On-line Admission Control and Circuit Routing for High Performance Computing and Communication

Baruch Awerbuch * Rainer Gawlick † Tom Leighton ‡ Yuval Rabani §

Abstract

This paper considers the problems of admission control and virtual circuit routing in high performance computing and communication systems. Admission control and virtual circuit routing problems arise in numerous applications, including video-servers, real-time database servers, and the provision of permanent virtual channels in large-scale communications networks. The paper describes both upper and lower bounds on the competitive ratio of algorithms for admission control and virtual circuit routing in trees, arrays, and hypercubes (the networks most commonly used in conjunction with high performance computing and communication). Our results include optimal algorithms for admission control and virtual circuit routing in trees, as well as the first competitive algorithms for these problems on non-tree networks. A key result of our research is the development of on-line algorithms that substantially outperform the greedy-based approaches that are used in practice.

1 Introduction

1.1 The Problem

This paper considers the problems of admission control and virtual circuit routing in networks for high performance computing and communication. Admission control and virtual circuit routing problems can arise whenever there is a request to send a large amount of data from one node in a network to another node. The *admission control* aspect of the problem is to decide whether or not the network can or should accommodate the request, and the *virtual circuit routing* aspect of the problem is to decide how to route the data if the request is to be accommodated. (The data is routed by establishing a path, called a *virtual circuit*, through the network that connects the two nodes that wish to communicate. The data packets can then be sent in a pipelined fashion along the path. In some networks, the bandwidth along the path is explicitly reserved.)

The admission control and virtual circuit routing problems arise in a variety of applications. For example, in parallel supercomputers, it is often necessary to have fast access to potentially large amounts of data that is stored remotely. Hence there needs to be a communications network embedded in the supercomputer that is capable of supporting such requests for data. In some supercomputers this data is routed using some form of virtual circuit routing. (In the past, many supercomputers have used packet routing where each packet uses its own path and no bandwidth reservations can be made. While this approach works for cooperative scientific applications, it may not be effective in commercial applications where the various tasks may not be cooperating. By reserving a certain amount of bandwidth on a virtual circuit, a particular task can be assured good performance.)

A large-scale video server can be constructed by using a supercomputer network to connect a large disk farm to a set of telecommunications lines. The network of the supercomputer is then used to route video (e.g. movies) to subscribers in real time. Oracle's Media Server, which currently runs on the NCube supercomputer, is an example of such a system [Buck94]. Each customer has a virtual circuit through the NCube that connects the disk containing the customer's movie to the customer's telecommu-

*Johns Hopkins University, Baltimore, MD 21218-2694, and MIT Laboratory for Computer Science, Cambridge MA 02139. Email: baruch@blaze.cs.jhu.edu. Supported by Air Force Contract AFOSR F49620-92-J-0125, ARPA/Army contract DABT63-93-C-0038, NSF contract CCR-9114440, DARPA Contract N00014-92-J-1799, and a special grant from IBM.

†MIT Laboratory for Computer Science, Cambridge, MA 02139. Supported by ARPA/Army contract DABT63-93-C-0038.

‡Department of Mathematics and Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139. This research was supported in part by Air Force Contract AFOSR F49620-92-J-0125 and DARPA Contracts N00014-91-J-1698 and N00014-92-J-1799.

§MIT Laboratory for Computer Science, Cambridge, MA 02139. Supported by ARPA/Army contract DABT63-93-C-0038.

0272-5428/94 $04.00 © 1994 IEEE

nications port. (This application demonstrates why circuit routing is better than packet routing for many tasks. Using packet routing, packets can arrive out of order, some packet can be seriously delayed, and performance guarantees are not possible. All of these issues can have a serious negative effect on the quality of the movie transmission.)

Real-time database servers represent another important application area. Many database applications, such as the join of large tables in a relational database, require the exchange of substantial amounts of data. Virtual circuit routing is useful in such applications since the operation, once it commences, can rely on a particular level of service from the network.

Admission control and virtual circuit routing also arise in the sale of bandwidth in telecommunications networks. In this situation, there is a company that owns a large telecommunications network and subscribers that purchase *permanent virtual circuits*, which they can then use at their discretion [GKR94]. (This is an example of where the bandwidth for the virtual circuit is explicitly reserved.)

Finally, we note that the supercomputer community has recently shown interest in constructing systems by interconnecting workstation-like nodes via high speed LANs [Lei93]. IBM's SP-2 is an example of such a supercomputer system. In view of the emergence of the ATM standard, which is based on virtual circuit routing, as the preferred architecture for high speed data networks, virtual circuit routing algorithms may become increasingly important for future supercomputers.

In all of these applications, the admission control and routing decisions need to be made in an *on-line* fashion. In particular, each decision must be made without knowledge of future requests. The on-line nature of the problem is a significant complication since a long virtual circuit path could block many potential short virtual circuit paths. Generally, systems attempt to maximize the number of requests that can be served. Thus complications due to the on-line nature of the problem arise when a request for a long path comes in just before several conflicting requests for short paths.

In practice, the greedy algorithm is typically used for admission control (i.e., if the bandwidth needed for a requested virtual circuit is available, then it is allocated). This greedy strategy seems natural especially when nothing is known about future requests. This paper shows, however, that the greedy strategy is often suboptimal over the long term. (Linear lower bounds for virtual circuit routing in the absence of admission control in general networks appear in [GKR94].)

In this paper we focus on the development of optimal (or near optimal) on-line admission control algorithms for trees, arrays, and hypercubes. We have chosen to focus on these networks for several reasons. First, these networks (or close variations thereof) form the architectural basis of most high performance communications networks. Second, the structure of these networks is rich enough so that the task of designing optimal algorithms for admission control and circuit routing is nontrivial. As a consequence, the networks provide a framework within which novel approaches to these problems can be exhibited. Lastly, the task of solving the admission control and circuit routing problems for general networks is currently beyond our reach. Indeed, any solution to these problems for general networks would provide an approximation algorithm for the maximum disjoint path problem, which appears to be very difficult even if the on-line constraint is lifted. (Given a graph G with a collection of source nodes $s_1, s_2, \ldots, s_n$ and terminating nodes $t_1, t_2, \ldots, t_n$ the *Maximum Disjoint Path Problem* is to connect as many of the sources to their respective sinks as possible using edge disjoint paths in G.)

