Upper and Lower Bounds for Selection on the Mesh *

Anne Condon
Department of Computer Science
University of Wisconsin
Madison, WI 53706
Ph: 608-262-3158
Fax: 608-262-9777
condon@cs.wisc.edu

Lata Narayanan
Department of Computer Science
Concordia University
Montreal, Canada H3G 1M8
Ph: 514-848-3029
Fax: 514-848-2830
lata@cs.concordia.ca

August 22, 1997

Abstract

A distance-optimal algorithm for selection on the mesh has proved to be elusive, although distance-optimal algorithms for the related problems of routing and sorting have recently been discovered. In this paper, we explain, using the notion of *adaptiveness*, why techniques used in the currently best selection algorithms cannot lead to a distance-optimal algorithm.

For worst-case inputs, we apply new techniques to improve the previous best upper bound of $1.22n$ of Kaklamanis *et al.* [7] to $1.15n$. This improvement is obtained in part by increasing the adaptiveness of previous algorithms.

Keywords: Lower bound, selection, mesh, randomized algorithm, adaptiveness.

*Condon's research supported by NSF grant numbers CCR-9100886 and CCR-9257241 and by awards from Digital Equipment Corporation and the AT&T Foundation. Narayanan's research supported in part by NSERC grant number OGP0155204.

1 Introduction

A distance-optimal algorithm for selection on the mesh has proved to be elusive, although a series of algorithmic refinements has led to the development of distance-optimal algorithms for the related problems of routing and sorting. In this paper, we explain, using the notion of adaptiveness, why the techniques used in the best known algorithms for selection cannot lead to a distance-optimal algorithm. Furthermore, we develop an improved algorithm for median-finding on the mesh, in part by increasing the adaptiveness of previous algorithms for selection.

The selection problem is to find the element of some given rank k from $N = n^2$ totally ordered elements on a $n \times n$ mesh. Initially each processor contains exactly one element. The rank of an element x in a set S is $|\{y \in S \mid y \leq x\}|$. At the end of the algorithm the selected element must be at the center processor of the mesh. Because of the simplicity of its architecture, the mesh model is a well-studied and popular model of parallel computation [12] and is the basis for several parallel machines [1, 14]. We consider a standard model [6, 7, 12, 13, 16] in which a processor is allowed to communicate one packet of information to each of its neighbors during a single time step and to queue a constant number of packets between time steps.

Any selection algorithm requires $n-1$ steps, since this is the distance from the corners to the center of the mesh. This is the only known lower bound for selection on the standard model of the mesh (Kunde's lower bound of $2n - o(n)$ steps [10] applies only to a very restricted model of the mesh). The previously best algorithm for this problem runs in $1.22n$ steps [7, 8, 15]. Thus, the complexity of selection on the mesh is well-known to be $\Theta(n)$; the main open question is whether there exists a distance-optimal, or $n + o(n)$-time, algorithm for selection on the mesh. This question is especially interesting in light of the fact that distance-optimal algorithms for the related problems of sorting and routing have recently been discovered [6, 8, 13]. We provide a partial answer for this question in this paper.

Our main lower bound shows that the techniques used in the best previous selection algorithms cannot yield a distance-optimal algorithm. To explain this, we define a notion of adaptiveness for comparison-based algorithms on the mesh, and show that "weakly-adaptive" algorithms cannot be distance-optimal. Intuitively, median-finding algorithms adapt over time, based on new information learned from comparisons. For example, packets that appear likely to be the median, based on comparisons with a sample of elements, may be routed towards the center early on. However, in all known median-finding algorithms, such adaptive routing decisions are made only once or twice. This is in part because gathering large samples of elements is expensive.

To precisely limit the degree of adaptiveness of an algorithm, we limit the set of comparison results on which a processor's computation can depend. Roughly in our model, at fixed steps called knowledge steps, each processor learns the results of comparisons between every pair of elements that could possibly have reached the processor at that step. Between knowledge steps, processors may not perform new comparisons. However, at a step which is not a knowledge step, a processor may still learn new comparison results in the following way: it learns the

comparison results of its neighbors at the previous step. We say that an algorithm is *weakly-adaptive* if it has $O(1)$ knowledge steps; otherwise we say it is *highly-adaptive*. A *maximally adaptive* algorithm is one where every step is a knowledge step. We define this notion precisely in Section 2.2 and explain why the best previous algorithms for selection on the mesh are weakly adaptive.

In this paper, we show that there can be no distance-optimal weakly adaptive algorithm for selection on the mesh. We also show several other lower bounds for selection for highly adaptive algorithms that are restricted in other ways, such as in the number of packets that can be at one processor at any given time. Roughly, to prove our main result, we exploit the non-adaptiveness of the algorithm to show that there must be many possible candidates for the median at some time t. Since each processor can only store a constant number of packets, some candidate must be of distance much greater than $n - t$ from the center. Furthermore, on a possibly different input, this candidate is the true median, and is also sufficiently far from the center that it cannot reach the center in $(1 + \epsilon)n$ steps for some constant $\epsilon > 0$.

We next describe our upper bound results. We improve the previous best upper bound for randomized (Las Vegas) algorithms, from $1.22n$ to $1.15n$. This upper bound is obtained in part by increasing the number of knowledge steps, and hence the adaptiveness, of the algorithm. Our new algorithm can be described as a "filtering" method: initially all elements are considered to be possible candidates for the median and are routed towards the center; then, over time, unlikely candidates are filtered out by a set of filtering processors to reduce the routing bottleneck close to the center. Processors that are equidistant from the center form a diamond-shaped filter at a given time. Each filtering processor uses sampling techniques to compute, at this time, a restricted range that is likely to contain the median; henceforth, elements routed to that processor which lie outside this range are discarded. Thus, the routes of packets are adapted at the filtering steps.

The success of the filtering method depends on the routing scheme, the locations of the filters and the times that filtering is done. The previous best algorithm of Kaklamanis *et al.* [7] can also be viewed as a filtering algorithm, but filtering is only done once, based on a single sample which is collected at the center. Our scheme uses three filters, and uses a new distributed sampling method to enable the filtering processors to filter elements earlier in the algorithm. A second contribution of our algorithm is in the routing scheme. In our algorithm, packets may be routed out of their quadrants, thus spreading the likely candidates for the median more uniformly over the mesh. This increases the effectiveness of the filtering method. We claim in our concluding section that our techniques push the filtering method to its limits, in that little further improvement to the running time can be obtained by increasing the number of filters.

To summarize, the main results of this paper are the following new upper and lower bounds for the median-finding problem on the standard, 2-dimensional mesh.

- We show that there is no weakly-adaptive, comparison-based, distance-optimal algorithm for selection on the mesh. Our result holds even for randomized algorithms.
- We present a new $1.15n$ step randomized algorithm for selection on the mesh.

The rest of the paper is organized as follows. Our upper and lower bound models are defined and justified in Section 2. Our lower bounds are presented in Section 3, our upper bounds in Section 4. For simplicity, all our results are proved for the median-finding problem, but can be easily generalized for the general selection problem. Conclusions are presented in Section 5.

2 Upper and Lower Bound Models

2.1 Upper Bound Model

Our upper bound model is a standard one for describing routing algorithms on the mesh [6, 7, 13, 16]. The $n \times n$ mesh-connected array of processors (or two-dimensional mesh) contains $N = n^2$ processors arranged in a two-dimensional grid. More precisely, it corresponds to the graph, $G = (V, E)$, with the set of processors $V = \{(x, y) \mid x, y \in \langle n \rangle\}$ and the set of links, or edges $E = \{((x, y), (x, y+1)) \mid x \in \langle n \rangle, y \in \langle n-1 \rangle\} \cup \{((x, y), (x+1, y)) \mid x \in \langle n-1 \rangle, y \in \langle n \rangle\}$, where $\langle n \rangle$ denotes the set $\{1, \ldots, n\}$. In this paper, we will consider the case when n is odd; the case when n is even is treated similarly.

During a single parallel communication step, each processor can send and receive a single *packet* along each of its incident edges, where a packet consists of at most a single input element along with $O(\log N)$ bits of header information used for routing and counting purposes. Between communication steps, processors can store packets in their local queues, which are of bounded size (the *queue size*). Furthermore, a processor can perform a constant number of simple operations (such as copying, addition or comparison) on the elements and the header information of packets. In particular, a processor may *replicate* or create copies of a packet, and send different copies to different locations. The communication and computation of a processor may be a function of its internal memory, and, in the case of a randomized algorithm, of a constant number of coin flips. In measuring the performance of an algorithm, we count the number of communication steps, and assume that the bounded number of local operations that are performed by processors between communication steps are subsumed by the time needed for communication.

2.2 Lower Bound Model

We now explain and justify our lower bound model in detail. Our lower bound model applies to *comparison-based* algorithms for a class of problems where the input is a collection of packets, distributed among the processors of the mesh, and the output is just a reordering of the input. Such problems include sorting and selection. By restricting our model to comparison-based algorithms, we preclude the possibility, for instance, of processors routing packets differently based on whether they are odd or even valued. All previous algorithms for sorting and selection on the mesh are comparison-based algorithms.

To define a lower bound model which limits adaptiveness, we limit a processor's access to comparison results as follows. Fix an input I (if the algorithm is randomized, also fix the coin tosses of every processor). With each time t and processor p we associate a set of comparison results $S(p, I, t)$ as follows. Let $S(p, I, 0) = \emptyset$. In defining $S(p, I, t)$ for $t > 0$, we distinguish between two types of time steps. If t is a *knowledge step*, we let $S(p, I, t)$ be the set of comparison results between all pairs of packets that are *accessible to* p at time t. Here, an element Q is accessible to a processor p at time t if there is a path of length $\leq t$ from Q's initial location to processor p. If t is not a knowledge step, we let $S(p, I, t)$ be the union of $S(p, I, t-1)$ with the sets $S(p', I, t-1)$, where p' ranges over the processors adjacent to processor p in the mesh. Note that this does not imply that p has access to the results of comparisons between all pairs of elements accessible to it at time t. For example, suppose time step t is not a knowledge step, and A and B are packets with adjacent rank. Suppose furthermore that for any processor p', it is not the case that both A and B are accessible to p' at the most recent knowledge step. Then even if A and B are accessible to processor p at time t, $S(p, I, t)$ does not contain the result of the comparison between A and B.

With the definition of $S(p, I, t)$ in hand, we can now specify our lower bound model. As in our upper bound model, the queue size of each processor is bounded. We assume that the knowledge steps of the algorithm are a function of the input size n, say $g_1(n) < g_2(n) < \ldots < g_k(n)$. On a fixed input, at a given time step t, a processor p may do arbitrary internal computation, using its coin flips and the comparison results in $S(p, I, t)$. Based on this computation, a processor may send a packet along every edge incident to it. Finally, a processor may receive a packet along every edge incident to it.

To understand the constraints on how data is allowed to move, consider the following trivial algorithm on this model: Every processor sleeps until time step n, at which time there is a knowledge step. Since, at time n, all elements are accessible to the center processor, say p', it can use $S(p', I, n)$ to calculate the value of the median. However, recall that we require the packet containing the median element to be *routed* to the center processor. Therefore, in this algorithm, since the packet containing the median may be at one of the corners of the mesh, an additional $n - 1$ steps are needed just to route the median to the center.

We say that an algorithm is $k(n)$-*adaptive* if for all n, the algorithm has at most $k(n)$ knowledge steps. An algorithm is *weakly-adaptive* if it is $O(1)$-adaptive; otherwise we say it is

highly-adaptive. An algorithm is *maximally adaptive* if every step of the algorithm is a knowledge step.

