Optimal Partition of QoS Requirements on Unicast Paths and Multicast Trees

Dean H. Lorenz and Ariel Orda

Abstract—**We investigate the problem of optimal resource allocation for end-to-end QoS requirements on unicast paths and multicast trees. Specifically, we consider a framework in which resource allocation is based on local QoS requirements at each network link, and associated with each link is a cost function that increases with the severity of the QoS requirement. Accordingly, the problem that we address is how to partition an end-to-end QoS requirement into local requirements, such that the overall cost is minimized. We establish efficient (polynomial) solutions for both unicast and multicast connections. These results provide the required foundations for the corresponding QoS routing schemes, which identify either paths or trees that lead to minimal overall cost. In addition, we show that our framework provides better tools for coping with other fundamental multicast problems, such as dynamic tree maintenance.**

Keywords—**QoS partitioning, QoS-dependent costs, Unicast, Multicast, Routing, Broadband networks, Convex costs.**

I. Introduction

Broadband integrated services networks are expected to support multiple and diverse applications, with various quality of service (QoS) requirements. Accordingly, a key issue in the design of broadband architectures is how to provide the resources in order to meet the requirements of each connection.

Supporting QoS connections requires the existence of several network mechanisms. One is a QoS routing mechanism, which sets the connection's topology, *i.e.*, a unicast path or multicast tree. A second mechanism is one that provides QoS guarantees given the connection requirements and its topology. Providing these guarantees involves allocating resources, e.g., bandwidth and buffers, on the various network elements. Such a consumption of resources has an obvious cost in terms of network performance. The cost at each network element inherently depends on the local availability of resources. For instance, consuming all the available bandwidth of a link, considerably increases the blocking probability of future connections. Clearly, the cost of establishing a connection (and allocating the necessary resources) should be a major consideration of the connection (call) admission process. Hence, an important network optimization problem is how to establish QoS connections in a way that minimizes their implied costs. Addressing this problem impacts both the routing process and the allocation of resources on the selected topology. The latter translates into an end-to-end QoS requirement partition problem, namely *local* allocation of QoS *requirements* along the topology.

The support of QoS connections has been the subject of extensive research in the past few years. Several studies and proposals considered the issue of QoS routing, e.g., [1], [3], [9], [21], [24] and references therein. Mechanisms for providing various QoS guarantees have been also widely investigated, e.g. [8], [22], [25], [26]. Although there are proposals for resource reservation, most notably RSVP [4], they address only the signaling mechanisms and do not provide the allocation policy. Indeed, the issue of *optimal* resource allocation, from a network perspective, has been scarcely addressed. Some studies, e.g. [13], consider the specific, simple, case of constant link costs, which are independent of the QoS (delay) supported by the link. Pricing, as a network optimization mechanism, has been the subject of recent studies, however they mostly considered either a basic *best effort* service environment, e.g. [14], [18], or simple, single link [17] and parallel link [20] topologies.

In this paper, we investigate the problem of optimal resource allocation for end-to-end QoS requirements on given unicast paths and multicast trees. Specifically, we consider a framework in which resource allocation is based on the partition of the end-to-end QoS requirement into local QoS requirements at each network element (link). We associate with each link a cost function that increases with the severity of the local QoS requirement. As will be demonstrated in the next section, this framework is consistent with the proposals for QoS support on broadband networks. Accordingly, the problem that we address is how to partition an end-to-end QoS requirement into local requirements, such that the overall cost is minimized. This is shown to be intractable even in the (simpler) case of unicast connections. Yet, we are able to establish efficient (polynomial) solutions for both unicast and multicast connections, by imposing some (weak) assumptions on the costs. These results provide the required foundations for the corresponding QoS routing schemes, which identify either paths or trees that lead to minimal overall cost. Moreover, we indicate how the above framework provides better tools for coping with fundamental multicast problems, such as the dynamic maintenance of multicast trees.

A similar framework was investigated in [5], [19]. There too, it was proposed that end-to-end QoS requirements should be partitioned into local (link) requirements and the motivation for this approach was extensively discussed. [19] discussed *unicast* connections and focused on loss rate guarantees. It considered a utility function, which is equivalent to (the minus of) our cost function. However, rather than maximizing the overall utility, [19] focused on the optimization of the *bottleneck* utility over the connection's path. That is, their goal was to partition the end-to-end loss rate requirement into link requirements over a given path, so as to maximize the minimal utility value over the path links. Specifically, [19] investigated the performance of a heuristic that equally partitioned the loss rate requirements over the links. By way of simulations, it was indicated that the performance of that heuristic was reasonable for paths with few (up to five) links and tight loss rate requirements; this finding was further supported by analysis. However, it was indicated that performance deteriorated when either the number of links became larger or when the connection was more tolerant to packet loss. It was concluded that for such cases, as well as for alternate

Dean Lorenz and Ariel Orda are at the Department of Electrical Engineering at the Technion—Israel Institute of Technology. email:{deanh@tx,ariel@ee}.technion.ac.il

QoS requirements (such as delay), the development of *optimal* QoS partition schemes is of interest. [5] considered *multicast* trees and a cost function that is a special case of ours. Each tree link was assigned an upper bound on the cost, and the goal was to partition the end-to-end QoS into local requirements, so that no link cost exceeds its bound. Specifically, [5] considered two heuristics, namely *equal* and *proportional* partitions, and investigated their performance by way of simulations. It was demonstrated that proportional partition offers better performance than equal partition, however it is not optimal. [5] too concluded that more complex (optimal) partition schemes should be investigated. These two studies provide interesting insights into our framework, and strongly motivate the optimization problems that we investigate.

Another sequence of studies that is related to the present one is [9], [16]. These studies investigated QoS partitioning and routing for unicast connections, in networks with uncertain parameters. Their goal was to select a path, and partition the QoS requirements along it, so as to maximize the probability of meeting the QoS requirements. As shall be shown, the link probability distribution functions considered in [9], [16] correspond to a special case of the cost functions considered in the present paper. The algorithms presented in [16] solve both the routing and the QoS partition problem for unicast connections, under certain assumptions. The present study offers an improved, less restrictive, solution for unicast, and, more importantly, a generalized solution for multicast.

The general *resource allocation* problem is a constrained optimization problem. Due to its simple structure, this problem is encountered in a variety of applications and has been studied extensively [12]. Optimal Partition of end-to-end QoS requirements over *unicast* paths is a special case of that problem, however the *multicast* version is not. Our main contribution is in solving the problem for multicast connections. We also present several algorithms for the unicast problem, emphasizing network related aspects, such as dynamic implementation.

The rest of this paper is structured as follows. Section II formulates the model and problems, and relates our framework to QoS network architectures. The optimal partition problem for unicast connections is investigated in Section III. The optimal partition problem for multicast connections is discussed in Section IV. This problem is solved using a similar approach to that used for unicast, nonetheless the analysis and solution structure turn out to be much more complex. Section V relates these findings to unicast and multicast QoS routing. Finally, concluding remarks are presented in Section VI. Due to space limits, some technical details and proofs are omitted from this version and can be found in [15].

II. Model and Problems

In this section we present our framework and introduce the QoS partition problem. We assume a single connection with a given topology, *i.e.*, a path $\boldsymbol{p}$ for unicast, or a tree $\boldsymbol{T}$ for multicast. The problem of finding such topologies, namely QoS routing, is briefly discussed in Section V. For clarity, we detail here only the framework for unicast connections. The definitions and terminology for multicast trees are similar and are presented, together with the corresponding solution, in Section IV.

A. QoS Requirements

A QoS partition of an end-to-end QoS requirement Q, on a path $\boldsymbol{p}$, is a vector $\boldsymbol{x_p} = \{x_l\}_{l\in\boldsymbol{p}}$ of local QoS requirements, which satisfies the end-to-end QoS requirement, Q.

There are two fundamental classes of QoS parameters: *bottleneck* parameters, such as bandwidth, and *additive* parameters, such as delay and jitter. Each class induces a different form of our problem, and the complexities of the solutions are vastly different. For *bottleneck* parameters, we necessarily have $x_l = Q$ for all $l \in \boldsymbol{p}$,[1] because Q is determined by the bottleneck link, *i.e.*, $Q = \min_{l\in\boldsymbol{p}} x_l$. Since allocating more than Q induces a higher cost, yet does not improve the overall QoS, the optimal partition is $\boldsymbol{x_p^*} = \{Q\}_{l\in\boldsymbol{p}}$. For *additive* QoS requirements, a feasible partition, $\boldsymbol{x_p}$, must satisfy $\sum_{l\in\boldsymbol{p}} x_l \le Q$. In this case, the optimal QoS partition problem is intractable [16], however we will show that, by restricting ourselves to *convex* cost functions, we can achieve an efficient (tractable) solution. Some QoS parameters, such as loss rate, are *multiplicative*, *i.e.*, $Q = \prod_{l\in\boldsymbol{p}}(x_l)$. For instance, for a loss rate QoS requirement L, we have $Q = 1 - L$. This case too can be expressed as an additive requirement, by solving for $(-\log Q)$; indeed, the end-to-end requirement becomes $(-\log Q) = \sum_{l\in\boldsymbol{p}}(-\log x_l)$, *i.e.*, an additive requirement.

There are QoS provision mechanisms, in which the (additive) delay bounds are determined by a (bottleneck) "rate". A notable example is the Guaranteed Service architecture for IP [23], which is based on rate-based schedulers [8], [22], [25], [26]. In some cases, such mechanisms may allow to translate a delay requirement on a given path into a bottleneck (rate) requirement, hence the partitioning is straightforward. However, such a translation cannot be applied in general, e.g. due to complications created by topology aggregation and hierarchical routing.[2] Hence, our study focuses on the general partition problem of additive QoS requirements.

B. Cost Functions

As mentioned, we associate with each local QoS requirement value x_l, a cost $c_l(x_l)$. We make the natural assumption that $c_l(x_l)$ is higher as x_l is tighter. For instance, when x_l stands for delay, $c_l(x_l)$ is a nonincreasing function, whereas when x_l stands for bandwidth, $c_l(x_l)$ is nondecreasing. We further assume that the cost functions are (weakly) convex. The overall cost of a partition is the sum of the local costs, *i.e.*, $c(\boldsymbol{x_p}) = \sum_{l\in\boldsymbol{p}} c_l(x_l)$.

The cost may reflect the resources, such as bandwidth, needed to guarantee the QoS requirement. Alternatively, the cost may be the price that the user is required to pay to guarantee a specific QoS. The cost may be associated with either the set-up or the run-time phase of a connection. Our model considers a single connection, however the affects of other connections can be integrated into the cost function. For example, it may be used for network management to discourage the usage of congested links, by assigning higher costs to those links.

A particular form of cost evolves in models that consider uncertainty in the available parameters at the connection setup

[1] This is also true for any multicast tree.

[2] Indeed, the ATM hierarchical QoS routing protocol [21], requires local (per cluster) QoS guarantees.

phase [9], [16], which we now briefly overview. In such models, associated with each link is a probability of failure $f_l(x_l)$, when trying to set up a local QoS requirement of x_l. The optimal QoS partition problem is then to find a QoS partition that minimizes the probability of failure; that is, it minimizes the product $\prod_{l\in\boldsymbol{p}} f_l(x_l)$. Since we have $\log\left(\prod_{l\in\boldsymbol{p}} f_l(x_l)\right) = \sum_{l\in\boldsymbol{p}} \log f_l(x_l)$, we can restate this problem back as a summation, namely we define a cost function for each link, $c_l(x) = \log f_l(x)$, and solve for these costs.

