SRC Technical Note
1998 - 010

May 8, 1998

Scheduling Multicasts on Unit-Capacity Trees and Meshes

Monika Rauch Henzinger and Stefano Leonardi

Systems Research Center
130 Lytton Avenue
Palo Alto, California 94301

http://www.research.digital.com/SRC/

Stefano Leonardi is at the Max-Planck Institute für Informatik, Saarbrücken, Germany and at the Dipartimento di Informatica Sistemistica, Università di Roma “La Sapienza”, via Salaria 113, 00198-Roma, Italia. His electronic mail addresses is: leon@dis.uniroma1.it

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.

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 [3]. 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 [7] 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 [12] 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 necessarily a realistic assumption. There are many applications, for instance 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 [6]. 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.

This paper studies 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 [5, 6, 21, 17], they are at the basis of topologies for communication networks such as trees of rings, often considered as interconnection of SONET rings optical networks [21, 17], or topologies for connecting high performance multicomputers systems as trees of meshes [6] 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. [6, 14, 19, 16]).

Note that the multicast routing problem in unit-capacity graphs reduces to the *edge-disjoint paths problem* if only one request is presented for each multicast, also 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 [11], 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". They formulated the *escape problem* as an interesting subproblem and gave a constant-factor approximation for the escape problem in the unicast setting. A straightforward extension of their approach leads to a $O(\log n)$-factor approximation for the escape problem for multicasts. We use instead a recursive approach that achieves a $O((\log \log n)^2)$-factor approximation.

For the online problem no algorithm, not even a randomized one, has a polylogarithmic competitive ratio for any network topology [8]. This holds even in the unicast setting. Therefore in the unicast setting restricted graph topologies like trees, meshes, and "densely embedded, nearly-Eulerian graphs" [5, 6, 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 contains the MAX-3SAT problem and is thus MAX-SNP hard [1]. We present a polynomial time approximation algorithm for unit-capacity trees that achieves an approximation ratio of 18. The algorithm presents an interesting version of 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 on unit-capacity meshes. Our polynomial time algorithm obtains an approximation ratio of $O((\log \log n)^2)$. It formulates the multicast routing problem as a fractional packing problem [10, 18, 23] 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.

Our online results. An online algorithm A for the multicast routing problem has *competitive ratio c* if on any input sequence the ratio of the number of requests accepted by the optimum offline algorithm to the expected number of requests accepted by A is at most c.

This paper shows how to achieve polylogarithmic-competitive algorithms for restricted topologies in the multicast setting: For trees we present an $O(\log n(\log n + \log \log \mathcal{M}) \log \mathcal{M})$-competitive multicast algorithm and for meshes we give an $O(\log^2 n(\log n + \log \log \mathcal{M}) \log \mathcal{M})$-competitive multicast algorithm. 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 [3]. 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. Thus, our algorithms accept each multicast that pass an initial screening for "routability" with roughly equal probability.

Section 2 of this paper presents the constant approximation algorithm on trees, Section 3 contains the online algorithm for trees, Section 4 gives the offline algorithm on meshes, Section 5 presents the online algorithm on meshes.

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 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 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 multicast such that multicasts "higher" in the tree have higher weight and hence are added to S "later", except if they are very profitable. Given a submulticast (i, V) with root r and 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 $max(i, r)$. For each multicast (i, V) and each possible root position r, we define the *weight* $w(i, r)$ to be

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

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

Proof. Whenever a multicast (i, V) is added to S and a set S_{del} of submulticasts is deleted, $p_{res}(i) > 0$. It follows that $p(i, V) > \alpha p(S_{del})$. Thus, $p(S)$ increases in each iteration by at least 1. The maximum value it can assume is $n\mathcal{M}$. ■

We prove in the next subsection that this algorithm gives a constant factor approximation of the optimum solution. In the following subsection we show how to implement each iteration in polynomial time.

2.1 Proof of the constant approximation ratio

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 *Let S be a set of nonoverlapping submulticasts that are all internal or d-touching to a submulticast (i, V) such that S contains at most one submulticast for each multicast i'. Then $\sum_{(i',V')\in S} w(i', root(i', V')) \le w(i, root(i, V))$.*

Proof. Let $r = root(i, V)$ and let $(i', V') \in S$ be a submulticast with root r'. Note that r' is a true descendant of r. Furthermore (i', V') and (i, V) overlap, thus $max(i, r)$ and $max(i', r')$ overlap. Hence, $r' \in R(i', i, r)$ and $w(i', r') \le \max_{r''\in R(i',i,r)} w(i', r'')$.

Thus,

$$\begin{aligned}
\sum_{(i',V')\in S} w(i', root(i', V')) &\le \sum_{(i',V')\in S} \max_{r''\in R(i',i,r)} w(i', r'') \\
&\le \sum_{i'\ne i} \max_{r''\in R(i',i,r)} w(i', r'') \\
&< w(i, root(i, V))
\end{aligned}$$

■

The following lemma is used repeatedly.

Lemma 3 *Let S be a set of non-overlapping submulticasts.*

Then for each submulticast (i, V), $\sum_{(i',V')\in S \text{ a-touches } (i,V)} p(i', V', i, V) \le 2p(i, V)$.

Proof. Let S' be the set of submulticasts of S that a-touch (i, V). Let r be the root and let s be the source of (i, V). Furthermore let e be the edge on the path from

s to r that is incident to r. Let (i^*, V^*) be the submulticast in S' whose multicast tree contains e if such a submulticast exists.

For each submulticast $(i', V') \neq (i^*, V^*)$ in S' let $children(i', V')$ be the children v of r such that (v, r) belongs to $T(i', V')$. Then the overlapping profit $p(i', V', i, V)$ is at most the profit of (i, V) in the subtrees of $children(i', V')$. Since the submulticasts in S' are non-overlapping, the set $children(i', V')$ and $children(i'', V'')$ are disjoint for any pair $(i', V'), (i'', V'') \in S'$. Thus,

$$\sum_{(i',V')\in S', (i',V')\neq(i^*,V^*)} p(i', V', i, V) \leq p(i, V).$$

Since $p(i^*, V^*, i, V) \leq p(i, V)$, the lemma follows. ■

Next we show that the above algorithm gives a constant approximation of the optimum result: 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 4 $p(S_1) \leq 2\alpha p(S_f)$

Proof. Let $(i', V') \in S_1$. Let S'_f be the set of submulticasts of S_f that are a-touched by (i', V'). Note that $S'_f = O(i', V') \cap S_f$ and that the residual profit of (i', V') is not positive at termination. Thus,

$$p(i', V') \leq \alpha \sum_{(i,V)\in S_f\cap O(i',V')} p(i', V', i, V).$$

Thus, by Lemma 3,

$$p(S_1) = \sum_{(i',V')\in S_1} p(i', V') \leq \alpha \sum_{(i'V')\in S_1} \sum_{(i,V)\in S_f\cap O(i',V')} p(i', V', i, V)$$

$$= \alpha \sum_{(i,V)\in S_f} \sum_{(i',V')\in S_1 \text{ a-touches } (i,V)} p(i', V', i, V) \leq \alpha \sum_{(i,V)\in S_f} 2p(i, V) \leq 2\alpha p(S_f)$$

■

Next we handle submulticasts in S_2.

Lemma 5 $p(S_2) \leq (10 + 2\alpha)p(S_f)$ *for* $\alpha \geq 2$.

Proof. The lemma follows from the following claim which we show 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 $\cup_{k \le j} S_k$. Then*

$$\sum_{(i',V') \in D_j} p(i',V') \le 10 \sum_{(i,V) \in S_j} p(i,V) + \alpha \sum_{(i,V) \in S_j} \sum_{(i',V') \in D_j : (i',V') \text{ a-touches } (i,V)} p(i',V',i,V).$$

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 $p(i',V',i'',V'') \le p(i',V',i'',V''_1) + p(i',V',i'',V''_2)$. By Lemma 3

$$\sum_{(i'',V''_2) \in S_{del}} \sum_{(i',V') \in D_{i-1} : (i',V') \text{ a-touches } (i'',V''_2)} p(i',V',i'',V''_2) \le 2p(S_{del})$$

Thus,

$$\sum_{(i'',V'') \in S_{j-1}} \sum_{(i',V') \in D_{i-1} : (i',V') \text{ a-touches } (i'',V'')} p(i',V',i'',V'') \le$$

$$\sum_{(i'',V''_1) \in S_j} \sum_{(i',V') \in D_{i-1} : (i',V') \text{ a-touches } (i'',V''_1)} p(i',V',i'',V''_1) + 2p(S_{del}).$$

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

$$10p(i,V) + \alpha \sum_{(i'',V'') \in S_j} \sum_{(i',V') \in \Delta : (i',V') \text{ a-touches } (i'',V'')} p(i',V',i'',V'') - (10+2\alpha)p(S_{del}).$$

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

We show below that

$$\sum_{(i',V') \in \Delta} p(i',V') \le 3p(i,V) + \alpha \sum_{(i'',V'') \in S_j} \sum_{(i',V') \in \Delta : (i',V') \text{ a-touches } (i'',V'')} p(i',V',i'',V''). \quad (*)$$

Thus, the inductive claim continues to hold.

To show $(*)$ we consider two cases. Let $A = \Delta \cap S_{del}$. Let B be the rest of Δ.

Bounding A: As shown above, $\alpha p(S_{del}) \leq p(i, V)$. Thus, $p(A) \leq p(i, V)/\alpha \leq p(i, V)$.

Bounding B: Consider a multicasts (i', V') in B. When (i, V) is added, (i', V') does not belong to S_{j-1} and it is not selected by the algorithm. Thus, the ratio $r(i', V')$ is at most the ratio $r(i, V)$. Note that each multicast has at most one submulticast in B. Thus, using Lemma 2 it follows that

$$\sum_{(i',V')\in B} p_{res}(i', V') = \sum_{(i',V')\in B} r(i', V')w(i', root(i', V')) \leq$$

$$r(i, V) \sum_{(i',V')\in B} w(i', root(i', V')) \leq r(i, V)w(i, root(i, V)) = p_{res}(i, V).$$

Thus,

$$\sum_{(i',V')\in B} \big(p(i', V') - \alpha \sum_{(i'',V'')\in S_{j-1}\cap O(i',V')} p(i', V', i'', V'')\big) \leq p(i, V) - \alpha p(S_{del}).$$

By definition no submulticast (i', V') in B d-touches or is internal to a submulticast in S_{j-1}. Thus, all submulticasts in $S_{j-1} \cap O(i', V')$ are a-touched by (i', V'). Using Lemma 3 for the second inequality in the same way as above shows that

$$\begin{aligned} p(B) &\leq p(i, V) - \alpha p(S_{del}) + \alpha \sum_{(i'',V'')\in S_{j-1}} \sum_{(i',V')\in B:(i',V') \text{ a-touches } (i'',V'')} p(i', V', i'', V'') \\ &\leq p(i, V) + \alpha p(S_{del}) + \alpha \sum_{(i'',V_1'')\in S_j} \sum_{(i',V')\in B:(i',V') \text{ a-touches } (i'',V_1'')} p(i', V', i'', V_1'') \\ &\leq 2p(i, V) + \alpha \sum_{(i'',V_1'')\in S_j} \sum_{(i',V')\in B:(i',V') \text{ a-touches } (i'',V_1'')} p(i', V', i'', V_1''). \end{aligned}$$

Since $p(\Delta) = p(A) + p(B)$, this shows $(*)$. ■

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.