The maximally-adaptive model is similar to the lower bound model of Schnorr and Shamir [18], Kunde [10] and Han *et al.* [5]. However, our model allows replication of packets, which is not allowed in their model. On the other hand, their model is more general than our maximally adaptive model in that an algorithm may depend on the values of accessible packets, and not just on the outcomes of comparisons. However, we need the restriction to comparison-based algorithms only in proving lower bounds for weakly-adaptive algorithms, and in fact our lower bounds for maximally-adaptive algorithms also hold for algorithms which are not comparison-based. Thus, our lower bounds for maximally-adaptive algorithms are more general than previous results for such algorithms.

The previous best algorithm for selection of Kaklamanis *et al.* [7] is a weakly adaptive algorithm. Roughly, a small sample of elements is routed close to the center of the mesh and sorted. From this sample, splitters are computed which are broadcast to all processors within a certain range of the center. In our lower bound model, the splitters can be obtained "for free" in a single knowledge step. The routes of packets change only once during the course of the algorithm, based on the values of the splitters, and once at the end, when the true median is computed. Hence, the Kaklamanis *et al.* algorithm is actually a 2-adaptive algorithm. We will show that our algorithm of Section 4 is 4-adaptive.

In Section 3, we prove that there is no distance-optimal, weakly-adaptive algorithm for selection. Thus, if a distance-optimal algorithm for selection exists, the route of a packet must adapt, a number of times that grows with the size of the mesh, on the values of accessible elements at those times.

2.3 Notation

In discussing both upper and lower bounds, we use the following notation. The *center* processor is $(\lceil \frac{n}{2} \rceil, \lceil \frac{n}{2} \rceil)$. The *distance* between two processors (x_1, y_1) and (x_2, y_2) is $|x_1 - x_2| + |y_1 - y_2|$. For $r \leq n/2$, the *r-diamond* is the set of $2\lfloor r \rfloor^2 + 2\lfloor r \rfloor + 1$ processors that are at most r distant from the center processor. We consider the mesh to be composed of four *quadrants*, NW, NE, SW and SE. The NW quadrant contains exactly the processors in the set $\{(i,j) \mid 1 \leq i, j \leq \lceil n/2 \rceil\}$; the others are defined similarly. Since n is odd, the quadrants overlap at the borders and the center processor is in all quadrants. The restriction of the r-diamond to a given quadrant is called the *r-triangle* in that quadrant, and contains $(1/2)\lfloor r \rfloor(\lfloor r \rfloor + 1)$ processors. The *r-corner* of a quadrant is the set of processors in that quadrant of distance at least $n - r$ from the center. For example, in the NW quadrant, this is the set of processors $\{(i,j) \mid 1 \leq i + j \leq r\}$.

3 Lower Bounds

In this section, we present our main lower bound result, that there is no distance-optimal, weakly-adaptive algorithm for finding the median. Our lower bound holds even for randomized algorithms.

We prove our lower bound for a slightly more general model than that presented in the Section 2.2. Let $S(I,t)$ be the union over all p of $S(p,I,t)$. In this new model, at every step, the computation of a processor may depend on $S(I,t)$ and not just $S(p,I,t)$. Intuitively, in this model, at each knowledge step a processor learns the comparison results accessible to *every* processor in the mesh. Between knowledge steps, a processor learns of no further comparisons. Clearly this is more general than before, since $S(p,I,t) \subseteq S(I,t)$ for all t. Therefore, any lower bound on this model also applies to our model described in Section 2.

Our lower bound proof is somewhat simpler to present for this more general model. We need the following definition.

Definition 3.1 *Fix a deterministic algorithm and an input I. Element Q is* live *at time t if there is some input I' such that*

(i) Q is in the same initial position in I and I',

(ii) Q is the median of I' and

(iii) $S(I,t) = S(I',t)$.

In Lemma 3.1, we show that for a fixed, weakly-adaptive, deterministic algorithm and input, if some element Q is live at time $n-t$ and is far from the center at this time, then the algorithm cannot be distance optimal. In Theorem 3.1, we construct an input I which has many live elements at time $n-t$. We use the bound on the queue size to conclude that one of these must be far from the center at time $n-t$, and apply Lemma 3.1 to obtain the main result. In Theorem 3.2, we generalize this idea to randomized algorithms.

Lemma 3.1 *Fix a deterministic algorithm and an input I. Suppose that there is an $\epsilon > 0$ such that all packets containing some live element Q are of distance $\geq t + \epsilon n$ from the center, at some time $n-t$. Then the algorithm is not distance optimal.*

Proof: Since Q is live at time $n-t$, there is some input I' such that (i) Q is in the same initial position in I and I', (ii) Q is the median of I' and (iii) $S(I,n-t) = S(I',n-t)$. From (iii), it follows that in fact for all $t' \leq n-t$, $S(I,t') = S(I',t')$. Thus on inputs I and I', processors perform exactly the same operations at every step up to time $n-t$. Hence on inputs I and I', all packets containing Q are in the same positions at time $n-t$. Thus, on input I', the time to get Q to the center is at least $(n-t)+(t+\epsilon n) = (1+\epsilon)n$. Since Q is the median of input I', the algorithm is not distance-optimal. $\Box$

We can now prove that there is no distance-optimal, weakly adaptive, deterministic algorithm for finding the median.

Theorem 3.1 *There is no distance-optimal, deterministic, weakly-adaptive algorithm for finding the median, for any constant queue size.*

Proof: Fix a weakly-adaptive algorithm which has k knowledge steps between time 0 and n, where we assume without loss of generality that there are also knowledge steps at time 0 and n. Note that we do not care about knowledge steps after time n. Let these $k+2$ knowledge steps be at times $n - f_0(n) < \ldots < n - f_{k+1}(n)$. Hence $f_0(n) = n$ and $f_{k+1}(n) = 0$. Let q denote the (constant) queue size and let $n > (2q)^k$. Then there must be an index i where $1 \leq i \leq k+1$ such that $f_{i-1}(n) > \frac{2n}{(2q)^k}$ and $f_i(n) < \frac{f_{i-1}(n)}{2q}$.

Let S be the set of elements in the $(f_{i-1}(n)-1)$-corner of the NW quadrant. Let T be the set of elements in the $(f_{i-1}(n)-1)$-corner of the SE quadrant. Note that $\frac{(f_{i-1}(n))^2}{2} - O(n) \leq |S|$. Our goal is to construct an input I such that every element Q in S is live at time $n - f_i(n) - 1$. The input I' that will "witness" the fact that Q is live, will differ from I only in the set of input elements in T.

The input I is as follows, where all elements are integers. The elements in S are those elements $\leq$ median whose rank is within $|S|-1$ of the rank of the median, and the elements in T are those elements greater than the median whose rank is within $|T|$ of the rank of the median. Also, the difference between each pair of elements in S is at least $|T|$.

We next show that at time $n - f_{i-1}(n)$, all elements of S are live. Since $S(I, n - f_{i-1}(n)) = S(I, n - f_i(n) - 1)$, it follows immediately that all elements of S are live at time $n - f_i(n) - 1$.

Let Q be any element of S. Suppose that the rank of Q is m less than the rank of the median, $0 \leq m \leq |S| - 1$. To show Q is live, we define I' to be the same as I, except that the m smallest elements in T are now chosen to be the elements just smaller than Q. Clearly, Q is in the same initial position in I and I', and Q is the median of I'. Hence properties (i) and (ii) of Definition 3.1 are satisfied.

It remains to show that $S(I, n - f_{i-1}(n)) = S(I', n - f_{i-1}(n))$. Suppose that (P, R) is any pair of elements in I (originating at processors p and r respectively) and (P', R') is the corresponding pair of elements in I'. First suppose that both elements of one pair are in S (or both are in T). Then, by construction, the relative order of P and R is the same as the relative order of P' and R'. Thus, if $S(I, n - f_{i-1}(n))$ contains the result of the comparison between P and R, then $S(I', n - f_{i-1}(n))$ contains the same result between P' and R'. It remains to consider the case that one element of the pair is in S and the other is in T. We claim that in this case, in both the partial orders consistent with $S(I, n - f_{i-1}(n))$ and $S(I', n - f_{i-1}(n))$, the pairs (P, R) and (P', R'), respectively, are incomparable. From this it follows that $S(I, n - f_{i-1}(n)) = S(I', n - f_{i-1}(n))$.

We will show that in the partial orders consistent with $S(I, n - f_{i-1}(n))$ and $S(I', n - f_{i-1}(n))$, every element in S is incomparable with every element of T. This follows from two

facts. The first is that on any input, at time $n - f_{i-1}(n)$ it is not possible that both an element of S and an element of T are accessible to the same processor. This is because the initial distance between any element in S and any element in T is at least $2n - 2f_{i-1}(n) + 2$; also any two elements can get at most a distance of 2 closer in one step. Thus, at any time $\leq n - f_{i-1}(n)$, an element in S and an element in T are of distance greater than $(2n - 2f_{i-1}(n) + 2) - (2n - 2f_{i-1}(n)) > 0$ from each other, and therefore cannot both reach the same processor. The second fact we use is that in both I and I', the elements in $S \cup T$ are exactly those elements whose ranks are in the range between $|S| - 1$ below the median and $|T|$ above the median. Hence, if R is an element not in $S \cup T$, the elements in $S \cup T$ are all either $> R$ or $< R$, and so the relative order of an element of S and an element of T cannot be deduced from their order with respect to other elements. This concludes the proof that $S(I, n - f_{i-1}(n)) = S(I', n - f_{i-1}(n))$, and hence that Q is live at time $n - f_{i-1}(n)$.

The number of packets that can be accommodated in the queues of processors in the $\frac{f_{i-1}(n)}{q}$-diamond is at most $2\frac{(f_{i-1}(n))^2}{q}$. Thus, since $|S| > \frac{(f_{i-1}(n))^2}{2} - O(n)$, for $q > 4$ and n large enough, there is an element $Q \in S$ such that all packets containing Q are outside the $\frac{f_{i-1}(n)}{q}$-diamond at time $n - f_i(n) - 1 > n - \frac{f_{i-1}(n)}{2q} - 1$. Then from Lemma 3.1, it follows that the algorithm is not distance-optimal. □

Our lower bound depends crucially on the fact that the queue size is constant. If the queues could grow to size $O(\log n)$, for instance, then all live packets could conceivably be within distance $n/\log n$ of the center processor. Thus it may be possible for the median to reach the center processor in $n + o(n)$ steps. However, no algorithm for selection is known even on the fully adaptive model with unlimited queue size. It would be interesting to see if a lower bound can be proved for weakly adaptive algorithms with queues of non-constant size.

We now extend our result to randomized algorithms. To prove a lower bound on the running time of a randomized algorithm, we need to show that there exists some input on which the expected running time is at least $(1 + \epsilon)n$, for some $\epsilon > 0$. Thus, our technique of fixing an input and later modifying it (as in Theorem 3.1) is not valid for randomized algorithms. This is because it is not valid to modify the input *after* random bits of the algorithm are already chosen. Instead, we use a probabilistic argument. We show that there is a set of inputs, such that if one is chosen uniformly at random, then the expected running time (averaged now over both the uniform input distribution and the internal randomness of the algorithm) is $\geq (1+\epsilon)n$. We can therefore conclude that there exists an input on which the expected running time is at least $(1 + \epsilon)n$. We note that this lower bound holds even if all processors have full knowledge of all coin tosses of all other processors. However, recall that the knowledge steps are fixed and cannot be determined by the results of the coin tosses.

Theorem 3.2 *There is no distance-optimal, randomized, weakly-adaptive algorithm for finding the median, for any constant queue size.*

Proof: Let the $k + 2$ knowledge steps, q and i be defined as in Theorem 3.1. Consider the following set of inputs. Let S and T be the set of elements in the $(f_{i-1}(n) - 1)$-corner of the

NW quadrant and the $(f_{i-1}(n)-1)$-corner of the SE quadrant, respectively. The elements in the mesh outside of the set $S \cup T$ are chosen so that exactly half are greater than the elements of $S \cup T$ and exactly half are less than the elements of $S \cup T$. The elements of S are chosen to have values such that there is a difference of at least $|T|$ between any two members of S. The set T is chosen randomly in such a way that each of the elements of S is equally likely to be the median, and so that the relative order of the elements of T is the same for all random choices.