C. Problem Formulation

The optimal QoS partition problem is then defined as follows.

Problem OPQ (Optimal Partition of QoS): Given a path $\boldsymbol{p}$ and an end-to-end QoS requirement Q, find a QoS partition $\boldsymbol{x}^*_{\boldsymbol{p}} = \{x^*_l\}_{l\in\boldsymbol{p}}$, such that $c(\boldsymbol{x}^*_{\boldsymbol{p}}) \le c(\boldsymbol{x}_{\boldsymbol{p}})$, for any (other) QoS partition $\boldsymbol{x}_{\boldsymbol{p}}$.

This study focuses on the solution of Problem OPQ for additive QoS parameters, which, as mentioned, presents a considerably more complex problem than the bottleneck version. In Section III we solve the problem for unicast paths, and in Section IV we generalize the solution to multicast trees. For clarity, and without loss of generality, we concretize the presentation on end-to-end delay requirements.

D. Example

We illustrate Problem OPQ through an example. Consider the network of Fig. 1. The source is node A and the destination is node C. The cost function for each link i is

$$c_i(x) = \begin{cases} \frac{\kappa_i}{x-\kappa_i} & \text{if} \quad x > \kappa_i, \\ \infty & \text{if} \quad x \le \kappa_i; \end{cases}$$

where κ_i is the minimal delay that can be guaranteed on link i when utilizing all its available resources. The cost increases as the delay requirement approaches this limit and is infinite for $x \le \kappa_i$. We set $\kappa_1 = 1, \kappa_2 = 3$ and the end-to-end delay requirement is $D = 12$.

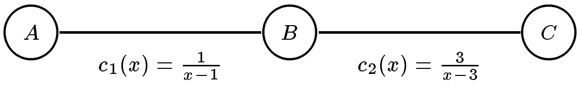


Fig. 1. Example: instance of Problem OPQ

We consider three allocation schemes: *equal* [5], [19], *proportional* [5] and the *optimal* allocation.

equal allocation $x_1 = x_2 = 12/2 = 6$. In this case $c_1(6) = 1/5, c_2(6) = 1$ and $c(\boldsymbol{x}) = 1.2$.

proportional allocation The allocation is proportional to the "utilization" of the links, namely $x_1 : x_2 = \kappa_1 : \kappa_2 \Rightarrow x_1 = 3, x_2 = 9$. We get $c_1(3) = c_2(9) = 1/2$ and $c(\boldsymbol{x}) = 1$.

optimal allocation The solution of Problem OPQ in this case is $x_1 = 4, x_2 = 8$. The costs are $c_1(4) = 1/3, c_2(8) = 3/5$ and the overall optimal cost is $c(\boldsymbol{x}) = 0.93$.

We see that the proportional allocation scheme renders a better cost than the equal allocation, however it is inferior to the optimal allocation.

The difference between the respective performances grows with the number of hops. Moreover, it may be substantially large even for the above simple two-hop path, with an alternate set of cost functions. For instance, consider the following cost function, which incorporates a higher penalty for utilizing link resources:

$$c_i(x) = \frac{\kappa_i}{(x-\kappa_i)^n},$$

where $n > 1$. For $n = 3$, the overall cost $c(\boldsymbol{x})$ is 0.12 with an equal allocation, 0.14 with a proportional allocation and 0.06 with the optimal one. We see that, in this case, the proportional allocation is worse than the equal allocation, and the optimal cost is significantly better than the rest.

III. Solution to Problem OPQ

In this section we investigate the properties of optimal solutions to Problem OPQ for additive QoS parameters and present efficient algorithms. These results will be used in the next section to solve Problem MOPQ, *i.e.*, the generalization of Problem OPQ to multicast trees. As mentioned, Problem OPQ is a special case of the resource allocation problem. The fastest solution to this problem, [6], requires $O(|\boldsymbol{p}|\log(D/|\boldsymbol{p}|))$. In Section III-B we present a *greedy* pseudo-polynomial solution, which provides appealing advantages for distributed and dynamic implementations, as discussed in Section III-C. In Section III-D we present a polynomial solution that, albeit of slightly higher complexity than that of [6], provides the foundations of our solution to Problem MOPQ. Finally, in Section III-E we discuss special cases with lower complexity.

As mentioned, we assume that the QoS parameter is end-to-end delay and an allocation is a delay requirement on each link. In practice, network resources cannot be reserved in an arbitrary fine granularity, thus our resource allocation problem is discrete, *i.e.*, we can assume that all parameters are integers.[3] We further assume that the link cost functions are nonincreasing with the delay and (weakly) convex.

A. Notations

$\boldsymbol{x}_{\boldsymbol{p}}(D)$ is a *feasible* partition of an end-to-end delay requirement D on the path $\boldsymbol{p}$ if it satisfies $\sum_{l\in\boldsymbol{p}} x_l \le D$. We omit the subscript $\boldsymbol{p}$ and/or the argument D when there is no ambiguity. $\boldsymbol{x}^*_{\boldsymbol{p}}(D)$ denotes the optimal partition, namely the solution of Problem OPQ for an end-to-end delay requirement D and a path $\boldsymbol{p}$. We denote by $|\boldsymbol{x}_{\boldsymbol{p}}|$ the norm $\sum_{l\in\boldsymbol{p}} |x_l|$, hence $\boldsymbol{x}$ is feasible if $|\boldsymbol{x}| \le D$.

The average *δ-increment* gain for a link l , *i.e.*, the average gain in cost when incrementing the delay allocation to l by δ, is denoted by

$$\Delta_l(x,\delta) \equiv (c_l(x+\delta) - c_l(x))/|\delta|.$$

Note that, as the cost is nonincreasing with the delay, $\delta > 0$ implies $\Delta_l(x,\delta) \le 0$. We assume that, for each link l, $c_l(x)$ is weakly convex, namely $\Delta_l(x,\delta)$ is nonincreasing with x for any negative (fixed) δ. As the allocation to each link must be nonnegative, we define $\Delta_l(0,\delta) = \infty$ for any $\delta < 0$. The average *δ-move* gain from a link e to a link l, *i.e.*, the average gain in cost

[3] One may consider our solutions to be *1-optimal*, however we allow arbitrary precision and the complexity is linear with the size of the number of bits required to represent a delay.

when moving δ from the allocation to link e to the allocation to link l, is denoted by

$$\Delta_{e\to l}\left(\boldsymbol{x},\delta\right) \equiv \Delta_l\left(x_l,\delta\right) + \Delta_e\left(x_e,-\delta\right).$$

B. Pseudo-polynomial Solution

Problem OPQ is a special case of the general *resource allocation* problem which has been extensively investigated [6], [11], [12]. With the (weak) convexity assumption on the cost functions, it is a convex optimization problem with a simple constraint. It can be proved [12] that a greedy approach is applicable for such problems, namely it is possible to find an optimal solution by performing *locally* optimal decisions.

GREEDY-ADD *($D, \delta, c(\cdot), \boldsymbol{p}$):*
1 $\boldsymbol{x} \leftarrow \{0\}_{l\in\boldsymbol{p}}$
2 while $|\boldsymbol{x}| < D$ do
3 $\quad e \leftarrow arg\min_{l\in\boldsymbol{p}} \Delta_l\left(x_l,\delta\right)$
4 $\quad x_e \leftarrow x_e + \delta$
5 **return** $\boldsymbol{x}$

Fig. 2. Algorithm GREEDY-ADD

Algorithm GREEDY-ADD (Fig. 2) employs such a greedy approach, with a granularity of δ. It starts from the zero allocation and adds the delay δ units at a time, each time augmenting the link where the (negative) δ-increment gain is minimal, namely where it most affects the cost. Using an efficient data structure (e.g., a heap), each iteration requires $O(\log|\boldsymbol{p}|)$, which leads to an overall complexity of $O(\frac{D}{\delta}\log|\boldsymbol{p}|)$. In [11] it is shown that the solution is δ-optimal in the following sense: if $\boldsymbol{x}^\delta$ is the output of the algorithm, then there exists an optimal solution $\boldsymbol{x}^*$, such that $x^*_l \geq x^\delta_l - \delta$ for all $l \in \boldsymbol{p}$ and $|\boldsymbol{x}^* - \boldsymbol{x}^\delta| \leq |\boldsymbol{p}|\delta$.[4]

Algorithm GREEDY-MOVE (Fig. 3) is a modification of Algorithm GREEDY-ADD that, as shall be explained in Section III-C, has important practical advantages. The algorithm starts from *any* feasible allocation and modifies it until it reaches an optimal one. Each iteration performs a greedy move, namely the δ-move with minimal (negative) gain.

GREEDY-MOVE *($\boldsymbol{x}, \delta, c(\cdot), \boldsymbol{p}$):*
1 loop
2 $\quad e,l \leftarrow arg\min_{e,l\in\boldsymbol{p}} \Delta_{e\to l}\left(\boldsymbol{x},\delta\right)$
3 $\quad$ if $\Delta_{e\to l}\left(\boldsymbol{x},\delta\right) \geq 0$ **return** $\boldsymbol{x}$
4 $\quad x_e \leftarrow x_e - \delta$
5 $\quad x_l \leftarrow x_l + \delta$

Fig. 3. Algorithm GREEDY-MOVE

Let $\varphi(\boldsymbol{x})$ be the distance of a given partition from the optimal one, namely $\varphi(\boldsymbol{x}) \equiv |\boldsymbol{x} - \boldsymbol{x}^*|$, where $\boldsymbol{x}^*$ is the optimal partition that is nearest to $\boldsymbol{x}$. The next lemma implies that Algorithm GREEDY-MOVE indeed reaches a δ-optimal solution.

Lemma 1: Each iteration of Algorithm GREEDY-MOVE decreases $\varphi(\boldsymbol{x})$ by at least δ, unless $\boldsymbol{x}$ is a δ-optimal partition.

Lemma 1 implies that Line (3) can be used as a (δ-)optimality check. It also implies that the algorithm terminates with a δ-optimal solution and that the number of iterations is proportional to $\varphi(\boldsymbol{x})$. Theorem 1 summarizes this discussion.

[4] In particular, for $\delta = 1$ we have $|\boldsymbol{x}^* - \boldsymbol{x}^1| \leq |\boldsymbol{p}|$, which establishes the error due to our assumption of integer allocations.

Theorem 1: Given a feasible partition $\boldsymbol{x}$, Algorithm GREEDY-MOVE finds a δ-optimal solution to Problem OPQ in $O(\frac{\varphi(\boldsymbol{x})}{\delta}\log|\boldsymbol{p}|)$.

Proof: By Lemma 1, there are at most $\varphi(\boldsymbol{x})/\delta$ iterations and a δ-optimal solution is achieved. Each iteration can be implemented in $O(\log|\boldsymbol{p}|)$ and the result follows. ■

Setting $\delta = 1$ we have an immediate corollary for optimal solutions.

Corollary 1: Given a feasible partition $\boldsymbol{x}$, Algorithm GREEDY-MOVE solves Problem OPQ in $O(\varphi(\boldsymbol{x})\log|\boldsymbol{p}|)$.

The linear dependency on $\varphi(\boldsymbol{x})$ means that the closer the initial allocation is to an optimal allocation, the less computations are needed to reach a solution. That is, by starting from a "good" initial allocation we can improve the complexity. The initial allocation could be based on some (simple) heuristic assumption or, alternatively, on a previously used allocation.

