Optimal Oblivious Path Selection on the Mesh

Costas Busch, *Member, IEEE*, Malik Magdon-Ismail, *Member, IEEE*, Jing Xi

***Abstract*— In the oblivious path selection problem, each packet in the network independently chooses a path, which is an important property if the routing algorithm is to be independent of the traffic distribution. The quality of the paths is determined by the congestion C, the maximum number of paths crossing an edge, and the dilation D, the maximum path length. So far, the oblivious algorithms studied in the literature have focused on minimizing the congestion while ignoring the dilation.**

An open problem is to give algorithms for networks in which C and D can be controlled simultaneously. Here, we solve this problem for the d-dimensional mesh. We present an oblivious algorithm for which C and D are both within $O(d^2)$ of optimal. The algorithm uses randomization, and we show that the number of random bits required per packet is within $O(d)$ of the minimum number of random bits required by any algorithm that obtains the same congestion. For fixed d, our algorithm is asymptotically optimal.

***Index Terms*— Oblivious routing, path stretch, network congestion, mesh topology, algorithm randomization.**

I. Introduction

Given a set of packet transfer requests in a communication network, a *routing* algorithm must select the paths that will be traversed in order to fulfill all the requests. A routing algorithm is *oblivious* if every path that is selected for each request is chosen independently of every other path. Oblivious algorithms are by their nature distributed and capable of solving online routing problems, where packets continuously arrive in the network. Hence, oblivious routing (path selection) is preferred to non-oblivious routing, since one does not need to make assumptions regarding the nature of the traffic.

A preliminary version of this paper appears in the 19th International Parallel and Distributed Processing Symposium (IPDPS) [7].

Costas Busch is in the Computer Science Department, Louisiana State University, Baton Rouge, LA 70803, USA, busch@csc.lsu.edu.

Malik Magdon-Ismail is in the Computer Science Department, Rensselaer Polytechnic Institute, Troy, NY 12180 USA, magdon@cs.rpi.edu.

Jing Xi is in the Computer Science Department, Rensselaer Polytechnic Institute, Troy, NY 12180 USA, xij2@cs.rpi.edu.

We study oblivious algorithms in the context of a synchronous routing model in which at most one packet traverses any edge during a time step. Initially, at time zero, a set of packets must simultaneously select their paths. The quality of the paths selected can be measured by two parameters: the *(edge) congestion* C, the maximum number of paths that use any edge in the network, and the *dilation* D, the maximum length of any path. A trivial lower bound for the total time to transfer all the packets along the selected paths is $\Omega(C+D)$, hence $C+D$ is a natural metric by which to measure the quality of the paths outputed by a routing algorithm.

The bandwidth of the network is a major bottleneck, hence optimizing the load distribution, that is, minimizing the congestion, has been the focus of existing research. In particular, Maggs et al. [13] gave the first oblivious path selection algorithm with optimal congestion on the mesh. Since then, generalizations have been given to oblivious path selection algorithms for arbitrary networks that achieve near optimal congestion [3], [4], [9], [16]. However, these algorithms do not control the dilation. For example, a packet path may be *stretched* by an arbitrary amount – a packet that has destination at a neighboring node may traverse the entire network before reaching its destination.

An interesting open problem is to obtain small path stretch in addition to low congestion. Here, we study this problem for the d-dimensional mesh network, for which we show that it is indeed possible to obtain near optimal congestion while maintaining small stretch. For general networks this is not possible, as is explained in Section VI.

A. Optimal Oblivious Routing

Given a routing problem (collection of sources and destinations), we define C^* to be the optimal congestion attainable by any routing algorithm (oblivious or not). We define C^*_{obl} to be the optimal congestion attainable if the routing algorithm is restricted to being oblivious. For the d-dimensional mesh with n nodes, Maggs et al. [13] give the lower bound $C^*_{obl} = \Omega(\frac{C^*}{d} \log n)$ in the worst case. We will compare the congestion of our algorithm with the lower bound on C^*_{obl}. The stretch of a path from a source

to a destination is the ratio of the path length to the shortest path length. Clearly, the smallest stretch factor is 1, since a packet can follow a shortest path; however, the choice of shortest paths may increase the congestion.

B. Contribution

We show the surprising result that it is possible to obtain small stretch for any routing instance, using an oblivious algorithm, and without sacrificing on the congestion. Specifically, we give an oblivious routing algorithm for the d-dimensional mesh with n nodes, that achieves congestion $O(dC^* \log n)$, and stretch $O(d^2)$. Considering the class of oblivious algorithms, our algorithm is within $O(d^2)$ of optimal for both congestion and dilation, which means that the routing time bound $(C+D)$ is within $O(d^2)$ of optimal[1]. For fixed d, our algorithm is optimal to within constant factors.

Our algorithm is based upon a hierarchical decomposition of the mesh. Starting at its source node, a packet constructs its path by randomly selecting intermediate points in submeshes of increasing size until the current submesh contains the destination node. Then random intermediate points are selected in submeshes of decreasing size until the destination node is reached. The key new idea that we introduce is the notion of "bridge" submeshes that make it possible to move from a source to a destination more quickly, without increasing the congestion. These bridge submeshes are instrumental in controlling the stretch, while maintaining low congestion.

Our algorithm uses randomization, and we show that randomization is essential for obtaining low congestion. In a deterministic algorithm, a packet path is fixed, given its source and destination. It can be shown that for any deterministic algorithm, there exists a routing problem for which the deterministic algorithm obtains suboptimal congestion [11]. The only alternative is to use randomization: a packet selects a path probabilistically from a choice of κ alternatives, where κ may depend on the source and destination. Such an algorithm requires at least $\log \kappa$ random bits per packet. We give a lower bound on the number of bits required by any algorithm which attains at most the congestion of our algorithm for any routing problem. Specifically, for every distance ℓ, $\Omega((1-\frac{1}{d}) \log \frac{\ell}{d})$ bits per packet are required for some source destination pair a distance ℓ apart. Since our algorithm attains near-optimal congestion for any routing problem, this same lower bound on the number of bits per packet holds for any oblivious algorithm which guarantees near-optimal congestion. The number of random bits required by our algorithm is within $O(d)$ of this lower bound, and hence our algorithm is near-optimal with respect to the number of bits that are used per packet.

[1] $C + D$ is a well known lower bound on the routing time and there exist scheduling algorithms with performance close to this lower bound, with routing time $O((C + D) \cdot 2^{O(\log^*(C+D))})$, [12]

C. Related Work

Most related to our work is the original paper by Maggs et al. [13] where they present an oblivious algorithm with congestion $O(dC^* \log n)$. However, the stretch factor in that algorithm is unbounded. To control stretch, we generalize the hierarchical decomposition for the mesh denoted by an *access tree* [13] to a more general *access graph*. In the access tree, only one path exists between a particular source and destination, however, our decomposition offers several paths, in particular much shorter paths. The original work was most concerned with data management, and a single access tree is essential in this setting, however, the use of a single access tree gives unbounded stretch.

We build on this work by Our algorithm uses randomized dimension by dimension routing, which alone can improve the result in [13] by a factor of d to $O(C^* \log n)$. The reason for this is discussed after the proof of Lemma 4.9. This improvement has also been obtained in [17, p.46-48] using alternate means. Our access graph strategy becomes critical for controlling the stretch. Following the work in [13], there have been extensions to general networks, [3], [4], [9], [16], where progressively better oblivious algorithms with near optimal congestion are given. Once again, the stretch is unbounded. For general networks, the stretch and congestion cannot be controlled independently, as is explained in Section VI with a counter-example (see also [17, p. 59] for a very similar example). Oblivious congestion minimizing algorithms which control the stretch have been considered by Scheideler [19] for the 2 dimensional mesh, and by Busch et al. [6] for networks which have good embeddings. Scheideler uses a different approach to the 2 dimensional mesh by routing within a square containing the source and destination and building an access tree specific to this square. Busch et al. [6] give a single intermediate point algorithm for networks which have good embeddings in 2 dimensions. Here we consider the d-dimensional mesh and also give lower bounds on the number of random

bits required by any oblivious algorithms which minimizes congestion.

Non-oblivious approaches to optimizing $C + D$ have received considerable attention, and near optimal algorithms are discussed in [1], [2], [18], [20]. As already mentioned, such offline algorithms require knowledge of the traffic distribution *a priori* and generally do not scale well with the number of packets. We show that for the mesh, distributed and oblivious algorithms are within a logarithmic factor from the optimal offline performance, hence there is no significant benefit from using the offline algorithm. Tradeoffs between stretch and congestion have been studied in wireless networks [8], [14], where they show that there exists a tradeoff between the congestion and the stretch. For the d-dimensional mesh, however, we show the surprising result that it is possible to simultaneously obtain constant stretch and optimal congestion.