2.2 The polynomial time implementation of the algorithm

We are left with showing how to implement each iteration in polynomial time. Given a set S, the algorithm must compute at each step a submulticast of maximum ratio $r(i, V)$. Note that it suffices to compute for each multicast i and each possible root position 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

$$\frac{p_{res}(best(i, r^*))}{w(i, r^*).}$$

Let T_v be the subtree of T rooted at node v.

We describe a polynomial time procedure based on dynamic programming that finds for each multicast i and for each possible root position r^* a submulticast $best(i, r^*)$ with maximum residual profit. To be precise it suffices to consider all submulticasts of $max(i, r^*)$.

Let (i, V) be the submulticast of i that currently belongs to S ($V = \emptyset$ if no such submulticast exists.) To find $best(i, r^*)$ root the tree T at the source s of $max(i, r^*)$. (This rooting is completely independent of the rooting in the previous section.) We first compute a cost $cost(e)$ for each edge in the tree. Then we construct a binary tree T' from the original tree for use in a dynamic program. Next, we use the edge costs to compute two cost functions $cost_v^0$ and $cost_v^1$ for each vertex $v \in T'$ using bottom-up dynamic programming. Finally, $cost_s^1$ can be used to determine $best(i, r^*)$.

Computing the edge costs: Use a depth-first traversal of the tree starting at s. When traversing edge e, a cost is assigned to e. The cost of edge e will be the profit that is lost by S when this edge is assigned to multicast i and thus is no longer available for submulticasts in S, ignoring the profit that S already lost on the edges along the path from e to s.

Formally, we have the following definition: If e belongs to $(i, V) \in S$, or e does not belong to any submulticast in S, or e does not belong to the multicast tree of $max(i, r^*)$, $cost(e)$ is 0. Otherwise, e belongs to the multicast tree of $max(i, r^*)$ and to a submulticast (i', V') of S with $i' \neq i$. Let $e = (u, v)$ and let u be the parent of v. Let $p(i', V', s, x)$ be the number of requests of (i', V') that use an edge on the path from s to x. Then $cost(e) = p(i', V', s, v) - p(i', V', s, u)$.

Obviously all edge costs can be computed in polynomial time.

To illustrate the edge costs, assume for a moment that S contains only 1 submulticast (i', V'). Let s' be the source of i'. Then the only edge costs that have non-zero value are the ones on the path from s' to s or incident to a node on the path from s' to s. Call these edge costs the *edge costs for* (i', V'). If S consists of more than one submulticast (i_j, V_j), then the edge costs are simply assigned considering separately each (i_j, V_j). Thus, a multicast $i' \neq i$ of S with an edge of positive cost incident to v has its source in the subtree rooted at v.

Constructing the binary tree T': For the bottom-up dynamic programming approach we need to transform the tree into a binary tree T' by introducing additional nodes and edges. The cost of each additional edge is 0. Let v be a node in the original tree with $d \geq 1$ children. We call an edge incident to v *unused* if it does not belong to a submulticast of S or has $cost(e) = 0$. All the other edges incident to v have positive cost and belong to a submulticast of S. Recall that a multicast $i' \neq i$ of S with an edge of positive cost incident to v has its source in the subtree rooted at v. Let q be the number of different multicasts with a edge of positive cost incident to vertex v.

We replace v by a binary tree with $q + 1$ leaves, if there are unused edges incident to v, and with q leaves otherwise. The first q are called *special* and correspond to a submulticast in S incident to v with at least an edge of positive cost; the last leaf L, if applicable, corresponds to all unused edges incident to v. Leaf L is the root of a binary tree with l leaves, where l is the number of unused edges incident to v. Each unused edge incident to v in T is incident to one of these leaves such that each leaf is incident to one of the unused edges.

Let $v(i')$ be the special node corresponding to submulticast $(i', V') \in S$ and assume t edges of (i', V') are incident to v. Then $v(i')$ has two children, one corresponding to the edge e connecting v to the source of i' and the other being the root of a binary tree with $t - 1$ leaves. The *subtree for $v(i')$ in T'* consists of $v(i')$, its children and the binary tree of $t - 1$ leaves. Edge e is connected to "its"' leaf and each other edge of (i', V') incident to v is incident to one of the other leaves such that each leaf is incident to one of these edges. Note that these nodes have one child each in T', i.e., are not leaves in T'.

The root of the binary tree for v is labeled v. A leaf of the binary tree corresponding to a leaf v of T is also labeled v.

All submulticasts are extended in a natural way to T'.

Computing the cost functions $cost^0$ and $cost^1$: We want to compute a submulticast of $max(i, r^*)$ with maximum residual profit. Thus, we find for each possible profit b, $b = 1, 2, \ldots, n$, the submulticast (i, V^*) that gets b profit *and* maximizes $b - \alpha cost(b)$, where $cost(b)$ is the minimum profit that S "looses" if (i, V^*) is accepted. This is equivalent to finding the submulticast with profit b that minimizes $cost(b)$. The basic idea is to use bottom-up dynamic programming to compute for each node v' in T' the submulticast that minimizes $cost_{v'}(b)$, where $cost(b)_{v'}$ is the minimum cost that S "looses" in the subtree rooted at v' if the submulticast is accepted.

However, there is a complication at special nodes that requires the use of two cost functions, $cost^0_{v'}$ and $cost^1_{v'}$: Let w be a node of T with 2 children y and z, both belonging to submulticast $(i', V') \in S$. Let y lie on the path from v to the source of i'. Let v' be the special node $v(i')$ in the subtree of w in T' and let l' and r' be its children such that $T'_{l'}$ contains the node labeled y. Assume profit $0 < j < b$ is achieved in T_y and profit $b - j$ is achieved in T_z. Then we do not want to add both

$cost(w,y)$ and $cost(w,z)$ to $cost_{v'}$, since $cost(w,y)$ considers already the cost of "loosing" all the requests of (i', V') whose path uses w. Instead we want to only add $cost(w,y)$ and *not* $cost(w,z)$ to $cost_{v'}$. Thus, to "tell" the recursion on r' that it should not add in any more "lost profit" of (i', V') we use the cost function $cost^0_{r'}$. If the recursion on r' is allowed to add in more "lost profit" of (i', V') (since e.g. no profit was achieved in T_y), we use $cost^1_{r'}$. Note that this complication arises only at special nodes.

We need to use $cost^0$ in $T'_{r'}$ until we reach the leaf x' of the subtree of $v(i')$ in T' whose edge e to its child corresponds to (w,z). When computing $cost^0_{x'}$ we do not add in the cost $cost(w,z)$ of e and then use $cost^1$ to recurse on the children of x'. Thus, $cost^0_{x'}$ is only needed at nodes belonging to the subtree of $v(i')$ in T' for some submulticast $(i', V') \in S$.

We next give the formal definitions. When we want to avoid that the dynamic program chooses a specific $cost^t_{v'}(b)$, $t = 0, 1$, combination, we set its value to ∞.

(1) For all $v' \in T'$ and for $t = 0, 1$: $cost^t_{v'}(0) = 0$.

(2) For all $v' \in T'$ such that there are less than b requests of $max(i, r^*)$ in the subtree of v in T' and for $t = 0, 1$: $cost^t_{v'}(b) = \infty$.

(3) For any $b > 0$, for $t = 0, 1$, and for all $v' \in T'$ with one child, call the missing child r'. We assume that $cost^t_{r'}(b) = \infty$.

(4) For all non-special $v' \in T'$, let c be the number of requests of $\max(i, r^*)$ at v'. Then,

$$\begin{aligned} cost^1_{v'}(b) = \quad & \min\{\min_{\{j=1,..,b-c-1\}} cost(v', l') + cost^1_{l'}(j) + cost(v', r') \\ & \qquad\qquad + cost^1_{r'}(b - c - j), \\ & cost(v', l') + cost^1_{l'}(b - c), cost(v', r') + cost^1_{r'}(b - c)\} \end{aligned}$$

For all special $v' = v(i') \in T'$, let l' be the child of v that leads to the source of multicast i'. Then

$$\begin{aligned} cost^1_{v'}(b) \quad = \quad & \min\{\min_{\{j=1,..,b-1\}} cost(v', l') + cost^1_{l'}(j) + cost^0_{r'}(b - j), \\ & cost(v', l') + cost^1_{l'}(b), cost(v', r') + cost^1_{r'}(b)\} \end{aligned}$$

(5) For all $v' \in T'$ whose children are not labeled by a vertex of T, let c be the number of requests of $\max(i, r^*)$ at v'. Then

$$cost^0_{v'}(b) \quad = \quad \min_{\{j=0,..,b-c\}}(cost^0_{l'}(j) + cost^0_{r'}(b - c - j))$$

For all $v' \in T'$ whose children are labeled by a vertex of T

$$cost^0_{v'}(b) \quad = \quad \min_{\{j=0,..,b\}}(cost^1_{l'}(j) + cost^1_{r'}(b - j))$$

In (5) we are *not* adding in $cost(v', l')$ and $cost(v', r')$.

Computing $cost^t_{v'}(b)$ for a given v' and b by bottom-up dynamic programming takes time $O(n)$. Since there are n different values for b and $O(n)$ nodes in the tree, we spend time $O(n^3)$ for multicast i and possible root position r^* to compute all $cost^0$ and $cost^1$ values.

Lemma 6 *Let s' be the node of T' labeled with the source s of i. The largest residual profit of any submulticast of $max(i, r^*)$ is* $\max_b(b - \alpha cost^1_{s'}(b))$.

Proof. Let $T'_{v'}$ be the subtree of T' rooted at v'. We prove the following claims by bottom-up induction:

(1) For each vertex $v' \in T'$ $cost^1_{v'}(b)$ equals

- the minimum number of requests in $T'_{v'}$ of submulticasts in S that cannot be accepted if a submulticast of $max(i, r^*)$ accepts b submulticasts in $T'_{v'}$,
- and ∞ otherwise.

(2) For each vertex $v' \in T'$ that belongs to the subtree of $v(i')$ for some $(i', V') \in S$, $cost^0_{v'}(b)$ equals

- the minimum number of requests in $T'_{v'}$ of submulticasts in $S \setminus (i', V')$ that cannot be accepted if a submulticast of $max(i, r^*)$ accepts b submulticasts in $T'_{v'}$,
- and ∞ otherwise.

The second part of each claim follows by the definition of $cost^1_{v'}$ and $cost^0_{v'}$, we only need to show the first part.

(1) For the basis of the induction, if v' is a leaf of T' it is labeled with a leaf v of T. We can assume that b requests of $max(i, r^*)$ are at v'. Note that any submulticast (i', V') of S with a request at v' also must have its source at v'. Thus, the request of (i', V') can be accepted, even if b requests of $max(i, r^*)$ are accepted. Hence, $cost^1_{v'}(b) = 0$, and the claim holds.

If v' is an internal vertex of T', we distinguish two cases, depending on whether v' is a special node or not. Let l' and r' be the two children of v'.

Case 1: v' of T' is not a special node. If v' is not a special node of T', then $cost^1_{v'}(b)$ is defined by the first part of rule (4).

If only one edge (v', l') is incident to v' then $cost^1_{v'}(b) = cost(v', l') + cost^1_{l'}(b - c)$ and the claim holds by induction on vertex l'. Otherwise, $cost^1_{v'}(b)$ is defined as the minimum over all the possible partitions of $b - c$ between the two subtrees $T'_{l'}$ and $T'_{r'}$. Since inductively $cost^1_{l'}$ and $cost^1_{r'}$ are equal to the minimum profit that S looses in $T'_{l'}$ and $T'_{r'}$, the claim holds also for $cost^1_{v'}$.

