SIAM J. COMPUT.
Vol. 31, No. 1, pp. 86–112

ON-LINE RANDOMIZED CALL CONTROL REVISITED*

STEFANO LEONARDI†, ALBERTO MARCHETTI-SPACCAMELA†,
ALESSIO PRESCIUTTI†, AND ADI ROSÉN‡

Abstract. We consider the problem of on-line call admission and routing on trees and meshes. Previous work gave randomized on-line algorithms for these problems and proved that they have optimal (up to constant factors) *competitive ratios*. However, these algorithms can obtain very low profit with high probability. We investigate the question of devising for these problems on-line competitive algorithms that also guarantee a "good" solution with "good" probability.

We give a new family of randomized algorithms with asymptotically optimal competitive ratios and "good" probability to get a profit close to the expectation. We complement these results by providing bounds on the probability of any optimally competitive randomized on-line algorithm for the problems we consider to get a profit close to the expectation. To the best of our knowledge, this is the first study of the relationship between the tail distribution and the competitive ratio of randomized on-line benefit algorithms.

Key words. on-line algorithms, competitive analysis, randomized algorithms, call admission control

AMS subject classifications. 68Q25, 68W20

PII. S0097539798346706

1. Introduction.

The area of communication networks gives rise to a large number of on-line problems. One of the most extensively studied problems in this area is the problem of on-line assignment of virtual circuits in networks: a sequence of requests for calls is given on-line to an algorithm which has to select a virtual circuit between the communicating parties (or reject the request), obeying the network constraints (such as link capacities). Two kinds of decisions are involved: *admission control*, i.e., the choice of whether to accept the call or not, and *route selection*, i.e., the decision on the route of an accepted call. Each call is processed before any future call is known, while the algorithm has to make both decisions. The basic form of this problem (when link capacities are 1, and each call requests bandwidth 1) is an on-line version of the well-known problem of maximizing the number of edge-disjoint paths.

The off-line version of the maximum edge-disjoint path problem is known to be NP-hard on general networks [13]. The problem is solvable in polynomial time on tree networks [11] while it is still NP-hard on mesh networks [17]. Kleinberg and Tardos [16] give an $O(1)$ approximation algorithm for meshes and densely embedded nearly Eulerian graphs. For general graphs an $O(\sqrt{m})$ approximation is given in [14], where m is the number of edges in the graph. Recently, an $\Omega(n^{1/2-\epsilon})$ inapproximability lower bound for the maximum edge-disjoint path problem on general graphs was proved in [12] (where n is the number of nodes in the graph).

*Received by the editors November 6, 1998; accepted for publication (in revised form) February 21, 2001; published electronically June 5, 2001. A preliminary version of this paper appeared in *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, 1998, pp. 323–332.

http://www.siam.org/journals/sicomp/31-1/34670.html

†Dipartimento di Informatica Sistemistica, Università di Roma "La Sapienza," via Salaria 113, 00198-Roma, Italia (leon@dis.uniroma1.it, alberto@dis.uniroma1.it, presciutti@dis.uniroma1.it). The work of these authors was partly supported by EU ESPRIT Long Term Research Project ALCOM-IT under contract 20244 and by Italian Ministry of Scientific Research Project 40% "Algoritmi, Modelli di Calcolo e Strutture Informative."

‡Computer Science Department, The Technion, Technion City, Haifa 32000, Israel (adiro@cs.technion.ac.il).

Algorithms for on-line maximization problems are usually analyzed in terms of their *competitive ratio* [20], i.e., the worst case, over all input sequences, of the ratio between the benefit obtainable by an off-line optimal algorithm (that knows the whole sequence in advance), and the benefit obtained by the on-line algorithm. It is easy to see that greedy strategies for the problems we consider here are noncompetitive; hence more sophisticated strategies are necessary. Awerbuch, Azar, and Plotkin [2] gave a deterministic on-line algorithm (denoted by AAP in this paper) for the case in which the bandwidth request of each call is small when compared with the link capacities of the network. Their algorithm works for any network topology and achieves a benefit that is at least a logarithmic fraction (in the size of the network) of the optimal solution. No deterministic algorithm can have an asymptotically better competitive ratio [2]. However, if calls may request the full bandwidth of links in the network, Awerbuch, Azar, and Plotkin [2] prove a lower bound of $\Omega(n)$ for deterministic on-line algorithms for a line network, where n is the number of vertices. (Deterministic algorithms with good competitive ratios are still possible for the maximum edge-disjoint paths problem on some particular networks like expander graphs [15].)

Therefore randomized on-line algorithms have been considered for these problems. A randomized on-line algorithm, $\mathcal{A}$, is said to be c-competitive (against an oblivious adversary [7]) if for *any* sequence of requests σ, $E[\mathcal{A}(\sigma)] \geq \frac{1}{c} \cdot OPT(\sigma)$, where $\mathcal{A}(\sigma)$ and $OPT(\sigma)$ denote the algorithm's and the optimal benefit over sequence σ, and the expectation is taken over the random choices of $\mathcal{A}$. Awerbuch et al. [3] proposed an algorithm with an $O(\log n)$-competitive ratio for trees of n vertices. This result has been improved to $O(\log D)$, where D is the diameter of the tree, by Awerbuch et al. [4]. Kleinberg and Tardos [16] presented an algorithm with an $O(\log n)$-competitive ratio for meshes and a class of planar graphs. For all these topologies, matching (up to constant factors) randomized lower bounds have also been proved. On the other hand, Bartal, Fiat, and Leonardi [5] proved an $\Omega(n^{\epsilon})$ lower bound for randomized on-line algorithms on a specific network.

However, all the aforementioned algorithms have only the property that their *competitive ratio* is "good." They have not been analyzed with respect to their probability to get a "good" solution, and they only guarantee that for any sequence of requests the *expected* benefit is "close" (usually a logarithmic fraction) to the optimum. However, it may happen that, with high probability, a very poor benefit is achieved (and the main contribution to the average is given by a high benefit, obtained with low probability).[1] Indeed, algorithms of this kind have been criticized on this basis as not giving an appropriate solution for the problems they attempt to solve. For instance, the $O(\log n)$-competitive algorithm of [3] for the maximum edge-disjoint path problem on trees obtains for some sequences σ, even with $OPT(\sigma) = \Omega(n)$, a benefit of 0 with probability $1 - \frac{1}{\log n}$. The $O(\log D)$-competitive algorithm for trees [4] behaves somewhat better; still *unless* $OPT(\sigma) = \Omega(D^{\epsilon})$, the probability of getting any constant fraction of the expectation remains $o(1)$ (as a function of D). As to meshes, the algorithm of [16] decides with probability 1/2 to accept only "short" calls and with probability 1/2 to accept only "long" calls. On sequences of calls that are composed of only one kind of call, the algorithm would obtain benefit 0 with probability at least 1/2.

[1] Note that using the Markov inequality it is possible to obtain bounds on the probability of deviating from the expectation for algorithms for on-line minimization (cost) problems as a direct consequence of the analysis of the competitive ratio. This is not the case for maximization (benefit) problems.

In this paper we investigate the question of whether it is possible to obtain competitive randomized on-line algorithms (for the problems at hand) which also guarantee a "good" solution with "good" probability. Obviously, the probability of achieving a benefit close to the expectation is related to the quality of the expectation: the expectation of deterministic algorithms is obtained with probability 1; however, these algorithms have very poor competitive ratios for the problems we consider.

We formulate the following questions. Our results give answers to most of them in the context of the on-line maximum edge-disjoint paths problem.

1. Can a randomized on-line algorithm with optimal (up to a constant factor) competitive ratio guarantee a benefit of a constant fraction of the expectation with constant probability?

2. Can a randomized on-line algorithm with optimal competitive ratio (up to a constant factor) achieve a constant fraction of the expected benefit with probability that tends to 1 (say, as the size of the optimal solution grows)? What is the possible rate of convergence?

3. What is the interplay between the competitive ratio of a randomized algorithm, and the concentration of the benefit around its expectation?

1.1. Results of the paper. We address the above questions for the call admission problem in the basic case of infinite duration, requested bandwidth and link capacities 1, on the topologies of trees and meshes. Pairs of nodes are given on-line to an algorithm, which has to accept or reject every call immediately when it is received. An accepted call has to be assigned a route which is edge-disjoint with respect to all previously assigned routes. The aim is that of maximizing the number of accepted calls. This problem is equivalent to the on-line version of the maximum edge-disjoint paths problem for the above topologies. We present new randomized algorithms for trees and meshes with asymptotically optimal competitive ratios *and* good concentration of the benefit around its expectation.

Our work contains the following results for trees:

1. We present a family of randomized algorithms, parameterized by $k \geq 12$, with competitive ratios of $2k\lceil \log 2D \rceil$, and with the property that the probability to get any fixed constant fraction of the expected benefit tends to a constant, as $OPT/\log D$ tends to infinity. In addition, these algorithms have the property that their probability to obtain a $1-\delta$ fraction of the expectation, $\delta \in (0,1]$, is at least some constant value $1-\mathcal{P}(k,\delta)$ when $\frac{OPT}{\log D}$ is a large enough constant. The value of $1-\mathcal{P}(k,\delta)$ is such that it can be made as close to 1 as desired at the expense of a larger k, i.e., a larger constant in the competitive ratio.

We also present an $O(\log D)$-competitive algorithm that for *any* sequence of calls guarantees some constant fraction of the expected benefit with probability $\Omega(1)$ (equivalently, this algorithm guarantees an $O(\log D)$ fraction of the *optimal* solution with probability $\Omega(1)$).

2. We show that no optimally competitive algorithm can guarantee a constant fraction of the expected benefit with "very high" probability, unless the optimal solution is "very high." (For a formal statement see section 3.)

3. We also show that other algorithms from the family of algorithms with a slightly worse competitive ratio of $O(\log^{1+\epsilon} D)$, for arbitrary $\epsilon > 0$, have the property to achieve any fixed constant fraction of the expectation with probability that tends

to $1 - 1/\Omega(\log^{\epsilon} D)$, as $\frac{OPT}{\log^{1+3\epsilon} D}$ tends to infinity.

4. Our family of algorithms for trees is based on a new and simple scheme. We are also able to derive an algorithm with the best known competitive ratio for the problem, $6\lceil \log 4D \rceil$, as opposed to the previously known ratio of $48 \log 2D$ [4]. Finally we point out that our algorithms, and in fact *any* algorithm for link capacities 1, can be made to apply to trees with arbitrary uniform capacities.

5. We also study the problem on meshes and we present an asymptotically optimal $O(\log n)$-competitive algorithm for meshes that obtains any constant fraction of the expected benefit with probability that tends to 1, as $\frac{OPT}{\log^4 n}$ tends to infinity. This algorithm is based on some of the ideas of [16] and [6] and on new ideas presented here.

2. The algorithms for trees. The design of our family of algorithms is based on a new approach. They are composed of two conceptual steps. In the first step we apply an on-line *deterministic filter* to the input sequence. Those calls that are *not* filtered out are called *candidate calls* and may be accepted. The set of the candidate calls has two important properties: first, its cardinality is a constant fraction of the cardinality of the largest (optimal) set of calls that can be accepted (by an off-line adversary). Second, although this set cannot be fully accepted (as some of the calls intersect), the intersections between the calls of this set exhibit "nice" properties. When a call passes the deterministic filter and becomes a candidate it is presented to an on-line *randomized selection procedure* that determines if the call is actually accepted. The randomized selection procedures that we use are very simple, and the various algorithms are distinguished by the randomized selection procedure used.

Before we describe the algorithms, we give some notations: we denote by $\sigma = (s_1, t_1), (s_2, t_2), \ldots$ the sequence of requests. We denote by $OPT(\sigma)$ and $\mathcal{A}(\sigma)$ the set of calls accepted by an optimal (off-line) algorithm and by an on-line algorithm $\mathcal{A}$, respectively, out of the sequence σ. We denote by $\mathcal{C}(\sigma)$ the set of candidate calls passed on by the deterministic filtering procedure to the randomized selection procedure. By D we indicate the diameter of the tree. We abuse notation and denote by $OPT(\sigma)$, $\mathcal{A}(\sigma)$, and $\mathcal{C}(\sigma)$ also the cardinality of the corresponding sets. We start now by describing the properties of the deterministic filter that we use.