Lower bounds on the competitive ratio of oblivious routing has been studied for various types of networks. Maggs et al. [13] give the $\Omega(\frac{C^*}{d} \log n)$ lower bound on the competitive ratio of an oblivious algorithm on the mesh. Valiant and Brebner [21] perform a worst case theoretical analysis on oblivious routing on specific network topologies such as the hypercube. Borodin and Hopcroft [5] and Kaklamanis et al. [10] showed that deterministic oblivious routing algorithms can not approximate the minimal load on most non-trivial networks, which justifies the necessity for randomization. Here, we give a lower bound on the amount of required randomization.

D. Paper Outline

We begin with some preliminary definitions in Section II. In Section III, we continue with our mesh decomposition algorithm in 2-dimensions, which we generalize to d-dimensions in Section IV. We discuss randomization requirements in Section V, and we conclude in Section VI.

II. Preliminaries

The d-dimensional mesh M is a d-dimensional grid of nodes with side length m_i in dimension i. There is a link connecting a node with each of its $2d$ neighbors (except for the nodes at the boundaries of the mesh). We denote by n the size of M, $n = \mathsf{size}(M) = \prod_{i=1}^{d} m_i$, and by $|E|$ the number of edges in the network. Each node has a coordinate. For example, in the 2 dimensional mesh, the top-left node has coordinate $(0, 0)$. We refer to specific submeshes by giving its end points in every dimension, for example, $[0, 3][2, 5]$ refers to a 4×4 submesh, with the x coordinate ranging from 0 to 3 and the y coordinate from 2 to 5.

The input for the path selection problem is a set of N sources and destinations (i.e. packets), $\Pi = \{s_i, t_i\}_{i=1}^{N}$ and the mesh M. The output is a set of paths, $P = \{p_i\}$, where each path $p_i \in P$ is from node s_i to node t_i. The length of path p, denoted $|p|$, is the number of edges it uses. We denote the length of the shortest path from s to t by $\mathsf{dist}(s, t)$. We will denote by D^* the maximum shortest distance, $\max_i \mathsf{dist}(s_i, t_i)$. The *stretch* of a path p_i, denoted $\mathsf{stretch}(p_i)$, is the ratio of the path length to the shortest path length between its source and destination, $\mathsf{stretch}(p_i) = |p_i|/\mathsf{dist}(s_i, t_i)$. The *stretch factor* for the collection of paths P, denoted $\mathsf{stretch}(P)$, is the maximum stretch of any path in P, $\mathsf{stretch}(P) = \max_i \mathsf{stretch}(p_i)$.

For a submesh $M' \subseteq M$, let $\mathsf{out}(M')$ denote the number of edges at the boundary of M', which connect nodes in M' with nodes outside M'. For any routing problem Π, we define the *boundary congestion* as follows. Consider some submesh of the network M'. Let Π' denote the packets (pairs of sources and destinations) in Π which have either their source or destination in M', but not both. All the packets in Π' will cross the boundary of M'. The paths of these packets will cause congestion at least $|\Pi'|/\mathsf{out}(M')$. We define the boundary congestion of M' to be $B(M', \Pi) = |\Pi'|/\mathsf{out}(M')$. For the routing problem Π, the boundary congestion B is the maximum boundary congestion over all its submeshes, i.e. $B = \max_{M' \subseteq M} B(M', \Pi)$. Clearly, $C^* \geq B$.

III. The 2-Dimensional Mesh

Here we show how to select the paths in a 2-dimensional mesh with equal side lengths $m = 2^k$, $k \geq 0$. We consider this case here for expository ease, however the result generalizes to the case of unequal side lengths which are not necessarily powers of 2. We use the 2-dimensional case to illustrate the main ideas, before generalizing to the d-dimensional case in the next section. The path selection algorithm relies on a decomposition of the mesh to submeshes, and then constructing an access graph, as we describe next.

A. Decomposition to Submeshes

We decompose the mesh M into two types of submeshes, type-1 and type-2, as follows.

*a) Type-*1 *Submeshes.:* We define the type-1 submeshes recursively. There are $k+1$ levels of type-1 submeshes, $\ell = 0, \ldots, k$. The mesh M itself is the only level 0 submesh. Every submesh at level ℓ can be partitioned into 4 submeshes by dividing each side by 2. Each resulting submesh is a type-1 submesh at level $\ell+1$. This construction is illustrated in Figure 1. In general, at level ℓ there are $2^{2\ell}$ submeshes each with side $m_\ell = 2^{k-\ell}$. Note that the level k submeshes are the individual nodes of the mesh.

*b) Type-*2 *Submeshes.:* There are $k-1$ levels of type-2 submeshes, $\ell = 1, \ldots, k-1$. The type-2 submeshes at level ℓ are obtained by first extending the grid of type-1 meshes by adding one layer of type-1 meshes along every dimension. The resulting grid is then translated by the vector $-(m_\ell/2, m_\ell/2)$. In this enlarged and translated grid, some of the resulting translated submeshes are entirely within M. These are the *internal* type-2 submeshes. For the remaining $external$ type-2 submeshes, we keep only their intersection with M, except that we discard all the "corner" submeshes, because they will be included in the type-1 submeshes at the next level. Notice that all the type-2 submeshes have at least 1 side of length m_ℓ nodes. Figure 1 illustrates the construction.

A submesh of M is *regular* if it is either type-1 or type-2. Unless otherwise stated, a submesh will always refer to regular submeshes. The following lemma follows from the construction of the regular submeshes.

Lemma 3.1: The mesh decomposition satisfies the following properties.

(1) The type-1 submeshes at a given level are disjoint, as are the type-2 submeshes.
(2) Every regular submesh at level ℓ can be partitioned into type-1 submeshes at level $\ell+1$.
(3) Every regular submesh at level $\ell+1$ is completely contained in a submesh at level ℓ of either type-1 or type-2, or both.

B. Access Graph

The access graph $G(M)$, for the mesh M, is a leveled graph with $k+1$ levels of nodes, $\ell = 0, \ldots, k$. The nodes in the access graph correspond to the distinct regular submeshes. Specifically, every level-ℓ submesh (type-1 or type-2) corresponds to a level ℓ node in $G(M)$. Edges exist only between adjacent levels of the graph. Let u_ℓ, $u_{\ell+1}$ be level ℓ and $\ell+1$ nodes of $G(M)$ respectively. The edge $(u_\ell, u_{\ell+1})$ exists if the regular submesh corresponding to u_ℓ completely contains the regular submesh corresponding to $u_{\ell+1}$. We borrow some terminology from trees. We say that u_ℓ is a *parent* of $u_{\ell+1}$ in $G(M)$; the parent relationship is illustrated in Figure 1, for the corresponding submeshes. Note that the access graph is not necessarily a tree, since a node can have two parents (a consequence of Lemma 3.1, part (3)). The depth of a node is the same as its level ℓ, and its height is $k-\ell$. Nodes at height 0 have no children, and are leaves. The leaves in $G(M)$ correspond to single nodes in the mesh. There is a unique root at level 0, which corresponds to the whole mesh M.

Let $p = (u_1, u_2, \ldots, u_k)$ be a path in $G(M)$. We say that p is *monotonic* if every node is of increasing level (i.e., the level of u_i is higher than the level of u_{i+1}), and the respective submeshes of nodes $u_2, \ldots, u_k$ are all of type-1. If p is monotonic, then we say that u_1 is *ancestor* of u_k. We will use a function g to map nodes in the access graph to submeshes. Let u be a node in the access graph with corresponding submesh M'. We define the function g so that $g(u) = M'$. Denote by g^{-1} the inverse of function g, that is, $g^{-1}(M') = u$. Using induction on the height of $G(M)$, and part (2) of Lemma 3.1, we obtain the following lemma:

Lemma 3.2: Let v be any node (1×1 submesh) of a regular submesh $M' \subseteq M$, then $g^{-1}(M')$ is an ancestor of $g^{-1}(v)$.

Let u and v be two leaves of $G(M)$, and let A be their (not necessarily unique) deepest common ancestor; note that A exists and in the worst case is $g^{-1}(M)$ (a consequence of Lemma 3.2). Let $p = (u, \ldots, A, \ldots, v)$, be the concatenation of two monotonic paths, one from A to u and the other from A to v. We will refer to p as the *bitonic* path between u and v. Submesh $g(A)$ may be type-1 or type-2, all the other submeshes in p are of type-1. We will refer to $g(A)$ as a "bridge" submesh, since it provides the connecting point between two monotonic paths. Note that type-2 submeshes can be used as bridges between type-1 submeshes, when constructing bitonic paths between leaves. Further, only one type-2 submesh is ever needed in a bitonic path. These access graph paths will be used by the path selection algorithm. Suppose that $height(A) = h_A$. The length of a bitonic path from u to v is $2h_A$. We now show that h_A cannot be too large. This will be important in proving that the path selection algorithm gives constant stretch.

Lemma 3.3: The deepest common ancestor of two leaves u and v has height at most $\lceil \log \mathsf{dist}(g(u), g(v)) \rceil + 2$.

