Scheduling Multicasts on Unit-Capacity Trees and Meshes

Monika R. Henzinger * Stefano Leonardi †

Abstract

This paper studies the multicast routing and admission control problem on unit-capacity tree and mesh topologies in the throughput-model. The problem is a generalization of the edge-disjoint paths problem and is *NP-hard* both on trees and meshes.

We study both the offline and the online version of the problem: In the offline setting, we give the first constant-factor approximation algorithm for trees, and an $O((\log\log n)^2)$-factor approximation algorithm for meshes, where n is the number of nodes in the graph.

In the online setting, we give the first polylogarithmic competitive online algorithm for tree and mesh topologies. No polylogarithmic-competitive algorithm is possible on general network topologies [8] and there exists a polylogarithmic lower bound on the competitive ratio of any online algorithm on tree topologies [1]. We prove the same lower bound for meshes.

1 Introduction.

Multicast routing and admission control are the basic operations required by future high-speed communication networks that use bandwidth-reservation for quality-of-service guarantees. A number of applications from collective communication to data distribution will be based on efficient multicast communication.

Formally, the *multicast routing and admission control problem* with $\mathcal{M}$ multicasts consists of an n-node graph G and a sequence or set of requests (t, s_i), where the *request node* t and the *source node* s_i are nodes in G and $i \in \{1, 2, \ldots, \mathcal{M}\}$. Multicast i consists of all requests with source s_i. For each request the algorithm has to decide whether to accept or reject it. If request (t, s_i) is accepted, the algorithm has to connect node t to the *multicast tree* connecting the already accepted requests of multicast i with source s_i. In the *unit-capacity* setting, each link can be assigned to only one multicast tree: the trees spanning different multicasts must be edge-disjoint. The objective function is to maximize the total number of accepted requests. In the *online* version the requests form a sequence and when processing a request, the algorithm must decide without knowledge of future requests. In the *offline* version the requests form a set, which is given before the algorithm decides which requests to accept.

Online multicast routing was recently studied under the *small bandwidth assumption* that the link bandwidth required by every connection is at most a fraction logarithmic in the size of the network. Awerbuch and Singh [6] gave an $O(\log n(\log n + \log\log \mathcal{M})\log \mathcal{M})$-competitive algorithm for the case in which all the requests to a given multicast arrive before the next multicast is created. Goel, Henzinger, and Plotkin [10] extended the study to the case in which requests to different multicasts can be interleaved.

With the sizes of networks growing faster than the link capacity, the small bandwidth request assumption is not always a realistic assumption. There are various applications, e.g. a multimedia server managed by a supercomputer, in which large amount of data must be transferred in a local network where a single communication path consumes a large fraction of the available bandwidth on a link [4]. Thus, the situation where the bandwidth required by a connection is a large fraction of the link capacity needs to be studied as well for the multicast routing problem. In this paper we take a first step into this direction by assuming that every connection uses the total bandwidth on a link. We call this the *unit-capacity* case.

We study both the offline and the online version of the multicast routing and admission control problem in unit-capacity graphs. The offline problem models the case of arrival of a batch of connection requests to several multicasts. It is also motivated by all those situations where the answer to the user can be delayed for a limited time while other requests are collected.

We present algorithms for tree and mesh topologies, which are at the basis of many communication networks. Trees are important practical network topologies [3, 4, 20, 17], they are at the basis of topologies for

*Compaq Systems Research Center, 130 Lytton Ave, Palo Alto CA 94301. Email: monika@pa.dec.com

†Dipartimento di Informatica Sistemistica, Università di Roma "La Sapienza", via Salaria 113, 00198-Roma, Italia. This work was partly done while the author was visiting the Max-Planck-Institute für Informatik, Saarbrücken, Germany. This work was partly supported by EU ESPRIT Long term Research Project ALCOM-IT under contract n. 20244, and by Italian Ministry of Scientific Research Project 40% "Algoritmi, Modelli di Calcolo e Strutture Informative". Email: leon@dis.uniroma1.it

communication networks such as trees of rings, often considered as interconnection of SONET rings optical networks [20, 17], or topologies for connecting high performance multicomputers systems as trees of meshes [4] and fat trees. The multicast routing problem on trees, when all the multicast groups use the same spanning tree, is then a basic problem to solve in this context. There has also been an extensive study of the unicast problem on these network topologies motivated by virtual circuit assignment and optical communication. Meshes topologies are often the basis of the interconnecting topology of high performance multiprocessor systems. They are also relevant as a first approximation of nearly-planar communication networks [14]. The offline problem on meshes arises also in FPGA-routing, where various subsets of components have to be connected by trees such that the trees of different subsets do not overlap and the underlying routing fabric is a mesh. The unicast problem for meshes was recently studied in both the offline and the online version (see e.g. [4, 14, 19, 16]).

The multicast routing problem in unit-capacity graphs reduces to the *edge-disjoint paths problem* if only one request is presented for each multicast, called the *unicast* setting. Multicast routing is also an interesting extension of the maximum coverage problem [13].

Previous work on unit-capacity networks. All previous work on unit-capacity networks studied unicast routing. Unlike multicast routing, the offline unicast problem is still polynomial on trees [12], but it is *NP-hard* on meshes. Kleinberg and Tardos [14] proposed the first constant approximation algorithm for edge-disjoint paths on meshes and on a class of planar graphs called "densely embedded, nearly-Eulerian graphs".

For the online problem no algorithm, not even a randomized one, has a polylogarithmic competitive ratio for any network topology [8]. Deterministic algorithms for the unicast problem have a very high lower bound even for line networks [2]. (This clearly extends also to the multicast problem.) Therefore in the unicast setting randomized algorithms for restricted graph topologies like trees, meshes, and "densely embedded, nearly-Eulerian graphs" [3, 4, 14, 16] were studied before and algorithms with logarithmic competitive ratio were proposed for all these network topologies.

Our offline results. The problem on trees is MAX-SNP hard since it contains the MAX-3SAT problem [5]. We give a polynomial time 18-approximation algorithm for unit-capacity trees using a greedy strategy. Each step schedules the "densest residual subtree" for a multicast and discards the overlapping subtrees of different multicasts already selected. The "densest residual subtree" of a multicast is the subtree maximizing the ratio between a value related to the net increase of the objective function after the selection, and a weight associated with the subtree itself. The algorithm can be easily implemented using a dynamic programming approach. To the best of our knowledge no approximation algorithm was known for this problem before.

