Potential Function Analysis of Greedy Hot-Potato Routing EXTENDED ABSTRACT

Amir Ben-Dor* Shai Halevi† Assaf Schuster‡

January 21, 1994

Abstract

In this work we study the problem of packet routing in synchronous networks of processors, in which at most one packet can traverse any communication link in each time step.

We consider a class of algorithms known as *hot-potato* or *deflection* routing algorithms. The important characteristic of these algorithms is that they use no buffer space for storing delayed packets. Each packet, unless already arrived to its destination, must leave the processor at the step following its arrival. The main advantage in hot-potato routing is that there is no need to store delayed packets in the processors, and therefore, the processors can be much simpler, and contain less hardware.

This work is concerned with *greedy* routing, in which a packet is bound to use an out-going link in the direction of its destination, whenever such a link is available. In this way, greediness guarantees that, unless some global congestion forbids it, packets go in the shortest path to their destinations.

We present upper bounds on the worst-case running time of greedy hot-potato routing algorithms in the 2-dimensional and d-dimensional mesh. We consider the general case of *many-to-many* routing problems. A novel technique for potential function analysis is presented. We use this technique to obtain an upper bound of $O(n\sqrt{k})$ on the running time of a wide class of algorithms in the 2-dimensional $n \times n$ mesh, for routing problems that contain k packets. We then generalize this analysis 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 that contains k packets.

Our methods give hope for further improvements by presenting a framework for the analysis of algorithms and potential functions. Indeed, there are many features of a specific network which may be utilized by an algorithm. Once better algorithms and corresponding potential functions are found, the analysis show the improvement of the bounds.

*E-mail: amirbd@cs.technion.ac.il

†E-mail: shaih@theory.lcs.mit.edu

‡Contact address: Computer Science Dept., Technion, Haifa 32000, ISRAEL. E-mail: assaf@cs.technion.ac.il

1 Introduction

In this work we study the problem of batch packet routing in synchronous networks of processors 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 [AS91, GG86, GH92, Haj91, LP80, Max89, Szy90, ZA91, NS89]. The important characteristic of these algorithms is that they use no buffer space for storing delayed packets. Each packet, unless already arrived to its destination, must leave the processor at the step following its arrival. Packets may arrive to a processor from all its neighbors and have to be redirected each on a different outgoing link. This may cause some packets to be "deflected" away from their preferred direction. Such unfortunate situation cannot happen in the traditional "store-and-forward" routing in which a packet can be 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 [Smi81] and the Connection Machine [Hil85] and by high-speed communication networks [Max89]. In particular, hot potato routing is very important in fine-grained massively-parallel computers, such as the Caltech Mosaic C [Sei92]. For such machines the addition of even a small sized storage buffer at each processor will cause 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 rates. It is more feasible (even if one pays with longer routes) to deflect the blocked messages.

Most of the recent work on hot potato routing focus on *structured* routing. In structured routing "good behavior" is enforced on the packets in the network by sending them in pre-specified directions. Although this method was found to guarantee some asymptotically optimal results [NS92], it has a fundamental flaw: Consider a packet which originate very close to its destination, then obviously we expect it to reach its destination very fast. This, however, is not 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 which determines a fixed, pre-specified route. Moreover, long unnecessary routes are taken even when the actual number of packets is much smaller than the number of nodes, because the algorithms are not sensitive to the total load. In contrast, in *greedy* routing a packet is bound to use an out-going link in the direction of its destination whenever such a link is available. In other words, whenever some packet is *deflected* away from its preferred direction, it is because some other packet is currently using this out-link. In this way, greediness guarantees that, unless some global congestion forbids it, packets go in the shortest path to their destinations.

Another practical aspect of greedy routing is its simplicity. The amount of hardware and the cycle time of the routing mechanism (at every processor) depend to a large extent on the complexity of the routing algorithm. Structured algorithms are often designed in *phases*, where the algorithm changes at each phase. Moreover, the routing choices are usually complex, and might differ for different processors during the same phase. Thus structured algorithms typically rely on complex routing mechanisms. Greedy algorithms, on the other hand, do not consist on phases. All nodes perform the same (simple) routing policy at each step of the algorithm, from the very first step to termination, thus ensuring the simplicity of the routing hardware.

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 show their superiority over the structured ones. In fact, the simulation results are so great, one must be converted to greediness when one sees them. Unfortunately, the analysis of greedy hot potato routing algorithms is considerably more difficult than that of the structured ones. The difficulty stems from the adaptive nature of the algorithms:

- The specific route taken by a packet is changed due to packets it encounters on its way.
- The length of the route taken by a packet is effected by the congestion it experiences on its way.

Thus, 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 [Bar64]. Borodin and Hopcroft, in a landmark paper suggested a greedy hot-potato algorithm for the hypercube [BH85]. 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 [AS91, GG86, GH92, LP80, Max89, Pra86]. Prager [Pra86] showed that the Borodin-Hopcroft algorithm terminates in n steps on the 2^n-nodes hypercube for a special class of permutations. Hajek [Haj91] 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 algorithm by Hajek gives priority to packets that are *closer* to their destinations [Haj91]. It is straightforward to see that using this basic rule, any greedy packet routing algorithm terminates in $O(k \cdot diam)$ where k is the number of packets in the system and $diam$ is the network diameter. This implies, for example, $O(n^3)$ upper bound for such an algorithm on the two-dimensional $n \times n$ mesh, where each node is the origin of a single packet.