Proof: Let $s, t \in M$ such that $s = g(u)$ and $t = g(v)$.

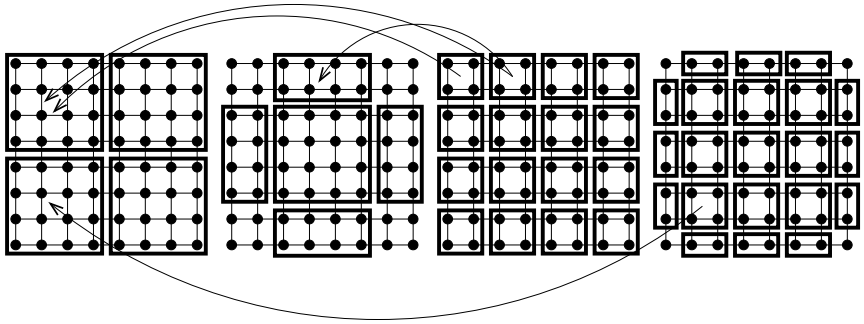

Fig. 1. Mesh decomposition for the $2^3 \times 2^3$ mesh. Arrows indicate the parents of a submesh.

We show that there is a common ancestor with height at most $\lceil \log \mathsf{dist}(s,t) \rceil + 2$.

Assume, first, that instead of a mesh, the network is a torus (the same result holds for the mesh, with a minor technical detail in the proof due to edge effects, which we will discuss later). In this case, all the type-2 meshes are of the same size. We obtain the regular submeshes in the original mesh after truncation of the submeshes at the borders of the torus. Note that all distances, however, are measured on the mesh.

Let $\mu = 2^{\lceil \log \mathsf{dist}(s,t) \rceil} \geq \mathsf{dist}(s,t)$. If $4\mu \geq 2^k$, then the root, $g^{-1}(M)$, is a common ancestor with height at most $\lceil \log \mathsf{dist}(s,t) \rceil + 2$, so assume that $4\mu < 2^k$. Node s is contained in some type-1 submesh of side length 4μ. Without loss of generality (since we are on a torus), assume that this submesh is $M_1 = [0, 4\mu-1]^2$. If M_1 also contains t, then we are done, since by Lemma 3.2, $g^{-1}(M_1)$ is a common ancestor at height $\lceil \log \mathsf{dist}(s,t) \rceil + 2$. So suppose that t is contained in some other (adjacent) type-1 submesh M_2. There are two possibilities for M_2.

(i) M_1 and M_2 are diagonaly adjacent, so without loss of generality, let $M_2 = [4\mu, 8\mu-1][4\mu, 8\mu-1]$. Since $\mathsf{dist}(s,t) \leq \mu$, $s \in [3\mu, 4\mu-1]^2$ and $t \in [4\mu, 5\mu-1]^2$, and so the type-2 submesh $[2\mu, 6\mu-1]^2$ contains both s and t.

(ii) M_2 is laterally adjacent to M_1, so, without loss of generality, let M_2 be to the right of M_1, i.e. $M_2 = [0, 4\mu-1][4\mu, 8\mu-1]$. In this case, s must be in the right half of M_1 and t in the left half of M_2, i.e. $s \in [0, 4\mu-1][3\mu, 4\mu-1]$ and $t \in [0, 4\mu-1][4\mu, 5\mu-1]$. There are four cases:

(a) $s \in [0, 2\mu-1][3\mu, 4\mu-1]$ and $t \in [0, 2\mu-1][4\mu, 5\mu-1]$, in which case the type-2 submesh $[-2\mu, 2\mu-1][2\mu, 6\mu-1]$ contains s, t;

(b) $s \in [2\mu, 4\mu-1][3\mu, 4\mu-1]$ and $t \in [2\mu, 4\mu-1][4\mu, 5\mu-1]$, in which case the type-2 submesh $[2\mu, 6\mu-1]^2$ contains s, t;

(c) $s \in [\mu, 2\mu-1][3\mu, 4\mu-1]$ and $t \in [2\mu, 3\mu-1][4\mu, 5\mu-1]$, in which case the type-2 submesh $[\mu, 3\mu-1][3\mu, 5\mu-1]$ at height $\lceil \log \mathsf{dist}(s,t) \rceil + 1$ contains s, t;

(d) $s \in [2\mu, 3\mu-1][3\mu, 4\mu-1]$ and $t \in [\mu, 2\mu-1][4\mu, 5\mu-1]$, which is similar to (c).

In all cases, s and t are contained in a submesh of height at most $\lceil \log \mathsf{dist}(s,t) \rceil + 2$.

To complete the argument, we now suppose that the network is a mesh (instead of a torus). If the ancestor submesh constructed in the torus is also a regular submesh in the mesh, then there is nothing to prove. So, assume that the ancestor constructed in the torus is not a regular submesh of the mesh. In particular, the ancestor constructed on the torus must be composed of two type-2 submeshes, on opposite sides of the mesh. If s, t are both contained in one of these submeshes, then they are both contained in a type-2 submesh of height at most $\lceil \log \mathsf{dist}(s,t) \rceil + 2$. The only remaining case is that s is in one of these submeshes and t is in the other. In this case, $\mathsf{dist}(s,t) \geq 2^{k-1}$, and since $\mu \geq \mathsf{dist}(s,t)$, we have that $\mu \geq 2^{k-1}$, or that $4\mu \geq 2^{k+1}$ which contradicts the assumption that $4\mu < 2^k$, concluding the proof. ■

C. Path Selection

Given the access graph, the procedure to determine a path from a given source s to a destination t is summarized in the following algorithm.

Path Selection Algorithm:

Input: Source s and destination t in the mesh M;

Output: Path $p(s,t)$ from s to t in M;

1: Let $(u_0, \ldots, u_l)$ denote a bitonic path in $G(M)$ from $g^{-1}(s)$ to $g^{-1}(t)$;
2: **for** $i = 0$ to l **do**
3: Select a node v_i in $g(u_i)$ uniformly at random; $\{v_0 = s$ and $v_l = t\}$
4: **if** $1 \leq i \leq l$ **then**
5: Construct subpath r_i from v_{i-1} to v_i by picking a dimension by dimension shortest path [2] (an at most one-bend path), according to a random ordering of the dimensions;
6: The path $p(s,t)$ is obtained by concatenating the subpaths r_i, $p(s,t) = r_0 r_1 \cdots r_{l-1}$;

Note that the algorithm is oblivious and local, since each source-destination pair can obtain a path independently of the other paths. We will now show that our algorithm with the generalized access graph, in addition to obtaining optimal congestion, also controls the stretch. First we show the constant stretch property of the selected paths.

Theorem 3.4: For any two distinct nodes s, t, $\mathsf{stretch}(p(s,t)) \leq 64$.

Proof: Let h be the height of the deepest common ancestor of s and t. Then $p(s,t)$ is the concatenation of paths constructed from the dimension by dimension paths in meshes of sides $2^1, \ldots, 2^{h-1}, 2^h, 2^{h-1}, \ldots, 2^1$. A path in a mesh of side ℓ has length at most $2\ell - 1$, so by adding the lengths of these paths, we have that $|p(s,t)| \leq 2(2^1 + \cdots + 2^h + 2^h + \cdots + 2^1 - 2h)$ which implies that $|p(s,t)| \leq 2^{h+3} - 4h$. Since s and t are distinct, $h \geq 1$. By Lemma 3.3, $h \leq \log \mathsf{dist}(s,t) + 3$, and the theorem follows. ∎

We now relate the congestion of the paths selected to the optimal congestion C^*. Let e denote an edge in M. Let $C(e)$ denote the load on e, i.e., the number of times that edge e is used by the paths of all the packets. We will get an upper bound on $E[C(e)]$, and then using a Chernoff bound we will obtain a concentration result.

We start by bounding the probability that some particular subpath formed by the path selection algorithm uses edge e. Consider the formation of a subpath r_i from a submesh M_1 to a submesh M_2, such that M_2 completely contains M_1, and e is a member of M_2. According to the path selection algorithm, mesh M_1 is of type-1, thus all of its sides are equal to m_ℓ, where ℓ is the level of M_1. We show the following lemma.

Lemma 3.5: Subpath r_i uses edge e with probability at most $2/m_\ell$.

Proof: For subpath r_i, let v_1 be the starting node in M_1 and v_2 the ending node in M_2. Suppose $e = (v_3, v_4)$. Without loss of generality, suppose e is vertical. Since the subpath is a one-bend path, edge e can be used only when either v_1 or v_2 have the same x coordinate as e. This event occurs with probability at most $2/m_\ell$. ∎

Let P' be the set of paths that go from M_1 to M_2 or vice-versa. Let $C'(e)$ denote the congestion that the packets P' cause on e. We show:

Lemma 3.6: $E[C'(e)] \leq 2|P'|/m_\ell$.

