Theory Comput. Systems **31**, 41–61 (1998)

Theory of Computing Systems

Potential Function Analysis of Greedy Hot-Potato Routing*

A. Ben-Dor,[1] S. Halevi,[2] and A. Schuster[1]

[1]Computer Science Department, Technion,
Haifa 32000, Israel
{amirbd,assaf}@cs.technion.ac.il

[2]Laboratory for Computer Science, MIT,
545 Technology Square, Cambridge, MA 02142, USA
shaih@theory.lcs.mit.edu

Abstract. We study the problem of packet routing in synchronous networks. We put forward a notion of *greedy hot-potato routing algorithms* and devise techniques for analyzing such algorithms. A greedy hot-potato routing algorithm is one where:

- The processors have no buffer space for storing delayed packets. Therefore, each packet must leave any intermediate processor at the step following its arrival.
- Packets always advance toward their destination if they can. Namely, a packet must leave its current intermediate node via a link which takes it closer to its destination, unless all these links are taken by other packets. Moreover, in this case all these other packets must advance toward their destinations.

We use potential function analysis to obtain an upper bound of $O(n\sqrt{k})$ on the running time of a wide class of algorithms in the two-dimensional $n \times n$ mesh, for routing problems with a total of k packets. The same techniques can be generalized to obtain an upper bound of $O(\exp(d)n^{d-1}k^{1/d})$ on the running time of a wide class of algorithms in the d-dimensional n^d mesh, for routing problems with a total of k packets.

* A preliminary version was presented at the 13th ACM Symposium on Principles of Distributed Computing, August 1994. The work of the second author was done while at the Computer Science Department, Technion, Israel.

1. Introduction

In this work we study the problem of batch packet routing in synchronous networks in which at most one packet can traverse any directed link in each time step. We consider a class of algorithms known as *hot-potato* or *deflection* routing algorithms [AS], [GG], [GH], [Haj], [LP], [Ma], [Sz], [ZA], [NS1]. The important characteristic of these algorithms 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. This may cause some packets to be "deflected" away from their preferred direction. Such unfortunate situations cannot happen in the traditional "store-and-forward" routing in which a packet is stored at a processor until it can be transmitted to its preferred direction.

Variants of hot-potato routing are used by parallel machines such as the HEP multiprocessor [Sm] and the Connection Machine [Hil] and by high-speed communication networks [Ma]. In particular, hot-potato routing is very important in fine-grained massively parallel computers, such as the Caltech Mosaic C [Se], where the addition of even a small-sized storage buffer at each processor may cause a substantial increase in cost. Another domain in which deflection-type routing is highly desirable is optical networks [AS], [GG], [Sz], [ZA]. 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 with optical transmission rates. It is more feasible to deflect the blocked messages, even if one pays with longer routes.

Most of the recent work on hot-potato routing is focused on *structured* routing. In structured routing "good behavior" is enforced on the packets in the network by sending them in prespecified directions. Although this method was found to guarantee some asymptotically optimal results [NS2] many of the structured algorithms suffer from "overstructuring": Consider for example a packet that originates very close to its destination, then obviously we expect it to reach its destination very fast. This, however, may not be achieved in structured routing: A packet initially very close to its destination might find itself moved to a distant region of the network, due to some obstinate policy that determines a fixed, prespecified route. Moreover, long unnecessary routes may be taken even when the actual number of packets is much smaller than the number of nodes, because the algorithms may not be sensitive to the total load.

Another common problem with structured algorithms is their complexity. These are often designed in *phases*, where the algorithm changes at each phase. Moreover, the routing choices are sometimes complex, and may differ for different processors during the same phase. Thus those structured algorithms rely on complex routing mechanisms which require additional hardware in the processors.

In an attempt to avoid these problem, this paper puts forward the concept of *greedy hot-potato algorithms*. A greedy algorithm is one where packets always advance toward their destination if they can. In other words, whenever some packet is deflected away from its preferred direction, it is because some other packet is currently using that outlink to advance toward its own destination. In this way, unless some global congestion forbids it, packets go in the shortest path to their destinations.

We restrict ourselves further to consider only "simple" greedy algorithms. That is,

in the algorithms we consider, all nodes perform the same (simple) routing policy at each step of the algorithm, from the very first step to termination.[1]

Although greediness might cause congestion in certain regions of the network, deflection is hoped to "spread the load" so that the total routing time is decreased. Indeed, simulations of greedy hot potato routing algorithms present superb performance (see, e.g., [AS] and [Ma]). Unfortunately, the analysis of greedy hot-potato routing algorithms is considerably more difficult than that of structured ones. The difficulty stems from the adaptive nature of the algorithms, as the route that is taken by a packet is changed in an unpredictable manner due to packets it encounters on its way. A packet may find itself in a distant, remote region of the network due to an unexpected sequence of deflections. In fact, certain chains of deflections may eventually result back in the original configuration, thus raising the question whether the algorithm ever terminates. Such infinite loops are called *livelock*.

1.1. *Related Work*

The first greedy hot-potato algorithm was proposed by Baran [Ba]. Borodin and Hopcroft, in a landmark paper [BH], suggested a greedy hot-potato algorithm for the hypercube. Although they did not give a complete analysis of its behavior they observed that "experimentally the algorithm appears promising." During the years, this phenomenon was observed over and over again: *although fairly simple greedy hot-potato algorithms perform very well in simulations, they resist formal analysis attacks*. Numerous experimental results on hot-potato routing have been published [AS], [GG], [GH], [LP], [Ma], [Pr].

Prager [Pr] showed that the Borodin–Hopcroft algorithm terminates in n steps on the 2^n-node hypercube for a special class of permutations. Hajek [Haj] presented a simple greedy algorithm for the same network that runs in $2k + n$ steps, where k is the number of packets in the system. The work of Hajek was simplified and generalized in a work by Brassil and Cruz [BC]. For any regular network with undirected edges (such as the mesh and the hypercube) the algorithm of Brassil and Cruz assumes some prespecified order on the destinations, and packets are given priority according to the rank of their destination in that order. They show a bound of $diam + P + 2(k-1)$, where k is as above, $diam$ is the diameter of the network, and P is the length of a walk connecting all destinations.