The work of Hajek was simplified and generalized in a work by Brassil and Cruz [BC91]. For any regular network with undirected edges (such as the mesh and the hypercube) the algorithm of Brassil and Cruz assumes some pre-specified 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 and $diam$ is as above and P is the length of a walk connecting all destinations. For some worst case situations, this result is optimal up to constant factors. However, the technique that is used has some fundamental flaws. First, all processors must know the full order of the destinations, and should be able to give priority to packets according to this order. This task is far from being efficient when using non trivial orders. Moreover, there is no reason to assume that the set of destinations is known in advance and that a short walk was computed. Thus, in the common case, a simple Hamiltonian tour of the network will be used, hence P may become as large as $k \cdot diam$. For instance, the time to route n packets on the $n \times n$ mesh will be bounded by $O(n^2)$ in the worst case. The second problem is that the method that is used is "inherently sequential": Suppose some important parameter improves, for example, the maximal distance to destination becomes smaller, or the network degree becomes higher. Clearly, this accelerates the routing time. Using the method in [Haj91] and [BC91], however, such information fails to improve the bound.

Some recent results concern structured (non-greedy) hot potato permutation routing: Feige and Raghavan [FR92] presented an algorithm for the 2-dimensional torus, that routes most of the routing problems in no more then $2n + O(\log n)$ steps. Newman and Schuster [NS92] have presented an algorithm that is based on sorting for permutation routing in the 2-dimensional mesh. Their algorithm routes every permutation in $7n + o(n)$ steps. Bar-Noy et al. [BNRST93] present a relatively simple algorithm for the 2-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 have presented a much more complex algorithm (which is not based on sorting) that routes every permutation in the 2-dimensional mesh and torus within $O(n^{1+\varepsilon})$ steps (for every $\varepsilon > 0$). Kaklamanis, Krizanc and Rao [KKR93] have presented an algorithm for permutation routing 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 2-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* (so that each processor is the origin of at most $2d$ packets). 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 [NS89, Haj91]. Hence, simple restrictions must be added to the routing mechanisms in order to ensure termination. Even when imposing weak restrictions on the algorithm, the analysis seems extremely hard. As the algorithm is fully dynamic, there is no a-priori bound on the time a packet may travel before arriving to its target.

We present upper bounds on the running time of a wide class of greedy algorithms. Some of these bounds are tight, 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 d-dimensional 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, there is high chance of finding 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 isopertimetric inequality can be shown to improve rapidly, yielding an improved bound.

The rest of this work is organized as follows: in Section 2 we briefly give some preliminaries. In Sections 3, 4 and 5 we present the potential function analysis: Section 3 gives the general method in terms of an upper bound on the running time of any algorithm for which there exists a potential function which 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 2-dimensional $n \times n$ mesh, for routing problems that contain k packets. In Section 5 and Appendix A we generalize the potential function to apply for the d-dimensional mesh, for $d > 2$. We use it to show 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

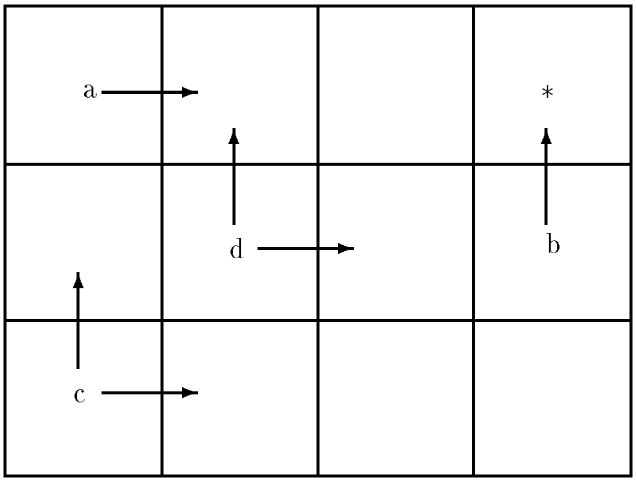

Figure 1: The packets a and b are restricted. The packets c and 1-3d are not.

n^d-nodes mesh. In Section 6 we give some concluding remarks and open problems.

2 Some Preliminaries

In this work, we consider *many-to-many batch packet routing* in synchronous networks. In this problem, there is a set of packets that are originated 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.

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. We consider *greedy* hot-potato routing algorithms. In such algorithms, a packet always advances, unless all its good arcs are used by other packets. We make this formal in the following definition.

Definition 1: A hot-potato routing algorithm is said to be *greedy*, if it satisfies the following condition: Whenever a packet p leaves its current node via a bad arc, all its good arcs must be used by other packets. Moreover, every arc that is good for p must also be good for the packet that uses it (that is - all these "other packets" must actually advance towards their destinations).

An important notion that is used throughout the paper is that of a *restricted packet.*

Definition 2: A packet in the 2-dimensional mesh is said to be *restricted* if it has only one good direction. This means that the packet is either in the row or in the column of its destination node, see Fig. 1.

3 Potential Function Analysis

In this section, we derive a general method of using a potential function analysis. In Sections 4, 5 and A we show some applications of this method.

Let $\mathcal{A}$ be a greedy routing algorithm. Suppose that for every packet p we have a potential function $\phi_p(t)$ such that $0 \le \phi_p(t) \le M$ for every t (where M is some positive constant) and $\phi_p(t) = 0$ only when the packet p has reached its destination by step t. Consider the "global" potential function $\Phi(t)$ that is defined as $\Phi(t) \stackrel{\mathrm{def}}{=} \sum_p \phi_p(t)$. 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.

In order to define the local property we need, let us first define the notion of a node that *looses potential.*

Definition 3: We say that a node S in the mesh *looses potential* in step t if the sum of the potential of the packets that entered S at time t is greater than the sum of the potential of these 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 now be stated as follows:

Property 4: Let S be a node in the d-dimensional mesh that contains ℓ packets in step t.

