Efficient QoS Partition and Routing of Unicast and Multicast

Dean H. Lorenz, Ariel Orda, *Fellow, IEEE*, Danny Raz, *Member, IEEE*, and Yuval Shavitt, *Senior Member, IEEE*

***Abstract*—In this paper, we study problems related to supporting unicast and multicast connections with quality of service (QoS) requirements. We investigate the problem of optimal routing and resource allocation in the context of performance dependent costs. In this context, each network element can offer several QoS guarantees, each associated with a different cost. This is a natural extension to the commonly used bi-criteria model, where each link is associated with a single delay and a single cost. This framework is simple yet strong enough to model many practical interesting networking problems.**

An important problems in this framework is finding a good path for a connection that minimizes the cost while retaining the end-to-end delay requirement. Once such a path (or a tree, in the multicast case) is found, one needs to partition the end-to-end QoS requirements among the links of the path (tree). We consider the case of general integer cost functions (where delays and cost are integers). As the related problem is NP complete, we concentrate on finding efficient ε-approximation solutions. We improve on recent previous results by Ergün *et al.* Lorenz and Orda, and Raz and Shavitt, both in terms of generality as well as in terms of complexity of the solution. In particular, we present novel approximation techniques that yield the best known complexity for the unicast QoS routing problem, and the first approximation algorithm for the QoS partition problem on trees, both for the centralized and distributed cases.

***Index Terms*—Approximation, multicast, QoS-dependent costs, QoS, resource allocation, routing.**

I. Introduction

QUALITY OF SERVICE (QoS) support is a growing need in broadband networks. Many modern applications require better service than the Internet's best effort mechanism. There have been numerous suggestions for QoS provisioning and it has been the focus of many recent studies. Indeed, there is a growing consensus that QoS support in the Internet is necessary. Almost any QoS framework requires a QoS Routing (QoSR) mechanism, and this has been the subject of many proposals, as described in [1]–[3] and references therein. QoSR aims at setting the connection topology for an application, i.e., a path for unicast and a tree for multicast, based on its QoS requirements and some optimization criteria.

Many of the QoSR algorithms first restrict the route selection to a set of *feasible* routes, which have sufficient resources to guarantee the QoS requirements of the application, and then choose an optimal route out of this set. The optimization criterion is generally defined in terms of a "cost", namely: there is a cost associated with ensuring a specific QoS guarantee on a specific route. Naturally, this cost is higher for more stringent requirements, such as larger bandwidth or shorter delay. In many cases, the cost is not explicitly given but rather implied. An implied cost mechanism is quite flexible and may be used to incorporate different considerations.

Link considerations The cost may represent the consumption of local resources that must be reserved on every link of the route to support the QoS guarantee. These may include buffer or bandwidth reservations.

Network considerations QoSR may be used to improve overall network efficiency or enforce fairness. The cost may represent the decrease in overall network performance from establishing the selected connection. For instance, there may be loss of revenue due to blocked future calls, or there may be management costs.

User considerations There are several proposals for pricing schemes for different QoS classes. Given such a pricing scheme, the user would attempt to choose the cheapest feasible route.

Other considerations Other optimization criteria may be expressed in terms of costs. For instance, where there is parameter uncertainty, the cost may represent the probability of a bad estimate.

Identifying feasible routes may be a difficult task, and its complexity corresponds to the intricacy of the QoS mechanisms (scheduling, signaling, and resource reservation) and of the required QoS guarantee. The constraints on the feasible set may be relaxed to include routes that are feasible with just "high" probability or that provide just statistical guarantees. The QoS guarantees themselves may be imposed on the whole connection or on each individual link. The latter typically requires mapping the application's end-to-end requirements into local requirements.

In this paper, we investigate a model in which a performance-dependent cost is associated with each network link. The goal of the QoSR process is to identify a route and a set of *local* demands on its links as to minimize the overall cost incurred. A feasible allocation of demands must satisfy the end-to-end requirement of the application.

In order to illustrate the framework of our study, consider a voice-over-IP session. Such an application requires a delay

Manuscript received May 21, 2003; revised January 5, 2005, and August 8, 2005; approved by IEEE/ACM Transactions on Networking Editor L. Gao.

D. H. Lorenz is with the IBM Haifa Research Laboratories, IBM, Mount Carmel, Haifa 31905, Israel. Part of this work was done while he was with the Department of Electrical Engineering, Technion, Haifa 32000, Israel (e-mail: dean@il.ibm.com).

A. Orda is with the Department of Electrical Engineering, Technion, Haifa 32000, Israel (e-mail: ariel@ee.technion.ac.il).

D. Raz is with the Department of Computer Science, Technion, Haifa 32000, Israel. Part of this work was done while he was with Bell Labs, Lucent Technologies (e-mail: danny@cs.technion.ac.il).

Y. Shavitt is with the Department of Electrical Engineering, Tel Aviv University, Ramat Aviv 69978, Israel. Part of this work was done while he was with Bell Labs, Lucent Technologies (e-mail: shavitt@eng.tau.ac.il).

Digital Object Identifier 10.1109/TNET.2006.886323

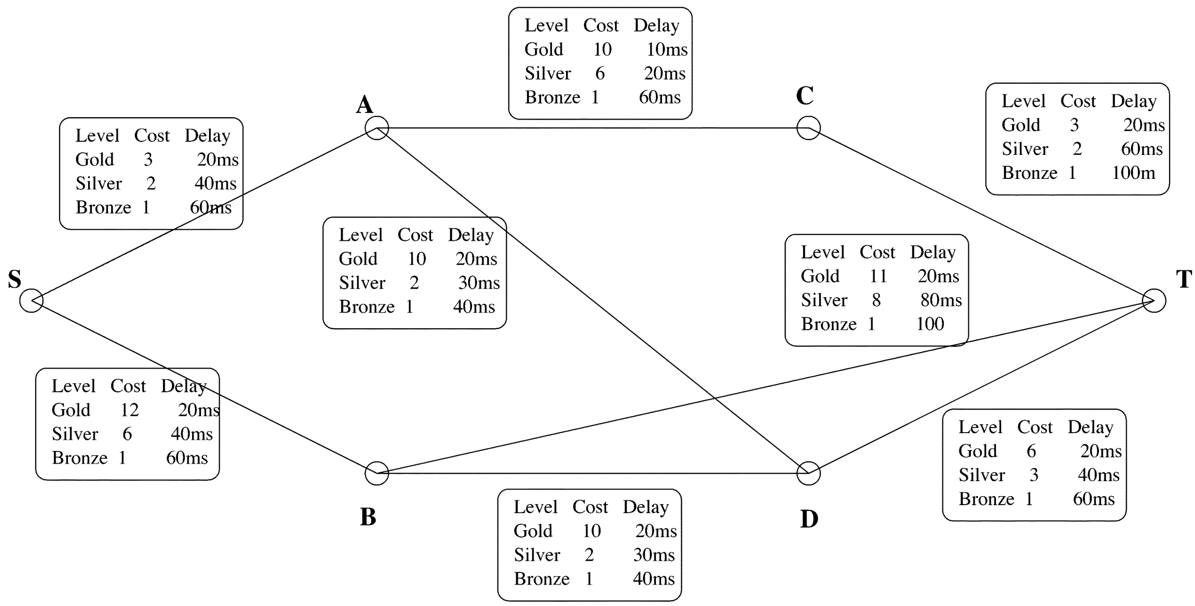

Fig. 1. Example: QoS routing and partition.

bound of say 120 ms in order to be at an acceptable quality level. Assume further that the participants in the call are located at different locations and the links between them can travel through different administrative domains. One could construct different paths connecting the participants in the call, each resulting with different QoS (delay) guaranteed by the different providers according to the different service level agreements (SLAs) we have with each. Even if the path is fixed (i.e., we cannot change the routing), we could still decide to partition the delay "budget" in various ways among the different providers. Among all such partitions that guarantee the required QoS, we would seek to use the one that is most cost-effective. Fig. 1 illustrates such a scenario. Assume a voice session from node S to node T. There are several possible paths for this session, each using different links that represent different ISPs. Each ISP offers three levels of service (Gold, Silver, Bronze) each having a different QoS (delay) guaranteed for a different cost (as described in the small tables along each link). If our delay limit is 120 ms, then we can choose the path $S-B-T$ with service levels Bronze on the first link and Gold on the second. This results in a guaranteed delay of 120 ms at a cost of 12 units. However, we can also choose the path $S-A-C-T$, with Silver levels at each of the links and have the same guaranteed delay for a cost of only 10 units. As we indicated above, even for a fixed path one can get different costs for the same delay via different partitions. For example, by choosing the service levels Gold, Bronze, and Gold, on the respective links of the path $S-A-C-T$, instead of Silver levels at all links, we have a guaranteed delay of 100 ms for a reduced cost of only 7 units.

The optimal solution must be chosen out of all combinations of route and demand allocation, namely it is a combined routing and resource allocation optimization problem. In order to associate costs to delays, we employ *integer functions*, which better fit practical purposes (see [4] and references therein). We also focus on additive (e.g., delay) QoS requirements, which are typically harder to solve for than bottleneck (e.g, rate) requirements (see [5] for a more detailed discussion).