Case 2: v' of T' is a special node. If $v' = v(i')$ is a special node of T', then $cost^1_{v'}(b)$ is defined by the second part of rule (4). Let $(i', V') \in S$.

The minimum number of lost requests is the minimum over all the possible partitions of b between the subtree $T'_{l'}$ that contains the edge leading to the source of i' and the subtree $T'_{r'}$ containing the other edges of $T(i', V')$. Let w be the node in T to whose binary tree n T' v' belongs and let y be the child of w such that T_y contains the source of i'. Note that in T' the edge e from l to its only child in T' has cost $cost(w, y)$.

If the submulticast of $max(i, r^*)$ accepts requests in $T'_{l'}$ then the submulticast contains e. By the definition of $cost(w, y)$, e's cost $cost(w, y)$ contains already all the profit of (i', V') lost in $T'_{r'}$. Thus, no further lost profit in $T'_{r'}$ should not be added to $cost^1_{v'}$. By the inductive claim, $cost^0_{r'}$ computes the minimum profit lost by S in $T'_{r'}$, *not* considering the profit lost by the submulticast to which edge (v', r') belongs, i.e., the profit of (i', V').

Since $cost^1_{v'}(b)$ is again defined as the minimum over all the possible partitions of $b - c$ between the two subtrees, using $cost^0_{l'}$ for $T'_{l'}$ and $cost^0_{r'}$ for $T'_{r'}$. Thus, the claim follows as in case 1.

If, however, the submulticast of $max(i, r^*)$ does not accept requests in $T'_{l'}$, then the lost profit equals $cost(v', r') + cost^1_{r'}(b - c)$ and the claim holds by induction on r'.

(2) The edges in T' with non-zero cost are edges that connect nodes labeled by a vertex of T to their parents. Let $e = (x', y')$ be such an edge belonging to submulticast $(i', V') \in S$. Note that the higher endpoint x' of e belongs to the subtree of $v(i')$ in T' and that all children of x', including y', are labeled with a node in T. Then the cost of no further edge in $T'_{y'}$ is related to the profit of (i', V'), i.e., if (i', V') were the only submulticast in S, all edges in $T'_{y'}$ would have zero cost.

To prove the claim for $cost^0_{v'}(b)$ we distinguish again two cases.

Case 1: v''s children are labeled by vertices of T. Note that v' is not labeled and thus, there is no request of $max(i, r^*)$ at v'. By the above observation, no edges in the subtree of v''s children have costs related to the profit of (i', V'). Using induction on l' and r', the minimum number of requests in $T'_{v'}$ of submulticasts in $S \setminus (i', V')$ that cannot be accepted if a submulticast of $max(i, r^*)$ accepts b submulticasts in $T'_{v'}$ is given by the minimum over all $0 \leq j \leq b$ of $cost^1_{l'}(j) + cost^1_{r'}(b - j)$. Thus, the claim holds.

Case 2: v''s children are not labeled by vertices of T. Let (i', V') be the submulticast such that v' belongs to the subtree of $v(i')$. Note that the edges from v' to its children have cost 0 and that v' is an internal node of the subtree of $v(i')$. Thus, the children l' and r' of v' also belong to the subtree of $v(i')$. Inductively their

$cost^0$ equals the minimum number of requests in T'_l, resp. $T'_{r'}$, of submulticasts in $S \setminus (i', V')$ that cannot be accepted if a submulticast of $max(i, r^*)$ accepts b submulticasts in T'_l, resp. $T'_{r'}$. Hence the minimum for $T'_{v'}$ is given by the minimum over all $0 \leq j \leq b - c$ of $cost^0_{l'}(j) + cost^0_{r'}(b - c - j)$. Thus, the claim holds. ■

Determining $best(i, r^*)$: By Lemma 6 the largest residual profit of a submulticast of $max(i, r^*)$ is found by determining the value b_{best} that maximizes the function $b - \alpha cost^1_s(b)$. The submulticast $best(i, r^*)$ is found by reconstructing which edges were used for the computation of $cost^1_s(b_{best})$.

3 The online algorithm for trees

We describe an online algorithm ST for unit-capacity trees. For sake of simplicity we assume that the sources as well as the members of the multicasts are leaves of the tree. The general case can be easily reduced to this setting. The algorithm consists of two stages. The first stage is a randomized procedure that selects a subset of requests that will form the set of "candidate" requests $\mathcal{C}$ for the second stage. The second stage decides for each request in $\mathcal{C}$ whether to accept or reject it. The requests accepted by the second stage are the requests accepted by ST.

The first stage of the algorithm.
The first stage runs the multicast algorithm, called MC, of [12] on a tree with capacity u, where $u = \log \mu$ and $\mu = 4m^6\mathcal{M}$. Then stage 1 adds the accepted requests to $\mathcal{C}$.

On a graph where each link has capacity u, MC achieves a competitive ration of $O((\log n + \log\log \mathcal{M})(\log n + \mathcal{M}) \log n)$. However, when applied to trees and compared to an offline algorithm with link capacity 1, MC's competitive ratio is $O(\log n + \log \mathcal{M})$: 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. (In particular, in this setting the claim of Lemma 5.1 of [12] can be proved without the $O(\log n)$ factor). An additional $O(\log \mu)$ factor is saved since the online algorithm has $O(\log \mu)$ more capacity on the edges. This is proved in the same way as for unicast (see [16] and [15]). (In particular in [12] a factor of $O(\log n + \log\log \mathcal{M})$ is saved in Claim 5.7). Thus, MC, when applied to a tree network with $O(\log \mu)$ more capacity on the edges is $O(\log n + \log \mathcal{M})$-competitive.

The second stage of the algorithm.
In the second stage all the vertices of the tree are partitioned into $O(\log n)$ different classes, by recursively finding a balanced tree separator. A *balanced tree separator* [22] 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 also used in [5] 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 $O(\log n)$ levels is chosen at random by the algorithm before to process the sequence of requests. We denote with i the level selected.

Stage 2 decides to accept or reject a request in $\mathcal{C}$ using the following algorithm:

1. If the request is not of level i then reject it and stop.

2. 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 3 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.

3. 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 7 *The algorithm ST expects to accept an $\frac{1}{O(\log n \log \mu)}$ fraction of the requests accepted by MC.*

Proof. The expected number of requests passed from step 1 to step 2 of stage 2 is a $\frac{1}{O(\log n)}$ fraction of the requests accepted by MC, since the level of each request is chosen uniformly at random with probability $\frac{1}{O(\log n)}$. The expected number of requests passed from step 2 to step 3 is a fraction $\frac{1}{u}$ of all the requests received from step 1. This follows since all the requests for a multicast are passed to step 3 with probability $\frac{1}{u}$.

We are left to prove that each request received at step 3 is accepted with constant probability. Consider a pair of requests for different multicasts. If they intersect, the intersection is on an edge adjacent to a level i vertex. Each request is connected to the source through at most two edges adjacent to a level i vertex. The probability that a request is accepted is then given by the probability that these 2 edges are not assigned to other multicasts.

Any edge of the tree is part of at most u multicasts given the maximum capacity of the edges of the tree for the online algorithm in the MC solution. The edge is assigned to each of these multicasts with probability $\frac{1}{u}$, if the algorithm decides to accept requests from this multicast. When a request arrives at step 3, the probability that an edge adjacent to the level i vertex on the path to the source has not been assigned to a different multicast is then lower bounded by $(1 - \frac{1}{u})^u \geq 1/e$. The probability that both edges adjacent to a level i vertex have not been assigned is then lower bounded by $1/2e$, thus proving the claim. ■

This leads to the following theorem:

Theorem 8 *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.*

A randomized lower bound of $\Omega(\log n \log \mathcal{M})$ follows from [3] since the multicast routing problem on trees of unit capacity contains the online set cover problem.

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 the $n \times m$ 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 $\mathcal{A}$, $E[|OPT(\mathcal{C})|] \geq |OPT(\mathcal{A})|/25$ since for every request (t, s) the probability that t and s both belong to R^2 is at least $1/25$.

Lemma 9 *Given two nodes t and s they both belong to region R^2 of their squares with probability at least* $1/25$.

Proof. We prove separately that the x and the y coordinates of t and s are within region R^2 with constant probability. Consider interval $I = [2B+1, ..., 7B]$ from which a is chosen uniformly at random. Since region R^1 has width $2B$, x_t falls within region R^2 if a is chosen out of a subset I_s of I of size at least $3B$. Analogously, x_s falls within region R^2 if a is chosen out of a subset I_t of I of size at least $3B$. Since I has size $5B$, for at least B out of $5B$ possible values of a, i.e. with probability at least 1/5, both x_t and x_s fall with region R^2. By symmetry, y_t and y_s fall within region R^2 with probability at least 1/5. It follows that with probability at least $1/25$ every request has both endpoints in region R^2. ■

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 accept either only long requests or only short requests. We describe next the algorithm specialized for long requests then the algorithm specialized for short requests.

4.1 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 of G, every vertex u of region R_S^2 has a corresponding vertex u' in G'. For any pair of adjacent vertices u, v in R_S^2, vertices u', v' in G' are linked with an edge of unit capacity. Every vertex of the border of R_S^2 is connected to x_S by an edge of unit capacity. For every multicast i and for 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:

$$\sum_{T \in \mathcal{T} : e \in E(T)} x_T \leq c(e), \ \forall e \in E'; \tag{1}$$

$$\sum_{T \in \mathcal{T}_i : e \in E(T)} x_T \leq 1, \ \forall e \in E', \forall i \in \mathcal{M}. \tag{2}$$

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 replacing the integrality constraints on variables x_T with constraints $x_T \geq 0$. We also drop edge constraints (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 [10] 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 $b(\tilde{T})/\sum_{e \in \tilde{T}} y(e) \geq opt/\alpha$ then the algorithm of [10] also gives a $1+\epsilon$-factor approximation of the fractional multicast problem on G'.

As was previously observed by [2] 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 $1+\epsilon$-factor 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 appendix 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 $\mathcal{L}$ (Lemma 17), 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 (Lemma 18).

Let $\mathcal{L}_i^1 = \mathcal{L}_i \cap V(\overline{T}_i)$ 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, $\mathcal{L}_i^1 = \emptyset$ otherwise, and let $\mathcal{L}^1 = \sum_{i \in \mathcal{M}} \mathcal{L}_i^1$. We prove that with at least constant probability no edge $(x_S, x_{S'})$ of G' is violated , i.e., $\mathcal{L}^1 \neq \emptyset$ (Lemma 19). If $\mathcal{L}^1 = \emptyset$, the algorithm terminates without accepting any multicast. Otherwise, each request of $\mathcal{L}_i$ accepted by the

[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.

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_S^2 into *subsquares* of size $O(\log\log n)$, and each subsquare Q into a subregions R_Q^1 and R_Q^2. Requests are routed to the border of R_Q^2 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_S^2. 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_Q^2 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_S^2 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_S^2 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_S^2. Each submesh Q created by this partitioning is called a *subsquare*. If Q does not contain nodes of the border of R_S^2, *region* R_Q^1 of subsquare Q consists of rings 1 to B_S, *region* R_Q^2 consists of the remaining part of Q. If Q contains nodes of the border of R_S^2, 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_S^2. Regions $R_{Q'}^1$ and $R_{Q'}^2$ are defined as above. Region R_Q^1 is then $R_{Q'}^1 \cap R_S^2$ and region R_Q^2 is $R_{Q'}^2 \cap R_S^2$. By the choice of a_S and b_S and the definition of R_Q^2 there are gate vertices in S that belong to R_Q^2 if subsquare Q lies on the border of R_S^2.