- If $\ell \le d$ then S looses at least ℓ potential units at step t (that is $\Delta\Phi_S \ge \ell$).
- If $\ell > d$ then S looses at least $2d - \ell$ potential units at step t (that is $\Delta\Phi_S \ge 2d - \ell$).

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.

We show now that Property 4, together with the constraints on ϕ_p, yields an upper bound on the running time of the algorithm $\mathcal{A}$. Later, we show some examples of routing algorithms and potential functions that satisfy this property.

For the sake of analysis, we divide the nodes in the mesh into "good" nodes and "bad" nodes.

Definition 5: 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 number of packets in bad nodes at time t, and $G(t)$ to denote number of packets in good nodes at time t.

We can interpret Property 4 as follows: If S is a good node, it looses at least one potential unit for every packet in it. If S is a bad node, it looses at least one potential unit for every "missing" packet. As an immediate corollary we get

Corollary 3.1: For every step t, $\Phi(t+1) \le \Phi(t) - G(t)$.

In particular, from Corollary 3.1 it follows that the potential function is a monotonic non-increasing function.

The main difficulty we face is that we do not have any a-priory 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 Fig. 2), 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.

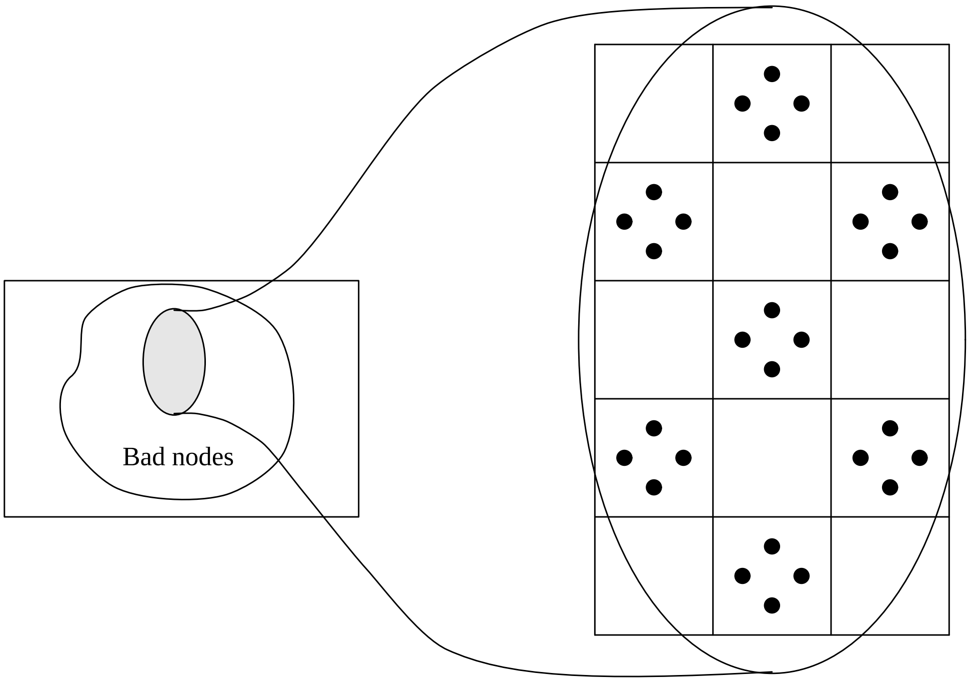

Figure 2: An area of bad nodes in the 2-dimensional mesh

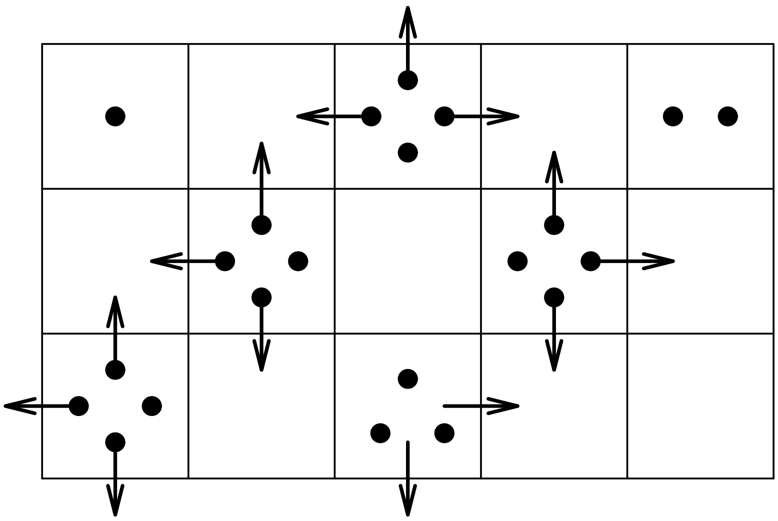

Figure 3: Frontier arcs in the 2-dimensional mesh

Definition 6: We say that an arc e, which goes out of a node S is a *frontier arc* if

- The node S is a bad node.
- Either the node in distance 2 from S in the direction of e is a good node, or S does not have a node in distance 2 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 that is at the edge of the mesh) as a frontier arc(see Fig. 3). We denote the number of frontier arcs at time t by $F(t)$.

Lemma 3.2: For every t holds $\Phi(t+2) \leq \Phi(t) - F(t)$.

Basically, we show that for every frontier arc we can account a loss of at least one potential unit either at step t or at step $t+1$. The proof is omitted in this extended abstract.

Next we show that if there are many bad nodes, then there must also be many frontier arcs.