Some recent results concern (nongreedy) hot-potato algorithms for permutation routing: Feige and Raghavan [FR] presented an algorithm for the two-dimensional torus, that routes most of the routing problems in no more than $2n + O(\log n)$ steps. Newman and Schuster [NS2] presented an asymptotically optimal algorithm for permutation routing in the two-dimensional mesh. Their algorithm routes every permutation in $7n + o(n)$ steps. This was improved in [KLS] to $3.5n + o(n)$ steps. Bar-Noy *et al*. [BRST] presented a fairly simple algorithm for the two-dimensional mesh and torus, that routes every routing problem in $O(n\sqrt{m})$ steps, where m is the maximum number of packets destined to a single column. In addition, they presented a much more complex algorithm that routes every permutation in the two-dimensional mesh and torus within $O(n^{1+\varepsilon})$ steps (for every $\varepsilon > 0$). Kaklamanis *et al*. [KKR] presented an algorithm for permutation routing

[1] This is by no means the only possible way to define what a "simple" algorithm is. Other notions were suggested e.g., in [BRS].

in the d-dimensional torus that routes most of the permutations within $\frac{1}{2}dn + O(\log^2 n)$ steps, and an algorithm for permutation routing in the two-dimensional mesh that routes most of the permutations within $2n + O(\log^2 n)$ steps.

1.2. *This Work*

In this work we concentrate on many-to-many routing problems in the d-dimensional *mesh connected network*. Our goal is to develop general methodologies for the analysis of greedy hot-potato algorithms. In this spirit, we try to refrain from enforcing further constraints on the algorithm, except for greediness. Unfortunately, it is rather easy to come up with a livelock situation whenever greediness is the only routing policy [NS1], [Haj]. Hence, simple restrictions must be added to the routing mechanisms in order to ensure termination.

We present upper bounds on the running time of a wide class of greedy algorithms. Some of these bounds are tight for some special cases, in the sense that there exist configurations in which *no routing algorithm* can work faster than our bound. The bounds are given in the form of a general method for using potential function analysis, and presenting algorithms and potential functions for meshes.

The method we develop is generic in the sense that once algorithms with better corresponding potential functions are found, the respective bound immediately improves. Indeed, it may very well be possible to find such algorithms for more specific routing problems, or when the network parameters (diameter, degree) improve. Moreover, the general result consists of a worst-case evaluation of some isoperimetric inequality. For specific routing problems this isoperimetric inequality can be shown to improve rapidly, yielding an improved bound.

The rest of this work is organized as follows: in Section 2 we describe our model and give basic definitions. Section 3 describes a general method for potential function analysis of greedy hot-potato routing in meshes. We present an upper bound on the running time of any algorithm for which there exists a potential function that obeys some simple conditions. Section 4 uses this technique to obtain an upper bound of $O(n\sqrt{k})$ on the running time of a wide class of algorithms in the two-dimensional $n \times n$ mesh, for routing problems that contain a total of k packets. In Section 5 we describe how the same method can be generalized to apply for the d-dimensional mesh for $d > 2$. This generalization yields an upper bound of $O(\exp(d)n^{d-1}k^{1/d})$ on the running time of some class of algorithms in the d-dimensional n^d-nodes mesh. Finally, in Section 6 we give some concluding remarks and open problems.

Following the notion of *greedy hot-potato routing algorithms* that was presented in a preliminary version of this work, there were several related results. In Section 6.1 we give a brief overview of these results.

2. Model, Terminology, and Definitions

We consider a network of processors as a graph. The nodes in the graph represent processors and the arcs represent communication links. Since the communication links in the network are bidirectional, every arc in the graph has an antiparallel arc.

In this work we deal with the problem of *many-to-many batch packet routing* in a synchronous network. In this problem there is a set of packets which originate at time $t = 0$ in some nodes of the network. Every packet has a destination node in the network. No node can be the origin of more packets than its out-degree. Note that the many-to-many paradigm neither requires that every node sends a packet, nor that every node is the destination of a packet. Note also that a node may be the destination of many packets.

The networks we consider are synchronous, which means that packets are sent in discrete time steps. The time it takes for a routing algorithm to solve a routing problem is the number of steps that elapse until the last packet reaches its destination.

In the *hot-potato* routing style, a packet cannot stay in a node (other than its destination node), so it must leave every intermediate node in the step that follows its arrival. Hence, every node in the network performs the following scheme in every step:

1. Get the packets that were sent to you in the previous step from your incoming arcs.
2. Make a local computation, which may depend on the sources and destinations of the packets that arrived at the beginning of this step, and on the arcs through which they have entered. In the algorithms we consider in this work, we never use the source of packets in this local computation.
3. According to the results of the local computation, assign outgoing arcs for all these packets.

A *hot-potato routing algorithm* is a collection of such schemes (one for every node in the network) guiding the local computations in these nodes. We are interested in "uniform" algorithms consisting of a single decision rule, that is applied in each step at every node.

2.1. *The d-Dimensional Mesh*

The network we are dealing with is the d-dimensional mesh.

Definition 1. The *d-dimensional mesh* is a graph with a set of n^d nodes that correspond to all d-dimensional vectors over $\{1, \ldots, n\}$. There is an arc between the nodes $\bar{a} = \langle a_1, \ldots, a_d \rangle$ and $\bar{b} = \langle b_1, \ldots, b_d \rangle$ if and only if the L_1 distance between these two vectors is one.

In our discussion the nodes represent processors and we consider each arc as a bidirectional link between two processors. Some obvious properties of the mesh that we use are:

- The diameter of the d-dimensional mesh is $d(n-1)$, and the degree of the nodes in the network is between $2d$ (for interior nodes) and d (for nodes in the corners of the mesh).
- Let $\bar{a} = \langle a_1, \ldots, a_d \rangle$ and $\bar{b} = \langle b_1, \ldots, b_d \rangle$ be two nodes in the d-dimensional mesh, then the distance between $\bar{a}$ and $\bar{b}$ (i.e., the length of the shortest path between them) is $\sum_{i=1}^{d} |a_i - b_i|$.

In this paper we are interested in the routing of packets in the mesh, where each packet has an origin node and a destination node. An important notion for our discussion is the distance between a packet and its destination.

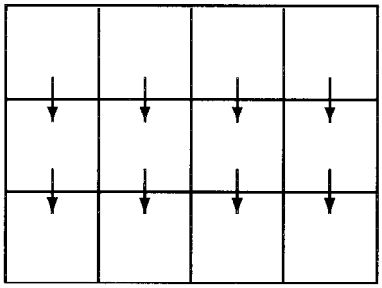

Fig. 1. Direction "−" in the second coordinate in a two-dimensional mesh. The squares in the picture represent nodes, and their faces represent arcs.

Definition 2. Let p be a packet in the mesh. The distance between p and its destination at time t, is the distance between the node that contains p in the beginning of step t and the destination node of p.

Other useful notions are *directions* in the mesh and the 2-*neighbors relation*: since every arc in the mesh connects two nodes with id's that are different in exactly one location, we can divide the arcs into $2d$ distinct directions (e.g., arcs that increase the first coordinate, arcs that decrease the fourth coordinate, etc.)