Fix any run of the algorithm, that is, fix the coin tosses of all processors. Consider any two inputs I and I' defined by different random choices of T. With respect to the fixed run, $S(I,t) = S(I',t)$ for all $t \leq n - f_i(n) - 1$. This can be proved in a manner similar to that used in Theorem 3.1 to show that $S(I, n - f_{i-1}(n)) = S(I', n - f_{i-1}(n))$. Hence, as in Lemma 3.1, with respect to the fixed run, the positions of all elements of S are the same on inputs I and I'. Then for $q > 16/3$, it is straightforward to see that all packets containing at least $3/4$ of the elements of S are outside the $\frac{f_{i-1}(n)}{q}$-diamond at time $n - f_i(n) - 1$.

Thus, the probability that the median is inside the $\frac{f_{i-1}(n)}{q}$-diamond at time $n - f_i(n) - 1$ is at most $1/4$ (where the probability is taken over the random choices of T since the run is fixed). If the median is inside the $\frac{f_{i-1}(n)}{q}$-diamond, the time to get it to the center is at least $n - f_i(n) - 1 > n - \frac{f_{i-1}(n)}{2q} - 1$. If the median is not inside the $\frac{f_{i-1}(n)}{q}$-diamond, the time to get it to the center is at least $n + \frac{f_{i-1}(n)}{2q} - 1$. Hence, the total expected time of the algorithm, with respect to the fixed run, is at least $(n - \frac{f_{i-1}(n)}{2q} - 1)/4 + 3(n + \frac{f_{i-1}(n)}{2q} - 1)/4 \geq n + \frac{f_{i-1}(n)}{4q} - 1$. Since this is true for any fixed run, the total expected time, over all runs of the algorithm, also satisfies the same lower bound. □

Finally, we note that using similar techniques, we can obtain lower bounds for maximally-adaptive algorithms, if we restrict the queue-size to 1. In Theorem 3.3, we state this result, along with several other lower bounds that we can obtain by considering the following additional restrictions to the model. One is simply that packets may not be replicated (see the upper bound model, Section 2.1 for a definition). The other is that packets must stay in the quadrant in which they are located initially; we say that packets are *quadrant constrained*. These restrictions are interesting because they again apply to all previously known algorithms for selection on the mesh. Our new algorithm in Section 4 is the first algorithm for selection which is not quadrant constrained.

Parts (a) and (b) of Theorem 3.3 extend in different ways a previous result of Kunde [10], who proved a lower bound of $2n - o(n)$ steps for maximally-adaptive algorithms with queue size of 1, and the additional restriction that packets are not replicated.

Theorem 3.3 *There is no distance-optimal, maximally adaptive deterministic algorithm for finding the median:*

(a) if the queue size is 1.

(b) if packets are not replicated and the queue size is 2.

(c) if packets are quadrant constrained and the queue size is 3.

(d) if packets are not replicated, are quadrant constrained and the queue size is 4.

The proof of Theorem 3.3 can be found in Section A.4.

4 A Randomized Algorithm for Selection

The high-level description of our algorithm is similar to previous algorithms of [7, 9]. However, we use several new techniques, including greater adaptiveness, in order to obtain gains in the running time. We start this section with an outline of the general scheme used in previous algorithms, and then describe our new techniques in detail. For clarity of exposition, we assume in this section that processors have queues of unbounded size. In Section 4.5, we provide a brief sketch of how to extend our techniques to achieve constant size queues.

Briefly, the algorithm is as follows. All packets are routed inside a diamond, which we call the *gathering* diamond. At the same time, using sampling techniques, *bracketing elements* are computed by the center processor, and are broadcast to all processors in the gathering diamond. With high probability, the number of elements that lie between the bracketing elements is $O(N^{1-\delta})$ for some $\delta > 0$, and the median lies between the bracketing elements. All packets that lie between the bracketing elements are routed to a small block near the center. Since there are few of them, standard sorting algorithms can be used to sort them in $o(n)$ time. Once they are sorted and the exact ranks of the bracketing elements are computed, the median can be identified and sent to the center. We summarize these steps in Figure 1.

1. All packets are routed inside gathering diamond.
2. The center processor selects bracketing elements $b < b'$ and broadcasts these to all processors in the gathering diamond. With high probability, there are $O(N^{1-\delta})$ input elements in the range $[b, b']$ and the median lies in this range.
3. All packets Q with $b \leq Q \leq b'$ are routed to a central block of side $o(n)$, and are sorted.
4. The ranks of b and b' are computed and are broadcast to all processors in the central block.
5. The element in the central block of rank $\lfloor N/2 \rfloor - rank(b)$ is the median. The processor with this element sends it to the center.

Figure 1: High-level Algorithm Description (Similar to Previous Algorithms).

It turns out that the main bottleneck to the running time of this algorithm is Step 1, that of routing all packets inside the gathering diamond. Note that the number of packets that can be routed inside the gathering diamond in one step is at most $8g$ where g (the gathering radius) is the distance from the center to the boundary of the gathering diamond. Thus at least $n^2/8g$ steps are required to complete this step. To overcome this bottleneck, our algorithm introduces

several new techniques, which we now describe. The result is a more sophisticated Step 1, which is summarized in Figure 2 below.

First, our new algorithm filters out packets that are unlikely to be the median and routes only the remaining packets inside the gathering diamond. Since there are fewer packets to route, the value of g is reduced. Elimination or filtering of packets is done as follows. First, in Step 1a, all packets are routed inside the $n/8$-diamond (larger than the gathering diamond). The processors on the boundary of the $n/8$-diamond are called the *filtering processors*. When Step 1a has been completed, a pair of *splitters* is selected by each filtering processor, such that the median is very likely to lie between the splitters. All packets that have been routed to filtering processor P and do not lie between P's splitters are thus eliminated.

1. A set of packets, which contains the median with high probability, is routed inside the gathering diamond, as follows.
   (a) All packets are routed inside the $n/8$-diamond. For each packet Q, either Q is routed inside the gathering diamond, or it is routed to a filtering processor P (i.e. on the $n/8$-diamond), chosen randomly and uniformly from $3n/8$ filtering processors, according to the schedule described in Lemma 4.1.
   (b) Each filtering processor P selects a pair of splitters, $s_1(P) < s_1'(P)$, at time $n - 0.03n + o(n)$. The elements routed to P in Step 1a, which also lie in the range $[s_1(P), s_1'(P)]$, are called P's tentatively live elements.
   (c) At time $n - 0.03n + o(n)$, each filtering processor selects, from its tentatively live packets, a random subset of size $0.03n$. The selected packets are routed towards the gathering diamond by time $n + o(n)$.
   (d) Each filtering processor P selects a pair of splitters, $s_2(P) < s_2'(P)$, at time $n + o(n)$. The elements routed to P in Step 1a, which also lie in the range $[s_1(P), s_1'(P)] \cap [s_2(P), s_2'(P)]$, are called P's live elements.
   (e) All live elements remaining at the filtering processors are routed inside the gathering diamond.

Figure 2: New Algorithm, Step 1.

Only the remaining packets, called the *live* packets, need to be forwarded inside the gathering diamond. Identification of the live elements and their routing are done in Steps 1d and 1e. Steps 1d and 1e are sped up by intermediate phases Steps 1b and 1c, which are overlapped with Step 1a and are completed before Step 1d begins. In Step 1b, a superset of the live packets, called the *tentatively live* packets, is identified. In Step 1c, approximately half of these are sent towards the gathering diamond. As a result, approximately half of the live packets at each filtering processor are already on their way to the gathering diamond by Step 1d, and so only half need to be sent during Step 1e. In Step 1b, the tentatively live packets are identified by computing yet another pair of splitters.

We note that our use of splitters to eliminate packets in Steps 1c and 1e are natural approximations of knowledge steps. Information is collected at selected processors to enable them to

adapt the routes of the packets that reside at them. Our overall algorithm is therefore consistent with the weakly-adaptive model with 4 knowledge steps, one at time $n - 0.03n$ in Step 1c, the second at time n in Step 1e, and two more in Steps 3 and 5. In contrast, previous algorithms used only two knowledge steps, in Steps 3 and 5. Thus our algorithm is more adaptive than previous algorithms.

One further idea is used to minimize the number of live packets at every processor. In Step 1 of previous algorithms, each packet Q is routed inside the gathering diamond, via a processor P on the boundary of the gathering diamond, where P is in the same quadrant as Q's initial location. In contrast, for a given packet, Step 1a of our new algorithm randomly selects a filtering processor from three quadrants, as the possible destination of the packet. The resulting routing scheme is quite complex, but we will see that as a result, packets that are likely to be the median are distributed over a larger number of processors, and hence, each processor has fewer live packets. We also note that in addition to not being quadrant-constrained, our algorithm replicates some packets. For example, elements that are chosen to be sample elements in various steps of the algorithm have multiple copies in the mesh.

These techniques combined yield a significant improvement to Step 1. The remaining steps are just as before, although the implementation of Step 4 is slightly different than in previous algorithms.

This algorithm successfully halts in time $1.15n$ with high probability; by this we mean with probability $1 - O(N^{-\beta})$ for $\beta = 3$ (a larger value of β is possible simply by changing some parameters in the algorithm, for example, the size of the sample). If the algorithm fails to route the median to the center processor in the required time, then the center processor broadcasts a message to restart, and a naive $O(n)$ algorithm can be executed (such as an algorithm based on sorting [18]). Since the probability of failure is $O(N^{-3})$, the expected running time of the whole algorithm is still $1.15n$.

It is now possible to give an overview of the running time of the algorithm. First, consider Step 2. The bracketing elements are selected by the center processor using sampling techniques (see [17, 7]); to get a sample of the entire input to the center requires n steps since this is the maximum distance of an element to the center. Broadcasting the bracketing elements similarly requires time equal to g, the distance from the center to the processors on the boundary of the gathering diamond. Thus, Step 2 requires time $n + g$. Similarly, Step 3 requires time equal to $g - o(n)$, since elements have to be routed in from the processors on the boundary of the gathering diamond to the central block. It turns out that Step 4 can be overlapped with Step 3; and Step 5 requires only $o(n)$ time, since it involves computation in a small central block. Thus, the total time of Steps 2 through 5 must be at least $n + 2g$. In fact, they can be completed in time $n + 2g + o(n)$ as shown in [7].

It remains to consider Step 1; we summarize the analysis here. Since at most n packets can be routed inside the $n/8$-diamond in one step, n steps are required to complete Step 1a, and we show that in fact this step can be completed in time $n+o(n)$ with high probability. Furthermore,

Steps 1b and 1c can be overlapped with Step 1a, and Step 1d takes only an additional $o(n)$ steps. Also, with high probability, each filtering processor P has at most $0.06n+o(n)$ tentatively live elements and the median lies in the range $[s_1(P), s_1'(P)]$. Thus, at the end of Step 1c, a fraction $0.03n/0.06n = 1/2$ of the tentatively live packets at each filtering processor are already routed towards the gathering diamond.

Consider the time to complete Step 1e. We will show that, with high probability, each filtering processor P has at most $n/24 + o(n)$ live elements and the median lies in the range $[s_2(P), s_2'(P)]$. Of these live packets, a fraction $1/2$ are already routed towards the gathering diamond; thus in Step 1e, $(1-1/2)(n/24)+o(n) = n/48+o(n)$ live packets are pipelined from P towards the gathering diamond. Each of these live packets must travel a distance equal to $n/8 - g$. Hence, the total time to complete Step 1e is $n/48 + n/8 - g + o(n)$.