Proof: We can write $P' = P_1 \cup P_2$, where P_1 is the set of subpaths from M_1 to M_2, and P_2 is the subpaths from M_2 to M_1. Then, from Lemma 3.5, the expected congestion on edge e due to the subpaths in P_1 is bounded by $2|P_1|/m_\ell$. Using a similar analysis, the expected congestion on e due to subpaths in P_2 is bounded by $2|P_2|/m_\ell$. Since the congestion on e due to the paths in P' is the sum of the congestions due to P_1 and P_2, we obtain $E[C'(e)] \leq 2(|P_1| + |P_2|)/m_\ell = 2|P'|/m_\ell$. ∎

From the definition of the boundary congestion, we have that $B \geq B(M_1, \Pi) \geq |P'|/\mathsf{out}(M_1)$. Therefore, $C^* \geq |P'|/\mathsf{out}(M_1)$. Since each side of M_1 has m_ℓ nodes, we have that $\mathsf{out}(M_1) \leq 4m_\ell$. From Lemma 3.6, we therefore obtain:

Lemma 3.7: $E[C'(e)] \leq 8C^*$.

We "charge" this congestion to submesh M_2. By Lemma 3.3, only submeshes up to height $h < \log D^* + 3$ can contribute to the congestion on edge e (submeshes of type-1). By summing the congestions due to these at most $2(\log D^* + 3)$ submeshes (a type-1 and a type-2 submesh at each level), and by using Lemma 3.7, we arrive at an upper bound for the expected congestion on edge e:

Lemma 3.8: $E[C(e)] \leq 16C^*(\log D^* + 3)$.

Note that without increasing the expected congestion, we can always remove any cycles in a path, so without loss of generality, we will assume that the paths obtained are acyclic. We now obtain a concentration result on the congestion C obtained by our algorithm, using the fact that every packet selects its path independently of every other packet.

Theorem 3.9: $C = O(C^* \log n)$ with high probability.

Proof: Let $X_i = 1$ if path p_i uses edge e, and 0 otherwise. Then $E[C(e)] = E[\sum_i X_i] \leq 16C^*(\log D^* + 3)$.

[2]The dimension by dimension shortest path is the path which moves in the direction of the destination by as much as possible in one dimension, and then in another and so on until reaching the destination. In 2 dimensions, this is just a manhattan path.

Let $|E|$ be the number of edges in the mesh. For $|E| > 8$, $E[C(e)] \leq 16C^* \log(|E|D^*)$. Let $\kappa > 2e$, then applying a Chernoff bound [15], and using the fact that $C^* \geq 1$ we find that $P[C(e) > 16\kappa C^* \log(|E|D^*)] < (|E|D^*)^{-16\kappa}$. Taking a union bound over all the edges, we obtain

$$P[\max_{e \in E} C(e) > 16\kappa C^* \log(|E|D^*)] < \frac{1}{(|E|D^*)^{16\kappa - 1}}.$$

Using the fact that $D^* = O(|E|)$, $|E| = O(n^2)$, and choosing $\kappa = 2e + 1$, we get $C = O(C^* \log n)$ with high probability. ■

IV. The d-Dimensional Mesh

The 2-dimensional decomposition can be directly generalized to a d-dimensional mesh with equal side lengths (2^k, $k \geq 0$) in each dimension. The type-1 meshes would then be translated in each of the dimensions independently. Thus, there would be $2^d - 1$ type-2 bridge submeshes corresponding to a particular type-1 submesh. The stretch would be $O(d)$, however, the congestion becomes $O(2^d C^* \log n)$, which is excessively high for large d. The reason for this exponential behavior of the congestion is that there are now 2^d different submeshes which could contain an edge e at every level. Thus, summing up the congestion contributed by each of these submeshes to e over all the $\log n$ levels gives an additional factor of $2^d \log n$ in the congestion. In order to alleviate the problem, we present an alternative decomposition using a similar idea to the bridge (type-2) submesh for which the path selection algorithm has congestion $O(dC^* \log n)$, and stretch $O(d^2)$. Thus, by sacrificing a little in the stretch, we can drastically improve the congestion. For fixed d, these are constant factors from optimal. The main idea is that the translated submeshes (corresponding to the type-2 submeshes in the 2-dimensional case) are not translated independently in every dimension. Instead they are translated simultaneously in every dimension, and hence at every level there are now only a linear (in d) number of submeshes which contain an edge e. In order to accomplish this, we now have to introduce more types of bridge submeshes.

A. Decomposition

As in two dimensions, we have the type-1 meshes, and the translated meshes. Recall that the type-1 submeshes at level $\ell+1$ are obtained by dividing the type-1 submeshes in the previous level ℓ by 2 in every dimension *simultaneously* (as opposed to one dimension at a time as in [13]), resulting in 2^d type-1 submeshes at level $\ell+1$. Each type-1 submesh is translated to obtain the other types of bridge submeshes which we will need in our path selection algorithm. In fact, we now introduce $\Theta(d)$ types of translated meshes at each level. To be specific, consider the level ℓ type-1 d-dimensional submesh with side length $m_\ell = 2^{k-\ell}$. Set $\lambda_\ell = \max\{1, m_\ell/2^{\lceil \log(d+1) \rceil}\}$. Since $\log(d+1) \leq \lceil \log(d+1) \rceil \leq 1 + \log(d+1)$,

Lemma 4.1: $\max\{1, m_\ell/2(d+1)\} \leq \lambda_\ell \leq \max\{1, m_\ell/(d+1)\}$.

We shift the type-1 submeshes by $(j-1)\lambda_\ell$ nodes in each dimension to get the type-j submeshes, for $j > 1$. If a resulting type-j submesh is not contained in the mesh M, we only retain that part of it that is in the mesh. Since we discard some part of a submesh after translation (if that part is not within the mesh), the resulting type-j submeshes for $j > 1$ may not be square. It may also be that a resulting level ℓ type-j submesh for $j > 1$ may be equal to a type-1 submesh at a higher level, but this will not affect our arguments. We refer to any of these type-j submeshes for $j = 1, 2, \ldots$ as a regular submesh.

Lemma 4.2: For $m_\ell \geq d+1$, the number of types of submeshes is at least $d+1$ and at most $2(d+1)$. For $m_\ell < d+1$, the number of different types of submeshes is m_ℓ.

Proof: If $m_\ell < d+1$, then $\lambda_\ell = 1$, and the type-j submeshes for $j > 1$ are the type-1 submeshes translated by $j-1$. After translating by m_ℓ, we recover the original type-1 submeshes, so there is no type-$(m_\ell+1)$ submesh, i.e., there are m_ℓ types of regular submeshes.

If $m_\ell \geq d+1$, since m_ℓ is a power of 2, $m_\ell \geq 2^{\lceil \log(d+1) \rceil}$. Thus $\lambda_\ell = m_\ell/2^{\lceil \log(d+1) \rceil}$. After translating type-1 submeshes by $\frac{m_\ell}{\lambda_\ell} \cdot \lambda_\ell$, we recover the type-1 submeshes. Thus, there are m_ℓ/λ_ℓ types of submeshes. To conclude, we use Lemma 4.1 and the fact that $\lambda_\ell = m_\ell/2^{\lceil \log(d+1) \rceil}$. ■

Thus, an edge is contained in $\Theta(d)$ submeshes at every level. Figure 2 shows an example for $d = 3$, $m_l = 4$, and $\lambda_\ell = \frac{m_l}{4} = 1$ (only a two dimensional slice is shown). The next lemma shows that the type-j submeshes at level ℓ can be partitioned into type-1 (square) submeshes of side λ_ℓ. The level of these type-1 submeshes is $\min\{k, \ell + \lceil \log(d+1) \rceil$.

Lemma 4.3: Type-j submeshes at level ℓ or lower can be partitioned into type-1 submeshes of side λ_ℓ.

Proof: If $\lambda_\ell = 1$ then there is nothing to prove, so assume that $\lambda_\ell = m_\ell/2^{\lceil \log(d+1) \rceil}$. We define the anchor node of a submesh as the node with the smallest coordinate in every dimension. The anchor node of any type-j submesh

is the translation of an anchor node of a type-1 submesh by $a \cdot \lambda_\ell$ in every dimension (where a is an integer). It follows that the anchor node of every type-j submesh is an anchor node of a type-1 submesh of side λ_ℓ. The claim follows now because every side length of a type-j submesh is a power of two times λ_ℓ. A similar argument holds for any regular submesh at a lower level than ℓ. ■

The next observation is exactly analogous to the 2-dimensional case:

Lemma 4.4: Every type-1 submesh at level ℓ can be partitioned into type-1 submeshes at any level $\ell' > \ell$.

If $\lambda_\ell = m_\ell / 2^{\lceil \log(d+1) \rceil}$, then $\lambda_{\ell-1} = 2\lambda_\ell$. From this observation, and the fact that the smallest side-length of a submesh is λ_ℓ we obtain the following useful claim:

Lemma 4.5: The smallest side-length of a level $\ell - 1$ regular submesh is at least twice λ_ℓ.