Definition 3. Direction "+" in the ith coordinate is the set of all arcs in the mesh of the form

$$\langle a_1, \ldots, a_{i-1},\ a_i,\ a_{i+1}, \ldots, a_d\rangle \rightarrow \langle a_1, \ldots, a_{i-1},\ a_i + 1,\ a_{i+1}, \ldots, a_d\rangle.$$

Similarly, direction "−" in the ith coordinate is the set of all arcs in the mesh of the form

$$\langle a_1, \ldots, a_{i-1},\ a_i,\ a_{i+1}, \ldots, a_d\rangle \rightarrow \langle a_1, \ldots, a_{i-1},\ a_i - 1,\ a_{i+1}, \ldots, a_d\rangle.$$

(See Figure 1.)

We sometime say that an arc that goes out from some node is "going in direction X," if that arc belongs to direction X.

Definition 4. We say that a node $\bar{b}$ is a 2-*neighbor* of the node $\bar{a}$ in the direction X, if there is a path of length 2 from $\bar{a}$ to $\bar{b}$ that contains only arcs in the direction X. We say that $\bar{b}$ *is a* 2-*neighbor of* $\bar{a}$ if there is a direction X such that $\bar{b}$ is a 2-neighbor of $\bar{a}$ in the direction X (see Figure 2).

For example, in a two-dimensional mesh, the node $\langle 1, 2\rangle$ is a 2-neighbor of the node $\langle 3, 2\rangle$ (in the direction "−" in the first coordinate), but the node $\langle 2, 3\rangle$ is not a 2-neighbor of the node $\langle 3, 2\rangle$: although there are paths of length 2 between these nodes, none of these paths contains two arcs in the same direction.

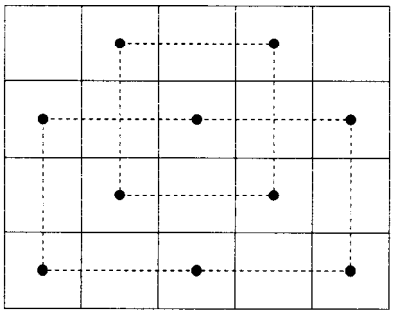

Fig. 2. 2-neighbors in the two-dimensional mesh. Nodes that are connected by dashed lines are 2-neighbors.

From the last definition, it follows that 2-neighbors is a symmetric relation. Therefore, the transitive closure of this relation is an equivalence relation. It is easy to see that it divides the mesh into 2^d equivalence classes, each of which is isomorphic to a d-dimensional mesh with $(\frac{1}{2}n)^d$ nodes (assuming n is even).

2.2. *Greedy Hot-Potato Routing Algorithms*

Definition 5. Let S be a node in the mesh, and let p be a packet in S. We say that an arc that goes out of S is a *good arc* for p if it enters a node that is closer to p's destination. Similarly, we say that an arc that goes out of S is a *bad arc* for p if it enters a node that is farther from p's destination.

We say that a certain direction is a *good direction* for p if there is a good arc for p that goes out of S in that direction. Otherwise, we say that it is a *bad direction*. That is, a bad direction for p either contains a bad arc for p or does not contain any arc that goes out of S (if S is a node on an edge of the mesh).

For example, a packet p in the five-dimensional mesh that is currently in node $\langle 1, 3, 2, 6, 1\rangle$ and its destination is node $\langle 4, 3, 8, 2, 1\rangle$ has three good directions: "+" in the first coordinate, "+" in the third coordinate, and "−" in the fourth coordinate. All the other directions are bad for p.

We say that a packet p *advances* in step t if it gets closer to its destination in that step. Otherwise, we say that p is *being deflected*. When p and q are two packets that are in the same node at the beginning of step t, we say that p is deflected by q (or q is deflecting p) in step t if

(a) p is deflected in step t, and
(b) q is advancing in that step via an arc that is good for p.

In this work we consider *greedy* hot-potato routing algorithms in which a packet always attempts to advance. We make this formal in the following definition.

Definition 6. A hot-potato routing algorithm is said to be *greedy* if, whenever a packet p is being deflected, all its good arcs are used by other advancing packets.

From the definition it follows that in a greedy algorithm, in order to deflect a packet that has i good directions, at least i other packets must advance from the same node.

3. Potential Function Analysis

In the rest of the paper we use potential function analysis to obtain upper bounds on the running time of greedy algorithms in meshes. In this section we present a general method of using potential function analysis to obtain upper bounds. In Sections 4 we use this method to obtain upper bounds for routing algorithms in the two-dimensional mesh, and in Section 5 we describe how this approach can be generalized to the d-dimensional mesh.

3.1. *Potential Function*

For the rest of this section, let $\mathcal{A}$ be a greedy routing algorithm, and suppose that we have a potential function $\varphi_p(t)$ for every packet p, such that $0 \leq \varphi_p(t) \leq M$ in every step t (where M is some positive constant), and that $\varphi_p(t) = 0$ only if packet p reached its destination by step t. Consider the "global" potential function $\Phi(t)$ that is defined by $\Phi(t) \stackrel{\text{def}}{=} \sum_p \varphi_p(t)$. We give a general scheme which yields an upper bound on the running time of $\mathcal{A}$. This scheme uses some local property of the potential function, which we define next. In order to define this property, we first define the notion of a node that *loses potential*.

Definition 7. We say that a node S in the mesh *loses potential* in step t if the sum of potential of the packets that entered S at time t is greater than the sum of potential of the same packets at time $t+1$. We denote the difference between the two sums by $\Delta\Phi_S$.

The local property of Φ that we need can be stated as follows:

Property 8. *Let S be a node in the d-dimensional mesh that contains ℓ packets in step t.*

- *If $\ell \leq d$, then S loses at least ℓ potential units at step t (that is, $\Delta\Phi_S \geq \ell$).*
- *If $\ell > d$, then S loses at least $2d-\ell$ potential units at step t (that is, $\Delta\Phi_S \geq 2d-\ell$).*

We stress that this property must be satisfied in every node of the mesh (even for nodes near the edge of the mesh) in every step.

In the rest of this section we assume that the potential function Φ satisfies the above property with respect to the algorithm $\mathcal{A}$, and show how we can use it to derive an upper bound on the running time of $\mathcal{A}$. In following sections we define specific potential functions which satisfy this property with respect to some classes of algorithms.

3.2. *Analysis*

Let $\mathcal{A}$ be a greedy routing algorithm, and let Φ be a potential function which satisfies Property 8. For the sake of analysis, we divide the nodes in the mesh into "good" nodes and "bad" nodes.

