Optimal Partition of QoS Requirements on Unicast Paths and Multicast Trees

Dean H. Lorenz and Ariel Orda
Department of Electrical Engineering*
Technion—Israel Institute of Technology

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, QoS-dependent costs, Multicast, Routing, Broadband networks.**

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., [2], [8], [9], [21], [24] and references therein. Mechanisms for providing various QoS guarantees have been also widely investigated, e.g. [7], [22]. Although there are proposals for resource reservation, most notably RSVP [3], 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 either considered 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 [4], [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

*email.{deanh@tx,ariel@ee}.technion.ac.il

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 QoS requirements (such as delay), the development of *optimal* QoS partition schemes is of interest. [4] 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, [4] 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. [4] 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 studied 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 constraint 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 aspect, such as distributed 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 QoS 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 applies these findings to unicast and multicast QoS routing. Finally, concluding remarks are presented in Section VI. Due to space limits, many 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 that the connection topology is given, *i.e.*, a path $\mathbf{p}$ for unicast, or a tree $\mathbf{T}$ 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 $\mathbf{p}$, is a vector $\mathbf{x_p} = \{x_l\}_{l \in \mathbf{p}}$ 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 $l \in \mathbf{p}$,[1] because Q is determined by the bottleneck link, *i.e.*, $Q = \min_{l \in \mathbf{p}} x_l$. Since allocating more than Q induces a higher cost, yet does not improve the overall QoS, the optimal partition is $\mathbf{x}^*_\mathbf{p} = \{Q\}_{l \in \mathbf{p}}$. For *additive* QoS requirements, a feasible partition, $\mathbf{x_p}$, must satisfy $\sum_{l \in \mathbf{p}} x_l \leq Q$. 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.*, $Q = \prod_{l \in \mathbf{p}} (x_l)$. 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 $(-\log Q) = \sum_{l \in \mathbf{p}} (-\log x_l)$, *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 [7], [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 $c_l(x_l)$, and 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 non-increasing function, whereas when x_l stands for bandwidth, $c_l(x_l)$ is non-decreasing. The overall cost of a partition is the sum of the local costs, *i.e.*, $c(\mathbf{x_p}) = \sum_{l \in \mathbf{p}} c_l(x_l)$.

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. Also, it may be used for

[1] This is also true for a multicast tree $\mathbf{T}$.

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

network management to discourage the use 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 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 $\prod_{l\in\mathbf{p}} f_l(x_l)$. Since we have $\log\left(\prod_{l\in\mathbf{p}} f_l(x_l)\right) = \sum_{l\in\mathbf{p}} \log f_l(x_l)$, 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 $\mathbf{p}$ and an end-to-end QoS requirement Q, find a QoS partition $\mathbf{x}^*_{\mathbf{p}} = \{x^*_l\}_{l\in\mathbf{p}}$, such that $c\left(\mathbf{x}^*_{\mathbf{p}}\right) \le c\left(\mathbf{x}'_{\mathbf{p}}\right)$, for any (other) QoS partition $\mathbf{x}'_{\mathbf{p}}$.

This study focuses on the solution of Problem OPQ for additive QoS parameters, which, as mentioned, is considerably more complex than its 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.

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 specific case of the resource allocation problem. The fastest solution to this problem, [5], requires $O(|\mathbf{p}|\log D/|\mathbf{p}|)$.[3] In Section III-B we present a *greedy* pseudo-polynomial solution. This solution 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 [5], 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. We further assume that all parameters are integers, and that the link cost functions are non-increasing with the delay and (weakly) convex.

A. Notations

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

The average δ-*increment* gain for a link l is denoted by $\Delta_l(x,\delta) \equiv (c_l(x+\delta) - c_l(x))/\delta$. The average δ-*move* gain is denoted by $\Delta_{e\to l}(\mathbf{x},\delta) \equiv \Delta_l(x_l+\delta) + \Delta_e(x_e-\delta)$.