C. Dynamic and Distributed Implementation

Algorithm GREEDY-MOVE can be used as a dynamic scheme that *reacts* to changes in the cost functions *after* an optimal partition has been established. Note that the complexity is proportional to the allocation modification implied by the cost changes, meaning that small allocation changes incur only a small number of computations.

Algorithm GREEDY-MOVE can be employed in a distributed fashion. Each iteration can be implemented by a control message that traverses back and forth between the source and destination. At each traversal, the links e, l of Line (2) are identified, and the allocation change of the *previous* iteration is performed. This requires $O(|\boldsymbol{p}|\varphi(\boldsymbol{x})/\delta)$ end-to-end messages (where $\delta \geq 1$). Such a distributed implementation also exempts us from having to advertise the updated link cost functions.

D. Polynomial Solution

In this section we present an improved algorithm of polynomial complexity in the input size (*i.e.*, $|\boldsymbol{p}|$ and $\log D$). In Section IV, we derive a solution to Problem MOPQ using a similar technique.

Algorithm BINARY-OPQ (Fig. 4) finds optimal solutions for different values of δ. The algorithm consecutively considers smaller values of δ, until the minimal possible value is reached, at which point a (global) optimum is identified.

BINARY-OPQ *($D, c(\cdot), \boldsymbol{p}$):*
1 $\delta \leftarrow D/|\boldsymbol{p}|$
2 start from the partition $\boldsymbol{x} = \{D, 0, \ldots, 0\}$
3 repeat
4 $\quad \boldsymbol{x} \leftarrow$ GREEDY-MOVE$(\boldsymbol{x}, \delta, c(\cdot), \boldsymbol{p})$
5 $\quad \delta \leftarrow \lceil \delta/2 \rceil$
6 until $\delta \leq 1$

Fig. 4. Algorithm BINARY-OPQ

Obviously, the algorithm finds an optimal solution, since its last call to GREEDY-MOVE is with $\delta = 1$. The number of iterations is clearly of order $O\left(\log(D/|\boldsymbol{p}|)\right)$. We need to bound the number of steps required to find the δ-optimal partition at Line (4). Each iteration (except the first, for which $D/\delta = |\boldsymbol{p}|$) starts from a 2δ-optimal partition and employs greedy moves until it reaches a δ-optimal partition. This bound is the same for all iterations, since it is a bound on the distance between a 2δ-optimal partition and a δ-optimal partition.

Lemma 2: Let $\boldsymbol{x} \leftarrow$GREEDY-MOVE$(\boldsymbol{x}, 2, c(\cdot), \boldsymbol{p})$. Then $\varphi(\boldsymbol{x}) \leq |\boldsymbol{p}|$.

The above lemma, proven in [15], resembles the proximity theorem presented in [11].

Theorem 2: Algorithm BINARY-OPQ solves Problem OPQ in $O\left(|\boldsymbol{p}| \log |\boldsymbol{p}| \log(D/|\boldsymbol{p}|)\right)$.

Proof: The final call to GREEDY-MOVE is with $\delta = 1$, which produces an optimal allocation. By Lemma 2 and Theorem 1, each call to GREEDY-MOVE requires $O(|\boldsymbol{p}| \log |\boldsymbol{p}|)$. Since there are $O(\log(D/|\boldsymbol{p}|))$ such calls, the result follows. ∎

E. Faster Solutions

The following lemma, which is a different form of the optimality test in Algorithm GREEDY-MOVE, provides a useful threshold property of optimal partitions.

Lemma 3: Let $\Delta^* \equiv \min_{l \in \boldsymbol{p}} \Delta_l\left(x_l, -1\right)$. Then, $\boldsymbol{x}$ is optimal if and only if

$$\Delta_e\left(x_e, -1\right) \geq \Delta^* \geq -\Delta_l\left(x_l, 1\right) \quad \text{for all } e, l \in \boldsymbol{p}.$$

For all $l \in \boldsymbol{p}$, $\Delta_l(d, -1)$ is a nonincreasing function (since c_l is convex) and (by definition) $\Delta_l(d+1, -1) = -\Delta_l(d, 1)$. Thus, the threshold Δ^* in Lemma 3 relates to the optimal allocation as follows:

$$\forall l \in \boldsymbol{p} \quad x_l^* \geq d \Leftrightarrow \Delta_l\left(d, -1\right) \geq \Delta^*. \tag{1}$$

From the monotonicity of $\Delta_l(d, -1)$, we get $\gamma \geq \Delta^*$ for all $\gamma \in \{\Delta_l(d, -1) \mid 0 \leq d \leq x_l^*, l \in \boldsymbol{p}\}$, and $\gamma < \Delta^*$ for all $\gamma \in \{\Delta_l(d, -1) \mid x_l^* < d \leq D, l \in \boldsymbol{p}\}$. This implies that Δ^* (and subsequently an optimal solution to Problem OPQ) can be found by selecting the D largest element from the set $\{\Delta_l(d, -1) \mid 0 \leq d \leq D, l \in \boldsymbol{p}\}$. Alternatively, *given* Δ^*, the optimal allocation on each link can be found in $O(\log |D|)$ and $O(|\boldsymbol{p}| \log |D|)$ for all. If the cost functions are *strictly* convex then the inverse functions of $\Delta_l(d, -1)$ exist and, in some cases, given Δ^*, one can use (1) to find an optimal solution in $O(|\boldsymbol{p}|)$. In [16], a binary search is employed for finding Δ^*, leading to an overall solution of $O(|\boldsymbol{p}| \log(\Delta^{\max}))$, where $\Delta^{\max} \equiv \max\{\Delta_l(d, -1) \mid 0 \leq d \leq D, l \in \boldsymbol{p}\}$. Note that $\log(\Delta^{\max})$ is bounded by the complexity of representing a cost value. Moreover, for certain cost functions, Δ^* can be found analytically. For instance, in [16] we provide an $O(|\boldsymbol{p}|)$ solution for cost functions that correspond to delay uncertainty with uniform probability distributions.

IV. SOLUTION TO MULTICAST OPQ (MOPQ)

In this section we solve Problem OPQ for multicast trees. Specifically, given a multicast tree, we need to allocate the delay on each link, such that the end-to-end bound is satisfied on every path from the source to any member of the multicast group, and the cost associated with the whole multicast tree is minimized.

We denote the source (root) of the multicast by s and the set of destinations, *i.e.*, the multicast group, by M. A multicast tree is a set of edges $\boldsymbol{T} \subseteq E$ such that for all $v \in M$ there exists a path, $\boldsymbol{p}^{\boldsymbol{T}}(s, v)$, from s to v on links that belong to the tree $\boldsymbol{T}$. We assume there is only one outgoing link from the source s,[5] and denote this link by r.

$N\left(l = (u, v)\right)$ denotes all the outgoing links from v, *i.e.*, all of l's neighbors; when $N(l)$ is an empty set then we call l a *leaf*. $\boldsymbol{T}_l$ is the whole sub-tree originating from l (including l itself). The *branches* of $\boldsymbol{T}$ are denoted by $\hat{\boldsymbol{T}} \equiv \boldsymbol{T} \setminus \{r\}$. Observe that $\hat{\boldsymbol{T}} = \cup_{l \in N(r)} \boldsymbol{T}_l$.

A *feasible* delay partition for a multicast tree $\boldsymbol{T}$, is a set of link-requirements $\boldsymbol{x_T}(D) = \{x_l\}_{l \in \boldsymbol{T}}$ such that $\sum_{l \in \boldsymbol{p}^{\boldsymbol{T}}(s,v)} x_l \leq D$ for all $v \in M$.[6]

We can now define Problem OPQ for multicast trees.

Problem MOPQ (Multicast OPQ): Given a multicast tree $\boldsymbol{T}$ and an end-to-end delay requirement D, find a feasible partition $\boldsymbol{x}_{\boldsymbol{T}}^*(D)$, such that $c\left(\boldsymbol{x}_{\boldsymbol{T}}^*(D)\right) \leq c\left(\boldsymbol{x_T}(D)\right)$, for every (other) feasible partition $\boldsymbol{x_T}(D)$.

Remark 1: If there is more than one outgoing link from the source, then we can simply solve Problem MOPQ independently, for each tree $\boldsymbol{T}_{r_i}$ corresponding to an outgoing link r_i from s. Thus, our assumption that there is only one outgoing link from r, does not limit the solution.

Remark 2: All the multicast group members have the same delay requirement, hence feasibility for the leaves implies feasibility for all nodes of the tree. Thus, any multicast group members that are intermediate nodes of the tree can be ignored. In Section IV-E we address the heterogeneous case where this observation is no longer true.

We denote by MOPQ$(\boldsymbol{T}, D)$ the set of optimal partitions on a tree $\boldsymbol{T}$ with delay D. $c_{\boldsymbol{T}}\left(d\right)$ denotes the *tree cost function*, *i.e.*, the cost of (optimally) allocating a delay d on the tree $\boldsymbol{T}$. In other words, $c_{\boldsymbol{T}}\left(d\right) = c\left(\boldsymbol{x}_{\boldsymbol{T}}^*(d)\right)$, where $\boldsymbol{x}_{\boldsymbol{T}}^* \in$ MOPQ$(\boldsymbol{T}, d)$.

A. Greedy Properties

The general resource allocation problem can be stated with tree-structured constraints and solved in a greedy fashion [12]. An efficient $O(|\boldsymbol{T}| \log |\boldsymbol{T}| \log D)$ algorithm is given in [11]. However, that version of the resource allocation problem (henceforth, the "tree version") has a different structure than Problem MOPQ. In particular, the simple greedy approach, namely repeated augmentation of the link that most improves the overall cost, fails for Problem MOPQ. However, as we show below, some greedy structure is maintained, as follows: if at each iteration we augment the *sub-tree* that most improves the overall cost, then an optimal solution is achieved.

The main difference between the "tree version" and Problem MOPQ is that, in the latter, the constraints are not on sub-trees, but rather on *paths*. The greedy approach fails because of the dependencies among paths. On the other hand, we note that the "tree version" may be applicable to some other multicast resource allocation problems, in which a feasible allocation must recursively satisfy, for any sub-tree $\boldsymbol{T}_e$, $\sum_{l \in \boldsymbol{T}_e} x_l \leq Q(\boldsymbol{T}_e)$, where $Q(\boldsymbol{T}_e)$ is some arbitrary (sub-tree) constraint.

We proceed to establish the greedy structure of Problem MOPQ. First, we show that if all link cost functions are convex, then so is the tree cost function.

[5] See Remark 1 in the following.

[6] Again, when no ambiguity exists, we omit the sub-script $\boldsymbol{T}$ and/or the argument D.

The next lemma (see appendix for proof) shows that this is true for a tree with depth 2, *i.e.*, when all links in $N(r)$ are leaves.

Lemma 4: If $\{c_l\}_{l\in\boldsymbol{T}}$ are convex, and $\boldsymbol{T}$ has a depth of 2, then $c_{\boldsymbol{T}}(d)$ is convex.

We can now generalize Lemma 4 to deeper trees by starting from the leaves of the tree and working our way towards the root, replacing depth-2 sub-trees with equivalent convex links. We can continue this process, until the whole tree collapses into a single convex link. This is formally stated in the next lemma (see appendix for proof).

Lemma 5: If $\{c_l\}_{l\in\boldsymbol{T}}$ are convex then so is $c_{\boldsymbol{T}}(d)$.