THEOREM 2.1. *There exists a deterministic on-line filtering procedure for trees of diameter D such that for any sequence σ the following properties hold:*

- $\mathcal{C}(\sigma) \geq OPT(\sigma)/6$.
- *The number of pairs of calls in $\mathcal{C}(\sigma)$ that intersect is at most $\mathcal{C}(\sigma) \cdot \lceil \log 2D \rceil$.*

We prove this theorem in section 2.2. We also show in section 2.3 an alternative filtering procedure which makes use of the AAP algorithm but achieves somewhat worse performances.

The different algorithms of the family of algorithms are distinguished by the randomized selection procedure used. We use a single procedure parameterized by a parameter p ($p \leq \frac{1}{2\lceil \log 2D \rceil}$). Algorithm $\mathcal{A}_p$ presents the candidate calls selected by the deterministic filter, one by one as they are selected to the on-line randomized selection procedure $\mathbf{RS}_p$ defined below.

$\mathbf{RS}_p$. Any candidate call $(s, t) \in \mathcal{C}$ becomes a *considered* call with probability p and is rejected with probability $1 - p$. A considered call is accepted if it does not

intersect any previous *considered* call.[2]

2.1. Analysis of the algorithms.

THEOREM 2.2. *Algorithm $\mathcal{A}_p$ is a $\frac{6}{p(1-p\lceil \log 2D\rceil\ )}$-competitive randomized on-line algorithm for call admission on trees of diameter D. For any $\delta \in (0,1]$, and any sequence σ, $Pr[\mathcal{A}_p(\sigma) < (1-\delta)E[\mathcal{A}_p(\sigma)]] \leq \exp(-\frac{OPT(\sigma)p}{48}(\delta(1-p\lceil \log 2D\rceil\ ))^2) + \frac{1}{1+\frac{1}{2}\delta\left(\frac{1}{p\lceil \log 2D\rceil}-1\right)}$.*

Before proving the theorem we explain its consequences and give some corollaries. The proof of the first corollary follows from simple calculations.

COROLLARY 2.3. *For any fixed constant $k \geq 12$, algorithm $\mathcal{A}_p$ with $p = \frac{6}{k\lceil \log 2D\rceil}$ is a $2k\lceil \log 2D\rceil$-competitive randomized algorithm for on-line call admission on trees of diameter D. Let $P(k,\delta,\sigma) = 1 - Pr[\mathcal{A}_p(\sigma) < (1-\delta)E[\mathcal{A}_p(\sigma)]]$. Then for fixed $k \geq 12$ and fixed $\delta > 0$, $P(k,\delta,\sigma) \to (1-\mathcal{P}(k,\delta))$, for $= \mathcal{P}(k,\delta) = \frac{1}{1+\frac{1}{2}\delta(\frac{k}{6}-1)}$, as $\frac{OPT(\sigma)}{\log D} \to \infty$. Furthermore, when $OPT(\sigma) \geq \beta_k \log D$, for some constant β_k, then $P(k,\delta,\sigma) \geq 1-c$ for some fixed constant $c < 1$.[3] Note that $1-\mathcal{P}(k,\delta)$ is a constant bounded away from 0, if $k \geq 12$, and that it can be made as close to 1 as desired, at the expense of a larger constant k, i.e., a larger constant in the competitive ratio.*

Based on the above corollary we also define a slightly different algorithm, $\mathcal{B}_p$, that guarantees for *any* sequence a constant fraction of the expected benefit with probability $\Omega(1)$. This algorithm, with probability 1/2 accepts the first call (and stops); with probability 1/2 it runs algorithm $\mathcal{A}_p$ on σ.[4] Although $\mathcal{B}_p$ may have benefit only 1 with probability 1/2, it has the desirable property that it achieves an $O(\log D)$ fraction of the *optimal* solution with constant probability: if $OPT(\sigma) = O(\log D)$, then $\mathcal{B}_p$ will have an $O(\log D)$ fraction of the optimal benefit with probability 1/2 (if it takes the first call). If $OPT = \Omega(\log D)$, then with probability 1/2, $\mathcal{B}_p$ runs $\mathcal{A}_p$, which, for such sequences, achieves an $O(\log D)$ fraction of the optimum with constant probability by Corollary 2.3.

COROLLARY 2.4. *There exists an $O(\log D)$-competitive algorithm for call admission on trees of diameter D such that for any sequence of calls the benefit obtained is within some constant fraction of the expectation with constant probability.*

We now give another corollary, relative to a family of algorithms that obtain any fixed constant fraction of the expectation with probability that tends asymptotically to 1 (as $OPT(\sigma)$ and D grow) at the expense of a competitive ratio slightly higher than the optimum.

COROLLARY 2.5. *Algorithm $\mathcal{A}_p$, with $p = \Theta(\frac{1}{\log^{1+\epsilon} D})$, $\epsilon > 0$, is an $O(\log^{1+\epsilon} D)$-competitive randomized algorithm for on-line call admission on trees of diameter D.*

[2]The more natural algorithm that immediately rejects a candidate call if it intersects a previous *accepted* call, and otherwise accepts the candidate with probability p, performs at least as well. We leave the description as above for simplicity of the proof. To see that, note that for any given sequence of random choices, one for each candidate, any candidate accepted by $\mathbf{RS}_p$ is also accepted by the more natural algorithm. Since the call is accepted by $\mathbf{RS}_p$ its corresponding random choice is "positive," and it does not intersect any previous considered call (i.e., it does not intersect any previous candidate with a "positive" random choice). In the more natural algorithm this call cannot intersect any previously accepted call (as no previous candidate that intersects it had a "positive" random choice), and its own random choice is "positive." Hence it is accepted.

[3]To see that, note that $p = \Theta(1/(2\log 2D))$, and thus, for $OPT(\sigma) \geq \beta_k \log D$, and large enough β_k, the absolute value of the expression in the exponent of the first summand is at least a constant, and the second summand is bounded from above by a constant.

[4]A more natural algorithm would continue to run $\mathcal{A}_p$ after taking the first call in the first case. However, in the worst case its behavior is no better than the above algorithm.

For any constant $\delta \in (0,1]$, $Pr[\mathcal{A}_p(\sigma) < (1-\delta)E[\mathcal{A}_p(\sigma)]] < \exp(-\delta^2 \frac{OPT(\sigma)}{\Theta(\log^{1+3\epsilon} D)}) + \frac{1}{\Omega(\delta \log^{\epsilon} D)}$.

Proof of Theorem 2.2. We prove that the expected number of calls accepted by $\mathcal{A}_p$ on a sequence σ is at least $\mathcal{C}(\sigma) \cdot p(1-p\lceil \log 2D \rceil)$, where $\mathcal{C}(\sigma)$ is the set of candidate calls picked from σ. By Theorem 2.1, $\mathcal{C}(\sigma) \geq OPT(\sigma)/6$, and the competitive ratio follows.

We number the *candidate* calls by the order that they arrive from 1 to $\ell = C(\sigma)$. If calls i and j intersect we denote this by $i \cap j$. Let c_i, $i \geq 1$, be an indicator random variable which is 1 if and only if the ith candidate is *considered*. Let $C = \sum_i c_i$. Let $y_{i,j}$, for $i < j$ and $i \cap j$, be an indicator random variable which is 1 if and only if both calls i and j are considered, and let $Y = \sum_{i<j, i\cap j} y_{i,j}$. Let r_j be an indicator random variable which is 1 if and only if there is an index $i < j$, $i \cap j$, such that $y_{i,j} = 1$. Let $R = \sum_j r_j$. Note that $R \leq Y$.

A candidate is accepted if it was considered, but none of its (previously presented) intersecting candidates was considered. Thus, if candidate j is considered but not accepted into the on-line solution, then there is a candidate $i < j$, $i \cap j$, that was considered. That is, $c_j = 1$, and $r_j = 1$. Therefore $\mathcal{A}_p(\sigma) = C - R \geq C - Y$, and we get $E[\mathcal{A}_p(\sigma)] \geq E[C] - E[Y]$.

Clearly $E[C] = \mathcal{C}(\sigma) \cdot p$ since each candidate is considered with probability p. For any pair $i < j$, $i \cap j$, $Pr[y_{i,j} = 1] = p^2$ since the event occurs if and only if call i and call j are considered. By Theorem 2.1 there are at most $\mathcal{C}(\sigma) \cdot \lceil \log 2D \rceil$ pairs $i < j$ such that $i \cap j$. Therefore $E[Y] \leq \mathcal{C}(\sigma)\lceil \log 2D \rceil \cdot p^2$. We get that $E[\mathcal{A}_p(\sigma)] \geq E[C] - E[Y] \geq \mathcal{C}(\sigma)(p - \lceil \log 2D \rceil \cdot p^2) = \mathcal{C}(\sigma) \cdot p(1 - p\lceil \log 2D \rceil)$.

We now turn to prove the second part of the claim. For $\delta \in (0,1]$, since $E[C] \geq \frac{E[Y]}{p\lceil \log 2D \rceil} \geq \frac{E[R]}{p\lceil \log 2D \rceil}$, we have

$$
\begin{aligned}
&Pr[\mathcal{A}_p(\sigma) < (1-\delta)E[\mathcal{A}_p(\sigma)]] \\
&= Pr[(C-R) < (1-\delta)E[C-R]] \\
&\leq Pr\left[C < E[C] - \frac{1}{2}\delta E[C-R]\right] \\
&\quad + Pr\left[R > E(R) + \frac{1}{2}\delta E[C-R]\right] \\
&\leq Pr\left[C < E[C] - \frac{1}{2}\delta(1-p\lceil \log 2D \rceil)E[C]\right] \\
&\quad + Pr\left[R > E[R] + \frac{1}{2}\delta\left(\frac{1}{p\lceil \log 2D \rceil} - 1\right)E[R]\right] \\
&= Pr\left[C < \left(1 - \frac{1}{2}\delta(1-p\lceil \log 2D \rceil)\right) \cdot E[C]\right] \\
&\quad + Pr\left[R > \left(1 + \frac{1}{2}\delta\left(\frac{1}{p\lceil \log 2D \rceil} - 1\right)\right) \cdot E[R]\right] .
\end{aligned}
$$

The variable $C = \sum_{j=1}^{l} c_j$ is the sum of $\ell = \mathcal{C}(\sigma)$ random variables $c_j \in \{0,1\}$, set by independent Bernoulli trials to 1 with equal probability p and to 0 with equal probability $1-p$. We use the following version of Chernoff's bound (cf. [19]) to bound the first summand: let $\mu = E[C] = p\mathcal{C}(\sigma)$ be the expected value of variable C; then

for every $\gamma \in (0,1]$,

$$Pr[C < (1-\gamma)\mu] < e^{(-\mu\gamma^2/2)}.$$

The variable R is the sum of dependent random variables, as for all pairs $i, j, i < j$, $y_{i,j}$ depends on the random choice for candidate i. We use the Markov inequality for the second summand.

Using $E[C] \geq \frac{OPT(\sigma)p}{6}$, we have

$$\begin{aligned}
&Pr[\mathcal{A}_p(\sigma) < (1-\delta)E[\mathcal{A}_p(\sigma)]] \\
&< \exp\left(-\frac{E[C]}{2}\left(\frac{1}{2}\delta(1-p\lceil\log 2D\rceil)\right)^2\right) \\
&\quad + \frac{1}{1+\frac{1}{2}\delta\left(\frac{1}{p\lceil\log 2D\rceil}-1\right)} \\
&= \exp\left(-\frac{\mathcal{C}(\sigma)p}{2}\left(\frac{1}{2}\delta(1-p\lceil\log 2D\rceil)\right)^2\right) + \frac{1}{1+\frac{1}{2}\delta\left(\frac{1}{p\lceil\log 2D\rceil}-1\right)} \\
&\leq \exp\left(-\frac{OPT(\sigma)p}{12}\left(\frac{1}{2}\delta(1-p\lceil\log 2D\rceil)\right)^2\right) \\
&\quad + \frac{1}{1+\frac{1}{2}\delta\left(\frac{1}{p\lceil\log 2D\rceil}-1\right)}\,. \qquad \square
\end{aligned}$$

2.2. The deterministic filter. When call (s,t) is presented to the algorithm, it is first passed through the deterministic filter, where it is discarded or becomes a candidate. We describe in the following a deterministic filter appropriately designed for trees.

