CCIT Report #347
July 2001

A Scalable Approach to the Partition of QoS Requirements in Unicast and Multicast*

Ariel Orda *Alexander Sprintson*
Department of Electrical Engineering
Technion—Israel Institute of Technology
Haifa 32000, Israel

Abstract

Supporting Quality of Service (QoS) in large-scale broadband networks poses major challenges, due to the intrinsic complexity of the corresponding resource allocation problems. An important problem in this context is how to partition QoS requirements along a selected topology (path for unicast, tree for multicast). As networks grow in size, the scalability of the solution becomes increasingly important. This requires to devise ef£cient algorithms, whose computational complexity is less dependent on the network size. In addition, recently proposed *precomputation*-based methods can be employed to facilitate scalability by signi£cantly reducing the time needed for handling incoming requests.

We present a novel solution technique to the QoS partition problem(s), based on a "divide and conquer" scheme. As opposed to previous solutions, our technique considerably reduces the computational complexity in terms of dependence on network size; moreover, it enables the development of precomputation schemes. Hence, our technique provides a scalable approach to the QoS partition problem, for both unicast and multicast. In addition, our algorithms readily generalize to support QoS routing in typical settings of large-scale networks.

Index Terms

QoS partition, Performance-dependent costs, Multicast, Routing, Resource allocation.

I. Introduction

Future communication networks are expected to support applications with quality of service (QoS) requirements. Supporting QoS poses major challenges due to the large size and complex structure of networks. A key issue in the design of broadband architectures is how to allocate network resources in order to meet end-to-end QoS requirements in a way that maximizes the overall network performance. Several network mechanisms need to be introduced to support QoS. One is a QoS routing mechanism, whose purpose is to £nd a suitable topology (path for unicast, tree for multicast) that can support the connection(s) QoS requirements. Then, a second mechanism is required, in order to optimally allocate

*A preliminary version appeared in the Proceedings of IEEE Infocom'02, New York, NY, USA, June, 2002.

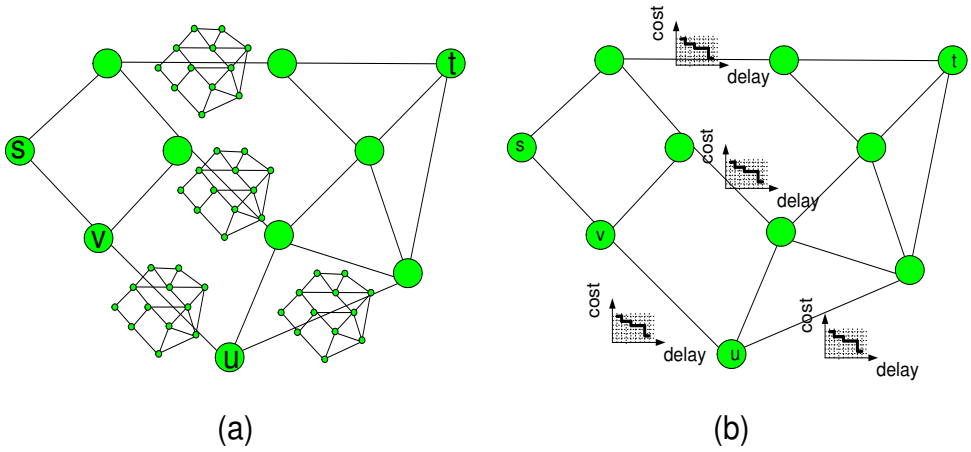

Fig. 1. (a) Original network (b) Aggregated network; subnetworks are represented by link cost functions

resources (*e.g.*, bandwidth, buffer space) along the selected topology such that the required QoS can be guaranteed at minimal cost.

A network link (or element) can offer several levels of QoS guarantees, each associated with a certain cost. The link's cost represents the consumption of local resources that must be reserved on the link in order to support the QoS guarantee. For example, in the DiffServ architecture [1] a service provider can offer several types of service at different prices. Moreover, links may aggregate subnetworks (*e.g.*, accordingly to the ATM PNNI recommendations [9]), in which case each link represents several paths that support different QoS requirements at different cost values. Accordingly, we consider a network model, in which each link is associated with a *performance-dependent cost function*. For example, Fig. 1(b) shows an aggregated network that corresponds to the original network depicted on Fig. 1(b). Each link in the aggregated network is associated with a cost-delay function that represents the corresponding subnetwork.

The problem of optimal partition of QoS requirements was formulated in [6] and has been the subject of several studies [2], [3], [5], [7], [10]. Ef£cient optimal solutions for the special case of *convex* cost functions for both unicast and multicast were established in [6]. However, the convexity assumption is not valid in many cases of practical interest. Since in the general case the problem of optimal partition is intractable (*i.e.*, $\mathcal{NP}$-hard [6]), suitable approximation schemes were presented in [2], [7], [10]. While the computational complexity of those approximations is polynomial, it depends heavily on the size of the topology, which renders these solutions *unscalable*. The high complexity, in turn, results in a high response time to each connection request, which adversely affects the service to network users.

Accordingly, the purpose of this study is to provide scalable solution schemes to the problem. This is achieved in two ways. First, we establish algorithmic solutions that are considerably less dependent on the size of the routing topology than previous proposals. Second (and independently), we employ a *precomputation* approach, in order to further enhance scalability. We proceed to discuss each of these two

contributions.

The major contribution of this study is a novel solution technique that better exploits the speci£c structure of routing topologies (paths and trees). More speci£cally, we employ a divide-and-conquer scheme, which £rst computes the costs of supporting various QoS requirements through smaller components (subpaths and subtrees), and then combines the results in order to obtain solutions for larger components. Our technique allows to easily distribute the computational effort among network nodes. Furthermore, it can be generalized to handle the combined problem of routing and partition of QoS in typical settings of large-scale networks.

Precomputation-based methods have recently been proposed [4], [8] (in the context of QoS routing) as an instrument to facilitate scalability, improve response time and reduce the computational load on network elements. The key idea is to effectively reduce the time needed to handle a request, by performing a certain amount of computations in *advance*, *i.e.*, prior to the request's arrival. Such advance computations are performed as background processes, *i.e.*, when a network element is idle or underutilized, thus resulting in better utilization of the computational capabilities of network elements. In addition, when the rate of incoming requests is high, a considerable reduction in overall computational load is achieved. Accordingly, we employ the precomputation approach in order to improve the scalability of our solutions.

Precomputation is performed by means of a two-phase procedure, referred to as a *precomputation scheme*. The £rst phase is executed in advance and its purpose is to precompute the optimal partition *a priori*, for each delay constraint supported by the path or tree. The computations performed at this phase are then summarized into a database for later usage. The purpose of the second phase is to provide an adequate solution on demand, *i.e.*, upon an incoming request. The second phase either selects one of the solutions precomputed at the £rst phase, or, if necessary, performs additional computations.

The rest of this paper is organized as follows. In Section II, we formulate the network model and formally state the considered problems. Section III deals with unicast topologies and presents solutions both for performing on-demand computation as well as precomputation. Section IV presents similar solutions for the much more complex setting of multicast. Finally, conclusions are presented in Section V.

II. Model and Problem Formulation

This section formulates the general model and main problems addressed in this paper. For clarity of presentation, we focus here on unicast; the de£nitions and terminology for multicast are presented in Section IV.

A *network* is represented by a directed graph $G(V, E)$, where V is the set of nodes and E is the set of links. Let $N = |V|$ and $M = |E|$. A *path* is a £nite sequence of nodes $\mathcal{P} = \{v_0, v_1, \cdots, v_n\}$, such that, for $0 \leq i \leq n-1$, $(v_i, v_{i+1}) \in E$; $n = |\mathcal{P}|$ is then said to be the *number of hops* (*or hop count)* of $\mathcal{P}$. The subpath of $\mathcal{P}$ that extends from v_i to v_j is denoted by $\mathcal{P}_{(v_i,v_j)}$. We assume that the connection's topology, *i.e.*, a path $\mathcal{P}$, is given.

Each link $l \in E$ offers different (integer) QoS guarantees $\{d_l\}$, whose signi£cance depends on the type of considered QoS requirement. For example, when the QoS requirement is an upper bound on the

end-to-end delay, the values $\{d_l\}$ are delay guarantees supported by link l. A QoS partition on a unicast path $\mathcal{P}$ is a set $\{d_l\}_{l\in\mathcal{P}}$ of local QoS requirements, which satis£es the end-to-end QoS requirement D.

QoS requirements may be *additive*, such as delay and jitter, or *bottleneck*, such as bandwidth. As is easy to verify, the QoS partition problem is straightforward for bottleneck metrics, hence we focus on additive QoS requirements. In other words, a partition of a QoS requirement D is a set $\{d_l\}_{l\in\mathcal{P}}$ on a path $\mathcal{P}$ such that $\sum_{l\in\mathcal{P}} d_l \leq D$. For clarity of presentation and without loss of generality, we describe our model and problems in terms of end-to-end delay requirements.

For each link $l \in E$, there is a *link cost function*, $c_l(d)$, which assigns a cost to each delay guarantee d that the link offers. We assume the $c_l(d)$ is higher for tighter delay constraints, *i.e.*, the function $c_l(d)$ is monotonically decreasing. For clarity of presentation we assume that if a delay guarantee d is not supported by a link l, then $c_l(d) = \infty$. The link cost function estimates the quality of the link in terms of resource utilization; it may depend on various factors, *e.g.*, the link's available bandwidth, its location, *etc.*. The link cost function can be speci£ed by either an algebraic expression or by a table that speci£es costs for supporting various delay guaranties. In the latter case, we say it is a *discrete* cost function. We shall assume that all parameters (both delay guaranties and costs) are (positive) integers. The overall cost of a partition $\{d_l\}_{l\in\mathcal{P}}$ is the sum of the local costs, *i.e.*, $\sum_{l\in\mathcal{P}} c_l(d_l)$.

The optimal QoS partition problem is then de£ned as follows.

Problem OPQ (Optimal Partition of QoS): Given a path $\mathcal{P} = \{v_0, \cdots, v_n\}$ and a delay constraint D, £nd a QoS partition $\{d_l\}_{l\in\mathcal{P}}$ such that $\sum_{l\in\mathcal{P}} d_l \leq D$ and $\sum_{l\in\mathcal{P}} c_l(d_l)$ is minimized.

The solution $\{\hat{d}_l\}_{l\in\mathcal{P}}$ of Problem OPQ is referred to as an *optimal partition of a QoS requirement D along $\mathcal{P}$*.

Fig.2 demonstrates an instance of Problem OPQ. Suppose we need to establish a connection with a delay requirement 8 between v_1 and v_4. For this purpose we use path $\mathcal{P} = \{v_0, \cdots, v_4\}$, with link cost functions, as depicted in Fig. 2. The optimal partition for this instance is $\{2, 2, 1, 3\}$, *i.e.*, the delay requirement for the £rst link is 2, for the second link is 2, *etc.* The cost of the optimal partition is 17.

As mentioned in the Introduction, we devise ef£cient schemes for precomputation of optimal partitions for a wide range of delay constraints. The related problem is de£ned as follows:

Problem POPQ (Precomputation of Optimal Partition of QoS): Given a path $\mathcal{P} = \{v_0, \cdots, v_n\}$, £nd, *for each delay requirement D*, a QoS partition $\{d_l\}_{l\in\mathcal{P}}$ such that $\sum_{l\in\mathcal{P}} d_l \leq D$ and $\sum_{l\in\mathcal{P}} c_l(d_l)$ is minimized.

For clarity of presentation we make the following simplifying assumptions:

1) The number of links n in path $\mathcal{P}$ is a power of 2, *i.e.*, $n = 2^K$, for some integer K.
2) Given delay constraint d, the cost $c_l(d)$ of supporting d by link l can be computed in $\mathcal{O}(1)$ time.
3) Given a cost c, the minimum delay constraint d supported by link l at cost c, can be computed in $\mathcal{O}(1)$ time.

Dropping Assumption 1 requires a mild and straightforward modi£cation of our results, with no penalty in terms of computational complexity. Assumptions 2 and 3 require that the functions $c_l(d)$ and the

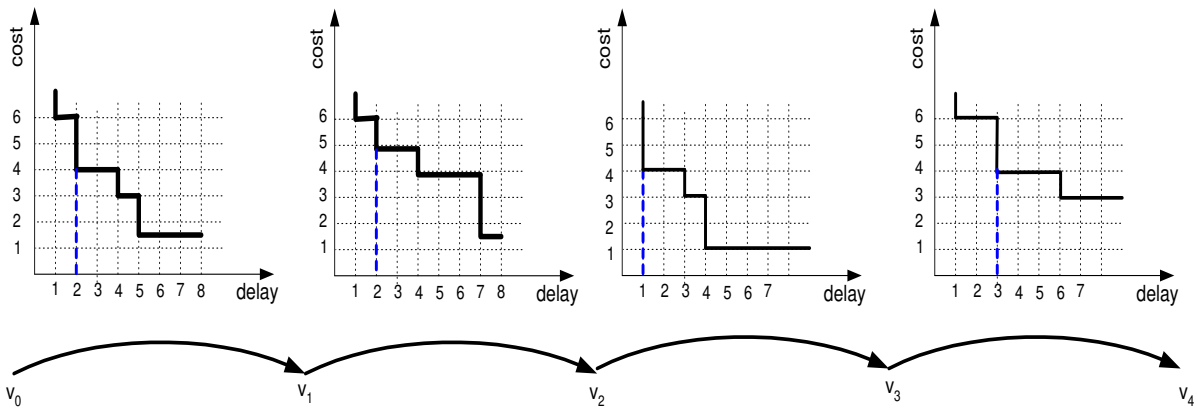

Fig. 2. An instance of Problem OPQ

corresponding inverse functions can be easily computed. Since a value of the inverse function can be computed through binary search, dropping Assumption 3 results in a small penalty in terms of computational complexity.