By Lemma 5, we can replace $\boldsymbol{T}$ by an equivalent convex link. Any sub-tree $\boldsymbol{T}_l$, can also be replaced with an equivalent convex link, hence so can $\hat{\boldsymbol{T}}$. However, these results apply only if the allocation on every sub-tree is optimal for the sub-tree. This property is sometimes referred to as the "optimal sub-structure" property [2], and is the hallmark of the applicability of both *dynamic-programming* and *greedy* methods.

The next lemma implies that, for any optimally partitioned trees, we can apply the greedy properties of Section III. That is, the partition on r and $\hat{\boldsymbol{T}}$ is a solution to Problem OPQ on the 2-link path $(r, \hat{\boldsymbol{T}})$. This suggests that employing greedy moves between r and $\hat{\boldsymbol{T}}$ solves Problem MOPQ, and this method can be applied recursively for the sub-trees of $\boldsymbol{T}$. Indeed, this scheme is used by the algorithms presented in the next sections.

Lemma 6: Let $\boldsymbol{x}^*_{\boldsymbol{T}} \in$ MOPQ$(\boldsymbol{T}, D)$. Let $e \in N(r)$ and let the *sub-partition* $\boldsymbol{x}^e_{\boldsymbol{T}_e}(D^e)$ be defined as $\boldsymbol{x}^e_{\boldsymbol{T}_e}(D^e) = \{x^e_l = x^*_l\}_{l\in\boldsymbol{T}_e}$, where $D^e = D - x^*_r$. Then $\boldsymbol{x}^e_{\boldsymbol{T}_e} \in$ MOPQ$(\boldsymbol{T}_e, D^e)$.

Proof: Let v be the root of the sub-tree $\boldsymbol{T}_e$, and let u be any member of both the multicast group and the sub-tree, *i.e.*, $u \in M \cap \boldsymbol{T}_e$. Since there are no cycles in the tree $\boldsymbol{T}$, and u is reachable through r, we must have $\sum_{l\in\boldsymbol{p}^{\boldsymbol{T}_e}(v,u)} x^e_l = \sum_{l\in\boldsymbol{p}^{\boldsymbol{T}}(s,u)} x^*_l - x^*_r = D^e$. Hence, the sub-partition $\boldsymbol{x}^e_{\boldsymbol{T}_e}(D^e)$ is feasible for a delay bound D^e on $\boldsymbol{T}_e$. The converse is also true, *i.e.*, any feasible partition $\boldsymbol{x}^A_{\boldsymbol{T}_e}(D^e)$ satisfies $D_{(s,u)} \le D$ for all $u \in M \cap \boldsymbol{T}_e$. Therefore, the partition $\boldsymbol{x}_{\boldsymbol{T}}(D) = \left(\boldsymbol{x}^*_{\boldsymbol{T}} \setminus \boldsymbol{x}^e_{\boldsymbol{T}_e}\right) \cup \boldsymbol{x}^A_{\boldsymbol{T}_e}$ is feasible. The corresponding cost is $c\left(\boldsymbol{x}^*_{\boldsymbol{T}}\right) - c\left(\boldsymbol{x}^e_{\boldsymbol{T}_e}\right) + c\left(\boldsymbol{x}^A_{\boldsymbol{T}_e}\right)$. Since $\boldsymbol{x}^*_{\boldsymbol{T}}$ is an optimal partition, this cost is at least $c\left(\boldsymbol{x}^*_{\boldsymbol{T}}\right)$, which implies that $c\left(\boldsymbol{x}^e_{\boldsymbol{T}_e}\right) \le c\left(\boldsymbol{x}^A_{\boldsymbol{T}_e}\right)$. As $\boldsymbol{x}^A_{\boldsymbol{T}_e}$ is any feasible sub-partition on $\boldsymbol{T}_e$, we must have $\boldsymbol{x}^e_{\boldsymbol{T}_e}(D^e) \in$ MOPQ$(\boldsymbol{T}_e, D^e)$. ■

B. Pseudo-polynomial Solution

We present first a pseudo-polynomial solution, which is based on the employment of moves between r and $\hat{\boldsymbol{T}}$. The major difficulty of this method is the fact that $c_{\hat{\boldsymbol{T}}}(d)$ is unavailable. Computing $c_{\hat{\boldsymbol{T}}}(d)$ for a specific d requires solving Problem MOPQ for $\hat{\boldsymbol{T}}$. Fortunately, we can easily compute $c_{\hat{\boldsymbol{T}}}(d+\delta)$, given some $\boldsymbol{x}^*_{\hat{\boldsymbol{T}}}(d) \in$ MOPQ$(\hat{\boldsymbol{T}}, d)$. Since the greedy approach is applicable, we may simply perform a greedy augmentation and recompute the cost. Note that adding δ to $\hat{\boldsymbol{T}}$ requires adding δ to all $\{\boldsymbol{T}_l\}_{l\in N(r)}$. In the worst case, this must be done recursively for all the sub-trees and $O(|\boldsymbol{T}|)$ links are augmented.

Procedure TREE-ADD (Fig. 5) performs a δ-augmentation on a tree $\boldsymbol{T}$. We assume that, for each sub-tree $\boldsymbol{T}_l$, the value of

```
TREE-ADD ( x, δ, T ):
1  if Δ_r(x_r, δ) = Δ_T(δ) then
2        x_r ← x_r + δ
3  else
4        for each l ∈ N(r) do
5              TREE-ADD (x, δ, T_l)
6  Δ_T(δ) ← min{Δ_r(x_r, δ), Σ_{l∈N(r)} Δ_{T_l}(δ)} ^a
7  Δ_T(−δ) ← min{Δ_r(x_r, −δ), Σ_{l∈N(r)} Δ_{T_l}(−δ)} ^a

^a If r is a leaf we define the sum to be ∞.
```

Fig. 5. Procedure TREE-ADD

$\Delta_{\boldsymbol{T}_l}(D^{\boldsymbol{T}_l}, \delta)$ for the current allocation is stored in the variable $\Delta_{\boldsymbol{T}_l}(\delta)$. At Line (1) it is decided if r or $\hat{\boldsymbol{T}}$ would be augmented. $\Delta_{\boldsymbol{T}}(\delta)$ is either $\Delta_{\hat{\boldsymbol{T}}}(D^{\hat{\boldsymbol{T}}}, \delta)$ or $\Delta_r(x_r, \delta)$, therefore this decision can be made by a simple comparison. If $\hat{\boldsymbol{T}}$ should be augmented, then Procedure TREE-ADD is called recursively on its components. Finally, $\Delta_{\boldsymbol{T}}(\pm\delta)$ is updated at Lines (6)–(7).

B.1 Example of Procedure TREE-ADD

Consider the network of Fig. 6. The cost function is the same on all links and is given in Table I. The current allocation (Fig. 6(a)) is 2 on link AC and 1 on all other links. The three values by each link l are $x_l, -\Delta_l(x_l, 1), -\Delta_{\boldsymbol{T}_l}(1)$. Note that for this initial allocation $\Delta_{\boldsymbol{T}_l}(-1)$ is ∞ for all links except AC. If we call TREE-ADD$(\boldsymbol{x}, 1, \boldsymbol{T})$ then on Line (1) the **if** would fail since $\Delta_r(x_r, 1) = -8 > -12 = \Delta_{\boldsymbol{T}}(1)$. This means $\hat{\boldsymbol{T}}$ (namely, both $\boldsymbol{T}_{AB}$ and $\boldsymbol{T}_{AC}$) would be augmented and the allocation to links AB and AC would increase by 1. Link AB is augmented rather than link BD since the gains for either link is the same, hence the **if** would succeed. Fig. 6(b) shows the new allocation and the new values of $-\Delta_l(x_l, 1), -\Delta_{\boldsymbol{T}_l}(1)$. The next iteration would again augment $\boldsymbol{T}_{AB}$ and $\boldsymbol{T}_{AC}$, however the augmentation to $\boldsymbol{T}_{AB}$ would augment link BD. The result is shown in Fig. 6(c). Note that now link SA becomes the dominant gain, and the next augmentation would add to its allocation. The outcome of such an augmentation is shown in Fig. 6(d).

B.2 The algorithm

Algorithm BALANCE (Fig. 7) is a dynamic programming scheme that solves Problem MOPQ. It starts from any feasible tree partition and performs greedy moves between r and $\hat{\boldsymbol{T}}$. The *while* loop condition at Line (7) computes $\Delta_{r\to\hat{\boldsymbol{T}}}(\boldsymbol{x}, \delta)$. If it is negative then moving δ from r to $\hat{\boldsymbol{T}}$ reduces the overall cost. The augmentation of $\hat{\boldsymbol{T}}$ is done by calling TREE-ADD on each of its components. The *while* loop at Line (11) performs moves from $\hat{\boldsymbol{T}}$ to r in a similar way.

To be able to check the *while* condition and for calling TREE-ADD, we must have $\Delta_{\boldsymbol{T}_l}(\pm\delta)$ for all $l \in \boldsymbol{T}$. This requires an optimal partition on each sub-tree. Algorithm BALANCE makes sure that this is indeed the case by recursively calling itself (Line (5)) on the components of $\hat{\boldsymbol{T}}$. Since any allocation to a leaf is an optimal partition on it, the recursion is stopped once we reach a leaf. After the tree is balanced, the algorithm updates $\Delta_{\boldsymbol{T}}(\pm\delta)$, which is used by the calling iteration.

We proceed to analyze the complexity of Algorithm BALANCE. We first define a *distance* $\varphi_{\boldsymbol{T}}(\boldsymbol{x}_{\boldsymbol{T}})$ which is the tree version of the path distance defined in Section III-B. Let $\varphi_r(\boldsymbol{x}_{\boldsymbol{T}}) \equiv$

TABLE I

THE COST FUNCTIONS FOR ALL LINKS OF FIG. 6

$$c(x) = \begin{cases} \infty & x < 1 \\ 2^{(5-x)} & 1 \leq x \leq 5 \\ 0 & 5 < x \end{cases}$$

x	1	2	3	4	5	6	7
$c(x)$	16	8	4	2	1	0	0
$\Delta(x,-1)$	∞	8	4	2	1	1	0
$\Delta(x,1)$	-8	-4	-2	-1	-1	0	0

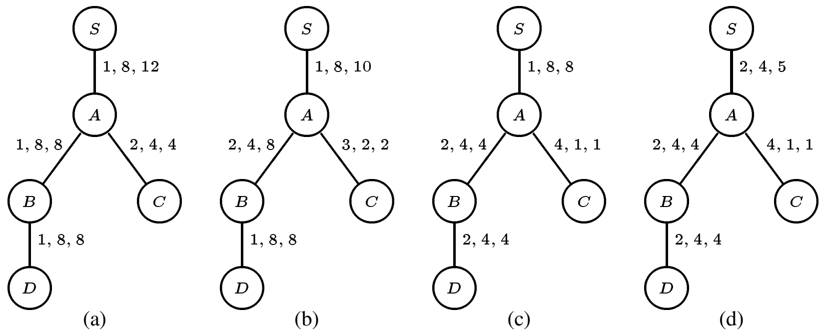


Fig. 6. Example: application of Procedure TREE-ADD
The numbers next to each link l are: $x_l, -\Delta_l(x_l, 1), -\Delta_{\boldsymbol{T}_l}(1)$.