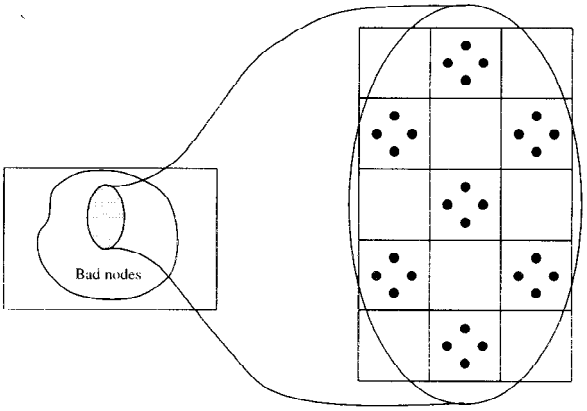

Fig. 3. An area of bad nodes in the two-dimensional mesh.

Definition 9. We say that a node in the mesh is a *bad node* at a certain step if it contains more than d packets at the beginning of that step. Otherwise it is a *good node*. We use $B(t)$ to denote the total number of packets in bad nodes at time t, and $G(t)$ to denote the total number of packets in good nodes at time t.

We can interpret Property 8 as follows: If S is a good node, it loses at least one potential unit for every packet in it. If S is a bad node, it loses at least one potential unit for every "missing" packet. As an immediate corollary we get

Corollary 10. *For every step* t, $\Phi(t+1) \leq \Phi(t) - G(t)$.

In particular, from Corollary 10 it follows that the potential function is a monotonic nonincreasing function. The main difficulty we face is that we do not have any *a priori* lower bound on $G(t)$, as almost all the packets can be in bad nodes at any given time. In order to deal with situations where $G(t)$ is almost zero, we consider the d-dimensional "volume" of bad nodes (see Figure 3), and prove that in every two successive steps, there is a loss of at least one potential unit for every arc on the surface of that volume.

Definition 11. We say that an arc e that goes out of a node S is a *surface arc* if:

- The node S is a bad node.
- Either the 2-neighbor of S in the direction of e is a good node, or S does not have a 2-neighbor in this direction (i.e., if it is on an edge of the mesh).

We also consider an arc that leads "out of the mesh" (in a bad node on the edge of the mesh) as a surface arc (see Figure 4). We denote the number of surface arcs at time t by $F(t)$.

Lemma 12. *For every step* t, $\Phi(t+2) \leq \Phi(t) - F(t)$.

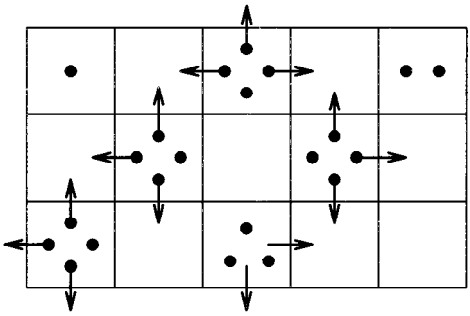

Fig. 4. Surface arcs in the two-dimensional mesh.

Proof. We show that for every surface arc we can account a loss of at least one potential unit either at step t or at step $t+1$. Let S be a bad node, and consider some surface arc that goes out of S. Denote the node in the other side of the surface arc (if exists) by C, and the node in the other side of C in the same direction (if exists) by N:

N is a good node	N
The arc from S to C is a surface arc	C
S is a bad node	S

If no packet is leaving S toward C, then S contains less than $2d$ packets, and thus, by Property 8, S loses at least one potential unit for every "missing" packet at step t. One unit of that loss can be accounted to the arc from S to C. Notice that this case holds even if the node C is "out of the mesh" (i.e., C does not exist).

If there is a packet that leaves S toward C, then we consider two cases:

1. A packet entered C from N at step t. Since N is a good node in step t, it loses one potential unit for every packet in it. Thus, we can account the loss of the packet that entered C to the arc from S to C.
2. No packet entered C from N at step t. In this case, C contains less than $2d$ packets. If C is a good node at time $t+1$, then it loses one potential unit for every packet in it. Thus, we can account the loss of potential caused by the packet that entered C from S, to the surface arc from S to C.

   If C is a bad node, then it loses one potential unit for every "missing" packet. Thus we can account the loss due to the "missing" packet from N to the arc from S to C. Notice that this case holds even if the node N is "out of the mesh" (i.e., N does not exist). □

We now present a geometric interpretation of a d-dimensional mesh: Each node is represented by a d-dimensional unit cube (1^d). The $2d$ faces of each cube correspond to the arcs that go out of the node. Two nodes in the d-dimensional mesh are adjacent if and only if their cubes share a common face.

Let e be a surface arc that goes out of a bad node S, and consider the equivalence class of the node S under the transitive closure of the 2-neighbor relation. Since e is a surface arc, the 2-neighbor of S (in the original mesh) in the direction of e is either a good node, or does not exist at all. Thus, the face that corresponds to the arc e in the equivalence class of S has a bad node in one side (S itself) and a good node (or no node at all) on the other side.

Therefore, if we consider the d-dimensional volume that is composed of the bad nodes in every equivalence class, then every face in the surface of that volume corresponds to a surface arc in the mesh. In other words, the surface of the "bad volume" of a certain equivalence class is equal to the number of surface arcs that go out of nodes in this class.

We show that the surface of any d-dimensional volume V that is composed of d-dimensional unit cubes is at least $2dV^{(d-1)/d}$.

Claim 13. *Any d-dimensional volume V that is composed of d-dimensional unit cubes has a surface of size at least $2dV^{(d-1)/d}$.*

Proof. For every subset of the dimensions, $I \subseteq \{1, \ldots, d\}$, we denote by $\pi_I(V)$ the projection of V on the dimensions in I. That is, $\pi_I(V)$ is the image of V in a $|I|$-dimensional plane. For example, in the three-dimensional grid ($d = 3$) $\pi_{\{1,2\}}(V)$ is the projection of V on the XY-plane, and $\pi_{\{2,3\}}(V)$ is the projection of V on the YZ-plane.