The total time to complete Step 1 is therefore $n(1+1/8+n/48)+o(n)-g = 55n/48-g+o(n)$ steps. Steps 1 and 2 are overlapped; hence, to minimize the total running time, their running times are balanced. Thus, $55n/48 - g = n + g$. Solving this, $g = 7n/96$. The total running time of the algorithm is then $n + 2g + o(n) < 1.15n$. Thus,

Theorem 4.1 *There is a randomized algorithm that, with high probability (at least $1-O(N^{-3})$), selects the element of rank k out of $N = n^2$ elements on a $n \times n$ mesh in $1.15n$ steps using constant-size queues.*

In the following sections, we describe Steps 1 and 4 of our algorithm in detail. We omit discussion of the remaining steps, since these are almost identical to previous work [7] and a complete description can be found there. Throughout, by "with high probability" we mean with probability $1 - O(N^{-3})$.

4.1 Step 1a: Initial Routing

Lemma 4.1 *Routing all the packets into the diamond of radius $n/8$ can be done in $n + o(n)$ steps using constant size queues with high probability in such a way that:*

(i) Half of the packets are routed inside the gathering diamond.

(ii) Each of the remaining packets is routed to a destination chosen uniformly at random from a set of $3n/8$ of the filtering processors (the processors on the boundary of the $n/8$ diamond).

(iii) Each of the filtering processors is a potential destination for an equal number of packets.

Proof: Note that part (ii) implies that packets must be routed outside their initial quadrant. Essentially the mesh is divided up into several regions, and packets are assigned destinations based on which region they originate in; the routing is done greedily. The proof of the running time is technically complicated and can be found in Section A.1. □

4.2 Steps 1b and 1d: Selecting the Splitters

Selection of the splitter elements is done using the following set sampling method of [17], which was also used in [7]. Each element in the set tosses a coin, which is heads with probability $\alpha N^{5\delta-1} \ln N$, for some constants α and δ ($\alpha = 18$ and $\delta = 1/6$ is sufficient). The set of elements which toss heads is the sample. Every $(\alpha N^{4\delta} \ln N)$-th element of the sample is called a divider. The following lemma is proved in [17].

Lemma 4.2 *For sufficiently small constants δ, given a set of $m = \Theta(N), m \leq N$ elements, the above scheme produces $mN^{\delta-1}$ dividers which divide the elements into buckets of size $N^{1-\delta}(1 \pm N^{-2\delta})$. Each element will be in a bucket whose rank is off by at most one relative to the rank of the corresponding bucket given perfect dividing information. The probability that the algorithm fails is smaller than $N^{-\alpha/5}$.*

The bracketing elements are then selected as follows. The center processor samples the entire input. The resulting sample is called the global sample. Let j be such that

$$(j-1)N^{1-\delta} < \lfloor N/2 \rfloor < jN^{1-\delta}$$

in the set of dividers.

A processor P on the $n/8$-diamond computes splitters as follows. P samples two sets, S_1 and S_2 of elements, namely those accessible to P at times $n - 0.03n$ and n. Let D_1 and D_2 be the resulting sets of dividers. Let $a_1 = n/8 + 0.03n$ and $a_2 = n/8$. Let k_i and $l_i, 1 \leq i \leq 2$ be such that

$$(k_i - 1)N^{1-\delta} < \lfloor |S_i|/2 \rfloor - \lceil a_i^2/2 \rceil - 3N^{1-\delta} < k_i N^{1-\delta}$$

and

$$(l_i - 1)N^{1-\delta} < \lfloor |S_i|/2 \rfloor + \lceil a_i^2/2 \rceil + 3N^{1-\delta} < l_i N^{1-\delta}.$$

Then, $s_i(P), s_i'(P)$ are the elements of ranks $k_i - 2$ and $l_i + 1$, respectively from set D_i.

Selecting splitters at every filtering processor in this way is infeasible for constant queue size, because there are too many packets to be stored at each processor. Instead, splitters are selected only by a set of $\lfloor n^{1-11\delta/2} \rfloor$ evenly spaced filtering processors (where δ is a constant $\leq 1/6$). For each such processor, first sample elements are chosen from among the elements present in a square block of side $n^{11\delta/2}$ centered at it. Then splitters are computed from these sample elements as described above. Each processor which does not select a splitter then receives the splitters of the closest processor which does select a splitter.

We now show that this method of selecting splitters can be implemented efficiently on the mesh. We then prove that the splitters satisfy the properties described in Steps 1b and 1d.

Lemma 4.3 *With high probability, for all filtering processors P, the splitters $s_1(P), s_1'(P)$ can be selected by time $n-0.03n+o(n)$ and the splitters $s_2(P), s_2'(P)$ can be selected by time $n+o(n)$, without increasing the size of the queues by more than a constant additive term.*

Proof: Define P's *sample block* to be the square block of side $\lfloor n^{11\delta/2} \rfloor$ centered at P. Each sample element in P's sample chooses a destination processor uniformly at random from among the processors in P's sample block, and is routed greedily to it. The analysis of the routing is straightforward, and can be found in Section A.2. □

Lemma 4.4 *With high probability, for all filtering processors P,*

(i) at most $(n/8+0.03n)^2+O(N^{1-\delta})$ input elements accessible to P lie between the splitters $s_1(P), s'_1(P)$,

(ii) at most $(n/8)^2 + O(N^{1-\delta})$ input elements accessible to P lie between the splitters $s_2(P), s'_2(P)$, and

(iii) the bracketing elements and the median lie between both pairs of splitters.

Proof: Consider the splitters $s_1(P), s'_1(P)$ of some filtering processor P. (The analysis of splitters $s_2(P), s'_2(P)$ is similar). From Lemma 4.2, with probability $1-O(1/N^{\alpha/5})$, the number of elements in the set of elements accessible to P at time n, that also lie between the splitters is at most $(n/8+0.03n)^2+O(N^{1-\delta})$.

We next show that the bracketing elements lie between the splitters with probability $1-O(1/N^{\alpha/5})$. Let $A(P)$ be the set of elements accessible to P at time $n-0.03n$ and M be the median of the elements of $A(P)$. Let $a_1 = n/8+0.03n$. Then, $|A(P)| \geq n^2 - \lceil a_1^2 \rceil$; to see this, note that Figure 3 illustrates the set of inaccessible elements of a filtering processor P on the border of a quadrant. The size of the set of inaccessible elements is maximized for such a processor.

Let $rank(E)$ for any input element E be the rank of E in the entire input set I. Then it is easy to see that with high probability, $rank(b') \leq \lfloor N/2 \rfloor + 3N^{1-\delta}$ (for details see [7]). We claim that $\lfloor N/2 \rfloor \leq rank(M) + \lceil a_1^2/2 \rceil$. To see this, note that the $rank(M)$ is minimized if all elements not in $A(P)$ are greater than all the elements in $A(P)$. In this case, $rank(M) = \lfloor |A(P)|/2 \rfloor$. Also, the median of the input I is the element of I of rank

$$\lfloor n^2/2 \rfloor = \lfloor |A(P)|/2 + (n^2 - |A(P)|)/2 \rfloor \leq rank(M) + \lceil a_1^2/2 \rceil.$$

Let Q be the element in $A(P)$ whose rank in $A(P)$ is $rank(M) + \lceil a_1^2/2 \rceil + 3N^{1-\delta}$. Then, Q is an upper bound on the bracketing element b'. This is because the rank of Q in I is at least $rank(M) + \lceil a_1^2/2 \rceil + 3N^{1-\delta}$, and

$$rank(b') \leq \lfloor n^2/2 \rfloor + 3N^{1-\delta} \leq rank(M) + \lceil a_1^2/2 \rceil + 3N^{1-\delta} \leq rank(Q).$$

Using the notation introduced to define the splitters, we note that if, in the sample S_1, the dividers in D_1 were perfect, then element Q would lie between dividers $l_1 - 1$ and l_1. By Lemma 4.2, with probability at least $1 - N^{-\alpha/5}$, element Q is bounded by the splitter of rank $l_1 + 1$ in

D_1. But this is exactly $s'_1(P)$. Hence, we have shown that with probability at least $1 - N^{-\alpha/5}$, the median is bounded above by $s'_1(P)$.

A similar argument shows that $s_1(P)$ is a lower bound for the bracketing element b with probability at least $1 - N^{-\alpha/5}$. Also, since with probability at least $1 - N^{-\alpha/5}$, the median lies between the bracketing elements, it also lies in between all pairs of splitters, Thus, for a fixed processor P, the lemma is true with probability $1 - O(1/N^{\alpha/5})$. Since there are at most $N^{1/2-11\delta/4}$ blocks, the probability that the algorithm succeeds on all blocks is $1-(N^{1/2-11\delta/4})(O(1/N^{\alpha/5}))$. This is $1 - O(N^{-3})$ for $\alpha \geq 18$. □

Finally, we prove the bounds on the number of tentatively live and live elements of a filtering processor P, as claimed in Steps 1b and 1c.

Lemma 4.5 *With high probability, all filtering processors P have at most $0.06n + o(n)$ tentatively live elements and at most $n/24 + o(n)$ live elements.*

Proof: Consider the number of live elements of a processor P; the argument for the tentatively live elements is similar. By Lemma 4.4, with probability at least $1 - N^{-\alpha/5}$, at most $(n/8)^2 + o(n^2)$ elements accessible to P lie in the range $[s_2(P), s'_2(P)]$. In the worst case, all of these elements are in the set of elements that can be routed to P. In this case, each of these elements chooses P with probability $8/3n$, by Lemma 4.1. The expected number of these packets that are routed to processor P is therefore at most $((n/8)^2+o(n^2))(8/3n) = n/24+o(n)$. Using Fact A.1, it follows that with probability at least $1 - e^{-(\log n)^2}$, P has at most $n/24+o(n)$ live elements, given that the number of input elements in the range $[s_2(P), s'_2(P)]$ is at most $(n/8)^2 + o(n^2)$.

Hence, with probability at least $1-N^{-\alpha/5}-e^{-(\log n)^2}$, P has at most $n/24+o(n)$ live elements. Since there are $O(n)$ filtering processors, the probability that all have at most $n/24 + o(n)$ live elements is at least $1 - N^{1/2}(N^{-\alpha/5} + e^{-(\log n)^2}) = 1 - O(N^{-3})$ for $\alpha \geq 18$. □

4.3 Steps 1c and 1e: Routing

Lemma 4.6 *With high probability, Step 1c can be completed by time $n + o(n)$ and Step 1e can be completed in $7n/48 - g + o(n)$ additional steps.*

Proof: Consider any filtering processor P. All of the packets routed from P in these steps follow a no-turn path to the gathering diamond, that is, they use either only row edges or only column edges. There is at least one such path, since $g \geq n/16$. Thus, P sends at most one packet per step, and there are no routing conflicts.

Choose $B = 0.06n + o(n)$ so that, with high probability, from Lemma 4.5, B bounds the number of tentatively live packets at every filtering processor. Consider the set of tentatively live packets of some processor P. From this set, a random subset of size $0.03n$ is selected and the packets in this subset, called the *selected packets*, are routed towards the gathering diamond

starting at time $n - 0.03n$. To determine which are the selected packets, all the tentatively live packets are ordered as follows. Suppose that $B - k$ are already at P. The tentatively live packets already at P are ordered arbitrarily from 1 to $B - k$. The remaining tentatively live packets are ordered $B - k + 1, \ldots$ according to when they arrive at P; if two arrive at the same time, order the one that arrives along a row edge first. A subset of size $0.03n$ is chosen from the range $1, \ldots, B$, uniformly and randomly from all such subsets. A packet is selected if and only if its number in the ordering is selected.

We claim that with high probability, all processors P can route all the selected packets by time $n + o(n)$. To prove this, fix a processor P and divide the routing into two phases as follows. The first phase lasts as long as there are selected packets queued at P. Clearly P can send these in, one step at a time. The second phase begins at the first step in which there are no selected packets queued at P. Suppose that this occurs at time $n - t$. Then $2t + o(n)$ packets are still due to arrive at P. Of these, at most t are in the randomly selected set, since there are $0.03n$ packets in this set, and $0.03n - t$ have already been routed in the first phase.