This model and related problems were recently addressed by several works. Some studies assumed that the route (i.e., unicast path or multicast tree) is given and only the resource allocation part of the problem is solved. Heuristics for loss rate guarantees on unicast connections were presented in [6]. Optimal solutions for convex cost functions were discussed in [7] under the broader scope of a general resource allocation problem. An optimal solution for (weakly) convex cost functions and improved results for specific cost functions were presented in [4], [5], and [8]. Heuristics for the resource allocation problem for multicast connections were given in [9], and the problem was optimally solved in [5]. A variant of this problem for rate guarantees was studied in [10], and a more efficient solution was given in [11].[1] Distributed optimal solutions were presented in [5], and a detailed version for multicast connections was given in [4].

The combined problem of partition and routing of QoS requirements was also addressed. Optimal multicast tree construction is a very complex problem even in simpler frameworks (e.g., constrained Steiner tree [12]), thus the combined route selection and resource allocation problem was solved only for unicast connections. Optimal solutions were presented in [13] for rate demands and rate-based delay requirements, and in [8] for general (integer) delay requirements with convex cost functions.

Although these problems have been proven to be intractable, efficient ε-approximations may be derived. The approximate solutions are ε-optimal in the sense that their cost is within a factor of $1+\varepsilon$ of the optimal cost. The running time is polynomial in $1/\varepsilon$, that is, there is a tradeoff between the accuracy of the solution and the computational effort needed to find it. An approximation scheme for the combined routing and resource allocation problem was introduced in [8]. That approximation scheme

[1]Ref. [11] presented efficient solutions for a broader family of optimization problems, which includes the one discussed in [10].

required several limiting assumptions, including convexity of the cost functions. A fully polynomial approximation scheme (FPAS) for general (integer) costs was recently obtained by [14].

A special case of practical interest was studied by [4]. That study assumed *discrete* costs,[2] meaning that each link offers only a limited number of QoS guarantees (and costs) instead of the complete spectrum of requirements. Under this assumption, [4] presented *strictly* polynomial approximations for the combined routing and resource allocation problem and for the multicast resource allocation problem as well.

This paper presents efficient approximation schemes for general integer cost functions and end-to-end delay requirements. Previous approximation schemes [4], [8], [14] were all derived from approximations to the *restricted shortest path* problem obtained by [15] and were restricted only to either *acyclic* graphs or *nonzero* costs. A more efficient approximation scheme for that problem was proposed in [16]. We use similar techniques, which apply to general (i.e., also cyclic) graphs and allow for links with zero cost; in addition we present the first polynomial time approximation scheme for the optimal resource allocation problem on multicast trees with general integer cost functions. Furthermore, our results improve upon the previous ones in terms of time complexity, namely: they have a better time complexity than the results of [14] for integer costs and the results of [4] for discrete costs.

To illustrate the significance of this latter contribution, consider a network of a moderate size, say 100 nodes, and suppose the delay is measured in milliseconds, and the cost values range from 1 to 1000. Then, in order to approximate the optimal solution within 1% (i.e., $\varepsilon = 0.01$), the best previously known algorithm would require an order of 10^{10} instructions, while our algorithm only needs an order of 10^6 instructions. In other words, a solution of the same quality can now be achieved within *seconds* rather than within *hours*. For networks of larger size, of say 1000 nodes, our algorithm would require an order of 10^7 instructions, while the best previously known algorithm would require an order of 10^{12} instructions. Practical aspects of this work and an evaluation of the actual (rather than worst-case) running time and solution quality are out of the scope of this study and can be found in [17].

The rest of this paper is structured as follows. Sections II—V discuss the combined path selection and resource allocation problem for unicast connections. Section II formulates the model and problems and presents pseudo-polynomial solutions, which are the basis for our approximations. In Section III, we present approximation techniques, which rely on tight lower and upper bounds on the cost of an optimal solution. The problem of efficiently finding such bounds is solved in Section IV and the full approximation process is given in Section V. Section VI applies similar approximation techniques to solve the resource allocation problem on multicast trees. Finally, concluding remarks are given in Section VII.

II. Preliminaries

In this section, we give a formal definition of the problem and present simple dynamic programming pseudo-polynomial solutions. These pseudo-polynomial algorithms are used as "building blocks" for the approximation algorithms presented in the rest of the paper. Similar (and more detailed) solutions can be found in previous works [4], [5], [8].

```
RSP: (G(V,E), {d_l, c_l}_{l∈E}, D, U)
1  for all v ≠ s
2     D(v,0) ← ∞
3  D(s,0) ← 0
4  for i = 1,2,...,U
5     for v ∈ V
6        D(v,i) ← D(v,i−1)
7        for l ∈ {(u,v) | c_(u,v) ≤ i}
8           D(v,i) ← min{D(v,i), d_l + D(u,i−c_l)}
9     if D(t,i) ≤ D
10       return the corresponding path
11 return FAIL
```

Fig. 2. Algorithm RSP.

The network is represented as a graph $G(V,E)$, where $|V| = n$ and $|E| = m$. There is a single source node s and a single target node t. We denote an st-path by $\boldsymbol{p}$, and its length (number of hops) by $|\boldsymbol{p}|$. Each link $l \in E$ offers different delays at different costs. The cost associated with a link is a (link-dependent) function $c_l(d_l)$ of the delay allocated to it. Each link cost function $c_l(d_l)$ is nonincreasing with the delay and both the delays and associated costs are assumed to be integers. We will further assume that the minimal cost of any link is 1. However, we shall relax these assumptions in Section V-B.

A delay *partition* on a path $\boldsymbol{p}$ specifies the delay allocated on each link, i. e., is a set of link delays $\boldsymbol{d} \equiv \{d_l\}_{l\in\boldsymbol{p}}$. The cost of a partition $\boldsymbol{d}$ on a path $\boldsymbol{p}$ is defined as the sum of all link costs, namely $cost(\boldsymbol{p},\boldsymbol{d}) \equiv \sum_{l\in\boldsymbol{p}} c_l(d_l)$.[3]

A. Restricted Shortest Path

The Restricted Shortest Path Problem (see e.g., [15]) can be viewed as a special case of our problem. Each link $l \in E$ offers a single delay and cost, which are denoted by d_l and c_l respectively, and are assumed to be integers; In other words, the cost function is only defined for a single delay value. We define the cost of a path as $c(\boldsymbol{p}) \equiv \sum_{l\in\boldsymbol{p}} c_l$ and the delay of a path by $delay(\boldsymbol{p}) \equiv \sum_{l\in\boldsymbol{p}} d_l$.

Problm RSP—Restricted Shortest Path: Given a network $G(V,E)$, a delay/cost pair for each link $\{d_l, c_l\}_{l\in E}$, and an end-to-end requirement D. Find the minimal cost path among all paths that satisfy $delay(\boldsymbol{p}) \le D$.

Algorithm RSP (Fig. 2) is a pseudo-polynomial dynamic programming algorithm that solves RSP.

The parameter U is an upper bound on the cost of the solution. The algorithm returns the minimal cost path that satisfies the delay requirement if the cost of this path is no greater than U, otherwise it fails.

Complexity For each i, each link is examined at most once, thus the overall complexity is $O(mU)$. If a solution is found then the complexity is $O(mc^*)$, where c^* is the cost of the optimal solution.

Note If some links have a zero cost, then the updates in line 8 may not be performed in an arbitrary order. For acyclic graphs,

[2]We follow the term used by [4].

[3]The minimal link cost is assumed to be 1.

OPQR: $(G(V,E), \{c_l(d)\}_{l \in E}, D, U)$
1 for all $v \neq s$
2 $\quad D(v,0) \leftarrow \infty$
3 $D(s,0) \leftarrow 0$
4 for $i = 1, 2, \ldots, U$
5 $\quad$ for $v \in V$
6 $\quad\quad D(v,i) \leftarrow D(v,i-1)$
7 $\quad\quad$ for $l \in \{(u,v) \mid v \in V\}$
8 $\quad\quad\quad$ for $j = 1, 2, \ldots, i$
9 $\quad\quad\quad\quad d_l(j) = \min\{d \mid c_l(d) \leq j\}$
10 $\quad\quad\quad\quad D(v,i) \leftarrow \min\{D(v,i), d_l(j) + D(u, i-j)\}$
11 $\quad$ if $D(t,i) \leq D$
12 $\quad\quad$ return the corresponding path and partition.
13 return FAIL

Fig. 3. Algorithm OPQR.

the "natural" partial order induced by the graph ensures correctness, however establishing a correct update order for general graphs requires a shortest path computation at each iteration of the algorithm and adds to the complexity.

B. Optimal QoS Partition and Routing

We now generalize the results for *integer cost functions*. Each link may offer different (integer) delay guarantees, d_l, each associated with a (integer) cost $c_l(d_l)$. The cost of a path $\boldsymbol{p}$ with a given delay partition $\{d_l\}_{l \in \boldsymbol{p}}$ is defined as $c(\boldsymbol{p}) = \sum_{l \in \boldsymbol{p}} c_l(d_l)$.