The computational complexity of our solutions depends on the maximum cost $C^{\max}$ of supporting a delay constraint by a link in $\mathcal{P}$. More speci£cally, let $d_l^{\min}$ be the minimum delay constraint supported by a link l, *i.e.*, $d_l^{\min} = \min_{l \in \mathcal{P}} \{d \mid c_l(d) \neq \infty\}$. Then $C^{\max} = c_l(d_l^{\min})$.

In general, Problem OPQ and Problem POPQ are intractable, *i.e.*, $\mathcal{NP}$-hard [6]. Accordingly, in this work we resort to scalable ε-approximate solutions for £xed $0 < \varepsilon \leq 1$, *i.e.*, solutions of (low) polynomial complexity, whose cost is at most $(1+\varepsilon)$ times higher than the cost of the optimal solution. Speci£cally, we present algorithms for Problems OPQ and POPQ whose computational complexity is $\mathcal{O}(\frac{1}{\varepsilon^2} n \log\left(\frac{n}{\varepsilon}\right) + n \log\log C^{\max})$ and $\mathcal{O}(\frac{1}{\varepsilon^2} n \log(nC^{\max}))$, respectively.

III. QoS Partition for Unicast

In this section we deal with the partition of QoS requirements along unicast paths. We begin by presenting our novel approximation approach. Then, we present approximation schemes for Problems OPQ and POPQ.

A. Our approach

We observe that the optimal solution to the problem of QoS partition on a path contains within it optimal solutions for its subpaths. For example in Fig. 2 the optimal partition $\{2, 2, 1, 3\}$ of delay constraint 8 for path $\mathcal{P} = \{v_0, \cdots, v_4\}$ contains within it the optimal partition $\{2, 2\}$ of delay constraint 4 for the subpath $\mathcal{P}_{(v_0, v_2)}$. Accordingly, we compute the solutions to Problems OPQ and POPQ in the following "divide-and-conquer" fashion. We recursively split the given unicast path $\mathcal{P}$ into two disjoint subpaths. We compute the set of delay guaranties supported by each subpath at different costs. These delay guaranties and the corresponding partitions are summarized by means of *delay functions*, de£ned below. We then obtain a

solution to the original problem, *i.e.*, a partition of delay constraint D on path $\mathcal{P}$, by recursively combining the delay functions obtained for the subpaths.

1) Delay Functions

We begin by de£ning a new structure, namely *optimal delay functions*, whose purpose is to summarize the delay guaranties that can be offered by subpaths at different costs.

De£nition 1: The *optimal delay function* $D^{opt}_{(v_i,v_j)}(c)$ of a subpath $\mathcal{P}_{(v_i,v_j)}$ of $\mathcal{P}$ is de£ned as the minimum delay requirement D supported by $\mathcal{P}_{(v_i,v_j)}$ at cost c, *i.e.*,

$$D^{opt}_{(v_i,v_j)}(c) = \min\left\{ D \;\middle|\; \exists\ \{d_l\}_{l\in\mathcal{P}_{(v_i,v_j)}} \text{ such that } \sum_{l\in\mathcal{P}_{(v_i,v_j)}} d_l \le D \text{ and } \sum_{l\in\mathcal{P}_{(v_i,v_j)}} c_l(d_l) \le c \right\}.$$

Note that if the subpath comprises of a single link (v_i, v_{i+1}), then $D^{opt}_{(v_i,v_{i+1})}(c)$ is the inverse of the cost function of that link, *i.e.*, $D^{opt}_{(v_i,v_{i+1})}(c) = \min\left\{d \mid c_{(v_i,v_{i+1})}(d) \le c\right\}$.

While optimal delay functions accurately capture the delays supported by subpaths of $\mathcal{P}$ at different costs, they are impractical, since their computation is intractable, and moreover, their storage requirements are prohibitively large. Accordingly, we resort to $\bar{\varepsilon}$-*approximate* delay functions, whose computation and storage requirements are feasible.

De£nition 2: An $\bar{\varepsilon}$-*approximate (subpath) delay function* $D_{(v_i,v_j)}(c)$ of a subpath $\mathcal{P}_{(v_i,v_j)}$ of $\mathcal{P}$ is a function that satis£es, for each $c \ge 0$,

$$D_{(v_i,v_j)}\left(c(1+\bar{\varepsilon})\right) \le D^{opt}_{(v_i,v_j)}\left(c\right).$$

For the sake of clarity, and when no ambiguity exists, $\bar{\varepsilon}$-approximate delay functions shall be referred to as just *delay functions*.

2) Logarithmic Sampling

Approximate delay functions can be constructed from optimal delay functions by employing *logarithmic sampling*. The idea is to sample the optimal delay function at cost values $\{1, 1+\bar{\varepsilon}, (1+\bar{\varepsilon})^2, \cdots\}$. For each cost $c, (1+\bar{\varepsilon})^i \le c \le (1+\bar{\varepsilon})^{i+1}$ and for each $i \ge 0$, the value of the approximate delay function at cost c is equal to the optimal delay function at cost $(1+\bar{\varepsilon})^i$, *i.e.*,

$$D_{(v_i,v_j)}(c) = D^{opt}_{(v_i,v_j)}(c') \text{ , where } c' = \max\{(1+\bar{\varepsilon})^t \mid (1+\bar{\varepsilon})^t \le c, t = 1, 2, \cdots\}.$$

For example, consider the optimal delay function $D^{opt}_{(v_i,v_j)}(c)$ depicted in Fig. 3(a). The function is sampled at £ve cost values $\{1, 1+\bar{\varepsilon}, \cdots (1+\bar{\varepsilon})^4\}$. The resulting approximate delay function $D_{(v_i,v_j)}(c)$, depicted in Fig. 3(b), is a piecewise-constant function whose segments correspond to the values $\{d_1, \cdots, d_4\}$ of the optimal function at sampled costs. In our approximation schemes, we store a delay function $D_{(v_i,v_j)}(c)$ by keeping the values of cost and delay for each segment of $D_{(v_i,v_j)}(c)$.

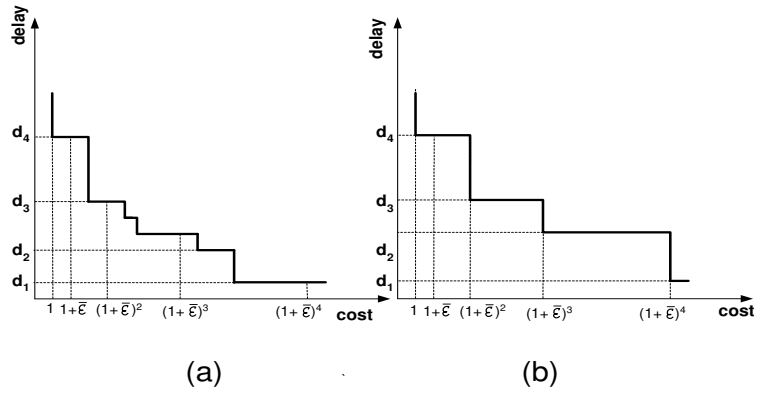

Fig. 3. (a) Optimum delay function (b) Approximate delay function.

3) Layers

We proceed to describe our approach in more detail. Consider a unicast path $\mathcal{P} = \{v_0, \cdots, v_n\}$, which is referred to as a *layer*-0 path. Recall that our assumption is that $n = 2^K$. We split $\mathcal{P}$ into two *layer*-1 subpaths $\mathcal{P}_{(v_0,v_b)}$ and $\mathcal{P}_{(v_b,v_n)}$, where $b = n/2$. Then, for each value $k, k = 1, 2, \cdots, K-1$, each layer-k subpath $\mathcal{P}_{(v_i,v_j)}$ is split into two layer-$(k+1)$ subpaths $\mathcal{P}_{(v_i,v_b)}$ and $\mathcal{P}_{(v_b,v_j)}$, where $b = (i+j)/2$. Note that layer-K subpaths comprise of just a single link. Clearly, the number of subpaths of a layer k is $\mathcal{O}(2^k)$.

The goal our scheme is to compute, for each layer k, $0 \leq k \leq K$, the delay functions of layer-k subpaths. We begin by computing the delay functions of subpaths of layer K. Since these subpaths comprise of just a single link, their delay functions can be obtained by applying logarithmic sampling on the inverted cost functions for corresponding links. Then, for each $k, 1 \leq k \leq K-1$, we compute the delay functions of layer-k subpaths by *merging* previously computed delay functions for subpaths of layer-$(k-1)$. The merging procedure is discussed in detail in Section III-A.4.

The computation of a delay function introduces some error at each layer, which accumulates as we proceed to lower layers. The error depends on the approximation parameter $\bar{\varepsilon}$ used in the logarithmic sampling process. The key idea of our scheme is to use different approximation parameters ε_k for different layers. The values ε_k are chosen in a way that minimizes the computational complexity of the algorithm, while insuring that the total accumulated error does not exceed a desired approximation ratio ε. The assignment of ε_k is discussed in detail in Section III-A.5.

4) Procedure MERGE

The merging procedure receives, as input, the delay functions $D_{(v_i,v_b)}(c)$ and $D_{(v_b,v_j)}(c)$ of two layer-$k+1$ subpaths, $\mathcal{P}_{(v_i,v_b)}$ and $\mathcal{P}_{(v_b,v_j)}$, and an upper bound U. The procedure computes the values of delay function $D_{(v_i,v_j)}(c)$ of layer-k subpath $\mathcal{P}_{(v_i,v_j)}$ for each $1 \leq c \leq U$.

The merging procedure employs logarithmic sampling for an approximation parameter ε_k. Its goal is to compute for each c, $1 \leq c \leq U$, the partition (c_1, c_2) of a budget c between the subpaths, which minimizes the delay of the subpath $\mathcal{P}_{(v_i,v_j)}$ under budget constraint c.

A straightforward solution would be to examine all possible partitions (c_1, c_2) of the budget c. Since the choice of c_1 determines c_2, it is sufficient to consider each $c_1 \leq c$. Moreover, since the delay function $D_{(v_i,v_b)}(c)$ is obtained by logarithmic sampling at costs $\{1, 1+\varepsilon_{k+1}, (1+\varepsilon_{k+1})^2, \cdots\}$, only these costs should be considered.

The merging can be performed more efficiently by the following procedure. We divide the set $S = \{(c_1, c_2) \mid c_1 + c_2 \leq c\}$ of feasible partitions into two subsets S_1 and S_2. The subset S_1 includes the partitions for which $c_1 \leq c/2$ and the subset S_2 includes the partitions for which $c_1 > c/2$, *i.e.*, $S_1 = \{(c_1, c_2) \mid 1 \leq c_1 \leq c/2, c/2 < c_2 \leq c\}$ and $S_2 = \{(c_1, c_2) \mid c/2 < c_1 \leq c, 1 \leq c_2 \leq c/2\}$. Then we identify, for each subset, the partition that minimizes the delay of the subpath $\mathcal{P}_{(v_i,v_j)}$. For the subset S_1, we note that it is sufficient to examine partitions for which values of c_2 correspond to costs $\{(1+\varepsilon_{k+1})^t \mid c/2 < (1+\varepsilon_{k+1})^t \leq c\}$. Thus, and since $c/2 < c_2 \leq c$, we need to consider only $\mathcal{O}(\log_{(1+\varepsilon_{k+1})} 2) = \mathcal{O}(1/\varepsilon_{k+1})$ partitions. Similarly, for the second subset, it is sufficient to consider only $\mathcal{O}(1/\varepsilon_{k+1})$ values of $c_1 \in \{(1+\varepsilon_{k+1})^t \mid c/2 < (1+\varepsilon_{k+1})^t \leq c\}$. We conclude that the optimal partition of the budget c requires $\mathcal{O}(1/\varepsilon_{k+1})$ time. Since we need to find a partition for $\mathcal{O}(\log U/\varepsilon_k)$ budget values, the total complexity incurred is $\mathcal{O}(\log U/\varepsilon_k^2)$. This procedure is referred to as Procedure MERGE and its formal specification is presented in Fig. 4. For clarity of presentation, Procedure MERGE identifies only the delay function $D_{(v_i,v_j)}(c)$ of $\mathcal{P}_{(v_i,v_j)}$. The corresponding partitions can be identified by a mild and straightforward modification of the procedure, with no penalty in terms of computational complexity.

Lemma 1: Given are layer-$(k+1)$ subpaths $\mathcal{P}_{(v_i,v_b)}$ and $\mathcal{P}_{(v_b,v_j)}$ with corresponding $\bar{\varepsilon}$-approximate delay functions $D_{(v_i,v_b)}(c)$ and $D_{(v_b,v_j)}(c)$. Then, the execution of Procedure MERGE yields an $\tilde{\varepsilon}$-approximate delay function $D_{(v_i,v_j)}(c)$ for the subpath $\mathcal{P}_{(v_i,v_j)}$, where $\tilde{\varepsilon} = (1+\varepsilon_k)(1+\bar{\varepsilon}) - 1$.

Proof: See Appendix. ■

Lemma 2: The computational complexity of Procedure MERGE is $\mathcal{O}(\frac{1}{\varepsilon_k^2} \log U)$.