As a performance measure for our algorithms, we will use *competitive analysis* [ST85, KMRS88]. The *competitive ratio* of an on-line virtual circuit routing algorithm is the maximum over all request sequences of the ratio of the number of requests that the optimal algorithm accepts for that sequence to the number of requests that the on-line algorithm accepts on the same request sequence. Specifically, for request sequence σ, let $A(\sigma)$ be the number of requests accepted by algorithm A on σ and let $O(\sigma)$ be the optimal number of requests that can be accepted from σ. Then the competitive ratio for A is the maximum over all σ of $A(\sigma)/O(\sigma)$. An algorithm with a low competitive ratio is one that performs close to the optimal algorithm on all request sequences. Since it does not make assumptions about the sequence of virtual circuit requests offered to the network, the competitive ratio provides a very robust performance measure.

Competitive analysis can be extended to randomized algorithms [BLS87]. Let $E[A(\sigma)]$ be the expected performance of randomized algorithm A on request sequence σ. Then the competitive ratio for A is the maximum over all request sequences σ of $E[A(\sigma)]/O(\sigma)$. This competitive ratio is called ***oblivious*** since the request sequence is chosen independently of the random choices made by A. In contrast, an ***adaptive*** competitive ratio permits the chosen sequence to depend on

the random choices of the algorithm [RS, BBK+90]. In this paper we will be concerned with oblivious competitive ratios. We note that all the lower bounds on the competitive ratio for deterministic algorithms mentioned in this paper are also lower bounds on the adaptive competitive ratio for randomized algorithms.

1.2 Previous Work

Admission control and virtual circuit routing problems have been previously considered in a variety of contexts. Garay and Gopal [GG92] and Garay, Gopal, Kutten, Mansour, and Yung [GGK+93] developed competitive algorithms for these problems in the scenario where preemption is allowed and the network is constrained to be a straight line. When preemption is allowed, the network may decide to terminate any virtual circuit at any time. Preemption is undesirable in most of the applications (such as video-servers and the sale of permanent virtual circuits) that we mentioned previously.

Awerbuch, Azar and Plotkin [AAP93] develop competitive algorithms for general networks, but with the restriction that every virtual circuit request at most $1/\log n$ of the capacity of the lowest capacity link. They provide an $O(\log nT)$ competitive algorithm, where n is the number of nodes in the network and T is the ratio of the longest to the shortest duration of any virtual circuit. If all virtual circuits have infinite duration, the competitive ratio of the algorithm in [AAP93] is $O(\log n)$. In many applications, the assumption that each virtual circuit uses only $1/\log 2n$ of the bandwidth of each link is unrealistic. For example, permanent virtual circuits may have connections that use $45Mbps$ of bandwidth which is fairly large compared to a typical backbone bandwidth of $155Mbps$. Oracle's Media Server has $7.5Mbps$ channels while an MPEG video stream uses $1.5Mbps$. In this case, the algorithm in [AAP93] can support at most a 16 node network.

Aspnes, Azar, Fiat, Plotkin and Waarts [AAF+93] consider a slightly different model. Here there is no admission control problem since all requests are accepted. [AAF+93] presents a competitive algorithm that on any link requires at most $O(\log n)$ more capacity than is required by the optimal off-line algorithm, where n is the number of nodes in the network. Note that the virtual circuits in [AAF+93] all have infinite duration. The result is extended to virtual circuit with finite duration in [AKP+93]. Both [AAF+93] and [AAP93] use minimum cost routing where the cost metric is an exponential function of the used bandwidth [SM90].

In [ABFR94] Awerbuch, Bartal, Fiat and Rosen consider the admission control and virtual circuit routing problem on trees. Their basic algorithm focuses on virtual circuits that request the entire bandwidth of a link and have infinite duration. The algorithm is randomized and has an $O(\log n)$ competitive ratio. For the line, they show a matching lower bound of $\Omega(\log n)$. By combining their basic algorithm with the algorithm in [AAP93], the authors provide an $O(\log^2 n)$ competitive algorithm for virtual circuits of arbitrary bandwidth. A previous, unpublished version of the Awerbuch et al paper [ABFR93] includes a non-greedy randomized admission control and virtual circuit routing algorithm whose analysis can be modified trivially to give an $O(\log d)$ competitive ratio on trees with radius d. This algorithm seems restricted to virtual circuits that request the entire bandwidth of a link. The basic algorithm in [ABFR94] can also be extended to deal with finite duration requests at the cost of an $O(\log T)$ multiplicitive term in the competitive ratio, where T is the ratio of the maximum to the minimum duration of a virtual circuit. Blum, Fiat, Karloff, and Rabani report a deterministic $O(n)$ algorithm for the $n \times n$ mesh, a deterministic $\Omega(\sqrt{n})$ lower bound on the $n \times n$ mesh, and an $O(\log n)$ deterministic algorithm with preemption for n node trees [BFKR93].

1.3 Our Results

In this paper, we improve upon and extend the work of [ABFR94, BFKR93]. For simplicity, we initially focus on the situation where each virtual circuit requests the entire link bandwidth. This is the situation where the $O(\log n)$ algorithm of [AAP93] fails. We also focus on the case that the duration is infinite. For several of our results, we show how these restrictions can be relaxed. In particular our results for trees hold for virtual circuits that use a fixed fraction of the link bandwidth. In general, the restriction to infinite durations can be relaxed using the techniques in [ABFR94] at the cost of an $O(\log T)$ multiplicitive term in the competitive ratio, where T is the ratio of the maximum duration to the minimum duration of a virtual circuit.

In the case of trees, we give tight (up to a constant factor) upper and lower bounds for all trees. For trees with radius d, we show bounds of $\Theta(\log d)$ on the competitive ratio of randomized algorithms for virtual circuit routing. If virtual circuits are requested between leaves only, we show tight bounds of $\Theta(\log d')$, where d' is the radius of the tree derived from the original network by shrinking degree-2 vertices. We show that these results hold even for trees with arbitrary

link capacity, as long as all links have the same capacity. Thus, we extend the unpublished $O(\log d)$ result of [ABFR93] to new domains. (Note, our algorithm uses different techniques than the $O(\log d)$ algorithm of [ABFR93].) Our randomized algorithms overcome the trivial $\Omega(d)$ lower bound on the competitive ratio for deterministic algorithms on trees.

For an $n \times n$ tree of meshes we develop a randomized algorithm that achieves an $O(\log \log n)$ competitive ratio. This matches the $\Omega(\log \log n)$ lower bound we give.

The algorithm for the $n \times n$ tree of meshes can be used to provide a randomized algorithm that achieves a competitive ratio of $O(\log n \log \log n)$ on an $n \times n$ mesh. Our algorithm overcomes the $\Omega(\sqrt{n})$ lower bound on the competitive ratio for deterministic algorithms on an $n \times n$ mesh [BFKR93]. We also show how to achieve a deterministic competitive ratio of $O(\log^2 n)$ for the hex, which is surprising given the lower bound of $\Omega(\sqrt{n})$ on the competitive ratio of an $n \times n$ mesh. Our competitive algorithms for the grid and hex are the first competitive algorithms for admission control and circuit routing on non-tree networks. (We also improve the upper bound on the deterministic competitive ratio for the mesh to $O(\sqrt{n})$, but we have omitted the details since the algorithm is not particularly useful.) In addition, we are able to show an $\Omega(\log n)$ lower bound on the competitive ratio for randomized algorithms for the $n \times n$ mesh.