Problem OPQR—Optimal QoS Partition & Routing: Given a network $G(V,E)$, a delay/cost *function* for each link $\{c_l(d)\}_{l \in E}$, and an end-to-end requirement D. Find the minimal cost path $\boldsymbol{p}$ and partition $\{d_l\}_{l \in \boldsymbol{p}}$ that satisfies the end-to-end delay requirement D.

We denote the optimal path by $\boldsymbol{p}^*$ and the optimal partition by $\boldsymbol{d}^* = \{d_l^*\}_{l \in \boldsymbol{p}^*}$ with optimal cost c^*.

The following dynamic programming algorithm (Fig. 3) solves Problem OPQR. The general idea behind the algorithm is to view each link l as a set of links $\{l_1, l_2, \ldots, l_U\}$ corresponding to all possible costs on the link. The dynamic program iteratively considers cost values. For each such cost value (i) and for each node in the network, we compute the minimum delay that can be obtained from the source to that node, incurring a cost of at most i units. Note that, in general, the nondecreasing cost function may have several delay values associated with the same cost value. Hence, in line 9, for each cost value, we chose among all delay values that correspond to this cost, the smallest one. The delay associated with each of these links is the minimal delay that achieves the specified cost (line 9). The algorithm iteratively considers increasing cost values, and halts upon the first cost value for which the minimal possible delay between source and destination does not exceed the end-to-end requirement D.

Complexity For each possible cost value i, each link is examined i times (in line 8), i.e., $O(U^2)$ examinations overall. In each examination in line 9, we need to find the minimal delay that has a cost no greater than j, which requires $O(\log D)$ steps, implying an overall complexity of $O(mU^2 \log D)$. If we save the $d_l(j)$ values and compute it only for new values of j, then repeated examinations can be done in $O(1)$. At most U new computations (each requiring $O(\log D)$) are required per link. The overall complexity would then be $O(mU^2 + mU \log D) = O(mU(U + \log D))$. If a solution is found, then the overall complexity is $O(mc^*(c^* + \log D))$, where c^* is the cost of the optimal solution.

Note 1 In some cases the complexity of the computation in line 9 can be done in less than $O(\log D)$. For instance, if the inverse functions $\{d_l(c)\}_{l \in E}$ are available (e.g., they have an explicit analytic expression which has an inverse form) then it can be computed in $O(1)$ and the $\log D$ can be eliminated from the complexity.

Note 2 If the cost functions are (weakly) convex, then Algorithm OP-MP of [8] can be applied.[4] The resulting complexity is $O(mU(\log U + \log D))$.

III. Sampling and Scaling

In this section, we present approximation techniques based on sampling and scaling. The two methods are used in succession at a preliminary stage to produce an instance of either Problem RSP or Problem OPQR with smaller integer parameters. We then find an approximated solution by calling the appropriate pseudo-polynomial algorithm presented in the previous section. Since the complexities of Algorithm RSP and Algorithm OPQR depend on their integer input parameters, reducing the values of these parameters improves the complexity.

On the other hand, both sampling and scaling introduce an error in the cost on every link, because they affect the granularity of the parameters. There is a tradeoff between the accuracy of the obtained solution and the complexity of the algorithms. We seek an ε-approximation, namely a solution with cost no greater than a factor of $(1+\varepsilon)$ from the optimum. The value of ε is an input to the algorithms and the complexities polynomially depend on $1/\varepsilon$.

In this section, we assume that an upper bound and a lower bound on the optimal solution are given. In the next section, we show how to efficiently obtain these bounds. Note that the tighter these bounds are, the lower is the complexity of finding a solution.

A. Logarithmic Sampling

In this section, we employ logarithmic sampling on the cost functions. The idea is not to check the cost functions for every possible cost, as is done by Algorithm OPQR. Instead, we check delays that correspond to specific costs on a logarithmic scale. Specifically, we check delays that correspond to costs of $1, (1+\varepsilon), (1+\varepsilon)^2, \ldots, U$, where U is an upper bound on the maximal cost. We replace each link with a set of links, each corresponding to a specific delay (and cost), and then we solve Problem RSP. Algorithm L-OPQR (Fig. 4) finds an ε-approximation to OPQR.

Lines 1–7 select the delays on logarithmic scale costs, line8 calls Algoithm RSP, and lines 10–13 compute the partition in terms of the original problem.

Complexity Let $\hat{m} = mI^{\varepsilon} = O(m \log U/\varepsilon)$. Initializing $\hat{G}$ requires $O(\hat{m} \log D)$. Calling Algorithm RSP requires $O(\hat{m}U)$. The overall complexity is therefore $O((m \log U/\varepsilon)(\log D + U))$.

[4]*Without* the additional assumptions of [8] (e.g., bounds on the cost of each link).

L-OPQR$: (G(V,E), \{c_l(d)\}_{l\in E}, D, U, \varepsilon)$
1 $I^{\varepsilon} = \lceil \log_{1+\varepsilon} U \rceil$
2 for each $l \in E$
3 for each $j = 0, 1, \ldots, I^{\varepsilon}$
4 $d_{lj} \leftarrow \min\{d \mid c_l(d) \le (1+\varepsilon)^j\}$
5 $c_{lj} \leftarrow (1+\varepsilon)^j$
6 $\hat{E} = \{lj \mid l \in E, j = 0 \ldots I^{\varepsilon}\}$
7 $\hat{U} = (1+\varepsilon)U$
8 $\hat{\boldsymbol{p}} \leftarrow$ RSP $\left(\hat{G}(V,\hat{E}), \{d_{lj}, c_{lj}\}_{lj\in\hat{E}}, D, \hat{U}\right)$
9 if $\hat{\boldsymbol{p}}$ = FAIL then return FAIL
10 (else) $\boldsymbol{p} \leftarrow \{l \mid \exists lj \in \hat{\boldsymbol{p}}\}$
11 for each $l \in \boldsymbol{p}$
12 $\hat{\jmath}_l \leftarrow \{j \mid lj \in \hat{\boldsymbol{p}}\}$
13 return $\boldsymbol{p}, \{d_{l\hat{\jmath}_l}\}_{l\in\boldsymbol{p}}$

Fig. 4. Algorithm L-OPQR.

Note If $I^{\varepsilon} > U$ then logarithmic scaling does not improve the complexity, and an exact solution can be found in $O(mU(\log D + U))$ by using Algorithm OPQR.

Theorem 1: If Algorithm L-OPQR returns FAIL then $c^* > U$. Otherwise the returned path $\boldsymbol{p}$ and its corresponding partition are a feasible solution to Problem OPQR with cost

$$c(\boldsymbol{p}) \le (1+\varepsilon)\min\{c^*, U\}$$

namely $\boldsymbol{p}$ is an $1+\varepsilon$ approximate solution.

Proof: For each $l \in \boldsymbol{p}^*$, let $j_l^* = \lceil \log_{1+\varepsilon} c_l^* \rceil$.[5] Obviously

$$c_l^* \le (1+\varepsilon)^{j_l^*} \le (1+\varepsilon)c_l^*. \tag{1}$$

From the definition of line 4, and since $c_l^* = c_l(d_l^*) \le (1+\varepsilon)^{j_l^*}$, we get $d_{lj_l^*} \le d_l^*$. Thus, $\sum_{l\in\boldsymbol{p}^*} d_{lj_l^*} \le D$, i.e., the path $\{lj_l^*\}_{l\in\boldsymbol{p}^*}$ is a feasible solution on $\hat{G}$. By the same definition (line 4), $c_{lj_l^*} \le (1+\varepsilon)^{j_l^*}$. Inserting this into (1) and summing we get

$$\hat{c}^* \equiv \sum_{l\in\boldsymbol{p}^*} c_{lj_l^*} \le \sum_{l\in\boldsymbol{p}^*}(1+\varepsilon)c_l^* = (1+\varepsilon)c^*. \tag{2}$$

If $U \ge c^*$ then $j_l^* \le I^{\varepsilon}$ and from (2) $\hat{c}^* \le (1+\varepsilon)U = \hat{U}$. Therefore, the feasible path $\{lj_l^*\}_{l\in\boldsymbol{p}^*}$ must be examined by the call to Algorithm RSP, and thus Algorithm L-OPQR will not return FAIL. Hence, Algorithm L-OPQR may return FAIL only if $c^* > U$.

Algorithm RSP finds the minimal cost feasible path on $\hat{G}$, with cost at most $\hat{U}$, therefore $c(\boldsymbol{p}) \le \min\{\hat{c}^*, \hat{U}\}$. Using $\hat{U} = (1+\varepsilon)U$ and (2) we get $c(\boldsymbol{p}) \le (1+\varepsilon)\min\{c^*, U\}$, as claimed. ■

B. Linear Scaling

Here we present an approximation based on linear scaling of the costs. Scaling is applied to all costs to produce an instance of Problem OPQR with smaller costs. We then call Algorithm L-OPQR to find the optimal solution.

[5]Note that the assumption $c_l \ge 1$ is needed here.