Each point in the projection of V on any $(d-1)$-dimensional plane contributes at least two to the surface, as follows: Assume without loss of generality that $I = \{1, 2, \ldots, d-1\}$, and consider a point $\bar{x}$ in the projection $\pi_I(V)$. Let $x_{\min}$ and $x_{\max}$ be the minimal and the maximal values, correspondingly, so that $(\bar{x}, x_{\min})$ and $(\bar{x}, x_{\max})$ are in V. Hence, $(\bar{x}, x_{\min} - 1)$ and $(\bar{x}, x_{\max} + 1)$ are not in V. The two $(d-1)$-dimensional faces between the two points in V and the two points outside V, contribute a unit each to the surface. Thus, we get

$$\text{surface}(V) \geq 2 \cdot \sum_{|I|=d-1} |\pi_I(V)|. \tag{1}$$

Consider now the random vector $\bar{X} = (X_1, X_2, \ldots, X_d)$ that is defined over V with uniform distribution, i.e.,

$$\Pr(\bar{X} = \bar{x}) = \begin{cases} \dfrac{1}{|V|}, & \bar{x} \in V, \\ 0, & \bar{x} \notin V. \end{cases}$$

For every $I \subseteq \{1, \ldots, d\}$ let $\bar{X}_I$ denote the random vector $(X_i : i \in I)$. From the definitions of $\bar{X}_I$ and $\pi_I(V)$ it is clear that $|\pi_I(V)|$ is exactly the number of different values the vector $\bar{X}_I$ can get. Therefore,

$$H(\bar{X}_I) \leq \log|\pi_I(V)|, \tag{2}$$

where H is the entropy function. On the other hand, since $\bar{X}$ is distributed uniformly over V, then

$$H(\bar{X}) = \log|V|. \tag{3}$$

Using an entropy inequality that was proven in [CGFS], we get that (for any d-dimensional random vector)

$$(d-1)H(\bar{X}) \leq \sum_{|I|=d-1} H(\bar{X}_I). \tag{4}$$

Combining this with (2) and (3), we get

$$(d-1)\log|V| \leq \sum_{|I|=d-1} \log|\pi_I(V)| \quad \Rightarrow \quad |V|^{d-1} \leq \prod_{|I|=d-1} |\pi_I(V)|. \tag{5}$$

From (1) and the arithmetic-geometric mean inequality, we have

$$\text{surface}(V) \geq 2 \cdot \sum_{|I|=d-1} |\pi_I(V)| \geq 2d \left(\prod_{|I|=d-1} |\pi_I(V)| \right)^{1/d}.$$

Now from (5) we conclude that $\text{surface}(V) \geq 2d \cdot |V|^{(d-1)/d}$. □

Using Claim 13 and our geometric interpretation of the mesh, we now show that if there are many bad nodes, then there must also be many surface arcs.

Lemma 14. *If there are $B(t)$ packets in bad nodes at the beginning of step t, then the number of surface arcs in that step is at least $(2d)^{1/d} \cdot B(t)^{(d-1)/d}$.*

Proof. Recall that the n^d mesh can be divided into 2^d equivalence classes of the 2-neighborhood relation. Each equivalence class is isomorphic to an $(n/2)^d$ d-dimensional mesh (assuming n is even). Two nodes in such an equivalence class are adjacent iff they are 2-neighbors in the original mesh.

Since there are at most $2d$ packets in every bad node, the number of bad nodes is at least $B(t)/2d$. Therefore we have 2^d d-dimensional volumes of bad nodes (one for every equivalence class) of total volume $V = B(t)/2d$. Every face in the surface of each of these volumes corresponds to a different surface arc in the mesh. From Claim 13 it follows that the number of surface arcs is at least

$$2d \cdot \left(\frac{B(t)}{2d} \right)^{(d-1)/d} = (2d)^{1/d} \cdot B(t)^{(d-1)/d}.$$ □

Lemma 15. *If the potential at the beginning of step t is $\Phi(t)$, then in the following two steps the potential is decreased by at least $(2d)^{1/d}(\Phi(t)/2M)^{(d-1)/d}$ (where M is the a priori bound on the potential of every packet).*

Proof. From Corollary 10 we get that

$$\Phi(t+2) \leq \Phi(t+1) \leq \Phi(t) - G(t).$$

From Lemmas 12 and 14 we also get that

$$\Phi(t+2) \leq \Phi(t) - (2d)^{1/d} \cdot B(t)^{(d-1)/d}.$$

We denote the number of packets in the mesh at time t by $L(t)$, then $L(t) = B(t) + G(t)$. Therefore, we have that

$$\Phi(t) - \Phi(t+2) \geq \max\{L(t) - B(t),\ \ (2d)^{1/d} \cdot B(t)^{(d-1)/d}\}.$$

This bound is minimal when

$$L(t) - B(t) = (2d)^{1/d} \cdot B(t)^{(d-1)/d}. \tag{6}$$

Denote by $B_0(t)$ the value of $B(t)$ for which (6) holds.

Claim 16. $B_0(t) \geq \frac{1}{2}L(t)$.

Proof. Notice that the left-hand side of (6) decreases as a function of $B(t)$, and the right-hand side of this equation increases as a function of $B(t)$. Therefore, in order to show that $\frac{1}{2}L(t)$ is a lower bound for $B(t)$, it suffice to show that under the assignment $B(t) = \frac{1}{2}L(t)$, the left-hand side of (6) is no less than its right-hand side. We consider two cases:

1. $L(t) \geq 4d$. In this case, using some algebra, we get

   $\frac{1}{2}L(t) \geq (2d)^{1/d}(\frac{1}{2}L(t))^{(d-1)/d}$

   and therefore $L(t) - B(t) \geq (2d)^{1/d} \cdot B(t)^{(d-1)/d}$.
2. Otherwise, $L(t) < 4d$. Since there are less than $4d$ packets, then the number of bad nodes is at most three. An easy (though tedious) case analysis (for 0, 1, 2, and 3 bad nodes) shows that the claim hold for this case too. □

From Claim 16 it follows that

$$\begin{aligned}\max\{L(t) - B(t),\ \ (2d)^{1/d} \cdot B(t)^{(d-1)/d}\} &\geq (2d)^{1/d} \cdot B_0(t)^{(d-1)/d} \\ &\geq (2d)^{1/d} \cdot (\tfrac{1}{2}L(t))^{(d-1)/d}. \end{aligned}\tag{7}$$

Thus we have

$$\Phi(t) - \Phi(t+2) \geq (2d)^{1/d}(\tfrac{1}{2}L(t))^{(d-1)/d}.$$

Since a packet in the mesh can have at most M units of potential, we get that $\Phi(t) \leq M \cdot L(t)$, or $L(t) \geq \Phi(t)/M$. Therefore,

$$\Phi(t) - \Phi(t+2) \geq (2d)^{1/d}\left(\frac{\Phi(t)}{2M}\right)^{(d-1)/d}. \qquad \square$$

Theorem 17. *If $\mathcal{A}$ is a routing algorithm and Φ is a potential function which satisfies Property 8, then $\mathcal{A}$ solves every routing problem with k packets in the d-dimensional mesh within at most $(4d)^{1-1/d} \cdot k^{1/d} \cdot M$ steps.*

Proof. Our potential function satisfies the following conditions:

1. At the beginning of the algorithm, $\Phi(0) = \Phi_0 \leq k \cdot M$, since there are k packets and each packet has at most M units of potential.
2. By Lemma 15, $\Phi(t+2) \leq \Phi(t) - (2d)^{1/d} \cdot (\Phi(t)/2M)^{(d-1)/d}$.