Lemma 3.3: If there are $B(t)$ packets in bad nodes at the beginning of step t, then the number of frontier arcs in that step is at least $(2d)^{1/d} \cdot B(t)^{(d-1)/d}$. Namely, for every t it holds that

$$F(t) \geq (2d)^{1/d} \cdot B(t)^{(d-1)/d}$$

The proof uses a geometric interpretation of a d-dimensional mesh: Each node is represented by a d-dimensional 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. In the full paper we show that for a given d-dimensional volume V, that is composed of d-dimensional cubes, the minimal possible surface is $2dV^{(d-1)/d}$. The rest of the proof goes by manipulating this bound, and is omitted in this extended abstract.

Lemma 3.4: 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 $(2d)^{\frac{1}{d}}\left(\frac{\Phi(t)}{2M}\right)^{\frac{d-1}{d}}$. Namely, for every t it holds that

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

The proof of this lemma goes by using the convexity of the bounds on the decrease of the potential (Corollary 3.1, Lemma 3.2 and Lemma 3.3), and is omitted in this extended abstract.

Theorem 1: If $\mathcal{A}$ is a routing algorithm and Φ is a potential function which satisfies Property 4, then $\mathcal{A}$ solves every routing problem with k packets in the d-dimensional mesh within at most $2^{(2d-2)/d} \cdot d^{(d-1)/d} \cdot k^{1/d} \cdot M$ steps.

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 3.4, $\Phi(t+2) \leq \Phi(t) - (2d)^{1/d} \cdot \left(\frac{\Phi(t)}{2M}\right)^{(d-1)/d}$

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 rest of the proof involves some infinite sums manipulation, and can be found in the full paper.

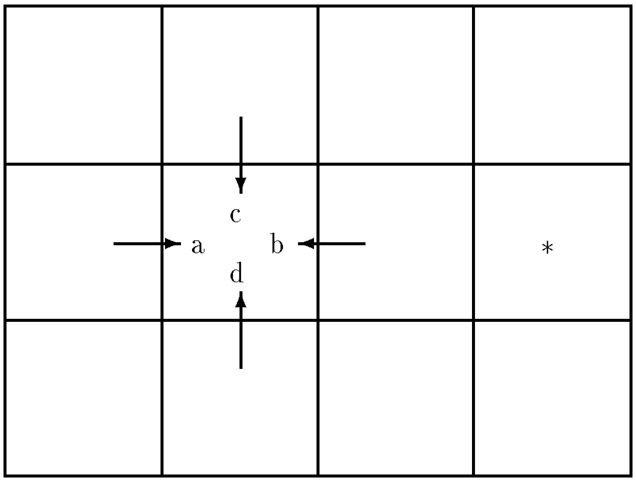

Figure 4: Restricted packets: b, c and d are of type B, and a is of type A

4 Potential Function in the 2-dimensional mesh

In this section, we show a potential function in the 2-dimensional mesh which satisfies Property 4, and use Theorem 1 to obtain an upper bound on the running time of a large class of greedy algorithms. To this end, we need to restrict somewhat the routing algorithm.

4.1 Priority to Restricted Packets

Recall that a packet in the 2-dimensional mesh is *restricted* if it has exactly one good direction. We divide the restricted packets into two types (see Fig. 4):

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 7: We say that a greedy routing algorithm *prefers restricted packets* if a non-restricted packet can not 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 property of algorithms that prefer restricted packets.

Claim 4.1:

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.

The idea in the proof is that a restricted packet that is advancing, keeps moving in the same direction all the time. The proof appears in the full version.

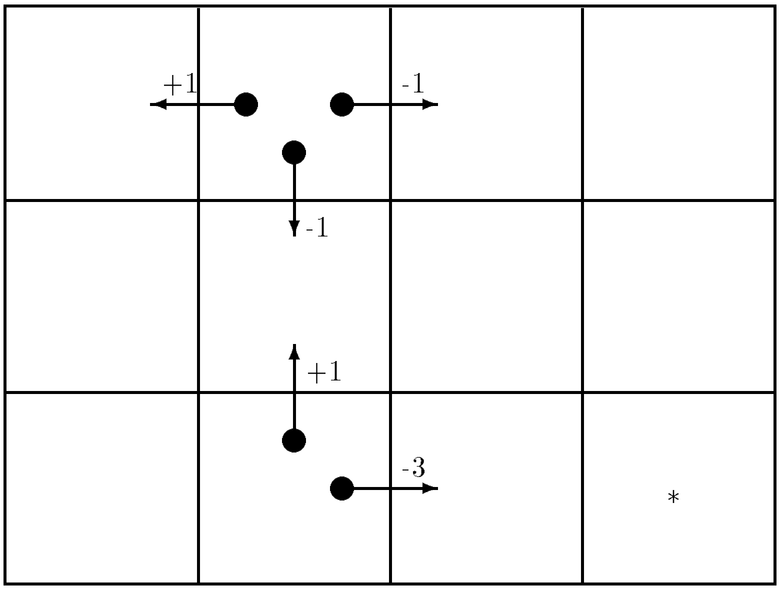

Figure 5: Changes in the potential of packets in one step

4.2 Definition of the Potential Function

We now show that a routing algorithm in the 2-dimensional mesh that 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.

Initially, every packet p has $2n$ additional units of potential. As long as p is not 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 2 units of the potential it has just received from the deflected packet.

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

1. Initially, for every p, $C_p(0) = 2n$.

2. If after step t the 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 the 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 (from Claim 4.1 it follows that 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 Claim 4.1 it follows that in case 3.b, the 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, the packet q had $2n$ units of additional potential (since it was deflected), and the packet p has $C_q(t-1)-2$ units of additional potential. Thus, the sum of additional potential of the 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 $\phi_p(t) \stackrel{\text{def}}{=} dist_p(t) + C_p(t)$. The potential of the mesh at the beginning of step t, is $\Phi(t) \stackrel{\text{def}}{=} \sum_p \phi_p(t)$. We proceed to show that the potential function Φ satisfies Property 4.