Our technique for proving lower bounds on the competitive ratio for randomized algorithms extends naturally to several other networks. In particular, we observe an $\Omega(\frac{1}{k} \log n)$ lower bound for a k dimensional mesh with n nodes, an $\Omega(\log \log n)$ lower bound for an n^2-leaf tree of meshes, and an $\Omega(\log \log n)$ lower bound for the $\log n$ dimensional hypercube. Finally, we observe that there is an $O(1)$ competitive randomized algorithm for routing permutations on the $O(1)$-dilated hypercube, using a result of Aiello, Leighton, Maggs, and Newman [ALMN91].

The remainder of the paper is organized as follows. Our algorithms for trees are presented in Section 2. Section 3 presents our algorithm for an $n \times n$ tree of meshes. The algorithm for $n \times n$ meshes is presented in Section 4. All lower bounds are presented in Section 5.

2 An Optimal Algorithm for Trees

This section presents our algorithm for virtual circuit routing and admission control on trees. Let n be the number of nodes in the tree, let d be its diameter, and let d' be the diameter after shrinking degree-2 vertices. We describe in detail the algorithm for routing between leaves, then show how to extend it to the case of routing between any pairs of vertices. In what follows we assume that degree-2 vertices are already shrunk. If the remaining tree has a single edge, accept the first request that comes in. The optimal offline algorithm accepts at most one call. So, we may assume that the tree contains a non-leaf node. For convenience, root the tree at one of its non-leaf nodes, denoted r. This induces, for every vertex $v \neq r$, a unique parent, which we denote by $P(v)$. Also, for every pair of nodes u, v, denote their least common ancestor in the rooted tree by $LCA(u, v)$. Denote by $PATH(u, v)$ the unique path connecting u and v.

First, we introduce the concept of a *roadblock*. In response to a virtual circuit request, our algorithm may place roadblocks at edges of the tree. The existence of a roadblock on an edge blocks future requests whose paths use that edge, and causes them to be rejected.

Somewhat surprisingly, we will find that by denying some requests at random and by also denying all future requests that significantly overlap some denied requests (where the measure of significance is also random), we can significantly improve overall performance. In other words, by denying requests in a special randomized way, we will be able to accommodate more requests overall.

2.1 The Algorithm

Consider a particular virtual circuit request $\rho = \{u, v\}$. If $PATH(u, v)$ either crosses a roadblock or intersects with another virtual circuit request that was accepted, reject it. Otherwise, ρ is a *candidate*. If the request becomes a candidate, accept it with probability 1/2. Otherwise, reject it and place roadblocks as follows. Pick a random integer ℓ uniformly in $[1, \log d' + 1]$. Consider $PATH(u, v)$. Number the edges 1, 2, ... along the path from u to v (where we assume wlog that $u < v$). Place a roadblock on edges numbered $i2^\ell$ for all $0 < i \leq d'/2^{\ell-1}$. The roadblocks partition $PATH(u, v)$ into segments of equal length (except, perhaps, for the last segment). Also, place an extra roadblock on the edge between $LCA(u, v)$ and $P(LCA(u, v))$. We call this roadblock an *extra* roadblock.

2.2 The Analysis

Consider any request sequence $\sigma = \sigma_1\sigma_2\cdots\sigma_{|\sigma|}$. Denote $\sigma_i = \{u_i, v_i\}$. Define C^* to be the set of requests accepted by the optimal off-line strategy for this sequence. We call the requests in C^* the *optimal* requests. For the purpose of analyzing the algorithm we maintain a *feasible* subset $C \subseteq C^*$ that is updated each time a candidate request is handled. Let $C_0 = C^*$ be the initial value of C. Let C_i, $i = 1, 2, \ldots, |\sigma|$ be the value of the subset after the ith request is handled. For $C \subseteq C^*$ and a request ρ, let $C \cap \rho$ denote the set of calls in C whose path intersects the path of ρ at an edge. We also use *tokens* as a bookkeeping tool for the analysis. The algorithm is not aware of their existence. Tokens are distributed by candidate requests that are not accepted. A candidate request gives at most one token in each of the segments its path gets partitioned into by the roadblocks.

Next we explain how C is updated and how tokens are distributed. Consider a request σ_i. There are three cases:

Case 1. σ_i is not a candidate.
In this case $C_i = C_{i-1}$ and no tokens are distributed.

Case 2. σ_i is a candidate that is accepted.
Let $C := C_{i-1}$ and then remove from C the calls in $C_{i-1} \cap \sigma_i$. Furthermore, for any previously rejected candidate σ_j that σ_i intersects, consider the segment of σ_j's path in which the intersection occurs (there cannot be more than one segment, otherwise σ_i would pass across a roadblock). If there is a call in C_{i-1} that has tokens, intersects σ_j in the same segment, and received its last token from σ_j, we remove it from C. Now $C_i := C$. No tokens are distributed.

Case 3. σ_i is a candidate that is rejected.
In this case σ_i 's path is divided by the roadblocks into disjoint *segments* of equal length (except, perhaps the last segment). For each segment, we give a token to at most one optimal request in $C_{i-1} \cap \sigma_i$. The optimal requests that receive a token are determined as follows. For every $\rho \in C_{i-1} \cap \sigma_i$, ρ is of level j if it will not be blocked by roadblocks spaced 2^j apart, but will be blocked by roadblocks spaced 2^{j-1} apart on σ_i. If ℓ is the random spacing picked for σ_i, we give a token to every level ℓ optimal request in $C_{i-1} \cap \sigma_i$. Now define $C := C_i$ using the construction of C_i given for an accepted candidate request. Return to C the optimal requests that receive a token. If one of these requests is blocked by the extra roadblock, remove it from C. Now, set $C_i := C$.

We introduce some notation. Let $a^* = |C^*|$. Let a be the number of requests accepted by the algorithm. Let c be the number of candidates. Let t be the number of optimal requests that received tokens (notice that a call may receive more than one token, but we ignore multiple tokens). Let b be the number of optimal requests that were removed by extra roadblocks. Let f be the number of optimal requests that did not intersect any other candidate and were not even removed by extra roadblocks (notice that they must have been candidates themselves). Notice that a, c, t, b, and f are random variables By the definition of the algorithm

Fact 1 $E[a] = \frac{1}{2}E[c]$.