We also present the first approximation algorithm for multicast routing on on unit-capacity meshes. It combines several ideas from the unicast routing algorithm [14] with a formulation of the multicast routing problem as a fractional packing problem [11, 18, 22] which is solved using duality-based algorithms. The fractional solution is then rounded probabilistically, leading to a potentially infeasible set of multicast trees, which are used to guide the construction of an integral solution. As part of our algorithm we must solve the *escape problem*, also considered by [14] for the unicast routing problem. A straightforward extension of the approach of [14] to multicast routing leads to an $O(\log n)$-factor approximation for the escape problem for multicasts. We use instead a recursive approach to the escape problem that allows to acheive an $O((\log\log n)^2)$-factor approximation for the whole problem.

Our online results. We show that in the multicast setting polylogarithmic-competitive randomized algorithms are also possible for restricted topologies: We present an $O(\log n(\log n + \log\log \mathcal{M})\log \mathcal{M})$-competitive multicast algorithm for trees and an $O(\log^2 n(\log n + \log\log \mathcal{M})\log \mathcal{M})$-competitive multicast algorithm for meshes. The algorithm for meshes extends several ideas of the online unicast algorithm for meshes [14] and uses the multicast routing algorithm for general networks with small bandwidth requestsi [10] as a subroutine. We also show a randomized lower bound of $\Omega((\log n \log \mathcal{M})/d)$ for a connected graph with minimum degree d. This gives a lower bound of $\Omega(\log n \log \mathcal{M})$ for meshes. The same lower bound for trees follows from [1]. No competitive multicast algorithms were known for these topologies before. There are various difficulties that multicast algorithms face over unicast algorithms. One of them is that latter multicasts might be more profitable than earlier ones. To deal with this problem our algorithms accept each multicast that pass an initial screening for "routability" with roughly equal probability – no matter at what time the requests of the multicast arrive.

Section 2 and 3 present the offline respectively online algorithm on trees. Section 4 gives the offline algorithm on meshes, section 5 presents the online algorithm on meshes. The proofs and details omitted in this extended abstract are given in the full version of the paper available at http://www.research.digital.com/SRC/personal/monika/papers.html.

2 The offline algorithm for trees.

We present a constant-factor approximation algorithm on trees. To denote the i-th multicast whose request node set is V we use the pair (i, V). A *submulticast* (i, V') of (i, V) is a multicast with source s_i and request node set $V' \subseteq V$. Our approach is to use a greedy algorithm that maintains an initially empty set S of (potentially) accepted submulticasts and assigns a *weight* and a *residual profit* to each submulticast. The algorithm repeatedly adds to S the submulticast that maximizes the ratio of its residual profit to its weight. Since the algorithm is offline, it can first accept a submulticast and then later add or subtract from it. We indicate this by saying that (i, V) *is added to or removed from* the current set S of submulticasts. Two submulticasts (i, V) and (i', V') *overlap* if they share an edge. We only add (i, V) to S if its residual profit is positive which will imply that its profit is significantly larger than the profit lost by submulticasts which overlap with (i, V).

We root the tree T at an arbitrary leaf. This defines an ancestor-descendant relation on the nodes of the tree. Let $T(i, V)$ be the tree connecting the nodes of V to the source of i. The highest node of $T(i, V)$ is called the root $root(i, V)$ of (i, V). Note that the root does not have to belong to V. We say r is a *subroot* of (i, V) if r is the root of one of the submulticasts of (i, V). For each multicast (i, v) and each subroot r we say (i, V') is the *maximum submulticast* $max(i, r)$ of (i, V) if (i, V') is the submulticast of (i, V) with root r that has the maximum number of requests.

Next we define a weight for each submulticast such that submulticasts "higher" in the tree have higher weight. Hence are added to S "later", except if they are very profitable. Note that the number of submulticasts can be exponential in n. To give a polynoial time algorithm we define the weight such that all submulticasts with the same root have the same weight, i.e., the weight of a submulticast only depends on its root and its multicast. Let (i, V) be a multicast and let r be one of its submulticast. Given a multicast (i', V') with $i' \neq i$, let $R(i', i, r)$ be the set of subroots r' of (i', V') such that r' is a true descendant of r and $max(i', r')$ overlaps with $max(i, r)$. We define the *weight* $w(i, r)$ to be

$$w(i, r) = 1 + \sum_{i' \neq i} \max_{r' \in R(i', i, r)} w(i', r').$$

The weight of a submulticast (i, V') with root r is $w(i, r)$.

For all multicasts (i, V) and (i', V') with $i \neq i'$ and all subroots r, $max(i, r)$ and $R(i', i, r)$ can be computed in polynomial time. Thus, $w(i, r)$ can be computed in polynomial time by a bottom-up traversal of the tree.

The *profit* $p(i, V)$ of a submulticast (i, V) is the number of requests in (i, V). For $i \neq i'$ the *overlapping profit* $p(i, V, i', V')$ of submulticast (i, V) and (i', V') is defined to be the profit of the maximum submulticast of (i', V') whose requests cannot be accepted if (i, V) is accepted, i.e., the number of requests of (i', V') that cannot be accepted in (i, V) is accepted. For $i = i'$ the *overlapping profit* $p(i, V, i', V')$ of submulticast (i, V) and (i', V') is defined to be the profit of $(i', V' \cap V)$. Note that in general $p(i, V, i', V') \neq p(i', V', i, V)$.

Let $O(i, V)$ be the set of submulticasts overlapping with (i, V). For a submulticast (i, V) the *residual profit*

$$p_{res}(i, V) = p(i, V) - \alpha \sum_{(i', V') \in S \cap O(i, V)} p(i, V, i', V'),$$

where $\alpha > 1$ is a constant to be chosen later. Let the *ratio* $r(i, V)$ of a submulticast be defined to be $p_{res}(i, V)/w(i, r)$, where $r = root(i, V)$. Now the greedy algorithm works as follows:

(1) $S = \emptyset$
(2) for each submulticast (i, V):
    the residual profit $p_{res}(i, V) = p(i, V)$
(3) while there exists a submulticast not in S with positive residual profit:
(4) Let (i, V) be a submulticast with maximum $r(i, V)$ of all submulticasts not in S.
(5) Let $S_{del} = \{(i', V''), (i', V'')$ is the maximum submulticast of $(i', V') \in S$ whose requests cannot be accepted together with $(i, V)\}$
(6) $S = S \cup (i, V) \setminus S_{del}$
(7) Update the residual profit for each submulticast.

Let the profit $p(S)$ of set S of submulticasts be $\sum_{(i,V) \in S} p(i, V)$. If (i, V) is added to S, then

$$\sum_{(i', V') \in S \cap O(i, V)} p(i, V, i', V') = p(S_{del}).$$

Thus, the residual profit of a submulticast compares its profit with the profit lost from S if the submulticast is added to S. We first show that the algorithm terminates.