Lemma 4.2: 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.

The proof of the lemma appears in the full version. Since the potential function satisfies Property 4, and $0 \leq \phi_p(t) \leq 4n$, we can use Theorem 1 with $d = 2$ and $M = 4n$ to obtain

Theorem 2: 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.

5 Potential Functions in the d-dimensional mesh

In this section we generalize the method of the previous section, to the d-dimensional mesh.

Definition 8: Let p be a packet in the mesh. We say that p is in the *i'th level* if p has exactly i good directions. (In particular, a restricted packet is in level 1).

For every level i, we divide the packets that are in the i'th level into two types.

Type A: Packets that have advanced in level i in the previous step.

Type B: All the other packets in this level. I.e., packets that were either deflected in step $t-1$, or that advanced from level $i+1$ to level i in the previous step.

The generalization of an algorithm that gives priority to restricted packets, is an algorithm that gives priority to packets from lower levels. In order to generalize our analysis to the d-dimensional mesh, we need to add some more constraints on the algorithm.

Definition 9: An algorithm which obeys the following rules is said to *prefer lower levels*:

- A packet in the i'th level has priority over a packet in the j'th level for every $i < j$. This means that when a packet of level i is deflected, all the packets that deflect it are from levels $\leq i$.

- Between packets in the same level, a packet of type A has priority over a packet of type B.

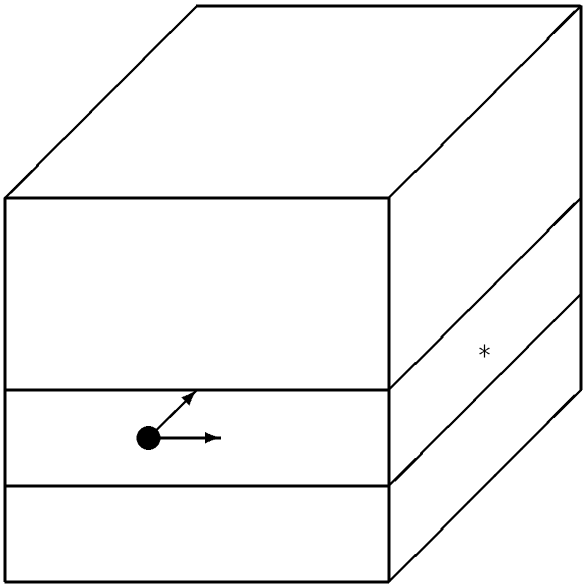

Figure 6: A packet in level 2 in the 3-dimensional mesh

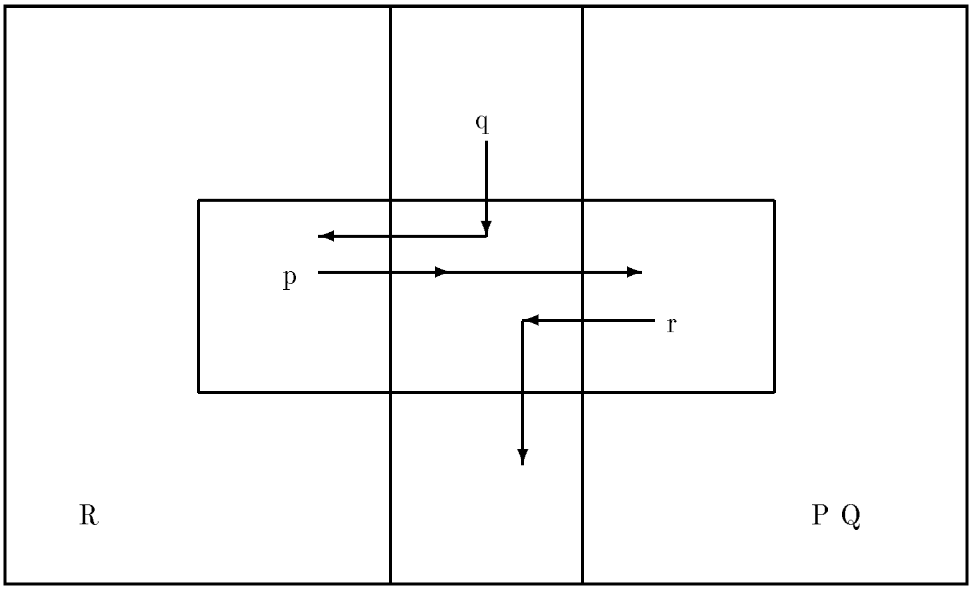

Figure 7: A violation of Rule 4: Only packets p and r advance (Small letters denote packets, big letters denote destinations).

- The algorithm maximizes the number of packets that advance, among the packets of type A in every level. (This rule is intended to prevent pathological events such as the one depicted in Figure 7.)

The reason we need to add these additional constraints is that we need to make sure that a packet from level i will not deflect another advancing packet from the same level that has different good directions.

Claim 5.1: Let S be a node in the d-dimensional mesh, and let i be any level ($1 \leq i \leq d$).

1. If there is a packet in level i that is deflected from S in step t, then there are at least i packets in levels $\leq i$ that advance from S in this step.
2. If there are x type-A packets in level i that are deflected from S in step t, then there are at least x packets in levels $\leq i-1$ that advance from S in this step.

The proof is omitted in this extended abstract.