(1) The algorithm rejects all requests whose source or request node belongs to the region R_Q^1 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 (Lemma 23). (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 (Lemma 24). (3) Every request node belonging to an invalid square is discarded and every multicast whose source belongs to 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 $\mathcal{L}^4$ (Lemma 25 and 26).

4.1.1 Routing Algorithm

We next show how to route the accepted requests, i.e., how to construct the multicast tree T_i for multicast i. The basic idea is to route using the same squares and subsquares as the tree $\overline{T}_i$ but to route "through" a square S in R^1_S and "through" a subsquare Q in R^1_Q.

Let $\overline{T}^f_i$ be the subtree of $\overline{T}_i$ spanning s_i and the vertices in $\overline{T}_i \cap \mathcal{L}^6$, the request vertices accepted for multicast i. We first explain how to connect gate vertices. For every vertex x_S of $\overline{T}^f_i$ the algorithm assigns a ring $r_{S,i}$ of region R^1_S to multicast i. For every edge $(x_S, x_{S'})$ of $\overline{T}^f_i$ let r be the ring of $\{r_{S,i}, r_{S',i}\}$ with larger index. The algorithm assigns to multicast i the straightline extension of r from one corner of r to the other ring, thereby connecting $r_{S,i}$ and $r_{S',i}$. For a vertex x_S of $\overline{T}^f_i$ and every gate vertex q of multicast i belonging to the horizontal (vertical) side of the border of R^2_S and to a valid subsquare of S, the algorithm assigns to multicast i the vertical (horizontal) straightline path from q to $r_{S,i}$. Note that all gate vertices of multicast i in S are connected to the same ring $r_{S,i}$.

Next we describe how to connect a request node p of multicast i in S to its gate vertex $q = g(p)$. Let Q be the subsquare of p and Q' be the subsquare of q. Recall that $g(p)$ lies in region R^2_Q. Let P be the path between p and q in $\overline{T}_i$ and let e_p be the edge on P closest to p and incident to exactly one node of R^2_Q. Let e_q be defined symmetrically.

For every subsquare Q such that (i) $\overline{T}^f_i$ contains a vertex of R^2_Q and (ii) s_i or a request node of $\overline{T}_i$ belongs to S we assign a ring $r_{Q,i}$ of R^1_Q to multicast i, where S is the square containing Q.

If $Q = Q'$ and p is connected to q in $\overline{T}^f_i$ by a path in R^2_Q then p is connected to q in the same way in T_i. If $Q = Q'$ and p is connected to q in $\overline{T}^f_i$ by a path P with edges outside R^2_Q, we connect both p and q to $r_{Q,i}$ in T_i. Request vertex p is connected to $r_{Q,i}$ in T_i by the part of P from p to e_p and then the straightline extension from e_p to $r_{Q,i}$. Gate node q is connected to $r_{Q,i}$ in T_i in an analogue way.

If $Q \neq Q'$, then p is connected to q by the same path as above to $r_{Q,i}$, then the path P' described below to $r_{Q',i}$, and finally the same path as above from $r_{Q',i}$ to q. To construct path P' the above algorithm to construct a path between gate vertices is applied: for each neighboring pair (Q_1, Q_2) of subsquares on P a horizontal or vertical connection between rings $r_{Q_1,i}$ and $r_{Q_2,i}$ is assigned to the multicast. Path P' is a simple path between $r_{Q,i}$ and $r_{Q',i}$ created from the assigned rings and connections.

Next we describe how to connect the source of i to one of the gate vertices q of i in S. This suffices since all gate vertices of i in S are connected to the same ring $r_{S,i}$. Recall that one of the gate vertices, say q, belongs to R^2_S. The algorithm connects s to q in the same way as it connected a request node p to q.

4.1.2 Proof of Correctness

We show that the resulting multicast trees T_i are edge-disjoint. The multicast tree for multicast i is contained in the rings $r_{S,i}$ of squares and rings $r_{Q,i}$ of subsquares assigned to i, the assigned horizontal or vertical connections between assigned rings and from gate vertices or edges e_p to rings, and paths of $\overline{T}^f_i$ in regions R^2_Q of subsquares.

We distinguish the following seven types of paths used by the multi casts and show below that (1) for each square S the algorithm can assign a unique ring $r_{S,i}$ if $x_S \in \overline{T}^f_i$ (Claim 10); (2) for each subsquare Q the algorithm can assign a unique ring $r_{Q,i}$ if $\overline{T}^f_i$ contains a vertex of R^2_Q and s_i or a request node of $\overline{T}_i$ belongs to the square of Q (Claim 11); (3) for every edge $(x_S, x_{S'})$ in $\overline{T}^f_i$ the algorithm can assign a unique horizontal or vertical connection between $r_{S,i}$ and $r_{S',i}$ (Claim 12); (4) for each gate vertex q of multicast i in square S the algorithm can assign a unique horizontal or vertical connection between q and $r_{S,i}$ if q belongs to a valid subsquare (Claim 13); (5) for every neighboring pair (Q_1, Q_2) of subsquares and for each multicast i the algorithm can assign a unique horizontal or vertical connection between $r_{Q_1,i}$ and $r_{Q_2,i}$ if they both exist, (Claim 14); (6) if p is either a request vertex in $\overline{T}_i^f$ or a gate vertex such that a request node of $\overline{T}_i^f$ in S_p connects through p to s_i, then the algorithm can assign a unique horizontal or vertical connection between e_p to $r_{Q,i}$, where Q is the subsquare containing p (Claim 15); (7) if p is either a request vertex in $\overline{T}_i^f$ or a gate vertex such that a request node of $\overline{T}_i^f$ in S_p connects through p to s_i, then the algorithm can uniquely assign the path in $\overline{T}_i$ from p to e_p or to $g(p)$ if this path is contained in R_Q^2, where Q is the subsquare containing p (Claim 16). Since the set of edges that can be used for type-(j) paths is disjoint from the set of edges that can be used for type-(j') paths with $j \neq j'$, proving the claims shows the correctness of the algorithm.

Claim 10 *The number of trees $\overline{T}_i^f$ including vertex x_S for a square S is at most the number of rings B of region R_S^1.*

Proof. The number of trees $\overline{T}_i^f$ including an edge $e = (x_S, x_{S'})$ does not exceed $c(e)$. Every tree containing x_S but not a terminal in R_S^2 takes 2 units of capacity on edges $(x_S, x_{S'})$. Every tree containing a terminal in R_S^2 takes at least one unit of capacity on edges $(x_S, x_{S'})$. Since the overall capacity of edges $(x_S, x_{S'})$ is at most $4\lfloor \log n \rfloor = B$, a ring of R_S^1 can be assigned to every tree $\overline{T}_i^f$ containing x_S. ∎

Claim 11 *There are at most B_S trees $\overline{T}_i^f$ such that (i) $\overline{T}_i^f$ contains a vertex of region R_Q^2 of a subsquare Q and (ii) s_i or a request node of $\overline{T}_i^f$ belongs to S, where S is the square containing Q.*

Proof. Assume by contradiction that a subsquare Q does not fulfill the claim. The trees $\overline{T}_i^f$ that contain a vertex of region R_Q^2 each use at least one edge with exactly one endpoint in Q. Thus there exists a neighboring subsquare Q' of Q in S such that more than $B_S/4$ trees $\overline{T}_i^f$ contain an edge incident to Q and Q'. It follows that S is invalid and the algorithm discards all request nodes in S and all multicasts whose source is in S. Thus, none of the trees $\overline{T}_i^f$ contains a request node or source in S. Contradiction. ∎

Claim 12 *Let S and S' be a pair of adjacent squares. For each multicast i with edge $(x_S, x_{S'}) \in \overline{T}_i^f$ the algorithm can assign a unique horizontal or vertical straightline connection between $r_{S,i}$ and $r_{S',i}$.*

Proof. Since each ring is assigned to at most one multicast, there are at most two multicasts that potentially want to use a straightline extension, namely one from ring r in S and one from ring r in S'. Since there are two "corners" of ring r, there are two straightline extensions from ring r between S and S' and hence each of the two multicasts can be assigned a unique one. ■

Claim 13 *For each gate vertex q of multicast i in a valid subsquare Q the algorithm can assign a unique horizontal or vertical connection between q and $r_{S,i}$, where S is the square containing Q.*

Proof. Since the subsquare Q of q is valid, the edge (q, x_S) belongs to at most one multicast and the horizontal or vertical connection between q and any ring of R_S^1 can be uniquely assigned to this multicast. ■

Claim 14 *For every neighboring pair (Q_1, Q_2) of subsquares and for each multicast i the algorithm can assign a unique horizontal or vertical connection between $r_{Q_1,i}$ and $r_{Q_2,i}$.*

Proof. The same proof as for Claim 12 applies. ■

For the following two claims, let i be a multicast, and let p either be a request node in $\overline{T}_i^f$ or a gate node such that a request node p' of $\overline{T}_i^f$ in S_p connects through p to s_i. Let Q be the subsquare of p. If p is a request node, let e_p be the edge closest to p and incident to Q on the path from p to s_i in $\overline{T}_i^f$. If p is a gate node, let e_p be the edge closest to p and incident to Q on the path from p to p'.

Claim 15 *The algorithm can assign a unique horizontal or vertical connection between e_p to $r_{Q,i}$.*

Proof. If p is a request node, then Q is valid since otherwise p would not belong to $\overline{T}_i^f$. If p is a gate node, then Q is valid since otherwise the request node connecting through p would have been discarded.

It follows that each edge in Q belongs at most one tree $\overline{T}_i$, and the algorithm assigns the straightline extension from e_p to $r_{Q,i}$ only to this multicast. ■

Claim 16 *If the path in $\overline{T}_i$ from p to e_p or to $g(p)$ is contained in R_Q^2, it is edge-disjoint from any other multicast tree.*

Proof. The same argument as in Claim 15 shows that each edge in Q belongs at most one tree $\overline{T}_i$. Thus the edges on the path in $\overline{T}_i$ from p to e_p or to $g(p)$ are used only for one multicast. ■

4.1.3 Proof of the approximation ratio

We prove that the algorithm gives an $O((\log\log n)^2)$ approximation using the following steps: Recall that S is an upper bound of the optimal fractional solution. Lemma 17 shows that the value S is within a constant factor of the optimal integral solution on set of requests $\mathcal{L}$. Lemma 18 shows that the expected profit of the trees $\overline{T}_i$ is at least sS/e^2. Lemma 19 shows that the capacity of no edge $(x_S, x_{S'})$ is violated with at least constant probability. Lemma 21 shows that $E[|\mathcal{L}^1|] \geq sS/e^2$. Lemma 22 shows that $E[|\mathcal{L}^2|] \geq |\mathcal{L}^1|/25$, Lemma 23 shows that $E[|\mathcal{L}^3|] \geq |\mathcal{L}^2|/2$, Lemma 24 shows that $E[|\mathcal{L}^4|] \geq |\mathcal{L}^3|/4$, Lemma 25 shows that $E[|\mathcal{L}^5|] \geq |\mathcal{L}^4|/25$, and finally Lemma 26 shows that $E[|\mathcal{L}^6|] \geq |\mathcal{L}^5|/4$.