BALANCE $(\boldsymbol{x}, \delta, \boldsymbol{T})$:

if $\boldsymbol{T}$ is a leaf then
    $\Delta_{\boldsymbol{T}}(\delta) \leftarrow \Delta_r(x_r, \delta)$
    $\Delta_{\boldsymbol{T}}(-\delta) \leftarrow \Delta_r(x_r, -\delta)$
    return
(else) for each $l \in N(r)$ do
    BALANCE $(\boldsymbol{x}, \delta, \boldsymbol{T}_l)$
while $\sum_{l \in N(r)} \Delta_{\boldsymbol{T}_l}(\delta) + \Delta_r(x_r, -\delta) < 0$
    $x_r \leftarrow x_r - \delta$
    for each $l \in N(r)$ do
        TREE-ADD $(\boldsymbol{x}, \delta, \boldsymbol{T}_l)$
while $\Delta_r(x_r, \delta) + \sum_{l \in N(r)} \Delta_{\boldsymbol{T}_l}(-\delta) < 0$
    $x_r \leftarrow x_r + \delta$
    for each $l \in N(r)$ do
        TREE-ADD $(\boldsymbol{x}, -\delta, \boldsymbol{T}_l)$
$\Delta_{\boldsymbol{T}}(\delta) \leftarrow \min\{\Delta_r(x_r, \delta), \sum_{l \in N(r)} \Delta_{\boldsymbol{T}_l}(\delta)\}$
$\Delta_{\boldsymbol{T}}(-\delta) \leftarrow \min\{\Delta_r(x_r, -\delta, \sum_{l \in N(r)} \Delta_{\boldsymbol{T}_l}(-\delta)\}$

Fig. 7. Algorithm BALANCE

$|x_r - x^*_r|$, where $\boldsymbol{x}^*_{\boldsymbol{T}}$ is the optimal partition nearest to $\boldsymbol{T}$. Let $\boldsymbol{x}^l_{\boldsymbol{T}_l} = \left\{x^l_e = x^*_e\right\}_{e \in \boldsymbol{T}_l}$. We define the distance of $\boldsymbol{x}_{\boldsymbol{T}}$ from an optimal partition as $\varphi(\boldsymbol{x}_{\boldsymbol{T}}) \equiv \sum_{l \in \boldsymbol{T}} \varphi_r(\boldsymbol{x}^l_{\boldsymbol{T}_l})$.

Theorem 3: Given a feasible partition $\boldsymbol{x}$, Algorithm BALANCE finds a δ-optimal solution to Problem MOPQ in $O(|\boldsymbol{T}|(\varphi(\boldsymbol{x})/\delta) + |\boldsymbol{T}|)$

Proof: $\varphi(\boldsymbol{x})/\delta$ bounds the number of calls to TREE-ADD. In the worst case, Procedure TREE-ADD requires $O(|\boldsymbol{T}|)$ for each call. The recursive calls to Algorithm BALANCE also require $O(|\boldsymbol{T}|)$. ■

Remark 3: We can solve Problem MOPQ by applying Algorithm BALANCE on the feasible partition $\boldsymbol{x}_{\boldsymbol{T}} = \{x_r = D;\ x_l = 0 \quad \forall l \neq r\}$. Clearly, $\varphi(\boldsymbol{x}) \leq D$ in this case, hence Problem MOPQ can be solved in $O(|\boldsymbol{T}|D/\delta)$.

B.3 Example of Algorithm BALANCE

We illustrate Algorithm BALANCE by an example. Fig. 8 shows a simple tree with a source S and two destination nodes C, D. Fig. 8(a) shows an initial (nonoptimal) allocation for an end-to-end requirement of 12. The cost functions for the links are all of the form

$$c_i(x_i) = \frac{\kappa_i}{\kappa_i - 1/x_i}.$$

We assume $\kappa_{SA} = \kappa_{BD} = 1$ and $\kappa_{AB} = \kappa_{AC} = 2$. The values of $c_i(x), \Delta_i(x,-1)$ and $\Delta_i(x,1)$ are detailed in Tables II and III.

Fig. 8 shows the progress of $\boldsymbol{x}$ with the algorithm, where x_l is specified for each link. With the initial allocation (Fig. 8(a)), BD is dominant for the branch AD and $\Delta_{\boldsymbol{T}_{AB}}(\boldsymbol{x}, 1) = \Delta_{BD}(2,1) = -1/2$. The allocation on the branch can be easily verified to be optimal. For link AC we have $\Delta_{AC}(4,1) = -2/63$ and for the link SA we have $\Delta_{SA}(8,-1) = 1/42$. Summing all, we get $1/42 - 1/2 - 2/63 < 0$, hence Procedure TREE-ADD is called to augment the allocation to $\hat{\boldsymbol{T}}$ by 1. The result is shown in Fig. 8(b). Note that the procedure augmented BD and not AB, because $\Delta_{\boldsymbol{T}_{AB}}(\boldsymbol{x}, 1) = \Delta_{\hat{\boldsymbol{T}}_{AB}}(\boldsymbol{x}, 1) = \Delta_{BD}(2,1)$.

With a similar computation, the next call to Procedure TREE-ADD returns the allocation of Fig. 8(c). At this point, BD is no longer dominant on its branch, because $\Delta_{AB}(2,1) = -2/15 < \Delta_{BD}(4,1) = -1/12$. Thus, the next augmentation is to links AB and AC. The result is shown in Fig. 8(d). At this stage BD becomes dominant again on its branch and $\Delta_{\boldsymbol{T}_{AB}}(\boldsymbol{x}, 1) = \Delta_{BD}(4,1) = -1/12$. Although $\Delta_{SA}(5,-1) = 1/12$ too, the algorithm still augments $\hat{\boldsymbol{T}}$ because of the added (negligible) gain from $\Delta_{AC}(7,1) = -2/195$. The final (and optimal) allocation is shown in Fig. 8(e).

C. Distributed and Dynamic Implementation

Algorithm BALANCE can be readily applied in a distributed fashion. Each augmentation in Procedure TREE-ADD is propagated from the root to the leaves. A straightforward implementation requires $O(t)$ time,[7] where t is the depth of the tree. At

[7]assuming that traveling a link requires $O(1)$ time units.

TABLE II

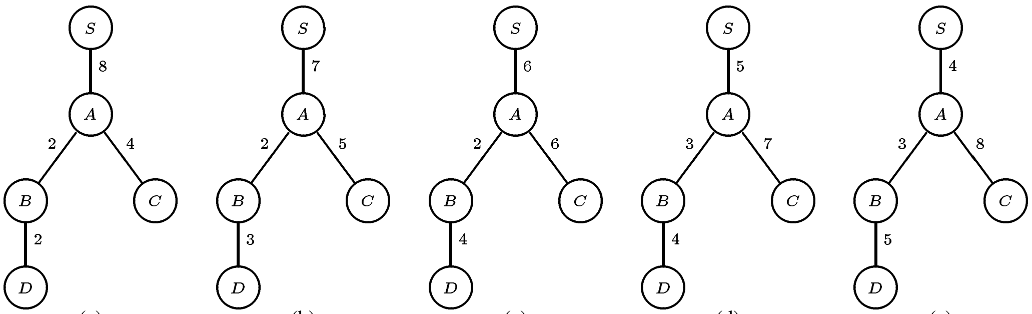
THE COST FUNCTIONS FOR LINKS SA AND BD

x	1	2	3	4	5	6	7	8	9
$c(x)$	∞	2	$3/2$	$4/3$	$5/4$	$6/5$	$7/6$	$8/7$	$9/8$
$\Delta(x,-1)$	∞	∞	$1/2$	$1/6$	$1/12$	$1/20$	$1/30$	$1/42$	$1/56$
$\Delta(x,1)$	$-\infty$	$-1/2$	$-1/6$	$-1/12$	$-1/20$	$-1/30$	$-1/42$	$-1/56$	$-1/72$

TABLE III

THE COST FUNCTIONS FOR LINKS AB AND AC

x	1	2	3	4	5	6	7	8	9
$c(x)$	2	$4/3$	$6/5$	$8/7$	$10/9$	$12/11$	$14/13$	$16/15$	$18/17$
$\Delta(x,-1)$	∞	$2/3$	$2/15$	$2/35$	$2/63$	$2/99$	$2/143$	$2/195$	$2/255$
$\Delta(x,1)$	$-2/3$	$-2/15$	$-2/35$	$-2/63$	$-2/99$	$-2/143$	$-2/195$	$-2/255$	$-2/323$



(a) (b) (c) (d) (e)

Fig. 8. Example: application of Algorithm BALANCE

most $O(t)$ recursive calls to Algorithm BALANCE are performed sequentially. Finally, the number of calls to TREE-ADD *after* the sub-trees are balanced, is bounded by $\varphi_r^{\max}(\boldsymbol{x})/\delta$, where $\varphi_r^{\max}(\boldsymbol{x}) \equiv \max_{l\in\boldsymbol{T}} \varphi_r(\boldsymbol{x}^l_{\boldsymbol{T}_l})$. The overall complexity is, therefore, $O\left(t^2\varphi_r^{\max}(\boldsymbol{x})/\delta\right)$. Note that, for balanced (in the topological sense) trees, $t = O(\log|\boldsymbol{T}|)$.

As is the case for Algorithm GREEDY-MOVE, Algorithm BALANCE can be used as a dynamic scheme that *reacts* to changes in the cost functions *after* an optimal partition is established. The complexity of Algorithm BALANCE is proportional to the distance from the new optimal allocation. Again, small changes (*i.e.*, small $\varphi(\boldsymbol{x})$) incur a small number of computations.

D. Polynomial Solution

Next, we present a polynomial solution. Algorithm BINARY-MOPQ (Fig. 9) uses an approach that is identical to the one used for the solution of Problem OPQ. The algorithm consecutively calls BALANCE for smaller values of δ, until the minimal possible value is reached, at which point an optimal partition is identified.

```
BINARY-MOPQ (D, c(·), T):
   x_r ← D
   x_l ← 0   ∀l ∈ T \ r
   δ ← D/|T|
   repeat
        x ← BALANCE(x, δ, T)
        δ ← ⌈δ/2⌉
   until δ ≤ 1
```

Fig. 9. Algorithm BINARY-MOPQ

We will show that, at each iteration of the algorithm, $\varphi_r^{\max}(\boldsymbol{x})$ is bounded by $t\delta$. Therefore, $\varphi(\boldsymbol{x})/\delta \leq t|\boldsymbol{T}|$ and each iteration requires $O(|\boldsymbol{T}|^2 t)$. Lemma 7 (see appendix for proof) is the multicast-equivalent of Lemma 2.

Lemma 7: Let $\boldsymbol{x} \leftarrow$BALANCE$(\boldsymbol{x}, 2, \boldsymbol{T})$. Then, $\varphi_r(\boldsymbol{x}) < t$.

The same bound applies to all iterations and the number of iterations is $\log(D/|\boldsymbol{T}|)$. Thus, we established the complexity of Algorithm BINARY-MOPQ.

Theorem 4: Algorithm BINARY-MOPQ solves Problem MOPQ in $O(|\boldsymbol{T}|^2 t\log(D/|\boldsymbol{T}|))$.

Comparing this result to the $O(|\boldsymbol{T}|D/\delta)$ complexity of Algorithm BALANCE (see Remark 3), indicates that Algorithm BALANCE is preferable when $D/\delta < |\boldsymbol{T}|t\log(D/|\boldsymbol{T}|)$.