LEMMA 2.1. *The algorithm terminates after at most $n\mathcal{M}$ iterations.*

To prove that this algorithm gives a constant factor approximation we distinguish three types of overlaps: If $T(i, V)$ contains an edge incident to the $root(i', V')$ then (i, V) is *ancestor-touching (a-touching)* (i', V').

Note that either $root(i',V') = root(i,V)$ or $root(i,V)$ is an ancestor of $root(i',V')$. If $root(i,V)$ is a true descendant of $root(i',V')$ and $T(i,V) \subseteq T(i',V')$ then (i,V) is *internal* to (i',V'). Otherwise, i.e., if (i,V) and (i',V') overlap, $root(i,V)$ is a true descendant of $root(i',V')$, but $T(i,V) \not\subseteq T(i',V')$ then (i,V) is *descendant-touching (d-touching)* (i',V').

The weight of a multicast was defined such that the following lemma holds.

LEMMA 2.2. *Let S be a set of nonoverlapping submulticasts that are internal or d-touching to a submulticast (i,V) such that S contains at most one submulticast for each multicast i'. Then*

$$\sum_{(i',V')\in S} w(i',root(i',V')) \leq w(i,root(i,V)).$$

The next lemma follows easily from the definition of a-touching.

LEMMA 2.3. *Let S be a set of nonoverlapping submulticasts. Then for each submulticast (i,V),* $\sum_{(i',V')\in S \text{ a-touches } (i,V)} p(i',V',i,V) \leq 2p(i,V)$.

Let S_{opt} be the set of submulticasts chosen by the optimum algorithm and let S_f be the final value of S. Note that every submulticast in S_{opt} overlaps with a submulticast in S_f. We partition S_{opt} as follows: Let S_2 be the set of submulticasts in S_{opt} that are d-touching or internal to a submulticast of S_f. Let S_1 be the set of submulticasts in S_{opt} that are a-touching to a submulticast of S_f, but are not internal or d-touching to any submulticast of S_f.

LEMMA 2.4. $p(S_1) \leq 2\alpha p(S_f)$

LEMMA 2.5. $p(S_2) \leq (10+2\alpha)p(S_f)$ *for* $\alpha \geq 2$.

Proof. To prove the lemma we show the following claim by induction on the number of iterations j: let S_j be the set S after iteration j. Let D_j be the subset of S_{opt} consisting of all submulticasts that d-touch or are internal to a submulticast in $\cup_{k\leq j} S_k$. Let $A(i,V)$ be the set of submulticasts that a-touch (i,V). Then

$$\sum_{(i',V')\in D_j} p(i',V') \leq 10 \sum_{(i,V)\in S_j} p(i,V) + \alpha \sum_{(i,V)\in S_j} \sum_{(i',V')\in D_j\cap A(i,V)} p(i',V',M,V).$$

The claim holds before iteration 1 since S_0 and D_0 are empty. Assume the claim holds before iteration j. Let (i,V) be added to S in iteration j and let S_{del} be deleted. Let $\Delta = D_j \setminus D_{j-1}$. Then the left side of the inequality increases by $\sum_{(i',V')\in\Delta} p(i',V')$. We need to show that the right side increases by at least so much.

Each $(i'',V'') \in S_{j-1}$ is partitioned into two submulticasts (i'',V_1'') and (i'',V_2'') with $(i'',V_1'') \in S_j$ and $(i'',V_2'') \in S_{del}$. Note that $p(i',V',i'',V'') \leq p(i',V',i'',V_1'') + p(i',V',i'',V_2'')$. By Lemma 2.3

$$\sum_{(i'',V_2'')\in S_{del}} \sum_{(i',V')\in D_{i-1}\cap A(i'',V_2'')} p(i',V',i'',V_2'') \leq 2p(S_{del})$$

Thus,

$$\sum_{(i'',V'')\in S_{j-1}} \sum_{(i',V')\in D_{i-1}\cap A(i'',V'')} p(i',V',i'',V'') \leq \sum_{(i'',V_1'')\in S_j} \sum_{(i',V')\in D_{i-1}\cap A(i'',V_1'')} p(i',V',i'',V_1'') + 2p(S_{del}).$$

Thus, the total decrease of the right side by removing S_{del} from S is at most $(10+2\alpha)p(S_{del})$. It follows that the right side increases by at least

$$\alpha \sum_{(i'',V'')\in S_j} \sum_{(i',V')\in\Delta\cap A(i'',V'')} p(i',V',i'',V'') + 10p(i,V) - (10+2\alpha)p(S_{del}).$$

We know that $p(i,V) \geq \alpha p(S_{del})$, which implies that $7p(i,V) \geq (10+2\alpha)p(S_{del})$ for $\alpha \geq 2$.

The inductive step of the proof is completed by showing

$$\sum_{(i',V')\in\Delta} p(i',V') \leq 3p(i,V) + \alpha \sum_{(i'',V'')\in S_j} \sum_{(i',V')\in\Delta\cap A(i'',V'')} p(i',V',i'',V'').$$

The proof of this last statement is omitted in this abstract.

It follows that $p(S_{opt}) = p(S_1) + p(S_2) \leq (4\alpha + 10)p(S)$ for $\alpha \geq 2$. Choosing $\alpha = 2$ gives an approximation factor of 18.

2.1 The polynomial time implementation of the algorithm. Given a set S, the algorithm must compute at each step a submulticast of maximum ratio $r(i,V)$ in polynomial time. Note that it suffices to compute for each multicast i and each subroot r^* the submulticast $best(i,r^*)$ with maximum residual profit. The desired submulticast is the one that maximizes over all multicasts i and all possible root positions r^* of i the ratio $p_{res}(best(i,r^*))/w(i,r^*)$. To find $best(i,r^*)$ we first compute a cost $cost(e)$ for each edge in the tree,

that is basically the profit that is lost by S if edge e is assigned to multicast i and is no longer available for submulticasts of S. Then we construct a rooted binary tree T' from the original tree and use bottom-up dynamic program on T' to determine $best(i, r^*)$. Details are given in the full version of the paper.

3 The online algorithm for trees.

In this section we assume that the sources and the request nodes are leaves of the tree. The general case can be easily reduced to this case. In the first stage the algorithm runs the small-bandwidth multicast algorithm, called MC, of [10] on a tree with capacity $\log \mu$, where $\mu = 4n^6\mathcal{M}$ and adds the accepted requests to $\mathcal{C}$. When applied to trees and compared to an offline algorithm with link capacity 1, MC is $O(\log n + \log \mathcal{M})$-competitive: A first $O(\log n)$ factor is saved since in a tree both the online algorithm and the offline algorithm connect the requests accepted by both algorithms to the root through the same multicast tree. An additional $O(\log \mu)$ factor is saved since the online algorithm has $\log \mu$ more capacity on the edges. This is proved in the same way as for unicast (see [16] and [15]).