Lemma 2

$$E[t + b + f] \geq \frac{a^*}{2(\log d' + 1)}.$$

Proof. Consider any optimal request ρ. Consider the event that it was not removed by an extra roadblock, but it was overlapped by candidate requests. Consider the first candidate σ_i that overlapped ρ. Obviously, $\rho \in C_{i-1}$. With probability $1/2$, σ_i was accepted and ρ does not get any tokens. Otherwise, with probability $1/(\log d' + 1)$, the level picked is ρ's level, and it gets a token. In the complement event, ρ contributes 1 to the sum $b + f$. ■

Lemma 3

$$E[c] \geq \frac{1}{4}E[t].$$

Proof. Consider a candidate σ_i from u_i to v_i. We show that it removes at most 4 calls with tokens from C_{i-1}. σ_i might have tokens itself. The extra roadblock σ_i lays may remove one call with tokens. So, it is sufficient to show that at most two additional calls with tokens are removed. Let $x = LCA(u_i, v_i)$. We show that each of the two parts of σ_i, $PATH(u_i, x)$ and $PATH(v_i, x)$, can cause the removal of at most one call with tokens. Without loss of generality, consider $PATH(u_i, x)$. Notice that since $PATH(u_i, x)$ cannot cross any extra roadblock, all previous rejected candidates it intersects must share the topmost edge of $PATH(u_i, x)$, and therefore they all intersect among themselves in the same segments that $PATH(u_i, x)$ intersects with them. Therefore, there is at most one surviving (i.e., not yet removed from C) call whose last token was assigned from one of these segments, namely, the one (if any) from the latest rejected candidate encountered. We consider several cases.

Case 1. $PATH(u_i, x)$ does not intersect any call with tokens. The above argument shows that we remove at most one call with tokens.

Case 2. $PATH(u_i, x)$ intersects calls with tokens. Follow the path starting at u_i until it first intersects a call $\rho = \{u, v\}$ in C_{i-1} with tokens. There are two possibilities:

Case 2a. The remainder of $PATH(u_i, x)$ is contained in $PATH(u, v)$. In this case, since calls with tokens belong to the optimal set C^*, we cannot intersect any other call with tokens. Furthermore, either we have not encountered any rejected candidates, or ρ must be the call with tokens from the latest rejected candidate encountered, since the topmost edge of $PATH(u_i, x)$ is contained in ρ.

Case 2b. $PATH(u_i, x)$ extends beyond $w = LCA(u, v)$. The previous case shows that the portion $PATH(u_i, w)$ does not remove any other calls with tokens. Since the edge $e = (w, P(w))$ must be contained in the rejected candidate σ_j that gave ρ its last token (otherwise ρ or σ_i would have been blocked by the extra roadblock), the entire portion $PATH(w, x)$ is contained in σ_j's path. Since σ_j would have removed any other optimal call intersecting the segment containing both $PATH(w, x)$ and the intersection of σ_j with ρ (notice that e cannot delimit a segment), $PATH(w, x)$ cannot intersect any other non-removed call with tokens. It is clear that σ_j is the latest rejected candidate encountered, since all other rejected candidates encountered intersect σ_j in the same segment that contains ρ, and if anyone would have come later, it either would have removed ρ or would have placed a token on ρ. ■

Theorem 4 *The above algorithm achieves a competitive ratio of* $O(\log d')$.

Proof. Obviously, $c \geq b$ and $c \geq f$. Combining this with Fact 1 and Lemmas 2 and 3 we get:

$$E[a] = \frac{1}{2}E[c] \geq \frac{1}{2}\max\left\{\frac{1}{4}E[t], E[b], E[f]\right\} \geq \frac{1}{24}E[t+b+f] \geq \frac{a^*}{48(\log d' + 1)}.$$

In order to handle requests between interior nodes, we reduce the problem to the special case of requests between leaves. Given a tree of diameter d, consider every non-leaf node v. Let $\{e_1, e_2, \ldots, e_k\}$ be the set of edges adjacent to v. For every edge e_j in that set, add a new leaf v_j connected to v. This does not increase the diameter of the tree. Consider a request sequence. In any request that has an interior node v as one of its endpoints, find the edge e_j adjacent to v through which the path of this request must go, and replace v in that request by v_j.

The algorithm can also be extended to handle virtual circuits that request less than the entire bandwidth of a link. Let the capacity of each link in the tree be k. In other words, k virtual circuits can simultaneously use any link. The algorithm views such a tree as k virtual trees, $T_1, \ldots, T_k$, where each link in each of the virtual trees has capacity one. When a request ρ arrives, find a tree T_i such that ρ is a candidate in T_i. If no such T_i exists, reject ρ. Otherwise, proceed with ρ on T_i as in the algorithm for capacity one trees. A constant fraction of the optimal requests can be accepted on these virtual trees. (To see that, notice that the optimal requests can be colored with $2k$ colors so that no two overlapping requests have the same color. There is a subset of k colors that contains at least half of the optimal requests.) With this assignment of optimal requests to virtual trees, the analysis proceeds essentially the same as the analysis for the capacity one tree.

3 Algorithm for Tree of Meshes

For n a power of 2, the $n \times n$ tree of meshes has the following structure. The network has a total of $2n^2 \log n$ nodes. These are arranged in $2\log n$ levels, each containing n^2 nodes. The levels are numbered 0 through $2\log n - 1$. Links connect nodes in the same level or in adjacent levels. Level i is a collection of disjoint $m_i \times n_i$ meshes, where $m_i = \frac{n}{2^i}$ and $n_i = \frac{n}{2^{2\lfloor i/2 \rfloor}}$. Notice that for even i $m_i = n_i = m_{i-1}$, and for odd i $2m_i = n_i = n_{i-1}$. Each level i mesh is connected to a unique pair of level $i+1$ meshes. For even i, nodes $(1,1)$, $(1,2)$, ..., $(1, n_i)$ of the level i mesh are connected to nodes $(1,1)$, $(1,2)$, ..., $(1, n_{i+1})$, respectively, of one of the level $i+1$ meshes, and nodes $(m_i, 1)$, $(m_i, 2)$, ..., (m_i, n_i) of the level i mesh are connected to nodes $(m_{i+1}, 1)$, $(m_{i+1}, 2)$, ..., (m_{i+1}, n_{i+1}), respectively, of the other level $i+1$ mesh. For odd i, nodes $(1,1)$, $(2,1)$, ..., $(m_i, 1)$ of the level i mesh are connected to nodes $(1,1)$, $(2,1)$, ..., $(m_{i+1}, 1)$, respectively, of one of the level $i+1$ meshes, and nodes $(1, n_i)$, $(2, n_i)$, ..., (m_i, n_i) of the level i mesh are connected to nodes $(1, n_{i+1})$, $(2, n_{i+1})$, ..., (m_{i+1}, n_{i+1}), respectively, of the other level $i+1$ mesh.