Proof: See Appendix. ■

5) Computing the Delay Function for $\mathcal{P}$

We proceed to present Procedure UNICAST, which computes the delay function of a unicast path $\mathcal{P}$. Procedure UNICAST receives, as input, a layer-k subpath $\mathcal{P}_{(v_i,v_j)}$ of $\mathcal{P}$ and an upper bound U. The procedure computes the values of delay function $D_{(v_i,v_j)}(c)$ of $\mathcal{P}_{(v_i,v_j)}$ for each $1 \leq c \leq U$ in the following recursive manner. If $\mathcal{P}_{(v_i,v_j)}$ is a layer-K path, *i.e.*, $\mathcal{P}_{(v_i,v_j)}$ comprises of a single link l, then the corresponding delay function is obtained by logarithmic sampling (see Section III-A.2) of the inverse of the link cost function $c_l(d)$. Else, for layer-k paths, $1 \leq k \leq K-1$, we first recursively compute the delay functions for subpaths $\mathcal{P}_{(v_i,v_b)}$ and $\mathcal{P}_{(v_b,v_j)}$ of $\mathcal{P}_{(v_i,v_j)}$, where $b = (i+j)/2$; the delay function of the path $\mathcal{P}_{(v_i,v_b)}$ is then computed by merging the the delay functions for $\mathcal{P}_{(v_i,v_b)}$ and $\mathcal{P}_{(v_b,v_j)}$.

Procedure MERGE *($D_{(v_i,v_b)}(c), D_{(v_b,v_j)}(c), \varepsilon_k, \varepsilon_{k+1}, U$):*

parameters

$D_{(v_i,v_b)}(c)$- the delay function for subpath $\mathcal{P}'_{(v_i,v_b)}$

$D_{(v_b,v_j)}(c)$- the delay function for subpath $\mathcal{P}'_{(v_b,v_j)}$

ε_k- the approximation parameter for layer-k

ε_{k+1} - the approximation parameter for layer-$(k+1)$

U- the upper bound on the cost of a partition.

1 $c = 1$
2 **while** $c \leq U$ **do**
3 $\quad D_{(v_i,v_j)}(c) \leftarrow \infty$
4 $\quad c'_2 \leftarrow \max\{(1+\varepsilon_k)^t \leq c/2\}$
5 $\quad c_2 \leftarrow c'_2$
6 $\quad$ **while** $c_2 \leq c$ **do**
7 $\quad\quad D_{(v_i,v_j)}(c) \leftarrow \min\left\{D_{(v_i,v_j)}(c), D_{(v_i,v_b)}(c-c_2) + D_{(v_b,v_j)}(c_2)\right\}$
8 $\quad\quad c_2 \leftarrow c_2 \cdot (1+\varepsilon_{k+1})$
9 $\quad c'_1 \leftarrow \max\{(1+\varepsilon_k)^t \leq c/2\}$
10 $\quad c_1 \leftarrow c'_1$
11 $\quad$ **while** $c_1 \leq c$ **do**
12 $\quad\quad D_{(v_i,v_j)}(c) \leftarrow \min\left\{D_{(v_i,v_j)}(c), D_{(v_i,v_b)}(c_1) + D_{(v_b,v_j)}(c-c_1)\right\}$
13 $\quad\quad c_1 \leftarrow c_1 \cdot (1+\varepsilon_{k+1})$
14 $\quad c \leftarrow (1+\varepsilon_k) \cdot c$
15 **return** $D_{(v_i,v_j)}(c)$

Fig. 4. Procedure MERGE

As mentioned, the computation of a delay function introduces some error at each layer k, which accumulates as we proceed to lower layers. The error at a layer k depends on the approximation parameter ε_k used for this layer. The accumulated error at layer-k (*i.e.*, accumulated along the layers $k, k+1, \cdots, K$) is denoted by $\varepsilon^{(k)}$.

The major consideration in choosing the approximation parameters ε_k for each layer $1 \leq k \leq K$ is to minimize the computational complexity of the scheme. In addition, the values ε_k must be chosen such that the total accumulated error for the path $\mathcal{P}$ does not exceed a desired approximation ratio ε. From Lemma 1 it follows that the accumulated error for at layer 0 is

$$\varepsilon^{(0)} = \prod_{k=1}^{K}(1+\varepsilon_k) - 1 \leq 2\sum_{k=1}^{K}\varepsilon_k,$$

where the last inequality follows from the fact that, for $0 \leq x \leq 1$, it holds that $\ln(1+x) \leq x \leq \ln(1+2x)$. Note that a layer k adds ε_k to the accumulated error.

By Lemma 2, the computational complexity of invoking Procedure MERGE for a layer k subpath is $\mathcal{O}(\frac{1}{\varepsilon_k^2}\log U)$. As there are 2^k subpaths at that layer, the time needed for processing all layer-k paths is $\mathcal{O}(\frac{1}{\varepsilon_k^2}2^k \log U)$. Therefore, the total computational complexity of the algorithm is $\sum_{k=1}^{K}\frac{1}{\varepsilon_k^2}2^k \log U$. Thus, in order to £nd the optimal assignment of approximation parameters $\{\varepsilon_k\}$, we need to solve the following optimization problem:

$$\begin{aligned} & \min \textstyle\sum_{k=1}^{K} \frac{2^k}{\varepsilon_k^2}. \\ \text{subject to :} \quad & \\ & 2\textstyle\sum_{k=1}^{K} \varepsilon_k \leq \varepsilon. \end{aligned} \tag{1}$$

Noting that the objective function in (1) is a sum of convex functions, it is easy to verify that the optimal assignment of $\{\varepsilon_k\}$ is:

$$\varepsilon_k = \frac{\varepsilon}{8\sqrt[3]{2^{K-k}}} \text{ for } k = 1, \cdots, K. \tag{2}$$

With this assignment, the total running time required for all invocations of Procedure MERGE is $\mathcal{O}(\frac{n}{\varepsilon^2} \log U)$.

The detailed description of Procedure UNICAST appears in Fig. 5. As was the case with Procedure MERGE, the partitions that correspond to the delay function $D_{(v_i,v_j)}(c)$ of $\mathcal{P}_{(v_i,v_j)}$ can be identified by a mild and straightforward modification of Procedure UNICAST, with no penalty in terms of computational complexity.

Procedure UNICAST *($\mathcal{P}_{(v_i,v_j)}, k, \{c_l\}_{l\in\mathcal{P}_{(v_i,v_j)}}, \varepsilon, U$):*

parameters

$\mathcal{P}_{(v_i,v_j)}$- a subpath of $\mathcal{P}$ of layer k;
$\{c_l\}_{l\in\mathcal{P}_{(v_i,v_j)}}$- the links' cost functions;
ε - approximation ratio;
U- the upper bound on the cost of an optimal partition.

1 $\varepsilon_k \leftarrow \frac{\varepsilon}{8\sqrt[3]{2^{K-k}}}$
2 **if** $k = K$ **then**
3 **for** each $c \in \left\{(1+\varepsilon_k)^t \mid (1+\varepsilon_k)^t \leq U, t = 1, 2, \cdots\right\}$ **do**
4 $D_{(v_i,v_j)}(c) \leftarrow \min\left\{d \mid c_{(v_i,v_j)}(d) \leq c\right\}$
5 **return** $D_{(v_i,v_j)}(c)$
6 $\varepsilon_{k+1} \leftarrow \varepsilon_k \cdot \sqrt[3]{2}$
7 $b = \frac{i+j}{2}$
8 $D_{(v_i,v_b)}(c) \leftarrow$UNICAST$(\mathcal{P}_{(v_i,v_b)}, k+1, \{c_l\}_{l\in\mathcal{P}_{(v_i,v_b)}}, \varepsilon, U)$
9 $D_{(v_b,v_j)}(c) \leftarrow$UNICAST$(\mathcal{P}_{(v_b,v_j)}, k+1, \{c_l\}_{l\in\mathcal{P}_{(v_b,v_j)}}, \varepsilon, U)$
10 $D_{(v_i,v_j)}(c) \leftarrow$MERGE$(D_{(v_i,v_b)}(c), D_{(v_b,v_j)}(c), \varepsilon_k, \varepsilon_{k+1}, (1+\varepsilon)U)$
11 **return** $D_{(v_i,v_j)}(c)$

Fig. 5. Procedure UNICAST

Theorem 1: Procedure UNICAST identifies, in $\mathcal{O}(\frac{1}{\varepsilon^2} n \cdot \log U)$ time, an ε-approximate delay function $D_{(v_0,v_n)}(c)$ for a path $\mathcal{P}$.

Proof: See Appendix. ■

B. On-demand computation: Problem OPQ

In this section we present an algorithm for computing a suitable QoS partition upon an incoming request. The algorithm comprises of the following steps. First, we obtain sufficiently tight lower and upper bounds, L and U, on the cost of an optimal partition. Then, we use these bounds in order to perform *linear scaling* on link cost functions. The purpose of linear scaling is to "scale down" all the costs, *i.e.*, reduce all the

costs by dividing them by some £xed parameter. The resulting graph has smaller costs, which reduces the overall running time. Next, we £nd a suitable partition by using Procedure UNICAST. The obtained solution is then rounded back to the original costs, *i.e.*, prior to the linear scaling, incurring a small error.

1) Upper and Lower Bounds

Following the algorithmic technique presented in [7], we start with trivial bounds, and proceed to iteratively improve them, until they become suf£ciently tight.

Let $\{d_l\}_{l\in\mathcal{P}}$ be a partition that satis£es the delay constraint D. We observe that, for each $l \in \mathcal{P}$, it holds that $d_l^{\min} \leq d_l \leq D$, where $d_l^{\min}$ is the minimum delay constraint supported by a link l, *i.e.*, $d_l^{\min} = \min\{d \mid c_l(d) \neq \infty\}$. This implies that $L = \sum_{l\in\mathcal{P}} c_l(D)$ and $U = \sum_{l\in\mathcal{P}} c_l(d_l^{\min})$ constitute obvious lower and upper bounds on the cost of an optimal partition. Since $L \geq n$ and $U \leq n \cdot C^{\max}$, we have $U/L \leq C^{\max}$.

Reduction of the ratio U/L is achieved by performing a binary search on the interval (L, U) on a logarithmic scale. First, we compute for each $l \in \mathcal{P}$ the minimum value of the QoS requirement d_l that can be supported by allocating a budget $c = \sqrt{\frac{U\cdot L}{n}}$ to l. Then, we check whether the resulting partition $\{d_l\}_{l\in\mathcal{P}}$ satis£es the delay constraint, *i.e.*, $\sum_{l\in\mathcal{P}} d_l \leq D$. Clearly, if $\{d_l\}_{l\in\mathcal{P}}$ satis£es the delay constraint, then $c\cdot n$ is an upper bound on the cost of the optimal solution, hence we set $U \leftarrow c\cdot n$. Otherwise, the cost of the optimal solution is at least c, hence we set $L \leftarrow c$. The process continues as long as $U > 2 \cdot n \cdot L$.

We denote by β the ratio of the initial upper and lower bounds, and by β_i the ratio of upper and lower bounds after iteration i. Let L and U be the values of the lower and upper bounds, respectively, at the beginning of the iteration i. During an iteration, either the lower or the upper bound is updated. In the former case, we have

$$\beta_i = \frac{U}{c} = \sqrt{\frac{nU}{L}} = \sqrt{n\beta_{i-1}},$$

while in the latter case, the value of β_i is

$$\beta_i = \frac{nc}{L} = \sqrt{\frac{nU}{L}} = \sqrt{n\beta_{i-1}}.$$

In both cases, we have $\beta_i = \sqrt{n\beta_{i-1}}$. Thus, the value of β_i at iteration i is bounded by

$$\beta_i \leq n\beta^{\frac{1}{2^i}}.$$

Note that, at iteration $i = \lceil \log\log\beta \rceil$, we have $\beta_i \leq 2 \cdot n$. We conclude that just $\mathcal{O}(\log\log\beta) = \mathcal{O}(\log\log C^{\max})$ iterations are necessary in order to achieve $U/L \leq 2\cdot n$. Since each iteration requires $\mathcal{O}(n)$ time, the computational complexity of £nding lower and upper bounds, L and U, for which $U/L \leq 2 \cdot n$, is $\mathcal{O}(n \log\log C^{\max})$.

2) The algorithm

Having computed suitable bounds U and L, *i.e.*, bounds for which $U/L \leq 2 \cdot n$, we apply a scaling and rounding procedure on the link cost functions. To that end, a new cost function is de£ned for each link l,

as follows:

$$c_l^*(d_l) = \left\lfloor \frac{2n \cdot c_l(d_l)}{\varepsilon L} \right\rfloor . \tag{3}$$

With modi£ed link costs, the new cost c^* of a partition with original cost c is bounded by

$$\frac{c \cdot n}{(\varepsilon/2) \cdot L} - n \leq c^* \leq \frac{c \cdot n}{(\varepsilon/2) \cdot L}. \tag{4}$$

Thus, and since $U \leq 2n \cdot L$, the upper bound on the solution with respect to the new link cost functions is $U^* = \frac{4n^2}{\varepsilon}$. Finally, the problem is solved by applying Procedure UNICAST to a path with the scaled cost functions $c_l^*(d_l)$. The procedure is invoked with the upper bound U^* and the approximation parameter $\varepsilon/2$. The procedure returns an $\frac{\varepsilon}{2}$-approximate solution with respect to the new link costs. Theorem 2 below implies that the cost of this solution, under the original cost functions, is at most $(1+\varepsilon)$ times larger than that of the optimal solution. Algorithm OPQ, described in Fig. 6, summarizes the above discussion.

Algorithm OPQ *($\mathcal{P}, \{c_l\}_{l\in\mathcal{P}}, \varepsilon, D$):*

parameters

$\mathcal{P} = \{v_0, ..., v_n\}$- a QoS path;
$\{c_l\}_{l\in\mathcal{P}}$- the links' cost functions;
ε - approximation ratio;
D- delay constraint.