First we designate an arbitrary internal vertex as the root of the tree and denote it by r. Without loss of generality we can restrict our attention to calls between two leaves of the tree. This is shown by constructing a new instance of the problem where all the calls are between leaf vertices of the tree, such that any solution to the new instance can be transformed back to a solution of the original instance, with the same size. The tree network of the new instance is obtained by adding to the original tree, for every internal vertex v and every edge e adjacent to v, a new leaf vertex v_e connected to v. For a call (s,t) of sequence σ, let e_s, e_t be the first and the last edges of the path connecting s to t. A new sequence σ' is obtained by transforming every call (s,t) into a call (s_{e_s}, t_{e_t}). It is easy to show that any subset of calls of σ can be accepted as a solution in the original tree if and only if the corresponding subset of calls of σ' can be accepted as a solution in the new tree. We note that this construction does not change the diameter of the tree.

We denote by $lca(s,t)$ the *least common ancestor* of vertices s and t in the tree rooted at r and by $p(u)$ the parent of vertex u in that tree. Path $path(s,t)$ is the *set* of edges in the path connecting s to t. Finally, $\mathcal{T}$ is the set of all edges of the tree.

Calls will be discarded on the basis of two tests. To perform the first test we block two edges when a call becomes a candidate: for any candidate (s',t') the two edges on $path(s',t')$ adjacent to $lca(s',t')$ are blocked. The edges are said to be *blocked edges.*

For the purpose of the second test we associate a weight function $w(e)$ to each edge of the tree. The value $w(e)$ is determined as follows:

1. Initially, assign $w(e) = 1$ for all edges $e \in \mathcal{T}$.

2. Double $w(e)$ whenever edge e is included in a $path(s', t')$ for a call (s', t') that becomes a candidate.

On the basis of the value of the weight function we give the following definition.

DEFINITION. *For two vertices u and v, $path(u, v)$ is a* segment *at a certain time in the run of the algorithm if at this time, for some nonempty subset of the current candidates, denoted by $\mathcal{C}'$, $path(u, v)$ is the maximal path contained in $\bigcap_{(s,t)\in\mathcal{C}'} path(s, t)$ such that no edge of the path is included in a candidate call not in $\mathcal{C}'$. We denote a segment, and its set of edges, by $seg(u, v)$.*

Note that for two vertices u and v, $path(u, v)$ may at times be a segment but later cease to be a segment. Further note that the edges of a segment may increase their weight over time, but when they form a segment they have equal weight. We thus sometimes refer to the weight of the segment being the weight of each of its edges. In what follows we will count the number of segments (of weight $w > 2$) ever created during the run of the algorithm. We identify a segment by its two endpoints. Thus, the first time that $path(u, v)$ becomes a segment, it is counted as one. If the weights of its edges increase, and it continues to be a segment, we do not count it again.

As an example, the set of segments defined by a set of candidate calls is shown in Figure 2.1.

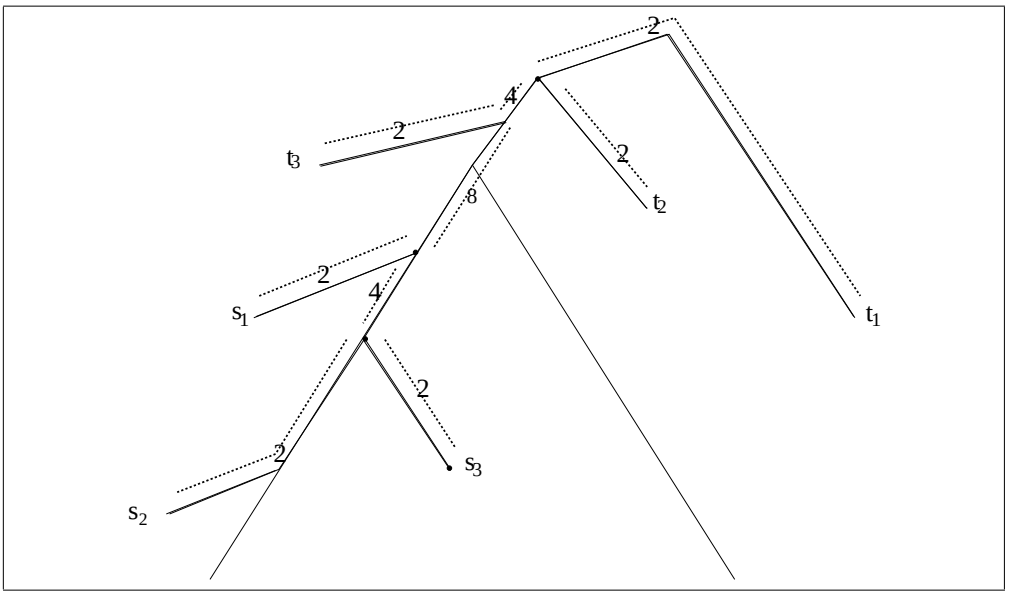

FIG. 2.1. *The segments created by the three candidate calls (s_1, t_1), (s_2, t_2), (s_3, t_3) are marked with dashed lines. The weight of each segment is indicated next to the dashed line.*

As mentioned above the filter is based on two tests:

Test 1. Discard call (s, t) if $path(s, t)$ contains a blocked edge.

If call (s, t) is not discarded by the first test, then it is submitted to the second test.

Test 2. Discard a call (s, t) if at the time the call is presented, there exists a segment $seg(u, v)$ of weight w such that $|path(lca(s, t), s) \cap seg(u, v)| \geq \frac{2D}{w}$ or $|path(lca(s, t), t) \cap seg(u, v)| \geq \frac{2D}{w}$.

A call that is not discarded by any of these test becomes a *candidate*.

DEFINITION. *For a sequence of calls σ, denote by $\mathcal{C}(\sigma)$ the set of candidates produced by the deterministic filter out of sequence σ. Denote by $\mathcal{C}_1(\sigma)$ those calls that are discarded by Test 1 and by $\mathcal{C}_2(\sigma)$ those calls that are discarded by Test 2. When the sequence σ is clear from the context, we sometimes use the notations $\mathcal{C}, \mathcal{C}_1, \mathcal{C}_2$. We also sometimes abuse notation and use $\mathcal{C}, \mathcal{C}_1, \mathcal{C}_2$ for the cardinality of the corresponding sets.*

We prove below that the number of candidates $\mathcal{C}(\sigma)$, i.e., those calls that are not discarded by any of the two tests, is at least $OPT(\sigma)/6$. In the following we will often say that edge e is included in a call (s,t) if $e \in path(s,t)$. We give some claims and lemmas as intermediate steps.

CLAIM 2.6. *A call (s,t) in $\mathcal{C} \cup \mathcal{C}_2$ intersects a previous candidate (s',t') only if (s',t') includes edge $(lca(s,t), p(lca(s,t)))$.*

Proof. If there is a previous candidate (s',t') that does not include edge $(lca(s,t), p(lca(s,t)))$ but intersects call (s,t), then $lca(s',t')$ is in the subtree rooted at $(lca(s,t))$. Then (s,t) intersects (s',t') in one of the two blocked edges of (s',t'), a contradiction to Test 1. □

CLAIM 2.7. *If the edges of a segment $seg(u,v)$ have weight $w > 2$, then either u is an ancestor of v or v is an ancestor of u.*

Proof. If $w > 2$, then the edges of the segment are included in more than one candidate call. Let (s,t) be the first such candidate call. If neither u is an ancestor of v nor v is an ancestor of u, then $lca(s,t)$ is internal to $seg(u,v)$. Two edges of $seg(u,v)$ would have been blocked when (s,t) became a candidate. The two blocked edges would also be part of any later call (s',t') that contains $seg(u,v)$, and (s',t') would not become a candidate, a contradiction. □

LEMMA 2.8. *Let (s,t) be a call in $\mathcal{C} \cup \mathcal{C}_2$ (i.e., a call that passes Test 1). Consider the intersection of $path(lca(s,t),s)$ (resp., $path(lca(s,t),t)$) with the set of calls that are candidate calls prior to the presentation of (s,t) and assume that this intersection is not empty. Then, there exists a sequence of nodes $v_0, \ldots, v_\ell$, $\ell \geq 1$, such that the following hold:*

1. *$v_0 = lca(s,t)$; v_ℓ is either equal to s (resp., t) or on $path(lca(s,t),s)$ (resp., $path(lca(s,t),t)$); and the intersection of $path(lca(s,t),s)$ (resp., $path(lca(s,t),t)$) with the set of previous candidate calls is $path(v_0, v_\ell)$.*
2. *For $0 \leq i \leq \ell - 1$, all edges between v_i and v_{i+1} have the same weight, which we denote W_i.*
3. *For $0 < i \leq \ell - 1$, $W_i < W_{i-1}$.*
4. *For $0 \leq i \leq \ell - 1$, there is a set of candidates $\mathcal{C}_i$ such that for any edge e in $path(v_i, v_{i+1})$, the set of calls that use e is exactly $\mathcal{C}_i$.*
5. *For $0 < i < \ell - 1$, $path(v_i, v_{i+1})$ is a segment.*

Proof. We prove the claim for the path from $lca(s,t)$ to s. An analogous proof holds for the path from $lca(s,t)$ to t.

Consider the time when call (s,t) is presented before it becomes a candidate. First consider an edge e on $path(lca(s,t),s)$. If there is a previous candidate call that uses this edge, then this candidate call also uses edge $(lca(s,t), p(lca(s,t))$ by Claim 2.6. It follows that for any edge e on $path(lca(s,t),s)$, if edge e is used by a previous candidate, then all edges between e and $lca(s,t)$ are used by this candidate. Let e be the edge on $path(lca(s,t),s)$ which is furthest away from $lca(s,t)$ and is included in a previous candidate call. Let the node adjacent to e and further away from $lca(s,t)$ be node v_ℓ, and let v_0 be $lca(s,t)$. This proves point 1.

Since for any edge e, any previous candidate that uses edge e also uses all edges

between edge e and $lca(s,t)$ (by Claim 2.6), it follows that going from $lca(s,t)$ to s, the weight of the edges is nonincreasing. If there is a decrease in the weight of the edges, we assign the point of decrease as one of the nodes v_i. We thus get a sequence of nodes v_i that satisfy points 2 and 3.

For $0 \leq i \leq \ell - 1$, now consider $path(v_i, v_{i+1})$. We claim that there is a set of candidates $\mathcal{C}_i$ such that all edges on $path(v_i, v_{i+1})$ are used by exactly the calls in $\mathcal{C}_i$. Consider two edges e and e' on this path, where e' is further away from $lca(s,t)$ than e. Any candidate that uses e' also uses e, following Claim 2.6. On the other hand, if there were a candidate that uses e, but not e', then the weights of e and e' would have been different. We thus establish point 4.

We further claim that for $0 < i < \ell - 1$, $path(v_i, v_{i+1})$ is the maximal path of edges that are used by exactly the calls in $\mathcal{C}_i$. For edges on $path(lca(s,t), s)$, any edge which is not on $path(v_i, v_{i+1})$ has weight not equal to W_i and thus cannot be used by exactly the calls in $\mathcal{C}_i$. Furthermore, observe that there is at least one call in $\mathcal{C}_i$ that uses also the edge leading from v_{i+1} towards s and must also reach $lca(s,t)$ (by Claim 2.6). Thus no edge outside of $path(lca(s,t), s)$ can possibly be used by the calls in $\mathcal{C}_i$. It follows from the definition of a segment that $path(v_i, v_{i+1})$ is a segment satisfying point 5. □

The number of calls that are discarded by Test 2, while accepted in the optimal solution, will be proved to be related to the number of segments of weight bigger than 2 ever created. We prove the following upper bound on the number of such segments.

LEMMA 2.9. *Consider a run of the algorithm on a sequence of calls σ. The number of segments ever created, and which at some point in time have weight bigger than 2, is at most $4\mathcal{C}(\sigma)$.*

Proof. Note that when a call is presented but does not become a candidate, there is no change in the segments in the tree. Therefore, we consider events when a new call becomes a candidate. To prove the lemma we give an upper bound on the number of segments that at such event either are created with weight bigger than 2 or reach a weight bigger than 2 (without changing their endpoints). We show that the number of such segments is at most 4 per such event.

Let (s,t) be a call that becomes a candidate. If it does not intersect any previous candidate, then no new segments of weight bigger than 2 are created. We therefore assume that call (s,t) intersects at least one previous candidate. We distinguish between two cases. The first one is when all the intersections of (s,t) with previous candidates are either on $path(lca(s,t), s)$ or on $path(lca(s,t), t)$. The second case is when there are such intersections on both $path(lca(s,t), s)$ and $path(lca(s,t), t)$.