Again, note that, for balanced trees, $t = O(\log|\boldsymbol{T}|)$. Also, we can implement Algorithm BINARY-MOPQ in a distributed fashion (as discussed in Section IV-C), with an overall complexity of $O(t^3\log(D/|\boldsymbol{T}|)$.

Remark 4: Algorithm BINARY-MOPQ starts from a coarse partition and improves the result by refining the partition at each iteration; this means that one may halt the computation once the result is "good enough", albeit not optimal.

E. Heterogeneous multicast groups

In general, the multicast group may be heterogeneous, in the sense that each member $m \in M$, may have a different end-to-end delay requirement D_m. The greedy properties still hold in this case, hence the algorithms presented above may be used to solve this generalized version of Problem MOPQ. However, we need to create an initial feasible allocation and assure that all the various requirements are satisfied.

The complexity of the solution depends on the distance between the initial allocation and the optimal one. An obvious allocation would be to allocate D_m to any member that is a leaf and allocate 0 to all other links. We use a more complex allocation that allows us to bound this distance. Procedure INIT-HETERO (Fig. 10) is used to create the initial allocation. It traverses the tree from the leaves to the root. On each sub-tree $\boldsymbol{T}_l$ it creates a feasible allocation according to the requirements of the multicast group members that belong to the sub-tree. This allocation has the property that the allocation to root link l is the strictest requirement of any member in $\boldsymbol{T}_l$. That is, the allocation to the root link is the *maximal* possible allocation.

INIT-HETERO *(* $\{D_m\}_{m\in \boldsymbol{T}\cap M}, \boldsymbol{T}$ *):*
1 if $\boldsymbol{T}$ is a leaf then
2 $\quad x_r \leftarrow D_m$, where m is the multicast group member at the leaf.
3 $\quad$ **return**
4 (else) for each $l \in N(r)$ do
5 $\quad$ INIT-HETERO($\{D_m\}_{m\in \boldsymbol{T}_l\cap M}, \boldsymbol{T}_l$)
6 $x_r \leftarrow \min_{l\in N(r)} x_l$
7 if $r = (u, m)$, where $m \in M$ then
8 $\quad x_r \leftarrow \min\{x_r, D_m\}$
9 for each $l \in N(r)$ do
10 $\quad x_l \leftarrow x_l - x_r$

Fig. 10. Procedure INIT-HETERO

The procedure starts at the leaves, where such a feasible allocation is simply the end-to-end delay requirement of the multicast group member at that leaf (Line (2)). As the algorithm moves towards the root, sub-trees are combined and their allocation is updated. As much delay as possible is pushed from the sub-trees to the root. The allocation for the root of the combined tree is computed at Lines (6)–(8): first, it cannot be larger than the strictest requirement on the sub-trees (Line (6)); second, if there is a multicast group member on the edge of the root link, then the allocation cannot exceed its end-to-end delay requirement (Line (8)). Line (10) updates the allocation to the sub-trees by reducing it in the amount allocated to the root link.

E.1 Example of Procedure INIT-HETERO

Consider the network of Fig. 11. The demand for each multicast member is specified on its node ("-" indicates that a node is not in the multicast group). Fig. 11(a) shows the initial allocation, where each leaf gets an allocation equal to the demand of its multicast group member. In the next iteration (Fig. 11(b)) the allocation on the subtree is propagated to its root. The final allocation is shown in Fig. 11(c).

E.2 Analysis

We proceed to analyze the complexity. The procedure performs one tree traversal. Each link is considered twice, once when INIT-HETERO is applied to its sub-tree, and once when it is applied to its immediate ancestor. Thus, the overall complexity of Procedure INIT-HETERO is $O(|\boldsymbol{T}|)$.[8]

The heterogeneous version of Problem MOPQ can be solved by first allocating the excess delay on each sub-tree and then solving a homogeneous allocation problem with the rest of the delay. Note that Procedure INIT-HETERO allocates this amount

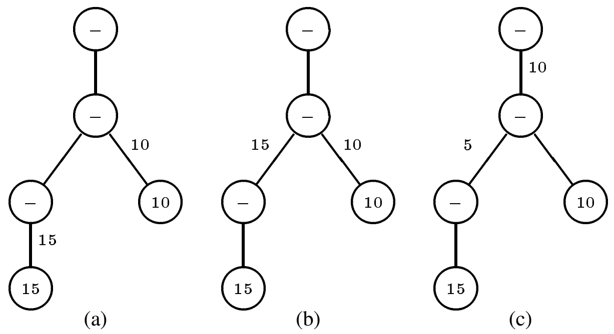


Fig. 11. Example: application of Procedure INIT-HETERO

exactly to the root link. That is, either Algorithm BALANCE or Algorithm BINARY-MOPQ is called on each sub-tree $\boldsymbol{T}_l$ with a delay bound of x_l^h, where x_l^h is the initial allocation returned by Procedure INIT-HETERO.

Lemma 8: The heterogeneous Problem MOPQ can be solved in

$$O\left(|\boldsymbol{T}|^2 (t \log(x^{\max}/|\boldsymbol{T}|) + \varphi(\boldsymbol{x}^h)/x^{\max})\right),$$

where $\boldsymbol{x}^h$ is the initial allocation created by Procedure INIT-HETERO and $x^{\max} = \max_{l\in\boldsymbol{T}} x_l^h$.

Proof: We use Algorithm BINARY-MOPQ, with the initial δ set to $x^{\max}/|\boldsymbol{T}|$. Without taking into account the creation of the initial allocation and the call to Algorithm BALANCE on it, the complexity would have been $O(|\boldsymbol{T}|^2(t\log(x^{\max}/|\boldsymbol{T}|)$. Creating the initial allocation requires $O(|\boldsymbol{T}|)$. By inserting $\delta = x^{\max}/|\boldsymbol{T}|$ in the complexity expression for Algorithm BALANCE, we get $O(|\boldsymbol{T}|^2\varphi(\boldsymbol{x}^h)/x^{\max})$. Thus, the overall complexity is $O\left(|\boldsymbol{T}|^2 \left(t \log(x^{\max}/|\boldsymbol{T}|) + \varphi(\boldsymbol{x}^h)/x^{\max}\right)\right)$. ■

We proceed to analyze $\varphi(\boldsymbol{x}^h)$. Since x_l^h is the maximal possible allocation for the sub-tree $\boldsymbol{T}_l$, any change in the allocation can only be made from l to $\hat{\boldsymbol{T}}_l$. Hence, we must have

$$\varphi_r(\boldsymbol{x}^h_{\boldsymbol{T}_l}) \le x_l^h, \tag{2}$$

and, from its definition, $\varphi(\boldsymbol{x}^h) \le \sum_{l\in\boldsymbol{T}} x_l^h$. Let I denote the *inner-tree* multicast group members, and let L denote the *leaf* members. Note that $|I| + |L| = |M|$ and that L includes *all* the leaves of the tree. After the update at Line (10) of INIT-HETERO, either $x_l = 0$ for at least one link $l \in N(r)$ or $r = (u, m)$, where $m \in I$. Thus, the number of links for which $x_l^h > 0$ is bounded by $\sum_{l\in\boldsymbol{T}}(N(l) - 1) + |I| = |L| + |I| = |M|$, which implies $\varphi(\boldsymbol{x}^h) \le x^{\max}|M|$. Combining this result with Lemma 8 we get an overall complexity of $O\left(|\boldsymbol{T}|^2 \left(t \log(x^{\max}/|\boldsymbol{T}|) + |M|\right)\right)$.

We can improve upon this result in the following way. First, we rewrite the complexity of Algorithm BALANCE as $O(|\boldsymbol{T}| + \sum_{l\in\boldsymbol{T}} \varphi_r(\boldsymbol{x}^l_{\boldsymbol{T}_l})|\boldsymbol{T}_l|/\delta)$. Using (2), the complexity of calling BALANCE on the initial allocation $\boldsymbol{x}^h$ is $O\left(|\boldsymbol{T}| + \sum_{l\in\boldsymbol{T}} x_l^h|\boldsymbol{T}_l|/\delta\right)$. Next, we observe that, for *any* feasible allocation $\boldsymbol{x}_{\boldsymbol{T}}(D)$, we have $\sum_{l\in\boldsymbol{T}} x_l|\boldsymbol{T}_l| \le D|\boldsymbol{T}|$.[9] In our case $D \le tx^{\max}$, hence creating the initial allocation and calling BALANCE on it requires

$$O\left(|\boldsymbol{T}|(2 + tx^{\max}/\delta)\right) = O\left(|\boldsymbol{T}|(t|\boldsymbol{T}|)\right).$$

[8] A distributed implementation requires $O(t)$.

[9] this can be proved by induction on the depth of the tree.

Thus, we established the following theorem.

Theorem 5: The heterogeneous Problem MOPQ can be solved in $O(|\boldsymbol{T}|^2 t \log(x^{\max}/|\boldsymbol{T}|))$.

V. Routing Issues

In the previous sections we addressed and solved optimal QoS partition problems for *given* topologies. These solutions have an obvious impact on the route selection process, as the quality of a route is determined by the cost of the eventual (optimal) QoS partition over it. Hence, the unicast partition problem OPQ induces a unicast routing problem, OPQ-R, which seeks a path on which the cost of the solution to Problem OPQ is minimal. Similarly, Problem MOPQ induces a multicast routing problem MOPQ-R. In this section we briefly discuss some current and future work in the context of these routing problems.

A. OPQ-R

As was the case with Problem OPQ, with OPQ-R too there is a significant difference between bottleneck QoS requirements and additive ones. As explained in Section II, for a bottleneck QoS requirement, the end-to-end requirement determines $Q_l = Q$ for all links in the path (or tree), and a corresponding link cost, $c_l(Q)$. Therefore, the routing problem OPQ-R boils down to a "standard" shortest-path problem with link length $c_l(Q)$.

As noted, in the context of Problem OPQ, providing delay requirements through rate-based schedulers [8], [22], [25], [26], translates the additive requirement into a (simpler) bottleneck requirement. However, in the context of Problem OPQ-R, such a translation is not possible anymore, since paths differ also in terms of constant (rate-independent) link delay components. Efficient solutions for Problem OPQ-R, under delay requirements and rate-based schedulers, have been presented in [9].

The general OPQ-R problem, under additive QoS requirements, is much more complex, and has been found to be intractable [16]. Note that, even in a simpler, *constant-cost* framework, where each link is characterized by a delay-cost pair (rather than a complete delay-cost function), routing is an intractable problem [7]. In the latter case, optimal routing can be achieved through a pseudo-polynomial dynamic-programming scheme, while ϵ-optimal solutions can be achieved in polynomial time [10].

The general OPQ-R problem, under the framework of the present study, was solved in [16]. The solution is based on dynamic-programming and assumes that the link cost functions are convex. An exact pseudo-polynomial solution, as well as an ϵ-optimal polynomial solution, have been presented. We note that a single execution of those algorithms finds a unicast route from the source to *every* destination and for *every* end-to-end delay requirement (up to a given bound).

B. MOPQ-R

As could be expected, finding optimal multicast trees under our framework is much more difficult than finding unicast paths. Even with bottleneck QoS requirements, MOPQ-R boils down to finding a Steiner tree, which is known to be an intractable problem [7].

We are currently working on solving MOPQ-R for additive QoS requirements. We established an efficient scheme for the fundamental problem of adding a new member to an existing multicast tree. This provides a useful building block for constructing multicast trees. Another important building block is the optimal sub-structure property (established in Section IV), which is an essential requirement for the application of greedy and dynamic programming solution approaches.