In the second stage all the nodes of the tree are partitioned into $O(\log n)$ different classes by recursively finding a balanced tree separator. A *balanced tree separator* [21] is a vertex whose removal splits the tree into pieces of at most $\frac{2}{3}n$ vertices. The tree separator of T is assigned *level 0*. Removing the level-0 node splits T into subtrees of *level-1*. In general, the tree separators of the level-j trees are assigned *level* j and removing them creates subtrees of *level* $j+1$. After a logarithmic number of recursions the trees obtained are single vertices and the procedure stops. A similar technique is used in [3] for the online call-control problem on trees.

Each of the requests in $\mathcal{C}$ is assigned to one of $O(\log n)$ classes as follows. A request from vertex v to multicast source s is assigned to *class* j if the vertex of lowest level on the path from v to s has level j. One of the $O(\log n)$ classes, called j, is chosen at random by the algorithm before to process the sequence of requests. A request in $\mathcal{C}$ is handled by the following algorithm: (A) If the request is not of class j then reject it and stop. If the request is the first one of multicast i seen at this step, then flip a coin with success probability $\frac{1}{u}$. If success then pass to step (B) the current and all the future requests to i seen at this step; otherwise reject all the future requests to i seen at this step and stop. (B) Accept a request from vertex v to source s if no edge on the path from v to s is assigned to other multicasts; otherwise reject.

The following lemma bounds the expected number of requests accepted by ST.

LEMMA 3.1. *The algorithm ST expects to accept an $\frac{1}{O(\log n \log \mu)}$ fraction of the requests accepted by MC.*

This leads to the theorem below. A randomized lower bound of $\Omega(\log n \log \mathcal{M})$ follows from [1] since multicast routing on trees of unit capacity contains the online set cover problem.

THEOREM 3.1. *There exists an $O((\log n + \mathcal{M})(\log n + \log\log \mathcal{M}) \log n)$-competitive algorithm for multicast routing on unit-capacity trees.*

4 The offline algorithm for a mesh.

We present an $O((\log\log n)^2)$-factor approximation algorithm on meshes. The algorithm partitions the mesh into squares of logarithmic size and divides every square into an external and an internal region. The external region of a square is reserved to route requests into, out of, and through the square. It is called the *crossbar structure* of the mesh. To avoid edge-overlapping we discard all requests whose request node or source belongs to an external region. From the remaining requests the algorithm considers with equal probability either only *short requests* directed from a request vertex to a source in the same square, or only *long requests* directed from a request vertex to a source in a different square. A randomized rounding technique based on a novel formulation of the multicast routing problem as an integer linear program is then used in conjunction with the use of a simulated network with edges of higher capacity.

Let G denote an $n \times m$ two dimensional mesh such that $m = \Theta(n)$. Wlog $m \geq n$. We assume n sufficiently large such that $\lfloor \log\log\log n \rfloor \geq 3$. Define $B = 4\lfloor \log n \rfloor$, $f(k) = k \text{ div } 9B$, and $f_1(k) = k \bmod 9B$. Given two integer values a and b an (a, b, B)-*partitioning* of the mesh G is a partitioning into $f(n) \times f(m)$ submeshes of $O(B)$ size induced by segmenting the horizontal and the vertical side of the mesh. The horizontal side is partitioned into a segment from column 1 to column a, followed by $f(m) - f_1(m)$ contiguous segments of size $9B$, by $f_1(m) - 1$ segments of size $9B + 1$ and by a last segment of size $9B + 1 - a$. The vertical side of the mesh is partitioned in a similar way with b used in place of a and n instead of m. By abuse of notation every resulting submesh is called a *square*, even though the size of the two sides of a square may differ. Note that each node belongs to exactly one square while an edge can be incident to nodes of two different squares. We denote the square containing a node t by S_t.

The *border of* G is formed by all nodes of degree less than 4. The *1st ring* in a square S consists of all nodes of S that are incident to a node outside of S or belong to the border of G. Recursively, the *i-th ring* of S with $i > 1$ consists of all nodes of S that are incident to a

node of ring i-1 of S. The innermost ring of a square is either a single vertex or a line of nodes. A ring that is not the innermost ring either forms a rectangle (if its square does not contain nodes of the border of G) or forms a rectangle with one or two borders of G. In any square S we define two *regions* R_S^1 and R_S^2. Region R_S^1 consists of rings from 1 to B, region R_S^2 contains all remaining rings of S. Ring $B+1$ is the *border* of R_S^2.

Let $\mathcal{A}$ be the sequence of requests. The algorithm chooses two integer values a and b uniformly at random in the interval $2B+1, ..., 7B$ and constructs an (a, b, B)-partitioning. Then it discards all requests (t, s) such that either t or s does not belong to the R^2 region of its square. The set of remaining requests is denoted by $\mathcal{C}$. The following lemma implies that for any input sequence $\mathcal{A}$, $E[|OPT(\mathcal{C})|] \geq |OPT(\mathcal{A})|/25$.

LEMMA 4.1. *Two nodes t and s both belong to region R^2 of their squares with probability at least* $1/25$.

By the choice of a and b, at least $2B$ rings are contained in a square. Thus, region R_S^1 is always complete, while region R_S^2 is formed by at least B rings.

The set of requests $\mathcal{C}$ is partitioned into the set of long requests $\mathcal{L} = \{(t,s) \in \mathcal{C} : S_t \neq S_s\}$ and the set of short requests $\mathcal{S} = \{(t,s) \in \mathcal{C} : S_t = S_s\}$. For $i \in \mathcal{M}$, denote by $\mathcal{L}_i = \{t : (t, s_i) \in \mathcal{L}\}$ the set of request nodes of multicast i. The algorithm decides with equal probability to either reject all requests of $\mathcal{L}$ or of $\mathcal{S}$ and to run a specialized algorithm for the remaining requests. We describe the algorithm specialized for long requests. The algorithm for short requests is in the full version of the paper.

Long requests. Our approach is to transform the problem into a problem on a network G', then formalize the problem on G' as IP, relax it to an LP, solve the LP, and round the LP solution probabilistically. Finally we use the rounded solution to construct a solution in G.