1 $L \leftarrow \sum_{l\in\mathcal{P}} c_l(D)$
2 $U \leftarrow \sum_{l\in\mathcal{P}} c_l(d_l^{\min})$, where $d_l^{\min} = \min\{d \mid c_l(d) \neq \infty\}$
3 **while** $\frac{U}{L} > 2 \cdot n$ **do**
4 $\quad c \leftarrow \sqrt{\frac{L \cdot U}{n}}$
5 $\quad$ **for** each $l \in \mathcal{P}$ **do**
6 $\quad\quad d_l = \min\{d \mid c_l(d) \leq c\}$
7 $\quad$ **if** $\sum_{l\in\mathcal{P}} d_l \leq D$ **then**
8 $\quad\quad U \leftarrow c \cdot n$
9 $\quad$ **else**
10 $\quad\quad L \leftarrow c$
11 $D_{(v_0,v_n)}(c) \leftarrow$UNICAST$\left(\mathcal{P}, 0, \left\{\left\lfloor \frac{c_l(d_l)\cdot n}{(\varepsilon/2)\cdot L}\right\rfloor\right\}, \varepsilon/2, \frac{4n^2}{\varepsilon}\right)$
12 $\hat{c} \leftarrow \min\left\{c \mid D_{(v_0,v_n)}(c) \leq D\right\})$
13 **return** partition that corresponds to $\hat{c}$

Fig. 6. Algorithm OPQ

Theorem 2: Algorithm OPQ provides, in $\mathcal{O}(\frac{1}{\varepsilon^2} n \cdot \log\frac{n}{\varepsilon} + n \cdot \log\log C^{\max})$ time, an ε-approximate solution to Problem OPQ, *i.e.*: given a connection request with delay constraint D, Algorithm OPQ identi£es a suitable QoS partition $\{d_l\}_{l\in\mathcal{P}}$, whose cost is at most $(1+\varepsilon)$ times higher than that of the optimal partition.

Proof: See Appendix. ■

C. Precomputation scheme: Problem POPQ

Precomputation is performed by means of a two-phase procedure, referred to as a *precomputation scheme*. The purpose of the £rst phase is to compute a delay function for the path $\mathcal{P}$, which summarizes a set

of suitable partitions, for each delay constraint. The second phase merely selects one of the solutions precomputed in the £rst phase.

1) First phase

The £rst phase is implemented as follows. We begin by invoking Procedure UNICAST with approximation parameter $\varepsilon/3$, which computes an $\varepsilon/3$-approximate delay function $D'_{(v_0,v_n)}(c)$ and the corresponding partitions. Then, we use $D'_{(v_0,v_n)}(c)$ in order to compute a delay function $D_{(v_0,v_n)}(c)$ whose storage requirements are signi£cantly smaller.

More speci£cally, the delay function $D'_{(v_0,v_n)}(c)$, obtained through Procedure UNICAST, is a piecewise-constant function whose segments correspond to costs $c \in \{1,(1+\varepsilon_0),\cdots,nC^{\max}\}$, where $\varepsilon_0 = \frac{\varepsilon}{8\sqrt[3]{n}}$ (according to Equation (2)). Thus, we need $\mathcal{O}(\frac{\sqrt[3]{n}}{\varepsilon}\log(nC^{\max}))$ space in order to store $D'_{(v_0,v_n)}(c)$. The storage requirement can be signi£cantly reduced by logarithmic sampling. Speci£cally, we compute new delay function $D_{(v_0,v_n)}(c)$ out of $D'_{(v_0,v_n)}(c)$ by logarithmic sampling at costs $\{1,(1+\varepsilon/3),(1+\varepsilon/3)^2,\cdots,nC^{\max}\}$. By Lemma 3 below, $D_{(v_0,v_n)}(c)$ is an ε-approximate delay function for $\mathcal{P}$. The detailed description of the £rst part of the precomputation scheme, implemented by Algorithm POPQ, appears in Fig. 7.

Algorithm POPQ *($\mathcal{P}, \{c_l\}_{l\in\mathcal{P}}, \varepsilon$):*

parameters

$\mathcal{P} = \{v_0, ..., v_n\}$- a QoS path;
$\{c_l\}_{l\in\mathcal{P}}$- the links' cost functions;
ε - approximation ratio;

1 $D'_{(v_0,v_n)}(c) \leftarrow$UNICAST$(\mathcal{P}, 0, \{c_l\}_{l\in\mathcal{P}}, \varepsilon/3, n\cdot C^{\max})$
2 **for** each $c \in \left\{(1+\varepsilon/3)^t \le U \mid t = 1,2,\cdots\right\}$ **do**
3 $\quad D_{(v_0,v_n)}(c) \leftarrow D'_{(v_0,v_n)}(c)$
4 **return** $D_{(v_0,v_n)}(c)$

Fig. 7. Algorithm POPQ

Lemma 3: Algorithm POPQ computes, in $\mathcal{O}(\frac{1}{\varepsilon^2}n\log(nC^{\max}))$ time, an ε-approximate delay function $D_{(v_0,v_n)}(c)$ for $\mathcal{P}$.

Proof: See Appendix. ■

2) Second phase

Upon a request with some QoS requirement D, the optimal partition is promptly identi£ed by examining the output of Algorithm POPQ. Speci£cally, we identify, through binary search, the cost c of a suitable partition, $c = \min\{c' = (1+\varepsilon/3)^t \mid D_{(v_0,v_n)}(c') \le D\}$, and return the corresponding partition. Since the total number of precomputed partitions is $\mathcal{O}(\frac{1}{\varepsilon}\log(nC^{\max}))$, the computational complexity of this procedure is $\mathcal{O}(\log\log(C^{\max})+\log(1/\varepsilon)+n)$. The term n in the complexity expression is due to the need to describe the partition.

D. Discussion

We proceed to compare the performance of our algorithms with that of its alternatives.

We begin with the on-demand setting. In [7] and [2], the problem of partitioning of QoS constraints was considered, in a broader context of QoS routing with cost-dependent functions. The proposed algorithms, when applied to Problem OPQ, yield computational complexities of $\mathcal{O}(n \log\log C^{\max} + \frac{n^2 \log(n/\varepsilon)}{\varepsilon^2})$ and $\mathcal{O}(\min(D, \frac{\log U}{\varepsilon}, \frac{n}{\varepsilon} D) \frac{1}{\varepsilon} n^2 \log\log C^{\max})$, respectively. The dominant terms of these expression are $\mathcal{O}(\frac{n^2 \log(n/\varepsilon)}{\varepsilon^2})$ and $\mathcal{O}(\frac{n^2 \log C^{\max}}{\varepsilon^2})$, respectively, while the dominant term in our solution is $\mathcal{O}(\frac{n \log(n/\varepsilon)}{\varepsilon^2})$. We thus conclude that the computational complexity of our algorithm is signi£cantly ($\Omega(n)$) less dependent on the topology size than that of [7] and [2], which renders it more scalable for large topologies. This improvement has been achieved by exploiting the topological structure of unicast paths.

Next, we note that our algorithm can be applied also in the practically important case of discrete cost functions, *i.e.*, step functions whose range is a discrete set of values. Such functions have been the focus of [10], and an $\mathcal{O}(rn^3 \log \frac{r}{\varepsilon})$ algorithm was presented there, where $r = \sum_{l \in \mathcal{P}} r_l$ and r_l is the number of different delay values supported by link l. We conclude that, even if $r = \mathcal{O}(n)$ (*i.e.*, each link supports a £xed number of delays), we achieve a major ($\Omega(n^3)$) reduction in terms of dependency on the topology size.

We described a precomputation scheme for Problem OPQ that provides ε-optimal solutions within a computational complexity of $\mathcal{O}(\frac{1}{\varepsilon^2} n \cdot \log(n \cdot C^{\max}))$ for the £rst phase and $\mathcal{O}(\log\log(n \cdot C^{\max}) + \log\frac{1}{\varepsilon} + n)$ for the second phase. Compared with an on-demand scheme, the precomputation scheme signi£cantly reduces the time required to £nd a suitable partition. Indeed, with precomputation, the computational complexity of £nding a suitable partition is dominated by the time necessary to describe a partition ($\mathcal{O}(n)$), *i.e.*, it is very close to the lower bound.

We note that a precomputation scheme can be trivially constructed out of any existing approximation algorithm for Problem OPQ (*e.g.*, [2], [7]), by just sequentially executing them for a certain range of delay values. Nonetheless, as it easy to verify, the computational complexity of such simplistic solutions is signi£cantly higher than that of our solution.

IV. QoS partition for Multicast

In this section we deal with the problem of QoS partition on multicast trees. Since we employ ideas that are quite similar to those of the unicast setting, we shall restrict ourselves to a brief discussion.

We begin by introducing the required de£nitions and terminology. A *directed tree* is a subgraph $\mathcal{T}$ of $G(V, E)$ having a unique node s such that every node is reached from s by a unique path; node s is referred to as the *source*. A *multicast connection* uses a tree $\mathcal{T}$ to interconnect the source s and the members of a multicast group $M = \{t_1, t_2, ...\}$. A path between source s and a terminal t_i on links that belong to the tree $\mathcal{T}$ is denoted by $\mathcal{P}_i$. Given a multicast tree $\mathcal{T}$, our goal is to (ef£ciently) allocate the delay on each link $l \in \mathcal{T}$ such that the end-to-end delay is satis£ed for each member t_i of the multicast group. A QoS partition on a multicast tree $\mathcal{T}$ is a set of link delay requirements $\{d_l\}_{l \in \mathcal{T}}$, which satis£es, for each

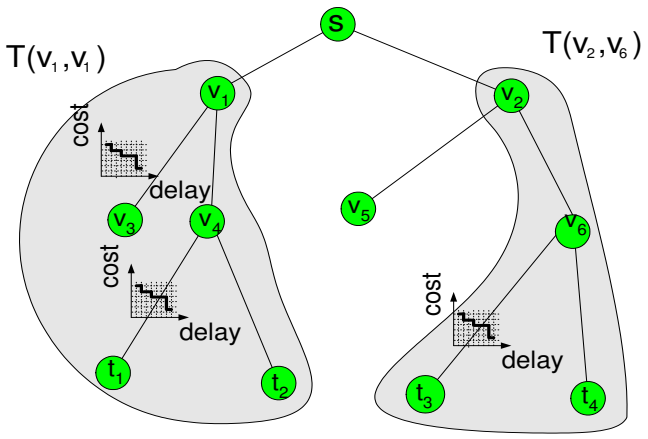

Fig. 8. An example of a multicast tree, $n = 11$ and $H = 4$.

$t_i \in M$, the end-to-end delay requirement D, *i.e.*, $\sum_{l \in \mathcal{P}_i} d_l \leq D$ for each $t_i \in M$. Each link is associated with a cost function $c_l(d)$, which speci£es the cost of supporting a delay requirement d. The cost of a QoS partition $\{d_l\}_{l \in \mathcal{T}}$ is the sum of the local costs, *i.e.*, $\sum_{l \in \mathcal{T}} c_l(d_l)$. We assume that all parameters (cost and delays) are (positive) integers.

The optimal QoS partition for a multicast tree is then de£ned as follows.

Problem MOPQ: (Muticast Optimal Partition of QoS) Given a tree $\mathcal{T}$ and a delay requirement D, £nd a QoS partition $\{d_l\}_{l \in \mathcal{T}}$ such that $\sum_{l \in \mathcal{P}_i} d_l \leq D$ for each $t_i \in M$ and $\sum_{l \in \mathcal{T}} c_l(d_l)$ is minimized.
We de£ne also the related precomputation problem.

Problem PMOPQ: (Precomputation of MOPQ) Given a tree $\mathcal{T}$, £nd, for each delay requirement D, a QoS partition $\{d_l\}_{l \in \mathcal{T}}$ such that $\sum_{l \in \mathcal{P}_i} d_l \leq D$ for each $t_i \in M$ and $\sum_{l \in \mathcal{T}} c_l(d_l)$ is minimized.

For clarity of exposition, we use the following notation. The number of nodes and the depth of the multicast tree are denoted by n and H, respectively. The number of children of a node v_i are denoted by m_i. The subtree originating from the node $v_i \in \mathcal{T}$ is denoted by $\mathcal{T}_{(v_i,v_i)}$. A *branch* $\mathcal{T}_{(v_i,v_j)}$ of the subtree $\mathcal{T}_{(v_i,v_i)}$ is a subtree originating from v_i, which includes the link (v_i, v_j) outgoing from i and all descendants of v_j. For example, Fig. 8 shows a multicast tree $\mathcal{T}$, a subtree $\mathcal{T}_{(v_1,v_1)}$ and a branch $\mathcal{T}_{(v_2,v_6)}$.

We employ the following "divide-and-conquer scheme". A multicast tree is recursively split into a number of disjoint subtrees. We compute the set of delay guaranties supported by each subtree at different costs. These delay guaranties and the corresponding partitions are summarized by means of *delay functions*, de£ned below. We then obtain a solution to the original problem, *i.e.*, a partition of delay constraint D on tree $\mathcal{T}$, by recursively combining the delay functions obtained for the subtrees of $\mathcal{T}$.

More speci£cally, consider a multicast tree $\mathcal{T}$, which is referred to as a *layer*-0 tree. We split $\mathcal{T}$ into a number of *layer*-1 subtrees $\left\{\mathcal{T}_{(v_i,v_i)}\right\}$, for each child node v_i of s. Then, for each value $k, k = 1, 2, ..., H$, each layer-k subtree $\mathcal{T}_{(v_j,v_j)}$ is split into a number of layer-$(k+1)$ subtrees, for each child node of v_j. Layer-H subtrees comprise of just a single node. For example, in the tree $\mathcal{T}$ depicted in Fig. 8, subtrees

$\mathcal{T}_{(v_1,v_1)}$ and $\mathcal{T}_{(v_2,v_2)}$ are layer-1 subtrees, while $\mathcal{T}_{(v_3,v_3)}$ and $\mathcal{T}_{(v_4,v_4)}$ are layer-2 subtrees. We denote by n_k the number of subtrees of layer-k. Clearly, $\sum_{k=1}^{H} n_k = n$.