B. Pseudo-polynomial solution

Problem OPQ is a special case of the general *resource allocation* problem which has been extensively investigated [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 $(D,\delta,c(\cdot),\mathbf{p})$:
$\mathbf{x} \leftarrow \{0\}_{l\in\mathbf{p}}$
while $|\mathbf{x}| < D$ do
    $e \leftarrow arg\min_{l\in\mathbf{p}} \Delta_l(x_l,\delta)$
    $x_e \leftarrow x_e + \delta$
return $\mathbf{x}$

Fig. 1. Algorithm Greedy-Add

Algorithm Greedy-Add (Figure 1) employs such a greedy approach. It starts from the zero allocation and adds the delay bit-by-bit, 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 $O(\log|\mathbf{p}|)$, which leads to an overall complexity of $O(\frac{D}{\delta}\log|\mathbf{p}|)$. In [11] it is shown that the solution is δ-optimal in the following sense: if $\mathbf{x}^\delta$ is the output of the algorithm and $\mathbf{x}^*$ is the optimal solution, then $|\mathbf{x}^* - \mathbf{x}^\delta| \le |\mathbf{p}|\delta$.[4]

Algorithm Greedy-Move (Figure 2) 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 partition. Each iteration performs a greedy move, namely the move with minimal (negative) δ-move gain.

Greedy-Move $(\mathbf{x},\delta,c(\cdot),\mathbf{p})$:
1 loop
2     $e,l \leftarrow arg\min_{e,l\in\mathbf{p}} \Delta_{e\to l}(\mathbf{x},\delta)$
3   if $\Delta_{e\to l}(\mathbf{x},\delta) \ge 0$ **return** $\mathbf{x}$
4     $x_e \leftarrow x_e - \delta$
5     $x_l \leftarrow x_l + \delta$

Fig. 2. Algorithm Greedy-Move

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

Lemma 1: Each iteration of Algorithm Greedy-Move decreases $\varphi(\mathbf{x})$ by at least δ, unless $\mathbf{x}$ 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 $\varphi(\mathbf{x})$. Theorem 1 summarizes this discussion.

[3] $|\mathbf{p}|\log(D/|\mathbf{p}|)$ is also a lower bound for solving Problem OPQ [11], that is no *fully* polynomial algorithms exist.

[4] This also established the error due to our assumption of integer parameters.

Theorem 1: Algorithm GREEDY-MOVE solves Problem OPQ in $O(\frac{\varphi(\mathbf{x})}{\delta}\log|\mathbf{p}|)$.

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

C. Distributed and dynamic implementation

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 e, l of Line 2 are identified and the allocation change of the *previous* iteration is performed. This requires $O(|\mathbf{p}|\varphi/\delta)$ end-to-end messages. Such a distributed implementation also exempts us from having to advertise the updated link cost functions.

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 a small number of computations.

D. Polynomial solution

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

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

Fig. 3. 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 $O(\log(D/|\mathbf{p}|))$. 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 $D/\delta = |\mathbf{p}|$) 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 $\mathbf{x} \leftarrow$GREEDY-MOVE$(\mathbf{x}, 2, c(\cdot), \mathbf{p})$. Then $\varphi(\mathbf{x}) \leq |\mathbf{p}|$.

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

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

Proof: By Lemma 2 and Theorem 1, each call to GREEDY-MOVE requires $O(|\mathbf{p}|\log|\mathbf{p}|)$. Since there are $O(D/|\mathbf{p}|)$ 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 $\Delta^* \equiv \min_{l\in\mathbf{p}} \Delta_l(x_l, -1)$. Then, $\mathbf{x}$ is optimal iff $\Delta_e(x_e, -1) \geq \Delta^* \geq -\Delta_l(x_l, 1)$ for all $e, l \in \mathbf{p}$.

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

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

This implies that an optimal solution to Problem OPQ can be found by selecting the D largest elements from the set $\{\Delta_l(d, -1) \mid 0 \leq d \leq D, l \in \mathbf{p}\}$.

For certain cost functions, this can be done analytically. For instance, in [16] we provide an $O(|\mathbf{p}|)$ solution for cost functions that correspond to delay uncertainty with uniform probability distributions.