Lemma 17 *Let OPT be the optimal integral solution on set of requests $\mathcal{L}$. Then $OPT \leq 37S$.*

Proof. Edge $e = (x_S, x_{S'})$ has capacity $c(e) = \lfloor \log n \rfloor$ in the fractional packing formulation on network G'. The border between square S and S' has size $9B + 1 \leq 37\lfloor \log n \rfloor$. The optimal integral solution for the multicast routing problem on set of requests $\mathcal{L}$ can be transformed into a feasible solution for the fractional packing problem assigning capacity $1/37$. The claim of the lemma follows. ■

Lemma 18 *Let $b = \sum_{\overline{T}_i} b(\overline{T}_i)$. Then $E[b] \geq sS/e^2$.*

Proof. Note that $E[b]$ equals the expected number of request nodes contained in the trees $\overline{T}_i$.

Let t be a request node of $\mathcal{L}_i$ and let $\overline{x}_{t'_i} = \sum_{T \in \mathcal{T}_i : t'_i \in V(T)} x^*_T$. We show below that

$$Pr[t'_i \in V(\overline{T}_i)] \geq \frac{s\overline{x}_{t'_i}}{e^2}.$$

Summing over all request nodes t'_i it follows that

$$E[b] \geq \sum_i \sum_t \frac{s\overline{x}_{t'_i}}{e^2} = \sum_i \sum_{T \in \mathcal{T}_i} b(T) s x^*_T / e^2 = sS/e^2.$$

We are left with showing the bound on $Pr[t'_i \in V(\overline{T}_i)]$. Let $\{T_i : x_{T_i} > 0, i = 1, .., r\}$ be the set of trees in the fractional solution spanning vertex t'_i. If $r = 1$, the claim is true, since $Pr[t'_i \in V(\overline{T}_i)] = s\overline{x}_{t_i}$. Consider $r \geq 2$. The probability that vertex t'_i is in graph $\overline{G}_i$ is lower bounded by $\sum_{j=1,\ldots,r} Pr[\overline{x}_{T_j} = 1 \text{ and} \overline{x}_{T_i} = 0, \forall i \neq j] = \sum_{j=1,\ldots,r} s x^*_{T_j} \prod_{i \neq j} (1 - s x^*_{T_i})$. This expression is minimized when all the probabilities have equal value $x^*_{T_j} = \frac{\overline{x}_{t'_i}}{r}$. We then have $Pr[t'_i \in V(\overline{T}_i)] \geq r(\frac{s\overline{x}_{t'_i}}{r})(1 - \frac{s\overline{x}_{t'_i}}{r})^{(r-1)} \geq s\overline{x}_{t'_i} exp - (s\overline{x}_{t'_i} \frac{r}{r-1}) \geq s\overline{x}_{t'_i} e^{-2}$, since $s\overline{x}_{t'_i} \leq 1$. ■

Lemma 19 *For every edge* $e = (x_S, x_{S'})$ *of* G' *with capacity* $c(e) = \lfloor \log n \rfloor$, $Pr[|\{\overline{T}_i : e \in E(\overline{T}_i)\}| > c(e)] \leq n^{-\log\log\log n}$.

Proof. Let $X(e) = |\{T : e \in E(T), \overline{x}_T = 1\}|$. Since $X(e) \geq |\{\overline{T}_i : e \in E(\overline{T}_i)\}|$ it suffices to show the bound for $Pr[X(e) > c(e)]$.

We prove the claim using the following version of Chernoff bounds [20]:

Proposition 20 *Let* $X_1, X_2, ..., X_n$ *be independent Poisson trials such that , for* $1 \leq i \leq n$, $Pr[X_i = 1] = p_i$, *where* $0 < p_i < 1$. *Then,for* $X = \sum_{i-1}^{n} X_i$, $\mu = \mathrm{E}[X] = \sum_{i=1}^{n} p_i$, *and any* $\delta > 0$,

$$Pr[X > (1+\delta)\mu] < \left[\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right]^{\mu}.$$

We need to bound $Pr[X > (1+\delta)\mu] = Pr[X(e) > c(e)]$, for which we set $\delta = \frac{c(e)}{\mu} - 1$. For the expected value μ we have $\mu = \sum_{T \in \mathcal{T}: e \in E(T)} x_T^* \leq sc(e)$. Applying Chernoff bounds we obtain:

$$Pr[X(e) > c(e)] \quad < \quad \left[\frac{e^{(\frac{c(e)}{\mu}-1)}}{\left(\frac{c(e)}{\mu}\right)^{\frac{c(e)}{\mu}}}\right]^{\mu} \leq \frac{1}{\left(\frac{c}{e}(\log\log n)^2\right)^{\lfloor \log n \rfloor}} \leq n^{-\log\log\log n}$$

■

Lemma 21 $E[|\mathcal{L}^1|] \geq sS/2e^2$

Proof. By Lemma 19, the capacity of an edge is violated with probability at most n^{-3}, since $\log\log\log n \geq 3$. No edge $(x_S, x_{S'})$ of network G' is then violated with probability at least $1/n$. The claim follows. ■

Lemma 22 $E[|\mathcal{L}^2|] \geq |\mathcal{L}^1|/25$,

Proof. Follows from Lemma 9. ■

Lemma 23 *There is a suitable choice of c such that for every subsquare Q,* $Pr[\exists e \in V(R_Q^2) : |\{\overline{T}_i : e \in E(\overline{T}_i)\}| > 1] \leq 1/2$.

Proof. Region R_Q^2 contains $O((\log\log n)^2)$ edges. Consider edge e of subsquare Q and consider the solution of the fractional packing problem. Every edge e of R_Q^2 is assigned to tree T with probability sx_T^*. Since the fractional solution is feasible, $\sum_{T \in \mathcal{T}} x_T^* \leq 1$. The "expected capacity" of edge e assigned to a tree $\overline{T}_i$ is then bounded by s. Using the Markov inequality we have that the probability that the capacity of an edge e is exceeded is bounded by s. With a suitable choice of c, the probability that the capacity of at least one edge of R_Q^2 is violated is bounded by $sO((\log\log n)^2) \leq 1/2$. ∎

Lemma 24 $E[|\mathcal{L}^4|] \geq |\mathcal{L}^3|/4$,

Proof. We show that the probability that a square is invalid is at most $1/2$. Let Q and Q' be neighboring subsquares. Let $\mathcal{T}_{QQ'}$ be the set of trees containing an edge between Q and Q' and having $x_T^* > 0$. Let $\mu = \sum_{T\in\mathcal{T}_{QQ'}} x_T^* \leq (9B_S+1)s$ and let X be the number of trees T in $\mathcal{T}_{QQ'}$ with $x_T = 1$. X is clearly an upper bound to the number of distinct trees crossing the border between two subsquares. Choosing a suitable large constant c, we prove in a way similar to Lemma 19 that $Pr[X > B_S/4] \leq ((\log n)^{-\log\log\log n})$. Since there are $O(\log n)^2$ adjacent subsquares in a square S, and $\log\log\log n \geq 3$, with probability at least $1-\Omega(1/\log n)$ no border between two adjacent subsquares is crossed by more than $B_S/4$ distinct trees. A request t of multicast i of $\mathcal{L}^3$ is in $\mathcal{L}^4$ if square S_t and square S_{s_i} are valid. The theorem then follows. ∎

Lemma 25 $E[|\mathcal{L}^5|] \geq |\mathcal{L}^4|/25$,

Proof. Let (s,t) be a request of $\mathcal{L}^4$ and let q be a gate vertex in S_s. Request (s,t) belongs to $\mathcal{L}^5$ if $q \in R^2_{S_s}$ and $g(p) \in R^2_{S_t}$. By Lemma 9 this happens with probability at least $1/25$. ∎

Lemma 26 $E[|\mathcal{L}^6|] \geq |\mathcal{L}^5|/4$,

Proof. As shown by Lemma 23 a subsquare is invalid with probability at most $1/2$. A request t of multicast i of $\mathcal{L}^5$ is in $\mathcal{L}^6$ if the subsquare $S_{g(t)}$ and subsquare $S_{g(s_i)}$ are valid. ∎

We then conclude with the following:

Lemma 27 *There exists an $O((\log\log n)^2)$ approximation algorithm for long requests for the multicast routing problem on meshes.*

4.2 Short requests

We complete the approximation algorithm for meshes describing the algorithm for short requests.

We separately consider every square S of G and set of requests $\mathcal{S}_S = \{(t,s) : S_s = S\}$. We apply recursively the algorithm for requests $\mathcal{S}_S$ in a square S. Square

S is partitioned in subsquares of size $O(\log\log n)$. Denote by Q the generic subsquare. Those requests of $\mathcal{S}_S$ that are considered long requests within square S are dealt by the algorithm of the previous section. Those requests of $\mathcal{S}_S$ that are considered short requests within a subsquare Q are dealt with exhaustive search of the best solution for the multicast routing problem in submesh Q of size $O(\log\log n)$, which takes time polynomial in n.

We then conclude with the following lemma for short requests.

Lemma 28 *There exists an $O((\log\log\log n)^2)$ approximation algorithm for short requests for the multicast routing problem on meshes.*

This shows the desired theorem.

Theorem 29 *The above algorithm gives an $O((\log\log n)^2)$-factor approximation for multicast routing on unit-capacity meshes.*

5 The online algorithm for meshes

We propose an algorithm with polylogarithmic competitive ratio on meshes. As in the unicast algorithm for meshes [14] the algorithm 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 our 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 [12] 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 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

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

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.

5.1 The first stage of the algorithm

Let $G = (V, E)$ denotes 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 $f = n$ div $\lfloor \log n \rfloor$ and let $f_1 = n$ mod $\lfloor \log n \rfloor$. We partition the mesh into f^2 submeshes of logarithmic size. The partition of the mesh is obtained 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 a 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. Each ring, except the innermost, forms a rectangle of nodes, the innermost ring either forms a rectangle or a line of nodes. A *corner* of a ring is a corner of the rectangle or an endpoint of the line. The first ring is also called the *border* 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 of the algorithm selects for each square one of its regions at random. The selected region is used to route paths “through” the square. All requests whose request node or source belong to the selected region are rejected to guarantee that they do not overlap with the paths routed through the selected region. Additionally each ring of the square is randomly *dedicated* either to sources or to request nodes and requests not following the dedication are rejected. Again the idea is to guarantee that requests with a source in a ring does not overlap with a request with a request node in the same ring.

The details of the first stage 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 the original sequence of requests be called $\mathcal{A}$ and let the remaining set of requests be called $\mathcal{C}$. Denote by $OPT(\mathcal{A})$ the sequence of requests accepted by the optimal algorithm over a set of requests $\mathcal{A}$.

Lemma 30 *For any input sequence σ, $E(|OPT(\mathcal{C})|) \geq \frac{1}{12}|OPT(\mathcal{A})|$.*

Proof. Let us consider the probability that a request (t, s) of $OPT(\mathcal{A})$ belongs to $\mathcal{C}$. Assume first that both t and s are within the same square. Then (t, s) belongs to $\mathcal{C}$ if (a) both are outside the selected region, and (b) the ring of s is dedicated to sources and the ring of t is either dedicated to requests or equal to s's ring. The condition (a) is fulfilled with probability at least $1/3$, condition (b) with probability at least $1/4$. Thus, (t, s) belongs to $\mathcal{C}$ with probability at least $1/12$.

Assume next that t and s belong to different squares. Then (t, s) belongs to $\mathcal{C}$ if (a) both are outside the selected region, and (b) the ring of s is dedicated to sources and the ring of t is dedicated to requests. The condition (a) is fulfilled with probability at least $4/9$, condition (b) with probability at least $1/4$. Thus, (t, s) belongs to $\mathcal{C}$ with probability at least $1/9$.