Essentially, on-line routing and admission control of requests between leaves (level $2\log n - 1$ nodes) of the $n \times n$ tree of meshes is equivalent, up to constant factors in the competitive ratio, to routing on the following binary tree. Take a complete binary tree of height $2\log n - 1$ (the tree has n^2 leaves). The root (level 0 node) is connected to each of its children by

an edge with capacity n. In general, level i nodes are connected to each of their children by an edge of capacity $n/2^{i/2}$, if i is even, or by an edge of capacity $n/2^{(i-1)/2}$, if i is odd. The idea is that routes taken on this tree can be duplicated on the tree of meshes using the intermediate meshes as crossbars.

Viewing the routing problem on the tree of meshes this way, it is easy to see that a deterministic greedy algorithm is $O(\log n)$ competitive. This is because the path of any accepted request is $O(\log n)$ long, and therefore may block at most $O(\log n)$ optimal requests. Our algorithm improves over the greedy algorithm by achieving an $O(\log\log n)$ competitive ratio. We show a matching lower bound in Section 5.

The description of our algorithm also views the tree of meshes as the above detailed binary tree. We divide requests into *local* and *long distance* requests. For a local request, the height of the lowest common ancestor must not be greater than $4\log\log\log n$ (the factor of 4 is an overshoot). All requests that are not local are long distance requests. Notice that the bandwidth for all links above a node of height $4\log\log\log n$ is at least $(\log\log n)^2$. View the subtrees rooted at nodes of height $4\log\log\log n$ as virtual nodes and call the tree whose leaves are these virtual nodes the *long distance tree*. We will use the algorithm in [AAP93] to route long distance requests in the long distance tree. To see that [AAP93] is applicable to our situation, we note that the analysis in [AAP93] can be modified easily to show that their algorithm achieves a competitive ratio of $O(\log d)$ for connections which use at most a $1/\log 2d$ fraction of the bandwidth, where d is the length of the longest simple path in the network. For our long distance network $d \leq 2\log n$. Our algorithm proceeds as follows. Flip an unbiased coin. With probability 1/2 route only local requests using greedy admission control. With the remaining probability route only long distance requests. Upon arrival of a long distance request, check if it is blocked in either of the two virtual nodes that contain its end points (in which case it has to be rejected). If not, use the algorithm in [AAP93] to do the admission control in the long distance tree.

Analysis.
Consider any request sequence. Let a^* be the number of optimal requests; i.e., the number of requests the optimal off-line algorithm accepts. Let a_s^* be the number of local optimal requests, and let a_ℓ^* be the number of long distance optimal requests. Define the following random variables. Let a be the number of requests accepted by our algorithm, let a_s be the number of accepted local requests, and let a_ℓ be the number of accepted long distance requests. Finally, let a'_ℓ be the number of long distance optimal requests that the algorithm feeds to the [AAP93] algorithm. (Notice that with probability 1/2, $a'_\ell = 0$.)

Since local requests have paths of length at most $8\log\log\log n$, each such request that is routed may block at most that many optimal local requests. We route local requests with probability 1/2. Therefore we get:

$$E[a_s] \geq \frac{a_s^*}{O(\log\log\log n)}. \tag{1}$$

With a simple modification of the proof in [AAP93], we show that:

$$a_\ell \geq \frac{a'_\ell}{O(\log\log n)}. \tag{2}$$

Consider the sum $O(\log\log\log n)a_\ell + a'_\ell$. With probability 1/2 this sum is 0. Otherwise, it is at least a_ℓ^*, because for each accepted long distance call, the portions of its path contained in virtual nodes are of total length $O(\log\log\log n)$ and therefore may block at most that many optimal long distance calls (before they are fed to the [AAP93] algorithm). Therefore, we get:

$$E[O(\log\log\log n)a_\ell + a'_\ell] \geq \frac{a_\ell^*}{2}. \tag{3}$$

Therefore,

Theorem 5 *The above algorithm has a competitive ratio of* $O(\log\log n)$.

Proof. Using 1, 2, and 3,

$$\begin{aligned} a^* &= a_\ell^* + a_s^* \\ &\leq 2E[O(\log\log\log n)a_\ell + a'_\ell] + O(\log\log\log n)E[a_s] \\ &\leq O(\log\log n)E[a_\ell] + O(\log\log\log n)E[a_s] \\ &\leq O(\log\log n)E[a_\ell + a_s] \\ &= O(\log\log n)E[a]. \end{aligned}$$ ■

4 Algorithms for Meshes

The two dimensional $m \times n$ mesh has the set of nodes $\{(r,c) \mid 1 \leq r \leq m,\ 1 \leq c \leq n\}$. Two nodes are connected by a link if their Hamming distance is one. Where c is some constant, it is easy to embed an $m \times n$ hex on a $cm \times cn$ mesh without reducing the optimal throughput by replacing each node of the hex with a $c \times c$ mesh that is used as a crossbar to route all requests that pass through that node.

We use the tree of meshes to derive an $O(\log n \log\log n)$ competitive randomized algorithm

for the mesh. Our algorithm for the mesh is based on a 1–1 embedding of the nodes of the $n \times n$ tree of meshes onto nodes of the $n \log n \times n \log n$ hex, such that the maximum congestion at an edge is 1 [BL84].

To simplify the presentation, we consider a $Kn \log n \times Kn \log n$ mesh, where K is some sufficiently large constant. Divide the mesh into n^2 disjoint $K \log n \times K \log n$ squares. Divide each square into $\log^2 n$ disjoint $K \times K$ squares. In each of these $n^2 \log^2 n$ constant size squares, pick independently and uniformly at random one vertex. Each such vertex is an *active* vertex. Every active vertex *controls* the $K \times K$ square it was picked from.

Now, in each $K \log n \times K \log n$ square, we place an H-embedding of a complete binary tree with $\log^2 n$ leaves. We can now connect every active vertex to such a binary tree by a path that is contained in the square controlled by that active vertex. (That is because every $K \times K$ square contains a leaf of the tree.) We now embed an $n \times n$ tree of meshes in our mesh. Each $K \log n \times K \log n$ square contains one leaf of the tree of meshes. We connect the root of the complete binary tree embedded in such a square to the leaf of the tree of meshes in the same square, using a path that does not leave the square. Notice that we may have used some of the edges of the mesh more than once. However, if K is large enough, we can move wires around so that every edge gets used at most once.