More generally, if the cost functions are *strictly* convex, then, *given* Δ^*, one can use (1) to find an optimal solution in $O(|\mathbf{p}|)$. In [16], a binary search is employed for finding Δ^*. Accordingly, the resulting overall solution is of $O(|\mathbf{p}|\log\Delta^{\max})$, where $\Delta^{\max} \equiv \max\{\Delta_l(d, -1) \mid 0 \leq d \leq D, l \in \mathbf{p}\}$. Note that $\log\Delta^{\max}$ is bounded by the complexity of representing a cost value.

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 $\mathbf{T} \subseteq E$ such that for all $v \in M$ there exists a path, $\mathbf{p}^{\mathbf{T}}(s, v)$, from s to v on links that belong to the tree $\mathbf{T}$. We assume there is only one outgoing link from the source s,[5] and denote this link by r.

$N(l = (u, v))$ 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*. $\mathbf{T}_l$ is the whole sub-tree originating from l (including l itself). The *branches* of $\mathbf{T}$ are denoted by $\hat{\mathbf{T}} \equiv \mathbf{T} \setminus \{r\}$. Observe that $\hat{\mathbf{T}} = \cup_{l\in N(r)}\mathbf{T}_l$.

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

We can now define Problem OPQ for multicast trees.

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

Remark 1: If there is more than one outgoing link from the source, then we can simply solve Problem MOPQ independently, for each tree $\mathbf{T}_{r_i}$ corresponding to an outgoing link r_i

[5] See Remark 1.

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

from s. Thus, our assumption, that there is only one outgoing link from r, does not limit the solution.

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

A. Greedy properties

The general resource allocation problem can be stated with tree-structured constraints and solved in a greedy fashion [12]. An efficient $O(|\mathbf{T}| \log |\mathbf{T}| \log D)$ algorithm is given in [11]. However, that "tree version" of the resource allocation problem has a different structure than Problem MOPQ. Indeed, the simple greedy approach, namely repeated augmentation of the link that most improves the overall cost, fails in our framework. 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 of our framework is that 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 of the resource allocation problem may be applicable to other multicast resource allocation problems, in which the constraints are also on sub-trees. For example, suppose a feasible allocation must recursively satisfy, for any sub-tree $\mathbf{T}_e$, $\sum_{l \in \mathbf{T}_e} x_l \leq Q(\mathbf{T}_e)$, where $Q(\mathbf{T}_e)$ 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.

Lemma 4: If $\{c_l\}_{l \in \mathbf{T}}$ are convex then so is $c_{\mathbf{T}}(d)$.

By Lemma 4, we can replace $\mathbf{T}$ by an equivalent convex link. Any sub-tree $\mathbf{T}_l$, can also be replaced with an equivalent convex link, hence so can $\hat{\mathbf{T}}$. 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 [1], and is the hallmark of the applicability of both *dynamic-programming* and *greedy* methods.

Lemma 5: Let $\mathbf{x}^*_{\mathbf{T}} \in$ MOPQ$(\mathbf{T}, D)$. Let $e \in N(r)$ and let the *sub-partition* $\mathbf{x}^e_{\mathbf{T}_e}(D^e) = \{x^e_l = x^*_l\}_{l \in \mathbf{T}_e}$, where $D^e = D - x^*_r$. Then $\mathbf{x}^e_{\mathbf{T}_e} \in$ MOPQ$(\mathbf{T}_e, D^e)$.

Lemma 5 implies that, for any optimally partitioned trees, we can apply the greedy properties of Section III. That is, the partition on r and $\hat{\mathbf{T}}$ is a solution to Problem OPQ on the 2-link path $(r, \hat{\mathbf{T}})$. This suggests that employing greedy moves between r and $\hat{\mathbf{T}}$ will solve Problem MOPQ, and this method can be applied recursively for the sub-trees of $\mathbf{T}$. Indeed, this scheme is used by the algorithms presented in the next sections.