This shows that $E[|OPT(\mathcal{A}) \cap \mathcal{C}|] \geq \frac{1}{12}|OPT(\mathcal{A})|$. Since $|OPT(\mathcal{C})| \geq |OPT(\mathcal{A}) \cap \mathcal{C}|$, the result follows. ■

5.2 The second stage of the algorithm.

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.

Recall that each square contains $13B$ concentric rings. 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. These tests guarantee that the following additional invariants are maintained.

(I2) No two request nodes of accepted requests belong to the same square.

(I3) No two sources of accepted requests of different multicasts belong to the same ring.

(I4) No two sources of accepted *long* requests of different multicasts belong to the same square.

(I5) No two sources of accepted *short* requests from the same square belong to the same square.

(I6) No two request nodes of accepted long requests belong to the same square.

Invariant (I6) follow from invariant (I2).

5.2.1 Long requests

The algorithm for long requests decides whether to accept or reject a request in four steps. Each step rejects the request if certain conditions are not fulfilled. 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:

(i) no multicast with source on the ring of s is already selected for *short* requests;

(ii) no multicast with source in S_s is already selected for *long* requests;

(iii) a coin toss with success probability $1/(4B)$ is successful; and

(iv) if s is in R^3 then the largest ring of R^3 in S_s is dedicated to sources.

When a multicast becomes selected for long requests, up to two of the rings in S_s are assigned to the multicast: (a) the ring containing s is assigned to the multicast; and (b) if s is in R^3, the largest ring of R^3 in S_s.

The second ring is needed for the following reason: When routing a long request to the border of S_s the algorithm needs to be able to use any available crossbar row or column. However, if s belongs to the central region, the ring of s does not intersect all crossbar rows and columns. Thus, we route the path from the ring of s to the largest ring of R^3 and from this ring the algorithm can connect to any crossbar row or column. To avoid that the connection between the ring of s and the largest ring of R^3 overlaps with any other multicast tree, the two rings are connected along a "straightline extension" of the internal ring assigned to s, which will guarantee that the connection does not use an edge of a short request (see Section 5.2.2).

However, the connection of s from the central region to the largest ring of R^3 overlaps a crossbar row or column that may be used to connect a request node $t' \in R^3$ of a different long request (t', s') to the border. Our algorithm will reject all long requests from a request node in the square if (i) a request to a source node in the square has been accepted, or (ii) if a long request to a request node in the square has been accepted. This implies that if a connection from a source in R^3 to the largest ring of R^3 exists then no further long requests are accepted. Thus, the problem is restricted to the situation when the long request from a request node in R^3 was accepted and *afterwards* a request with source in R^3 appears. We avoid the intersection by selecting an appropriate straightline extension connecting a ring of the central region to the largest ring of R^3 that avoids the crossbar row or column used by the long request with request node in R^3.

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 has been 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 of the request. 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.

Step 4 guarantees that the following invariant is maintained:

(I7) If a request (t, s) is added to $\mathcal{L}_3$, then a node of S_t is connected to s.

We show next how the algorithm routes a request of $\mathcal{L}_3$. For each such request s denotes the source node and v denotes the node of S_t to which the request is routed, i.e., either $v = t$ or $v = u$, where u is a node on the assigned ring of the selected region. For every such request we are given by MC a path in G' consisting of a sequence of neighboring squares. Let $S_1, S_2, \ldots, S_p$ be the sequence of squares such that $S_1 = S_t$ and S_p contains a node x of the multicast tree of s. If S_p contains s, let $x = s$. Otherwise, if S_p contains an accepted request node not in R^3, let x be this node. Otherwise, as we show in Lemma 34, a ring of the selected region was assigned to the multicast and at least one node on the ring is connected to s. In this case let x be this node.

Each request of $\mathcal{L}_3$ is routed from v to the border between S_1 and S_2, from there to the border between S_2 and S_3 and so forth, until finally to the border between S_{p-1} and S_p and from there to x. Let P denote the resulting path from v to x. We describe next each step in detail.

To route paths between two neighboring squares we reserve the *crossbar* of rows $6B + 1, \ldots, 7B$ and columns $6B + 1, \ldots, 7B$. All rows and columns in the crossbar cross the central region of the square. By Lemma 34 there is an unassigned crossbar row resp. column for each accepted request.

Case 1: From v to the border between S_1 and S_2: Wlog the border between S_1 and S_2 is a row. We assign an unassigned crossbar column and the ring of v to the multicast.

If v does not belong to the central region, we route P along the ring of v until it reaches the point on its assigned crossbar column closest to the border. At this point P is routed along the assigned column until it reaches the border.

If v belongs to the central region, we route P on the ring of v until a corner of the ring is reached. There P continues straight to the assigned ring of R^3. From the assigned ring of R^3 we continue as above.

Case 2: From one border of a square to another border of the square: Wlog the entering border is a column and the exiting border is a row. One unassigned ring of the selected region is assigned to the multicast. We additionally assign one of the unassigned crossbar columns between the current square and the next square to P.

Path P follows the entering row until it intersects the assigned ring. Then P is routed along the ring until it reaches the intersection point with the assigned column closest to the exiting border. There P switches to the assigned column until it reaches the border.

Case 3: From the border of S_p to the node x: Wlog the border between S_{p-1} and S_p is a row. There are three cases to consider: (i) $x = s$, (ii) $x \notin R^3$ is a request node of an accepted request of the multicast, or (iii) x is a node connected to s that belongs to a ring of the selected region assigned to the multicast.

(i) If s does not belong to R^3, we route P along the entering column until the ring of s is reached. At this point P is routed along the ring of s to s. If s belongs to R^3, P follows the entering column until the largest ring of R^3 is reached. Then, P is routed along this ring until it reaches a point from which a row or column is available that (1) connects straight to a corner of the ring of s and (2) does not intersect any existing multicast tree. By Lemma 34 such a row or column exists. Path P routes along this row or column to the ring of s and along the ring of s to s.

(ii) When the request at $x \notin R^3$ was accepted, the ring of x was assigned to the multicast. We route P along the entering column until the ring of x is reached. At this point P is routed along the ring of x to x.

(iii) We route P along the entering column until the assigned ring of the selected region is reached. At this point P is routed along the ring to x.

We need to show that this routing is always possible. We first show two properties of MC.

Lemma 31 *The maximum number of paths routed by MC between two adjacent squares is B.*

Proof. Each edge in G' has capacity B and MC does not violate the capacity constraints. ■ We say a path in G' is *routed through* a square S if S is an internal node of the path.

Lemma 32 *Let S be a square. There are at most $2B$ paths of MC routed through S. If a request node of a request in $\mathcal{L}_2$ belongs to S, then at most $2B-1$ paths of MC are routed through S.*

Proof. Every path routed through a square consumes two units of bandwidth on the edges incident to the vertex of G' associated with the square. Additionally, a long request consumes one unit of bandwidth. The overall bandwidth on the edges incident to a vertex of G' is $4B$. ■

We prove with the next two lemmata that all paths constructed for different multicasts are edge-disjoint.

Lemma 33 *Invariants (I4) and (I6) are maintained.*

Proof. Condition *(ii)* of being selected for long requests guarantees that invariant (I4) is maintained. Step 1 of the algorithm guarantees invariant (I6). ■

Lemma 34 *Let (t, s) be a request of $\mathcal{L}_3$ and let (v, x) be the corresponding pair of nodes that has to be connected by a path. Then the above algorithm succeeds in constructing a path from v to x. Furthermore, this path does not share any edge with paths constructed for requests in $\mathcal{L}_3$ for different multicasts.*

Proof. We first show that the algorithm succeeds in constructing a path from v to x. We need to show that (a) whenever the algorithm tries to assign an unassigned ring of the selected region to the multicast, such a ring is available; (b) if $s \notin S_p$ and no request node of this multicast in R^1 or R^2 was accepted then s is connected to a node x on a ring of the selected region of S_p that is assigned to the multicast; (c) whenever the algorithm tries to assign an unassigned column or row between two neighboring squares, such a column or row is available; (d) whenever the algorithm tries to connect the largest ring of R^3 with vertex s in R^3, such connection is possible without overlapping with a different multicast tree.

(a) The selected region consists of $2B$ rings. The algorithm tries to assign a ring of the selected region for each request of $\mathcal{L}_3$ with request node in the square and for each request routed through a square. Step (1) of the algorithm guarantees that at most one long request with request node in a square belongs to $\mathcal{L}_3$. If this happens, Lemma 32 shows that at most $2B-1$ paths are routed through the square, otherwise at most $2B$ paths are routed through the square. Thus, in either case there are enough rings in the selected region.

(b) If $s \notin S_p$ but MC routed the path from S_t to S_p, then MC connected S_p to S_s during an earlier request. Thus, an earlier request (t', s) of the same multicast was added to $\mathcal{L}_2$ and, since the multicast is selected for long requests, was also added to $\mathcal{L}_3$. Thus, a ring r of the selected region of S_p is assigned to the multicast.

If $(t', s) \in \mathcal{L}_3 \setminus \mathcal{L}_4$, then a node u of r is connected to s. If $(t', s) \in \mathcal{L}_4$, then by assumption $t' \in R^3$. Thus, the crossbar row or column on the path from t' to the border of S_p also contains a node of r.

(c) The crossbar consists of B rows and columns. By Lemma 31 there are at most B requests that have to cross the border between any two neighboring squares. Thus, it is guaranteed that whenever a path needs to be routed between two neighboring squares an unassigned row or column is available.

(d) The ring of s has at least one corner, and thus there are at least two possible rows and two possible columns through which the largest ring of R^3 can be connected with the ring of s. At most one long request (t', s') with t' in R^3 has been accepted in S_s, potentially using one of these crossbar rows or columns. Thus, there are still at least three possible ways to connect to the ring of s.

Let S be a square. We show next that the path from v to x does not share any edge in S with paths constructed for requests in $\mathcal{L}_3$ for different multicasts. The lemma follows.

Note first that no ring of S shares an edge with a crossbar row or column or a straightline segment connecting two rings. To complete the proof that no two paths of different multicasts share an edge in S, it then suffices to show that the rings of the two paths are edge-disjoint and that the straightline segments of the two paths are edge-disjoint. The straightline segments connect the various rings of a path and a ring with the border of S.

A path that is routed through S consists of an entering crossbar row or column, a part of a ring in the selected region, and a departing crossbar row or column. Both crossbar pieces go from the border to the selected region. By points (a) and (c) and the observation that the straightline extensions belong to R^3, which is in this case not the selection region, all these three pieces are disjoint from the paths used by any other request path with edges in S. Thus, any path routed through S is edge-disjoint from any path of a different multicast in S.

We are left with showing that no two paths of long requests starting or ending in S share an edge in S. Let P and P' be two such paths belonging to different multicasts and let y and y' be their endpoints in S. By invariants (I4) and (I6), y and y' can only be a source node and a non source node. Wlog., y is a source and y' is a non-source node. Node y belongs to a ring that is assigned to sources and does not belong to the selected region, node y' either belongs to the selected region or to a ring that is assigned to requests. In either case, the rings of y and of y' are edge-disjoint. Furthermore, the largest ring of R^3 might have been assigned to path P and in this case is dedicated to sources. A different ring of R^3, which is dedicated to request nodes, might have been assigned to path P'. Again these rings are edge-disjoint.