S-OPQR$: (G(V,E), \{c_l(d)\}_{l\in E}, D, U, L, \varepsilon)$
1 $S \leftarrow \frac{L\varepsilon}{n+1}$
2 for each $l \in E$
3 define $\tilde{c}_l(d) \equiv \lfloor c_l(d)/S \rfloor + 1$
4 $\tilde{U} \leftarrow \lceil U/S \rceil + n$
5 return L-OPQR$\left(\tilde{G}(V,E), \{\tilde{c}_l(d)\}_{l\in E}, D, \tilde{U}, \varepsilon\right)$

Fig. 5. Algorithm S-OPQR.

We use the lower bound L to compute a scale factor S that introduces an overall error no greater than a fraction of ε from L. If the lower bound is valid ($L \le c^*$), then this ensures the accuracy of the obtained solution. The tightness (ratio) of the upper and lower bounds determines the complexity of the algorithm. Algorithm S-OPQR (Fig. 5) uses scaling to find an ε-approximation to Problem OPQR.

Complexity The complexity is dominated by the call to Algorithm L-OPQR (line 5), which requires $O((m\log\tilde{U}/\varepsilon)(\log D + \tilde{U}))$. Let $\alpha \equiv U/L$. Then, $\tilde{U} = O(\alpha n/\varepsilon)$ and the overall complexity is

$$O\left(\frac{m}{\varepsilon}\log\frac{\alpha n}{\varepsilon}\left(\log D + \frac{\alpha n}{\varepsilon}\right)\right).$$

Note Since L is used only for scaling, it does not have to be a valid lower bound for the algorithm to produce a solution. However, it does affect the accuracy of the solution, namely we get an ε-approximation only if $L \le c^*$.

Theorem 2: If Algorithm S-OPQR returns FAIL then $c^* > U$. Otherwise, the path $\boldsymbol{p}$ returned and its partition are a feasible solution to Problem OPQR and

$$c(\boldsymbol{p}) \le (1+\varepsilon)(\min\{c^*, U\} + \varepsilon L).$$

Thus, if $L \le c^* \le U$ then $c(\boldsymbol{p}) \le c^*(1+\varepsilon)^2$, i.e., $\boldsymbol{p}$ is an $(1+\varepsilon)^2 \approx 1 + 2\varepsilon$ approximate solution.

Proof: For each $l \in E$ we have $c_l(d) \le S\tilde{c}_l(d) \le c_l(d) + S$. Summing for all links we get for any path $\boldsymbol{p}$:

$$c(\boldsymbol{p}) \le S\tilde{c}(\boldsymbol{p}) \le c(\boldsymbol{p}) + nS. \tag{3}$$

If $U \ge c^*$, then

$$\tilde{c}(\boldsymbol{p}^*) \le \frac{c^*}{S} + n \le \left\lceil\frac{U}{S}\right\rceil + n = \tilde{U}.$$

This implies that, if U is indeed an upper bound on G, then so is $\tilde{U}$ on $\tilde{G}$ (namely, with cost functions $\{\tilde{c}_l(d)\}_{l\in E}$). Therefore, if Algorithm S-OPQR returns FAIL (i.e., Algorithm L-OPQR returned FAIL), then $U < c^*$. Let $\tilde{c}^*$ be the cost of the optimal solution to Problem OPQR on $\tilde{G}$. Since $\boldsymbol{p}^*$ is a feasible partition on $\tilde{G}$ we must have

$$\tilde{c}^* \le \tilde{c}(\boldsymbol{p}^*) \le \frac{c^*}{S} + n. \tag{4}$$

The path $\boldsymbol{p}$ returned by Algorithm L-OPQR must satisfy $\tilde{c}(\boldsymbol{p}) \leq (1+\varepsilon)\min\{\tilde{c}^*, \tilde{U}\}$. Inserting into (3) and (4) we get

$$\begin{aligned} c(\boldsymbol{p}) &\leq S\tilde{c}(\boldsymbol{p}) \\ &\leq S(1+\varepsilon)\min\{\tilde{c}^*, \tilde{U}\} \\ &\leq S(1+\varepsilon)\min\left\{\tilde{c}(\boldsymbol{p}^*), \tilde{U}\right\} \\ &\leq S(1+\varepsilon)\min\left\{\frac{c^*}{S}+n, \tilde{U}\right\} \\ &\leq (1+\varepsilon)\left(\min\{c^*, U\}+(n+1)S\right) \\ &\leq (1+\varepsilon)\left(\min\{c^*, U\}+\varepsilon L\right) \end{aligned}$$

as claimed. ■

Remark 1: It is possible to replace the call to Algorithm L-OPQR on line 5 with a call to Algorithm OPQR. The overall complexity will then be $O(m(\alpha n/\varepsilon)(\log D + (\alpha n/\varepsilon)))$, which may be an improvement if ε is very small ($\log(\alpha n/\varepsilon) > \alpha n$). If a path $\boldsymbol{p}$ is returned by Algorithm S-OPQR then it satisfies $c(\boldsymbol{p}) \leq \min\{c^*, U\} + \varepsilon L$. The proof is similar to that of Theorem 2.

Remark 2: For convex cost functions, it is possible to apply scaling to the *exact* algorithm MP-OP of [8]. That is, in line 5 of Algorithm S-OPQR, Algorithm MP-OP of [8] is called instead of Algorithm L-OPQR. The overall complexity in this case is $O(m(\alpha n/\varepsilon)(\log D + \log(\alpha n/\varepsilon)))$, which is an improvement unless $D > \alpha n/\varepsilon > 2^{\alpha n}$.

IV. Finding Upper and Lower Bounds

In this section, we present algorithms for finding upper and lower bounds on the solution to Problem OPQR. We seek tight bounds, i.e., with $\alpha = U/L$ as small as possible. We can then use these bounds in the approximation algorithms of the previous section.

A. General Idea

We follow the method proposed by Hassin [15]. Suppose we have a test procedure, TEST λ, that checks whether λ is a valid upper bound. We can call TEST λ for all $\lambda \in \{1, 2, 4, 8, \ldots\}$. If for some λ^*, TEST λ^* returns FAIL and TEST $(2\lambda^*)$ succeeds then $\lambda^* \leq c^* \leq 2\lambda^*$. Clearly, since TEST λ returns FAIL for all $\lambda < c^*$, then if TEST(1) returns FAIL such a λ^* will be found in $O(\log c^*)$ tests.

Now, suppose that all we have is an *approximated* test procedure in the following sense.

Definition 1: A test procedure, TEST (λ), is an *f-approximated test procedure* if it satisfies the following:

1) if TEST (λ) returns FAIL then $\lambda < c^*$, otherwise
2) TEST(λ) returns $f(\lambda)$ and $f(\lambda) \geq c^*$.

TEST(λ) either returns a valid upper bound $f(\lambda) \geq c^*$ or FAIL. If TEST(λ) returns FAIL then λ is not a valid upper bound (i.e., $c^* > \lambda$, meaning that λ is actually a lower bound). If TEST(λ) returns $f(\lambda)$ then it is a valid upper bound, but λ may not be a valid upper bound.

```
BOUND: (TEST(), L, U)
1  if TEST(L) does not return FAIL then
2     return [L, f(L)]
3  if TEST(U) returns FAIL then
4     return ERROR
5  f ← f(U)
6  l ← log L
7  u ← log U
8  while u − l > 1
9     λ ← 2^((l+u)/2)
10    if TEST(λ) returns FAIL then
11       l ← log λ
12    else
13       u ← log λ
14       f ← f(λ)
15 return [2^l, f]
```

Fig. 6. Algorithm BOUND.

Note that the above definition is a generalization of Hassin's approximated test procedure. By setting $f(x) = (1+\varepsilon)x$, one obtains Hassin's ε-approximation test procedure [15].

We can apply the above method and call TEST(λ) for all $\lambda \in \{1, 2, 4, 8, \ldots\}$. If for some λ^*, TEST(λ^*) returns FAIL and TEST$(2\lambda^*)$ returns $f(2\lambda^*)$ then $\lambda^* \leq c^* \leq f(2\lambda^*)$. Again, if TEST(1) returns FAIL then such a λ^* must be found in $O(\log c^*)$ tests. Otherwise, if TEST(1) returns $f(1)$ then $0 \leq c^* \leq f(1)$.

If $f(\lambda)$ is a monotonic increasing function of λ and there are some known (possibly trivial) lower and upper bounds $L \leq c^* \leq U$, then the following algorithm (Fig. 6) may be used.

Algorithm BOUND performs a binary search on a logarithmic scale. This can be viewed as a search for λ^* on the group $\{L, 2L, 4L, \ldots, U\}$. The quality of the bounds we get (see line 14) depends on the accuracy of the test procedure, namely on $f(\lambda)$. Specifically, the returned bounds $[L, U]$ must satisfy

$$L \leq c^* \leq U \leq f(2L), \ i.e., \ \alpha \equiv \frac{U}{L} \leq \frac{f(2L)}{L}.$$

If, for instance, $f(\lambda) = \lambda$, then the bounds satisfy $L \leq c^* \leq U \leq 2L$, i.e., $\alpha \leq 2$.