B. Pseudo-polynomial solution

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

Procedure TREE-ADD (Figure 4) performs a δ-augmentation on a tree $\mathbf{T}$. We assume that for each sub-tree $\mathbf{T}_l$ the value of $\Delta_{\mathbf{T}_l}(D^{\mathbf{T}_l}, \delta)$ for the current allocation is stored in the variable $\Delta_{\mathbf{T}_l}(\delta)$. At Line 1 it is decided if r or $\hat{\mathbf{T}}$ would be augmented. $\Delta_{\mathbf{T}}(\delta)$ is either $\Delta_{\hat{\mathbf{T}}}(D^{\mathbf{T}_l})$ or $\Delta_r(x_r, \delta)$, therefore this decision can be made by a simple comparison. If $\hat{\mathbf{T}}$ should be augmented then Procedure TREE-ADD is called recursively on its components. Finally, $\Delta_{\mathbf{T}}(\pm\delta)$ is updated at Lines 6–7.

TREE-ADD $(\mathbf{x}, \delta, \mathbf{T})$:
1 if $\Delta_r(x_r, \delta) = \Delta_{\mathbf{T}}(\delta)$ then
2 $\quad x_r \leftarrow x_r + \delta$
3 else
4 $\quad$ for each $l \in N(r)$ do
5 $\quad\quad$ TREE-ADD $(\mathbf{x}, \delta, \mathbf{T}_l)$
6 $\Delta_{\mathbf{T}}(\delta) \leftarrow \min\{\Delta_r(x_r, \delta), \sum_{l \in N(r)} \Delta_{\mathbf{T}_l}(\delta)\}$ [a]
7 $\Delta_{\mathbf{T}}(-\delta) \leftarrow \min\{\Delta_r(x_r, -\delta), \sum_{l \in N(r)} \Delta_{\mathbf{T}_l}(-\delta)\}$ [6]

[a] If r is a leaf we define the sum to be ∞.

Fig. 4. Procedure TREE-ADD

Algorithm BALANCE (Figure 5) is a dynamic algorithm that solves Problem MOPQ. It starts from any feasible tree partition and performs greedy moves between r and $\hat{\mathbf{T}}$. The *while* loop at Line 7 computes $\Delta_{r \to \hat{\mathbf{T}}}(\mathbf{x}, \delta)$. If it is negative then moving δ from r to $\hat{\mathbf{T}}$ reduces the overall cost. The augmentation of $\hat{\mathbf{T}}$ is done by calling TREE-ADD on each of its components. The *while* loop at at Line 11 performs moves from $\hat{\mathbf{T}}$ to r in a similar way.

To be able to check the *while* condition and for calling TREE-ADD, we must have $\Delta_{\mathbf{T}_l}(\pm\delta)$ for all $l \in \mathbf{T}$. 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 $\hat{\mathbf{T}}$. 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 $\Delta_{\mathbf{T}}(\pm\delta)$, which is used by the calling iteration.

BALANCE $(\mathbf{x}, \delta, \mathbf{T})$:
1 if $\mathbf{T}$ is a leaf then
2 $\quad \Delta_{\mathbf{T}}(\delta) \leftarrow \Delta_r(x_r, \delta)$
3 $\quad \Delta_{\mathbf{T}}(-\delta) \leftarrow \Delta_r(x_r, -\delta)$
4 $\quad$ **return**
5 (else) for each $l \in N(r)$ do
6 $\quad$ BALANCE $(\mathbf{x}, \delta, \mathbf{T}_l)$
7 while $\sum_{l \in N(r)} \Delta_{\mathbf{T}_l}(\delta) + \Delta_r(x_r, -\delta) < 0$
8 $\quad x_r \leftarrow x_r - \delta$
9 $\quad$ for each $l \in N(r)$ do
10 $\quad\quad$ TREE-ADD $(\mathbf{x}, \delta, \mathbf{T}_l)$
11 while $\Delta_r(x_r, \delta) + \sum_{l \in N(r)} \Delta_{\mathbf{T}_l}(-\delta) < 0$
12 $\quad x_r \leftarrow x_r + \delta$
13 $\quad$ for each $l \in N(r)$ do
14 $\quad\quad$ TREE-ADD $(\mathbf{x}, -\delta, \mathbf{T}_l)$
15 $\Delta_{\mathbf{T}}(\delta) \leftarrow \min\{\Delta_r(x_r, \delta), \sum_{l \in N(r)} \Delta_{\mathbf{T}_l}(\delta)\}$
16 $\Delta_{\mathbf{T}}(-\delta) \leftarrow \min\{\Delta_r(x_r, -\delta, \sum_{l \in N(r)} \Delta_{\mathbf{T}_l}(-\delta)\}$