The access graph can be constructed using the regular submeshes in exactly the same way as we did for the 2-dimensional case. Specifically, the root node is the entire d-dimensional mesh. The parents of any regular submesh at level $\ell + 1$ are all regular submeshes at level ℓ which contain that submesh. The only difference is that we now also add ancestor links from every type-1 regular submesh to every other regular submesh that contains it. We will see the reason for this later, when we construct the bitonic paths.

As in the 2-dimensional case, we will now show that any two nodes are enclosed in a regular submesh of small height. This result will be instrumental in obtaining a path selection algorithm which controls the stretch. Suppose we are given two nodes s and t in the mesh. We will show that in the access graph there is some regular submesh (of some type-j) that completely contains s and t, and has side length $O(d \cdot \mathsf{dist}(s,t))$. Let $s = (s_1, \ldots, s_d)$ and $t = (t_1, \ldots, t_d)$. Clearly, $|t_i - s_i| \leq \mathsf{dist}(s,t)$. Let $R = [a_1, b_1] \cdots [a_d, b_d]$ be a region of the mesh defined by s and t, such that $[a_i, b_i] = [\min(s_i, t_i), \max(s_i, t_i)]$. Note that R contains s and t, and is a submesh of M. However, R may not be a regular submesh. We will find a regular submesh which completely contains R. Consider the deepest level that has type-1 submeshes of side at least $2(d+1) \cdot \mathsf{dist}(s,t)$. Let h be the height of this level. The side length of the type-1 submeshes at height h is $m_h = 2^h$, where,

$$4(d+1) \cdot \mathsf{dist}(s,t) > m_h \geq 2(d+1) \cdot \mathsf{dist}(s,t),$$

Let the shifting parameter λ for the type-j submeshes at this height be $\lambda_h = m_h / 2^{\lceil \log(d+1) \rceil} > m_h / 2^{1+\log(d+1)} \geq \mathsf{dist}(s,t)$ (the strict inequality is because $\lceil \log(d+1) \rceil < 1 + \log(d+1)$). We have:

Lemma 4.6: For any height $\ell \geq h$, R is contained in some regular submesh at height ℓ.

Proof: First consider height h. The *anchor* node of a submesh is the node with the smallest coordinate in each dimension. Consider dimension i. Let $\Delta_i = [a_i, b_i]$, be the side of R in dimension i, and $|\Delta_i| = b_i - a_i$. As we did for $d = 2$, assume first that we are on the torus.

By the definition of h, $\lambda_h > \mathsf{dist}(s,t)$, and $|\Delta_i| \leq \mathsf{dist}(s,t)$. Thus, by construction of the different types of submeshes, the interval Δ_i can contain the ith dimension of the anchor node of at most one type of submesh at height h. This is because the spacing between the ith dimension of the anchor nodes of the submeshes is $\lambda_h > \mathsf{dist}(s,t) \geq |\Delta_i|$. Since we have d dimensions and at least $d+1$ different types of submeshes, by the pigeonhole principle, there exists at least one type of submesh, say type-ζ, such that R does not contain the anchor node of any type-ζ submesh in any dimension. This implies that R is completely contained by a type-ζ submesh M' at height h, because the type-ζ submeshes provide a disjoint partition of the mesh.

Exactly the same argument that was used to extend the 2-dimensional torus result to the mesh in Lemma 3.3 can be used here as well, and we do not repeat it here. An identical argument can also be applied to heights $> h$. ■

B. Path Selection

The path selection is also similar to the 2-dimensional case, where the bridge submesh can be any type-j submesh. Specifically, the path is a bitonic path, in which all submeshes corresponding to nodes in the access graph are type-1, except for the bridge submesh. The bridge submesh will contain both the source and destination, and among all possible bridge submeshes, we will use one with small height.

Let $s = (s_1, \ldots, s_d)$ and $t = (t_1, \ldots, t_d)$, and define the region R to be the smallest volume submesh that contains both s and t, as in the previous section. From Lemma 4.6, it follows that there is some regular submesh M_2' at height h that contains R, where

$$4(d+1) \cdot \mathsf{dist}(s,t) \geq 2^h \geq 2(d+1) \cdot \mathsf{dist}(s,t).$$

Let $\lambda_h = 2^h / 2^{\lceil \log(d+1) \rceil} \geq \mathsf{dist}(s,t)$ be the shifting parameter for this height. Again, by Lemma 4.6, there is a regular submesh M_2 at height $h+1$ which contains R. For the path selection algorithm, we will use the bridge submesh

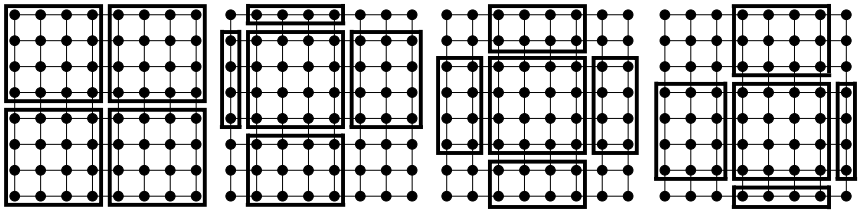

Fig. 2. Mesh decomposition for the 3−dimensional mesh. Only 2 of the 3 dimensions are depicted

M_2. By Lemma 4.3, M_2 can also be partitioned into type-1 submeshes of side λ_h. Since $\lambda_h \geq \mathsf{dist}(s,t)$, it follows that the height of these type-1 submeshes which partition M_2 is at least $\log \mathsf{dist}(s,t)$, therefore the type-1 submeshes at height $h' = \lfloor \log \mathsf{dist}(s,t) \rfloor$ also partition M_2. Exactly analogous to the path construction for 2-dimensions, we now construct a bitonic path on the access graph from $g^{-1}(s)$ to $g^{-1}(t)$ through this bridge submesh M_2 at height $h+1$. Except possibly for M_2, the other submeshes on the path are of type-1. Let M_1 be the type-1 submesh at height h' that contains s, and M_3 the type-1 submesh at height h' that contains t. The path is formed by first using submeshes of type-1 from s up to submesh M_1, one level at a time; the path then jumps from M_1 to the bridge submesh M_2, and from M_2 down to M_3; finally the path progresses down to t by using type-1 submeshes, one level at a time. Note that unlike in the 2-dimensional case, the jump from M_1 to M_2 is not a single level, but could be multiple levels, which is the reason for adding ancestor links to the access graph for the type-1 submeshes. Since the side-length of the type-1 submeshes in M_1 and M_3 is at most λ_h, by Lemma 4.5 the smallest side-length of M_2 is at least twice the side-length of the type-1 submeshes in M_1 and M_3. This is the reason for using M_2 instead of M_2' as the bridge submesh. We have the following lemma, which will be important in the congestion analysis:

Lemma 4.7: The side-length increases (in every dimension) by a factor of a least 2 in the bitonic path from $g^{-1}(s)$ up to M_2 and decreases a factor of at least 2 from M_2 down to $g^{-1}(t)$.

C. Stretch and Congestion

First, we compute the stretch of the path selection algorithm in d dimensions.

Theorem 4.8 (Stretch for d Dimensions): For any two distinct nodes s and t of the mesh, $\mathsf{stretch}(p(s,t)) = O(d^2)$.

Proof: Let p denote the path from s to t. Let r_1, r_2, and r_3, denote the respective subpaths from s to M_1, from M_1 to M_2 to M_3, and from M_3 to t. First we compute $|r_1|$. Subpath r_1 consists of h' subpaths where the subpath from height $i-1$ to height i has length at most $d(2^i - 1)$. Therefore, $|r_1| \leq 2d \sum_{i=0}^{h'} (2^i - 1) = 2d(2 \cdot 2^{h'} - 1 - h')$. Since $2^{h'} \leq \mathsf{dist}(s,t)$, we have $|r_1| \leq 2d(2 \cdot \mathsf{dist}(s,t) - 1 - h') = O(d \cdot \mathsf{dist}(s,t))$. The path length of r_2 is no more than $2d(2^{h+1} - 1)$. Since $2^{h+1} \leq 8(d+1) \cdot \mathsf{dist}(s,t)$, we have that $|r_2| \leq 2d(8(d+1) \cdot \mathsf{dist}(s,t) + 1) = O(d^2 \cdot \mathsf{dist}(s,t))$. Lastly, the maximum possible length of r_3 and r_1 are the same. Thus, $|p| = |r_1| + |r_2| + |r_3| = O(d^2 \cdot \mathsf{dist}(s,t))$. ■

We will use an analysis similar to the 2-dimensional case to bound the expected congestion caused on an edge e. Specifically, from subpaths that use edge e in going from a mesh of a lower level to a mesh of a higher level, the contribution to the expected congestion is $E[C'(e)] \leq 4C^*$.