We now consider the first case (without loss of generality assume that all intersections are on $path(lca(s,t), s)$). We use the sequence of nodes v_i, $0 \leq i \leq \ell$, guaranteed by Lemma 2.8. A new segment of weight bigger than 2 can be created either by the augmentation of the weight of edges of weight 2, or by the augmentation of the weight of edges of weight already bigger than 2, but while creating new endpoints for the segments (this may create more than one new segment).

If prior to (s,t) becoming a candidate there is a segment of weight bigger than 2 that includes $(lca(s,t), p(lca(s,t))$ and the edge e that leads from $lca(s,t)$ towards s, then two new segments of weight bigger than 2 are created with a new endpoint at $lca(s,t)$. One of these two segments is $path(v_0, v_1)$. See Figure 2.2.

Further distinguish the case where v_ℓ is inside a segment from the case where v_ℓ is not inside a segment.

If v_ℓ is inside a segment (see Figure 2.3), two new segments are created, both

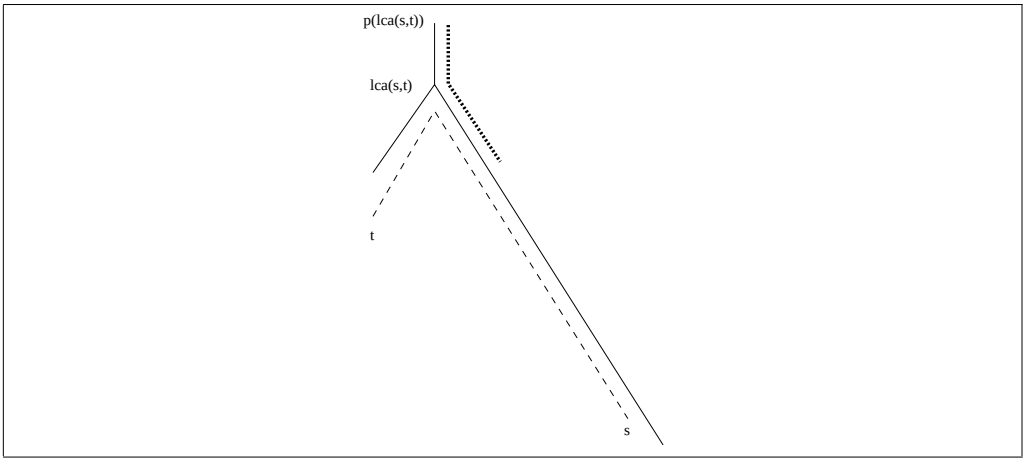

FIG. 2.2. *Two new segments are created with endpoint* $lca(s,t)$.

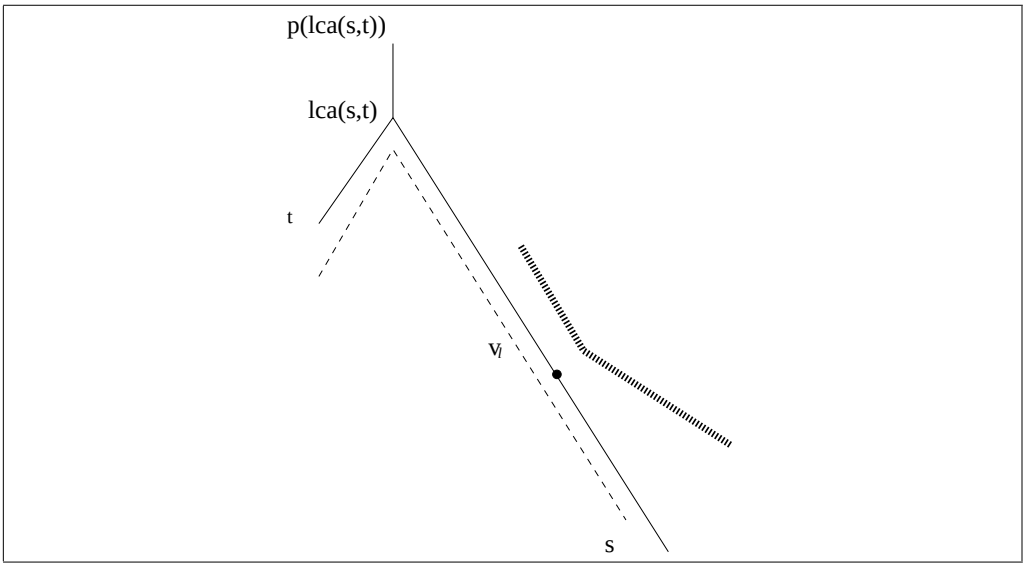

FIG. 2.3. *Vertex* v_ℓ *is inside a segment.*

with an endpoint at v_ℓ. If the weight of the segment was already bigger than 2, then two new segments of weight bigger than 2 are created. In this case all the segments $seg(v_{j-1}, v_j)$, $1 < j < \ell$, are already of weight bigger than 2; their weight will increase, but since their endpoints do not change there are no new segments here. If the weight of the segment was 2, then only one segment of weight bigger than 2 is created (while the other one is a segment of weight 2 with new endpoints). Whatever the weight of this segment was, all the segments $seg(v_{j-1}, v_j)$, $1 < j < \ell$, have weight bigger than 2 (by Lemma 2.8); their weight will increase, but since their endpoints do not change no new segments of weight bigger than 2 are created here. Altogether we get at most four new segments of weight bigger than 2.

For the case where v_ℓ is not inside a segment (see Figure 2.4), observe that the weights of the edges on $path(v_{\ell-1}, v_\ell)$ are doubled, thus possibly creating one new

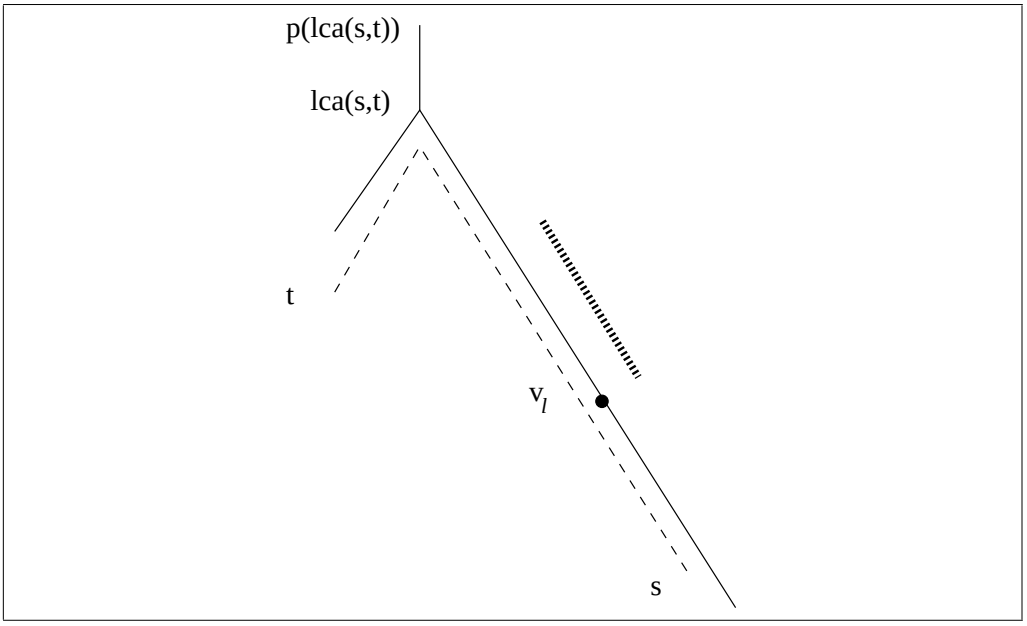

FIG. 2.4. *Vertex v_ℓ is not inside a segment.*

segment of weight bigger than 2 (which is $seg(v_{\ell-1}, v_\ell)$), if the weight of this segment was not bigger than 2 beforehand. Altogether we get for this case at most three new segments of weight bigger than 2.

We note that a new segment of weight 2 may be created as $seg(v_\ell, s)$ if $v_\ell \neq s$.

Now consider the second case where (s,t) intersects previous candidates on both $path(lca(s,t), s)$ and $path(lca(s,t), t)$. Consider the two edges e_1, and e_2, adjacent to $lca(s,t)$ and leading to s and t, respectively. Since by Claim 2.6 any previous candidate call that intersects (s,t) also uses the edge between $lca(s,t)$ and $p(lca(s,t))$, it follows that there are two candidate calls p_1 and p_2 such that p_1 uses e_1 and $(lca(s,t), p(lca(s,t))$, and p_2 uses e_2 and $(lca(s,t), p(lca(s,t))$. It follows that prior to (s,t) becoming a candidate, $(lca(s,t), p(lca(s,t))$ and e_1 (resp., e_2) cannot be in the same segment. We now use the sequence of nodes v_i guaranteed by Lemma 2.8 for each of $path(lca(s,t), s)$ and $path(lca(s,t), t)$. From the above argument it follows that the two paths $path(v_0, v_1)$ (on the two paths $path(lca(s,t), s)$ and $path(lca(s,t), t)$) are segments. No new segment will be created with a low endpoint at $lca(s,t)$ when (s,t) becomes a candidate. We therefore now consider separately $path(lca(s,t), s)$ and $path(lca(s,t), t)$. We consider in what follows $path(lca(s,t), s)$. Analogous arguments hold for $path(lca(s,t), t)$.

As argued above, prior to (s,t) becoming a candidate, $path(v_0, v_1)$ is a segment, $seg(v_0, v_1)$. When (s,t) becomes a candidate, the weight of this segment doubles.

If prior to (s,t) becoming a candidate the weight of $seg(v_0, v_1)$ was already bigger than 2, then there is no new segment at the top of $path(lca(s,t), s)$. Using the same arguments as in the case where all intersections are on $path(lca(s,t), s)$ (and not also on $path(lca(s,t), t)$) it follows that there are at most 1 or at most 2 new segments of weight bigger than 2 on $path(lca(s,t), s)$, depending on whether or not v_ℓ equals s.

If prior to (s,t) becoming a candidate the weight of $seg(v_0, v_1)$ was 2, then when the (s,t) becomes a candidate this segment becomes a new segment with weight bigger

than 2. However, it also follows that v_1 must equal v_ℓ, as there is only one previous candidate on $path(lca(s,t),s)$. No additional new segment of weight bigger than 2 will be created on $path(lca(s,t),s)$. (Note that a new segment of weight 2 may be created if $v_\ell \neq s$.)

We can conclude that on any of $path(lca(s,t),s)$ and $path(lca(s,t),t)$ at most two new segments of weight bigger than 2 can be created. Altogether at most four new segments of weight bigger than 2 are created when (s,t) becomes a candidate. □

We now prove that the number of calls not discarded (i.e., the number of calls that become candidates) is a good fraction of size of the optimal solution.

LEMMA 2.10. *For any sequence of calls σ, the number of candidate calls $\mathcal{C}(\sigma)$ has the property that $\mathcal{C}(\sigma) \geq \frac{OPT(\sigma)}{6}$.*

Proof. Since $\sigma = \mathcal{C} \cup \mathcal{C}_1 \cup \mathcal{C}_2$, we prove that the optimal solution accepts at most $2\mathcal{C}$ calls from $\mathcal{C} \cup \mathcal{C}_1$ and at most $4\mathcal{C}$ calls from $\mathcal{C}_2$.

First consider the calls from $\mathcal{C} \cup \mathcal{C}_1$ accepted in the optimal solution. Any call of $\mathcal{C} \cup \mathcal{C}_1$ includes at least one blocked edge. Since there are at most $2\mathcal{C}$ blocked edges in the tree, any solution can accept at most $2\mathcal{C}$ calls from $\mathcal{C} \cup \mathcal{C}_1$.

We now consider the calls from $\mathcal{C}_2$ accepted in the optimal solution. Consider a call $(s,t) \in \mathcal{C}_2$. It includes, when presented, at least $\frac{2D}{w}$ edges of some segment $seg(u',v')$ of weight $w > 2$ either on $path(lca(s,t),s)$ or on $path(lca(s,t),t)$. (Observe that no call is discarded because of a segment of weight 2 since $path(lca(s,t),s)$ and $path(lca(s,t),t))$ are of size at most $D-1$.) For the sake of analysis, we assign call (s,t) to a single such segment, $seg(u',v')$.