Complexity The number of calls to TEST(λ) is of order $\log(u - l)$ with the initial l, u, that is

$$O\left(\log(\log U - \log L)\right) = O\left(\log\log\frac{U}{L}\right) \equiv O(\log\log\alpha).$$

Note The initial lower bound L is assumed to be valid. On the other hand, U does not have to be a valid upper bound, but TEST(U) must not FAIL (i.e., $f(U)$ should be a valid upper bound). A valid upper bound could be chosen as the initial U in which case TEST(U) would not FAIL, however, this would be a pessimistic bound with relatively high complexity. It is better to choose the smallest known U for which TEST(U) does not FAIL.

Altogether, we have proven the following theorem.

Theorem 3: Given an f-approximated TEST() procedure, an upper bound, U, and a lower bound, L, such that $L \leq c^* \leq f(U)$, Algorithm BOUND finds correct upper and lower bounds, u and l, such that

$$\frac{u}{l} \leq \frac{f(2l)}{l}.$$

```
Procedure TEST1: (λ)
p ← S-OPQR(G(V,E), {c_l(d)}_{l∈E}, D, λ, λ, 1)
If p = FAIL
    return FAIL
else
    return c(p)
```

Fig. 7. Procedure TEST1.

```
Procedure TEST2: (λ)
for each l ∈ E
    d_l(λ) ≡ min{d | c_l(d) ≤ λ}
p ← Shortest_st Path (G(V,E), {d_l}_{l∈E})
if delay(p) > D
    return FAIL
else
    return c(p)
```

Fig. 8. Procedure TEST2.

An obvious valid initial lower bound is 1.[6] A slightly better bound is $\min_{l\in E} c_l(D)$, which is actually a lower bound on the cost of any link. This bound can be improved by computing the length of the shortest st-path with $\{c_l(D)\}_{l\in E}$ as link lengths, since the cost of each link l on any feasible partition is at least $c_l(D)$. A valid upper bound is $\sum_{l\in \boldsymbol{p}} c_l(D/|\boldsymbol{p}|)$ for some arbitrary st-path $\boldsymbol{p}$, because $\{D/|\boldsymbol{p}|\}_{l\in \boldsymbol{p}}$ is a feasible partition. Therefore, a valid upper bound is the length of the shortest st-path with $\{c_l(D/n)\}_{l\in E}$ as link lengths.

B. The Test Procedures

In this section, we present two test procedures that can be used with Algorithm BOUND. We assume that the test procedures are aware of the problem instance, i.e., $G(V,E)$, $\{c_l(d)\}_{l\in E}, D$. For notation simplicity, we omit the problem instance from the test procedure description.

The first test (Procedure TEST1, Fig. 7) is based on Algorithm S-OPQR. It is very accurate $(f(\lambda)\le 4\lambda)$, however, it has relatively high complexity.

Complexity Using the complexity expression for Algorithm S-OPQR, we get $(\alpha=1, \varepsilon=1)$

$$O\left(m\log n(\log D+n)\right).$$

Accuracy By Theorem 2, if TEST1 returns FAIL then $c^* > U\equiv\lambda$; and if it returns a path $\boldsymbol{p}$, then $c(\boldsymbol{p})\le(1+\varepsilon)(U+\varepsilon L)=(1+1)(\lambda+\lambda)=4\lambda$. Obviously, $f(\lambda)\equiv c(\boldsymbol{p})$ is a valid upper bound, and we have $f(\lambda)\le 4\lambda$.

The second test (Procedure TEST2, Fig. 8) is based on a "standard" shortest path computation. It is less accurate than TEST1, but has better complexity. The idea is to bound the highest cost incurred on any single link of the optimal solution.

[6]Recall that this is assumed to be the minimal cost of any link.

```
ε-OPQR: (G(V,E), {c_l(d)}_{l∈E}, D, ε)
1 U_1 ← max_{l∈E} c_l(D/n)
2 L_1 ← cost of Shortest_st Path (G(V,E), {c_l(D)}_{l∈E})
3 [L_2, U_2] ← BOUND(TEST2, L_1, U_1)
4 [L_3, U_3] ← BOUND(TEST1, L_2, U_2)
5 return S-OPQR(G(V,E), {c_l(d)}_{l∈E}, D, L_3, U_3, ε)
```

Fig. 9. Algorithm ε-OPQR.

Complexity Computing $d_l(\lambda)$ requires $O(\log D)$ for each link. Computing the shortest path requires $O(m+n\log n)$.[7] The overall complexity is thus

$$O(m\log D+n\log n).$$

Note If G is connected then TEST2$(\max_{l\in E} c_l(D/n))$ cannot FAIL. Therefore, $\max_{l\in E} c_l(D/n)$ can be used as an initial upper bound for Algorithm BOUND with TEST2. An even better bound can be found by computing λ such that $G(V,E(\lambda))$ has an st-path, where $E(\lambda)$ is defined as $\{l|c_l(D/n)\le\lambda\}$. Such a λ can be found in $O(m\log m)$ by sorting the links and then running $O(\log m)$ connectivity tests.

Theorem 4: Procedure TEST2 is a valid test procedure with $f(\lambda)\le n\lambda$.

Proof: If a feasible path $\boldsymbol{p}$ is found by the call to SHORTEST-PATH then by definition $c_l\le\lambda$ for all $l\in\boldsymbol{p}$, implying an overall cost $c(\boldsymbol{p})\le n\lambda$. Since $\{d_l(\lambda)\}_{l\in\boldsymbol{p}}$ is a feasible partition we must have $c^*\le c(\boldsymbol{p})\le n\lambda$. In other words $f(\lambda)\le\lambda n$.

Consider now the optimal solution to Problem OPQR. If $\lambda\ge c^*$ then since $c^*\ge c_l^*$ for every $l\in\boldsymbol{p}^*$, we have

$$d_l(\lambda)\le d_l(c^*)\le d_l(c_l^*)=d_l^* \quad \forall l\in\boldsymbol{p}^*.$$

Therefore, $\sum_{l\in\boldsymbol{p}^*} d_l(\lambda)\le\sum_{l\in\boldsymbol{p}^*} d_l^*\le D$, namely $\boldsymbol{p}$ is a feasible path w.r.t. $\{d_l(\lambda)\}_{l\in E}$ and the algorithm cannot fail. Thus, if the algorithm returns FAIL then $\lambda<c^*$. ■

V. PUTTING IT ALL TOGETHER

We can now present (see Fig. 9) a fully polynomial approximation algorithm to Problem OPQR.

Complexity L_1 is a valid lower bound and TEST2(U_1) cannot return FAIL. Thus, L_1, U_1 are a valid input to Algorithm BOUND in line 3. Computing both these bounds requires $O(m+n\log n)$. Algorithm BOUND requires $O(\log\log\beta)$ calls to TEST2, where β is the ratio of the initial bounds.[8] Thus, the overall complexity up to line 3 is $O(\log\log\beta(m\log D+n\log n))$.[9]

L_2 and U_2 are valid bounds on c^* and therefore are valid input to Algorithm BOUND. Since $U_2/L_2\le 2n$ the call to Algorithm

[7]Using Dijkstra's algorithm.

[8]Note that β is bounded by the maximal cost of any single link.

[9]Even if U_1 is replaced by the better bound suggested in the note in Section IV-B the complexity of finding the initial bounds is still dominated by the rest of the algorithm.

BOUND in line 4 requires $O(\log\log n)$ calls to Procedure TEST1 and an overall complexity of $O(\log\log n(m\log n(\log D+n)))$.

L_3, U_3 are valid bounds on c^* and therefore are valid input to Algorithm S-OPQR. $U_3/L_3 \leq 8$, hence the call to Algorithm S-OPQR requires

$$O\left(\frac{m}{\varepsilon}\log\frac{8n}{\varepsilon}\left(\log D+\frac{8n}{\varepsilon}\right)\right).$$

The overall complexity is therefore

$$\begin{aligned}
&O\left((m\log D+n\log n)\log\log\beta\right.\\
&\quad + m\log n(\log D+n)\log\log n\\
&\quad \left.+\frac{m}{\varepsilon}\log\frac{n}{\varepsilon}\left(\log D+\frac{n}{\varepsilon}\right)\right)\\
&\quad = O\left(m\log D\left(\log\log\beta+\log n\log\log n\frac{1}{\varepsilon}\log(n/\varepsilon)\right)\right.\\
&\qquad +n\log n(\log\log\beta+m\log\log n)\\
&\qquad \left.+\frac{1}{\varepsilon^2}mn\log(n/\varepsilon)\right).
\end{aligned}$$

Note 1 For very small values of ε, replacing $\log(n/\varepsilon)$ by n may improve the complexity (see Remark I in Section III-B).

Note 2 The complexity can also be improved for the case of convex cost functions (see Remark 2 in Section III-B).

Correctness As explained, U_1 and L_1 are valid bounds. Using Theorem 3 and Theorem 4 we get that U_2 and L_2 are also valid bounds. Applying Theorem 3 again with TEST1, together with Theorem 1 establish the algorithm correctness.

A. *Discrete Cost Functions*

In this section, we discuss the application of our approximation techniques to the more restricted case of *discrete* cost functions. This case, which was studied by [4], admits a strictly polynomial approximation scheme, meaning that the complexity does not depend on either $\log\log\beta$ or $\log D$. We follow [4] and use the term *discrete* to refer to cost functions with at most q discrete (delay, cost) values, where q is given as input. Next, we derive an improved complexity for the solution of Problem OPQR for discrete cost functions.