Now for the algorithm. Flip an unbiased coin. With probability 1/2 consider only requests that have both endpoints within the same $K \log n \times K \log n$ square. We name these requests *local calls*. We use a deterministic algorithm routing those requests. Specifically, let the *row-column* path be the path that travels in the row of the source until it reaches the column of the destination and then travels in the column of the destination. A local call is accepted if its row-column path is available. With the remaining probability, i.e. 1/2, consider only requests that have each endpoint in a different square. We name these requests *long distance calls*. Use the following method for these calls. When a request comes in, reject it if either of its two endpoints u, v is not an active vertex. Otherwise, it is considered a *candidate*. A candidate call uses the tree of meshes algorithm of Section 3 to determine a path from u's $K \log n \times K \log n$ square to v's $K \log n \times K \log n$ square in the embedded tree of meshes. If the tree of meshes algorithm rejects the call, the call is rejected for the mesh. If the tree of meshes algorithm accepts the call, take the following path: from u to the binary tree in its square, then along the binary tree to its root, from there to a leaf of the tree of meshes, from there to the leaf of the tree of meshes in the square containing v, from there to the root of the binary tree in that square, from there along the binary tree to where v connects to the tree, from there to v. Note that the path from the endpoints to leaves of the tree of meshes can never be blocked since each leaf in the tree of meshes can accept at most one call.

Analysis.

Consider any request sequence. Let a^* be the number of optimal requests, i.e. the number of requests the optimal off-line algorithm accepts. Let a_s^* be the number of local optimal requests, and let a_ℓ^* be the number of long distance optimal requests. We now define some random variables. Let a be the number of calls our algorithm accepts. Let a_s be the number of local calls our algorithm accepts, and let a_ℓ be the number of long distance calls our algorithm accepts. Let c^* be the number of optimal long distance requests that are candidates.

Lemma 6 $E[c^*] = \frac{a_\ell^*}{2K^4}$.

Proof. With probability 1/2 we choose to route only local calls and there are no candidates in that case. Otherwise, for every optimal request, each of its two endpoints has a probability of $1/K^2$ to be an active vertex. These events for the two end points are independent. ■

Lemma 7 $E[a_\ell] \geq \frac{c^*}{O(\log n \log\log n)}$, *where the expectation is over the coin tosses of the tree of meshes algorithm.*

Proof Sketch. Let c' be the set of candidate calls that would be accepted by an optimal off-line algorithm for the embedded tree of meshes. $c^* \leq O(\log n)c'$. (To see that, notice that by increasing the capacity of the tree of meshes by a factor of $O(\log n)$, all the c^* optimal long distance calls that are candidates can be routed on the tree [LS93]. Now, in general, reducing the capacity of a network by a certain factor may reduce the throughput by much more than that factor, but this nasty behavior does not happen in trees, as follows, e.g., from [GVY93].) Furthermore, from Theorem 5 we know that $c' \leq O(\log\log n)E[a_\ell]$. The lemma follows. ■

Lemma 8 $E[a_s] \geq \frac{a_s^*}{4K \log n}$.

Proof. Any row-column path taken by a local call blocks at most the calls whose endpoints are in the same row and column as the row-column path. Thus,

it blocks at most $2K \log n$ optimal calls. The result follows from the fact that we accept local calls with probability 1/2. ■

Theorem 9 *The above algorithm achieves a competitive ratio in* $O(\log n \log\log n)$.

Proof. Using Lemmas 6 and 7, we have

$$E[a_\ell] \geq \frac{E[c^*]}{O(\log n \log\log n)} = \frac{a^*_\ell}{O(\log n \log\log n)}.$$

Together with Lemma 8 we get

$$E[a] = E[a_s + a_\ell] \geq \frac{a^*_s}{4K \log n} + \frac{a^*_\ell}{O(\log n \log\log n)} \geq \frac{a^*}{O(\log n \log\log n)}.$$

In section 5 we prove an $\Omega(\log n)$ lower bound on the competitive ratio for any randomized algorithm on the $n \times n$ mesh.

5 Lower Bounds

In this section we give the lower bound for the mesh, the tree, the tree of meshes, and the hypercube. These lower bounds share a common proof technique. We give the details of the proof for meshes, and sketch the remaining proofs.

We prove an $\Omega(\log n)$ lower bound on the competitive ratio for any randomized algorithm on the $n \times n$ mesh. Using the principle of von Neumann (see [Yao77, BLS87]), it suffices to demonstrate a probability distribution over inputs that forces an $\Omega(\log n)$ lower bound on the competitive ratio of any deterministic algorithm A.

Consider an $n \times n$ mesh. For simplicity we assume that n is a power of 2. For every $0 \leq i < \log n$, we divide the mesh into 4^i submeshes of size $n/2^i \times n/2^i$. For example, the top left hand submesh of the ith division consists of the nodes:

$$\begin{matrix} (1,1) & \dots & (n/2^i, 1) \\ \vdots & & \vdots \\ (1, n/2^i) & \dots & (n/2^i, n/2^i) \end{matrix}$$

Denote by $\mathcal{L}_i$ the following requests. For each of the 4^i squares of size $n/2^i \times n/2^i$ request circuits from all top nodes to all bottom nodes, e.g. $\{(1,1),(1,n/2^i)\},\{(2,1),(2,n/2^i)\}$ $\dots\{(n/2^i,1),(n/2^i,n/2^i)\}$ for the top left hand square, and circuits from all left hand nodes to all right hand nodes, e.g. $\{(1,1),(n/2^i,1)\},\{(1,2),(n/2^i,2)\}$ $\dots\{(1,n/2^i),(n/2^i,n/2^i)\}$ for the top left hand square. Note that routing all requests in $\mathcal{L}_i$ would use all links in the mesh.

We generate a random request sequence as follows. With probability n^{-1}, we give no requests. Otherwise, the sequence consists of all requests in $\mathcal{L}_0, \dots, \mathcal{L}_X$, where $0 \leq X < \log n$ is an integer random variable which satisfies $\Pr[X = i] = \frac{1}{2}\Pr[X = i-1]$, for every $1 \leq i < \log n$. In other words, for $0 \leq i < \log n - 1$, the probability of the sequence $\mathcal{L}_0 \dots \mathcal{L}_i$ is 2^{-i-1}.

