QoS Routing: The Precomputation Perspective[†]

Ariel Orda and *Alexander Sprintson*
Department of Electrical Engineering
Technion—Israel Institute of Technology
Haifa 32000, Israel
{ariel@ee,spalex@tx}.technion.ac.il

Abstract

A major algorithmic challenge posed by QoS routing is the need to promptly identify a suitable path upon a connection request, while at the same time ensure that the selected path is satisfactory, both in terms of the connection's QoS requirements, as well as in terms of the global utilization of network resources. In many practical cases, a *precomputation* scheme offers a suitable solution to the problem: a background process prepares a data base, which enables to identify a suitable path upon each connection request, through a simple, fast, procedure.

While much work has been done in terms of path selection algorithms, the precomputation perspective got little attention. Simplistic adaptations of standard algorithms turn to be inefficient. Accordingly, we consider the precomputation perspective, focusing on two major settings of QoS routing. The first is the (practically important) special case where the QoS constraint is of the "bottleneck" type, e.g. a bandwidth requirement, and network optimization is sought through hop minimization. For this setting, the standard Bellman-Ford algorithm offers a straightforward precomputation scheme. However, we show that, by exploiting the typical hierarchical structure of large-scale networks, one can achieve a substantial improvement in terms of computational complexity. Then, we turn to consider the more general setting of "additive" QoS constraints (e.g., delay) and general link costs. As the routing problem becomes NP-hard, we focus on ε-optimal approximations, and derive a precomputation scheme that offers a major improvement over the standard approach.

Keywords

QoS, Routing, Precomputation, Hierarchical 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, and, moreover, how to meet that goal in a network-wide efficient manner. The establishment of efficient QoS routing schemes is, undoubtedly, one of the major building blocks in such architectures. Indeed, QoS routing has been the subject of several studies and proposals (see, e.g. [1], [5], [7], [10], [11], [13], [15], [17], [18], [16] and references therein). It has been recognized that the establishment of an efficient QoS routing scheme poses several complex challenges.

QoS routing is, in general, a complex problem, due to several reasons. One complication is the need to deal with several QoS requirements, each potentially imposing some constraints on the path choice. Then, beyond the need to address the requirements of individual connections, QoS routing needs to consider also the global use of network resources, since meeting the requirements of a QoS request implies the reservation of sufficient resources, e.g., bandwidth, along the selected path. Finally, the identity of the required ("optimal") path is connection-dependent, yet executing the path search procedure for each connection may turn out to be computationally prohibitive. Nonetheless, the above obstacles notwithstanding, QoS routing is facilitated in many practical settings by the following. First, while a connection may pose several QoS requirements, it turns out that these often translate mainly into a *bandwidth* requirement [2], [3]. Bandwidth, in turn, belongs to the class of "bottleneck" path requirements, which are much easier to handle than "additive" requirements, such as delay, loss and jitter [10], [13], [14]. As for global network optimization, often it turns out that much can be achieved by employing the simple criterion of *hop minimization* [2], [4]; indeed, a consequence of the need to reserve resources such as bandwidth on *each* link of the connection's path is that, with fewer hops one consumes fewer resources. As a result, hop-constrained path optimization has emerged as an important problem in several recent proposals for IP-oriented QoS routing protocols [7]. Luckily, hop minimization turns out to be an optimization criterion that is relatively easy to handle. Lastly, to avoid having to perform a separate path computation for each new

[†]This research was supported by the Consortium for Wideband Communications, administered by the Israeli Ministry of Industry and Commerce.

request, several proposed QoS routing protocols are based on *precomputing* paths for all possible QoS requirements [7], [2], [4] as a background process, hence considerably reducing the computational load upon each connection request.