We introduce the following *subtree delay functions*, which summarize the delay guaranties offered by a subtree at different costs.

De£nition 3: The *optimal delay function* $D^{opt}_{(v_i,v_i)}(c)$ of the subtree $\mathcal{T}_{(v_i,v_i)}$ of $\mathcal{T}$ is de£ned as the minimum delay requirement supported by $\mathcal{T}_{(v_i,v_i)}$ at cost c, *i.e.*,

$$D^{opt}_{(v_i,v_i)}(c) = \min\left\{ D \;\middle|\; \exists \{d_l\}_{l\in\mathcal{T}_{(v_i,v_i)}} \text{ such that } \max_{t_k\in\mathcal{T}_{(v_i,v_i)}} \sum_{l\in\mathcal{P}_k} d_l \le D \text{ and } \sum_{l\in\mathcal{T}_{(v_i,v_i)}} c_l(d_l) \le c \right\}.$$

Optimal delay functions for branches $\mathcal{T}_{(v_i,v_j)}$ of $\mathcal{T}_{(v_i,v_i)}$ are de£ned similarly.

In addition we de£ne, for each link $(v_i, v_j) \in \mathcal{T}$, the optimal delay function $\tilde{D}^{opt}_{(v_i,v_j)}(c)$ in a way that resembles the optimal delay function of a subpath (see De£nition 1, Section III-A.1). Speci£cally, the optimal delay function $\tilde{D}^{opt}_{(v_i,v_j)}(c)$ of link (v_i, v_j) is de£ned as the minimum delay requirement d supported by link (v_i, v_j) at cost c, *i.e.*, $\tilde{D}^{opt}_{(v_i,v_j)}(c) = \min\{d \mid c_{(v_i,v_j)}(d) \le c\}$.

De£nition 4: An *$\bar{\varepsilon}$-approximate delay function* $D_{(v_i,v_j)}(c)$ of a subtree $\mathcal{T}_{(v_i,v_j)}$ of $\mathcal{T}$ is a function that satis£es, for each $c \ge 0$, $D_{(v_i,v_j)}(c(1+\bar{\varepsilon})) \le D^{opt}_{(v_i,v_j)}(c)$.

We de£ne $\bar{\varepsilon}$-approximate delay functions for branches and links in a similar manner. When no ambiguity exists, $\bar{\varepsilon}$-approximate delay functions will be referred to as just delay functions. Delay functions are constructed by using the logarithmic sampling approach.

A. Computation of delay functions

In this section we present Procedure MULTICAST which identi£es the delay functions $D_{(v_i,v_i)}(c)$ and the corresponding partitions for each subtree of each layer. The delay functions are computed in a bottom-up manner, £rst for layer-H subtrees, then for layer-$(H-1)$ subtrees, *etc.*, up to layer 0. Note that each layer-H subtree $\mathcal{T}_{(v_i,v_i)}$ comprises of a single terminal node v_i. For each terminal node v_i the delay function $D_{(v_i,v_i)}(c)$ of subtree $\mathcal{T}_{(v_i,v_i)}$ is set to 0 for all c.

More speci£cally, we compute the delay function $D_{(v_i,v_i)}(c)$ of subtree $\mathcal{T}_{(v_i,v_i)}$ by performing the following steps:

1) If v_i is a terminal, then $D_{(v_i,v_i)}(c)$ is is set to 0 for all c. Otherwise, for each child node v_j of v_i:
   a) Recursively compute the delay function $D_{(v_j,v_j)}(c)$ of layer-$(k+1)$ subtree $\mathcal{T}_{(v_j,v_j)}$;
   b) Compute the delay function $\tilde{D}_{(v_i,v_j)}(c)$ of the link (v_i, v_j) by performing logarithmic sampling on the link cost function $c_{(v_i,v_j)}(d)$ of (v_i, v_j);
   c) Compute the delay function $D_{(v_i,v_j)}(c)$ of the branch $\mathcal{T}_{(v_i,v_j)}$ by merging the delay functions $\tilde{D}_{(v_i,v_j)}(c)$ and $D_{(v_j,v_j)}(c)$;
2) Compute the delay function $D_{(v_i,v_i)}(c)$ of the subtree $\mathcal{T}_{(v_i,v_i)}$ by merging the delay functions of all branches $\{\mathcal{T}_{(v_i,v_j)}\}$ of $\mathcal{T}_{(v_i,v_i)}$.

As it is the case for unicast, the critical part is to choose, for each layer k, the approximation parameter ε_k used for computing delay functions. The assignment of ε_k is discussed in detail in Section IV-A.2.

1) Merging procedures

As discussed above, in order to compute the delay function $D_{(s,s)}(c)$ we need to de£ne two merging procedures, which we proceed to describe in some more detail.

The £rst procedure receives, as an input, the delay functions $\tilde{D}_{(v_i,v_j)}(c)$ and $D_{(v_j,v_j)}(c)$ of link (v_i,v_j) and subtree $\mathcal{T}_{(v_j,v_j)}$, respectively, and an upper bound U. The function computes the values of the delay function $D_{(v_i,v_j)}(c)$ of the branch $\mathcal{T}_{(v_i,v_j)}$ for each $1 \leq c \leq U$. The goal of this procedure is to compute for each c, $1 \leq c \leq U$, the partition (c_1,c_2) of a budget c between the link (v_i,v_j) and subtree $\mathcal{T}_{(v_j,v_j)}$, which minimizes the delay supported by the branch $\mathcal{T}_{(v_i,v_j)}$ under budget constraint c. The procedure is similar to the merger of the delay functions of two subpaths, as discussed in Section III-A.4. Accordingly, we use Procedure MERGE that appears on Fig. 4.

The purpose of the second procedure, referred to as MIN-MAX-MERGE, is to calculate the delay function $D_{(v_i,v_i)}(c)$ of the subtree $\mathcal{T}_{(v_i,v_i)}$ out of the delay functions $D_{(v_i,v_j)}(c)$ of its branches. In order to compute $D_{(v_i,v_i)}(c)$, we £nd, for each cost value c, $1 \leq c \leq U$, the minimum delay that can be supported by the subtree $\mathcal{T}_{(v_i,v_i)}$ subject to budget c. For this purpose we need to £nd the local budget c_j for each branch $\mathcal{T}_{(v_i,v_j)}$ in such a way that the maximum delay between v_i and a terminal $t_i \in \mathcal{T}_{(v_i,v_i)}$ is minimized, *i.e.*,

$$D_{(v_i,v_i)}(c) = \min\left\{ D \;\middle|\; \exists \left\{c_j\right\}_{(v_i,v_j)\in\mathcal{T}} \text{ such that } \max_{(v_i,v_j)\in\mathcal{T}} D_{(v_i,v_j)}(c_j) \leq D \text{ and } \sum_{(v_i,v_j)\in\mathcal{T}} c_j \leq c \right\}. \quad (5)$$

Note that the delay function $D_{(v_i,v_j)}(c)$ of each branch $\mathcal{T}_{(v_i,v_j)}$ is piecewise-constant. Hence, the function $D_{(v_i,v_j)}(c)$ of the subtree $\mathcal{T}_{(v_i,v_i)}$ is also piecewise-constant and can be computed by identifying its segments. We begin with segments that correspond to lower costs, then proceed with segments that correspond to higher costs. Since the cost of supporting a delay requirement by each branch $\mathcal{T}_{(v_i,v_j)}$ is at least 1, then the minimum cost for the supporting a delay requirement by subtree $\mathcal{T}_{(v_i,v_i)}$ is m_i, hence we set:

$$D_{(v_i,v_i)}(m_i) = \max_{(v_i,v_j)\in\mathcal{T}} \left\{D_{(v_i,v_j)}(1)\right\}.$$

Thus, the £rst segment corresponds to cost m_i and delay $D_{(v_i,v_i)}(m_i)$. Suppose that we have identi£ed the segment of $D_{(v_i,v_i)}(c)$ that corresponds to delay constraint $\hat{d}$, cost $\hat{c}$ of supporting $\hat{d}$ and the corresponding partition $\{\hat{c}_j\}_{(v_i,v_j)\in\mathcal{T}}$, *i.e.*, $\hat{d} = D_{(v_i,v_i)}(\hat{c})$ and $\sum_{(v_i,v_j)\in\mathcal{T}} \hat{c}_j = \hat{c}$. We show how to identify the next segment of $D_{(v_i,v_i)}(c)$ that corresponds to delay $\hat{d}'$ and cost $\hat{c}' = \sum_{(v_i,v_j)\in\mathcal{T}} \hat{c}'_j$. Note that $\hat{c}'$ is the minimum cost that must be paid in order to support a delay constraint lower than $\hat{d}$, *i.e.*, $\hat{c}' = \min\left\{c \mid D_{(v_i,v_i)}(c) < \hat{d}\right\}$ and $\hat{d}' = D_{(v_i,v_i)}(\hat{c}')$.

We observe that, by Equation 5, there exists a link $(v_i,v_j) \in \mathcal{T}$ for which it holds that $D_{(v_i,v_j)}(\hat{c}_j) = D_{(v_i,v_i)}(\hat{c})$. We denote by $S = \left\{j \mid D_{(v_i,v_j)}(\hat{c}_j) = D_{(v_i,v_i)}(\hat{c})\right\}$. Since $\hat{d}' < \hat{d}$, for each $j \in S$, the delay supported by branch $\mathcal{T}_{(v_i,v_j)}$ at cost $\hat{c}'_j$ must be lower than $\hat{d}$. Thus, we set $\hat{c}'_j$ be the minimum cost

of supporting a delay lower than $\hat{d}$ by branch $\mathcal{T}_{(v_i,v_j)}$. For each $j \notin S$ we set $\hat{c}'_j = \hat{c}_j$. As we prove in Lemma 4 below, the next segment of $D_{(v_i,v_j)}(c)$ corresponds to cost $\hat{c}' = \sum_{(v_i,v_j)\in\mathcal{T}} \hat{c}'_j$ and delay $\hat{d}' = \max_{(v_i,v_j)\in\mathcal{T}} \left\{ D_{(v_i,v_j)}(\hat{c}'_j) \right\}$.

Having computed $D_{(v_i,v_i)}(c)$, we perform an additional procedure in order to reduce the number of segments in $D_{(v_i,v_i)}(c)$. Specifically, we perform logarithmic sampling at costs $1, 1+\varepsilon_k, (1+\varepsilon_k)^2, \cdots$. This yields a number of segments that is bounded by $\mathcal{O}(\frac{1}{\varepsilon_k}\log U)$. The formal description of Procedure MIN-MAX-MERGE appears on Fig. 9.

Procedure MIN-MAX-MERGE *($\mathcal{T}_{(v_i,v_i)}, \varepsilon_k, U$):*

parameters
$\mathcal{T}_{(v_i,v_i)}$ the subtree of $\mathcal{T}$
ε_k - the approximation parameter for the output functions
U- the upper bound on the cost of a partition.

1 **for** each $(v_i, v_j) \in \mathcal{T}$ **do**
2 $\quad c_j \leftarrow 1$
3 $\quad d_j \leftarrow D_{(v_i,v_j)}(c_j)$
4 $\quad c \leftarrow \sum_{(v_i,v_j)\in\mathcal{T}} c_j$
5 $\quad d \leftarrow \max_{(v_i,v_j)\in\mathcal{T}} d_j$
6 **while** $c \leq U$ **do**
7 $\quad D_{(v_i,v_i)}(c) \leftarrow d$
8 $\quad S \leftarrow \{j \mid D_{(v_i,v_j)}(c_j) = d\}$
9 $\quad$ **for** each $j \in S$ **do**
10 $\quad\quad c_j \leftarrow \min\{c \mid D_{(v_i,v_j)}(c) \leq d_j\}$
11 $\quad\quad d_j \leftarrow D_{(v_i,v_j)}(c_j)$
12 $\quad c \leftarrow \sum_{(v_i,v_j)\in\mathcal{T}} c_j$
13 $\quad d \leftarrow \max_{(v_i,v_j)\in\mathcal{T}} d_j$
14 **for** each $c \in \{(1+\varepsilon_k)^t \leq U \mid t = 1, 2, \cdots\}$ **do**
15 $\quad D'_{(v_i,v_i)}(c) \leftarrow D_{(v_i,v_i)}(c)$
16 **return** $D'_{(v_i,v_i)}(c)$

Fig. 9. Procedure MIN-MAX-MERGE

Lemma 4: Given are a layer-k subtree $\mathcal{T}_{(v_i,v_i)}$, layer-$(k+1)$ branches $\mathcal{T}_{(v_i,v_j)}$ of $\mathcal{T}_{(v_i,v_i)}$ with corresponding $\bar{\varepsilon}$-approximate delay functions $D_{(v_j,v_j)}(c)$. Then, Procedure MIN-MAX-MERGE computes, in $\mathcal{O}(\frac{1}{\varepsilon_k} m_i \log m_i \log U)$ time, an $\tilde{\varepsilon}$-approximate delay function $D_{(v_i,v_i)}(c)$ for the subtree $\mathcal{T}_{(v_i,v_i)}$, where $\tilde{\varepsilon} = (1+\varepsilon_k)(1+\bar{\varepsilon}) - 1$.

Proof: See Appendix. ■

2) Tuning the parameters

The computation of the delay functions introduces some error at each layer k, which accumulates as we proceed to lower layers. The error at a layer k depends on the approximation parameter ε_k used for this layer. The accumulated error at layer-k is denoted by $\varepsilon^{(k)}$.