We now show that the length of $seg(u',v')$ is less than $\frac{4D}{w}$. By definition, the edges of segment $seg(u',v')$ are included in the intersection of a set $\mathcal{C}'$ of candidate calls. Let (s',t') be the call of $\mathcal{C}'$ presented last, and let $\mathcal{C}'' = \mathcal{C}' \setminus \{(s',t')\}$. Set $\mathcal{C}''$ defines a segment $seg(u'',v'')$ at the time call (s',t') is presented, and this segment includes segment $seg(u',v')$. To see that, observe that $path(u',v')$ is included in the intersection of the calls in $\mathcal{C}''$ and that at the above time no other candidate includes any edge of $path(u',v')$. We know that the weight of this segment at the time call (s',t') is presented is $\frac{w}{2}$ (since when (s',t') becomes a candidate it doubles the weights to w). Since call (s',t') becomes a candidate it passes Test 2, and we can conclude that its intersection with $seg(u'',v'')$, on either $path(lca(s',t'),s')$ or $path(lca(s',t'),t')$, is of length less than $\frac{4D}{w}$. However, by Claim 2.7, either u' is an ancestor of v' or vice versa, as $w > 2$, and $path(u',v')$ is included in call (s',t'), as $seg(u',v')$ is defined by $\mathcal{C}'$. Therefore all of $path(u',v')$ is included in this intersection. If follows that the length of segment $seg(u',v')$ is less than $\frac{4D}{w}$.

It now follows that if we consider the calls of the optimal solution, at most one call can be assigned to each segment of weight bigger than 2 in the way we described. This is because any assigned call uses more than half the edges of the segment it is assigned to, and thus any two would intersect. By Lemma 2.9, at most $4\mathcal{C}$ segments of weight bigger than 2 are ever created while the algorithm processes the sequence. We conclude that at most $4\mathcal{C}$ calls of $\mathcal{C}_2$ can be in the optimal solution. □

LEMMA 2.11. *Every candidate call intersects at most $\lceil \log 2D \rceil$ previous candidates.*

Proof. By Claim 2.6, for any candidate (s,t), all the previous intersecting candidates include edge $(lca(s,t), p(lca(s,t)))$. Their number is at most $\lceil \log 2D \rceil$, since once an edge is included in $\lceil \log 2D \rceil$ candidates, it has weight of at least $2D$. No call can become a candidate if it includes an edge of weight $2D$, as it will fail Test 2. □

Proof of Theorem 2.1. The first part of the theorem is proved in Lemma 2.10. For the second part, by Lemma 2.11 every candidate call of $\mathcal{C}(\sigma)$ intersects at most $\lceil \log 2D \rceil$ previous candidates. The number of intersecting pairs is thus at most $\mathcal{C}(\sigma) \cdot \lceil \log 2D \rceil$. □

2.3. An alternative deterministic filter. In this section we describe an alternative design of the deterministic filter at the cost of having a somewhat worse performance. This filter uses the AAP (deterministic) call control algorithm [2], originally designed for general networks of high capacity. The test runs the AAP algorithm on the tree, assuming link capacities of $\log 4D$. A call that is accepted by the AAP algorithm becomes a candidate, while a call that is rejected by AAP is also rejected by the filter. This filter relies on the following property: if the capacity is increased in that way, then the set of calls accepted by AAP is a constant fraction of the optimal set accepted by an adversary that has only capacity 1. (A similar result is also derived in [18] if the capacity is increased by a factor of $\Omega(\log n)$.) On the other hand, since the capacity of each link is bounded, the number of intersections between the candidate calls is small.

In Appendix A we give for completeness the AAP algorithm for this specific setting and prove the following claim.

THEOREM 2.12. *If for every $e \in E$ the AAP algorithm has capacity $u(e) = \log 4D$ while the adversary has capacity $u(e) = 1$, then the AAP algorithm has competitive ratio* 5.

Based on the above theorem we can now give the properties of the filter.

LEMMA 2.13. *The above filter based on the AAP algorithm, applied to trees of diameter D, achieves the following for any sequence σ:* (1) $\mathcal{C}(\sigma) \geq OPT(\sigma)/5$; (2) *The number of pairs of calls in $\mathcal{C}(\sigma)$ that intersect is at most $\mathcal{C}(\sigma) \cdot 2\lceil \log 2D \rceil$.*

Proof. The first part of the claim follows from Theorem 2.12. For the second part of the claim we consider an intersection graph of paths on a tree.[5] Our claim follows from the fact that if each edge in the tree is included in at most g of these paths, then the intersection graph is a $(2(g-1))$-inductive graph (see [6]).[6] Since the capacity used by the AAP algorithm is $\log 4D$, any edge is used by at most $\log 4D$ calls. Therefore the intersection graph of the candidate calls is a $2(\log 4D - 1)$-inductive graph, which implies that the total number of edges in this graph is at most $\mathcal{C}(\sigma) \cdot 2(\log 4D - 1) = \mathcal{C}(\sigma) \cdot 2 \log 2D$. This is a bound on the number of pairs of candidates that intersect. □

The randomized selection procedure used in conjunction with this filter is the same as for our first filter. The results stated in Theorem 2.2 are proved in a similar way with necessary modifications in the constants.

3. A bound on the probability to achieve a "good" solution. We have presented $O(\log D)$-competitive algorithms for call admission on trees that for any sequence σ, with $OPT(\sigma) = \Omega(\log D)$, achieve any constant fraction of the expected benefit with at least constant probability. The probability tends to some constant as $OPT(\sigma)$ tends to infinity. The obvious question is then if the rate of convergence and the limit probability can be improved for $O(\log D)$-competitive algorithms and in particular if the limit can be 1.

[5] The intersection graph has a node for each one of the paths. There is an edge between two nodes in the intersection graph if and only if the corresponding two paths intersect.

[6] A d-inductive graph is a graph $G = (V, E)$ for which there is an ordering of the vertices such that for any i, $|\{j : i < j, (i, j) \in E\}| \leq d$.

In this section we give a partial answer to this question. We show that for any algorithm with an $O(\log D)$-competitive ratio, there is a sequence σ with $OPT(\sigma) = \Theta(D^{1-c})$, for any $0 < c < 1/2$, for which with probability $\Omega(c)$, the on-line benefit is $o(OPT(\sigma)/\log D)$ (in fact the on-line benefit is only a $n^{-\epsilon}$ fraction of the expectation for some constant $\epsilon > 0$).

We give our lower bound on a line network of $n+1$ vertices, with $n = 2^s$, for some s. Observe that on such a line network we have $D = n$. Given a randomized $k \log n$-competitive algorithm for the line, we present a sequence of requests formed by $l = \lfloor \alpha \cdot \log n \rfloor + 1$ ($\alpha < 1/2$) phases, plus a final extra phase. At phase i, $0 \le i \le l$, 2^i pairwise disjoint calls of size $\frac{n}{2^i}$ are presented. These are calls $((j-1)\frac{n}{2^i}+1, j\frac{n}{2^i}+1)$, $1 \le j \le 2^i$. Thus, during phase l, 2^l calls of size $n/2^l$ are presented. We will prove that for any $k \log n$-competitive algorithm, at least one of these calls has the property that with probability at least $\frac{\alpha}{4k}$ all its edges are occupied by the time phase l ends. Then, we present $n/2^l$ disjoint calls of length 1 on these edges. The optimal solution is of size $n/2^l \ge n/2^{\alpha \log n} = n^{1-\alpha}$, while with probability at least $\frac{\alpha}{4k}$ the on-line algorithm can achieve a benefit of at most $2^l \le 2^{\alpha \log n} = n^\alpha$. We get that the algorithm cannot be close (by any constant fraction) to its expectation with probability $1 - o(1)$ (as a function of D), even when the size of the optimal solution is $\Omega(n^{1-c})$.

We now turn to prove our claim. Let p_m^i be the absolute probability that the mth call of phase i is accepted. We denote by P_m^l the sum of the absolute probabilities of acceptance of all the calls that include call m of level l, plus the probability that this call itself is accepted. We will show that for any $k \log n$-competitive randomized algorithm, $\sum_{m=1}^{2^l} P_m^l \ge 2^l \cdot \frac{\alpha}{4k}$. Therefore, there is at least one call m' such that $P_{m'}^l \ge \frac{\alpha}{4k}$, from which our claim follows.

We have

$$\begin{aligned}\sum_{m=1}^{2^l} P_m^l &= \sum_{i=0}^{l} 2^{l-i} \left(\sum_{m=1}^{2^i} p_m^i \right) \\ &\ge \sum_{i=0}^{l} 2^l \frac{1}{2} \sum_{j=i}^{l} 2^{-j} \left(\sum_{m=1}^{2^i} p_m^i \right) \\ &= 2^{l-1} \sum_{j=0}^{l} 2^{-j} \sum_{i=0}^{j} \left(\sum_{m=1}^{2^i} p_m^i \right).\end{aligned}$$

The first equation follows since the absolute probability p_m^i of accepting call m of phase i is summed up for the 2^{l-i} included calls of level l. Now, observe that $\sum_{i=0}^{j}(\sum_{m=1}^{2^i} p_m^i)$ is the algorithm's expected benefit after the calls of phase j have been presented.

We continue with $2^{-j}\sum_{i=0}^{j}(\sum_{m=1}^{2^i} p_m^i) \ge \frac{1}{k \log n}$, since the algorithm is $k \log n$-competitive, and the optimal solution can accept 2^j calls after phase j. We get

$$\sum_{m=1}^{2^l} P_m^l \ge 2^{l-1} \frac{l+1}{k \log n} > 2^l \cdot \frac{\alpha}{4k}$$

(for large enough n), which proves our claim. We can conclude the following theorem.

THEOREM 3.1. *For any $k \log n$-competitive algorithm for call admission on the line of n nodes, and any constant probability p ($p < \frac{1}{8k}$), there is a sequence σ with*

$OPT(\sigma) = n^{1-O(p)}$ *such that with probability at least* p*, the algorithm* does not *achieve any constant fraction of the expected benefit.*

4. Further extensions for trees. In this section we show that a better competitive ratio can be obtained at the expense of a larger deviation. We obtain a competitive ratio of $6\lceil \log 4D \rceil$ which improves upon the previously known bound of $48 \log 2D$ [4]. We also consider extensions of our algorithms to trees with edges of any uniform capacity and point out that *any* algorithm for call admission on trees can be converted to apply to uniform-capacity trees with almost the same competitive ratio.

First, we give an algorithm with a competitive ratio of $6\lceil \log 4D \rceil$. This is done at the expense of a larger deviation from the expectation. To obtain the improved competitive ratio we use our general framework, with the first version of the deterministic filter (section 2.2), but a different randomized selection procedure. We now use a variation of the "classify and randomly select" technique [3]: we use a set of $\lceil \log 4D \rceil$ colors and define an arbitrary order on them. We first choose uniformly at random one color among these colors, and we denote it A. When a new candidate call is presented to the randomized selection procedure, we assign to it the first color (according to the defined order) that was not assigned to any of the previous candidates that intersect the present one (i.e., we color the candidate calls with a proper coloring in their intersection graph). By Lemma 2.11, $\lceil \log 4D \rceil$ colors are sufficient, since each candidate intersects at most $\lceil \log 2D \rceil$ previous candidates. The algorithm will now accept those candidates that are assigned the color A. Since the above coloring procedure classifies the candidate calls into $\lceil \log 4D \rceil$ disjoint classes, each class fully acceptable, we obtain the following theorem.