Consider an off-line strategy that accepts all the requests in $\mathcal{L}_i$ given that the request sequence is $\mathcal{L}_0 \dots \mathcal{L}_i$. The number of connections in $\mathcal{L}_i$ is $4^i 2n/2^i = 2^{i+1}n$. Furthermore, the probability that the request sequence generated terminates with the requests in $\mathcal{L}_i$ is 2^{-i-1}. Thus, the expected number of requests routed by this strategy is bounded by

$$\sum_{i=0}^{\log n - 1} 2^{-i-1} 2^{i+1} n = n \log n. \tag{4}$$

Next we bound the expected number of connections a deterministic on-line algorithm A can accept. Notice that any route for any call from $\mathcal{L}_i$ consumes at least $n/2^i - 1$ links. Therefore, if k links remain unused by calls accepted from $\mathcal{L}_0, \dots, \mathcal{L}_{i-1}$, and requests from $\mathcal{L}_i$ arrive, we can hope to accept at most $k/(n/2^i - 1)$ of those calls. Conditioning upon $X \geq i$, denote by $A(i,k)$ the maximum expected number of calls accepted from $\mathcal{L}_i, \mathcal{L}_{i+1}, \dots$, where the maximum is taken over all possible ways to accept calls from $\mathcal{L}_0, \dots, \mathcal{L}_{i-1}$ so that at most k links are free. We have the following recurrence relation.

$$A(i,k) \leq \max_{\ell \leq k/(n/2^i - 1)} \left\{ \ell + \frac{1}{2} A(i+1, k - \ell(n/2^i - 1)) \right\},$$

with the initial condition

$$A(\log n - 1, k) \leq k.$$

Clearly, $A(0, 2n(n-1))$ is an upper bound on the expected number of calls that the on-line algorithm accepts.

Consider the following claim.

Claim 10 *For all* i, k, $A(i,k) \leq k/(n/2^i - 1)$.

Proof. The proof is by induction on $j = \log n - i$. The base case ($i = \log n - 1$) follows immediately from

the initial condition on A. Now, assume that for all k, $A(i+1,k) \leq k/(n/2^{i+1}-1)$. We have that

$$\begin{aligned} A(i,k) &\leq \max_{\ell \leq k/(n/2^i-1)} \left\{ \ell + \frac{1}{2} A(i+1, k-\ell(n/2^i-1)) \right\} \\ &\leq \max_{\ell \leq k/(n/2^i-1)} \left\{ \ell + \frac{k-\ell(n/2^i-1)}{2(n/2^{i+1}-1)} \right\} \\ &\leq \frac{k}{n/2^i-1}. \quad \blacksquare \end{aligned}$$

As a corollary we get the following theorem.

Theorem 11 *Any randomized on-line algorithm for routing and admission control on $n \times n$ meshes has a competitive ratio in $\Omega(\log n)$.*

Proof. By claim 10, $A(0, 2n(n-1)) \leq 2n$. Together with Equation 4, the theorem follows. ■

The lower bounds for trees, for the hypercube, and for the tree of meshes are derived by an argument analogous to the one used for the mesh. We use the following structure to describe the proofs for each of these networks. To show a lower bound of $\Omega(\log D)$, we define $\log D$ sets of requests $\mathcal{L}_0, \mathcal{L}_1, \ldots, \mathcal{L}_{\log D-1}$. These sets have the following properties.

1. For each set, accepting as many calls as possible from that set blocks the entire network (or a portion of it that cannot be bypassed).
2. Any call that is accepted from set $\mathcal{L}_i$ reduces the number of calls that can be accepted from $\mathcal{L}_{i+1}$ by 2, from $\mathcal{L}_{i+2}$ by 4, etc.

We then generate a probability distribution over request sequences, by requesting all requests in $\mathcal{L}_0 \cup \mathcal{L}_1 \cup \cdots \cup \mathcal{L}_X$, where $0 \leq X < \log D$ is an integer random variable which satisfies $\Pr[X = i] = \frac{1}{2}\Pr[X = i-1]$, for every $1 \leq i < \log D$. The specific bound is then proven as it is for the mesh. For each network, we sketch the proof by giving the definition of $\mathcal{L}_0, \mathcal{L}_1, \ldots, \mathcal{L}_{\log D-1}$ for that network.

Trees. For simplicity assume that the diameter d of the tree is a power of 2. Take a path between two leaves u, v whose length equals the diameter of the tree. $\mathcal{L}_0$ consists of the single request $\{u, v\}$. To construct $\mathcal{L}_1$, divide $PATH(u, v)$ into two equal length paths and request each of the two. To construct $\mathcal{L}_i$, divide the path of each request in $\mathcal{L}_{i-1}$ into two equal length paths, and request all those paths. Using the above arguments gives a lower bound of $\Omega(\log d)$ in this case. A slight modification of this idea gives the $\Omega(\log d')$ bound for requests between leaves. (Note that this argument also provides an alternative proof of the $\Omega(\log n)$ lower bound of [ABFR94] for admission control on an n node line.)

The hypercube. For simplicity, let n be an integer such that $\log n$ is a power of 2. Consider the $\log n$ dimensional hypercube with node set $\{0,1\}^{\log n}$ and edges connecting every pair of nodes with Hamming distance 1. $\mathcal{L}_0$ consists of requests whose endpoints are bitwise complements of each other. The resulting paths have length $\log n$. Assume that each request uses a path that complements bits from left to right. We wish to specify a set of requests so that the resulting paths use exactly all of the edges in the hypercube. This is the first of the two above mentioned properties for $\mathcal{L}_0, \mathcal{L}_1, \ldots, \mathcal{L}_{\log D-1}$. We call the property $P1$.

Consider the set of requests between every node and its bitwise complement. That is, the pairs $000\cdots00$ and $111\cdots11$, $000\cdots01$ and $111\cdots10$, and so forth. (Notice that every pair is requested twice.) We show inductively that this request set can be partitioned into two equal sized sets, each satisfying $P1$. For the two dimensional hypercube, the two requests between 00 and 11 represent one such set and the two requests between 01 and 10 the other set.

Now consider a $\log n$-dimensional hypercube. Consider the set of requests whose paths originate with a node that has a left bit of 0. The paths do not intersect on the first edge and the remaining edges of each path are all in the $(\log n - 1)$-dimensional hypercube defined by the nodes with a left bit of 1. By the induction hypothesis, these paths can be partitioned into two equal subsets such that each satisfies $P1$ in $(\log n - 1)$-dimensional hypercube and uses half of the edges that connect the $(\log n - 1)$-dimensional hypercube to the $(\log n - 1)$-dimensional hypercube defined by the nodes with a left bit of 0. Pick one of the subsets. Call it S_1. Now, consider the set of requests whose paths originate with a node that has a left bit of 1 and where the first edge is not already used by the path for a request in S_1. Call it S_2. It is easy to see that the set of requests in S_2 cover the remaining edges in the hypercube. Thus, $S_1 \cup S_2$ satisfies $P1$. Furthermore, $S_1 \cup S_2$ consists of exactly half of the requests between every node and its bitwise complement. The set of requests not in $S_1 \cup S_2$ together also satisfy $P1$.

