}{n}$ with $\frac{1}{d}$, which is the bound relevant in this case, and we replace the degree of the target by an upper bound of $2d$.

Let a *heavy level* at a certain step be a level of nodes which holds at least $3d\ln d+2d$ packets at the beginning of that step. We get the following adaptation of Lemma 3.12.

Lemma 4.5 *Let l_0 be the highest heavy level at step 1 and let t_0 be the first step in which there are no more than $dN(3d\ln d+2d)$ packets. Then at least d packets reach the destination at every even step j, $l_0+2\le j\le t_0-2$, with probability at least* $1-d^{-2}$.

The proof of the lemma is exactly as the proof of Lemma 3.12 where dN replaces n as a bound on the number of levels, and with the proper definition of heaviness.

We complete the proof of the theorem by the same steps as in the proof of Theorem 3.13, plugging in the relevant success probability of Lemma 4.5. ∎

4.1 Multi-Target Routing on Meshes of High Dimensions

Similar to Sec. 3.3 Algorithm MESH_SINGLE_TARGET can be modified for multi-target routing on d-dimensional meshes, especially when there are many more packets than targets. In the following algorithm we use the same terminology as in Sec. 3.3.

Algorithm MESH_MULTI_TARGET.

Let each node of the mesh order arriving packets by class from the lowest to the highest numbered class. Starting from the lowest numbered class, packets are assigned to a preferred exit, if available, according to Algorithm MESH_SINGLE_TARGET.

We get a similar theorem, of which the proof is immediate as in Section 3.3.

Theorem 4.6 *Suppose a batch of k packets is to be delivered to a set of M targets. Then Algorithm* MESH_MULTI_TARGET *routes the packets in at most* $(1+\frac{\alpha(d)}{d})\frac{2k}{d}+O(MNd^2\log d)$ *steps with probability at least* $1-Me^{-\frac{\alpha(d)k}{8d^3}}$.

5 Conclusions and Open Problems

There are two main problems in the hot potato routing theory. The first is to design 'simple' algorithms. The second is to prove that hot potato algorithms, and in particular, 'simple' algorithms, can do better than the obvious. In a network of degree d and diameter D, the second goal is to route a general routing problem with k packets in time $O(D + \frac{k}{d})$ (which is obviously the lower bound). Previous to this work the state of the art was only $O(D + k)$, even for the single target case.

In this work we presented algorithms which achieve this goal for single target routing in the hypercube and in higher dimensional meshes. These results indicate that the goals above, and in particular that of achieving a time inverse proportional to the network degree, might be achievable. We hope that the tools presented here will be useful in further developments in this direction.

References

[AS91] A.S. Acampora and S.I.A. Shah. Multihop Lightwave Networks: a comparison of store-and-forward and hot-potato routing. In *IEEE INFOCOM*, pages 10–19, 1991.

[Bar64] P. Baran. On Distributed Communications Networks. *IEEE Transactions on Communications*, 12:1–9, 1964.

[BC91] 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.

[BH85] A. Borodin and J.E. Hopcroft. Routing, Merging, and Sorting on Parallel Models of Computation. *Journal of Computer and System Sciences*, 30:130–145, 1985.

[BHS94] A. Ben-Dor, S. Halevi, and A. Schuster. Potential function analysis of greedy hot-potato routing. In *Proc. of 13th ACM Symp. on Principles of Distr. Comput.*, Los Angeles, August 1994.

[BNRST93] 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. ACM, 1993.

[BNS94] I. Ben-Aroya, I. Newman, and A. Schuster. Randomized single target hot potato routing. Technical Report LPCR TR #9406, Technion, CS, June 1994.

[BRS94] A. Borodin, Y. Rabani, and B. Schieber. Deterministic many-to-many hot potato routing. Manuscript, 1994.

[BS94] I. Ben-Aroya and A. Schuster. A CLT-type lower bound for hot-potato permutation routing. Technical Report LPCR #9405, CS dept. Technion, May 1994.

[BTS95] I. Ben-Aroya, E. Tamar, and A. Schuster. Greedy hot-potato routing on the two-dimensional mesh. *Distributed computing*, 9(1):3–19, 1995. (Also in proc. 2nd European Symp. on Algorithms, Utrecht, 1994).

[Che52] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Math. Stat.*, 23:493–507, 1952.

[Fei94] U. Feige. Observations on hot potato routing. In *In this proceedings*, 1994.

[FR92] U. Feige and P. Raghavan. Exact Analysis of Hot-Potato Routing. In *Proc. 33rd symp. on of Foundations of Computer Science*, pages 553–562. IEEE, November 1992.

[GG86] A.G. Greenberg and J. Goodman. Sharp Approximate Models of Adaptive Routing in Mesh Networks. In O.J. Boxma, J.W. Cohen, and H.C. Tijms, editors, *Teletraffic Analysis and Computer Performance Evaluation*, pages 255–270. Elsevier, Amsterdam, 1986. Revised 1988.

[GH92] A.G. Greenberg and B. Hajek. Deflection Routing in Hypercube Networks. *IEEE Transactions on Communications*, June 1992.

[Haj91] B. Hajek. Bounds on evacuation time for deflection routing. *Distributed Computing*, 5:1–6, 1991.

[KKR93] C. Kaklamanis, D. Krizanc, and Satish Rao. Hot-Potato Routing on Processor Arrays. In *Proc. 5th symp. on Parallel Algorithms and Architectures*, pages 273–282. ACM, 1993.

[KLS94] M. Kaufmann, H. Lauer, and H. Schröder. Fast deterministic hot-potato routing on meshes. In *proc. 5th symp. on Algorithms and Computation (ISAAC)*, number 834 in LNCS, pages 333–341. Springer-Verlag, 1994.

[Max89] N.F. Maxemchuk. Comparison of deflection and store and forward techniques in the manhattan street and shuffle exchange networks. In *IEEE INFOCOM*, pages 800–809, 1989.

[NS89] J.Y. Ngai and C.L. Seitz. A framework for adaptive routing in multicomputer networks. In *proc. 1st Ann. symp. on parallel algorithms and architectures*, pages 1–9. ACM, 1989.

[NS92] I. Newman and A. Schuster. Hot-potato algorithms for permutation routing. Technical Report LPCR #9201, CS dept. Technion, November 1992.

[Pra86] R. Prager. An algorithm for routing in hypercube networks. Master's thesis, 1986.

[Sei92] C.L. Seitz. The Caltech Mosaic C: an experimental, fine-grain multicomputer. In *4th symp. on Parallel Algorithms and Architectures*, San Diego, June 1992. ACM. Keynote Speech.

[Smi81] B. Smith. Architecture and applications of the HEP multiprocessor computer system. In *Proc. (SPIE) Real Time Signal Processing IV*, pages 241–248, 1981.

[Szy90] T. Szymanski. An analysis of hot potato routing in a fiber optic packet switched hypercube. In *Proc. IEEE INFOCOM*, pages 918–926, 1990.

[ZA91] Z. Zhang and A.S. Acampora. Performance analysis of multihop lightwave networks with hot potato routing and distance age priorities. In *Proc. IEEE INFOCOM*, pages 1012–1021, 1991.