We ask how long it will take Φ to reach zero. We divide the reduction of Φ into phases. During the ith phase, Φ decreases from $\Phi_0/(1+\varepsilon)^i$ to $\Phi_0/(1+\varepsilon)^{i+1}$ (where ε is some small positive constant). At the end of the last phase, the potential is decreased to zero. The total decrease in the potential during the ith phase is

$$\frac{\Phi_0}{(1+\varepsilon)^i} - \frac{\Phi_0}{(1+\varepsilon)^{i+1}} = \varepsilon \frac{\Phi_0}{(1+\varepsilon)^{i+1}}. \tag{8}$$

During the ith phase, the potential is at least $\Phi_0/(1+\varepsilon)^{i+1}$. From Lemma 15, we know that the potential decreases in every two steps by at least

$$(2d)^{1/d} \left[\frac{\Phi_0}{(1+\varepsilon)^{i+1} \cdot 2M} \right]^{(d-1)/d}. \tag{9}$$

Thus, an upper bound for the length of the ith phase can be obtained by dividing (8) by (9). Hence, the number of steps we need during the ith phase is at most

$$\begin{aligned} &2 \frac{\varepsilon(\Phi_0/(1+\varepsilon)^{i+1})}{(2d)^{1/d}[\Phi_0/2M(1+\varepsilon)^{i+1}]^{(d-1)/d}} \\ &\quad = \frac{2}{(2d)^{1/d}} \cdot \Phi_0^{1/d} \cdot (2M)^{(d-1)/d} \cdot \varepsilon (1+\varepsilon)^{-(i+1)/d} \end{aligned}$$

(the factor of two is because (9) describes the potential drop in two steps). We denote $C \stackrel{\text{def}}{=} (2/(2d)^{1/d}) \cdot \Phi_0^{1/d} \cdot (2M)^{(d-1)/d}$, then the total number of steps until the potential reaches zero is at most

$$\begin{aligned} &C \cdot \varepsilon \sum_{i=0}^{\infty} (1+\varepsilon)^{-(i+1)/d} \\ &\quad = C \cdot \varepsilon \sum_{j=0}^{\infty} [(1+\varepsilon)^{-(j+1/d)} + (1+\varepsilon)^{-(j+2/d)} + \cdots + (1+\varepsilon)^{-(j+(d-1)/d)} \\ &\qquad\qquad + (1+\varepsilon)^{-(j+1)}] \\ &\quad \leq C \cdot \varepsilon \sum_{j=0}^{\infty} d(1+\varepsilon)^{-(j+1/d)} = C \cdot \varepsilon (1+\varepsilon)^{-1/d} \cdot d \sum_{j=0}^{\infty} (1+\varepsilon)^{-j} \\ &\quad = C \cdot d \cdot (1+\varepsilon)^{(d-1)/d} \stackrel{\varepsilon \to 0}{\longrightarrow} C \cdot d = (2d)^{(d-1)/d} \cdot \Phi_0^{1/d} \cdot (2M)^{(d-1)/d}. \end{aligned}$$

Since $\Phi_0 \leq k \cdot M$, then $\Phi_0^{1/d} \leq k^{1/d} \cdot M^{1/d}$, and, thus, the total number of steps required before the potential is decreased to zero is at most $(4d)^{1-1/d} \cdot k^{1/d} \cdot M$. □

4. Potential Function in the Two-Dimensional Mesh

In this section we show a potential function in the two-dimensional mesh which satisfies Property 8, and use Theorem 17 to obtain an upper bound on the running time of a

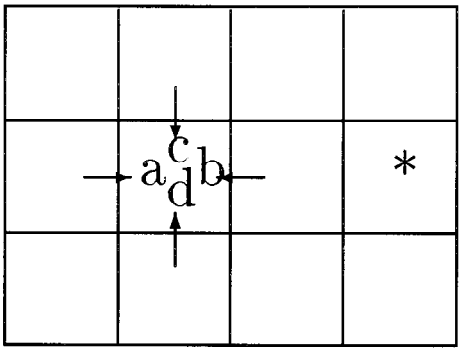

Fig. 5. Restricted packets: *b*, *c*, and *d* are of type B, and *a* is of type A.

large class of greedy algorithms. To this end, we need to restrict somewhat the routing algorithm.

4.1. *Priority to Restricted Packets*

We say that a packet in the two-dimensional mesh is *restricted* if it has exactly one good direction. We divide the restricted packets into two types (see Figure 5):

Type A. Packets that were restricted in the previous step, and advanced in it.

Type B. All the other packets, i.e., either packets that were deflected in the previous step, or packets that were not restricted in the previous step.

Definition 18. We say that a greedy routing algorithm *prefers restricted packets* if a nonrestricted packet cannot deflect a restricted one. This implies that whenever a restricted packet is deflected, the packets that deflect it must also be restricted.

For the definition of the potential function, we need the following properties of algorithms that prefer restricted packets:

1. A restricted packet can deflect at most one type A packet at every step.
2. If a type A packet, q, is deflected by another restricted packet, p, then p must be of type B.

4.2. *Definition of the Potential Function*

We now show that a routing algorithm in the two-dimensional mesh which prefers restricted packets routes any routing problem with k packets in at most $8\sqrt{2} \cdot n\sqrt{k}$ steps.

To this end, we define the potential function as follows: The potential of a packet p in step t includes the distance from p to its destination, denoted by $dist_p(t)$, and some additional amount of potential. Changes in the potential are depicted in Figure 6.

Initially, every packet p has $2n$ additional units of potential. As long as p is not

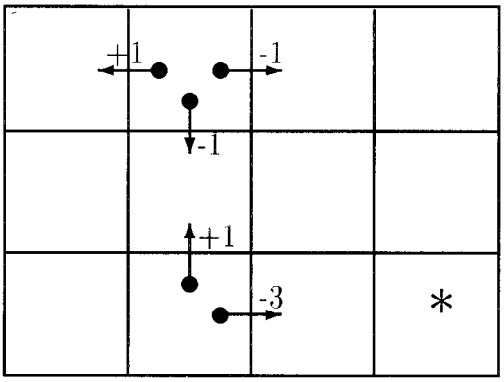

Fig. 6. Changes in the potential of packets in one step.

of type A, its additional potential remains fixed. When p becomes a type A packet, it "drops" two units of its additional potential in every step.