Consider the number of selected packets that arrive in each remaining interval of length $\sqrt{t}$. Using Fact A.1, there are at most $\sqrt{t} + o(\sqrt{n})$ of these in each interval with probability at least $1 - e^{-\log^2 n}$. Of these, $\sqrt{t}$ will be routed from P by the end of the next time interval. Hence, with probability at least $1 - \sqrt{t}e^{-\log^2 n} = 1 - O(\sqrt{n}e^{-\log^2 n})$, at most $\sqrt{t}(1 + o(\sqrt{n}))$ time is needed to complete the routing after time n, in which case all the randomly selected packets are routed from P by time $n + o(n)$.

We justify our choice of the time $n - 0.03n$ to collect the first set of splitters. Suppose the first set of splitters was calculated at time $n - cn$ for some constant c, where $0 < c < 1$. The number of tentatively live elements at a given processor P can be easily seen to be $(n/8 + cn)^2(8/3n)$ as in Lemma 4.5. Let $A = (n/8 + cn)^2(8/3n)$. In the worst case, all the tentatively live elements may arrive at the filtering processor two at each step in the last $\lceil A/2 \rceil$ steps preceding step n. In this case, if $cn > A/2$, there are no tentatively live elements at the processor until step $n - A/2$ and at most $A/2$ packets may be routed in before step n. Thus, the fraction of tentatively live elements that may be routed towards the gathering diamond before step n can be at most $1/2$ for any value of c such that $cn > A/2$. On the other hand, if $cn < A/2$, then we can route at most cn tentatively live elements before step n and the fraction of elements that can be routed in is $cn/A < 1/2$. To maximize the fraction of the tentatively live elements that can be routed in before time step n, we equate $cn = A$, which yields the value $c = 0.03$.

Thus, by time $n + o(n)$, with probability $1 - O(\sqrt{n}e^{-\log^2 n})$, a fraction $(0.03n)/(0.06n) + o(1) = 1/2 + o(1)$ of the tentatively live packets are already routed from P to the gathering diamond. Moreover, the set of routed packets was chosen randomly from the set of tentatively live packets, of which the set of live packets is a subset. Hence, with probability $1 - O(\sqrt{n}e^{-\log^2 n})$, a fraction $1/2 - o(1)$ of the live packets are already routed towards the gathering diamond by time $n + o(n)$. In this case, $(1 - 1/2)(n/24) + o(n) = n/48 + o(n)$ packets remain to be routed from P. Hence, processor P completes Step 1e in $n/48 + n/8 - g + o(n)$ steps, with probability $1 - O(\sqrt{n}e^{-\log^2 n})$.

Since there are $O(n)$ filtering processors P, all complete Step 1e in

$$7n/48 - g + o(n)$$

steps, with probability $1 - O(n\sqrt{n}e^{-\log^2 n}) = 1 - O(N^{-3})$. □

4.4 Step 4

Lemma 4.7 *With high probability, Step 4, computing the global ranks of the bracketing elements, can be done in $n + 2g + o(n)$ steps without increasing the queue size by more than a constant additive term.*

Proof: The ranks of the bracketing elements can be calculated by collecting counts of the numbers of elements smaller than each of the bracketing elements. This is straightforward to do in $n+2g$ steps for the elements lying inside the g-diamond. The elements outside the g-diamond need a little more care. However, note that these elements are inside the filtering diamond at time $n + o(n)$ and therefore information about them can be propagated to the middle diamond by the required time. Details can be found in Section A.3. □

4.5 Putting the Steps Together and Reducing the Queue Size

We now note that several of the steps of the algorithm can be done simultaneously. Step 1c can be done simultaneously with Step 1a in $n + o(n)$ steps. This is because packets performing Step 1c use only edges inside the $n/8$-diamond, and they can follow the packets performing the deterministic algorithm in Step 1a. Since the last packet performing the deterministic algorithm must enter the g-diamond within $n + o(n)$ steps, it cannot delay the packets performing Step 1c after $n - (n/8 - g)$ steps. Only $0.03n$ packets are routed in Step 1c, and for all values of g such that $n/8 - g > 0.03n$, Step 1c can be finished in $n + o(n)$ steps. Steps 1b and 1d can also be done simultaneously with Steps 1a and 1c in $n + o(n)$ steps with high probability by giving priority to the sample elements. Since there at most $o(n)$ sample packets traveling in any row or column with high probability (see Lemma 4.3), the packets performing Step 1a and 1c are not delayed by more than $o(n)$ steps. Step 1e requires an additional $(n/48 + 1/8)n - g + o(n)$ steps (Lemma 4.6).

Steps 1 and 2 can be done simultaneously by giving priority to packets performing Step 2; this causes an additional delay of at most $o(n)$ steps with high probability as in [7]. Step 2 can be done in $n + g + o(n)$ steps, and Step 1 can be done in $n(1 + 1/8 + n/48) - g + o(n)$ steps. To minimize the running time, we equate the two to get $g \simeq 0.07n$ steps. Packets performing Step 5 have priority over packets performing other steps. Since there are at most $o(n)$ such packets, they do not cause a delay of more than $o(n)$ steps (Lemma 4.7). Therefore the entire algorithm can be finished in $n + 2g + o(n) = 1.15n$ (see [7] for the analysis of Steps 3 and 4).

In the description above, the only steps that require non-constant size queues are Steps 1a, 1c, and 1e. To achieve constant size queues here, we use a redistribution technique described in [9]. Essentially, packets that are moving along a row towards the perimeter of the diamond stop at the first node which has a non-full queue. Therefore, each packet can be within distance ϵn of its destination, assuming each processor has a queue of size $O(1/\epsilon)$, where $\epsilon > 0$ is a constant. This technique adds an additional $O(\epsilon n)$ additive term to the running time (where ϵ is a constant that depends on the queue size).

This concludes the proof of Theorem 4.1.

5 Conclusions

Our main lower bound result shows that there is no distance-optimal algorithm for selection on a mesh in a model that allows constant-size queues, replication of packets, and a constant number of knowledge steps. The best previously known algorithm for selection, and also the algorithm in this paper, satisfy the restrictions of this model. Other lower bounds are also presented, which improve on previous work of Kunde [10].

In contrast to our negative results, we obtained a randomized algorithm for selection at the center processor that runs in $1.15n$ steps with high probability. Our algorithm is more adaptive than previous algorithms and makes clever use of sampling at different regions of the mesh, and of routing packets outside their initial quadrant, to effectively eliminate packets that are unlikely to be the median. In this way, our algorithm reduces the routing bottleneck of previous algorithms to obtain an improved running time.

Can our algorithm be improved to yield further gains in the running time? It is natural to ask whether our choice of filtering diamond at $n/8$ is optimal, and whether adapting the routes an additional constant number of times could improve the running time even further. We claim that such variants of our algorithm can at best lead to minor improvements in the running time, while greatly increasing the complexity of the algorithm. In the next two paragraphs, we give a brief justification of this claim.

Note that our algorithm pays two penalties that cause the running time to be sub-optimal. First, n steps are required to route all n^2 elements inside the $n/8$ filtering diamond, because each of the $n/2$ filtering processors can send in at most two packets in one step. Optimally, the distance bound of $7n/8$ steps should be sufficient to get all likely candidates for the median inside the $n/8$ diamond. Hence, our algorithm pays a penalty of $n/8 = 0.125n$ steps. We call this the bandwidth penalty, since it is due to the limited bandwidth of the filtering diamond. The second penalty paid by our algorithm is due to the pipelining of packets from the filtering processors, at Step 1e. Our algorithm succeeds in keeping the pipelining penalty low, at about $0.02n$ steps. Two reasons that the pipelining penalty is so low are as follows. First, the number of live elements accessible to each filtering processor decreases over time; in our algorithm, filtering does not occur until time n, by which time the number of live elements accessible to

each filtering processor is relatively small. Second, our routing scheme may route elements far outside their quadrants, thus increasing the number of elements which are eliminated in the filtering step.

The bandwidth penalty can be reduced if the radius of the filtering diamond is increased, or alternatively if additional filtering steps are performed. However, the pipelining penalty must increase as a result, because either the number of live elements routed to each filtering processor increases, or because elements may no longer be routed far outside their quadrants, or for both of these reasons. Hence, there is a trade-off between the bandwidth penalty and the pipelining penalty. It is infeasible to devise a general way to measure this trade-off for different choices of filters, since the trade-off depends on several factors, such as the routing scheme, which needs to be completely redesigned for each choice of filters. We considered several filtering strategies, and in all cases we found that any potential reduction of the bandwidth penalty results in a corresponding increase in the pipelining penalty. We conclude that an approach based on filtering is not likely to lead to a significant improvement in running time.

It remains an open question whether a distance-optimal algorithm for selection on the mesh exists. Our lower bounds show that, in order to achieve such a bound, the number of knowledge steps of an algorithm must grow with the input size. The previous paragraphs argue that, even with an arbitrary number of knowledge steps, it is unlikely that a distance optimal algorithm can be constructed using a filtering approach, where filtering diamonds periodically eliminate all elements which are no longer live. An alternative approach might be to construct a highly dynamic scheme to route elements towards the center, where the route of an element at a given time depends not just on whether it is live or not. Instead, the route of an element may depend on such parameters as its relative order among the live elements that are accessible to the processor at which it currently resides. Analyzing or implementing an algorithm based on this approach would likely be difficult, however.

Finally, we note that our upper bound result is a randomized algorithm. Are there deterministic algorithms that match these bounds? Recent work of Kaufmann *et. al.* [8] may yield a deterministic $1.15n$ algorithm. Another open question is whether the algorithms developed in this paper can be used to to improve on previous algorithms for higher dimensional meshes.

6 Acknowledgements

We would like to thank Danny Krizanc and Assaf Schuster for encouragement and useful insights regarding this work and the anonymous referees for useful comments.

References

[1] A. Agarwal, B. Lim, D. Kranz, and J. Kubiatowicz. APRIL: A processor architecture for

multiprocessing. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 104–114, 1990.

[2] D. Angluin and L. Valiant. Fast probabilistic algorithms for hamiltonian circuits and matchings. *Journal of Computer and System Science*, 18:155–193, 1979.

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

[4] A. Condon and L. Narayanan. Upper Bounds for Sorting and Selection on Meshes. DIMACS Technical Report Number 94-34, 1994.

[5] Y. Han, Y. Igarashi, and M. Truszczynski. Indexing schemes and lower bounds for sorting on a mesh-connected computer. Technical Report Number 114-88, University of Kentucky, Lexington, 1988.

[6] C. Kaklamanis and D. Krizanc. Optimal sorting on mesh-connected processor arrays. In *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architecture*, pages 50–59, 1992.

[7] C. Kaklamanis, D. Krizanc, L. Narayanan, and A. Tsantilas. Randomized sorting and selection on mesh-connected processor arrays. In *Proceedings of the 3rd Annual ACM Symposium on Parallel Algorithms and Architecture*, pages 17–28, 1991.

[8] M. Kaufmann and J. Sibeyn and T. Suel. Derandomizing algorithms for routing and sorting on meshes. processors. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 669–679, 1994.

[9] D. Krizanc, L. Narayanan, and R. Raman. Fast deterministic selection on mesh-connected processor arrays. *Algorithmica*, 15: 319–332, 1996.

[10] M. Kunde. l-selection and related problems on grids of processors. *Journal of New Generation Computer Systems*, 2:129–143, 1989.

[11] F. T. Leighton. Average case analysis of greedy routing algorithms on arrays. In *Proceedings of the 2nd Annual ACM Symposium on Parallel Algorithms and Architecture*, pages 2–10, 1990.

[12] T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan-Kaufmann, San Mateo, 1992.