Fig. 5. Algorithm BALANCE

We proceed to analyze the complexity of BALANCE. We first define a *distance* $\varphi_{\mathbf{T}}(\mathbf{x})$ which is the tree version of the path distance defined in Section III-B. Let $\varphi_r(\mathbf{x_T}) \equiv |x_r - x_r^*|$, where $\mathbf{x}_\mathbf{T}^*$ is the optimal partition nearest to $\mathbf{T}$. Let $\mathbf{x}_{\mathbf{T}_l}^l = \{x_e^l = x_e^*\}_{e \in \mathbf{T}_l}$. We define $\varphi(\mathbf{x_T}) \equiv \sum_{l \in \mathbf{T}} \varphi_r(\mathbf{x}_{\mathbf{T}_l}^l)$.

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

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

Remark 2: We can apply Algorithm BALANCE on the feasible partition $\mathbf{x_T} = \{x_r = D;\ x_l = 0 \quad \forall l \neq r\}$. Clearly, $\varphi(\mathbf{x}) \leq D$ in this case. Thus, Problem MOPQ can be solved in $O(|\mathbf{T}|D/\delta)$.

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 leafs. A straightforward implementation requires $O(t)$ time,[7] where t is the depth of the tree. At most $O(t)$ recursive calls to BALANCE are performed sequentially. Finally, the number of calls to TREE-ADD *after* the sub-trees are balanced, is bounded by $\varphi_r^{\max}(\mathbf{x})/\delta$, where $\varphi_r^{\max}(\mathbf{x}) \equiv \max_{l \in \mathbf{T}} \varphi_r(\mathbf{x}_{\mathbf{T}_l}^l)$. The overall complexity is, therefore, $O\left(t^2 \varphi_r^{\max}(\mathbf{x})/\delta\right)$. Note that for balanced trees $t = O(\log|\mathbf{T}|)$.

Algorithm BALANCE (as is the case for Algorithm GREEDY-MOVE) 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 $\varphi(\mathbf{x})$) incur a small number of computations.

D. *Polynomial solution*

We can now present a polynomial solution. Algorithm BINARY-MOPQ 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. 6. Algorithm BINARY-MOPQ

We will show that, at each iteration of the algorithm, $\varphi_r^{\max}(\mathbf{x})$ is bounded by $t\delta$. Therefore, $\varphi(\mathbf{x})/\delta \leq t|\mathbf{T}|$ and the overall complexity of this algorithm is $O(|\mathbf{T}|^2 t \log(D/|\mathbf{T}|))$. The Lemma 6 is the equivalent of Lemma 2 for multicast.

Lemma 6: Let $\mathbf{x} \leftarrow$BALANCE$(\mathbf{x}, 2, \mathbf{T})$. Then, $\varphi_r(\mathbf{x}) < t$.

[7] assuming that traveling a link requires one time unit.

Let $\Delta_\mathbf{T}(d, \delta)$ denote the value of $\Delta_\mathbf{T}(\delta)$ at the termination of BALANCE$(\mathbf{x}(d), \delta, \mathbf{T})$. Note that $\Delta_\mathbf{T}(d, \delta)$ assumes a δ-optimal partition on the tree, hence it is different from $(c_\mathbf{T}(d + \delta) - c_\mathbf{T}(d + \delta))/\delta$, which assumes the optimal partition.

Theorem 4 states the complexity of Algorithm BINARY-MOPQ.

Theorem 4: Algorithm BINARY-MOPQ finds a solution to Problem MOPQ in $O(|\mathbf{T}|^2 t \log(D/|\mathbf{T}|))$.

Comparing this result to the $O(|\mathbf{T}|D/\delta)$ complexity of Algorithm BALANCE (see Remark 2), indicates that Algorithm BALANCE is preferable when $D/\delta < |\mathbf{T}| t \log(D/|\mathbf{T}|)$.

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

Remark 3: 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.