Mesh G is transformed into a network $G' = (V', E')$ as follows. For every square S of G, network G' contains vertex x_S. The vertices x_S and $x_{S'}$ of two adjacent squares S and S' are connected by an edge of capacity $\lfloor \log n \rfloor$. For every square S, every vertex u of region R_S^2 has a corresponding vertex u' in G'. For any pair of adjacent nodes u, v in R_S^2, nodes u', v' in G' are linked with an edge of unit capacity. Every vertex on the border of R_S^2 is connected to x_S by an edge of unit capacity. For every multicast i and every request vertex $u \in \mathcal{L}_i$, a vertex u_i' is connected to vertex u' with a unit-capacity edge. The input sequence for the multicast routing problem on G' is created by transforming every request $(t, s) \in \mathcal{L}_i$ into a request (t_i', s_i') in G'.

The next step is to formulate the multicast routing problem in G' as a packing problem: For every multicast i consider the set $\mathcal{T}_i$ consisting of all trees containing s_i and a non-empty subset of the request nodes t_i'. Since we introduced the nodes t_i', $\mathcal{T}_i \cap \mathcal{T}_j = \emptyset$ for $i \neq j$. Let $\mathcal{T} = \cup_{i \in \mathcal{M}} \mathcal{T}_i$. Denote by $V(T)$ the set of vertices of tree T and by $E(T)$ the set of edges. Let the benefit of tree $T \in \mathcal{T}_i$ be $b(T) = |\{t_i' \in V(T) : t \in \mathcal{L}_i\}|$. We associate a variable $x_T \in \{0, 1\}$ with every tree $T \in \mathcal{T}$. Edges of E' are subject to constraints:

$$\sum_{T \in \mathcal{T} : e \in E(T)} x_T \leq c(e), \ \forall e \in E'; \tag{4.1}$$

$$\sum_{T \in \mathcal{T}_i : e \in E(T)} x_T \leq 1, \ \forall e \in E', \forall i \in \mathcal{M}. \tag{4.2}$$

The multicast routing problem consists in maximizing the following objective function:

$$S = \sum_{T \in \mathcal{T}} b(T) x_T.$$

The fractional packing problem is obtained by replacing the integrality constraints on variables x_T with constraints $x_T \geq 0$. We also drop edge constraints (4.2) to obtain a linear program where every edge is involved in a single constraint and solve it using the polynomial time ϵ-approximation algorithm of Garg and Könemann [11] based on duality. The algorithm assigns a dual variable $y(e)$ to every edge $e \in G'$. The central step of the algorithm requires to find the variable x_T with maximum ratio $opt = b(T)/\sum_{e \in T} y(e)$. This problem is *NP-hard*, since it corresponds to finding the densest tree in the network G' where edges are weighted with the values of the dual variables. However it is easily checked that if we find a variable $x_{\tilde{T}}$ with $b(\tilde{T})/\sum_{e \in \tilde{T}} y(e) \geq opt/\alpha$ for some constant α then the algorithm of [11] also gives a ϵ-approximation of the fractional multicast problem on G'.

As was previously observed by [7] a k-MST algorithm can be used to solve the densest tree problem. The 3-approximate k-MST algorithm of Garg [9] can be adapted to work in the case the k vertices are restricted to be request vertices of the same multicast. Thus, for every multicast i and every $k = 1, ..., |\mathcal{L}_i|$, the 3-approximate k-MST algorithm is applied. It finds the tree $T_i(k)$ spanning k request vertices of $\mathcal{L}_i$ such that $\sum_{e \in T_i(k)} y(e) \leq 3opt_{k,i}$, where $opt_{k,i} = \min\{\sum_{e \in T} y(e), b(T) = k, T \in \mathcal{T}_i\}$. Then the tree of maximum ratio $k/\sum_{e \in T_i(k)} y(e)$ over all k and all i is selected. Since this ratio has value at least $opt/3$, this results in an ϵ-approximation algorithm for the fractional multicast problem.

Denote by x^*_T the solution of the fractional multicast routing problem[1]. Let $s = 1/((c \log\log n)^2)$, where $c \geq e$ is an appropriate constant to be fixed later. The algorithm rounds variable x_T to $\overline{x}_T = 1$ with probability sx^*_T, and to $\overline{x}_T = 0$ with probability $1 - sx^*_T$. Let $\overline{G}_i$ be the graph with edges $E(\overline{G}_i) = \cup_{T \in \mathcal{T}_i : \overline{x}_T = 1} E(T)$. For any multicast i the algorithm selects an arbitrary spanning tree $\overline{T}_i$ of graph $\overline{G}_i$. The trees $\overline{T}_i$ do not form the integral solution since there might be violated edge capacities for the unit-capacity edges. However, as described below, the requests accepted by the final solution form a subset of the requests accepted by the trees $\overline{T}_i$ and the size of the subset is a constant fraction of the requests accepted by the trees of $\overline{T}_i$. To prove the approximation bound we show in the full paper that the value S of the optimal solution of the fractional packing formulation is within a constant factor of the optimal integral solution on the set of requests $\mathcal{L}$, and that the expected number of request nodes contained in the trees $\overline{T}_i$ is within a factor of $O((\log\log n)^2)$ of the value S.

Let $\mathcal{L}^1_i = \mathcal{L}_i \cap V(\overline{T}_i)$ be the set of request vertices to multicast i that are spanned by tree $\overline{T}_i$ if no edge $(x_S, x_{S'})$ of G' is violated, $\mathcal{L}^1_i = \emptyset$ otherwise, and let $\mathcal{L}^1 = \sum_{i \in \mathcal{M}} \mathcal{L}^1_i$. We prove in the full version of the paper that with at least constant probability no edge $(x_S, x_{S'})$ of G' is violated , i.e., $\mathcal{L}^1 \neq \emptyset$. If $\mathcal{L}^1 = \emptyset$, the algorithm terminates without accepting any multicast. Otherwise, each request of $\mathcal{L}_i$ accepted by the final solution is routed along a path containing the same edges $(x_S, x_{S'})$ as its path to the source in $\overline{T}_i$. The remaining problem is to route request nodes and sources to the border of their square. This problem was called the *escape problem*. The solution proposed by Kleinberg and Tardos for unicast routing [14] uses the fact that the benefit collected in a square is of the same order as the maximum flow that can be routed through the border of the square. This is not true for multicast routing: the maximum benefit that can be collected in a square is $O(\log^2 n)$, while the maximum flow that can be routed through the border of the square is $O(\log n)$. Thus, using the same maximum flow approach as in [14], which means routing request nodes individually out of the square, leads to an $O(\log n)$-factor approximation. We give instead a recursive approach that achieves a $O((\log\log n)^2)$-factor approximation.