The above observations form the foundations of the present study. In particular, we focus on path precomputation, taking the view that it is a highly desirable, and at times necessary property, of an efficient QoS routing scheme. As shall be demonstrated, many of the algorithmic tools that are often proposed as building blocks for QoS routing were not designed with path precomputation in mind, and better solutions can be found when this property is required. This requirement, namely to efficiently precompute optimal paths for a whole range of (QoS) requirements, effectively opens a new area of research. A first step in that direction has been done in [9], which investigated the problem of precomputing paths of maximal bandwidth for each possible hop-count value; that problem was termed there as Problem *All-Hops (AHOP)*. While a trivial solution to that problem is offered by the standard Bellman-Ford shortest-path scheme [6], [9] presented an algorithm whose worst-case bound is lower; yet, the improvement is achieved only in dense (high connectivity) topologies, while communication networks usually have a sparse topology. On the other hand, it has been observed that exploiting the particular topological structure of large-scale broadband networks can often facilitate the establishment of more efficient solutions to (QoS) routing problems [10], [16]. Accordingly, in this study we consider the *hierarchical structure* that is typical of large-scale networks, and indeed obtain solutions for Problem AHOP which offer a substantial improvement, in terms of computational complexity, upon the standard (Bellman-Ford's) scheme. Then, we turn our attention to the harder case of *additive* QoS requirements (such as delay, jitter and packet loss) and *general* (additive) path optimization criteria, beyond hop minimization. The respective problem becomes a variant of the *Restricted Shortest Path (RSP)* problem, which is known to be NP-hard [8]. Some general approximation schemes that are ε-optimal have been proposed (see, e.g., [12] and references therein). However, those schemes have not been designed with precomputation in mind, and, consequently, are not adequate when precomputation is sought. Accordingly, in the present study we establish an approximation scheme that offers both efficient solutions as well as efficient performance, for precomputing "optimal" (minimum cost) paths for all possible values of an additive QoS requirement.

The rest of the paper is organized as follows. First, in Section II we formulate the network model and formally state the problems that we consider. Next, in Section III we consider the problem of hop minimization with bottleneck QoS constraints in hierarchical networks: first, in Subsection III-A, we formulate the concept of hierarchical topologies; then, we present and analyze our precomputation scheme, which is composed of two phases: the first,"background", phase is considered in Subsection III-C, while the second, "on-demand", phase is considered in Subsection III-D. The scheme is compared with other (standard) alternatives in Subsection III-E, and its advantages are discussed. Section IV concerns extension of our BH-HIE scheme to handle "all-to-all" (rather than "one-to-all") routing problems. In Section V-B we discuss application of our findings in an environment, such as PNNI's, where nodes are presented with only an *aggregated* image of the (real) topology. Next, in Section VI we analyze "dual" problems, where one is given a *cost budget*, and needs to determine a *minimum weight* path, among those that obey the budget. In Section VII we establish improved precomputation schemes for Problem BH-RSP in additional topologies of special interest, beyond the hierarchical class. Next, in Section VIII we consider additive QoS constraints and general (additive) path costs. Here too, we present and analyze a two-phase precomputation scheme, and establish its advantage over standard alternatives. Finally, conclusions appear in Section IX.

II. Model and Problem Formulation

This section formulates the general model and main problems addressed in this paper. We begin with the definition of a *general* communication network; a definition of a specific class, namely *hierarchical* networks, will be introduced in the next section.

A *network* is represented by a directed graph $G(V,E)$, where V is the set of nodes and E is the set of links. Let $N=|V|$ and $M=|E|$. A *path* is a finite sequence of nodes $\mathbf{p}=(v_0,v_1,...,v_h)$, such that, for $0\leq n\leq h-1$, $(v_n,v_{n+1})\in E$; $h=|\mathbf{p}|$ is then said to be the *number of hops* (*or hop count)* of $\mathbf{p}$. A path is *simple* if all its nodes are distinct. Let H be the maximum possible hop count of any simple path $\mathbf{p}$ in G which may be considered for routing purposes. Obviously, $H\leq N-1$, and it is much smaller in many typical network topologies; moreover, H is often restricted to a relatively small value by network control.

For concreteness of exposition, we consider a *link state* routing environment, where the source node has an image of the entire network. Each link $e\in E$ is assigned a positive *weight* $w(e)$, whose significance depends on the type of considered QoS requirement. For example, when the QoS requirement is an upper bound on the end-to-end delay, the link weight is its delay; whereas when a bandwidth requirement is considered, the link weight $w(e)$ is reciprocal to its available bandwidth $b(e)$ i.e. $w(e)=\frac{1}{b(e)}$. Accordingly, the *path weight* $W(\mathbf{p})$ of a path $\mathbf{p}$ is defined differently for additive metrics, such as delay, than for bottleneck metrics, such as bandwidth. Specifically:

Definition 1: When link weights $w(e)$ constitute an *additive* metric, the *weight* $W(\mathbf{p})$ of a path $\mathbf{p}$ is defined as the sum of weights of its links, namely:

- the weight of an empty path is $0 : W(\emptyset) = 0$;
- given a nonempty path $\mathbf{p}$, $W(\mathbf{p}) = \sum_{e \in \mathbf{p}} w(e)$.