THEOREM 4.1. *There exists a* $6\lceil \log 4D \rceil$*-competitive randomized algorithm for call admission on trees of diameter* D*.*

Proof. We partition $\mathcal{C}$ into a set of $\lceil \log 4D \rceil$ disjoint classes. Class $\mathcal{C}_i$, $i = 1, \ldots, \lceil \log 4D \rceil$, contains all the candidates which are assigned color i. All the candidates of a class can be accepted together in a solution since they are nonintersecting. The algorithm accepts a set of calls $\mathcal{A}$ that corresponds to the randomly selected class. The expected size of the on-line solution is $E(\mathcal{A}) \geq \frac{1}{\lceil \log 4D \rceil} \sum_{i=1} \mathcal{C}_i = \frac{1}{\lceil \log 4D \rceil} \mathcal{C} \geq \frac{OPT}{6\lceil \log 4D \rceil}$, since, by Lemma 2.10, $\mathcal{C} \geq \frac{OPT}{6}$. □

We also point out that our algorithms also apply to trees with any uniform capacity. In fact, *any* c-competitive algorithm for call admission on trees of capacity 1 can be converted into a $2\frac{e^{1/c}}{e^{1/c}-1} \leq (2(c+1))$-competitive algorithm for call admission on trees with arbitrary uniform capacity. Given any c-competitive algorithm for the capacity-1 case, we run a "first fit"-based technique [1] that uses k copies of the original algorithm on k copies of the tree but each with capacity 1. If the call is accepted by any of the algorithms we accept it into the actual k-capacity tree. Using [1, 9] we get for this case that we have a $\frac{e^{1/c}}{e^{1/c}-1}$ -competitive algorithm with respect to an adversary that uses k distinct, capacity-1 trees. Since this adversary is only a $1/2$ fraction away from the original adversary that has a single tree of uniform capacity k (cf. [4]), we obtain a $2\frac{e^{1/c}}{e^{1/c}-1} \leq (2(c+1))$-competitive algorithm for the problem.

5. Routing on meshes with high probability. In this section we propose a randomized algorithm for the on-line maximum edge-disjoint paths problem on meshes that has a logarithmic competitive ratio. The algorithm achieves the best possible competitive ratio up to a constant multiplicative factor *and* a benefit close to the expectation with high probability on any sequence of a large class of input instances.

Kleinberg and Tardos [16] presented the first $O(\log n)$-competitive algorithm (denoted KT in the following) for the on-line maximum edge-disjoint paths problem on meshes. They partition the input sequence into two classes: *short calls* and *long calls*. Define the distance between two vertices of the mesh as the number of edges on a shortest path connecting the two vertices. According to the partitioning of KT, every call with endpoints at distance bigger than a given value $d = \Theta(\log n)$ is a long call. KT decides with equal probability to accept only long calls or only short calls. KT has thus the drawback of obtaining benefit 0 with probability at least 1/2 on any input sequence containing only calls with endpoints at distance bigger than d (or only calls with endpoints at distance at most d).

We propose an algorithm with a logarithmic competitive ratio that achieves a constant fraction of its expected benefit with probability tending to 1 as the size of the optimal solution grows. Our algorithm for meshes is composed of a randomized stage followed by a deterministic stage. The first stage of the algorithm is a *randomized filter* that selects a subset $\mathcal{C}$ of *candidate calls* out of the input sequence σ. Calls of $\sigma \setminus \mathcal{C}$ are discarded to make space for the routing of candidate calls. The calls in $\mathcal{C}$ form a new input sequence for the second stage of the algorithm, which is a completely deterministic procedure. The subset of $\mathcal{C}$ accepted by the second stage is the set of calls finally accepted by the algorithm. Our algorithm uses a routing strategy different from that of KT. Some of its ideas are borrowed from the algorithm of Bartal and Leonardi [6] for on-line path coloring on meshes [6] .

Let us denote by $ON(\mathcal{T})$ and $OPT(\mathcal{T})$ the subsets, of some sequence $\mathcal{T}$, that are accepted (out of $\mathcal{T}$) by our on-line algorithm and by the optimal solution, respectively, and their cardinalities.

We prove (see Lemma 5.2) that $E(OPT(\mathcal{C}))$ is at least a constant fraction of $OPT(\sigma)$ and that $OPT(\mathcal{C})$ is within a constant fraction of $E(OPT(\mathcal{C}))$ with probability tending to 1 as the optimal solution grows. We present the deterministic procedure in section 5.2. We denote by $ON_D(\mathcal{T})$ the set of calls accepted by the deterministic procedure, out of some sequence $\mathcal{T}$, and its cardinality. We prove (see Lemma 5.8) that for any sequence of candidates $\mathcal{C}$, $OPT(\mathcal{C}) = O(\log n)ON_D(\mathcal{C})$. Combining the two statements (for the randomized filter and for the deterministic procedure), we conclude with the following theorem.

THEOREM 5.1. *There exists an $O(\log n)$-competitive algorithm for call control on the $n \times n$ mesh such that for any $\delta \in (0, 1]$ and any sequence σ, $Pr[ON(\sigma) \geq (1-\delta)E(ON(\sigma))] \geq 1 - 2\exp(-\frac{\delta^2 OPT(\sigma)}{O(\log^4 n)})$.*

Our algorithm thus has asymptotically optimal $O(\log n)$-competitive ratio and it is close to the expectation with probability tending to 1 as $\frac{OPT(\sigma)}{\log^4 n}$ tends to infinity.

5.1. The randomized filter. The $n \times n$ two dimensional mesh G is partitioned into a set of disjoint squares. Let $B = \lfloor \log n \rfloor$. We assume n large enough such that $B \geq 16$. Let $L = \lfloor \gamma \log n \rfloor$ for $\gamma = 13$. Let $s = \lfloor \frac{n}{L} \rfloor$, and let $s_1 = n \bmod L = n - s \cdot L$. We partition the mesh into $s \times s$ submeshes of logarithmic size. The partition of the mesh is obtained by segmenting each of its sides into $s - s_1$ contiguous segments of size L and then s_1 segments of size $L+1$. Note that with the above assumption on B this segmentation is always attainable, and that s_1 can be 0, in which case each side is segmented into s segments of size L. Every submesh is called a *square*, even if the size of its two sides differs by 1.

The first *ring* in a square S consists of all nodes of S that either are incident to a node outside of S or are on the border of the mesh. Recursively, the *ith ring* of S

for $i > 1$ 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 first ring of a square is also called the *border* of the square. Denote by S_v the square containing vertex v.

In any square S we define three *regions* S^1, S^2, and S^3. Region S^1 consists of rings 1 to $2B$, region S^2 consists of rings $2B+1$ to $4B$, and region S^3 is formed by rings $4B+1$ to $6B$. The remaining part of S is called the *central region* of S (see Figure 5.1). The central region is a rectangle with sides of size at least B.

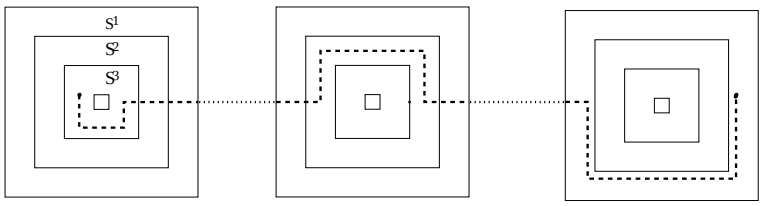

FIG. 5.1. *The routing of a long call.*

The randomized filter makes the following random choice before starting the processing of the sequence.

For every square S choose one of the regions S^1, S^2, S^3 with equal probability.

The region that has been chosen for square S is called the *selected region* of S.

Every call $(s,t) \in \sigma$, when presented, is submitted to the following test:

Add call (s,t) to $\mathcal{C}$ if s is not in the selected region of S_s *and* t is not in the selected region of S_t, otherwise discard (s,t).

As a result of this filtering procedure, no call (s,t) of $\mathcal{C}$ has an endpoint in the selected region of a square. The selected region will be used later to route calls through the square without blocking calls with endpoints inside the square.

LEMMA 5.2. *For any input sequence σ, $E(OPT(\mathcal{C})) \geq \frac{1}{3}OPT(\sigma)$. For any constant $\delta \in (0,1]$, for any sequence σ, $Pr[OPT(\mathcal{C}) \geq (1-\delta)E(OPT(\mathcal{C}))] \geq 1 - 2\exp(-\frac{OPT(\sigma)\delta^2}{82(\lfloor \gamma \log n \rfloor)^4})$.*

Proof. For the first part of the lemma we prove that every call (s,t) of $OPT(\sigma)$ is part of set $\mathcal{C}$ with probability at least $1/3$. A call (s,t) becomes a candidate if both s and t are not in the selected region of S_s and S_t. We distinguish between (i) calls with both endpoints in the same square and (ii) calls with endpoints in different squares.

(i) $S_s = S_t$. If s and t are in the same region of S_s, then this region is not selected with probability $2/3$. If s and t are in different regions, the probability that the region not containing s and t is selected is $1/3$. The claim then follows.

(ii) $S_s \neq S_t$. Vertex s (resp., t) is outside the selected region of S_s (resp., S_t) with probability $2/3$. The probability that both s and t are outside a selected region is then at least $4/9$. The first part of the claim is thus proved.

For the second part of the claim we use the "independent bounded differences inequality" in the formulation proposed by Maurey.

LEMMA 5.3 (see [10]). *Let $X_1, \ldots, X_m$ be independent random variables with X_k taking values in a set A_k for each k. Suppose that the (measurable) function $f : \prod_{k=1}^{m} A_k \to \Re$ satisfies $|f(\underline{x}) - f(\underline{x}')| \leq c_k$, whenever the vectors $\underline{x}$ and $\underline{x}'$ differ only in the kth coordinate, for some constant c_k. Let Y be the random variable $f(X_1, \ldots, X_m)$. Then, for any $t > 0$,*

$$Pr[|Y - E(Y)| \geq t] \leq 2\exp\left(\frac{-2t^2}{\sum_k c_k^2}\right).$$

In our problem we consider $m = s^2 = (\lfloor \frac{n}{L} \rfloor)^2$ independent random variables $X_1, \ldots, X_m$ to indicate the selected region for every square. Every random variable assumes one of three values with equal probability. For the function $f(\underline{x})$, we will use the function $|OPT(\sigma) \cap \mathcal{C}(\underline{x})|$, where $\underline{x}$ ranges over all the possible assignments of $\underline{x} = (X_1, \ldots, X_m)$. Note that $|OPT(\mathcal{C}(\underline{x}))| \geq |OPT(\sigma) \cap \mathcal{C}(\underline{x})|$. A different assignment of variable X_k results in a maximum absolute variation c_k for $f(\underline{x})$, where c_k is equal to the number of calls of $OPT(\sigma)$ with an endpoint in the kth square. For a square that does not contain any endpoint of a call of the optimal solution, we have $c_k = 0$.

Let K be the set of squares that contain the endpoint of at least one call of the optimal solution. We also denote by K the cardinality of the set K. Every square contains at most $2(L+1)^2$ edges. This is clearly a bound on the number of calls with both endpoints in the same square that can be accepted in a solution. The number of calls with only one endpoint in a square that can be accepted in a solution is bounded by the the number of edges that have one endpoint in the square and one endpoint outside of it, i.e., $4(L+1)$. Altogether, $c_k \leq 3(L+1)^2$ (using that L is large enough) for any square $k \in K$.

We have $\mu = E_{\underline{x}}(OPT(\sigma) \cap \mathcal{C}(\underline{x})) \geq \frac{1}{3} OPT(\sigma)$, and clearly $OPT(\sigma) \geq K/2$. From the bounded differences inequality of Lemma 5.3 it follows that

$$
\begin{aligned}
&Pr[|OPT(\sigma) \cap \mathcal{C}(\underline{x})| < \delta\mu] < 2\exp\left(\frac{-2(\delta\mu)^2}{\sum_k c_k^2}\right) \\
&\leq 2\exp\left(\frac{-2(\delta\frac{OPT(\sigma)}{3})^2}{K \cdot 9(L+1)^4}\right) \leq 2\exp\left(\frac{-\delta^2 OPT(\sigma)}{81(L+1)^4}\right) \\
&\leq 2\exp\left(\frac{-\delta^2 OPT(\sigma)}{82L^4}\right) \leq 2\exp\left(\frac{-\delta^2 OPT(\sigma)}{82(\lfloor \gamma \log n \rfloor)^4}\right),
\end{aligned}
$$

from which the claim follows. □

5.2. The deterministic procedure for meshes. The deterministic procedure receives as input the set of candidate calls $\mathcal{C}$ accepted by the randomized filter in the order they appear in the input sequence σ.

We follow KT and partition the set of calls into the set of *long* and the set of *short* calls. A call (s,t) is a short call if both s and t are in the same square and a long call if s and t are in different squares. Set $\mathcal{C}$ is partitioned into the set of long calls $\mathcal{L} = \{(s,t) \in \mathcal{C} : S_s \neq S_t\}$ and the set of short calls $\mathcal{S} = \{(s,t) \in \mathcal{C} : S_s = S_t\}$. Long and short calls are dealt with differently by the algorithm. However, as opposed to KT, we will accept both long calls and short calls at the same time.