Let M_1 and M_2 be two submeshes with respective side lengths $a_1, \ldots, a_d$ and $b_1, \ldots, b_d$, such that: (i) M_1 is of type-1, (ii) M_2 completely contains M_1, (iii) each side of M_2 is at least twice the side of M_1, $b_i \geq 2a_i$, for all $1 \leq i \leq d$. Note that d-dimensional algorithm preserves conditions (i)-(iii), for every pair of consecutive submeshes of the bitonic path (condition (iii) is preserved even when M_2 is a bridge submesh because we chose the height of the bridge to be $h+1$ (Lemma 4.7)).

Consider the formation of a subpath r from a submesh M_1 to M_2, where r is formed by following a shortest path from a randomly chosen node v_1 in M_1 to a randomly chosen node v_2 in M_2. The path is formed by following a dimension by dimension path, with a random ordering of dimensions. Since M_1 is of type-1, it holds that all of its sides are equal, $a_1 = a_2 = \cdots = a_d = a$. Let e be an edge

of M_2. We show the following result:[3]

Lemma 4.9: Subpath r uses edge e with probability at most $\frac{2}{da^{d-1}}$.

Proof: Without loss of generality, suppose that e is an edge in M_2 along dimension l, from $(x_1, \ldots, x_l, \ldots, x_d)$ to $(x_1, \ldots, x_l + 1, \ldots, x_d)$. Suppose $v_1 = (y_1, \ldots, y_d)$ and $v_2 = (z_1, \ldots, z_d)$. Let ρ be a random permutation which determines the order of dimensions. Suppose that dimension l occurs in position i, i.e., $\rho(i) = l$. Then the probability $P[e]$ that edge e is used by r is bounded from above by the probability that, $z_{\rho(1)} = x_{\rho(1)}, \ldots, z_{\rho(i-1)} = x_{\rho(i-1)}$, and, $y_{\rho(i+1)} = x_{\rho(i+1)}, \ldots, y_{\rho(d)} = x_{\rho(d)}$. We obtain:

$$\begin{aligned} P[e] &\leq \frac{1}{b_{\rho(1)} \cdots b_{\rho(i-1)}} \cdot \frac{1}{a_{\rho(i+1)} \cdots a_{\rho(d)}} \\ &\leq \frac{1}{2a_{\rho(1)} \cdots 2a_{\rho(i-1)}} \cdot \frac{1}{a_{\rho(i+1)} \cdots a_{\rho(d)}} \\ &= \frac{1}{2^{i-1} a^{d-1}}. \end{aligned}$$

Since ρ is a random permutation, the probability that $\rho(i) = l$ for any i is $1/d$, so multiplying by $1/d$, summing, and using the fact that $\sum_{i=1}^{d} \frac{1}{2^{i-1}} \leq 2$, we obtain:

$$P[e] \leq \frac{1}{d} \sum_{i=1}^{d} \frac{1}{2^{i-1} a^{d-1}} = \frac{1}{da^{d-1}} \sum_{i=1}^{d} \frac{1}{2^{i-1}} \leq \frac{2}{da^{d-1}}.$$

■

Note that the appearance of $\frac{1}{d}$ in the proof arises because of the randomized dimension by dimension routing. In the original result in [13], this randomized dimension by dimension routing was not used (a lexicographic ordering of the dimensions was used). Adding this minor change to the original algorithm would yield a factor of d savings. Note that this factor of d savings can also be achieved by alternate means, see for example [17, p.46-48]. Let P' be the set of paths that go from M_1 to M_2 or vice-versa. Let $C'(e)$ denote the congestion that the packets P' cause on e. We then have that:

Lemma 4.10: $E[C'(e)] \leq \frac{2|P'|}{da^{d-1}}$.

Proof: We can write $P' = P_1 \cup P_2$, where P_1 is the set of subpaths from M_1 to M_2, and P_2 is the subpaths from M_2 to M_1. Then, from Lemma 4.9, the expected congestion at edge e due to the subpaths in P_1 is bounded by $2|P_1|/(da^{d-1})$. With a similar analysis, we have that the expected congestion at e is bounded by $2|P_2|/(da^{d-1})$. Since the congestion on e is the sum of congestions due to P_1 and P_2, we obtain $E[C'(e)] \leq 2(|P_1| + |P_2|)/(da^{d-1}) = 2|P'|/(da^{d-1})$. ■

We have that $C^* \geq B(M_1, \Pi) \geq |P'|/\mathsf{out}(M_1)$. Since each side of M_1 has a nodes, we have that $\mathsf{out}(M_1) \leq 2da^{d-1}$. From Lemma 4.10, we obtain:

Lemma 4.11: $E[C'(e)] \leq 4C^*$.

We charge this congestion to the higher height mesh. Since paths use heights up to $O(\log(dD^*))$, and each height has $O(d)$ different types of submeshes, at most $O(d \log(dD^*))$ different submeshes can contribute to the congestion on e. Thus, $E[C(e)] = O(dC^* \log(dD^*))$. As with the 2-dimensional analysis, we get a concentration result by applying a Chernoff bounding argument and the facts that $dD^* = O(|E|)$, $|E| = O(dn)$, and $d = O(n)$:

Theorem 4.12 (Congestion for d Dimensions): $C = O(dC^* \log n)$ with high probability.

V. Randomization Requirements

Here we show that randomization is unavoidable for oblivious algorithms, if they are to obtain near optimal congestion: such algorithms need access to a substantial number of random bits, and we show that our algorithm uses a near minimal number of random bits. First we formally define the randomization requirements of a routing algorithm by the number of path choices it has when selecting a path from a source to a destination.

Let A be an oblivious routing algorithm. For a given pair of nodes s, t, let $\kappa(s, t)$ be the set of paths from s to t that it will select from in determining which path to use for a given packet from s to t. We refer to $\kappa(s, t)$ as the *choice set* for s and t. Two different packets from the same source to destination may select different paths in this set, i.e., algorithm A selects a path from $\kappa(s, t)$ randomly according to some probability distribution. If $|\kappa(s, t)| = 1$ for every source destination pair, then algorithm A is deterministic. We denote by $\kappa(\ell)$, the maximum size of the choice set over all source-destination pairs that are distance ℓ apart, $\kappa(\ell) = \max_{\mathsf{dist}(s,t)=\ell} |\kappa(s, t)|$. We refer to $\kappa(\ell)$ as the choice function of the algorithm, and we refer to algorithm A as a κ-choice algorithm. For routing problems with dilation D^*, there are source-destination pairs which require a selection to be made from $\kappa(D^*)$ choices. Thus such an algorithm requires access to $\Omega(\log \kappa(D^*))$ random bits per packet in the worst case. We will establish a lower bound on $\kappa(\ell)$ for any oblivious algorithm which attains near optimal congestion, thereby establishing a lower bound

[3]Note that this result cannot be used for the 2-dimensional analysis of Section III-C, since the condiction (iii) listed above does not hold for that algorithm – the bridge submesh may not have side-lengths in every dimension larger than twice the higher level type-1 submesh.

on the number of random bits per packet required by *any* oblivious algorithm that obtains near optimal congestion.

A. Path Choices and Congestion

Given an oblivious $\kappa(\ell)$-choice algorithm A, we now construct a routing problem and obtain a lower bound on the congestion that algorithm A attains on this routing problem. Consider a particular value of ℓ, and consider the d-dimensional mesh with side length $m = 2\ell$ in each dimension. The routing problem we construct will be specific to the algorithm A, and will be constructed using algorithm A itself. We will thus refer to this problem as $\Pi_A(\ell)$.

Divide the mesh into 2 submeshes by dividing the first dimension of the mesh into two intervals of side length ℓ. We refer to these two submeshes as the left and right submesh. Each node in the network is the source of one packet and the destination of one packet (a permutation routing problem). The distance of every packet to its destination will be ℓ, which is accomplished as follows. Each source in the left submesh has destination in the right submesh, and vice versa. In particular, the source $(x_1, \ldots, x_d)$ in the left submesh ($x_1 \leq \ell$) has destination $(x_1 + \ell, \ldots, x_d)$ in the right submesh, and the source $(x_1, \ldots, x_d)$ in the right submesh ($x_1 > \ell$) has destination $(x_1 - \ell, \ldots, x_d)$ in the left submesh.