By point (c) the crossbar rows or columns assigned to P and P' are different and, thus, edge-disjoint. Since the straightline extensions start at different rings, they are edge-disjoint. We are left to show that the straightline extensions connecting the rings assigned to y and y' are edge-disjoint with the crossbar rows or columns used in P and P'. If neither y nor y' belong to R^3, no straightline extension is necessary. If only one between y and y' belongs to R^3, one straightline extension exists, belonging completely to R^3, and the crossbar row or column of the other path does not reach R^3. Thus, of P and P' are edge-disjoint. If both y and y' belong to R^3, note that by Step 1 P' was accepted and routed before P. By point (d) above the straightline extension of P does not overlap with P'.

It follows that P and P' are edge-disjoint. ■

5.2.2 Short Requests

Recall that a request (t, s) is short if a node x in S_t belongs to the multicast tree of s. The algorithm for short requests decides whether to accept or reject a request in three steps. Each step rejects the request if certain conditions are not fulfilled. The requests which are not rejected after step i, $i = 1, 2, 3$ form a sequence $\mathcal{S}_i$.

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*. Note that this decision will only affect short requests with $S_t = S_s$, other short requests of the same multicast are accepted or rejected independent of this decision.

A multicast with source s is *selected for short requests* if all of the following conditions are fulfilled at the time of the test:

(i) no short request of a multicast with source in S_s has been added to $\mathcal{S}_2$;

(ii) a coin toss with success probability $1/2$ is successful.

Whenever a multicast is selected for short requests, the ring of s is assigned to the multicast.

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. (a) If a short request with request node in S_t has been accepted then reject (t, s) and stop. (b) If a long request with request node in S_t has been added to $\mathcal{L}_3$, reject (t, s) and stop. (c) 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 $\mathcal{S}_3$ and assign the ring of t to the multicast. If either t or y, but not both, belong to the central region of S_t, and the other vertex does not belong to R^3, assign one of rings $4B+1$ to $6B$ of S_t dedicated to requests to the multicast of s.

We describe next the routing part of the algorithm. Let P denote the path used by an accepted request (t, s) to connect t to y. If t and y belong to the same ring, simply connect them by a path along the ring.

If t and y do not belong to the same ring, let a denote the vertex of $\{t, y\}$ that belongs to a ring of S_t with larger index than the other vertex. Let b denote the other vertex. If a is outside the central region then P starts at a and follows the ring containing a until a corner of the ring is reached. Then, P continues along a straightline extension of the ring to the ring of b and from there along the ring of b to b.

If a is in the central region and b is not, then P starts at a, follows the ring of a until a corner of the ring is reached. There P continue straight to the ring in the external $2B$ rings of R^3 assigned to the multicast. Then, path P continues as described above, i.e., it routes to a corner of the ring, extends straight from there until it reaches the ring of b, and follows the ring of b to b.

If both a and b belong to the central region, P is routed from a along the ring of a until it reaches a corner of the ring. From there it continues straight until it reaches the ring of b and it follows the ring of b to b.

Lemma 35 *Invariant (I1), (I2), (I3), and (I5) hold.*

Proof. Step 1(a) and 1(b) of the algorithm for short requests and step 1 of the algorithm for long requests ensure that invariant (I2) holds.

Condition *(i)* and *(ii)* of being selected for a long request and condition *(i)* of being selected for a short request ensure invariant (I3).

Condition *(i)* of being selected for a short request ensure invariant (I5).

Finally, we show that invariant (I1) holds. The routing phase guarantees that edges on a ring are only used by the multicast to whom the ring is assigned. Thus, we need to show that each ring is assigned to at most one multicast. A ring is assigned to a multicast during request (t, s) (a) if it contains s, (b) if it contains t, (c) if it belongs to the selected region, (d) if it is the largest ring of S_s and dedicated to sources or, (e) if it is one of rings $4B + 1$ to $6B$ in S_t and dedicated to requests.

During a request (t, s) a type (c) assignment is only done if the ring was not previously assigned. Thus, we only need to show that type (a), (b), (d), and (e) assignments do not assign previously assigned rings. Recall that each ring is either dedicated to a source or to a request node. A request is rejected if its source lies on a ring dedicated to request nodes or its request node lies on a ring dedicated to sources and its source does not lie on this ring. In the following let r denote the ring that is assigned.

Type (a): The ring r of s is dedicated to sources. When the ring r of s is assigned to the multicast with source s, the multicast with source s is selected for long or short requests. If it becomes selected for long requests, by condition *(i)* and *(ii)* for selection for long requests no multicast with source on r is already selected for short or long requests. If the multicast becomes selected for short requests, note that $S_s = S_t$. By by condition *(i)* for selection for short requests and step 1c of the algorithm for short requests no multicast with source in S_s is already selected for short or long requests. Thus, r was not assigned to any other multicast with source on r. Furthermore, r does not belong to the selected region. Thus, r has not been assigned because of type (a), (b), (c), or (e) assignments. Finally a type (d) assignment can happen only if a previous source of a long request belongs to S_s. This is impossible by condition *(ii)* of being selected for a long request if the current request (t, s) is a long request, and by step (1c) of the algorithm for short requests if the current request (t, s) is a short request.

Type (b): The ring r is dedicated to request nodes and does not belong to the selected region. Thus, r was not previously assigned with type (a), (c), and (d) assignments. Since (t, s) is an accepted short request, no earlier request with request node in S_t was accepted by invariant (I2). Thus, r was not previously assigned by type (b) or (e) assignments.

Type (d): In this case ring r belongs to S_s and is dedicated to sources. Furthermore, $s \in R^3$, i.e., R^3 is not selected and the current multicast is selected for long requests. It follows that no multicast with source in S_s is already selected for long requests and no multicast with source on r was previously selected. The former implies that no type (d) assignment has happened to r before, the latter implies that no type (a) assignment has happened to r before. Since r does not belong to the selected region and is dedicated to sources, type (b), (c), and (e) assignments are not possible.

Type (e): In this case ring r belongs to S_t and is dedicated to request nodes. Furthermore, R^3 is not selected and $t \in R^3$. It follows that type (a), (c), and (d) assignments are not possible. By invariant (I2) no earlier request with request node in S_t was accepted. Thus, no previous type (b) or (e) assignment can have occurred. ■

Lemma 36 *No two paths constructed for different multicasts share an edge.*

Proof. Let S be a square. We show that no two paths P and P' constructed for different multicasts share an edge in S. By Lemma 34 no two paths assigned to long requests share an edge. Thus, we can assume that P is a path constructed for a short request. Let (t, s) be the request leading to the construction of P and let (t', s') be the request leading to the construction of P'.

By step (1a) of the algorithm, only one short request is accepted in a square. By step (1b) of the algorithm, if a long request from a request node in a square has been added to $\mathcal{L}_3$, no short request from that square is accepted. By step (1c) of the algorithm if a long request with source node in a square was added to $\mathcal{L}_3$, no short request from that square is accepted. Thus, it follows that when (t, s) is accepted, no previously accepted request has either its source or its request node in S_t. Thus, either P' routes through S_t or (t', s') is a long request accepted after (t, s). By step 1 of the algorithm for long requests it additionally follows that $S_t \neq S_{t'}$, i.e., that $S_t = S_{s'}$.

Lemma 35 shows that we maintain invariant (I1), i.e., the edges of a given ring are used by at most one multicast tree. Thus, no two different multicast trees share an edge on a ring. As argued in Lemma 34, no ring of S shares an edge with a crossbar row or column or a straightline segment connecting two rings.

We are left with proving that the subpaths of P' used to connect the (at most 2) rings of P' and the border(s) of the square are edge-disjoint with the (at most 2) subpaths of P that connect the (at most 3) rings used by P.

Each subpath used by P is a straightline extension either (a) from a corner of a ring outside the central region to a more external ring, (b) from a corner of a ring of the central region to a ring of R^3 with smaller index, or (c) from a corner of a ring of the central region to another ring of the central region. Each subpath used by P' is either (i) a straightline extension from a corner of a ring of R^3 to ring $4B + 1$, or (ii) a crossbar row or column from a vertex of ring $4B + 1$ or of a ring outside R^3 to the border of the square. Since they start from different rings, every type (i) subpath of P' is edge-disjoint with all type (a), (b), or (c) subpaths used by P. Every type (ii) subpath of P' is edge disjoint with any type (a) subpath of P, since a type (a) subpath does not use edges on the crossbar. Every type (ii) subpath of P' is also edge-disjoint with any type (b) or (c) subpath of P, since type (b) or (c) subpaths only use edges with both endpoints in R^3.

Thus, P and P' are edge-disjoint. ∎

5.2.3 The analysis

Let $\rho = \log n(\log n + \log\log \mathcal{M}) \log \mathcal{M}$. Recall that MC achieves a competitive ratio of $O(\rho)$. We prove in this section that 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 30 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

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

it suffices to show the following results:

$|OPT(\mathcal{C}) \cap ((\mathcal{L}_0 \setminus \mathcal{L}_1) \cup (\mathcal{S}_0 \setminus \mathcal{S}_1))| \leq 48 \log^2 n E[|ON(\mathcal{C})|]$ (Lemma 39),

$|OPT(\mathcal{C}) \cap \mathcal{L}_1| \leq O(\rho \log n) E[|ON(\mathcal{C})|]$ (Lemma 40), and

$|OPT(\mathcal{C}) \cap \mathcal{S}_1| \leq O(\log^2 n) E[|ON(\mathcal{C})|]$ (Lemma 42).

We first need to show the following claim. We assume here that each node can receive at most one request per multicast.

Claim 37 *At most $4 \log^2 n$ requests with request node in a given square can be accepted in a solution.*

Proof. Each node in a square is incident to four edges and thus can belong to at most four multicast trees. Thus at most $4 \log^2 n$ requests with request node in a square can be accepted by a solution. ■

Claim 38 *Every request of $\mathcal{L}_3$ is added to $\mathcal{L}_4$ with probability at least $1/2$, i.e., $E[|\mathcal{L}_4|] \geq |\mathcal{L}_3|/2$.*

Proof. Let (t, s) be such a request of $\mathcal{L}_3$. By Step 4 of the algorithm for long requests, (t, s) is not added to $\mathcal{L}_4$ if t belongs to the central region and rings $4B + 1$ to $6B$ are all dedicated to sources. If t belongs to R^3, R^3 is not the selected region, and thus every ring of R^3 is dedicated to request nodes with probability $1/2$. ■

Lemma 39 $|OPT(\mathcal{C}) \cap ((\mathcal{L}_0 \setminus \mathcal{L}_1) \cup (\mathcal{S}_0 \setminus \mathcal{S}_1))| \leq 4 \log^2 n E[|ON(\mathcal{C})|]$

Proof. We first show that $|OPT(\mathcal{C}) \cap (\mathcal{L}_0 \setminus \mathcal{L}_1)| \leq 24 \log^2 n E[|ON(\mathcal{C})|]$.

A request (t, s) of $\mathcal{L}_0$ is not added to $\mathcal{L}_1$ because (i) a long request with request node in S_t was added to $\mathcal{L}_3$ or (ii) a long request with source in S_t was added to $\mathcal{L}_3$ or (iii) a short request with request node in S_t was accepted. Consider the following charging from long requests in $\mathcal{L}_0 \setminus \mathcal{L}_1$ to squares of the mesh: (A) If a request (t, s) of $\mathcal{L}_0$ was not added to $\mathcal{L}_1$ because situation (i) or (iii) arose, the request is charged to S_t. (B) If (t, s) was not added because a type (ii) request (t', s') was added to $\mathcal{L}_3$, we charge (t, s) to $S_{t'}$. Claim 37 shows that there are at most $4 \log^2 n$ type (A) charges and at most $4 \log^2 n$ type (B) charges to a given square.