The selected region of every square is dedicated to the routing of long calls through the square. For every square S, we number the columns from left to right and the rows from top to bottom. The routes assigned to long calls will traverse the border between two adjacent squares only on columns or rows in the interval $6B+1, \ldots, 7B$. A *crossbar* row (column) of square S is defined as a path on a row (column) in the interval $6B+1, \ldots, 7B$ connecting the central region of S to the closest vertex of the border of S. For every square S and two rings a and b, with ring a of index higher than ring b, define a *straight-line extension* of ring a as a path from a corner of ring a to ring b that does not include edges of a. Observe that the straight-line extensions starting from two distinct rings of a square are edge-disjoint.

We say that a ring, a crossbar row or column, or a straight-line extension is assigned to an accepted call if they contain at least one edge of the path assigned to

the accepted call.

We first describe the algorithm for long calls and then the algorithm for short calls.

Long calls.

Every call (s,t) of $\mathcal{L}$ is first submitted to the following test:

1. If a short or a long call with endpoint in S_s or S_t has been previously accepted, then discard (s,t). Otherwise add (s,t) to set $\mathcal{L}'$.

Calls of $\mathcal{L}'$ are dealt with using the idea of KT to build a simulated network G', with links of higher capacity, and run AAP on this network. A vertex of G' is associated with every square of the original mesh. Two vertices of G' are connected by an edge if and only if there is at least one edge in the original mesh that connects two nodes, each on the border of one of the corresponding squares (i.e., the two squares are adjacent in the mesh).

Every call (s,t) of $\mathcal{L}'$ is transformed into a call between the two vertices of G' associated with S_s and S_t and then submitted to the AAP algorithm. If AAP accepts (s,t), then (s,t) is added to $ON_D(\mathcal{C})$, the set of calls accepted by the algorithm. Calls not accepted by AAP are discarded.

We use the AAP algorithm with the parameter $\epsilon = 6/7$. Observe that one can use AAP with this value of ϵ, since the capacity B of the edges in the simulated network is high enough. To see that, recall that we assumed $B = \lfloor \log n \rfloor \geq 16$. Thus (see also Appendix A)

$$\epsilon \log D + 1 + \epsilon \leq \frac{6}{7}\log n + 1 + \frac{6}{7} \leq \frac{6}{7}\log n + 2 \leq \frac{6}{7}(B+1) + \frac{1}{8}B \leq B \ .$$

If AAP accepts call (s,t) it also returns a route in G' that has a straightforward interpretation as a sequence of squares $S_1, \ldots, S_p$ in the original mesh with $S_1 = S_s$ and $S_p = S_t$. Our algorithm will assign call (s,t) to a path (i) from s to the border between S_1 and S_2, (ii) from the border between S_{i-1} and S_i to the border between S_i and S_{i+1}, $i = 2, \ldots, p-1$, and (iii) from the border between S_{p-1} and S_p to t. In the following description we give the details of this route assignment.

(i) From s to the border between S_1 and S_2.
Without loss of generality assume that the border between S_1 and S_2 is on a column. We consider two cases: (a) s is outside the central area; (b) s is inside the central area.

(a) The call is routed from s through the ring containing s until it meets an unassigned crossbar row leading to the border between S_1 and S_2; this crossbar row is followed until the border between S_1 and S_2.

(b) The call is routed from s through the ring containing s until a corner of the ring is reached, where a straight-line extension is followed until ring $6B$. From there the path continues as in point (a).

(ii) From the border between S_{i-1} and S_i to the border between S_i and S_{i+1}.
Without loss of generality, assume that the border between S_{i-1} and S_i and the border between S_i and S_{i+1} are on columns of G. The path enters the border between S_{i-1} and S_i on a crossbar row that is followed until an unassigned ring of the selected region. The path is routed through the ring of the selected region until an unassigned crossbar row leading to the border between S_i and S_{i+1} is met. This crossbar row is followed until the border between S_i and S_{i+1}.

(iii) From the border between S_{p-1} and S_p to t. This case is equal to case (i).

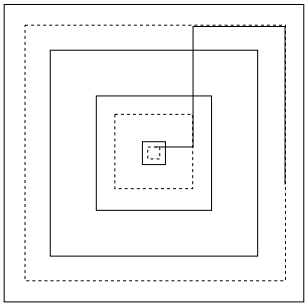

FIG. 5.2. *The routing of short calls.*

Figure 5.1 shows the routing of a long call with an endpoint in the first region and an endpoint in the third region.

Short calls.

A short call (s, t) is accepted on the basis of the following test:

1. If the ring containing s or the ring containing t has been previously assigned, then discard (s, t).
2. If either s or t, but not both, is in the central area, the other vertex is in region S^1 or S^2, but all rings in the range from $4B + 1$ to $6B$ are already assigned, then discard (s, t).
3. Otherwise, add (s, t) to $ON_D(\mathcal{C})$.

In the following we describe the route assigned to an accepted call (s, t).

If s and t are on the same ring, we route (s, t) on a path contained in the ring of s and t. Otherwise, we assume without loss of generality that the ring of s is internal to the ring of t and distinguish between two cases:

1. Both s and t are inside or are both outside the central area. We route (s, t) through the ring of s until a corner of the ring. There, we continue on a straight-line extension until the ring of t, which is then followed until t itself.
2. Either s or t, but not both, is in the central area. One of the rings from $4B + 1$ to $6B$ must be unassigned and will be assigned to (s, t) (possibly the ring containing t). A straight-line extension from a corner of the ring of s to an unassigned ring of S^3 not intersecting the crossbar row or column assigned to the single long call with endpoint in S_s is assigned to (s, t). The path follows the ring of s until the corner of the selected straight-line extension that is followed until the assigned ring of S^3. There, if t is in S^3, we follow the ring until t, otherwise we continue as at point 1.

Figure 5.2 shows the routing of short calls.

Proof of correctness. We prove that the algorithm described above routes accepted calls on edge-disjoint paths. To that end, we prove the following lemmas.

LEMMA 5.4. *The maximum number of calls routed between two adjacent squares is B.*

Proof. Each edge in the simulated network has capacity B and AAP does not violate the capacity constraints. □

LEMMA 5.5. *Let S be a square. There are at most $2B$ calls routed through square S.*

Proof. Every call routed through a square consumes two units of bandwidth on the edges incident to the vertex of G' associated with the square. The overall

bandwidth on the edges incident to a vertex of G' is $4B$; thus at most $2B$ calls are routed through a square. □

LEMMA 5.6. *The following claims hold at any time during the execution of the algorithm:*

1. *Every crossbar row or column is assigned to at most a single call.*
2. *Every ring is assigned to at most a single call.*
3. *Every straight-line extension is assigned to at most a single call.*

Proof. Clearly the claims hold before any call is accepted. We now assume that the claims hold before call (s,t) is accepted and prove that they still hold after call (s,t) is accepted.

1. Assume (s,t) is accepted on a path crossing the border between S and S'. Without loss of generality assume that the border is on a column. By Lemma 5.4 at most $B-1$ (long) calls have been previously routed between S and S'. We conclude that there is an unassigned crossbar row of S and an unassigned crossbar row of S' leading to the border between S and S', both contained in the same row, that can be assigned to (s,t).
2. Two cases: (i) (s,t) is a long call; (ii) (s,t) is a short call. (i) We prove that the rings assigned to call (s,t) are not assigned to any other call. Call (s,t) is accepted by AAP on a path crossing a sequence of squares $S_1,\ldots,S_p$, with $S_1 = S_s$ and $S_p = S_t$. A ring for every square S_i is assigned to (s,t). We first consider squares S_s and S_t. By step 1 of the algorithm for long calls, no long or short call is previously accepted in S_s and in S_t. Since vertices s and t are outside the selected regions of S_s and of S_t, the rings containing s and t are unassigned when (s,t) is accepted. Call (s,t) is also assigned to a ring of the selected region in every square S_i, $i = 2,\ldots,p-1$. By Lemma 5.5, at most $2B-1$ rings of the selected region of a square S_i are previously assigned when (s,t) is accepted. A ring of the selected region of S_i is then still available to be assigned to (s,t). (ii) By steps 1 and 2 of the routing algorithm for short calls, if (s,t) is accepted, it is assigned to at most three rings which were not previously assigned to any other call.
3. All the straight-line extensions in a square are edge-disjoint. A straight-line extension is assigned to a call if it starts from a ring that is also assigned to that call. By point 2 of the claim, every ring is assigned to at most a single call. Therefore, every straight-line extension is assigned to at most a single call. □

We can now prove the main lemma.

LEMMA 5.7. *Every pair of accepted calls is routed on two edge-disjoint paths.*

Proof. By Lemma 5.6, every crossbar row or column, ring, and straight-line extension is assigned to at most a single call. Rings and straight-line extensions are edge-disjoint, as are rings and crossbar rows and columns. Only straight-line extensions from the central region of a square may intersect crossbar rows or columns. We prove that the edges common to a straight-line extension from a ring of the central region and to a crossbar row or column are not assigned to more than one call.

Consider square S. We first exclude intersections between calls with an endpoint in S and calls that are routed through S. By point (i)b of the routing algorithm for long calls and by step 2 of the routing algorithm for short calls, a straight-line extension from a ring of the central region is followed by the route of a call at most until a ring of S^3. In this case the selected region in S is either S^1 or S^2. Therefore, by point (ii) of the routing algorithm for long calls, a call routed through S includes

only edges of S^1 and S^2 that belong to a crossbar row or column. Therefore, there is no intersection with calls routed through S.

We now consider a long call (s,t) with an endpoint, say, s, in S and a call (s',t') assigned to a straight-line extension from a ring of the central region. By step 1 of the algorithm for long calls, (s',t') must be a short call accepted after (s,t), (s,t) being the single long call with endpoint in S that is accepted. There could be a potential intersection only if (s',t') is in the central region, and (s,t) has an endpoint in S^3. There are four possible straight-line extensions that connect the ring of s' with a ring of S^3. By step 2 of the algorithm for short calls, (s',t') is assigned to a straight-line extension that does not overlap with the crossbar row or column assigned to (s,t). No edge of S^3 on the chosen straight-line extension is thus assigned to more than one call. □

The analysis. We conclude the proof that the algorithm is $O(\log n)$-competitive by showing the following lemma.

LEMMA 5.8. *For any set of candidate calls $\mathcal{C}$, $OPT(\mathcal{C}) = O(\log n) ON_D(\mathcal{C})$.*

Proof. We will prove the following lemmas:

1. $OPT(\mathcal{L} \setminus \mathcal{L}') = O(\log n)\ ON_D(\mathcal{C})$ (Lemma 5.9).
2. $OPT(\mathcal{L}') = O(\log n)\ ON_D(\mathcal{L}')$ (Lemma 5.10).
3. $OPT(\mathcal{S}) = O(\log n)\ ON_D(\mathcal{C})$ (Lemma 5.11).

The proof then easily follows:

$$\begin{aligned} OPT(\mathcal{C}) = OPT(\mathcal{L} \cup \mathcal{S}) &\leq OPT(\mathcal{L} \setminus \mathcal{L}') + OPT(\mathcal{L}') + OPT(\mathcal{S}) \\ &= O(\log n)\ ON_D(\mathcal{C})\ . \end{aligned}$$ □

LEMMA 5.9. $OPT(\mathcal{L} \setminus \mathcal{L}') = O(\log n)\ ON_D(\mathcal{C})$.

Proof. A call (s,t) of $\mathcal{L}$ is discarded if a long or a short call with an endpoint in S_s or in S_t has been previously added to $ON_D(\mathcal{C})$. At most $4(L+1)$ long calls with an endpoint in a given square can be accepted by the optimal solution, since the border of S is formed by at most $4(L+1)$ vertices (and edges). Then, for every call accepted in $ON_D(\mathcal{C})$, at most $8(L+1)$ calls of $\mathcal{L} \setminus \mathcal{L}'$ are accepted by the optimal solution. As $L = \lfloor \gamma \log n \rfloor$ the claim follows. □

LEMMA 5.10. $OPT(\mathcal{L}') = O(\log n)\ ON_D(\mathcal{L}')$.