Remark 4: It is possible to modify the algorithm to cope with heterogeneity in the QoS requirements of the multicast group members. In the worst case, the complexity of the solution grows by a factor of $O(|M|)$, while the complexity of the pseudo-polynomial solution remains unchanged. The details of this extension are omitted here.

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 [7], [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 [6]. 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.

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 [6].

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.

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.[8] 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 Figure 7. 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

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

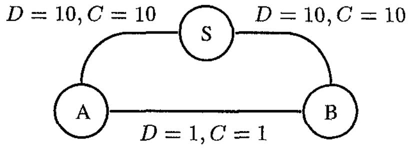

Fig. 7. Example: extending a multicast tree

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 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 in 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 implemented in a distributed fashion. The complexity of this solution is $O(\varphi(\mathbf{x}) \log |\mathbf{p}|)$, where $\varphi(\mathbf{x}) \leq D$ is the distance between the initial allocation, $\mathbf{x}$, and the optimal one. It can also be applied as a dynamic scheme to modify an existing allocation. This is useful in dynamic environments where the cost of resources changes from time to time. Note that the complexity is proportional to $\varphi(\mathbf{x})$, meaning that small cost changes require a small number of computations to regain optimality. Algorithm BINARY-OPQ is a polynomial solution from which we later build our solution to the multicast problem (MOPQ). The complexity of this solution is $O\left(|\mathbf{p}|\log|\mathbf{p}|\log(D/|\mathbf{p}|)\right)$.

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 an efficient solutions. Algorithm BALANCE is a pseudo-polynomial algorithm which can be applied as a dynamic scheme. Its complexity is $O(|\mathbf{T}|\varphi(\mathbf{x}) + |\mathbf{T}|)$, where $\varphi(\mathbf{x})$ is, again, the distance between the initial allocation and the optimal one. A distributed implementation requires $O(t^2\varphi_r^{\max}(\mathbf{x}))$, where t is the depth of the tree and $\varphi_r^{\max}(\mathbf{x})$ is the maximal distance of any *link*'s allocation from it optimal one. Note that for balanced trees $t = O(\log|\mathbf{T}|)$. Algorithm BINARY-MOPQ is a polynomial solution with a complexity of $O(|\mathbf{T}|^2 t\log(D/|\mathbf{T}|))$. A distributed implementation of this algorithm requires $O(t^3\log(D/|\mathbf{T}|)$. We note that our solutions are applicable to heterogeneous multicast members, each with a different delay requirement.

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 important aspect is the actual implementation of our solutions in 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 at network elements 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. Further study should consider the implications and potential advantages of our framework, when applied to the various problems and facets of QoS networking.

REFERENCES

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

[2] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick. A framework for QoS-based routing in the internet – RFC no. 2386. Internet RFC, April 1998.

[3] R. Braden (Ed.), L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource reSerVation Protocol (RSVP) version 1, functional specification – RFC no. 2205. Internet RFC, September 1997.

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

[5] Greg N. Frederickson and Donald B. Johnson. The complexity of selection and ranking in $X + Y$ and matrices with sorted columns. *Journal of Computer and System Sciences*, 24:197–208, 1982.

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

[7] L. Georgiadis, R. 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.

[8] R. Guérin, S. Kamat, A. Orda, T. Przygienda, and D. Williams. QoS routing mechanisms and OSPF extensions. Internet Draft, January 1998.

[9] R. Guérin and A. Orda. QoS-based routing in networks with inaccurate information: Theory and algorithms. In *Proceedings of the IEEE INFOCOM'97*, pages 75–83, Kobe, Japan, April 1997.

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

[11] Dorit 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] Toshihide Ibaraki and Naoki 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*, 1:286–292, 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] D. H. Lorenz and A. 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] D. H. Lorenz and A. Orda. QoS routing in networks with uncertain parameters. *IEEE/ACM Transactions on Networking*, 6(6), December 1998. to appear.

[17] S. 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] A. Orda and N. Shimkin. Incentive pricing in multi-class communication networks. In *Proceedings of the IEEE INFOCOM'97*, Kobe, Japan, April 1997.

[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 R. Guerin. Specification of guaranteed quality of service – RFC no. 2212. Internet RFC, November 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.