Let q be the number of squares to which at least one request of $\mathcal{L}_0 \setminus \mathcal{L}_1$ is charged. Then $|OPT(\mathcal{C}) \cap (\mathcal{L}_0 \setminus \mathcal{L}_1)| \leq 8\log^2 nq$. It suffices to prove $q \leq 3E[|ON(\mathcal{C})|]$

Every square with a type (A) charge contains the request node of an accepted request of $\mathcal{C}$. Every square with a type (B) charge contains the request node of a request in $\mathcal{L}_3$. By Claim 38, $|\mathcal{L}_3| \leq 2E[|ON(\mathcal{C})|]$. Thus, $q \leq |ON(\mathcal{C})| + 2E[|ON(\mathcal{C})|]$.

Then, $|OPT(\mathcal{C}) \cap (\mathcal{L}_0 \setminus \mathcal{L}_1)| \leq 8\log^2 nq \leq 24\log^2 nE[|ON(\mathcal{C})|]$

The same argument applies to $\mathcal{S}_0 \setminus \mathcal{S}_1$. ∎

Lemma 40 $|OPT(\mathcal{C}) \cap \mathcal{L}_1| \leq O(\rho \log n)E[|ON(\mathcal{C})|]$

Proof. The requests of $\mathcal{L}_1$ are transformed into requests on G', giving rise to a request sequence $\mathcal{L}_{1,G'}$ submitted to MC. The bandwidth on M is a factor of 13 larger than the bandwidth on G'. On the request sequence $\mathcal{L}_{1,G'}$ we compare MC on G' with an optimum algorithm on a mesh G'' whose topology is identical to G' and whose edge capacity is a factor of 13 larger. Then we compare the optimum algorithm on G'' with the request sequence $\mathcal{L}_{1,G'}$ with the optimum algorithm on M with the request sequence $\mathcal{L}_1$. Note that $ON_{MC}(\mathcal{L}_{1,G'}) = \mathcal{L}_2$.

The same arguments as in [4, 14] show that $|OPT(\mathcal{L}_{1,G''})| \leq O(\rho)E[|ON(\mathcal{L}_{1,G'})|] = O(\rho)E[|\mathcal{L}_2|]$.

Note that $|OPT(\mathcal{L}_{1,G''})| \geq |OPT(\mathcal{L}_1)|$, since routing requests on G'' is identical to routing requests in M where all edges inside a square have infinite capacity. Thus,

$$|OPT(\mathcal{C}) \cap \mathcal{L}_1| \leq |OPT(\mathcal{L}_1)| = O(\rho)E[|\mathcal{L}_2|].$$

Claim 41 below shows that $|\mathcal{L}_2| \leq O(\log n)E[|\mathcal{L}_3|]$. From Claim 38 we obtain that $|\mathcal{L}_3| \leq 2E[|\mathcal{L}_4|]$. Since $|\mathcal{L}_4| \leq |ON(\mathcal{C})|$, this concludes the proof of the lemma. ∎

Claim 41 $|\mathcal{L}_2| \leq O(\log n)E[|\mathcal{L}_3|]$

Proof. A request of $\mathcal{L}_2$ is added to $\mathcal{L}_3$ if its multicast is selected for long requests. We show that a multicast is selected for long requests with probability $\Omega(1/\log n)$. This shows the claim.

A multicast is selected for long requests if conditions *(i) – (iv)* hold. Let r be the ring of of the source s of the multicast. We bound next the probability that each condition holds.

Condition (i): The first short request with source on r added to $\mathcal{S}_2$ is selected for short requests with probability at most $1/2$. None of the short requests with source on r added later to $\mathcal{S}_2$ can become selected for short requests. Thus, condition *(i)* that no source on r is already selected for short requests holds with probability at least $1/2$.

Condition (ii): The total capacity of edges incident to S_s is $4B$. Thus there exist at most $4B$ multicasts with source in S_s in $\mathcal{L}_2$. Each of them is selected with probability at most $1/(4B)$. Thus the probability that condition *(ii)* holds, i.e., that none of them is selected, is at least $(1 - 1/(4B))^{4B} \geq 1/e$.

Condition (iii): The coin toss is successful with probability $1/(4B)$.

Condition (iv): The largest ring of R^3 in S_s is dedicated to sources with probability 1/2.

Thus a multicast is selected for long requests with probability at least $1/(16eB) = \Omega(1/\log n)$. ■

Lemma 42 $|OPT(\mathcal{C}) \cap \mathcal{S}_1| = O(\log^2 n)E[|ON(\mathcal{C})|]$

Proof. Since $|OPT(\mathcal{C}) \cap \mathcal{S}_1| = |OPT(\mathcal{C}) \cap (\mathcal{S}_1 \setminus \mathcal{S}_3)| + |OPT(\mathcal{C}) \cap \mathcal{S}_3|$ and $|OPT(\mathcal{C}) \cap \mathcal{S}_3| \leq |ON(\mathcal{C})|$ it suffices to show

$$|OPT(\mathcal{C}) \cap (\mathcal{S}_1 \setminus \mathcal{S}_3)| \leq 32 \log^2 n E[|ON(\mathcal{C})|].$$

Let S be a square and let $\mathcal{C} \cap S$ denote the sequence $\mathcal{C}$ restricted to the requests with either request or source node in S. We show that for each square S that

$$|OPT(\mathcal{C} \cap S) \cap (\mathcal{S}_1 \setminus \mathcal{S}_3)| \leq 16 \log^2 n E[|ON(\mathcal{C}) \cap S|].$$

Since

$$\begin{aligned} |OPT(\mathcal{C}) \cap (\mathcal{S}_1 \setminus \mathcal{S}_3)| &\leq \sum_S |OPT(\mathcal{C} \cap S) \cap (\mathcal{S}_1 \setminus \mathcal{S}_3)| \\ &\leq \sum_S 16 \log^2 n E[|ON(\mathcal{C}) \cap S|] \\ &\leq 32 \log^2 E[|ON(\mathcal{C})|]. \end{aligned}$$

By Claim 37,

$$|OPT(\mathcal{C} \cap S) \cap (\mathcal{S}_1 \setminus \mathcal{S}_3)| \leq |OPT(\mathcal{C} \cap S)| \leq 4 \log^2 n.$$

It suffices to show that

(*) For each square S with $\mathcal{S}_1 \cap S \neq \emptyset$, $E[|ON(\mathcal{C}) \cap S|] \geq 1/4$. Consider the first request (t, s) of $\mathcal{S}_1 \cap S$. It is added to $\mathcal{S}_2$ if t and y either both belong to the central region or both not belong to the central region. Otherwise, the request is added to $\mathcal{S}_2$ if at least one of rings $4B + 1$ to $6B$ is dedicated to request nodes. This happens with probability at least $1/2$.

If $y \neq s$, then the request is added to $\mathcal{S}_3$ and the claim is proved. If $y = s$, then the request is added to $\mathcal{S}_3$ if the multicast is selected for short requests. If Condition (i) of being selected for short request is not satisfied, there exists a short request (t', s') of $\mathcal{S}_2$ with $S_{s'} = S$ for which Condition (i) of being selected for short requests was satisfied, that has been added to $\mathcal{S}_3$ with probability $1/2$. If Condition (i) of being selected for short requests holds for source s, then (t, s) is added to $\mathcal{S}_3$ with probability $1/2$. Thus, with probability $1/2$ at least one request of $\mathcal{S}_2$ is added to $\mathcal{S}_3$. This shows that if $\mathcal{S}_1 \cap S \neq \emptyset$, then a request of $\mathcal{C} \cap S$ is accepted by the online algorithm with probability at least $1/4$. ■

Lemmata 39, 40 and 42 together prove the following theorem:

Theorem 43 *There exists a* $O(\log^2 n(\log n + \log\log \mathcal{M}) \log \mathcal{M})$ *competitive algorithm for multicast routing on unit capacity meshes.*

5.3 A lower bound for the online algorithm on meshes

In [12] a lower bound of $\Omega(\log(\mathcal{M}/u)\log n)$ against an oblivious adversary is given for the online multicast algorithm in a tree, where u is the (minimum) edge capacity, $\mathcal{M}$ is the number of multicasts, and n is the size of the tree. We show here how to modify the proof to give a lower bound of $\Omega((\log(\mathcal{M}/u)\log n)/d)$ in a connected graph whose minimum degree is d. This gives a polylogarithmic lower bound for a mesh.

We assume first that all demands and all edge capacities are 1 and prove a lower bound of $\Omega(\log \mathcal{M} \log n)$. Afterwards we show how to add edge capacities to the proof.

We restrict ourself to the case that $\sqrt{n} > \log \mathcal{M}$. Let $M = \mathcal{M}/\log n$ and $N = n/\log M$. Assume that the nodes are numbered with consecutive numbers from 0 to $n-1$ such that the node with minimum degree is labeled 0. Consider the following sequence of multicast requests. There are $\log N$ phases. In the following we describe phase i, with $1 \leq i \leq \log N$. In each phase the adversary generates M new multicasts. Each multicast has source 0. Its requests consists of $\log M$ sets. The first set of requests consists of the nodes 1 to 2^i. Then the adversary flips a coin and terminates the multicast with probability 1/2. The j-th set of requests consists of nodes $2^i * (j-1) + 1$ to $2^i * j$. After processing all M multicasts a new phase starts.

Let $c(i)$ be the capacity on any edge incident to node 0 used by the online algorithm during phase i. Also, let $p(i)$ be the profit obtained by the online algorithm during the i-th phase. We use $c^*(i)$ and $p^*(i)$ to denote the corresponding quantities for the optimal algorithm.

By contradiction we show below that there exists a phase k such that $\sum_{1\le i\le k} E[p(i)] \le 2^{k+1}d/\log N$. The adversary stops the sequence of requests after this phase k. Claim 8.2 of [12] proves that during phase k the optimal algorithm can obtain a profit of at least $(\log M/4)2^k$ by accepting the multicast with highest profit in this round. This gives a lower bound of $\Omega((\log M \log N)/d) = \Omega((\log \mathcal{M} \log n)/d)$.

We are left with showing that there exists a phase k such that $\sum_{1\le i\le k} E[p(i)] \le 2^{k+1}d/\log N$. Assume by contradiction that no such phase exists. Let $S(k) = (1/2^k)\sum_{1\le i\le k} E[p(i)]$. Then for all k, $S(k) > 2d/\log N$, i.e., $\sum_{1\le k\le \log N} S(k) > 2d$. But

$$\sum_{1\le k\le \log N} S(k) = \sum_{1\le k\le \log N} (1/2^k) \sum_{1\le i\le k} E[p(i)] = \sum_{1\le i\le \log N} E[p(i)]/2^{i-1}.$$

Claim 8.1 of [12] shows that $E[p(i)] < 2^i E[c(i)]$. Since $\sum_i c(i) \le d$, it follows that $\sum_{1\le k\le \log N} S(k) < \sum_{1\le i\le \log N} 2E[c(i)] \le 2d$, which gives a contradiction.

If the edge capacity is u, each phase is repeated u times. This increases the number of multicasts by a factor of u. Thus, the same proof as above gives a lower bound of $\Omega((\log(\mathcal{M}/u)\log n)/d)$. We summarize the result in the following theorem.

Theorem 44 *No randomized algorithm for online multicast routing on a connected graph with minimum degree $d > 0$ 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] P. Alimonti, 1997. Personal communication.

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

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[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] 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. Also submitted to FOCS 98.

[11] 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.

[12] 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.

[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. Probabilistic construction of deterministic algorithms: approximating packing integer programs. *Journal of Computer and Systems Sciences*, 2(37):130–143, 1988.

[21] 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.

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

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