Our basic idea is to recursively partition every region R^2_S into *subsquares* of size $O(\log\log n)$, and each subsquare Q into subregions R^1_Q and R^2_Q. Requests are routed to the border of R^2_Q on the same path as in the trees $\overline{T}_i$ and from there they use rings of the R^1 regions of subsquares to reach the border of R^2_S. The sequence of subsquares used for a request is the same as on the path in $\overline{T}_i$. Therefore we enforce that the trees $\overline{T}_i$ are edge-disjoint within the R^2_Q regions and that there are at most $O(\log\log n)$ trees connecting between any two neighboring subsquares.

We next give the details: A *gate vertex* for multicast i in square S is a vertex q on the border of R^2_S such that (q, x_S) belongs to $\overline{T}_i$. Let $g(p)$ be the gate vertex closest to node $p \in \mathcal{L}_i$ on the path from p to s_i in $\overline{T}_i$ closest to p. Let $g(s_i)$ be a gate vertex closest to s_i on a path from s_i to a node outside S_{s_i} in $\overline{T}_i$. The *escape problem* is the problem to connect each request node p to $g(p)$ and to connect each source to s to at least one $g(s_i)$. Let S be a square whose region R^2_S consists of a $k_1 \times k_2$ mesh. Let $k = \min(k_1, k_2)$ and let $B_S = 4\lfloor \log k \rfloor$. Note that $k \geq B$. The algorithm uniformly chooses two integer values a_S and b_S from the interval $2B_S + 1, \ldots, 7B_S$ for each square S and creates an (a_S, b_S, B_S)-partitioning for the region R^2_S. Each submesh Q created by this partitioning is called a *subsquare*. If Q does not contain nodes of the border of R^2_S, *region* R^1_Q of subsquare Q consists of rings 1 to B_S, *region* R^2_Q consists of the remaining part of Q. If Q contains nodes of the border of R^2_S, we need a different definition: Let Q be a $k_3 \times k_4$ mesh with $k_3, k_4 \leq 9B_S$. Assume Q is extended into a $9B_S \times 9B_S$ mesh Q' by nodes outside of R^2_S. Regions $R^1_{Q'}$ and $R^2_{Q'}$ are defined as above. Region R^1_Q is then $R^1_{Q'} \cap R^2_S$ and region R^2_Q is $R^2_{Q'} \cap R^2_S$. By the choice of a_S and b_S and the definition of R^2_Q there are gate vertices in S that belong to R^2_Q if subsquare Q lies on the border of R^2_S.

We give next the algorithm for long requests. (1) The algorithm rejects all requests whose source or request node belongs to the region R^1_Q of their subsquare Q. The remaining set of requests is called $\mathcal{L}^2$. A subsquare is called *invalid* if one of the edges of G' incident to a node in the subsquare belongs to more than one tree $\overline{T}_i$. Since every edge is assigned to a tree with probability $O(1/((\log\log n)^2)$, a subsquare in not invalid with at least constant probability. (2) Every request node belonging to an invalid subsquare is discarded and every multicast whose source belongs to an invalid subsquare is discarded. The set of remaining requests is called $\mathcal{L}^3$. A square S is called *invalid* if there exists a pair of neighboring subsquares Q and Q' of S such that more than $B_S/4$ trees $\overline{T}_i$ contain an edge incident to Q and Q'. Every square is proved to be not invalid with at least constant probability. (3) Every request node belonging to an invalid square is discarded and every multicast whose source belongs to

[1] Let for some i, $\{T^{(1)}, \ldots, T^{(j)}\}$ be the set of all the trees of $\mathcal{T}_i$ with $x_{T^{(l)}} = 1$, for all $1 \leq l \leq j$. Then $T^{(1)} \cup \ldots \cup T^{(j)}$ forms the multicast tree for multicast i.

an invalid subsquare is discarded. The set of remaining requests is called $\mathcal{L}^4$. (4) All request nodes p in $\mathcal{L}^4$ such that $g(p)$ belongs to R^1_Q for some subsquare Q are discarded and multicast i is discarded if all gate vertices of square S containing source s_i belong to R^1_S. The set of remaining requests is called $\mathcal{L}^5$. (5) Finally all requests p of $\mathcal{L}^5$ such that $g(p)$ belongs to an invalid subsquare are discarded, and multicast i is discarded if in a square S containing source s_i all gate vertices in S connected to s_i in $\overline{T}_i$ belong to invalid subsquares. The set of remaining requests, called $\mathcal{L}^6$, is accepted. Set $\mathcal{L}^6$ is expected to be at least a constant fraction of set $\mathcal{L}^4$.

Short requests are handled by running the algorithm in each square recursively and solving the "recursive short request problem" by brute force. Details of the algorithm, the routing, and of the analysis of the approximation ratio are given in the full paper.

We end stating the main theorem:

THEOREM 4.1. *There exists an $O((\log\log n)^2)$-factor approximation algorithm for multicast routing on unit-capacity meshes.*

5 The online algorithm for a mesh.

We propose an algorithm with polylogarithmic competitive ratio on meshes. It partitions the mesh into squares of size $13B \times 13B$, where $B = \Theta(\log n)$. Then it uses four main ideas: (1) It "filters" requests in stage one to "make space" for routing, but it guarantees that if a square contains requests, then at least one request of them survives the filtering. Thus, step one "looses" an $O(\log^2 n)$ factor. (2) Stage two contracts each square to a node and runs the algorithm MC of [10] on G'. For each accepted request MC returns a path consisting of a sequence of neighboring squares. To translate this sequence into a path in the original mesh we have to be able to construct B disjoint paths between neighboring squares. The idea is that a path from a neighboring square enters a square in the "middle" B links between the two squares. Within a square each path is assigned its own concentric ring on which it proceeds until it reaches either the appropriate row[2] or column to exit the square or its multicast tree. (3) However there can be requests accepted by MC which cannot be routed "locally", i.e., there is a conflict in the squares of the endpoints. These requests have to be rejected. In the unicast setting this causes no problem since the rejection of a request does not affect the routing of requests accepted later on. In the multicast setting, however, MC might output a path in G' that does not connect the request to its source in G since an earlier request of the same multicast was accepted by MC and rejected by our algorithm. We handle this situation by always connecting the same squares as MC even if the request is not accepted. (4) Since latter requests might be more profitable than earlier ones, the algorithm *selects* each multicast with roughly equal probability (after passing some additional screening for "routability") and discards all unselected multicasts.