5.1 Defining the Potential Function

As we did in the 2-dimensional mesh, we give every packet two forms of potential. The first is the distance between the packet and its destination, and in addition, every packet is given $d-1$ "batteries", one for every level (except for level d). These batteries include some additional units of potential.

Initially, the battery of the i'th level contains some amount of potential, denoted by Ψ_i. When a packet advances one step in the i'th level, it drops some other amount of potential (denoted by ψ_i) out of its i'th battery. The amounts Ψ_i and ψ_i will be determined later. When a packet is being deflected in the i'th level, its i'th battery is filled again with Ψ_i units of potential. Also, if a packet is being deflected from the i'th to the $i+1$'th level, then both its i'th and its $i+1$'th batteries are filled with Ψ_i and Ψ_{i+1} units of potential, respectively.
Formally, we denote the contents of the i'th battery of a packet p after step t by $C_i^p(t)$.

1. Initially, for every packet p and every level $1 \leq i \leq d-1$, $C_i^p(0) = \Psi_i$.
2. When the packet p advances in the i'th level then $C_i^p(t) = C_i^p(t-1) - \psi_i$.
3. When the packet p is being deflected in the i'th level then $C_i^p(t) = \Psi_i$.
4. When the packet p is being deflected from the i'th level to the $i+1$'th level then $C_i^p(t) = \Psi_i$ and $C_{i+1}^p(t) = \Psi_{i+1}$.
5. In every other case $C_i^p(t) = C_i^p(t-1)$. I.e., if p was not in the i'th level before step t, and did not enter the i'th level in step t then the contents of its i'th battery does not change.
6. When a packet p reaches its destination then for every level $1 \leq i \leq d-1$, $C_i^p(t) = 0$.

The potential of a packet p at the end of step t is defined to be $\phi_p(t) \stackrel{\rm def}{=} dist_p(t) + \sum_{i=1}^{d-1} C_i^p(t)$. The potential of the mesh at the end of step t is $\Phi(t) \stackrel{\rm def}{=} \sum_p \phi_p(t)$.

The amounts Ψ_i and ψ_i are set such that the following conditions will hold:

1. $\psi_{d-1} \geq 1$.

2. For every i, $\Psi_i \geq i(n-1) \cdot \psi_i$.

3. for every $i \leq d-2$, $\psi_i \geq \frac{2d-i}{i} \cdot (\Psi_{i+1} + \Psi_{i+2} + 1)$.

The reason for the last condition is that i packets that are advancing in the i'th level may have to "pay" for $2d-i$ other packets that are deflected from level $i+1$. From Claim 5.1 it follows that a packet in level i needs never "pay" for other packets in levels $\leq i$. The amounts Ψ_i and ψ_i we use are

$$\Psi_i = \frac{(2d-i)!}{d!} n^{d-i}, \quad \psi_i = \frac{\Psi_i}{i(n-1)}$$

It can be verified that these amounts indeed satisfy the above conditions.

5.2 Properties of the potential function

From the definition above it follows that the potential of a packet always satisfy the following conditions:

- If the packet p is currently in level $\ell > i$ then $C_i^p(t) = \Psi_i$.
- If the packet p was deflected in the i'th level in the last step then $C_i^p(t) = \Psi_i$.
- For every $1 \leq i \leq d-1$, $C_i^p(t) \geq 0$. This is because when a packet begin to advance in the i'th level, it has $C_i^p = \Psi_i$. Then, it can advance at most $i(n-1)$ in this level before it goes down to a lower level. At every step, the potential in the i'th battery is reduced by $\psi_i \leq \frac{\Psi_i}{i(n-1)}$. Thus C_i^p is always non-negative.
- For every packet p it always holds that

$$\phi_p(t) = disp_p(t) + \sum_{i=1}^{d-1} C_i^p(t) \leq d(n-1) + \sum_{i=1}^{d-1} \frac{(2d-i)!}{d!} n^{d-i} \quad \leq \quad 2\frac{(2d-1)!}{d!} \cdot n^{d-1}$$

We proceed to show that for an algorithm that gives priority to lower levels, the potential function above satisfies Property 4.

Lemma 5.2: Let $\mathcal{A}$ be a routing algorithm that gives priority to lower levels, and let S be a node in the mesh that contains ℓ packets at step t.

- If $\ell \leq d$ then S looses at least ℓ potential units at step t.
- If $\ell > d$ then S looses at least $2d - \ell$ potential units at step t.

We omit the proof in this extended abstract.

We have proved that for an algorithm that gives priority to lower levels, the potential function above satisfies Property 4. Applying Theorem 1 with $M = 2\frac{(2d-1)!}{d!} n^{d-1}$ we obtain

Theorem 3: For every hot potato greedy algorithm, $\mathcal{A}$, that gives priority to lower levels, and every routing problem with k packets in the d-dimensional mesh, the number of steps required by $\mathcal{A}$ to solve the routing problem is at most

$$2^{3-2/d} \cdot \frac{(2d-1)!}{d^{1/d}(d-1)!} \cdot k^{1/d} \cdot n^{d-1}$$

6 Conclusions and Open Problems

In this work we have shown upper bounds on the worst-case running times of greedy hot-potato routing algorithms in d-dimensional meshes. For routing requests in which the number of packets is of the same order as the number of processors (i.e., $k = \Theta(n^d)$) we obtain a bound of $O(exp(d) \cdot n^d)$. For a constant d, and for the worst case in which the number of packets, k, is n^d and all packets are destined to the same processor which is of degree d, our bound matches the $\Omega(k/d)$ lower bound up to a constant factor.

To obtain the above bounds we have developed a general method for the use of potential functions. Two important ingredients in this method are as follows.