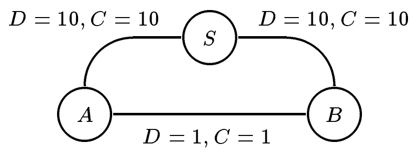


Fig. 12. Example: extending a multicast tree

Interestingly, the above problem, of adding members to multicast trees, may serve to illustrate the power of our framework over the simpler, *constant-cost* framework. In the latter, there is a single delay-cost pair for each link (rather than a complete delay-cost function), and the goal is to find a minimal cost tree that satisfies an end-to-end delay constraint.[10] Under that framework, it is often impossible to connect a new member to the "tree top", *i.e.*, the leaves and their neighborhood. This is a consequence of cost minimization considerations, which usually result with the consumption of all (or most of) the available delay at the leaves. For example, consider the network of Fig. 12. The source is S and the multicast group is $\{A, B\}$; the end-to-end delay bound is 10 and the link delay-cost pairs are specified. Suppose we start with a tree for node A, *i.e.*, the link (S, A). Since A exhausts all the available end-to-end delay, we cannot add B to the tree by extending A's branch with the (cheap) link (A, B); rather, we have to use the (expensive) link (S, B). Note that we would get the same result even if there were an additional link from S to A with shorter delay, say 9, and slightly higher cost, say 11.

Our framework allows a better solution, as it lets the link (S, A) advertise several delays and costs. For instance, it could advertise a delay of 10 with a cost of 10 and a delay of 9 with a cost of 11. When adding B to the tree, we can change the *delay allocation* on (S, A) from 10 to 9 (thus paying 11 instead of 10), which allows us to use the link (A, B) for adding B. The cost of the resulting tree is 12, as opposed to 20 in the previous solution (*i.e.*, using link (S, B)). Note that, when adding B, one can consider the "residual" cost for each link, *i.e.*, the cost of tightening the delay bound on existing allocations. In our example, the residual cost function of link (S, A) is 0 for a delay of 10 (*i.e.*, the current allocation) and 1 for a delay of 9 (*i.e.*, the added cost for tightening the requirement). The last observation implies that adding a new member to an existing tree boils down to finding an optimal unicast path, *with respect to the residual cost functions*, from the new member to the source; *i.e.*, an instance of Problem OPQ-R, for which efficient solutions have been established in [16].

VI. Conclusions

We investigated a framework for allocating QoS resources on unicast paths and multicast trees, which is based on partitioning

[10] That framework was the subject of numerous studies on constrained multicast trees.

QoS requirements among the network components. The quality of a partition is quantified by link cost functions, which increase with the severity of the QoS requirement. We indicated that this framework is consistent with the major proposals for provisioning QoS on networks. Indeed, the problem of how to efficiently partition QoS requirements among path or tree links has been considered in previous studies, however till now only heuristic approaches have been addressed. The present study is the first to provide a general optimal solution, both for unicast paths and multicast trees.

We demonstrated how the various classes of QoS requirements can be accommodated within our framework. We showed that the partitioning problems are simple when dealing with bottleneck requirements, such as bandwidth, however they become intractable for additive (or multiplicative) requirements, such as delay, jitter and loss rate. Yet we established that, by introducing a mild assumption of weak convexity on the cost functions, efficient solutions can be derived.

We note that weak convexity essentially means that, as the QoS requirement weakens, the rate of decrease of the cost function diminishes. This is a reasonable property, as cost functions are lower-bounded, e.g. by zero. Moreover, it indeed makes sense for a cost function to strongly discourage severe QoS requirements, yet gradually become indifferent to weak (and, eventually, practically null) requirements. Hence, the scope of our solutions is broad and general.

Specifically, we presented several greedy algorithms for the unicast problem (OPQ). Algorithm GREEDY-MOVE is a pseudo-polynomial solution, which can be efficiently implemented in a distributed fashion. The complexity of this solution is $O(\varphi(\boldsymbol{x})\log|\boldsymbol{p}|)$, where $\varphi(\boldsymbol{x}) \leq D$ is the distance between the initial allocation, $\boldsymbol{x}$, and the optimal one. It can also be applied as an incremental scheme to modify an existing allocation. This is useful in dynamic environments where the cost of resources changes from time to time. As the complexity is proportional to $\varphi(\boldsymbol{x})$, small cost changes require only a small number of computations to regain optimality. Algorithm BINARY-OPQ is a polynomial solution of complexity $O\left(|\boldsymbol{p}|\log|\boldsymbol{p}|\log(D/|\boldsymbol{p}|)\right)$, which served as the basis for our solution to the multicast problem MOPQ.

Next, we addressed the multicast problem MOPQ. We began by showing that the fundamental properties of convexity and optimal sub-structure generalize to multicast trees. Then, we established that Problem MOPQ also bears a greedy structure, although much more complex than its OPQ counterpart. Again, the greedy structure, together with the other established properties, provided the foundations for efficient solutions. Algorithm BALANCE is a pseudo-polynomial algorithm which can be applied as a dynamic (incremental) scheme. Its complexity is $O(|\boldsymbol{T}|\varphi(\boldsymbol{x}) + |\boldsymbol{T}|)$, where $\varphi(\boldsymbol{x})$ is, again, the distance between the initial allocation and the optimal one. A distributed implementation requires $O(t^2\varphi_r^{\max}(\boldsymbol{x}))$, where t is the depth of the tree and $\varphi_r^{\max}(\boldsymbol{x})$ is the maximal distance of the allocation of any *link* from its optimal value. Note that, for balanced trees, $t = O(\log|\boldsymbol{T}|)$. Algorithm BINARY-MOPQ is a polynomial solution with a complexity of $O(|\boldsymbol{T}|^2 t\log(D/|\boldsymbol{T}|))$. A distributed implementation of this algorithm requires $O(t^3\log(D/|\boldsymbol{T}|)$. We established that our solutions are applicable, with the same complexity, to the heterogeneous case, where multicast members may have different delay requirements.

Lastly, we discussed the related routing problems, OPQ-R and MOPQ-R. Here, the goal is to select either a unicast path or multicast tree, so that, after the QoS requirements are optimally partitioned over it, the resulting cost would be minimized. Again, unicast proves to be much easier than multicast. In particular, for bottleneck QoS requirements, OPQ-R boils down to a simple shortest-path problem. For additive requirements, OPQ-R is intractable, yet an efficient, ϵ-optimal solution has been established in [16]. For multicast, all the various versions of MOPQ-R are intractable. We are currently investigating Problem MOPQ-R under additive requirements, and have obtained an efficient scheme for adding new members to a multicast tree.

Several important issues are left for future work. One is multicast routing, *i.e.*, Problem MOPQ-R, for which just initial (yet encouraging) results have been obtained thus far. Another interesting direction is the investigation of similar resource allocation problems for multiple unicast connections. In this case resources for several connections are allocated on each link and the cost becomes a multivariable function. As can be expected, this problem is much more complex. Another important aspect is the actual implementation of our solutions within practical network architectures. In this respect, it is important to note that a compromise with optimality might be called for. Indeed, while our solutions are of reasonable complexity, a sub-optimal solution that runs substantially faster might be preferable in practice. Relatedly, one should consider the impact of the chosen solution for QoS partitioning on the routing process. The latter has to consider the quality of a selection (*i.e.*, path or tree) in terms of the eventual QoS partition. This means that simpler partitions should result in simpler routing decisions, which provides further motivation for compromising optimality for the sake of simplicity. The optimal solutions established in this study provide the required starting point in the search of such compromises.

Lastly, we believe that the framework investigated in this study, where QoS provisioning is characterized through cost functions, provides a powerful paradigm for dealing with QoS networking. We illustrated the potential benefits through an example of dynamic tree maintenance. Future study should further consider the implications and potential advantages of our framework, when applied to the various problems and facets of QoS networking.

APPENDIX

In the following we present some proofs that were omitted from the main text.

Lemma 4: If $\{c_l\}_{l\in\boldsymbol{T}}$ are convex, and $\boldsymbol{T}$ has a depth of 2, then $c_{\boldsymbol{T}}(d)$ is convex.

Proof: In any optimal partition $\boldsymbol{x}^*_{\boldsymbol{T}}$, we must have for all $l \in N(r)$, $x_l^* = D^{\hat{\boldsymbol{T}}} = D - x_r^*$. Hence, the cost associated with the allocation to $\hat{\boldsymbol{T}}$ is $c_{\hat{\boldsymbol{T}}}\left(D^{\hat{\boldsymbol{T}}}\right) = \sum_{l\in N(r)} c_l\left(D^{\hat{\boldsymbol{T}}}\right)$. This is a sum of convex functions, therefore it is convex too. Thus, $\hat{\boldsymbol{T}}$ maybe be replaced by an equivalent link with a convex cost function, that is, $\boldsymbol{x}^{**} = \left\{x_r, D^{\hat{\boldsymbol{T}}}\right\}$ is an optimal partition on a path of length 2. We can employ the results of Section III on this path. Thus, the greedy properties hold and we have

$$\begin{aligned} \Delta_{\boldsymbol{T}}(D,-1) &= \min\left\{\Delta_r(x_r,-1), \Delta_{\hat{\boldsymbol{T}}}(D^{\hat{\boldsymbol{T}}},-1)\right\}, \\ -\Delta_{\boldsymbol{T}}(D,+1) &= \max\left\{-\Delta_r(x_r,+1), -\Delta_{\hat{\boldsymbol{T}}}(D^{\hat{\boldsymbol{T}}},+1)\right\}. \end{aligned}$$

Using Lemma 3, we get that $\Delta_{\boldsymbol{T}}(D,-1) \geq -\Delta_{\boldsymbol{T}}(D,+1) = \Delta_{\boldsymbol{T}}(D+1,-1)$. This applies to any choice of D, hence $\Delta_{\boldsymbol{T}}(d,-1)$ is nonincreasing, which implies that $c_{\boldsymbol{T}}(d)$ is convex. ∎

Lemma 5: If $\{c_l\}_{l\in\boldsymbol{T}}$ are convex then so is $c_{\boldsymbol{T}}(d)$.

Proof: By induction on the depth of the tree, t. Lemma 4 is the basis of the induction.

Suppose Lemma 5 is true for trees of depth less than t. If $\boldsymbol{T}$ has depth t then for each $l \in N(r)$, the depth of the sub-tree $\boldsymbol{T}_l$ is no greater than $t-1$. By the inductive assumption, each such sub-tree can be collapsed to a single convex link. We get a depth-1 tree, which, by Lemma 4, is equivalent to a single convex link. ∎

Lemma 7: Let $\boldsymbol{x} \leftarrow$BALANCE$(\boldsymbol{x}, 2, \boldsymbol{T})$. Then, $\varphi_r(\boldsymbol{x}) < t$.

The proof of Lemma 7 relies on the following discussion. Let $\Delta_{\boldsymbol{T}}(d,\delta)$ denote the value of $\Delta_{\boldsymbol{T}}(\delta)$ at the termination of BALANCE$(\boldsymbol{x}(d),\delta,\boldsymbol{T})$. Note that $\Delta_{\boldsymbol{T}}(d,\delta)$ assumes a δ-optimal partition on the tree, hence it is different from $(c_{\boldsymbol{T}}(d+\delta) - c_{\boldsymbol{T}}(d+\delta))/\delta$, which assumes the optimal partition. We use the following two lemmas in our proof. Lemma A.1 relates a nonoptimal allocation to an optimal one. Lemma A.2 is the crux of the proof.