We continue the construction of the routing problem as follows. For every source s and destination t pair, algorithm A selects one of the paths $p(s,t)$ in $\kappa(s,t)$ with maximum probability (in the case that more than one path has the maximum probability to be selected, we arbitrarily choose one of them). The probability that this path is selected is at least $1/|\kappa(s,t)|$. Now consider the routing in which every packet with source s and destination t uses its most probable path $p(s,t)$. Thus, we have defined a possible routing which solves this routing problem. The number of nodes in the mesh is $(2\ell)^d$, which equals the number of sources, hence the number of edges traversed by the packets in this routing is at least $\ell \cdot (2\ell)^d$, since $|p(s,t)| \geq \ell$. The number of edges in this network is $d \cdot (2\ell)^d \cdot (1 - \frac{1}{2\ell})$. We therefore conclude, by the pigeon-hole principle, that at least one edge is used at least C_A times, where

$$C_A = \frac{\ell \cdot (2\ell)^d}{d \cdot (2\ell)^d \cdot (1 - \frac{1}{2\ell})} = \frac{\ell}{d \cdot (1 - \frac{1}{2\ell})} \geq \frac{\ell}{d}.$$

We are now ready to complete the construction of the routing problem $\Pi_A(\ell)$. Let e be one of the edges which is crossed by C_A packets. Arbitrarily select ℓ/d of these packets (which is possible, since $C_A \geq \ell/d$). The sources and destinations in the routing problem $\Pi_A(\ell)$ are exactly the sources and destinations of these ℓ/d packets that were selected. This concludes the construction of $\Pi_A(\ell)$. Note that $\Pi_A(\ell)$ keeps only a subset of the packets in the original permutation problem. We now give a result that relates the number of path choices $\kappa(\ell)$ of algorithm A to the congestion that algorithm A achieves on routing problem $\Pi_A(\ell)$.

Lemma 5.1: Consider any κ-choice routing algorithm A and any ℓ. The expected congestion Y achieved by algorithm A for routing problem $\Pi_A(\ell)$ is at least $\frac{\ell}{d\kappa(\ell)}$.

Proof: By the construction of Π_A, there is an edge e with the following property. For every source s and destination t in $\Pi_A(\ell)$, there is a path $p(s,t) \in \kappa(s,t)$ such that the probability that A selects this path $p(s,t)$ is at least $1/\kappa(s,t)$. Consider the expected congestion on edge e:

$$E[C(e)] \geq \sum_{(s,t)} \frac{1}{\kappa(s,t)} \geq \sum_{(s,t)} \frac{1}{\kappa(\ell)} = \frac{|\Pi_A|}{\kappa(\ell)}.$$

To conclude, note that $|\Pi_A| = \ell/d$ and that the expected congestion is at least the expected congestion on edge e, i.e., $Y = E[\max_{e_i} C(e_i)] \geq E[C(e)]$. ∎

B. Lower Bound on Randomization Requirements

By comparing the congestion achieved by any κ-choice algorithm with the optimal congestion, we will obtain a lower bound on the number of choices, $\kappa(\ell)$, that the algorithm must make if it is to obtain near optimal congestion for any routing problem on the d-dimensional mesh. This result will also establish a lower bound on the number of bits per packet required by any algorithm that achieves near optimal congestion. We need to obtain an upper bound on the optimal congestion, and we will use the congestion attained by our oblivious routing algorithm as the upper bound. We will refer to our hierarchical d-dimensional routing algorithm as algorithm H (presented in Section IV).

Consider an arbitrary κ-choice algorithm A, fix ℓ, and construct the corresponding routing problem $\Pi_A(\ell)$ as described in the previous section. Note that $|\Pi_A| = C_A \geq \ell/d$. We now give an upper bound on the congestion attained by algorithm H for routing problem $\Pi_A(\ell)$. Let C_H denote the expected congestion of H for Π_A. We will give an upper on C_H in terms of ℓ and d.

Lemma 5.2: $C_H = O\left((\ell/d)^{\frac{1}{d}} \cdot \log n\right)$.

Proof: From the analysis in Sections III and IV, $C_H = O(dB \log n) = O(dC^* \log n)$, where B is the boundary

congestion. We give an upper bound on B in terms of l and d.

For any arbitrary submesh M', we give an upper bound on $B(M', \Pi_A)$, which is also an upper bound for B. Assume M' is a $m_1 \times \cdots \times m_d$ with n' nodes. It can be shown that $\mathsf{out}(M') \geq d \cdot {n'}^{\frac{d-1}{d}}$. Let Π' denote the packets that have to cross the border of M' (have source outside and destination inside M' or vice versa). Clearly, $|\Pi'| \leq |\Pi_A| = C_A = \ell/d$. Also, $|\Pi'| \leq 2n'$, since a node is the source of at most one packet (which may have destination outside M'), and the destination of at most one packet (which may have source outside M'). Thus, $|\Pi'| \leq \min\{\ell/d, 2n'\}$. By definition of the boundary congestion, it can be easily shown that:

$$B(M', \Pi_A) = \frac{|\Pi'|}{\mathsf{out}(M')} \leq \frac{\min\{\frac{\ell}{d}, 2n'\}}{d \cdot {n'}^{\frac{d-1}{d}}} < \frac{2 \cdot \ell^{\frac{1}{d}}}{d^{(1+\frac{1}{d})}}.$$

Since M' was arbitrary, $B < 2\ell^{1/d}/d^{(1+\frac{1}{d})}$. To conclude, since $C_H = O(dB \log n)$, we have that $C_H = O\left((\ell/d)^{\frac{1}{d}} \cdot \log n\right)$. ∎

We know from Lemma 5.1 that Y, the expected congestion for algorithm A on routing problem Π_A, is $\Omega(\ell/(d\kappa(\ell)))$. Suppose that algorithm A attains congestion at least as low as C_H for any routing problem. It must be that $\ell/(d\kappa(\ell)) = O((\ell/d)^{\frac{1}{d}} \cdot \log n)$, or that:

Lemma 5.3: For any oblivious algorithm that achieves as good congestion as our algorithm on arbitrary routing problems, $\kappa(\ell) = \Omega\left(\left(\frac{\ell}{d}\right)^{1-\frac{1}{d}} \frac{1}{\log n}\right)$.

Lemma 5.3 gives a lower bound on the per packet bit requirement for any algorithm which is as good as ours. Let $r(\ell) = \log \kappa(\ell)$; $r(\ell)$ quantifies the randomization requirements of a routing algorithm, and is the maximum number of random bits needed by an algorithm to construct the path between a source and destination pair that are distance ℓ apart. Then, we have the following theorem:

Theorem 5.4: For any oblivious algorithm that achieves as good congestion as our algorithm on arbitrary routing problems, $r(\ell) = \Omega\left((1 - \frac{1}{d}) \log \frac{\ell}{d} - \log\log n\right)$.

Since our algorithm achieves near optimal congestion on arbitrary routing problems, i.e., $C_H = \tilde{O}(C^*)$, any oblivious algorithm which has near-optimal congestion must use this many bits. Specifically, for any near optimal oblivious algorithm, there exist source destination pairs that are a distance ℓ apart for which the algorithm requires $\Omega\left(\log \frac{\ell}{d} - \log\log n\right)$ random bits per packet (where we have used $1 - \frac{1}{d} = \Omega(1)$). We thus have:

Corollary 5.5: For any oblivious algorithm that achieves asymptotically optimal congestion on arbitrary routing problems, $r(\ell) = \Omega\left(\log \frac{\ell}{d} - \log\log n\right)$. When $\ell/d = \omega(\log n)$, $r(\ell) = \Omega(\log \ell)$.

Corollary 5.5 shows that, for any near optimal oblivious algorithm, there exist routing problems for which significant randomization is used.

C. The Randomization Requirement of Algorithm H

We now compare the randomization requirement for our hierarchical algorithm H with the lower bound. First, we compute an upper bound on the number of random bits per packet required by our algorithm. We will then show how to reduce the randomization requirements of our algorithm while still maintaining its optimality properties. The conclusion is that, to within a factor of d, our algorithm uses the minimum number of bits possible.

Consider a path formation from a source to a destination that are distance ℓ apart. This path is the concatenation of subpaths from a submesh M_1 to a submesh M_2 (we will refer to the construction of each such subpath as a "step" in the algorithm). The algorithm makes the following random selections in a step:

i. a random ordering of the dimensions from the $d!$ possible orderings, requiring $O(d \log d)$ random bits each time;
ii. a random node in M_2. The largest submesh in the bitonic path has size $O((d\ell)^d)$ (Lemma 4.6), so the number of random bits required each time is $O(d \log(d\ell))$.

Consequently, in a single step of the algorithm, the number of bits required per step is $O(d \log(d\ell))$. Since, the number of steps to select a path is at most $2 \log \ell + 1$, the total number of random bits needed per packet is $O(d \log^2(d\ell))$.

We can decrease the number of random bits needed by a factor of $\log(dD^*)$, as follows:

i. We select the random order of dimensions only once, at the beginning of the path creation and we use the same order at each subsequent step of the algorithm for the same path.
ii. We independently select two random nodes, v_1 and v_2, in the largest submesh of the bitonic path. For the path formation we only use bits from v_1 and v_2 to compute the (random) nodes in the submeshes of the bitonic path. We use as many bits as necessary from v_1 or v_2, depending on the size of the submesh considered each time. We alternate the use of nodes v_1 and v_2

in the path formation, so that if the path is formed on submeshes $M_1, \ldots, M_k$, we use bits from v_1 for $M_1, M_3, \ldots$, (odd indices) and from v_2 for $M_2, M_4, \ldots$ (even indices).