Proof. Every call of $\mathcal{L}'$ is submitted to the AAP algorithm on G' with edges of bandwidth $B = \lfloor \log n \rfloor$. Note that this capacity is large enough to allow AAP to work correctly if we use the AAP algorithm with $\epsilon = 6/7$ (and using the assumption that $B \geq 16$). The benefit $OPT(\mathcal{L}')$, obtained by an optimal solution on the set $\mathcal{L}'$, is bounded by the benefit obtained by an optimal algorithm on G' with edges of bandwidth $L+1$, which is the maximum number of calls that can be routed between two adjacent squares. The AAP algorithm is still $O(\log n)$-competitive even if the bandwidth used by the on-line algorithm (AAP) on the edges is smaller by a constant multiplicative factor than the bandwidth used by the off-line algorithm (see [2, 16]; for completeness this is also proved in the appendix). In our case this constant factor is at most 14. The claim then follows. □

LEMMA 5.11. $OPT(\mathcal{S}) = O(\log n)\ ON_D(\mathcal{C})$.

Proof. Let us restrict our attention to a single square. Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be the set of calls discarded at steps 1 and 2 of the algorithm for short calls. We have $OPT(\mathcal{S}) \leq OPT(\mathcal{S}_1) + OPT(\mathcal{S}_2) + ON_D(S)$. The proof of the lemma derives from the two following statements: (i) $OPT(\mathcal{S}_1) = O(\log n)\ ON_D(\mathcal{C})$; (ii) $OPT(\mathcal{S}_2) \leq 8\ ON_D(\mathcal{C})$.

(i) For every accepted call, at most three rings of a square are assigned by the algorithm to that call. The number of edges incident to a ring is at most $4(L+1)$. This

is a bound on the number of calls with an endpoint on a ring that can be accepted in an optimal solution. Therefore, the number of calls of $OPT(\mathcal{S})$ that are discarded at step 1 of the algorithm for short calls, per each call in $ON_D(\mathcal{C})$, is at most $12(L+1)$. As $L = \lfloor \gamma \log n \rfloor$, this proves the first statement.

(ii) Short calls with an endpoint in the central region and an endpoint outside the central region are discarded at step 2 of the algorithm if all the rings of S^3 outside the central region are assigned. At most two rings of S^3 are assigned to an accepted call. If a short call is discarded at step 2, we thus have the evidence that B calls with an endpoint in S have already been accepted. For the sake of the proof we charge, for every accepted long call, the value of $1/2$ to S and the value of $1/2$ to the square containing its other endpoint. A square is therefore already charged with at least the value of $B/2$ if a short call of the square is discarded at step 2. The number of short calls of $OPT(\mathcal{S}_2)$ with endpoints in S is bounded by $4B$, as this is the number of vertices on the border of the central region. The ratio between the number of calls of $\mathcal{S}_2$ with endpoints in S that belong to $OPT(\mathcal{S}_2)$, and the number of calls of $ON_D(\mathcal{C})$ charged to S, is thus at most 8. This proves the second statement. □

Appendix A. The AAP algorithm. For completeness we describe a restricted version of the AAP algorithm. We consider the case where all calls are of infinite duration, request bandwidth of 1, and are of uniform benefit that without loss of generality we assume equals to D, the length of the longest simple path in the network. Let $u(e)$, for $e \in E$, be the capacity of edge e and assume that for all $e \in E$,

$$u(e) \geq \epsilon \log D + 1 + \epsilon \quad \text{for some } 0 < \epsilon \leq 1. \tag{A.1}$$

Let $b_j(e)$ be the number of calls routed trough edge e by AAP among the first j presented calls. Define the relative load of edge e, just after call j has been processed, to be $\lambda_j(e) = \frac{b_j(e)}{u(e)}$. Define the cost of edge e just after call j has been processed to be $c_j(e) = u(e)[\mu^{\lambda_j(e)} - 1]$, where $\mu = 2^{1+1/\epsilon}D$. A request (s_j, t_j) is accepted to path P_j if $\sum_{e \in P_j)} \frac{1}{u(e)} \cdot c_{j-1}(e) \leq D$.

The above algorithm has competitive ratio $O(2^{1/\epsilon}/\epsilon + 2^{1/\epsilon} \log D)$. We give below a sketch of the proof. The proof follows the proof that appears in [8].

THEOREM A.1. *The above AAP algorithm has competitive ratio* $2^{1+1/\epsilon} \log \mu + 1$.

Proof. Let $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_n$, be the sequence of requests. Let A be the set of indices of requests that are accepted by AAP and let A' be the set of indices of requests that are rejected by AAP but are accepted by the adversary. Let $C = \sum_{e \in E} c_n(e)$. Let $\mathcal{Q}(\sigma)$ denote the benefit obtained by algorithm AAP on sequence σ. We prove the following two claims that yield the desired result.

1. $C \leq (2^{1+1/\epsilon} \log \mu) \cdot |A| \cdot D$.
2. $|A'| \cdot D \leq C$.

The result follows from the above two inequalities since $\mathcal{Q}(\sigma) = |A| \cdot D$, and $OPT(\sigma) \leq \mathcal{Q}(\sigma) + |A'| \cdot D$.

The proof of inequality 1 can be found in [8]. We repeat here the proof of inequality 2 since we slightly modify it below to give the corollaries we need.

For a call σ_j that is accepted by the off-line algorithm let P_j^* be the path that is used by the off-line algorithm to accept this call. For any $j \in A'$

$$D < \min_{P_j} \sum_{e \in P_j} \frac{1}{u(e)} c_{j-1}(e) \leq \sum_{e \in P_j^*} \frac{1}{u(e)} c_{j-1}(e) \ .$$

We get

$$\begin{aligned}
|A'| \cdot D &< \sum_{j \in A'} \sum_{e \in P_j^*} \frac{1}{u(e)} c_{j-1}(e) \\
&\leq \sum_{j \in A'} \sum_{e \in P_j^*} \frac{1}{u(e)} c_n(e) \\
&= \sum_e c_n(e) \sum_{j \in A' : e \in P_j^*} \frac{1}{u(e)} \\
&\leq \sum_e c_n(e) \\
&= C.
\end{aligned}$$

The last inequality follows since the capacity of the adversary on any edge e is $u(e)$, that is, it can route at most $u(e)$ calls though edge e. □

We now give two corollaries of the above theorem.

COROLLARY A.2. *If for all $e \in E$, $u(e)$ satisfied condition* A.1, *and the off-line algorithm has capacity $c \cdot u(e)$ for each edge e, for some constant c, then AAP is $(1 + c(2^{1+1/\epsilon} \log \mu))$-competitive.*

Proof. We slightly modify the above proof for inequality 2. Since for each $e \in E$ the capacity available to the off-line algorithm is $c \cdot u(e)$, then the last inequality in the proof would yield

$$|A'| \cdot D < \sum_e c_n(e) \cdot c = C \cdot c \ .$$

Thus we get $(|A'| \cdot D)/c \leq (2^{1+1/\epsilon} \log \mu) \cdot |A| \cdot D$. And we get

$$\begin{aligned}
OPT(\sigma) &\leq \mathcal{Q}(\sigma) + |A'| \cdot D \leq \mathcal{Q}(\sigma) + c(2^{1+1/\epsilon} \log \mu) \cdot |A| \cdot D \\
&= (1 + c(2^{1+1/\epsilon} \log \mu)) \cdot \mathcal{Q}(\sigma). \qquad \square
\end{aligned}$$

COROLLARY A.3. *If AAP, with $\epsilon = 1$, has for each $e \in E$ capacity $u(e) = \log 4D$, while the off-line algorithm has for all e capacity* 1, *then the competitive ratio is* 5.

Proof. We use $\epsilon = 1$ and slightly modify the above proof of inequality 2. We again modify the last inequality of the the proof. Since the off-line algorithm has for each e only capacity $u(e)/\log 4D$, we get

$$|A'| \cdot D < \sum_e c_n(e)/\log 4D \ ,$$

i.e.,

$$|A'| \cdot D < C/\log 4D \ .$$

We get

$$\begin{aligned} OPT(\sigma) &\le \mathcal{Q}(\sigma) + |A'| \cdot D \\ &\le \mathcal{Q}(\sigma) + C/\log 4D \\ &\le \mathcal{Q}(\sigma) + ((2^{1+1/\epsilon}\log\mu)\cdot|A|\cdot D)/\log 4D \\ &= \mathcal{Q}(\sigma) + ((2^{1+1/\epsilon}\log\mu)\cdot|A|\cdot D)/\log\mu \\ &= \mathcal{Q}(\sigma) + 4\cdot|A|\cdot D = 5\cdot\mathcal{Q}(\sigma)\ . \qquad \square \end{aligned}$$

Acknowledgments. Special thanks are due to Amos Fiat for suggesting to us to look into the relationship between the competitive ratio and the deviation of the result from its expectation in randomized benefit on-line algorithms. We thank Dimitri Achlioptas and Hannah Bast for useful discussions and suggestions. We also thank an anonymous referee for very useful comments.

REFERENCES

[1] B. Awerbuch, Y. Azar, A. Fiat, S. Leonardi, and A. Rosén, *On-line competitive algorithms for call admission in optical networks*, in Proceedings of the Fourth European Symposium on Algorithms, Lecture Notes in Comput. Sci. 1136, Springer, Berlin, 1996, pp. 431–444.

[2] B. Awerbuch, Y. Azar, and S. Plotkin, *Throughput-competitive online routing*, in Proceeding of the 34th Symposium Foundations of Computer Science, 1993, pp. 32–40.

[3] B. Awerbuch, Y. Bartal, A. Fiat, and A. Rosén, *Competitive non-preemptive call control*, in Proceedings of the Fifth Annual Symposium on Discrete Algorithms, Arlington, VA, 1994, pp. 312–320.

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

[5] 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 Symposium Theory of Computing, 1996, pp. 531–540.

[6] Y. Bartal and S. Leonardi, *On-line routing in all-optical networks*, in Proceedings of the 24th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Comput. Sci. 1256, Springer, Berlin, 1997, pp. 516–526.

[7] S. Ben-David, A. Borodin, R. M. Karp, G. Tardos, and A. Wigderson, *On the power of randomization in on-line algorithms*, Algorithmica, 11 (1994), pp. 2–14.

[8] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, Cambridge, UK, 1998; also available online from http://www.cup.org/Titles/56/0521563925.html.

[9] M. F. G. Cornuejols and G. Nemhauser, *Location of bank accounts to optimize float*, Management Sci., 62 (1977), pp. 789–810.

[10] C. M. Diarmid, *On the method of bounded difference*, in Surveys in Combinatorics, London Math. Soc. Lecture Note Ser. 141, J. Siemon, ed., Cambridge University Press, Cambridge, UK, 1989.

[11] N. Garg, V. V. Vazirani, and M. Yannakakis, *Primal-dual approximation algorithms for integral flow and multicut in trees, with applications to matching and set cover*, in Proceedings of the 20th International Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 700, Springer, 1993, pp. 64–75.

[12] V. Guruswami, S. Khanna, R. Rajaraman, B. Shepherd, and M. Yannakakis, *Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC99), 1999, pp. 19–28.

[13] R. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–103.

[14] J. Kleinberg, *Approximation Algorithms for Disjoint Paths Problems*, Ph.D. thesis, MIT, Cambridge, MA, 1996.

[15] J. Kleinberg and R. Rubenfield, *Short paths in expander graphs*, in Proceedings of the 37th Symposium Foundations of Computer Science, 1996, pp. 86–95.

[16] J. Kleinberg and E. Tardos, *Disjoint paths in densely embedded graphs*, in Proceedings of the 36th Symposium Foundations of Computer Science, 1995, pp. 52–61.

[17] M. Kramer and J. van Leeuwen, *The complexity of wire routing and finding the minimum area layouts for arbitrary vlsi circuits*, in Advances in Computing Research 2: VLSI Theory, F. Preparata, ed., JAI Press, London, 1984, pp. 129–146.

[18] S. Leonardi and A. Marchetti-Spaccamela, *On-line resource management with applications to routing and scheduling*, in Proceedings of the 22nd International Colloquium on Automata, Languages, and Programming, Lecture Notes in Comput. Sci. 944, Springer, Berlin, 1995, pp. 303–314.

[19] P. Raghavan, *Probabilistic construction of deterministic algorithms: Approximating packing integer program*, J. Comput. System Sci., 2 (1988), pp. 130–143.

[20] D. Sleator and R. E. Tarjan, *Amortized efficiency of list update and paging rules*, Comm. ACM, 28 (1985), pp. 202–208.