[13] F. Leighton, F. Makedon, and I. Tollis. A $2n - 2$ step algorithm for routing in an $n \times n$ array with constant size queues. In *Proceedings of the 1989 ACM Symposium on Parallel Algorithms and Architecture*, pages 328–335, 1989.

[14] S. L. Lillevik. Touchstone program overview. In *Proceedings of the 5th Distributed Memory Computing Conference*, Charleston, SC, April 9-12 1990.

[15] L. Narayanan. *Selection, Sorting, and Routing on Mesh-Connected Processor Arrays.* PhD thesis, University of Rochester, 1992.

[16] S. Rajasekaran and T. Tsantilas. Optimal algorithms for routing on the mesh. *Algorithmica*, 8:21–38, 1992.

[17] R. Reischuk. Probabilistic parallel algorithms for sorting and selection. *SIAM Journal of Computing*, 14(2):396–411, May 1985.

[18] C. Schnorr and A. Shamir. An optimal sorting algorithm for mesh-connected computers. In *Proceedings of the Eighteenth Annual ACM Symposium on the Theory of Computation*, pages 255–263, 1986.

A Appendix

A.1 Proof of Lemma 4.1

In our analysis, we make extensive use of the following fact [3, 2].

Fact A.1 *(Bernstein-Chernoff bounds) Let $S_{N,p}$ be a random variable having binomial distribution with parameters N and p ($S_{N,p}$ is the sum of N independent Bernoulli variables each with mean p). Then, for any h such that $0 \leq h \leq 1.8Np$,*

$$P(S_{N,p} \geq Np + h) \leq \exp\left(-h^2/3Np\right).$$

For any $h > 0$,

$$P(S_{N,p} \leq Np - h) \leq \exp\left(-h^2/2Np\right).$$

Lemma 4.1 *Routing all the packets into the diamond of radius $n/8$ can be done in $n + o(n)$ steps using constant size queues with high probability in such a way that:*

(i) Half of the packets are routed inside the gathering diamond.

(ii) Each of the remaining packets is routed to a destination chosen uniformly at random from a set of $3n/8$ of the filtering processors.

(iii) Each of the filtering processors is a potential destination for an equal number of packets.

Proof: We will describe here the routing of packets that were initially in the top left quadrant. We divide the packets into several different categories based on which region they originated in the quadrant. (see Figures 5 and 4). We route the packets in regions X, Y, X' and Y' using a deterministic algorithm into the gathering diamond. The remaining packets are routed by a randomized algorithm; the randomness is used solely in choosing a destination from a set of $3n/8$ filtering processors (to be specified later). Each packet is then routed by a simple greedy algorithm to its chosen destination.

First we analyze the packets performing the deterministic algorithm.

Claim A.1 *Packets in regions X, X', Y and Y' can be routed in to the g-diamond (where $n/16 \leq g < n/8$) in at most n steps.*

Proof: We will discuss the routing of the packets in the regions X and X' (see Figure 4). Packets in the bottom g rows in region X go straight into the boundary of the g-diamond in $n - g$ steps. Packets in the $n/8 - g$ rows above that use row edges to get past the boundary of the $n/8$-diamond, and then turn into one of the next $n/8 - g$ columns, in such a way that each of these columns has the same number of packets turning into it. Packets in the region X' first move $3n/8$ steps along column edges alone. Thus each such packet is now at a processor originally occupied by some packet in X. It is easy to see that in the row $n/2 - i$ there are $n/8 - i$ packets from the region X'. At this point, each of these packets uses the same algorithm that the corresponding packet originating in region X would have used. Packets in the regions Y and Y' follow a similar algorithm, except with rows and columns interchanged in the description.

We now analyze the traffic in column $n/2 - i$ (where $0 \leq i \leq n/8 - g$). In this column the last of the Y packets (that went straight in to the boundary of the g diamond) enters the g-diamond in $n/2 - g + i$ steps. After this, we can route the packets from region X that just turned in. Since an equal number of these turn into each of $n/8 - g$ columns, and there are $3n/8(n/8 - g) + (n/8 - g)^2/2$ packets in the region X that turn, column $n/2 - i$ gets at most $3n/8 + (n/8 - g)/2$ packets. When these are done, we will route in the packets that come in from the region Y'; in the column $n/2 - i$ there are $n/8 - i$ of them. Finally, we route in the packets from the region X' that turn in via column edges. There are at most $(n/8 - g)/2$ of these in column $n/2 - i$. The total amount of time taken is $(n/2 - g + i) + (3n/8 + (n/8 - g)/2) + (n/8 - i) + (n/8 - g)/2 = n + n/8 - 2g \leq n$. □

We go on to describe the randomized algorithm for the remaining packets, each of which will be routed to some filtering processor. Each packet chooses uniformly at random from among the $n/8$ processors in its own quadrant and the $n/4$ processors in the two adjacent quadrants. Each of these filtering processors can be reached via at least two edges (a row edge and a column edge; note that the processors at the points of the diamond can be reached via 3 edges) from outside the $n/8$-diamond. Packets are routed via a row edge to destination processors in the same vertical half as the origin, and by column edges to destination processors in the same horizontal half as the origin. For example, consider a packet in the NE quadrant. If its destination is a processor in the NW quadrant, the packet would use a row edge to get to that processor. If instead its destination is in the SE quadrant, then it must use a column edge to get to its destination. If the packet originates in the same quadrant as the destination processor, then it can be routed via either the row edge or the column edge incident on the destination processor with equal probability.

Each packet can therefore be thought of as entering the $n/8$-diamond via a set of $n/2$ edges called its *entry set*, and there is a probability distribution over the entry set. The packet chooses an edge from the entry set according to this distribution; this edge is called the *entry edge*. The

Phase	Time interval	Regions	Area	Packets contributed w.h.p.	Total
1	$4n/8+1$ to $5n/8$	A	$9n^2/128$	$3n/32 \pm o(n)$	$n/8 \pm o(n)$
		A'	$n^2/128$	$n/48 \pm o(n)$	
		I	$n^2/256$	$n/96 \pm o(n)$	
2	$5n/8+1$ to $6n/8$	B	$5n^2/128$	$5n/96 \pm o(n)$	$n/8 \pm o(n)$
		B'	$5n^2/256$	$5n/96 \pm o(n)$	
		J	$n^2/128$	$n/96 \pm o(n)$	
3	$6n/8+1$ to $7n/8$	C	$n^2/64$	$n/48 \pm o(n)$	$n/8 \pm o(n)$
		C'	$n^2/32$	$n/12 \pm o(n)$	
		K	$n^2/128$	$n/48 \pm o(n)$	
4	$7n/8+1$ to n	D'	$n^2/32$	$n/12 \pm o(n)$	$n/8 \pm o(n)$
		E'	$n^2/64$	$n/24 \pm o(n)$	

Table 1: Schedule for routing packets in the top left quadrant to the critical edge in row $5n/8$.

probability distribution is such that each entry edge in the same quadrant as the packet is chosen with probability $4/3n$ and each of the other edges is chosen with probability $8/3n$. This implies that each destination processor is chosen with equal probability $8/3n$.

For example, the entry set for packets in the top left quadrant contains the row edges (in the left half) that go into the $n/8$-diamond in the rows $\{3n/8, \cdots, 5n/8\}$ and the column edges (in the top half) in the same set of columns. The probability distribution described above ensures that each filtering processor is a potential destination for the same number of packets, as required by the statement of the lemma.

All packets are routed greedily to their destinations. A packet that first uses column edges to get to its destination row, and then uses row edges to get to its destination is said to use a *column-row* route. Similarly, packets that travel first in rows and then in columns are said to follow a *row-column* route. Packets whose entry edge is a row edge use a column-row greedy route and those whose entry edge is a column edge use a row-column greedy route. We will determine the last step when a packet enters the $n/8$ diamond via one of the edges detailed above. Consider the entry edge in the row $5n/8$. This is one of the two edges that is farthest from the packets in the top left quadrant. We call it the *critical edge* and analyze the traffic across it in detail. Our proof strategy is to show that with high probability, barring $o(n)$ steps, we can keep supplying the critical edge with packets that need to cross it.

At time $4n/8$, the last of the packets from regions X and Y has used the critical edge. We will analyze the algorithm in phases, consisting of $n/8$ steps each. We divide the mesh up into regions, and provide a schedule that specifies the regions whose packets cross the critical edge in each phase (see Table A.1).

Claim A.2 *The schedule in Table A.1 routes the last packet across the critical edge by time step $n + o(n)$ with high probability.*

Proof: We show that in each phase, packets can cross the critical edge according to Table A.1. We use the following fact extensively in our proof.

Fact A.2 *(a) Given any set of n^2/a packets in the top left quadrant, with high probability, $8n/3a \pm o(n)$ packets from the set choose to cross the critical edge.*

(b) Given any set of n^2/b packets from the bottom left quadrant, with high probability, $4n/3b \pm o(n)$ packets from the set choose to cross the critical edge.

Proof: Follows from a straightforward application of Fact A.1. □

We note that only packets from the top and bottom left quadrants attempt to use the critical edge. We use Fact A.2 to find the number of packets contributed by each region with high probability in Table A.1. We also note that the contention between the packets that use a greedy column-row routing strategy can only occur during the second phase (the routing in rows) and this can be resolved according to the schedule in Table A.1. Further, there can be no conflict between packets performing the column-row routing and those performing the row-column routing. This is because the second phase of the column-row routing is only done in the middle $n/4$ rows, and in those rows there are no packets performing the row-column algorithm.

According to Table A.1, each phase has $n/8 \pm o(n)$ packets assigned to be routed into the $n/8$-diamond. Since each phase is exactly $n/8$ steps long, there may be an excess or deficit of $o(n)$ packets in each phase. The delays caused by the deficits are waited out and the excesses are routed at the end of n steps, causing a delay of at most an additional $o(n)$ steps.

Phase 1 (after 4n/8 steps):

The region A initially has $9n^2/128$ packets, and using Fact A.2 as in Table A.1, it is evident that with high probability, the region A contributes $(9/2)(n/48) \pm o(n)$ packets, and A' contributes $n/48 \pm o(n)$ packets. This gives us a total of $11n/96 \pm o(n)$ packets that can all reach the critical edge in 4n/8 steps. We route these packets in for the next $11n/96$ steps.

Next we route in packets from the region I; from Table A.1 there are $n/96 \pm o(n)$ packets that choose the critical edge with high probability from this area, and each of them can reach the critical edge in $4n/8 + n/16$ steps.

Phase 2 (after 5n/8 steps):

¿From Table A.1, with high probability, the region B contributes $5n/96 \pm o(n)$ packets, and region B' contributes $5n/96 \pm o(n)$ packets. All of these can reach the critical edge in $4n/8$ steps, and we route these packets over the critical edge next, thus keeping it occupied until step $5n/8 + 5n/96 + 5n/96 = 5n/8 + 5n/48$.

Next we route in the packets from the region J; from Table A.1, there are $n/48 \pm o(n)$ packets, that choose the critical edge from this area with high probability, and each of them can reach it in $5n/8 + n/16$ steps.

Phase 3 (after 6n/8 steps):

¿From Table A.1, with high probability, the region C contributes $n/48 \pm o(n)$ packets and region C' contributes $n/12 \pm o(n)$ packets, all of which can reach the critical edge in $6n/8$ steps. We route these in next, keeping the critical edge occupied until step $6n/8 + n/48 + n/12 = 6n/8 + 5n/48$.

Next we route in the packets from the region K; using Table A.1, there are $n/48 \pm o(n)$ packets that choose the critical edge with high probability from this region, and each of them can reach it in $6n/8 + n/16$ steps.

Phase 4 (after 7n/8 steps):

¿From Table A.1, with high probability, the region D' contributes $n/12 \pm o(n)$ packets. We route these in next, keeping the critical edge occupied until step $7n/8 + n/12$ steps.