If a type A packet is deflected by some other packet, then the two packets "switch" their additional potential, and the advancing packet drops two units of the potential it has just received from the deflected packet.

Formally, we denote the amount of additional potential of packet p after step t by $C_p(t)$.

1. Initially, for every p, $C_p(0) = 2n$.
2. If after step t packet p is not restricted, or if it is a restricted packet of type B, then $C_p(t) = 2n$.
3. If after step t packet p is a restricted packet of type A, then there are two cases:
   (a) p did not deflect any restricted packet of type A in step t. In this case $C_p(t) = C_p(t-1) - 2$.
   (b) p deflected a restricted packet of type A. Denote this packet by q (there is exactly one such packet). In this case $C_p(t+1) = C_q(t) - 2$.
4. If p has reached its destination by step t, then $C_p(t) = 0$.

From the properties of algorithms which prefer restricted packets, it follows that in case 3(b), packet p was a type B packet at the beginning of step t. Therefore, p had $2n$ units of additional potential. At the end of step t, packet q had $2n$ units of additional potential (since it was deflected), and packet p has $C_q(t-1) - 2$ units of additional potential. Thus, the sum of additional potential of packets p and q is $2n + C_q(t-1) - 2$. This is exactly the same amount we would have got if q would have deflected p.

Thus, as far as the potential function is concerned, it does not matter whether p deflects q or vice versa. Therefore, when we analyze the changes in the potential of a node, we can always assume that the type A packets are advancing.

The potential of a packet p at the beginning of step t is $\varphi_p(t) \stackrel{\text{def}}{=} dist_p(t) + C_p(t)$. The potential of the mesh at the beginning of step t is $\Phi(t) \stackrel{\text{def}}{=} \sum_p \varphi_p(t)$. We proceed to show that the potential function Φ satisfies Property 8.

Lemma 19. *Consider an algorithm which prefers restricted packets, and let S be a node in the mesh. If S contains $\ell \leq 2$ packets in step t, then it looses at least ℓ units of potential in that step, and if S contains $\ell > 2$ packets in step t, then it looses at least $4 - \ell$ units of potential in that step.*

Proof. We consider two cases:

- S does not contain any restricted packet. Thus every packet in S has two good directions. As it takes two advancing packets to deflect one nonrestricted packet, it follows that if $\ell \leq 2$, then all the packets in S will advance. If $\ell > 2$, then at least two packets will advance from S, and at most $\ell - 2$ will be deflected.

  Since the additional potential of nonrestricted packets is fixed, only the "distance-potential" of the packet in S is changed. If $\ell \leq 2$ we get $\Delta_S \geq \ell$, and if $\ell \geq 3$ we get $\Delta_S \geq 2 - (\ell - 2) = 4 - \ell$.
- S contains restricted packets, so at least one restricted packet, p, will advance from S. Assume, without loss of generality, that no restricted packet of type A is deflected from S, so rule 3(b) above is not applied. Therefore, the potential of p will be decreased by three units. The potential of each of the other packets in S is increased by at most one unit, and the lemma follows. □

Since the potential function satisfies Property 8, and $0 \leq \varphi_p(t) \leq 4n$, we can use Theorem 17 with $d = 2$ and $M = 4n$ to obtain

Theorem 20. *Every greedy routing algorithm that prefers restricted packets solves every routing problem with k packets within at most $8\sqrt{2} \cdot n\sqrt{k}$ steps.*

Remark. A hot-potato routing problem in the mesh can be split into two independent problems according to the parity of the packets origin. These problems do not interfere with one another. Therefore, if every node is the origin of a single packet (that is $k = n^2$), we can strengthen the result a little, getting a bound of $8n^2$. Also, when every node is the origin of four packets, we get a bound of $16n^2$, which is only eight times the lower bound.

5. Potential Functions in the d-Dimensional Mesh

In this section we briefly describe how to generalize the method of Section 4 to the d-dimensional mesh. This generalization includes fairly complex technical details, and the bound that is obtained this way deteriorates exponentially with d. Therefore, at the advice of the referees, we choose to omit the detailed proof of this bound and to give only the ideas here. The full proof can be found in [Hal] and [BHS] or can be obtained directly from the authors.

We start by generalizing the notion of a *restricted packet*. Since packets in the d-dimensional mesh can have more that two good directions, we classify them by the number of good directions they have. Just like in the two-dimensional case, we divide

packets with the same number of good directions into "type A" and "type B" packets, according to whether or not they advanced in the previous step.

The natural generalization of algorithms which prefer restricted packets are algorithms which prefer packets with less good directions. A technical difficulty that arises here is that packets can deflect each other even if they do not have the exact same good directions, and this causes a problem in the proof. To fix this problem, we need to restrict the algorithm a little further and to require that it maximizes the number of advancing packets from every node.

Then we define a potential function which is a natural generalization of the one in the two-dimensional case. Namely, each packet has a "load of spare potential" from which it throws as it advances. The amount of "spare potential" it throws is chosen so that it can compensate for all the packets it may deflect.

Using the ideas above and working out the technicalities, we can show an upper bound of $4^{d+1-1/d} \cdot d^{1-1/d} \cdot k^{1/d} \cdot n^{d-1}$ on the running time of a natural class of greedy hot-potato algorithms in the d-dimensional mesh.

6. Conclusions and Open Problems

In this work we introduced the notion of greedy hot-potato routing algorithms in meshes. We developed a technique for the analysis of such algorithms using potential functions. Using this technique we have shown an upper bound of $8\sqrt{2}n\sqrt{k}$ on the running times of a large class of greedy hot-potato routing algorithms in the two-dimensional mesh. This can be generalized to obtain an $O(\exp(d)k^{1/d}n^{d-1})$ bound for algorithms in d-dimensional meshes.

Two important ingredients in our method are as follows:

- The specific routing algorithm and the corresponding potential function. These are to be plugged in the general result (see Theorem 17).
- The isoperimetric inequality to be plugged in Lemma 14, and for which the worst case is evaluated in Section 4.

An important observation here is that once better algorithms with better corresponding potential functions are found, the respective bound immediately improves. It is likely that such algorithms are found for more specific routing problems, or due to a change in the network parameters (diameter, degree).

Another improvement in the bound can be obtained when a better isoperimetric inequality is found. For example, a much better bound than that derived in Section 4 can be shown when the maximal distance to destination is small. However, in order to complete this observation to a better routing time of a small distance routing algorithm, it is still left to show that packets are not deflected much further, which seems to be a harder task.