We first observe that computing the inverse cost function (e.g., in line 9 of Algorithm OPQR) can be done in $O(\log q)$ instead of $O(\log D)$. This reduces the complexity of Algorithm OPQR to $O(mU(\log U + \log q))$. Alternatively, each link can be replaced by $O(q)$ links corresponding to its offered services. After this substitution, Algorithm RSP can be used with a complexity of $O(mqU)$. Our second observation is that we can reduce the number of calls to TEST2 by Algorithm BOUND. Instead of searching through the whole range of costs we can limit the search to the $O(mq)$ discrete cost values, which requires only $O(\log(mq))$ calls to TEST2. The initial sort requires additional $O(mq\log(mq))$ operations, however using techniques for searching in arrays with sorted columns [18], the additional number of operations can be reduced to $O(m\log q)$. The overall complexity, assuming $q = O((1/\varepsilon)\log(n/\varepsilon))$,[10] is

$$\begin{aligned}
&O\left((m\log q+n\log n)\log(mq)\right.\\
&\qquad\qquad \left.+m\log n(\log q+n)\log\log n+\frac{mqn}{\varepsilon}\right).
\end{aligned}$$

This is a significant (above n^2) improvement over the $O(mqn^3\log(mq)/\varepsilon)$ approximation obtained in [4].

B. *Zero and Noninteger Costs*

We shall now relax the assumption that the minimal cost on every link is at least 1. As noted in Section II-A, if there are links that have a zero cost and the graph contains cycles, then a shortest-path computation is required in every iteration of the exact pseudo-polynomial solution. This increases the complexity of Algorithm RSP by a factor of $\log n$, and adds to the complexity of all the approximations.

Both Algorithm OPQR and Algorithm L-OPQR assume a minimal cost of 1 on every link. On the other hand, these algorithms are called only through Algorithm S-OPQR, which assigns costs that cohere with this assumption. The rounding in line 3 of Algorithm S-OPQR ensures that the minimal cost assigned on any link is at least 1. The only requirement is that the scaling factor S is greater than zero. The scaling factor would be zero only if either L or ε is zero. If $\varepsilon = 0$, then we actually require an exact solution and therefore Algorithm S-OPQR cannot be used. We can still use Algorithm OPQR, as is, for acyclic graphs, or modify it (with increased complexity) to include a shortest-path computation in every iteration.

If $L = 0$, then the approximation scheme requires infinite time anyway, since in this case $\alpha = \infty$. On the other hand, for any positive L, Algorithm S-OPQR works fine, with the same complexity, even if $L \ll 1$. Also, Algorithm BOUND only requires $L > 0$, hence Algorithm ε-OPQR only requires $L_1 > 0$. The case of $c^* = 0$ can be easily checked by calling TEST2 with $\lambda = 0$. Note that, in this case, any feasible path returned by TEST2 is an (exact) optimal solution. If $c^* > 0$ but $L_1 = 0$, then some assumption (e.g., $L_1 \geq 1$) must be made. Except from its dependency on $\log\log\beta$, Algorithm ε-OPQR is totally independent of the cost values. Specifically, the costs do not need to have integer values.

VI. M-OPQ

In this section, we solve the multicast resource allocation version of Problem OPQR. We assume that the multicast tree is given and that the problem is to find the optimal resource allocation (delay partition) on it.

To illustrate this problem consider the multicast tree depicted in Fig. 10. The tree has been defined over the network of Fig. 10, in which the source is node S and the multicast target group is the set $\{C, T\}$. Assume that the end-to-end delay requirement is 120 ms. This means that the guaranteed end-to-end delay over each of the paths $S-B-T$ and $S-B-D-A-C$ should not exceed 120 ms. One way to achieve this is by choosing Gold service at links $B-D$, $B-T$, and $D-A$, Silver service at link

[10] This determines whether Algorithm OPQR or Algorithm RSP are used.

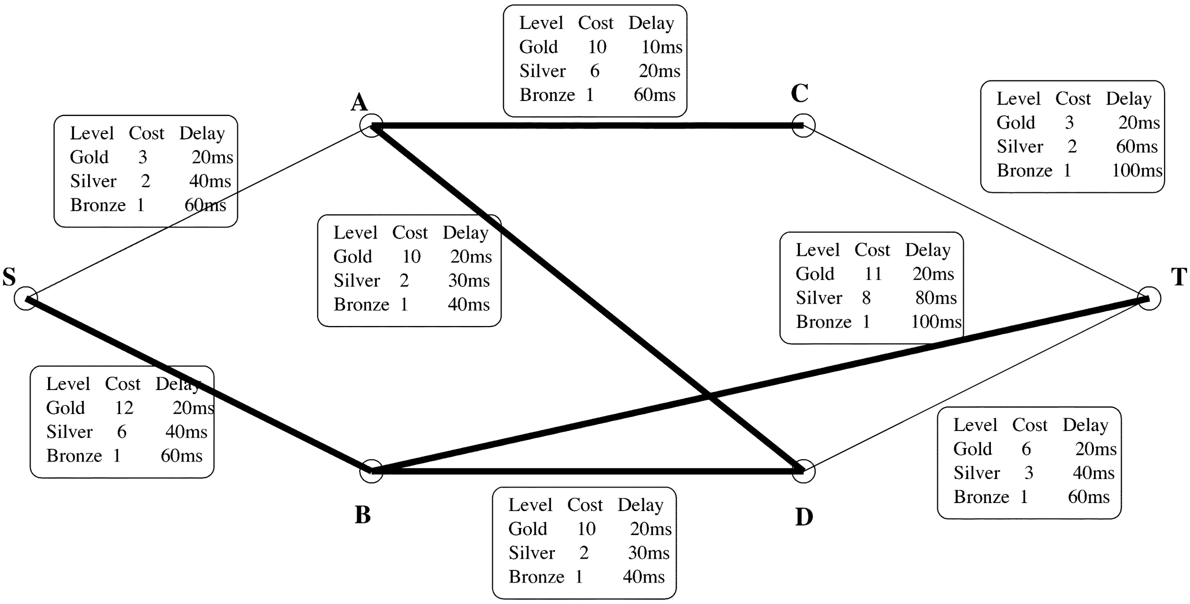

Fig. 10. Example: Multicast QoS partition.

$A-C$, and Bronze service at link $S-B$, incurring a total cost of 38 units. However, the delay requirements can be met also by choosing Gold service at $S-B$, Silver service at $A-C$, and Bronze sevice at all other links, incurring a total cost of only 21 units. Note that for the unicast partition problem defined on the path $S-B-T$, the optimal partition is different, namely Bronze at $S-B$ and Gold at $B-T$.

We denote a multicast tree by $\boldsymbol{T}$ and the multicast target group by $M=\{t_1,t_2,\ldots\}$. We denote a path from the source s to a node v by $\boldsymbol{p}^v$. The cost of a tree is defined as $c(\boldsymbol{T}) \equiv \sum_{l\in\boldsymbol{T}} c_l$. The delay of a tree is defined as the maximal delay of a path from the source to any member of the multicast group, namely $delay(\boldsymbol{T}) \equiv \max_{v\in M} delay(\boldsymbol{p}^v)$.

Problm M-OPQ—Multicast Optimal QoS Partition: Given a tree $\boldsymbol{T}$, a delay/cost *function* for each link $\{c_l(d)\}_{l\in\boldsymbol{T}}$, and an end-to-end requirement D. Find the optimal partition $\boldsymbol{d}=\{d_l\}_{l\in\boldsymbol{T}}$ that satisfies the end-to-end delay requirement $delay(\boldsymbol{T}) \le D$.

We present exact and ε-approximate solutions that apply to any integer cost functions. We assume all parameters (costs and delays) are integers.

A. Exact Solution

We solve Problem M-OPQ using the same techniques we used for Problem OPQR. We start with Algorithm M-OPQ (Fig. 11), which is an exact pseudo-polynomial solution. Without loss of generality, we assume a binary tree. The tree can be made binary by splitting each nonbinary node with x children to $x-1$ binary nodes. This adds a constant factor to the complexity. For a tree $\boldsymbol{T}$, let $\boldsymbol{T}^y$ and $\boldsymbol{T}^z$ be the corresponding left and right subtrees of $\boldsymbol{T}$. As before, $n, m(=n-1)$ denote the number of nodes and links in the tree. The height (depth) of the tree is denoted by H.