Table A.1 shows that with high probability, the region E' contributes $n/24 \pm o(n)$ packets, each of which can reach the critical edge within n steps. But in fact, many of the packets from region E' can reach the critical edge much sooner. In particular, using Fact A.1, we can show that with high probability, at least $\sqrt{n}/3$ packets arrive at the critical edge every $\sqrt{n}$ steps from the region E'. Therefore with high probability, all the packets from the region E' can be routed across the critical edge within $n + o(n)$ steps. This completes the discussion of the critical edge under discussion.

We claim that all the other entry edges can be analyzed in a similar manner. In row $n/2 - i$ $(0 \leq i \leq n/8)$, the last of the packets from region X has crossed the entry edge at time $3n/8 + i$, and we can start routing packets using the randomized algorithm from this time. Therefore the schedule for routing packets is shifted forward appropriately for each entry edge, and the regions that can reach the entry edge in time are also defined differently. Further, in the case of the entry edge in the row $5n/8$, no packets from the regions X' and Y' will cross it. However, this is not the case for an entry edge in any other row: there are packets doing the deterministic algorithm from regions X' and Y' that will be routed across it after time step $3n/8$. We give priority to the packets performing the deterministic algorithm to resolve any conflicts with packets performing the randomized algorithm. To illustrate this point, we briefly discuss the entry edge in row $n/2$. There are $n/8$ packets from the corner sub-mesh (that are performing the deterministic algorithm) that will cross this entry edge. They are routed across in time steps $6n/8$ to $7n/8$. Then the packets performing the randomized algorithm are routed in the next $n/8$ steps.

Finally, we note that the queue size can only increase when packets attempt to turn, and therefore, processors outside the middle $n/4$ rows and columns require at most constant size

queues. Consider a processor P in row $3n/8 + i$ where $0 \leq i < n/4$. There are at most $3n/4$ packets in its column that perform the randomized algorithm, and each will choose to turn at P with probability at most $8/3n$. Therefore, the expected number of packets that turn at P is a constant, and using Fact A.1, with high probability, $O(\sqrt{\log n})$ packets may turn into any of these processors, and at most $O(\log n)$ packets may turn into a set of $\log n$ processors neighboring P. We use a technique from [16] to ensure that each processor requires at most constant-size queues. Essentially, we divide each row and column into blocks of size $\log n$. Now, a packet that wishes to turn at processor P will instead attempt to turn at the first processor in the block containing P that does not have a full queue. The observation above shows that with high probability, there will be a non-full queue in the block containing P. The extra delay introduced is only $O(\log n)$. Therefore, only the destination processors (the filtering processors and the processors on the boundary of the gathering diamond) require non-constant queues.

This completes the proof that all the packets can be routed into the $n/8$-diamond as required by the lemma in $n + o(n)$ steps. □

A.2 Proof of Lemma 4.3

Lemma 4.3 *With high probability, for all filtering processors P, the splitters $s_1(P), s'_1(P)$ can be selected by time $n-0.03n+o(n)$ and the splitters $s_2(P), s'_2(P)$ can be selected by time $n+o(n)$, without increasing the size of the queues by more than a constant additive term.*

Proof: First, consider the routing of a single sample to some filtering processor P. Define P's *sample block* to be the square block of side $\lfloor n^{11\delta/2} \rfloor$ centered at P. Each sample element chooses a destination processor uniformly at random from among the processors in P's sample block. From Lemma 4.2 there are $O(N^{5\delta} \log N)$ elements in each sample, with high probability. There are $\lfloor N^{11\delta/2} \rfloor$ destinations in each block; therefore the expected number of sample elements choosing a particular destination processor is $O(N^{-\delta/2} \log N)$. From Fact A.1, it follows that with probability greater than $1 - e^{-N^{\delta/4}}$ at most a constant number of sample elements choose a given destination within P's sample block.

Consider a given column of the mesh. The expected number of sample elements from this column is $O(n^{5\delta} \log N)$. Again, using Fact A.1, with probability at least $1 - e^{-n^{5\delta}}$, there are $O(n^{5\delta} \log N)$ sample elements in this column.

Note that there are $\lfloor n^{1-11\delta/2} \rfloor$ different samples; therefore, from a given column, the total number of sample elements, over all the different samples, is $O(n^{1-\delta/2})$ which is only $o(n)$ packets. Each sample is routed using a greedy algorithm, where each sample element first travels to the right row and then to the right column. Clearly, no collisions can occur in the columns since each column is a linear array with at most one packet per node initially. The queue size can only increase when one or more packets turn at a node. Each of the elements from a particular sample picks one of $\lfloor n^{11\delta/2} \rfloor$ rows in its sample block to turn into. Using Fact A.1, with probability greater than $1 - e^{-n^{\delta/4}}$, the number of packets turning at a given

node does not exceed a constant. Hence, a single sample can be routed within $n - 0.03n + o(n)$ steps with probability $1 - O(e^{-n^{\delta/4}})$, and the queue size does not increase by more than a constant additive term.

Since there are at most $n^{1-11\delta/2}$ blocks, the probability that there exists any sample in the mesh where the routing cannot be done within $n - 0.03n$ steps is $O(n^{1-11\delta/2}e^{-n^{\delta/4}})$. Now we consider the simultaneous routing of the samples. Packets belonging to different samples clearly turn at different rows, and so different sample elements do not collide with each other while turning into rows. Thus, the sample elements are delayed at most $o(n)$ steps, and can be routed in $n - 0.03n + o(n)$ steps with high probability.

Finally, receiving the splitters from neighboring processors can be done in $o(n)$ steps since every processor is within distance $o(n)$ of a processor which samples. □

A.3 Proof of Lemma 4.7

Lemma 4.7 *With high probability, Step 4, computing the global ranks of the bracketing elements, can be done in $n + 2g + o(n)$ steps without increasing the queue size by more than a constant additive term.*

The ranks of the bracketing elements can be calculated by collecting counts of the numbers of elements smaller than each of the bracketing elements. At time $n + g + o(n)$ the bracketing elements have been broadcast to all the processors in the g-diamond, and counts containing the number of elements inside the g-diamond smaller than each of the bracketing elements can be propagated to the central processor in an additional $g + o(n)$ steps.

The same approach cannot be used for the packets outside the g-diamond since the time required to broadcast bracketing elements to the filtering processors and collect counts would be at least $n + n/4 > n + 2g$. Instead we use a different strategy. At time step $n + o(n)$, the packets that remain at a filtering processor are those that are either not tentatively live or were not selected to be routed to the gathering diamond. (Some live elements may also be present but they will be routed inside the the g-diamond within $n + g$ steps and will be accounted for as described earlier; we will not consider them here.) Recall that with high probability, the bracketing elements lie in between both pairs of splitters of each filtering processor P. The elements at P that are smaller than $s_2(P)$ (and not live) are therefore the only elements at P that are smaller than both the bracketing elements. Each filtering processor can send its count toward the center processor, and counts from different filtering processors are added and propagated further by intermediate processors. This count reaches the central processor in $n + n/8 + o(n) < n + 2g$ steps. In order to check for the possibility that the bracketing elements do not lie in between some pair of splitters, the center processor in each block of filtering processors sends the splitter elements to a destination in a unique row in the central block. Since all the destinations are in different rows, the routing can be accomplished without collisions. If the bracketing elements b and b' lie in between every pair of splitters (which is

true with high probability, by Lemma 4.4), then the counts of the number of elements smaller than each bracketing element outside and inside the gathering diamond are added to find their global ranks. □

A.4 Proof of Theorem 3.3

Our lower bounds for maximally adaptive algorithms apply to algorithms that are not restricted to be comparison-based. In other words, the action of a processor at time step t may depend on *values* of packets accessible to it, not just comparisons of packets accessible to it. This is similar to the models of Schnorr and Shamir and Kunde [18, 10]. We define a new notion of "liveness" of packets. The following definitions and lemmas pertain to a fixed maximally-adaptive algorithms. Given an element Q, let $Accessible(Q, I, t)$ be the union of the sets of input elements which are accessible to each packet containing element Q, on input I at the most recent knowledge step that occurs up to and including time t. We say an input I' is *consistent with* $Accessible(Q, I, t)$ if for each element $Q' \in Accessible(Q, I, t)$, element Q' is also in I', and is initially in the same location in both I and I'. We say element Q is *live at time* t if there is some input consistent with $Accessible(Q, I, t)$, such that Q is the median on that input. We next describe some useful facts about live elements, for a given algorithm and input I.

Fact A.3 *Q is live at time t if $|Accessible(Q, I, t)| \leq n^2/2$.*

Fact A.4 *Q is live at time t if Q is of rank at most k less than the median, and there is a set of at least k elements greater than the median which are not in $Accessible(Q, I, t)$.*

To see why Fact A.4 is true, let I' be an input consistent with $Accessible(Q, I, t)$, obtained by changing the k elements greater than the median that are not in $Accessible(Q, I, t)$, so that they are now $< Q$. Then, Q is the median element of input I'. Hence by definition, Q is live at time t.

Lemma A.1 *Fix an algorithm and an input I. If I' is consistent with $Accessible(Q, I, t)$ then*

$$Accessible(Q, I, t) = Accessible(Q, I', t).$$

The following claim is useful in proving part (b) of the theorem below.

Claim A.3 *Suppose a packet P is outside the d-diamond at time $t \geq n/2$ where d is an integer. Then the number of elements not accessible to P at time t is at least:*

(i) $(n-2-t)^2 + (1/2)(n-1-t+d)^2$, *if* $t - d > n/2$.

(ii) $(1/2)(n-2-t)^2 + (1/2)(n-1-t+d)^2$, *if* $t - d < n/2$,

Proof: The results are easily verified by referring to Figure 6. The shaded regions in these figures contain the locations of elements which are inaccessible to P', if P' in the NW quadrant, at distance d from the center, and is equidistant from the SW and NE quadrants. Also, the number of elements inaccessible to P' is a lower bound on the number of elements inaccessible to P. □

Lemma A.2 *$\epsilon > 0$ such that all packets containing some live element Q are at distance $\geq t+\epsilon n$ from the center at some time $n-t$. Then the algorithm is not distance-optimal.*

Proof: Since Q is live at time $n-t$, there is some input I' consistent with *Accessible*(Q, I, t) such that Q is the median element of input I'. On input I' at time $n-t$, all packets containing Q are also at distance $\geq t+\epsilon n$ from the center. This is because the positions of packets containing Q on inputs I and I' depend only on the input elements in *Accessible*(Q, I, t), *Accessible*(Q, I', t), respectively. Also, by Lemma A.1, *Accessible*(Q, I, t) = *Accessible*(Q, I', t). Hence, the time to get some packet containing Q to the center is at least $(1+\epsilon)n$, and so the algorithm cannot be distance-optimal. □

Theorem 3.3 *There is no distance-optimal, maximally adaptive deterministic algorithm for finding the median:*

(a) if the queue size is 1.

(b) if packets are not replicated and the queue size is 2.

(c) if packets are quadrant constrained and the queue size is 3.

(d) if packets are not replicated, are quadrant constrained and the queue size is 4.

Proof: Part (a): Fix any algorithm. The idea is to show that all elements are live at time $\lfloor n/2 \rfloor - 1$. Therefore, every element must be at some node. Since the queue size is 1, some live element is at a corner processor, and hence is at distance $n-1$ from the center processor at time $\lfloor n/2 \rfloor - 1$. Then it follows immediately from Lemma A.2 that the algorithm requires $1.5n - 3$ steps.

We now show by induction on t that all elements are live at time $t \leq \lfloor n/2 \rfloor - 1$. Clearly this is true at time $t = 0$. Suppose it is true at time $t-1$. Then at the tth step, no element can be discarded, because at the tth step, the algorithm depends only on those elements accessible at time $t-1$. Hence at time t, every element must be at some node, and so there is exactly one packet containing every element, since the queue size is 1.