By Lemmas 1 and 4, if the accumulated error at layer-$k+1$ is $\varepsilon^{(k+1)}$, then the accumulated error at layer k is $\varepsilon^{(k)} = (1+\varepsilon^{(k+1)})(1+\varepsilon_k) - 1$. Thus, the accumulated error at layer 0 is

$$\varepsilon^{(0)} = \prod_{k=0}^{H}(1+\varepsilon_k)^2 - 1 \leq 6\sum_{k=0}^{H}\varepsilon_k.$$

The time needed for processing all subtrees is dominated by the time required for the execution of Procedure MERGE for all subtrees of all layers. By Lemma 2, the computational complexity of the invocation of Procedure MERGE for a branch of layer-k subtree is $\mathcal{O}(\frac{\log U}{\varepsilon_k^2})$. As the number of branches of layer-k subtrees is n_{k+1}, the total running time required for invoking Procedure MERGE for layer k subtrees is $\mathcal{O}(\frac{n_{k+1}\log U}{\varepsilon_k^2})$. The total computational complexity of the algorithm is $\mathcal{O}(\sum_{k=0}^{H-1}\frac{n_{k+1}\log U}{\varepsilon_k^2})$. Thus, in order to £nd the optimal assignment of approximation parameters $\{\varepsilon_k\}$, we need to solve the following optimization problem:

$$\begin{aligned} & \min \textstyle\sum_{k=0}^{H-1}\frac{n_{k+1}}{\varepsilon_k^2}. \\ \text{subject to}: & \\ & 6\textstyle\sum_{k=0}^{H}\varepsilon_k \leq \varepsilon \end{aligned} \tag{6}$$

It is easy to verify that the optimal assingment of $\{\varepsilon_k\}$ is

$$\varepsilon_k = \begin{cases} \frac{\varepsilon\sqrt[3]{n_{k+1}}}{12\sum_{k=1}^{H}\sqrt[3]{n_k}} & \text{for } k = 0,\cdots,H-1. \\ \frac{\varepsilon}{2} & k = H \end{cases} \tag{7}$$

With this assignment of $\{\varepsilon_k\}$, total running time required for all invocations of Procedure MERGE is $\mathcal{O}(\frac{nH^2}{\varepsilon^2}\log U)$.

Note 1: If $\mathcal{T}$ is a balanced tree, then the optimal assignment is $\varepsilon_k = \frac{\varepsilon}{24}\sqrt[3]{\frac{2^k}{n}}$ for $k = 1,\cdots,H-1$.

The formal speci£cation of Procedure MULTICAST appears in Fig. 10. The partitions that correspond to the returned delay function $D_{(v_i,v_j)}(c)$, can be identi£ed by a mild and straightforward modi£cation of the Procedure MULTICAST, with no penalty in terms of computational complexity.

Theorem 3: Procedure MULTICAST identi£es, in $\mathcal{O}(\frac{1}{\varepsilon^2}nH\log U)$ time, a ε-approximate delay function $D_{(s,s)}(c)$ for a tree $\mathcal{T}$.

Proof: See Appendix. ■

Note 2: If the tree $\mathcal{T}$ is balanced, using the assignment of ε_k as speci£ed in Note 1 yields a computational complexity of $\mathcal{O}(\frac{1}{\varepsilon^2}n\log U)$.

B. On-demand computation: Problem MOPQ

We proceed to discuss the on-demand setting, in which a suitable QoS partition is computed upon an incoming request with some delay constraint D. $L = \sum_{l\in\mathcal{P}} c_l(D)$ and $U = \sum_{l\in\mathcal{P}} c_l(d_l^{\min})$ constitute obvious lower and upper bounds on the cost of an optimal partition, where $d_l^{\min} = \min\{d \mid c_l(d) \neq \infty\}$. Clearly, $U/L \leq C^{\max}$.

Reduction of the ratio U/L is achieved by performing binary search on the interval $[L,U]$ in a logarithmic scale. First, we compute for each $l \in \mathcal{T}$, the minimum value of the QoS requirement d_l that can be supported by allocating a budget $c = \sqrt{\frac{U\cdot L}{n}}$ to l. Then, we check whether the resulting partition $\{d_l\}_{l\in\mathcal{T}}$ satis£es the delay constraint, *i.e.*, for each terminal t_i it holds that $\sum_{l\in\mathcal{P}_i} d_l \leq D$. Clearly, if $\{d_l\}_{l\in\mathcal{T}}$ satis£es the

Procedure MULTICAST *($\mathcal{T}_{(v_i,v_i)}, k, \{c_e\}, \varepsilon, U$):*

parameters

$\mathcal{T}_{(v_i,v_i)}$- subtree of layer k

$\{c_e\}$- the links' cost functions;

ε - approximation ratio;

U- the upper bound on the cost of an optimal partition.

1 $\varepsilon_k \leftarrow \frac{\varepsilon \sqrt[3]{n_{k+1}}}{12 \sum_{k=1}^{H} \sqrt[3]{n_k}}$

2 **if** v_i is a terminal **then**

3 $\quad D_{(v_i,v_i)}(c) \leftarrow 0$ for all c

4 $\quad$ **return** $D_{(v_i,v_i)}(c)$

5 **if** $k = H-1$ **then** $\varepsilon_{k+1} \leftarrow \frac{\varepsilon}{2}$ **else** $\varepsilon_{k+1} \leftarrow \frac{\varepsilon \sqrt[3]{n_{k+2}}}{12 \sum_{k=1}^{H} \sqrt[3]{n_k}}$

6 **for** each $(v_i, v_j) \in \mathcal{T}$ **do**

7 $\quad D_{(v_j,v_j)}(c) \leftarrow$MULTICAST$(\mathcal{T}_{(v_j,v_j)}, k+1, \{c_e\}, \varepsilon, U)$

8 $\quad$ **for** each $c \in \left\{(1+\varepsilon_{k+1})^t \mid (1+\varepsilon_{k+1})^t \leq U, t = 1, 2, \cdots\right\}$ **do**

9 $\quad\quad \tilde{D}_{(v_i,v_j)}(c) \leftarrow \min\left\{d \mid c_{(v_i,v_j)}(d) \leq c\right\}$

10 $\quad D_{(v_i,v_j)}(c) \leftarrow$MERGE$(\tilde{D}_{(v_i,v_j)}(c), D_{(v_j,v_j)}(c), \varepsilon_k, \varepsilon_{k+1}, (1+\varepsilon)U)$

11 $D_{(v_i,v_i)}(c) \leftarrow$MIN-MAX-MERGE$(\mathcal{T}_{(v_i,v_i)}, \varepsilon_k, (1+\varepsilon)U)$

12 **return** $D_{(v_i,v_i)}(c)$

Fig. 10. Procedure MULTICAST

delay constraint, then $c \cdot n$ is an upper bound on the cost of the optimal solution, hence we set $U = c \cdot n$. Otherwise, the cost of the optimal solution is at least c, hence we set $L = c$. The process continues as long as $U > 2nL$. After $\mathcal{O}(\log\log C^{\max})$ iterations we obtain lower and upper bounds, U and L for which holds $\frac{U}{L} \leq 2n$. The procedure requires $\mathcal{O}(n \log\log C^{\max})$ time.

Having computed suitable bounds U and L, for which $U/L \leq 2n$, we apply a scaling and rounding procedure on the link cost functions. To that end, a new cost function is de£ned for each link l, according to Equation 3. Finally, the problem is solved by applying Procedure MULTICAST to a path with the scaled cost functions $c_l^*(d_l)$. The procedure is invoked with the upper bound $U^* = \frac{2n^2}{\varepsilon}$ and the approximation parameter $\frac{\varepsilon}{2}$. The procedure returns an $\frac{\varepsilon}{2}$-approximate solution with respect to the new link costs. As it is the case for the unicast, the cost of this solution under the original cost functions is at most $(1+\varepsilon)$ times larger than that of the optimal solution.

The above results are summarized by the following theorem.

Theorem 4: Given a connection request with delay constraint D, a suitable QoS partition $\{d_l\}_{l \in \mathcal{T}}$, whose cost is at most $(1+\varepsilon)$ times larger than that of the optimal partition, can be identi£ed in $\mathcal{O}(\frac{1}{\varepsilon^2} n \cdot H^2 \log\frac{n}{\varepsilon} + + n \log\log C^{\max})$ steps.

Proof: The proof is similar to that of Theorem 2. ■

Note 3: If the tree $\mathcal{T}$ is balanced, an ε-approximate solution can be identi£ed in $\mathcal{O}(\frac{1}{\varepsilon^2} n \log\frac{n}{\varepsilon} + n \log\log C^{\max})$ time.

C. Precomputation scheme: Problem PMOPQ

As explained in Section III-C, precomputation is performed by means of a two-phase procedure. The purpose of the £rst phase is to precompute the optimal partition *a priori*, for each delay constraint supported by the tree $\mathcal{T}$. We start by invoking Procedure MULTICAST with approximation parameter $\varepsilon/3$, which computes an $\varepsilon/3$-approximate delay function $D'_{(s,s)}(c)$ and the corresponding partitions. Then, we compute a new delay function $D_{(s,s)}(c)$ out of $D'_{(s,s)}(c)$ by performing logarithmic sampling at costs $\{1, (1+\varepsilon/3), (1+\varepsilon/3)^2, \cdots, nC^{\max}\}$. The computational complexity of the £rst phase is $\mathcal{O}(\frac{1}{\varepsilon^2} n \cdot H^2 \log(n \cdot C^{\max}))$. For the special case of balanced trees the computational complexity is $\mathcal{O}(\frac{1}{\varepsilon^2} n \cdot \log(n \cdot C^{\max}))$.

Then, at the second phase, and upon a request with some QoS requirement D, the suitable partition is promptly identi£ed by examining the delay function $D_{(s,s)}(c)$. Speci£cally, we identify, through binary search, the cost c of a suitable partition, $c = \min\{c' = (1+\varepsilon/3)^t \mid D_{(s,s)}(c') \leq D\}$, and return the corresponding partition. This procedure requires $\mathcal{O}(\log\log(nC^{\max}) + \log\frac{1}{\varepsilon} + n)$ time.

D. Discussion

We proceed to compare the performance of our algorithms with that of its alternatives.

The on-demand setting was considered in [7], where an ε-approximate solution to Problem MOPQ was presented. That algorithm yields a computational complexity of $\mathcal{O}(n \log D \log\log\beta + n^2(\log D + n)\log\log H + \frac{n^2}{\varepsilon}(\log D + \frac{n}{\varepsilon}))$. The dominant term of this expression is $\mathcal{O}(\frac{n^3}{\varepsilon^2})$, while the dominant term of our solution is $\mathcal{O}((1/\varepsilon^2) n \cdot H^2 \log\frac{n}{\varepsilon})$. It follows that, for most practical settings *i.e.*, when H is lower than n, the computational complexity of our algorithm is signi£cantly $(\Omega(\frac{n^2}{H^2 \log(n/\varepsilon)}))$ less dependent on the topology size than that of [7]. Moreover, we note that the depth H of a typical multicast tree is $\mathcal{O}(\log n)$, in which case our algorithm is $\Omega(\frac{n^2}{\log^2 n \log(n/\varepsilon)})$ times faster. Furthermore, in the special case of balanced trees, the computational complexity of our solution is just $\mathcal{O}(\frac{1}{\varepsilon^2} n \log\frac{n}{\varepsilon}) + n\log\log C^{\max})$, which is $\Omega(\frac{n^2}{\log(n/\varepsilon)})$ times faster than that of [7].

We described a precomputation scheme for Problem MOPQ that provides ε-optimal solutions within a computational complexity of $\mathcal{O}(\frac{1}{\varepsilon^2} n \cdot H^2 \log(nC^{\max}))$ for the £rst phase and $\mathcal{O}(\log\log(nC^{\max}) + \log\frac{1}{\varepsilon} + n)$ for the second phase. This precomputation scheme promptly provides a suitable partition upon an incoming request. The computational complexity of our scheme is signi£cantly lower than that of simplistic adaptations of existing approximation algorithms.

V. CONCLUSION

A fundamental problem in the support of QoS in networks is how to allocate resources along the connection's topology such that the required QoS can be guaranteed at minimum cost. This immediately translates into the optimization problem that has been the focus of this study, namely, how to optimally partition the end-to-end QoS requirement into local requirements. This problem poses major challenges in terms of algorithmic design, and has been the subject of several recent studies. These studies provided signi£cant insight into the essence of the problem and its potential solutions. However, the solutions

that have been proposed either relied on restrictive assumptions (such as convexity), or else proposed approximation schemes whose complexity considerably depended on the network size. Therefore, a scalable approach, which would be adequate for large scale networks, was called for. Such an approach should be less dependent on the size of the connection's topology, and, ultimately, provide a fast answer to the partition problem upon each incoming connection request.

Accordingly, in this study we considered the scalability perspective, taking two independent approaches. First, we proposed a novel algorithmic technique, which exploits the speci£c structure of the actual topologies on which connections are established, *i.e.* paths or trees. This technique resulted in a signi£cant improvement in terms of computational complexity, in particular dependence on the size of the topology. Indeed, for the "on-demand" setting, our approach typically offers almost-linear solutions, both for unicast and for multicast, in terms of dependence on topology size. These results *per se* constitute a signi£cant improvement upon previous solutions. Second, we devised a *precomputation scheme*. This scheme is based on the observation that, typically, network elements have the resources to perform much of the computation in advance. Hence, it enables to obtain fast solutions immediately upon each incoming connection request; in particular, at that time (*i.e.*, at the "second phase"), the computational complexity depends *only linearly* on the size of the topology, be it a unicast path or a multicast tree.

Several enhancements and extensions of this study are possible. For example, our layering approach allows to easily distribute the computational effort among network nodes. Indeed, at each layer, each component (subpath or subtree) is processed independently, hence the processing can be performed concurrently, at different nodes.

More generally, the schemes presented in this study can serve to tackle the scalability issue in other important networking problems. In particular, another fundamental problem in the context of QoS provision is that of QoS routing, *i.e.*, the proper selection of the connection's topology. The key observation there is that large-scale networks typically bear a hierarchical layering structure, which provides the grounds for an ef£cient application of our divide-and-conquer approach.