Algorithm M-OPQ (Fig. 11) finds the optimal partition on the whole tree by combining optimal partitions on the sub-trees. $\boldsymbol{W}$, $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{Z}$ are tables of size U which hold the best delay

M-OPQ*:* $(\boldsymbol{T}, \{c_l(d)\}_{l\in\boldsymbol{T}}, D, U)$

1 $\boldsymbol{Y} \leftarrow$M-OPQ$(\boldsymbol{T^y}, \{c_l(d)\}_{l\in\boldsymbol{T^y}}, D, U)$
2 $\boldsymbol{Z} \leftarrow$M-OPQ$(\boldsymbol{T^z}, \{c_l(d)\}_{l\in\boldsymbol{T^z}}, D, U)$
3 $\boldsymbol{W} \leftarrow$MERGE$(\boldsymbol{Y}, \boldsymbol{Z}, U, \max)$
4 for $c = 1 \ldots U$
5 $\quad \boldsymbol{X}(c) \leftarrow \min\{d \mid c_x(d) \le c\}$
6 $\boldsymbol{X} \leftarrow$MERGE$(\boldsymbol{X}, \boldsymbol{W}, U, \boldsymbol{\Sigma})$
7 if $\boldsymbol{X}(U) > D$
8 $\quad$ return FAIL(exit recursion)
9 (else) return $\begin{cases} \boldsymbol{X} & \text{(all except root)} \\ \min\{c \mid \boldsymbol{X}(c) \le D\} \text{ and the corresponding partition.} & \text{(root only)} \end{cases}$

Procedure MERGE*:* $(\boldsymbol{A}(c), \boldsymbol{B}(c), U, \mathbf{op})$

1 for $c = 0 \ldots U$
2 $\quad \boldsymbol{\mathcal{D}}(c) = \min_{1\le x\le c} \mathbf{op}\{\boldsymbol{A}(x), \boldsymbol{B}(c-x)\}$
3 return $\boldsymbol{\mathcal{D}}$

Fig. 11. Algorithm M-OPQ.

achieved for each and every cost. Such a table is computed for each sub-tree and for each link. The algorithm recursively merges tables of sub-trees (and links) until it reaches the root of the tree. As indicated in Fig. 11, the table $\boldsymbol{Y}$ contains the best delay achieved, for each cost c between 1 and the upper bound U with respect to the left subtree of $\boldsymbol{T}$, including the cost of the link that connects the subtree to the rest of the tree. Similarly, the table $\boldsymbol{Z}$ contains the best delay achieved for each cost with respect to the right subtree of $\boldsymbol{T}$ (and its connecting link. The values in this table are calculated recursively, so when we compute the table $\boldsymbol{X}$ containing the best delay achieved for each cost with respect to the full tree $\boldsymbol{T}$, their values are already available. The first step (Line 3) is combining tables $\boldsymbol{Y}$ and $\boldsymbol{Z}$, into a single table; this table ($\boldsymbol{W}$) contains for each cost c the minimum over all $i \le c$ of the maximum of $Y(i)$ and $Z(c-i)$ (see the code for Procedure MERGE in Fig. 11). The next step is to build the table $\boldsymbol{X}$ containing the delay data for the link x going up from the tree $\boldsymbol{T}$. Then, $\boldsymbol{X}$ is updated. This is done by computing for each cost value c the minimum over all $i \le c$ of the

SM-OPQ: $(\boldsymbol{T}, \{c_l(d)\}_{l\in\boldsymbol{T}}, D, U, L, \varepsilon)$
1 $S \leftarrow \frac{L\varepsilon}{n+1}$
2 for each $l \in \boldsymbol{T}$
3 define $\tilde{c}_l(d) \equiv \lfloor c_l(d)/S \rfloor + 1$
4 $\tilde{U} \leftarrow \lceil U/S \rceil + n$
5 return M-OPQ$\left(\tilde{\boldsymbol{T}}, \{\tilde{c}_l(d)\}_{l\in\boldsymbol{T}}, D, \tilde{U}, \varepsilon\right)$

Fig. 12. Algorithm SM-OPQ.

sum of $W(i)$ and $X(c-i)$ (see the code for Procedure MERGE in Fig. 11). Clearly, if both $\boldsymbol{Z}$ and $\boldsymbol{Y}$ contains the correct values for the subtrees, $\boldsymbol{X}$ contains the correct minimum delay value for the combined tree. Thus, when eventually we calculate the cost for the root, we get the correct minimal delay for the entire tree, $\boldsymbol{T}$.

Complexity Each call to Procedure Merge requires $O(U^2)$. There are two such calls for every node in the tree. Calculating $X(c)$ in line 5 requires $O(U \log D)$. The overall complexity is therefore $O(nU(\log D + U))$. A distributed algorithm which uses parallel calls to sub-trees (see [4] for detailed description) has a time complexity of $O(HU(\log D + U))$.

B. Approximation

We can use Algorithm S-OPQR to find an ε-approximation to Problem M-OPQ. To that end, it suffices to replace the call to Algorithm L-OPQR in line 5 of Algorithm S-OPQR with a call to Algorithm M-OPQ. Algorithm SM-OPQ (Fig. 12) is the modified version.

Complexity The complexity is dominated by the call to Algorithm M-OPQ (line 5), which requires $O(n\tilde{U}(\log D + \tilde{U}))$, where $\tilde{U} = O(\alpha n/\varepsilon)$, as in Algorithm S-OPQR. Thus, the overall complexity is

$$O\left(n\frac{\alpha n}{\varepsilon}\left(\log D + \frac{\alpha n}{\varepsilon}\right)\right) = O\left(\frac{\alpha n^2}{\varepsilon}\left(\log D + \frac{\alpha n}{\varepsilon}\right)\right).$$

The overall complexity for the distributed case is

$$O\left(H\frac{\alpha n}{\varepsilon}\left(\log D + \frac{\alpha n}{\varepsilon}\right)\right).$$

Note As for Algorithm S-OPQR, L does not have to be a valid lower bound, but it affects the accuracy of the solution.

Theorem 5: If Algorithm SM-OPQ returns FAIL then $c^* > U$. Otherwise the partition $\boldsymbol{d}(\boldsymbol{T})$ returned is a feasible solution to Problem M-OPQ and

$$c\left(d(\boldsymbol{T})\right) \leq \min\{c^*, U\} + \varepsilon L.$$

The proof is similar to that of Theorem 2.

We can find lower and upper bounds to Problem M-OPQ using Algorithm BOUND and apply Algorithm ε-OPQR with a few modifications (see Fig. 14). First, the initial bounds are

$$U_1 = \max_{l\in\boldsymbol{T}} c_l(D/H), L_1 = \sum_{l\in\boldsymbol{T}} c_l(D).$$

Second, we replace the call to Algorithm S-OPQR in Procedure TEST1 with a call to Algorithm SM-OPQ; and third, we use the following Procedure TEST3M (Fig. 13) instead of Procedure TEST2.

TEST2M: (λ)
1 for each $l \in \boldsymbol{T}$
2 $d_l(\lambda) \equiv \min\{d \mid c_l(d) \leq \lambda\}$
3 if $delay(\boldsymbol{T}) \leq D$ then
4 return $H\lambda$
5 else
6 return FAIL

Fig. 13. Algorithm TEST2M.

ε-M-OPQ: $(\boldsymbol{T}, \{c_l(d)\}_{l\in\boldsymbol{T}}, D, \varepsilon)$
1 $U_1 \leftarrow \max_{l\in\boldsymbol{T}} c_l(D/H)$
2 $L_1 \leftarrow \sum_{l\in\boldsymbol{T}} c_l(D)$
3 $[L_2, U_2] \leftarrow$ BOUND(TEST2M, L_1, U_1)
4 $[L_3, U_3] \leftarrow$ BOUND(TEST1, L_2, U_2)
5 return SM-OPQ($\boldsymbol{T}, \{c_l(d)\}_{l\in\boldsymbol{T}}, D, L_3, U_3, \varepsilon$)

Fig. 14. Algorithm ε-M-OPQ.

Complexity $O(n \log D)$; and $O(H \log D)$ for a distributed implementation.

The fully polynomial approximation algorithm to Problem M-OPQ is presented in Fig. 14.

Complexity Combining the complexity expressions of Algorithm SM-OPQ and the modified test procedures, we get the overall complexity of finding an ε-approximation to Problem M-OPQ:

$$O\left(n \log D \log\log\beta + n^2(\log D + n)\log\log H + \frac{n^2}{\varepsilon}(\log D + \frac{n}{\varepsilon})\right)$$

where

$$\beta = \frac{\max_{l\in\boldsymbol{T}} c_l(D/H)}{\sum_{l\in\mathrm{T}} c_l(D)}.$$

For the distributed case, the complexity is

$$O\left(H \log D \log\log\beta + nH(\log D + n)\log\log H + \frac{nH}{\varepsilon}\left(\log D + \frac{n}{\varepsilon}\right)\right).$$

Correctness Similar to the unicast case.

VII. CONCLUSION

In this paper, we studied efficient approximations to optimal routing and resource allocation in the context of performance-dependent costs.

We established fully polynomial approximation schemes for the following problems.

Problem OPQR The combined optimal routing and partition problem for unicast connections.

Problem M-OPQ Optimal partition of end-to-end QoS requirements on a multicast tree, including a distributed implementation. We also presented improved results for the two important special cases of convex cost functions and discrete cost functions.

We presented the first fully polynomial approximation scheme (FPAS) for Problem OPQR that is not limited to either acyclic networks or links with nonzero costs. Our approximations are valid for general costs, and in particular to nonconvex cost functions. In addition, we presented the first FPAS for Problem M-OPQ that applies to general cost functions.