When the dimension of the mesh increases, routing terminate faster due to the existence of more communication links and paths. Unfortunately, for d-dimensional meshes, our bound gets worse. Yet, as is explained in the above discussion, improvement can be obtained by finding better algorithms and potential functions, using the extra paths.

The dependence of our result on the number of packets in the system is suboptimal. A natural open problem is to improve the bound for sparse requests where the number of packets is small, i.e., $k \ll n^d$.

An interesting problem is to present upper bounds on the worst-case running time of a greedy algorithm for routing a permutation of messages. In such routing requests each node is the origin and the destination of at most one packet. Intuitively, permutation routing should terminate faster than the single destination case. For nongreedy routing there exists a well-developed theory of such algorithms, where most lower bounds are matched up to small constant factors, see [NS2] and [KLS]. In the greedy domain, however, although simulations predict superb performance, there were no better upper bounds than those presented in [BC] and in this work. After this work was completed there has been some progress in this direction, see below.

6.1. *Postscript*: *Following this Work*

Following the notion of *greedy hot-potato routing algorithms* introduced in this work there were several related results. In this subsection we give a brief overview of these results.

Ben-Aroya *et al*. showed a greedy algorithm on the two-dimensional mesh for routing a batch of k packets with maximal source-to-destination distance $d_{\max}$ in $2(k-1)+d_{\max}$ steps [BTS]. Independently, this bound was also obtained by Feige [Fe] and by Borodin *et al*. [BRS]. Furthermore, the bound in [BRS] holds also for higher-dimensional meshes. (The bound in [Fe] also holds for higher-dimensional meshes, however, the algorithm does not remain greedy.)

[Fe] and [BTS] define a stronger notion of greed. [Fe] contains several upper and lower bounds for greedy and strongly greedy routing algorithms. [BTS] presents a greedy single-target algorithm (all k packets destined to the same node) which exactly matches the lower bound of $d_{\max}+k$ in the two-dimensional mesh. [BTS] gives a seven-step greedy algorithm for routing permutations with $d_{\max}=3$ (a five-step algorithm in [Fe] for the same problem is nongreedy). A folklore algorithm achieves $O(n^2)$ routing time on the 2^n hypercube for the same problem [Bo].

Ben-Aroya *et al*. gave a randomized single-target greedy routing algorithm on d-dimensional meshes and the n-dimensional hypercube [BNS]. As far as we know this is the only greedy hot-potato routing algorithm for which the bound improves when higher dimensions are considered (i.e., the algorithm utilizes the higher in-degree of the nodes).

Borodin *et al*. were the first to obtain an $o(k)$ permutation greedy routing algorithm [BRS]. They observed that their greedy algorithm, which achieves $2k+d_{\max}$ on any routing instance, obeys the conditions for the analysis in [BRST], thus proving that it terminates in $O(n^{1.5})$ steps for permutation routing. It was recently claimed that this is also the lower bound for this algorithm with respect to permutation routing [Sy].

Ben-Aroya *et al*. gave lower bounds on the permutation routing time for adaptive algorithms which attempt to keep the paths to destinations nearly minimal [BCS]. They extend their lower bounds for hot-potato routing, and show a specific greedy algorithm that gives priority to restricted packets and which requires $\Omega(n^2)$ steps to route some worst-case permutations on the $n \times n$ mesh. This proves that the analysis that is given in this work (and that applies for all routing instances) is the best possible with respect

to the class of algorithms that give priority to restricted packets, even when the set of routing instances consist of permutations only.

Acknowledgments

We wish to thank Allan Borodin and Ilan Newman for helpful discussions, and to Roy Meshulam for helping us with the proof of the isoperimetric inequality in Section 4. We also wish to thank the anonymous referees for many very helpful comments.

References

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

[Ba] P. Baran. On distributed communications networks. *IEEE Transactions on Communications*, 12:1–9, 1964.

[BC] 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.

[BCS] I. Ben-Aroya, D.D. Chinn, and A. Schuster. A lower bound for nearly minimal adaptive and hot potato routing algorithms. To appear in *Algorithmica*. Preliminary version appeared in *Proc. 4th European Symp. on Algorithms*, Barcelona, Sept. 1996, pages 471–485.

[BH] 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.

[BHS] 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*, pages 225–234, Los Angeles, August 1994. ACM, New York. (Also Technion/LPCR TR #9303, January 1993.)

[BNS] I. Ben-Aroya, I. Newman, and A. Schuster. Randomized single target hot potato routing. *Journal of Algorithms*, 23:101–120, 1997. (Also in *Proc. 3rd Israeli Symp. on Theory of Computing and Systems*, January 1995, pp. 20–29.)

[Bo] A. Borodin. Private communication.

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

[BRST] 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, New York, 1993.

[BTS] 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, pages 365–376, 1994.)

[CGFS] F.R.K. Chung, R.L. Graham, P. Frankl, and J.B. Shearer. Some intersection theorems for ordered sets and graphs. *Journal of Combinatorial Theory, Series A*, 43:23–37, 1986.

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

[FR] 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, New York, November 1992.

[GG] 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.

[GH] A.G. Greenberg and B. Hajek. Deflection routing in hypercube networks. *IEEE Transactions on Communications*, June 1992.

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

[Hal] Shai Halevi. On greedy hot potato routing. Master's thesis (in Hebrew), Computer Science Department, Technion, July 1993.

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

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

[KLS] M. Kaufmann, H. Lauer, and H. Schröder. Fast deterministic hot-potato routing on meshes. In *Proc*. 5*th Symp*. *on Algorithms and Computation* (*ISAAC*), pages 333–341. LNCS, volume 834. Springer-Verlag, Berlin, 1994.

[LP] D.H. Lawrie and D.A. Padua. Analysis of message switching with shuffle-exchanges in multiprocessors. In *Proc*. *Workshop on Interconnection Networks for Parallel and Distributed Computing*, pages 116–123, 1980.

[Ma] 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, 1989.

[NS1] J.Y. Ngai and C.L. Seitz. A framework for adaptive routing in multicomputer networks. In *Proc*. 1*st Symp*. *on Parallel Algorithms and Architectures*, pages 1–9. ACM, New York, 1989.

[NS2] I. Newman and A. Schuster. Hot-potato algorithms for permutation routing. *IEEE Transactions on Parallel and Distributed Systems*, 6(11):1168–1176, 1995.

[Pr] R. Prager. An algorithm for routing in hypercube networks. Master's thesis, Computer Science Department, University of Toronto, 1986.

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

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

[Sy] A. Symvonis. Private communication.

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

[ZA] 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.

Received December 1993, *and in final form March* 1997.