REFERENCES

[1] S. Blake. An Architecture for Differentiated Services. - RFC No. 2475. Internet Engineering Task Force, December 1998.

[2] L. Zhang F. Ergun, R. Sinha. Qos Routing with Performance-Dependent Costs. In *Proceedings of IEEE INFOCOM'00*, Tel-Aviv, Israel, March-April 2000.

[3] V. Firoiu and T. Towsley. Call Admission and Resource Reservation for Multicast Sessions. In *Proceedings of IEEE INFOCOM'96*, San-Francisco, CA, April 1996.

[4] R. Guérin and A. Orda. Computing Shortest Paths for Any Number of Hops. *IEEE/ACM Transactions on Networking*, 10(5):613–620, October 2002.

[5] M. Kodialam and S. Low. Resource Allocation in a Multicast Tree. In *Proceedings of IEEE INFOCOM'99*, New York, NY, March 1999.

[6] D. H. Lorenz and A. Orda. Optimal Partition of QoS Requirements on Unicast Paths and Multicast Trees. *IEEE/ACM Transactions on Networking*, 10(1):102–114, February 2002.

[7] D.H. Lorenz, A. Orda, D. Raz, and Y. Shavitt. Ef£cient QoS Partition and Routing of Unicast and Multicast. In *Proceedings of IEEE/IFIP IWQoS*, Pittsburgh, PA, June 2000.

[8] A. Orda and A. Sprintson. QoS Routing: the Precomputation Perspective. In *Proceedings of IEEE INFOCOM'00*, Tel-Aviv, Israel, March-April 2000.

[9] Private Network-Network Interface Specification v1.0 (PNNI). ATM Forum Technical Committee, March 1996.

[10] D. Raz and Y. Shavitt. Optimal Partition of QoS Requirements with Discrete Cost Functions. *IEEE Journal on Selected Areas in Communications*, 18(12):2593–2602, December 2000.

APPENDIX

Lemma 1: Given are layer-$(k+1)$ subpaths $\mathcal{P}_{(v_i,v_b)}$ and $\mathcal{P}_{(v_b,v_j)}$ with corresponding $\bar{\varepsilon}$-approximate delay functions $D_{(v_i,v_b)}(c)$ and $D_{(v_b,v_j)}(c)$. Then, the execution of Procedure MERGE yields an $\tilde{\varepsilon}$-approximate delay function $D_{(v_i,v_j)}(c)$ for the subpath $\mathcal{P}_{(v_i,v_j)}$, where $\tilde{\varepsilon} = (1+\varepsilon_k)(1+\bar{\varepsilon}) - 1$.

Proof: Let $\hat{c}$ be an arbitrary cost. We denote by d^{opt} the minimum delay supported by $\mathcal{P}_{(v_i,v_j)}$ at cost $\hat{c}$, *i.e.*, $d^{opt} = D^{opt}_{(v_i,v_j)}(\hat{c})$ and by $\{d_l^{opt}\}_{l\in\mathcal{P}_{(v_i,v_j)}}$ the optimal partition of delay d^{opt}. In addition, we denote by $\hat{d}_1 = \sum_{l\in\mathcal{P}_{(v_i,v_b)}} d_l^{opt}$, $\hat{d}_2 = \sum_{l\in\mathcal{P}_{(v_b,v_j)}} d_l^{opt}$, $\hat{c}_1 = \min\{c \mid D_{(v_i,v_b)}(c) \le \hat{d}_1\}$ and $\hat{c}_2 = \min\{c \mid D_{(v_b,v_j)}(c) \le \hat{d}_2\}$. The condition of the lemma implies that $\hat{c}_1 \le (1+\bar{\varepsilon})\sum_{l\in\mathcal{P}_{(v_i,v_b)}} c_l(d_l^{opt})$ and $\hat{c}_2 \le (1+\bar{\varepsilon})\sum_{l\in\mathcal{P}_{(v_b,v_j)}} c_l(d_l^{opt})$. This, in turn, implies that $\hat{c}_1 + \hat{c}_2 \le (1+\bar{\varepsilon})\hat{c}$.

We prove that $D_{(v_i,v_j)}((1+\tilde{\varepsilon})\hat{c}) \le d^{opt}$. Consider the invocation of the loop that begins at line 2 for $c = \min\{(1+\varepsilon_k)^t \mid (1+\varepsilon_k)^t \ge \hat{c}_1 + \hat{c}_2\}$. Clearly, $\hat{c}_1 + \hat{c}_2 \le c \le (1+\varepsilon_k)(\hat{c}_1 + \hat{c}_2)$.

There are two possible cases.

1) $\hat{c}_1 \le \hat{c}_2$. Then after the iteration of the loop that begins on line 6 for $c_2 = \max\{\hat{c}_2, c_2'\}$, it holds that $D_{(v_i,v_j)}(c) \le D_{(v_i,v_b)}(\hat{c}_1) + D_{(v_b,v_j)}(\hat{c}_2) \le \hat{d}_1 + \hat{d}_2 = d^{opt}$.
2) $\hat{c}_2 < \hat{c}_1$. Then after the iteration of the loop that begins on line 11 for $c_1 = \max\{\hat{c}_1, c_1'\}$, it holds that $D_{(v_i,v_j)}(c) \le D_{(v_i,v_b)}(\hat{c}_1) + D_{(v_b,v_j)}(\hat{c}_2) \le \hat{d}_1 + \hat{d}_2 = d^{opt}$.

In both cases we showed that there exists $c \le (1+\varepsilon_k)(\hat{c}_1 + \hat{c}_2)$ for which $D_{(v_i,v_j)}(c) \le d^{opt}$. Since $\hat{c}_1 + \hat{c}_2 \le (1+\bar{\varepsilon})\hat{c}$, we have $c \le (1+\varepsilon_k)(1+\bar{\varepsilon})\hat{c} = (1+\tilde{\varepsilon})\hat{c}$, which in turn implies that $D_{(v_i,v_j)}((1+\tilde{\varepsilon})\hat{c}) \le d^{opt}$. Since $\hat{c}$ is arbitrary, the lemma follows. ■

Lemma 2: The computational complexity of Procedure MERGE is $\mathcal{O}(\frac{1}{\varepsilon_k^2}\log U)$.

Proof: First, let us count the number of iterations t of the procedure's main loop (*i.e.*, the loop beginning on line 2). Clearly, $(1+\varepsilon_k)^t \le U$ hence $t \le \frac{\ln U}{\ln(1+\varepsilon_k)}$. Since for all $0 \le x \le 1$ it holds that $x \le 2\ln(1+x)$, we have $t = \mathcal{O}\left(\frac{\log U}{\varepsilon_k}\right)$

The number of iterations t' of each of the loops beginning on lines 6 and 11 is $\mathcal{O}\left(\frac{\log 2(1+\varepsilon_k)}{\log(1+\varepsilon_{k+1})}\right)$. Since $\log(1+\varepsilon_k) = \mathcal{O}(1)$ for $\varepsilon_k \le 1$ and $1/\varepsilon_{k+1} = \mathcal{O}(1/\varepsilon_k)$, it holds that $t' = \mathcal{O}(\frac{1}{\varepsilon_k})$. The time required to execute both loops is also $\mathcal{O}(\frac{1}{\varepsilon_k})$, since a single iteration requires $\mathcal{O}(1)$ time.

We conclude that the computational complexity of the procedure is indeed $\mathcal{O}(\frac{1}{\varepsilon_k^2}\log U)$. ■

Theorem 1: Procedure UNICAST identifies, in $\mathcal{O}(\frac{1}{\varepsilon^2}n \cdot \log U)$ time, an ε-approximate delay function $D_{(v_0,v_n)}(c)$ for a path $\mathcal{P}$.

Proof: We begin by proving that Procedure UNICAST identifies, for each subpath of layer k, $1 \le k \le K$, an $\varepsilon^{(k)}$-approximate delay function $D_{(v_i,v_j)}(c)$, where $\varepsilon^{(k)} = \prod_{t=k}^{k}(1+\varepsilon_t) - 1$.

The proof is by induction on the layer number k. Consider a layer-K subpath $\mathcal{P}_{(v_i,v_{i+1})}$. It is immediate that lines 2 and 3 compute an ε_K-approximate delay function $D_{(v_i,v_{i+1})}(c)$. Assume inductively that the assertion holds for subpaths of layer-$(k+1)$, and consider a layer-k subpath $\mathcal{P}_{(v_i,v_j)}$. Since the assertion holds for the subpaths $\mathcal{P}_{(v_i,v_b)}$ and $\mathcal{P}_{(v_b,v_j)}$, $b = (j+i)/2$, the condition of Lemma 1 is satisfied for $\varepsilon^{(k+1)} = \prod_{t=k+1}^{K}(1+\varepsilon_t) - 1$. Lemma 1 implies, in turn, that the algorithm identifies an $\varepsilon^{(k)}$-approximate delay function $D_{(v_i,v_j)}(c)$, for $\varepsilon^{(k)} = \prod_{t=k}^{K}(1+\varepsilon_t) - 1$.

Since $\prod_{t=0}^{K}(1+\varepsilon_t) - 1 \leq 2\sum_{t=0}^{K}\varepsilon_t \leq \varepsilon$, the assertion implies that Procedure UNICAST ε-approximate delay function $D_{(v_0,v_n)}(c)$ for path $\mathcal{P}$.

We proceed to analyze the computational complexity of Procedure UNICAST. The procedure is applied recursively for each subpath of $\mathcal{P}$ of each layer k. The total time required for processing layer-K paths is $\mathcal{O}(\frac{1}{\varepsilon_K} n \cdot \log U)$. For $0 \leq k \leq K-1$, the time needed for processing a layer-k subpath is determined by the running time of Procedure MERGE. By Lemma 2, invocation of of Procedure MERGE for layer-k subpath requires $\mathcal{O}(\frac{\log U}{\varepsilon_k^2}) = \mathcal{O}(\frac{\sqrt[3]{2^{2(K-k)}}\log U}{\varepsilon^2})$ time. Since there are 2^k subpaths of layer-k, processing layer-k requires $\mathcal{O}(\frac{\sqrt[3]{2^{2K+k}}\log U}{\varepsilon^2})$ time. The total time needed for processing each subpath of each layer is:

$$\mathcal{O}\left(\frac{2^{\frac{2K}{3}}\log U}{\varepsilon^2}\sum_{k=0}^{K-1}2^{\frac{k}{3}}\right) = \mathcal{O}\left(\tfrac{1}{\varepsilon^2} n \cdot \log U\right).$$

We conclude the computational complexity of the algorithm is $\mathcal{O}(\frac{1}{\varepsilon^2} n \cdot \log U)$ and the theorem follows. ■

Theorem 2: Algorithm OPQ provides, in $\mathcal{O}(\frac{1}{\varepsilon^2} n \cdot \log \frac{n}{\varepsilon} + n \cdot \log\log C^{\max})$ time, an ε-approximate solution to Problem OPQ, *i.e.*: given a connection request with delay constraint D, Algorithm OPQ identifies a suitable QoS partition $\{d_l\}_{l\in\mathcal{P}}$, whose cost is at most $(1+\varepsilon)$ times higher than that of the optimal partition.

Proof: In lines 1 and 2 we compute obvious lower and upper, L and U bounds on the cost of the optimal solution. As discussed in Section III-B.2, the bounds remain valid during execution of the loop that begins at line 3 and after the execution of this loop it holds that $U/L \leq 2n$.

We denote by c_{opt} and c^*_{opt} the cost of the optimal solution under the original and scaled cost functions, respectively. Equation 4 implies that $c^*_{opt} \leq \frac{c_{opt}\cdot n}{(\varepsilon/2)\cdot L}$. By Theorem 1, Procedure UNICAST yields an $\varepsilon/2$-approximate delay function $D_{(v_0,v_n)}(c)$. Thus, after execution of line 12 it holds that $\hat{c} \leq (1+\varepsilon/2)c^*_{opt}$. Since $c^*_{opt} \leq \frac{c_{opt}\cdot n}{(\varepsilon/2)\cdot L}$, we have $\hat{c} \leq (1+\varepsilon/2)\frac{c_{opt}\cdot n}{(\varepsilon/2)\cdot L}$. Let $\{d_l\}_{l\in\mathcal{P}}$ be the partition that corresponds to cost $\hat{c}$. From the left part of Equation 4 it follows that the cost c of $\{d_l\}_{l\in\mathcal{P}}$ is at most $c \leq \frac{\varepsilon\hat{c}\cdot L}{2n} + \frac{\varepsilon L}{2} \leq (1+\varepsilon/2)\cdot c_{opt} + (\varepsilon/2)\cdot L \leq (1+\varepsilon)\cdot c_{opt}$. We conclude that the algorithm returns a feasible partition whose cost is at most $(1+\varepsilon)$ times more than the optimum.

We proceed to analyze the computational complexity of Algorithm OPQ. Lines 1 and 2 of the algorithm require $\mathcal{O}(n)$ time. Each iteration of the loop of line 3 requires also $\mathcal{O}(n)$ time. Since the total number of iterations is $\mathcal{O}(\log\log C^{\max})$, we conclude that the loop requires $\mathcal{O}(n\cdot\log\log C^{\max})$ time. Theorem 1 implies that the application of Procedure UNICAST for $U = \frac{4n^2}{\varepsilon}$ (line 11) requires $\mathcal{O}(\frac{1}{\varepsilon^2} n \cdot \log\frac{n}{\varepsilon})$ time. Thus, we conclude that the computational complexity of the algorithm is $\mathcal{O}(\frac{1}{\varepsilon^2} n \cdot \log\frac{n}{\varepsilon} + n\cdot\log\log C^{\max})$. ■

Lemma 3: Algorithm POPQ computes, in $\mathcal{O}(\frac{1}{\varepsilon^2} n \log(nC^{\max}))$ time, an ε-approximate delay function $D_{(v_0,v_n)}(c)$ for $\mathcal{P}$.