Our results significantly improve upon previous results, in every context of cost functions that has been investigated. Specifically:

General costs The approximation scheme of [14] achieves an overall complexity of $O(X(mn/\varepsilon)\log\log(nC^{\max}))$, where $C^{\max}$ is a trivial upper bound on the cost of any link and $X = \min\{D, (\log C^{\max}/\varepsilon) + \log D, (n/\varepsilon) + \log D\}$. Our approximation scheme provides a significant improvement in terms of computational complexity. The exact comparison involves a cumbersome algebra and is thus omitted; as an indication to the extent of the improvement, we note that our approximation outperforms that of [14] by a factor of more than either $\log\log\beta$ or n/ε, depending on the relative order of magnitude of the input parameters.

Convex costs For Problem OPQR, efficient approximations for convex cost functions were studied in [8]. However, that approximation requires several more assumptions on the cost functions, e.g., that the maximal cost on any link is bounded. Those assumptions were reasonable in the context studied in [8], namely uncertainty of network parameters, but they are too restrictive for the general case considered here. In contrast, our results do not rely on those assumptions. On the other hand, when only QoS *partitioning* is considered (i.e., the routing is given), the convexity assumption allows for *exact* polynomial solutions for both unicast and multicast [5]; moreover, the (exact) solution of [5] for Problem M-OPQ outperforms our approximation also in terms of complexity.

Discrete costs We improved the results of [4] for discrete cost functions (see Section V-A). Our approximation has a significantly better (above n^2) time complexity for both unicast and multicast connections.

Future research should focus on the open problem of multicast routing in this framework. Future work should also consider the application of our methods to specific cost functions, in particular those that arise in practical QoS applications. Such an investigation would potentially allow for more efficient approximations. We also believe that simple cases (e.g., uniform or linear cost functions) should simplify the task of multicast routing. The problems studied in this work handle a single connection request. This corresponds to the typical scenario in which sessions appear sequentially hence should be handled one at a time. Nevertheless, the case in which multiple concurrent sessions can be handled simultaneously also deserves attention and is left for future work.

REFERENCES

[1] G. Apostolopoulos, R. Guérin, S. Kamat, A. Orda, T. Przygienda, and D. Williams, "QoS Routing mechanisms and OSPF extensions," Internet RFC 2676, Aug. 1999.

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

[3] Z. Wang and J. Crowcroft, "Quality-of-service routing for supporting multimedia applications," *IEEE J. Sel. Areas Commun.*, vol. 14, no. 7, pp. 1234–1288, Sep. 1996.

[4] D. Raz and Y. Shavitt, "Optimal partition of QoS requirements with discrete cost functions," in *Proc. IEEE INFOCOM 2000*, Tel Aviv, Israel, Mar. 2000, pp. 613–622.

[5] D. H. Lorenz and A. Orda, "Optimal partition of QoS requirements on unicast paths and multicast trees," in *Proc. IEEE INFOCOM'99*, New York, Mar. 1999, pp. 246–253.

[6] R. Nagarajan, J. F. Kurose, and D. Towsley, "Allocation of local quality of service constraints to meet end-to-end requirements," presented at the IFIP Workshop on the Performance Analysis of ATM Systems, Martinique, Jan. 1993.

[7] T. Ibaraki and N. Katoh*, Resource Allocation Problems, the Foundations of Computing*. Cambridge, MA: MIT Press, 1988.

[8] D. H. Lorenz and A. Orda, "QoS routing in networks with uncertain parameters," *IEEE/ACM Trans. Netw.*, vol. 6, no. 6, pp. 768–778, Dec. 1998.

[9] V. Firoiu and D. Towsley, "Call admission and resource reservation for multicast sessions," in *Proc. IEEE INFOCOM'96*, San Francisco, CA, Apr. 1996, pp. 94–101.

[10] M. Kodialam and S. Low, "Resource allocation in a multicast tree," in *Proc. IEEE INFOCOM'99*, New York, Mar. 1999, pp. 262–266.

[11] D. S. Hochbaum, "Lower and upper bounds for the alloction problem and other nonlinear optimization problems," *Math. Oper. Res.*, vol. 19, no. 2, pp. 390–409, 1994.

[12] V. P. Kompella, J. C. Pasquale, and G. C. Polyzos, "Multicast routing for multimedia communication," *IEEE/ACM Trans. Netw.*, vol. 1, no. 3, pp. 286–292, Jun. 1993.

[13] R. Guérin and A. Orda, "QoS-based routing in networks with inaccurate information: theory and algorithms," *IEEE/ACM Trans. Netw.*, vol. 7, no. 3, pp. 350–364, Jun. 1999.

[14] F. Ergün, R. Sinha, and L. Zhang, "QoS routing with performance-dependent costs," in *Proc. IEEE INFOCOM 2000*, Tel Aviv, Israel, Mar. 2000, pp. 137–146.

[15] R. Hassin, "Approximation schemes for the restricted shortest path problem," *Math. Oper. Res.*, vol. 17, no. 1, pp. 36–42, Feb. 1992.

[16] D. H. Lorenz and D. Raz, "A simple efficient approximation scheme for the restricted shortest path problem," *Oper. Res. Lett.*, vol. 28, no. 5, pp. 213–219, Jun. 2001.

[17] F. A. Kuipers, A. Orda, D. Raz, and P. Van Mieghem, "A comparison of exact and ε-approximation algorithms for constrained routing," in *Proc. IFIP Networking 2006*, Coimbra, Portugal, May 2006, pp. 197–208.

[18] G. N. Frederickson and D. B. Johnson, "The complexity of selection and ranking in $X+Y$ and matrices with sorted columns," *J. Comput. Syst. Sci.*, vol. 24, pp. 197–208, 1982.

Dean H. Lorenz received the B.Sc. degree (*summa cum laude*) in computer engineering and the Ph.D. degree in electrical engineering from the Technion–Israel Institute of Technology, Haifa, Israel, in 1995 and 2004, respectively.

He was an officer in the Israeli Defense Force and a team member in a project dealing with distributed real-time information system (1986–1991), a Systems Analyst and Consultant at Sysware Technologies, Haifa, Israel (1993–1995), and a Senior Developer at Akamai Technologies, Cambridge, MA (1999–2000). He was a summer visitor at the Mathematics of Networks Department at Lucent Bell Laboratories, Murray Hill, NJ (1998) and a DIMACS visitor at the Department of Network and Services Management, Lucent Bell Labs, Holmdel, NJ (1999). Since 2001, he has been a Researcher at the System Architecture Group in the Scale-out System Technologies Department of the IBM Research Laboratory in Haifa, Israel. His research includes QoS routing, multicasting and optimization (Technion), grid computing, autonomous computing, group services, and scale-out technologies (IBM).

Dr. Lorenz received the Gutwirth Award for outstanding distinction in 1998 and 1999.

Ariel Orda (S'84–M'92–SM'97–F'06) 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. His current research interests include network routing, survivability, QoS provisioning, wireless networks, the application of game theory to computer networking and network pricing.

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, the Jacknow Award for Excellence in Teaching, an ICNP 2004 Best Paper Award, and the 2005 Klein Award for Excellence in Research. He served as Program co-chair of IEEE INFOCOM 2002, and is an Editor of the IEEE/ACM TRANSACTIONS ON NETWORKING and of the *Journal of Computer Networks*.

Danny Raz (M'98) received the Doctoral degree from the Weizmann Institute of Science, Israel, in 1995.

From 1995 to 1997, he was a Postdoctoral Fellow at the International Computer Science Institute, (ICSI) Berkeley, CA, and a visiting Lecturer at the University of California, Berkeley. Between 1997 and 2001, he was a Member of Technical Staff at the Networking Research Laboratory at Bell Labs, Lucent Technologies. In October 2000, he joined the faculty of the Computer Science Department at the Technion, Israel. His primary research interest is the theory and application of management-related problems in IP networks.

Yuval Shavitt (S'89–M'96–SM'00) received the B.Sc. degree in computer engineering (*cum laude*), the M.Sc. degree in electrical engineering, and the D.Sc. degree from the Technion–Israel Institute of Technology, Haifa, Israel, in 1986, 1992, and 1996, respectively.

From 1986 to 1991, he served in the Israel Defense Forces, first as a System Engineer and the last two years as a Software Engineering Team Leader. After graduation, he spent a year as a Postdoctoral Fellow at the Department of Computer Science at Johns Hopkins University, Baltimore, MD. Between 1997 and 2001, he was a Member of Technical Staff at the Networking Research Laboratory at Bell Labs, Lucent Technologies, Holmdel, NJ. Since October 2000, he has been a faculty member in the School of Electrical Engineering at Tel Aviv University. His recent research focuses on Internet measurement, mapping and characterization, and QoS routing and traffic engineering.

Dr. Shavitt served as a TPC member for IEEE INFOCOM 2000–2003 and 2005, IWQoS 2001 and 2002, ICNP 2001, IWAN 2002–2005, and other conferences, and on the executive committee of INFOCOM 2000, 2002, and 2003. He was an editor of *Computer Networks*, and served as a guest editor for the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS and the *Journal of the World Wide Web*.