Lemma A.1: $\min\{\Delta_r(A,\delta), \Delta_{\hat{\boldsymbol{T}}}(B,\delta)\} \leq \Delta_{\boldsymbol{T}}(A+B,\delta)$.

Proof: Let $\boldsymbol{x}^*_{\boldsymbol{T}} = (x^*_r, x^*_{\hat{\boldsymbol{T}}})$ be the optimal allocation for a delay requirement of $D = A+B$ on $\boldsymbol{T}$. Following the discussion of Section IV-A this is equivalent to an optimal allocation for a two-link path $(r,\hat{\boldsymbol{T}})$. Let $\Delta^* = \min\{\Delta_r(x^*_r,\delta), \Delta_{\hat{\boldsymbol{T}}}(x^*_{\hat{\boldsymbol{T}}},\delta)\}$. If $A \leq x^*_r$ then $B \geq x^*_{\hat{\boldsymbol{T}}}$ and vice versa. We can generalize (1) (Section III-E) to establish that $\min\{\Delta_r(A,\delta), \Delta_{\hat{\boldsymbol{T}}}(B,\delta)\} \leq \Delta^*$. The result follows since $\Delta_{\boldsymbol{T}}(A+B,\delta) = \Delta^*$. ∎

Lemma A.2: $\Delta_{\boldsymbol{T}}(D,2\delta) \leq \Delta_{\boldsymbol{T}}(D+t\delta,\delta)$.

Proof: The proof is by induction on the depth of the tree t.

Base: for $t=1$, $\boldsymbol{T}$ is a link and the lemma becomes $\Delta_l(D,\delta) \leq \Delta_l(D_l+\delta,\delta)$, which follows from the convexity of c_l.

Step: suppose the lemma holds for trees with depth less than t. Algorithm BALANCE terminates with

$$\Delta_{\boldsymbol{T}}(\delta) \leftarrow \min\{\Delta_r(x_r,\delta), \sum_{l\in N(r)} \Delta_{\boldsymbol{T}_l}(\delta)\}.$$

Each $\boldsymbol{T}_l$ has depth $t-1$. Applying the inductive assumption on r and on each $\boldsymbol{T}_l$, we get

$$\begin{aligned} &\Delta_{\boldsymbol{T}}(D,2\delta) \\ \leq\ &\min\left\{\Delta_r(x_r+\delta,\delta), \sum_{l\in N(r)} \Delta_{\boldsymbol{T}_l}(D^{\hat{\boldsymbol{T}}}+(t-1)\delta,\delta)\right\} \\ =\ &\min\left\{\Delta_r(x_r+\delta,\delta), \Delta_{\hat{\boldsymbol{T}}}(D^{\hat{\boldsymbol{T}}}+(t-1)\delta,\delta)\right\}. \end{aligned}$$

Using Lemma A.1 we get

$$\begin{aligned} \Delta_{\boldsymbol{T}}(D,2\delta) &\leq \Delta_{\boldsymbol{T}}((x_r+\delta)+(D^{\hat{\boldsymbol{T}}}+(t-1)\delta),\delta) \\ &= \Delta_{\boldsymbol{T}}(D+t\delta,\delta). \end{aligned}$$

∎

We can now prove Lemma 7.

Proof: From the the convexity of c_r , we have for $t \geq 2$,

$$\Delta_r(x_r-(t-1)\delta,-\delta) \geq \Delta_r(x_r,-2\delta);$$

from the 2δ-optimality of $\boldsymbol{x}$, $\Delta_r(x_r,-2\delta) \geq -\Delta_{\hat{\boldsymbol{T}}}(D^{\hat{\boldsymbol{T}}},2\delta)$. Applying Lemma A.2 on $\hat{\boldsymbol{T}}$, we have

$$\begin{aligned} -\Delta_{\hat{\boldsymbol{T}}}(D^{\hat{\boldsymbol{T}}},2\delta) &= -\sum_{l\in N(r)} \Delta_{\boldsymbol{T}_l}(D^{\hat{\boldsymbol{T}}},2\delta) \\ &\geq -\sum_{l\in N(r)} \Delta_{\boldsymbol{T}_l}(D^{\hat{\boldsymbol{T}}}+(t-1)\delta,\delta) \\ &= -\Delta_{\hat{\boldsymbol{T}}}(D^{\hat{\boldsymbol{T}}}+(t-1)\delta,\delta). \end{aligned}$$

Combining all, we get

$$\Delta_r(x_r-(t-1)\delta,-\delta) \geq -\Delta_{\hat{\boldsymbol{T}}}(D^{\hat{\boldsymbol{T}}}+(t-1)\delta,\delta).$$

Thus, no more than $(t-1)$ greedy δ-moves can be made from r to $\hat{\boldsymbol{T}}$. This applies for $\delta = \pm 1$, hence the allocation to r is not changed by more than $t-1$. This implies $\varphi_r(\boldsymbol{x}) < t$, as required. ∎

REFERENCES

[1] G. Apostolopoulos, Roch Guérin, S. Kamat, Ariel Orda, T. Przygienda, and D. Williams. QoS routing mechanisms and OSPF extensions. Internet RFC 2676, August 1999.

[2] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

[3] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick. A framework for QoS-based routing in the internet. Internet RFC 2386, August 1998.

[4] R. Braden (Ed.), L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource reSerVation Protocol (RSVP) version 1, functional specification. Internet RFC 2205, September 1997.

[5] V. Firoiu and D. Towsley. Call admission and resource reservation for multicast sessions. In *Proceedings of the IEEE INFOCOM'96*, pages 776–785, San Francisco, CA, April 1996.

[6] G. N. Frederickson and D. B. Johnson. The complexity of selection and ranking in $X+Y$ and matrices with sorted columns. *Journal of Computer and System Sciences*, 24(2), 1982.

[7] M.R. Garey and D.S. Johnson. *Computers and Intractability*. Freeman, San Francisco, 1979.

[8] L. Georgiadis, Roch Guérin, V. Peris, and K. N. Sivarajan. Efficient network QoS provisioning based on per node traffic shaping. *IEEE/ACM Transactions on Networking*, 4(4):482–501, August 1996.

[9] Roch Guérin and Ariel Orda. QoS-based routing in networks with inaccurate information: Theory and algorithms. *IEEE/ACM Transactions on Networking*, 7(3):350–364, June 1999.

[10] Refael Hassin. Approximation schemes for the restricted shortest path problem. *Mathematics of Operations Research*, 17(1):36–42, February 1992.

[11] D. S. Hochbaum. Lower and upper bounds for the alloction problem and other nonlinear optimization problems. *Mathematics of Operations Research*, 19(2):390–409, 1994.

[12] T. Ibaraki and N. Katoh. *Resource Allocation Problems*. the foundations of computing. MIT Press, Cambridge, MA, 1988.

[13] V. P. Kompella, J. C. Pasquale, and G. C. Polyzos. Multicast routing for multimedia communication. *IEEE/ACM Transactions on Networking*, 1(3):286–292, 1993.

[14] Y. A. Korilis, T. A. Varvarigou, and S. R. Ahuja. Incentive-compatible pricing strategies in noncooperative networks. In *Proceedings of the IEEE INFOCOM'98*, San Francisco, CA, April 1998.

[15] Dean H. Lorenz and Ariel Orda. Optimal partition of QoS requirements on unicast paths and multicast trees. Research Report EE Pub. No. 1167, Department of Electrical Engineering, Technion, Haifa, Israel, July 1998. Available ftp: ftp://ftp.technion.ac.il/pub/supported/ee/Network/lor.mopq98.ps.

[16] Dean H. Lorenz and Ariel Orda. QoS routing in networks with uncertain parameters. *IEEE/ACM Transactions on Networking*, 6(6):768–778, December 1998.

[17] Steven H. Low and P. P. Varaiya. A new approach to service provisioning in ATM networks. *IEEE/ACM Transactions on Networking*, 1(3):547–553, 1993.

[18] J. K. MacKie-Mason and H. R. Varian. Pricing congestable network resources. *IEEE Journal on Selected Areas in Communications*, 13(7):1141–1149, September 1995.

[19] R. Nagarajan, J.F. Kurose, and D. Towsley. Allocation of local quality of service constraints to meet end-to-end requirements. In *IFIP Workshop on the Performance Analysis of ATM Systems*, Martinique, January 1993.

[20] Ariel Orda and N. Shimkin. Incentive pricing in multi-class communication networks. *Telecommunication Systems*, 13(2/4):241–267, July 2000.

[21] Private network-network interface specification v1.0 (PNNI). ATM Forum Technical Committee, March 1996.

[22] A. K. Parekh and R.G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the multiple node case. *IEEE/ACM Transactions on Networking*, 2:137–150, 1994.

[23] S. Shenker, C. Partridge, and Roch Guérin. Specification of guaranteed quality of service. Internet RFC 2212, September 1997.

[24] Z. Wang and J. Crowcroft. Quality-of-service routing for supporting multimedia applications. *IEEE JSAC*, 14(7):1288–1234, September 1996.

[25] H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks. *Proceedings of the IEEE*, 83(10):1374–1399, October 1995.

[26] H. Zhang and D. Ferrari. Rate-controlled service disciplines. *Journal of High Speed Networks*, 3:389–412, 1994.

Dean H. Lorenz (S '98) received his B.Sc. (summa cum laude) in computer engineering from the Technion – Israel Institute of Technology, Haifa, Israel, in 1995. He is a Ph.D. student and teaching assistant at the Department of Electrical Engineering at the Technion.

During the summer of 1998 he was at the Mathematics of Networks Department at Lucent Bell Laboratories, Murray Hill, NJ. During the summer of 1999 he was a DIMACS visitor at the Department of Network and Services Management, Lucent Bell-Labs, Holmdel, NJ. His research interests are QoS routing and multicasting. Dean received the Gutwirth Award for outstanding distinction in 1998 and 1999.

Ariel Orda (S'94-M'92-SM'97) received the B.Sc. (summa cum laude), M.Sc., and D.Sc. degrees in Electrical Engineering from the Technion -Israel Institute of Technology, Haifa, Israel, in 1983, 1985, and 1991, respectively.

Since 1994, he has been with the Department of Electrical Engineering at the Technion, where he is currently an Associate Professor and the Academic Head of the Computer Networking Laboratory. In 1993–1994, he was with the Center for Telecommunication Research, Columbia University, New York, NY, as a Visiting Scientist, and with AT&T Bell Laboratories, Murray Hill, NJ, as a Scientific Consultant. During the summers of 1992 and 1995–1999, he held visiting positions at AT&T Bell Laboratories, Murray Hill, NJ, IBM Watson, Hawthorne, NY, and Bell Laboratories - Lucent Technologies, Holmdel, NJ. Since 1991 he has held several consulting positions with the Israeli industry.

His current research interests include QoS routing, the application of game theory to computer networking, network pricing, and distributed network algorithms.

Dr. Orda received the Award of the Chief Scientist in the Ministry of Communication in Israel, a Gutwirth Award for outstanding distinction, the Research Award of the Association of Computer and Electronic Industries in Israel, and the Jacknow Award for Excellence in Teaching. Dr. Orda is an Associate Editor of the IEEE/ACM Transactions on Networking, and is the Program Co-Chair of IEEE Infocom'2002.