Proof: By Theorem 1, Procedure UNICAST yields an $\varepsilon/3$-approximate delay function $D'_{(v_0,v_n)}(c)$.

Let $\hat{c}$ be an arbitrary cost. Since $D'_{(v_0,v_n)}(c)$ is an $\varepsilon/3$-approximate delay function, there exists $c' \leq (1+\varepsilon/3)c$ such that $D'_{(v_0,v_n)}(c') \leq D^{opt}_{(v_0,v_n)}(c)$. Furthermore, let $c'' = \min\left\{(1+\varepsilon/3)^t \mid (1+\varepsilon/3)^t > c'\right\}$. Since $c'' \leq (1+\varepsilon/3)c' \leq (1+\varepsilon/3)^2\hat{c}$, it holds that $c'' \leq (1+\varepsilon)\hat{c}$ for $\varepsilon \leq 1$. After execution of the loop that begins at line 2 it holds that $D_{(v_0,v_n)}(c'') \leq D'_{(v_0,v_n)}(c')$. Hence for $c'' \leq (1+\varepsilon)\hat{c}$ it holds that $D_{(v_0,v_n)}(c'') \leq D^{opt}_{(v_0,v_n)}(c)$. Since c is arbitrary, $D_{(v_0,v_n)}(c)$ is an ε-approximate delay function for $\mathcal{P}$.

By Theorem 1, the application of Procedure UNICAST for $U = n \cdot C^{\max}$ (line 1) requires $\mathcal{O}((1/\varepsilon^2)n \cdot \log(nC^{\max}))$ time, which is also the computational complexity of Algorithm POPQ. ■

Lemma 4: Given are a layer-k subtree $\mathcal{T}_{(v_i,v_i)}$, layer-$(k+1)$ branches $\mathcal{T}_{(v_i,v_j)}$ of $\mathcal{T}_{(v_i,v_i)}$ with corresponding $\bar{\varepsilon}$-approximate delay functions $D_{(v_j,v_j)}(c)$. Then, Procedure MIN-MAX-MERGE computes, in $\mathcal{O}(\frac{1}{\varepsilon_k} m_i \log m_i \log U)$ time, an $\tilde{\varepsilon}$-approximate delay function $D_{(v_i,v_i)}(c)$ for the subtree $\mathcal{T}_{(v_i,v_i)}$, where $\tilde{\varepsilon} = (1+\varepsilon_k)(1+\bar{\varepsilon}) - 1$.

Proof: First, we prove the following claim: at each iteration of the loop that begins on line 6 for each $(v_i, v_j) \in \mathcal{T}$ it holds that c_j is the minimum cost of supporting delay requirement d, *i.e.*, $c_j = \min\left\{c \mid D_{(v_i,v_j)}(c) \leq d\right\}$. Clearly, the claim holds at the beginning of iteration 1. Suppose inductively that, the claim holds at the beginning of iteration k, we prove that the claim holds at the end of the iteration. We denote the value of d at the beginning of the iteration by d' and in the end of the iteration by d''. Note that $d'' < d'$ and in the end of the iteration it holds that $D_{(v_i,v_j)}(c_j) \leq d''$ for each $(v_i, v_j) \in \mathcal{T}$. Thus, for each $j \notin S$, since the value of c_j does not change during the iteration, it holds that c_j is a minimum cost of supporting d''. For each $j \in S$, c_j is set to minimum cost of supporting a delay lower than d'. Thus, since $d'' < d'$ and $D_{(v_i,v_j)}(c_j) \leq d''$, it holds that c_j is a minimum cost of supporting d''.

Next, we prove, that for arbitrary cost $\hat{c}$, $1 \leq \hat{c} \leq U$ it holds that $D_{(v_i,v_i)}((1+\bar{\varepsilon})\hat{c}) \leq D^{opt}_{(v_i,v_i)}(\hat{c})$. We denote by $\hat{d}^{opt}$ the minimum delay supported by $\mathcal{T}_{(v_i,v_i)}$ at cost $\hat{c}$, *i.e.*, $\hat{d}^{opt} = D^{opt}_{(v_i,v_i)}(\hat{c})$. In addition, we denote, for each $(v_i, v_j) \in \mathcal{T}$, $\hat{c}^{opt}_j = \min\left\{c \mid D^{opt}_{(v_i,v_j)}(c) \leq \hat{d}^{opt}\right\}$, $\hat{c}_j = \min\left\{c \mid D_{(v_i,v_j)}(c) \leq \hat{d}^{opt}\right\}$, $\hat{d}_j = D_{(v_i,v_j)}(\hat{c}_j)$. Let $\hat{d} = \max_{(v_i,v_j)\in\mathcal{T}} \hat{d}_j$.

The condition of the lemma implies that, for each $(v_i, v_j) \in \mathcal{T}$, it holds that $\hat{c}_j \leq (1+\bar{\varepsilon})\hat{c}^{opt}_j$. Consider the iteration of the loop that begins on line 6 in which $d = \hat{d}$. The claim above implies that for each $(v_i, v_j) \in \mathcal{T}$, it holds that $c_j = \min\left\{c \;\middle|\; D_{(v_i,v_j)}(c) \leq \hat{d}\right\} = \hat{c}_j$. Thus, for each $(v_i, v_j) \in \mathcal{T}$, it holds that $c_j \leq (1+\bar{\varepsilon})\hat{c}^{opt}_j$ and, after execution of line 7 we have $D_{(v_i,v_i)}(c) \leq d^{opt}$, where $c = \sum_{(v_i,v_j)\in\mathcal{T}} c_j \leq (1+\bar{\varepsilon})\hat{c}^{opt}$. We thus proved that $D_{(v_i,v_i)}(c)$ is a $\bar{\varepsilon}$-approximate delay function of $\mathcal{T}_{(v_i,v_i)}$.

In the loop that begins in line 14 we compute the function $D'_{(v_i,v_i)}(c)$ by performing the logarithmic sampling of function $D_{(v_i,v_i)}(c)$ at costs $1, 1+\varepsilon_k, \cdots$. Thus, the resulting function is an $\tilde{\varepsilon}$-approximate delay function, where $\tilde{\varepsilon} = (1+\bar{\varepsilon})(1+\varepsilon_k) - 1$.

We proceed to analyze the computational complexity of Procedure MIN-MAX-MERGE. The loop that begins at line 1 requires $\mathcal{O}(m_i)$ time. At each iteration of the loop that begins at line 6 we examine

a segment of $D_{(v_i,v_j)}(c)$ for some branch $\mathcal{T}_{(v_i,v_j)}$ of $\mathcal{T}_{(v_i,v_i)}$. Since the delay function of the branch has $\mathcal{O}(\frac{\log U}{\varepsilon_k})$ segments, the number of iterations of the loop is $\mathcal{O}(\frac{1}{\varepsilon_k}m_i \log U)$. All lines in the loop, except for lines 11, 11 and 13, can be executed in $\mathcal{O}(1)$ time. The total computational complexity of line 10 is $\mathcal{O}(\frac{1}{\varepsilon_k}m_i \log U)$. If we use a binary tree to keep values of d_j, then the total computational complexity of lines 11 and 13 is $\mathcal{O}(\frac{1}{\varepsilon_k}m_i \log m_i \log U)$. The loop that begins at line 14 requires $\mathcal{O}(\frac{1}{\varepsilon_k}\log U)$ time. We conclude that the total computational complexity of the procedure is $\mathcal{O}(\frac{1}{\varepsilon_k}m_i \log m_i \log U)$. ■

Theorem 3: Procedure MULTICAST identifies, in $\mathcal{O}(\frac{1}{\varepsilon^2}nH \log U)$ time, a ε-approximate delay function $D_{(s,s)}(c)$ for a tree $\mathcal{T}$.

Proof: We begin by proving that Procedure MULTICAST identifies, for each subtree $\mathcal{T}_{(v_i,v_i)}$ of layer-k, $1 \leq k \leq H$, a $\varepsilon^{(k)}$-approximate delay function $D_{(v_i,v_i)}$, where $\varepsilon^{(k)} = \prod_{t=k}^{H}(1+\varepsilon_t)^2 - 1$.

The proof is by induction on layer number k. Consider a subtree $\mathcal{T}_{(v_i,v_i)}$ that corresponds to a terminal v_i. It is immediate that line 3 computes an optimal delay function $D_{(v_i,v_i)}(c)$. Assume inductively that the assertion holds for subtrees of layer $k+1$ and consider a layer-k subtree $\mathcal{T}_{(v_i,v_i)}$. Since the assertion holds for the subtrees $\left\{\mathcal{T}_{(v_j,v_j)}\right\}_{(v_i,v_j)\in\mathcal{T}}$, and since $\tilde{D}_{(v_i,v_j)}$ is a ε_{k+1}-approximate delay function for link (v_i,v_j) the condition of Lemma 1 is satisfied for $\varepsilon^{(k+1)} = \prod_{t=k+1}^{H}(1+\varepsilon_t)^2 - 1$. Lemma 1 implies, in turn, that the algorithm identifies an $\hat{\varepsilon}$-approximate function $D_{(v_i,v_j)}$ for each branch $\mathcal{T}_{(v_i,v_j)}$ of $\mathcal{T}_{(v_i,v_i)}$, where $\hat{\varepsilon} = (1+\varepsilon_k)(1+\varepsilon^{(k+1)}) - 1$. Hence, the condition of Lemma 4 is satisfied for $\bar{\varepsilon} = (1+\varepsilon_k)(1+\varepsilon^{(k+1)}) - 1$. Lemma 4 implies, in turn, that the algorithm identifies an $\varepsilon^{(k)}$-approximate function $D_{(v_i,v_i)}$ for $\mathcal{T}_{(v_i,v_i)}$, where $\varepsilon^{(k)} = (1+\varepsilon_k)^2(1+\varepsilon^{(k+1)}) - 1$ and the assertion follows.

We note that $\prod_{t=0}^{H}(1+\varepsilon_t)^2 = \prod_{t=0}^{H}(1+2\varepsilon_t+\varepsilon_t^2) \leq \prod_{t=0}^{H}(1+3\varepsilon_t)$ and $\prod_{t=0}^{H}(1+3\varepsilon_t) - 1 \leq \sum_{t=0}^{H} 3\varepsilon_t$. After substitution ε_t according to Equation 7 we have $\sum_{t=0}^{H} 3\varepsilon_t \leq \varepsilon$. We conclude that the procedure computes an ε-approximate delay function $D_{(s,s)}(c)$ for tree $\mathcal{T}$.

We proceed to analyze the computational complexity of Procedure MULTICAST. The complexity is dominated by time required to execute Procedure MERGE, which is executed for each subtree of each layer. Since the computational complexity of Procedure MERGE for layer-k subtree is $\mathcal{O}(\frac{1}{\varepsilon_{k-1}^2}\log U)$ and since there are n_k subtrees at layer k, the total time T_1 of required for execution of Procedure MERGE is $\log U \sum_{k=1} H - 1 \frac{1}{\varepsilon_{k-1}^2} n_k$. After substitution ε_t according to Equation 7 we have

$$T_1 \leq \frac{18 \log U}{\varepsilon}\left(\sum_{k=1}^{H}\sqrt[3]{n_k}\right)^2 \sum_{k=1}^{H}\sqrt[3]{n_k}.$$

Since the latter expression is maximized when $n_k = n/H$, we have $T_1 = \mathcal{O}(\frac{1}{\varepsilon^2}nH^2 \log U)$.

Next, we analyze the total time T_2 required for execution of Procedure MIN-MAX-MERGE. The procedure is also executed for each each subtree of each layer. By Lemma 4, the computational complexity of executing the procedure for a subtree $T_{(v_i,v_i)}$ of layer k is $\mathcal{O}(\frac{1}{\varepsilon_k}m_i \log m_i \log U)$, where $m_i \leq n$ is the number of branches of $T_{(v_i,v_i)}$. Thus, $T_2 = \log n \log U \sum_{k=0}^{H-1} \frac{1}{\varepsilon_k} n_{k+1}$. After substitution ε_k according to

Equation 7 we have

$$T_2 = \frac{\log n \log U}{\varepsilon} \sum_{k=1}^{H} \sqrt[3]{n}_k \sum_{k=1}^{H} \sqrt[3]{n_k^2}.$$

Since the latter expression is maximized when $n_k = n/H$, we have $T_2 = \mathcal{O}(\frac{1}{\varepsilon} nH \log n \log U)$.

Finally, in procedure Procedure MULTICAST we perform logarithmic sampling for each link $l \in \mathcal{T}$ (lines 8-9). Let $T_{(v_j,v_j)}$ be a subtree of layer k, $1 \leq k \leq H-1$ and let $\{v_i\}$ be the parent node of v_j. Performing logarithmic sampling for each link (v_i, v_j) requires $\mathcal{O}(\frac{1}{\varepsilon_k} \log U)$ time. Thus, handling all links between a root of a subtree of layer-k and its parent node is $\mathcal{O}(\frac{1}{\varepsilon_k} n_k \log U)$. Thus, we need $T_3 = \mathcal{O}(\log U \sum \sum_{k=1}^{H} \frac{1}{\varepsilon_k} n_k)$ time to process all links in $\mathcal{T}$. After substitution ε_k according to Equation 7, we conclude that the time required to process all links in $\mathcal{T}$ is $T_3 = \mathcal{O}(\frac{1}{\varepsilon} nH \log U)$. We conclude that the computational complexity of Procedure MULTICAST is $T_1 + T_2 + T_3 = \mathcal{O}(\frac{1}{\varepsilon^2} nH^2 \log U)$ and the theorem follows. ■