The computation of the expected congestion on edge e due to the paths from M_1 to M_2 (Lemmas 4.9 and 4.10) only rely on independence between the node selections in consecutive submeshes of the bitonic path. Thus the use of the bits in v_1 and v_2 alternately guarantees that the nodes in consecutive submeshes in the bitonic path are selected independently, and hence the congestion result continues to hold. The paths for different packets are chosen independently, so the high probability result is also not affected.

Therefore, only $O(O(d \log d))$ random bits are needed for the random order of dimensions, and $O(d \log(d\ell))$ random bits to randomly select v_1 and v_2. This gives us in total $O(d \log(d\ell))$ bits. We thus have:

Lemma 5.6: For our hierarchical d-dimensional algorithm H, $r(\ell) = O(d \log(d\ell))$.

From Lemma 5.5 we have that for any near optimal oblivious routing algorithm, asymptotically in ℓ, $r(\ell) = \Omega(\log l)$, from which we conclude the following theorem,

Theorem 5.7 (Near Optimal Randomization Requirements): The number of random bits used by algorithm H is asymptotically within $O(d)$ of optimal.

VI. Discussion

We have shown that for the mesh, one can *simultaneously* control both the stretch and the congestion. Further, we show that for any oblivious algorithm that achieves near optimal congestion, there are routing problems on which it must use a significant amount of randomization. In our algorithm, we use an asymptotically optimal number of random bits when the number of dimensions d is fixed.

In general, the congestion and stretch cannot be minimized simultaneously, as we show below with an example (this is folklore knowledge and similar examples have been considered, see for example [17, p. 59]). Consider a network in which there are two main nodes u and v and n disjoint paths that connect u to v. The first path has length 1 (an edge which directly connects the main nodes), while the remaining paths have length n. So, there are $\Theta(n^2)$ nodes in the network. Now consider $N = n$ packets which all wish to go from u to v. If all the packets use the first path then the stretch is optimal, and the congestion is n which is a factor of n from optimal. Any other path assignment leads to stretch n, which is also a factor n from optimal. Thus both congestion and stretch cannot be optimized simultaneously (obliviously or not).

There exist non-oblivious algorithms which obtain near optimal congestion plus dilation on general networks (see for example [1], [2], [18], [20]) – i.e., minimizing congestion plus dilation does not imply simultaneously optimizing congestion and dilation. The next natural question is whether there exist oblivious algorithms which obtain near optimal (within polylogarithmic factors) congestion plus dilation on general networks. The answer to this question is also negative, and we can see this using the same network constructed in the previous example. The proof is given in [17, p. 59]. The idea behind the proof is to assume that there exists a universal oblivious algorithm which attains near optimal congestion plus dilation. To route 1 packet from u to v, let the algorithm choose the first path of length 1 with probability p. Then, $E[C + D] = 1 + p + n(1 - p)$. Since the optimal value of $C + D$ is 2, it follows that $p = 1 - O(polylog(n)/n)$. Now consider the algorithm's behavior for n^2 packets each with source u and destination v. Since the algorithm is oblivious, it must treat every packet independently, as if it were the only packet in the network, and hence each packet will use the first path with probability p. Thus, $E[C + D] = \Omega(E[C]) = \Omega(n^2 p) = \Omega(n^2)$. Note that the optimal value for $C + D$ is $O(n)$ which can be obtained by spreading out the packets, n on each of the n paths. Thus, this oblivious algorithm is a factor $\Omega(n)$ from optimal, which contradicts it being near optimal.

The conclusion is that stretch and congestion can be simultaneously minimized only on particular network topologies. Further, in the domain of oblivious routing, congestion plus dilation can also only be minimized on particular network topologies. We have shown the surprising result that congestion and stretch can be simultaneously controlled on the d-dimensional mesh (hence implying that their sum is also near optimal). Interesting open problems that remain are to develop similar algorithms for other types of networks.

References

[1] J. Aspens, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. Online load balancing with applications to machine scheduling and virtual circuit routing. In *Proc. 25th Symp. on the Theory of Computing*, pages 623–631, 1993.

[2] B. Awerbuch and Y. Azar. Local optimization of global objectives: competitive distributed deadlock resolution and resource allocation. In *Proc. 35th Symp. on Foundations of Computer Science*, pages 240–249, 1994.

[3] Y. Azar, E. Cohen, A. Fiat, H. Kaplan, and H. Racke. Optimal oblivious routing in polynomial time. In *Proc. 35th Symp. on Theory of Computing*, pages 383–388, 2003.

[4] M. Bienkowski, M. Korzeniowski, and H. Räcke. A practical algorithm for constructing oblivious routing schemes. In *Proc. 15th Symp. on Parallelism in Algorithms and Architectures*, pages 24–33, 2003.

[5] A. Borodin and J. E. Hopcroft. Routing, merging, and sorting on parallel models of computation. *Journal of Computer and System Science*, 30:130–145, 1985.

[6] C. Busch, M. Magdon-Ismail, and J. Xi. Oblivious routing on geometric networks. In *Proc. 17th Symp. on Parallelism in Algorithms and Architectures*, pages 316–324, 2005.

[7] C. Busch, M. Magdon-Ismail, and J. Xi. Optimal oblivious path selection on the mesh. In *Proc. 19th International Parallel and Distributed Processing Symposium*, pages 82–91, 2005.

[8] J. Gao and L. Zhang. Tradeoff between stretch factor and load balancing ratio in wireless network routing. In *Proc. 23rd Symp. on Principles of Distributed Computing*, pages 189 – 196, 2004.

[9] C. Harrelson, K. Hildrum, and S. Rao. A polynomial-time tree decomposition to minize congestion. In *Proc. 15th Symp. on Parallelism in Algorithms and Architectures*, pages 34–43, 2003.

[10] C. Kaklamanis, D. Krizanc, and T. Tsantilas. Tight bounds for oblivious routing in the hypercube. In *Proc. 2nd Symp. on Parallelism in Algorithms and Architectures*, pages 31–36, 1990.

[11] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays - Trees - Hypercubes*. 1992.

[12] F. T. Leighton, B. M. Maggs, and S. B. Rao. Packet routing and job-scheduling in $O(congestion + dilation)$ steps. *Combinatorica*, 14:167–186, 1994.

[13] B. M. Maggs, F. Meyer auf der Heide, B. Vöcking, and M. Westermann. Exploiting locality in data management in systems of limited bandwidth. In *Proc. 38th Symp. on the Foundations of Computer Science*, pages 284–293, 1997.

[14] F. Meyer auf der Heide, C. Schindelhauer, K. Volbert, and M. Grünewald. Congestion, dilation, and energy in radio networks. *Theory of Computing Systems*, 37(3):343–370, 2004.

[15] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge, UK, 2000.

[16] H. Räcke. Minimizing congestion in general networks. In *Proc. 43rd Symp. on the Foundations of Computer Science*, pages 43–52, 2002.

[17] H. Räcke. *Data management and routing in general networks*. PhD thesis, University of Paderborn, 2003.

[18] P. Raghavan and C. D. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365–374, 1987.

[19] C. Scheideler. Web page: http://www14.in.tum.de/lehre/2005WS/na/index.html.en (unpublished course notes).

[20] A. Srinivasan and C.-P. Teo. A constant factor approximation algorithm for packet routing, and balancing local vs. global criteria. In *Proc. 29th Symp. on the Theory of Computing*, pages 636–643, 1997.

[21] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Proc. 13th Symp. on the Theory of Computing*, pages 263–277, 1981.

Costas Busch received a B.Sc. (1992) and M.Sc. (1995) in Computer Science from the University of Crete, Greece and a PhD in Computer Science from Brown University (2000). He was an Assistant Professor of Computer Science at Rensselaer Polytechnic Institute (2000-2007). He is currently an Assistant Professor of Computer Science at Louisiana State University. His research interests are in the area of distributed algorithms, communication algorithms for wireless and optical networks, constructions of distributed data structures. He has several journal and conference publications in this area of research, and served in the program committees of related conferences.

Malik Magdon-Ismail obtained a B.S. in Physics from Yale University in 1993 and a Masters in Physics (1995) and a PhD in Electrical Engineering with a minor in Physics from the California Institute of Technology in 1998. Since then, he has been a research fellow in the Learning Systems Group at Caltech (1998-2000), and is currently an Associate Professor of Computer Science at Rensselaer Polytechnic Institute (RPI), where he is a member of the Theory group. His research interests include the theory and applications of machine and computational learning (supervised, reinforcement and unsupervised), communication networks and computational finance. He has served on the program committees of several conferences, and is an Associate editor for Neurocomputing. He has numerous publications in refereed journals and conferences, has been a financial consultant, has collaborated with a number of companies, and has several active grants from NSF.

Jing Xi obtained a B.E. in Computer Software from University of Science and Technology of China (1998), a M.S. in Computer Architecture from the Institute of Computing Technology, Chinese Academy of Sciences (2001), and a PhD in Computer Science from Rensselaer Polytechnic Institute (2006). Her research interests are in distributed computing and communication algorithms.