- The specific routing algorithm and the corresponding potential function. These are to be plugged in the general result (see Theorem 1).
- The isoperimetric inequality to be plugged in Lemma 3.3, and for which the worst case is evaluated in the full paper.

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).

Perhaps even simpler is the improvement in the bound when a better isoperimetric inequality is found. For example, it is straightforward to show a much better bound than the one which we use as a worst case, 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. This is the topic of current research (Very recently, a greedy algorithm was shown to route every permutation with maximal distance 3 in at most 7 steps on the 2-dimensional mesh [BS93]. A folklore algorithm achieves $O(n^2)$ routing time on the 2^n hypercube for the same problem [Bor].).

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 a non-trivial upper bound 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 non-greedy routing there exists a well developed theory of such algorithms, where most lower bounds are matched up to small constant factors, see [NS92]. In the greedy domain, however,although simulations predict superb performance, there are no better upper bounds than those presented in [BC91] and in this work. As already mentioned, the method in [BC91] seems inadequate in this case. In fact, the bound given in [BC91] *improves* when the number of destination *decreases*. In our further research we focus on this problem using the tools that were developed in this work.

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.

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 *29th annual 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.

[BHS93] A. Ben-Dor, S. Halevi, and A. Schuster. On greedy hot-potatoe routing. Technical Report PCL Report #9204, CS dept. Technion, January 1993.

[BNRST93] A. Bar-Noy, P. Raghavan, B. Schieber, and H. Tamaki. Fast deflection routing for packets and worms. In *Proc. 12th ACM symp. on Principles Of Distributed Computing*, 1993.

[Bor] A. Borodin. Private communication.

[BS93] I. Ben-Aroya and A. Schuster. A greedy algorithm for short-distance permutation routing. Technical Report PCL Report #9320, CS dept. Technion, November 1993.

[CGFS86] F.R.K. Chung, R.L. Graham, P. Frankl, and J.B. Shearer. Some intersection theorems for ordered sets and graphs. *J. of Combinatorial Theory, Series A*, 43:23–37, 1986.

[FR92] U. Feige and P. Raghavan. Exact analysis of hot-potato routing. In *proc. IEEE symp. on foundations of computer Science*, 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.

[Hil85] W.D. Hillis. *The Connection Machine*. MIT press, 1985.

[KKR93] C. Kaklamanis, D. Krizanc, and Satish Rao. Hot-potato routing on processor arrays. In *Symp. of Parallel Algorithms and Architectures*, 1993.

[LP80] D.H. Lawrie and D.A. Padua. Analysis of message switching with shuffle-exchanges in multi-processors. *Proc. Workshop on Interconnection Networks for Parallel and distributed computing*, pages 116–123, 1980.

[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 PCL Report #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 Annual ACM Symp. on Parallel Algorithms and Architectures*, San Diego, June 1992. 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.

A Improving the bound

The bound that we have shown in Section 5 is asymptotically optimal for constant d, and for $k = O(n^d)$. However, the factor of d in it is super exponential. We now show how to improve this factor to an exponential factor. In the analysis above, there are two things that contribute to the size of the factor of d:

1. Since i packets in the i'th level can deflect in the worst case $2d - i$ other packets in the $i + 1$'th level, we got the requirement $i\psi_i \geq (2d - i)\Psi_{i+1}$.

2. Since a packet can travel at most $i(n-1)$ steps in the i'th level, we got the requirement $\Psi_i \geq i(n-1)\psi_i$.

As this analysis is local (that is, every node is considered separately) and since the situation in which i packets from level i deflect $(2d-i)$ packets in level $i+1$ can actually occur, it seems that the first factor can not be eliminated without a drastic change in the analysis. The second factor, however, can be decreased if we ensure that a packet will always travel in the same direction until it goes down from its current level (or until it is deflected). As a packet can travel at most $n-1$ steps in a certain direction until it goes down to a lower level, we can replace the second requirement by the weaker requirement, $\Psi_i \geq (n-1)\psi_i$. This decreases the bound by a factor of $d!$, and we obtain a bound with a factor of d that is about $\frac{(2d)!}{(d!)^2} \approx 2^{2d}$.

Definition 10: We say that a packet p *turns* in step t, if the direction in which it is sent in step t is not the same as the direction in which it was sent in step $t-1$.

A.1 Inertial Algorithm

Definition 11: (Inertial Algorithm) We say that a hot-potato routing algorithm is *inertial* if packets in lower levels have priority over packets in higher levels (like in the first condition in Definition 9). In addition, for every packet p that advanced in the previous step, and is now in some node S, one of the following holds:

- The direction in which p advanced in the previous step, is no longer a good direction for p. This means that p went down one level in the previous step.
- p is advancing from S in the same direction it entered S in the previous step.
- There is another packet that is advancing from S in the same direction in which p entered S in the previous step. Moreover, that packet is in lower level than p.

Notice that in the third case, both p and the other packet are turning in the current step.

A simple algorithm that satisfies the conditions of Definition 11 is the following: Each processor performs the following procedure in every step

1. For every level, from d down to 1, do
   (a) For every packet p in the current level – if the direction in which it has advanced in the previous level is still free, then send it in this direction.
   (b) For every packet p in the current level that were not sent in the previous stage – if there are still free arcs that are good for it then send it on one of these arcs.
2. Send every packet that was not yet sent in some arbitrary free direction.

From the definition of inertial algorithm, it follows that there is a $1-1$ correspondence between packets in level i that advance in step $t-1$ and make a turn in step t, and packets in levels $\leq i-1$ that advance and make a turn in step t.