Recall that the elements accessible to any given element Q are exactly those elements initially within distance t from the processor containing Q at time t. Since $t \leq \lfloor n/2 \rfloor - 1$, at most $2t^2 + 2t + 1 \leq n^2/2$ elements are accessible to any given element Q. Hence by Fact A.3, the element Q is live.

Part (b): Fix an input I. Just as in Theorem 3.3, since only one packet containing any element Q exists at any time, there are $\leq n^2/2$ elements in *Accessible*(Q, I, t) at time $\lfloor n/2 \rfloor - 1$. Hence every element Q is live, by Fact A.3.

Let $0 < \epsilon < 1/2$ (the value of ϵ will be chosen later). Consider the set S of live elements that are not inside the $(n/2 - \epsilon n)$-diamond at time $\lfloor n/2 \rfloor - 1$. There are at least $n^2 - 4(n/2 - \epsilon n)^2 - 4(n/2 - \epsilon n) - 2$ such elements, using the fact that the queue size is 2. At time $\lfloor n/2 \rfloor - 1 + \lfloor \epsilon n \rfloor$, these elements must still be outside the $(n/2 - 2\epsilon n)$-diamond.

At this time, applying Claim A.3 part (i), for each element $P \in S$, there are still at least $(1/2)(\lfloor n/2 \rfloor - \lfloor \epsilon n \rfloor)^2 + (1/2)(n - 3\epsilon n)^2$ elements not accessible to Q, that is, not in *Accessible*$(Q, I, \lfloor n/2 \rfloor + \lfloor \epsilon n \rfloor - 1)$. Choose ϵ such that $(1/2)(\lfloor n/2 \rfloor - \lfloor \epsilon n \rfloor)^2 + (1/2)(n - 3\epsilon n)^2 \geq n^2/2$. Then, all elements in S are live at time $\lfloor n/2 \rfloor + \lfloor \epsilon n \rfloor - 1$, by Fact A.3.

However, not all elements of S are inside the $(n/2 - \epsilon n)$-diamond. This is because at most $2\epsilon n(n/2 - \epsilon n)$ elements can cross into the $(n/2 - \epsilon n)$-diamond in $\lfloor \epsilon n \rfloor$ steps. Hence, at least $n^2 - 4(n/2 - \epsilon n)^2 - 4(n/2 - \epsilon n) - 2 - 2\epsilon n(n/2 - \epsilon n)$ elements of S are outside the $(n/2 - \epsilon n)$-diamond at time $\lfloor n/2 \rfloor + \lfloor \epsilon n \rfloor - 1$. This is $\Theta(n^2)$, for all ϵ in the range $0 < \epsilon < 1/2$. Hence for sufficiently large n, some element must be of distance at least $\lfloor n/2 \rfloor - \lfloor \epsilon n \rfloor + \epsilon' n$ from the center at time $\lfloor n/2 \rfloor + \lfloor \epsilon n \rfloor - 1$, for some $\epsilon' > 0$. By Lemma A.2, the algorithm runs in time at least $(1 + \epsilon')n$.

Part (c): Let $t = \lfloor n/4 \rfloor$. Consider an input I in which the set S of all $2t^2$ elements just below the median are in a fixed quadrant, say NW, and exactly those elements greater than the median are outside the $n/2$-diamond.

Suppose that the queue size is at most 3. Then, at any given time, the number of distinct elements of S that are contained in packets inside the $(t + \epsilon n)$-triangle of the NW quadrant is at most $(3/2)(t + \epsilon n)(t + \epsilon n + 1)$. Choose ϵ such that $(3/2)(t + \epsilon n)(t + \epsilon n + 1) < 2t^2$. Then, at time $n - t - 1$, some element $P \in S$ is not contained in any packet inside the $(t + \epsilon n)$-triangle.

But at time $n - t - 1$, all elements initially in the $2t$-corner of the SE quadrant are not accessible to any packet containing Q. There are at least $2t^2$ such elements. Thus by Fact A.4, Q is live at time $n - t - 1$. By Lemma A.2, at least $(1 + \epsilon)n - 1$ time is required to find the median.

Part (d): Let $t = \lfloor n/4 \rfloor$ and consider an input I, just as in Part (c), in which the set S of all $2t^2$ elements just below the median are in a fixed quadrant, say NW, and exactly those elements greater than the median are outside the $n/2$-diamond.

We claim that all $2t^2$ elements in S are live at time $n - t - 1$. This is because at least $2t^2$ elements outside the $(n/2 + 1)$-diamond are not accessible to every element of S (the set of $2t^2$ inaccessible elements may be different for different elements of S). Therefore, Fact A.4 applies.

Suppose that the queue size is at most 4. Then, at time $n-t-1$, the set of distinct elements that are contained in packets inside the $(t-\epsilon n)$-triangle is of size at most $2(t-\epsilon n)(t-\epsilon n+1)$.

Hence, there is a set S' of at least $2t^2 - 2(t-\epsilon n)(t-\epsilon n+1)$ live elements that are not contained in packets inside the $(t-\epsilon n)$-triangle at time $n-t-1$.

All of the elements in S' are still live at time $n-t-1+\lfloor \epsilon n \rfloor$. This is because all are still outside the $(t-2\epsilon n)$-diamond. Therefore, at least $(t-1-\epsilon n)^2 + (1/2)(2t-3\epsilon n)^2$ elements are not accessible, by Claim A.3, part (ii). For $0 < \epsilon < 1/32$, $(t-1-\epsilon n)^2 + (1/2)(2t-3\epsilon n)^2 \geq 2t^2$. Then, at least $2t^2$ elements greater than the median are not accessible to each element of S', and Fact A.4 applies.

Also, at most $2\epsilon n(t-\epsilon n)$ elements of S' have crossed into the $(t-\epsilon n)$-diamond at time $n-t-1+\lfloor \epsilon n \rfloor$. Hence at least $2t^2 - 2(t-\epsilon n)(t-\epsilon n+1) - 2\epsilon n(t-\epsilon n)$ live elements are still outside the $(t-\epsilon n)$-diamond at time $n-t-1+\lfloor \epsilon n \rfloor$. For any ϵ in the range $0 < \epsilon < 1/32$, $2t^2 - 2(t-\epsilon n)(t-\epsilon n+1) - 2\epsilon n(t-\epsilon n) > 0$. Hence, one of these live elements is of distance at least $t - \epsilon n + \epsilon' n$ from the center at time $n-t-1+\lfloor \epsilon n \rfloor$, for some $\epsilon' > 0$. Hence by Lemma A.2, the time required by the algorithm is at least $(1+\epsilon')n - 2$. $\square$

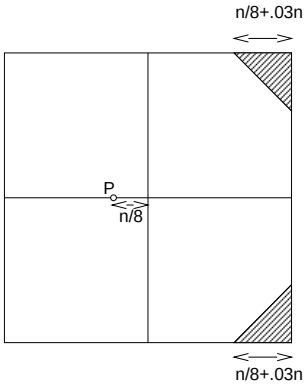

Figure 3: Elements inaccessible to filtering processor P at time $n - .03n$ are initially in the shaded area.

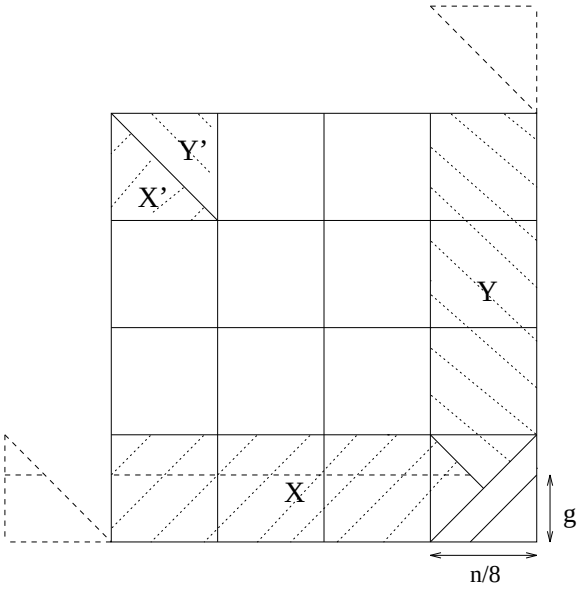

Figure 4: Routing packets using the deterministic algorithm.

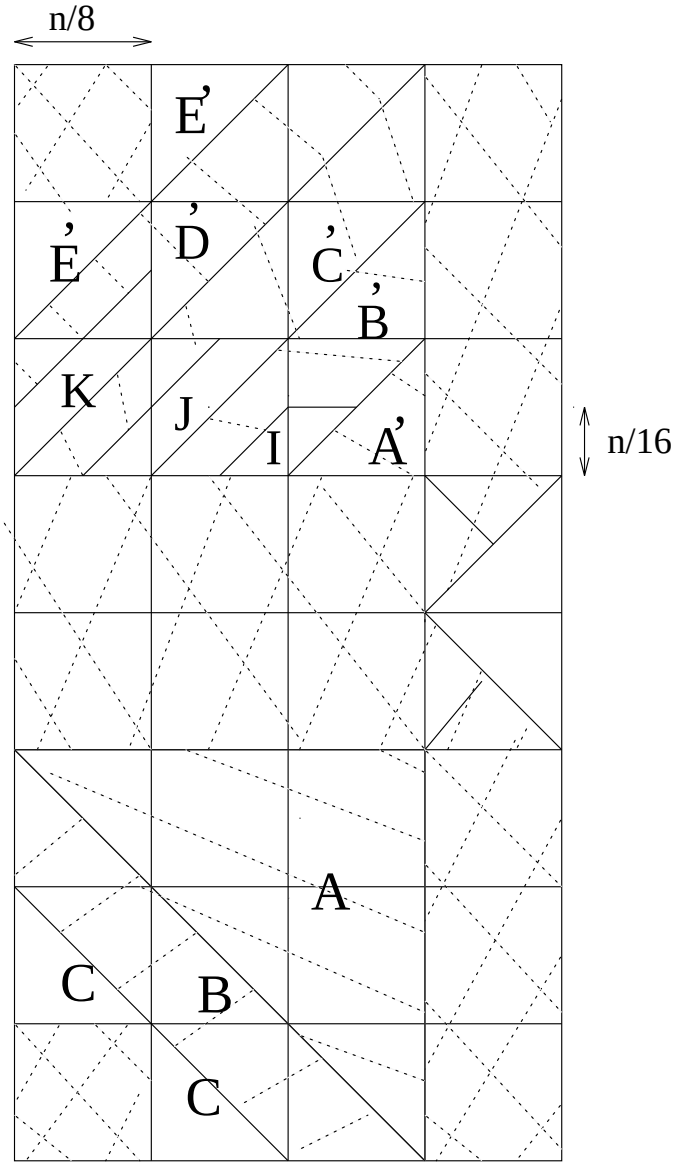

Figure 5: Schedule for routing packets using the randomized algorithm.

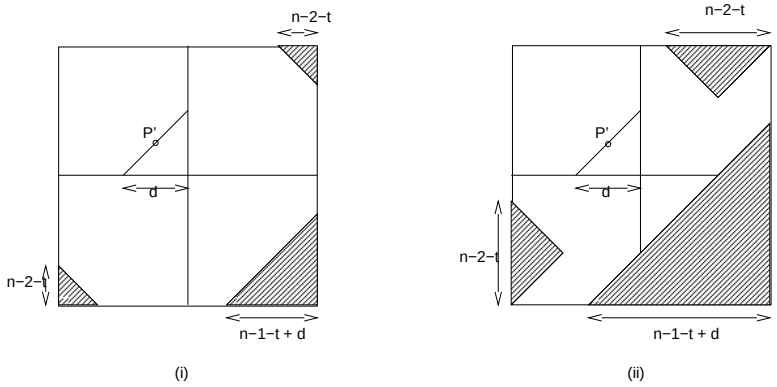

Figure 6: Packets in the shaded areas are inaccessible to processor P' at time t, where (i) $t - d < n/2$ and (ii) $t - d > n/2$.