Now, let $\mathcal{L}_0$ consist of the requests in $S_1 \cup S_2$. In order to construct $\mathcal{L}_1$, divide each of the paths suggested for the $\mathcal{L}_0$ requests into two equal length paths, then request the two endpoints of each of these shorter paths. We can use the shorter paths to route all requests in $\mathcal{L}_1$. In general, to construct $\mathcal{L}_i$ take the paths

suggested for routing the requests in $\mathcal{L}_{i-1}$, divide each of them into two equal length paths and request the two endpoints of each of these paths. Use the partitioned paths to route all requests in $\mathcal{L}_i$. The lower bound we get is $\Omega(\log \log n)$.

The tree of meshes. Consider the simplified, weighted binary tree of height $2 \log n - 1$ representation of the $n \times n$ tree of meshes. Each request we give is routed from a leaf to the root of a subtree, from there to a leaf in the other branch of that subtree. In order to specify our sets of requests, we specify leaf-to-root paths. The other half of each request's path is implied by symmetry. In each level of the tree, number the edges connecting that level's nodes to their children from left to right starting at 0. We refer to the level of an edge as the minimum level of its two endpoints. Our $\mathcal{L}_0$ requests all reach the root of the tree. There are n of them. They congest edge 0 in level 0, edges 0 and 2 in level 2, edges 0, 2, 8, and 10 in level 4, and in general, if edge j of level $2i$ is congested, so are edges $4j$ and $4j+2$ in level $2(i+1)$. Notice that every path participates in the congestion of one edge in every even level.

In order to construct $\mathcal{L}_1$, divide each path into two equal length parts, removing the middle edge. We want to place, for every path in $\mathcal{L}_0$, two paths in $\mathcal{L}_1$, one overlapping the bottom part of the $\mathcal{L}_0$ path and the other overlapping the top part. The bottom part paths are easy to construct, since they contain a leaf. To construct the top part paths, we take a look at the $\log n$ high subtrees that contain $\mathcal{L}_0$ congested edges. The edge connecting the root to its left child is congested by $\mathcal{L}_0$ paths in each subtree. We use the subtree rooted at the right child to connect all our top part paths to leafs so that there will be no overlap with bottom part paths. In general, we use similar ideas to construct the set $\mathcal{L}_i$ by dividing $\mathcal{L}_0$ paths into 2^i parts (removing some of the edges). We leave the formal details to the full version of the paper. This gives a lower bound of $\Omega(\log \log n)$ for the $n \times n$ tree of meshes.

References

[ALMN91] B. Aiello, T. Leighton, B. Maggs, M. Newman, Fast algorithms for bit-serial routing on a hypercube. *Mathematical Systems Theory*, 24(6):253–271, April 1991.

[AAF+93] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. In *Proc. of the 25rd Ann. ACM Symp. on Theory of Computing*, San Diego, May 1993.

[AAP93] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput competitive on-line routing. In *Proc. of the 34th Ann. Symp. on Foundations of Computer Science*, Palo Alto, November 1993.

[ABFR93] B. Awerbuch, Y. Bartal, A. Fiat, and A. Rosén. Competitive non-preemptive call control. Unpublished manuscript, 1993.

[ABFR94] B. Awerbuch, Y. Bartal, A. Fiat, and A. Rosén. Competitive non-preemptive call control. In *Proc. of the 5th Ann. ACM-SIAM Symp. on Discrete Algorithms*, 1994.

[AKP+93] Y. Azar, B. Kalyanasundaram, S. Plotkin, K. Pruhs, and O. Waarts. On-line load balancing of temporary tasks. In *Proc. of Workshop on Algorithms and Data Structures*, pages 119–130, August 1993.

[BBK+90] S. Ben-David, A. Borodin, R.M. Karp, G. Tardos, and A. Wigderson. On the Power of Randomization in On-Line Algorithms. In *Proc. of the 22nd Ann. ACM Symp. on Theory of Computing*, pages 379–386, May 1990.

[BL84] S. Bhatt and T. Leighton A framework for solving VLSI graph layout problems. In *Journal of Computer Science*, 28(2):300-334, April 1984.

[BFKR93] A. Blum, A. Fiat, H. Karloff, and Y. Rabani. Private communication, 1993.

[BLS87] A. Borodin, N. Linial, and M. Saks. An Optimal On-Line Algorithm for Metrical Task Systems. In *Proc. of the 19th Ann. ACM Symp. on Theory of Computing*, pages 373–382, May 1987.

[Buck94] R. Buck. The Oracle media server for nCUBE massively parallel systems. In *Proc. of the 8th International Parallel Processing Symp.*, Cancún, Mexico, pages 670–673, April 1994.

[deP91] M. de Prycker. *Asynchronous Transfer Mode, Solutions for Broadband ISDN.* Ellis Horwood, England, 1991.

[GG92] J.A. Garay and I.S. Gopal. Call preemption in communication networks. In *Proc. of INFOCOM '92*, volume 44, pages 1043–1050, Florence, Italy, 1992.

[GGK+93] J. Garay, I. Gopal, S. Kutten, Y. Mansour, and M. Yung. Efficient on-line call control algorithms. In *Proc. of 2nd Ann. Israel Conference on Theory of Computing and Systems*, 1993.

[GVY93] N. Garg, V.V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees, with applications to matching and set cover. In *Proc. of ICALP '93*.

[GKR94] R. Gawlick, C. Kamanek, and K.G. Ramakrishnan. On-line routing for virtual private networks. Unpublished manuscript, February 1994.

[KMRS88] A. Karlin, M. Manasse, L. Rudolpoh, and D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):79–119, 1988.

[Lei84] T. Leighton. New lower bound techniques for VLSI. *Mathematical Systems Theory*, 17(1):47–70, April 1984.

[LS93] T. Leighton and A. Shrivastava. *Theory of parallel and VLSI computation.* Lecture notes, MIT/LCS/RSS-24, September 1993.

[Lei93] C. Leiserson. Private communication, 1993.

[RS] P. Raghavan and M. Snir. Memory versus Randomization in On-line Algorithms. In *Proc. of ICALP '89*.

[SM90] F. Shahrokhi and D. Matula. The maximum concurrent flow problem. *Journal of the ACM*, 37:318 – 334, 1990.

[ST85] D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[Yao77] A. Yao. Probabilistic Computations: Towards a Unified Measure of Complexity. In *Proc. of the 18th Ann. IEEE Symp. on Foundations of Computer Science*, pages 222–227, October 1977.