The first stage. Let $G = (V, E)$ denote the $n \times n$ two dimensional mesh. We assume that n is sufficiently large such that $B = \lfloor \frac{\lfloor \log n \rfloor}{13} \rfloor \geq 1$. Let $f = n \text{ div } \lfloor \log n \rfloor$ and let $f_1 = n \bmod \lfloor \log n \rfloor$. We partition the mesh into f^2 submeshes of logarithmic size by segmenting every side into $f - f_1$ contiguous segments of size $\lfloor \log n \rfloor$ followed by f_1 segments of size $\lceil \log n \rceil$. By abuse of notation every submesh is called a *square*, even though the size of the two sides may differ by 1. We denote the square containing node t by S_t. The first *ring* in a square S consists of all nodes of S that either are incident to a node outside of S or have degree less than 4 in the mesh G. Recursively, the *i-th ring* of S with $i > 1$ consists of all nodes of S that are incident to a node of ring $i-1$ of S. In any square S we define three *regions* R^1, R^2 and R^3. Region R^1 consists of rings 1 to $2B$, region R^2 consists of rings $2B+1$ to $4B$, and region R^3 is formed by rings $4B+1$ to $6B$ and the remaining piece of S, called the *central region* of S. The central region is a rectangle with sides of size at least B, i.e., consisting of at least B rings.

The first stage (i) selects for each square one of its regions at random to route paths "through" the square and (ii) *dedicates* each ring randomly either to sources or request nodes. Requests not conforming to the random choices are rejected to guarantee (a) that they do not overlap with the paths routed in the selected region and (b) that they do not interfer with the routing of the source resp. request nodes chosen for the ring. The details are as follows: (1) Dedicate each ring to multicast sources with probability 1/2, otherwise to request nodes. (2) Select uniformly at random one of the three regions in each square. (3) Discard all the requests from vertex t to source s if t or s are in a selected region. (4) Discard all the requests from a vertex t on a ring dedicated to sources, unless the request is directed to a source s on the same ring of t. (5) Discard all the multicasts whose source is in a ring dedicated to requests.

Let C be the sequence of requests not discarded by stage one. Denote by $OPT(\mathcal{A})$ the requests out of a sequence $\mathcal{A}$ accepted by the optimal algorithm.

[2] We use *row* to denote a horizontal path and *column* to denote a vertical path in the mesh.

LEMMA 5.1. *For any input sequence $\mathcal{A}$, $E(|OPT(\mathcal{C})|) \geq \frac{1}{12}|OPT(\mathcal{A})|$.*

The second stage. The second stage of the algorithm receives as input the requests of $\mathcal{C}$ accepted by the first stage, in the order in which they are presented to the algorithm. It partitions $\mathcal{C}$ into the set $\mathcal{L}_0$ of *long requests*, and the set $\mathcal{S}_0$ of *short requests*. A request (t, s) is a *long* request if at presentation no branch of the multicast rooted at s is in S_t. Otherwise, (t, s) is a *short* request. The algorithm routes short requests "locally" within the square and uses MC for long requests. We sketch the admission control algorithm for long requests.

To guarantee that the trees used for different multicasts are edge-disjoint we maintain the invariant that *(I1) all edges of a ring that belong to any multicast tree belong to the same multicast tree.* To maintain the invariant each ring is *assigned* by the algorithm to at most one multicast and this is the only multicast whose tree is allowed to use edges of the ring. To achieve this each request of $\mathcal{C}$ has to pass various tests in four steps before it is accepted. The requests which are not rejected after step i, $i = 1, 2, 3, 4$, form a sequence $\mathcal{L}_i$.

Whenever the first request of a multicast is added to $\mathcal{L}_2$, the algorithm decides whether the multicast is *selected for long requests.* This is needed (1) to discard multicasts were the "local" routing causes potential conflicts and (2) to guarantee that latter multicasts have roughly the same probability of being accepted as earlier ones. A multicast with source s is *selected for long requests* if all of the following conditions are fulfilled at the time of the test: no multicast with source on the ring of s is already selected for *short* requests; no multicast with source in S_s is already selected for *long* requests; if s is in R^3 then the largest ring of R^3 in S_s is dedicated to sources; and a coin toss with success probability $1/(4B)$ is successful.

We now give the details of the decision algorithm when a request (t, s) arrives. Let G' be a mesh such that each square of the original mesh is represented by a vertex in G' and two vertices of G' are connected by an edge if the two corresponding squares are adjacent. Each edge has capacity B.

(1) If a long request with request node or source in S_t was previously added to $\mathcal{L}_3$, the algorithm rejects (t, s) and stops. If a short request with request node in S_t has been accepted, the algorithm rejects (t, s) and stops. Otherwise it adds the request to $\mathcal{L}_1$.

(2) The request (t, s) of $\mathcal{L}_1$ is transformed into a request between the two vertices S_t and S_s of G', and then submitted to MC. If MC accepts the transformed request, request (t, s) is added to $\mathcal{L}_2$. In this case MC also returns a route in G' which corresponds to a sequence of squares in the original mesh. Otherwise, the request is rejected and the algorithm stops.

(3) If the multicast of the request (t, s) in $\mathcal{L}_2$ is selected for long requests, the request is added to $\mathcal{L}_3$ and an unassigned ring of the selected region of S_t is assigned to the multicast. Otherwise the algorithm rejects the request and stops.

(4) If t is not in the central region of S_t, then (t, s) is added to $\mathcal{L}_4$. If t belongs to the central region of S_t, and one of rings $4B + 1, \ldots, 6B$ in S_t is dedicated to request nodes then (t, s) is added to $\mathcal{L}_4$ and one of rings $4B + 1, \ldots, 6B$ in S_t dedicated to request nodes is assigned to the multicast.

If (t, s) is added to $\mathcal{L}_4$ it is accepted. The ring of t is assigned to the multicast of (t, s) and t is connected to the multicast tree of s. Otherwise the request is rejected and an arbitrary node u on the assigned ring of the selected region is connected to the multicast tree of s.

Short Requests. The algorithm for short requests decides whether to accept or reject a request in three steps. The requests which are not rejected after step l, $l = 1, 2, 3$ form a sequence $\mathcal{S}_l$.

Whenever the first request with $S_t = S_s$ of a multicast is added to $\mathcal{S}_2$, the algorithm decides whether the multicast is *selected for short requests.* A multicast with source s is *selected for short requests* if at the time of the test no short request of a multicast with source in S_s was previously added to $\mathcal{S}_2$, and a coin toss with success probability $1/2$ is successful.

In the following let y denote a node of S_t that belongs to the multicast tree of s. Note that $y = s$ is possible. The decision part of the algorithm for short requests consists of three steps:

1. If a short request with request node in S_t has been accepted then reject (t, s) and stop. If a long request with request node in S_t has been added to $\mathcal{L}_3$, reject (t, s) and stop. If a long request with source in S_t has been added to $\mathcal{L}_3$, reject (t, s) and stop. Otherwise add (t, s) to $\mathcal{S}_1$.