A.2 The Potential Function

As we did before, we give every packet, p, two types of potential. The first is the distance between p and its destination, denoted by $dist_p$, and in addition, every packet is given $d-1$ "batteries" which include additional units of potential. Initially, the i'th battery contains Ψ_i units of potential.

As long as a packet advances in the i'th level without turning, it drops ψ_i units of potential out of its i'th battery. When a packet in the i'th level is turning, its i'th battery is filled again with Ψ_i units of potential, *even if it is not deflected.* If it is advancing, then ψ_i units of this potential are being dropped immediately, together with additional $\Psi_{i+1} + \Psi_{i+2}$ units (this additional amount is needed for the case where it is deflecting another packet from level $i+1$ to level $i+2$). As before, if a packet is being deflected from the i'th to the $i+1$'th level, then both its i'th and its $i+1$'th batteries are filled with Ψ_i and Ψ_{i+1} units of potential, respectively.

Formally, we denote the contents of the i'th battery of a packet p after step t by $C_i^p(t)$.

1. Initially, for every packet p and every level $1 \le i \le d-1$, $C_i^p(0) = \Psi_i$.
2. When a packet p is advancing without turning in the i'th level, $C_i^p(t) = C_i^p(t-1) - \psi_i$.
3. When a packet p is advancing in the i'th level, and is making a turn, $C_i^p(t) = \Psi_i - \psi_i - \Psi_{i+1} - \Psi_{i+2}$.
4. When a packet p is being deflected in the i'th level then $C_i^p(t) = \Psi_i$.
5. When the packet p is being deflected from the i'th level to the $i+1$'th level then $C_i^p(t) = \Psi_i$ and $C_{i+1}^p(t) = \Psi_{i+1}$.
6. In any other case, $C_i^p(t) = C_i^p(t-1)$.
7. When a packet p reaches its destination, then for every level $1 \le i \le d-1$, $C_i^p(t) = 0$.

The potential of a packet p after step t, is defined $\phi_p(t) \stackrel{\text{def}}{=} dist_p(t) + \sum_{i=1}^{d-1} C_i^p(t)$. The potential of the mesh after step t is $\Phi(t) \stackrel{\text{def}}{=} \sum_p \phi_p(t)$.

The improvement upon the previous bound is because here we can choose smaller amounts for Ψ_i and ψ_i. These amounts are set such that the following conditions will hold:

1. $\psi_{d-1} \ge 1$
2. For every i, $\Psi_i \ge (n-1)\psi_i + \Psi_{i+1} + \Psi_{i+2}$.
3. for every $i \le d-2$, $\psi_i \ge \dfrac{2d-i}{i} \cdot (\Psi_{i+1} + 1)$.

We use the potential function

$$\Psi_i = \frac{(2d-i)!(i-1)!}{(d+1)!(d-2)!} n^{d-i} - 1, \quad \psi_i = \frac{\Psi_i}{n-1}$$

It can be verified that these functions indeed satisfy the above conditions (the third condition is valid only when $n \ge \frac{1}{2}(d-1)$).

Notice that a packet, p, can advance at most $n-1$ steps in level i without turning, before it goes down to level $i-1$. In the first step it drops $\psi_i + \Psi_{i+1} + \Psi_{i+2}$ potential, and in each of the the other steps it drops ψ_i potential. Before p started to advance in that direction, the potential in its i'th battery was $\Psi_i \geq (n-1)\psi_i + \Psi_{i+1} + \Psi_{i+2}$. Thus, the amount of potential in the i'th battery is always non-negative. Therefore, for every packet it holds that $\phi_p(t) \geq 0$. Moreover, it also holds that

$$\phi_p(t) = dist_p(t) + \sum_{i=1}^{d-1} C_p^i(t) \leq d(n-1) + \sum_{i=1}^{d-1} \frac{(2d-i)!(i-1)!}{(d+1)!(d-2)!} n^{d-i} \leq 2\binom{2d-1}{d+1} n^{d-1}$$

A.3 Properties of the potential function

We show now that for an inertial algorithm, the potential function above satisfies Property 4. First we notice that for a packet p in level i, there are four possible cases:

1. p is a type-B packet, and is not deflected in this step. That is, p was either deflected in the previous step, or went down from level $i+1$ in the previous step, and is advancing in this step. In this case, there is no need to fill any of p's batteries.
2. p is a type-B packet, and is deflected in this step. In this case, we may need to fill its $i+1$'th battery (if it is deflected to level $i+1$). Notice that there is no need to fill its i'th battery, since it is already full.
3. p is a type-A packet, and is not turning in this step. This means that p advanced in level i in the previous step, and is advancing in the same direction in this step. In this case, there is no need to fill any of p's batteries.
4. p is a type-A packet, and is turning in this step. In this case, we need to fill p's i'th battery, and perhaps, its $i+1$'th battery too (if it is deflected to the $i+1$'th level).

Lemma A.1: Let $\mathcal{A}$ be an inertial routing algorithm, and let S be a node in the mesh that contains ℓ packets at step t.

- If $\ell \leq d$ then S looses at least ℓ potential units in step t.
- If $\ell > d$ then S looses at least $2d - \ell$ potential units in step t.

Proof omitted.

We proved that for an inertial algorithm, the potential function above satisfies Property 4. Applying Theorem 1 to this potential function with $M = 2\binom{2d-1}{d+1}n^{d-1} < 2^{2d}n^{d-1}$ we obtain

Theorem 4: Let $\mathcal{A}$ be an inertial algorithm. Then for every routing problem with k packets in the d-dimensional mesh, the number of steps required by $\mathcal{A}$ to solve this routing problem is at most

$$4^{d+1-1/d} \cdot d^{1-1/d} \cdot k^{1/d} \cdot n^{d-1}$$