2. If either t or y, but not both, is in the central region, the other vertex is not in R^3, and ring $4B+1$ to $6B$ of S_t are all dedicated to sources, then reject (t, s) and stop. Otherwise add (t, s) to $\mathcal{S}_2$.

3. If $S_t \neq S_s$ or if $S_t = S_s$ and the multicast with source s is selected for short requests, then add (t, s) to $\mathcal{S}_3$. Otherwise reject (t, s) and stop. Accept every request (t, s) in $\mathcal{S}_3$.

We omit the routing algorithm and sketch the proof of the competitiveness. Let $\rho = \log n(\log n + \log\log \mathcal{M}) \log \mathcal{M}$. We prove the expected number of requests accepted by the second stage of the algorithm is an $O(\rho \log n)$ fraction of $OPT(\mathcal{C})$, for any possible set $\mathcal{C}$. Together with Lemma 5.1 it follows that our algorithm is $O(\rho \log n) = O(\log^2 n(\log n + \log\log \mathcal{M}) \log \mathcal{M})$

competitive. Since

$$|OPT(\mathcal{C})| = |OPT(\mathcal{C}) \cap ((\mathcal{L}_0 \setminus \mathcal{L}_1) \cup (\mathcal{S}_0 \setminus \mathcal{S}_1))| + |OPT(\mathcal{C}) \cap \mathcal{L}_1| + |OPT(\mathcal{C}) \cap \mathcal{S}_1|$$

it suffices to show the following results whose proofs are omitted:

$|OPT(\mathcal{C}) \cap ((\mathcal{L}_0 \setminus \mathcal{L}_1) \cup (\mathcal{S}_0 \setminus \mathcal{S}_1))| \leq 48 \log^2 n E[|ON(\mathcal{C})|]$,
$|OPT(\mathcal{C}) \cap \mathcal{L}_1| \leq O(\rho \log n) E[|ON(\mathcal{C})|]$, and
$|OPT(\mathcal{C}) \cap \mathcal{S}_1| \leq O(\log^2 n) E[|ON(\mathcal{C})|]$.

THEOREM 5.1. *There exists an* $O(\log^2 n(\log n + \log\log \mathcal{M}) \log \mathcal{M})$*-competitive algorithm for multicast routing on unit-capacity meshes.*

The proof of the following lower bound closely follows [10].

THEOREM 5.2. *No algorithm for selective online multicast routing on a graph with minimum degree d can have a competitive ratio better than* $\Omega((\log(\mathcal{M}/u) \log n)/d)$ *even against an oblivious adversary, where u is the capacity of an edge.*

References

[1] B. Awerbuch, Y. Azar, A. Fiat, and T. Leighton. Making commitments in the face of uncertainty: How to pick a winner almost every time. In *Proc. of the 28th Annual ACM Symposium on Theory of Computing*, pages 519–530, 1996.

[2] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput-competitive online routing. In *34th IEEE Symposium on Foundations of Computer Science*, pages 32–40, 1993.

[3] B. Awerbuch, Y. Bartal, A. Fiat, and A. Rosén. Competitive non-preemptive call control. In *Proc. of 5th ACM-SIAM Symposium on Discrete Algorithms*, pages 312–320, 1994.

[4] B. Awerbuch, R. Gawlick, T. Leighton, and Y. Rabani. On-line admission control and circuit routing for high performance computing and communication. In *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science*, pages 412–423, 1994.

[5] P. Alimonti, 1997. Personal communication.

[6] B. Awerbuch and T. Singh. On-line algorithms for selective multicast and maximal dense trees. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 354–362, 1997.

[7] B. Awerbuch. Online selective multicastand maximal dense trees: A survey, 1996. Available as http://www.cs.jhu.edu/baruch/MULTICAST/index.html.

[8] Y. Bartal, A. Fiat, and S. Leonardi. Lower bounds for on-line graph problems with application to on-line circuit and optical routing. In *Proceedings of the 28th ACM Symposium on Theory of Computing*, pages 531–540, 1996.

[9] N. Garg. A 3-approximation for the minimum tree spanning k vertices. In *Proc. of the of 37th Annual IEEE Symposium on Foudations of Computer Science*, pages 302–309, 1996.

[10] A. Goel, M.R. Henzinger, and S. Plotkin. Online throughput-competitive algorithm for multicast routing and admission control. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete algorithms*, pages 97–106, 1998.

[11] N. Garg and J. Koenemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. Technical Report MPI-I-97-1-025, Max Planck Institute fuer Informatik, 1998. To appear in *Proceedings of the 39th IEEE Annual Symposium on Foundations of Computer Science*, Palo Alto, November 8-11, 1998.

[12] N. Garg, V.V. Vazirani, and M. Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. In *Proceedings of the International Colloquium on Automata, Languages and Programming*, LNCS, pages 64–75. Springer-Verlag, 1993.

[13] D. Hochbaum. *Approximation Algorithms for NP-Hard Problems.* PWS, 1997.

[14] J. Kleinberg and È. Tardos. Disjoint paths in densely embedded graphs. In *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science*, pages 52–61, 1995.

[15] S. Leonardi and A. Marchetti-Spaccamela. On-line resource management with applications to routing and scheduling. In *Proceedings of the 23rd International Colloqium on Automata, Languages and Programming*, LNCS 955, pages 303–314. Springer-Verlag, 1995.

[16] S. Leonardi, A. Marchetti-Spaccamela, A. Presciutti, and A. Rosèn. On-line randomized call-control revisited. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms*, pages 323–332, 1998.

[17] M. Mihail, C. Kaklamanis, and S. Rao. Efficient access to optical bandwidth. In *Proc. of the of 36th Annual IEEE Symposium on Foudations of Computer Science*, pages 548–557, 1995.

[18] S. Plotkin, D. Shmoys, and É Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20:257–301, 1995.

[19] Y. Rabani. Path-coloring on the mesh. In *Proceedings of the 37th Ann. IEEE Symposium on Foundations of Computer Science*, pages 400–409, 1996.

[20] P. Raghavan and E. Upfal. Efficient routing in all-optical networks. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 133–143, 1994.

[21] Jan van Leeuwen ed. *Handbook of theoretical computer science, Vol A, Algorithms and Complexity.* The MIT Press, 1990.

[22] N. Young. Randomized rounding without solving teh linear program. In *Proceedings of the 6th ACM-SIAM Symposium on Discrete Algorithms*, pages 170–178, 1995.