Definition 2: When link weights $w(e)$ constitute a *bottleneck* metric, the *weight* $W(\mathbf{p})$ of a path $\mathbf{p}$ is defined as the weight of its worst link, namely:

- the weight of an empty path is $0 : W(\emptyset) = 0$;
- given a nonempty path $\mathbf{p}$, $W(\mathbf{p}) = \max_{\mathbf{e} \in \mathbf{p}} \{w(e)\}$;

We can define the notion of a path that is "best" when only path weights are considered, i.e.:

Definition 3: A *minimum-weight* path between two nodes s and d is a path $\mathbf{p} = \{s, ..., d\}$ whose weight is no larger than that of any other path between those nodes.

Obviously, a minimum-weight path has the best performance with respect to the QoS requirement that is captured by the link weight metric; for instance, it is a path with minimum delay or maximum bandwidth. Minimum-weight paths can be efficiently found by Dijkstra's shortest-path algorithm, in $O(N \log N + M)$ computational complexity [6]. Alternatively, for bottleneck metrics, one can employ a binary search on the solution space, whose respective complexity is $O(M \log k)$, where k is the number of different weights assigned to the network links (hence $k \leq M$). Obviously, if the minimum-weight path fails to meet the connection's QoS requirement, then so does any other path. However, when the minimum-weight path does meet the QoS requirement, it is often not the "right" choice, as it may be wasteful in terms of global network usage, e.g., it may have a large number of hops or use "expensive" links.

Therefore, the goal of QoS routing is to identify a path that satisfies a given QoS requirement while consuming as few resources as possible. Since the amount of the resources consumed on a path depends to a large extent on the number of its links, the path hop count is considered to be a good criterion for estimating the path quality in terms of global resource utilization. When the hop count criterion is not satisfactory, one can define some *link cost* metric $c(e)$ that estimates the quality of each link e in terms of resource utilization; such a cost may depend on various factors, e.g., the link's available bandwidth, its location, etc. The *path cost* is then defined as the sum of the costs of its links, namely:

Definition 4: Given a path $\mathbf{p}$, its *cost* $C(\mathbf{p})$ is:

$$C(\mathbf{p}) = \sum_{e \in \mathbf{p}} c(e).$$

In the present study we shall consider both cases of global utilization criteria, namely hop count and general (integer) link costs. Note that the former is a special case of the latter.

We are now ready to formulate the main problems that are considered in this study. Given a connection request between a source node $s \in V$ to a destination node $d \in V$ with some QoS requirements, and given network utilization preferences as captured by some link costs, the goal of the QoS routing scheme is to identify a path $\mathbf{p}$ between s and d, which meets the QoS requirements at minimum cost. This can be formulated as a *restricted shortest path (RSP)* problem:

Problem RSP (Restricted Shortest Path): Given are a source node s, a destination node d and a *QoS requirement* $\hat{w}$. Find a path $\hat{\mathbf{p}}$ from s to d such that:

1. $W(\hat{\mathbf{p}}) \leq \hat{w}$,
2. $C(\hat{\mathbf{p}}) \leq C(\mathbf{p})$ for every other path $\mathbf{p}$ that satisfies the restriction $W(\mathbf{p}) \leq \hat{w}$,
3. there does not exist another path $\tilde{\mathbf{p}}$, for which $C(\tilde{\mathbf{p}}) = C(\mathbf{p})$ and $W(\tilde{\mathbf{p}}) < W(\hat{\mathbf{p}})$.

Note that the third requirement is not part of the standard definition of the RSP problem; we introduce it since, if there exist more than a single solution to the standard problem, we would typically prefer one that offers better performance. We refer to a solution of Problem RSP as a *$\hat{w}$-weight constrained optimum path from s to d.*

For additive weights and general costs, Problem RSP is intractable, i.e., NP-hard [8]. However, there exist pseudo-polynomial solutions, based on dynamic programing, which give rise to fully polynomial approximation schemes (FPAS), whose computational complexity is reasonable (see, e.g., [12] and references therein).

As mentioned in the Introduction, many QoS routing problems consist of identifying, for each connection request, a path of minimum hops that still meets the connection's bandwidth requirement. In other words, the path weight is a "bottleneck" metric, and its cost is equal to its number of hops. Effectively, these problems can be formulated as variants of Problem RSP, for which (i) weights are of the bottleneck type and (ii) links have equal costs; each of these two simplifications renders Problem RSP to be tractable. A main focus of this study is to provide efficient precomputation schemes for this class of problems, whose formal definition is presented next.

Problem BH-RSP (Bottleneck weight Hop cost RSP): Given are a source node s, a destination node d and a bottleneck *QoS requirement* $\hat{w}$. Find a path $\hat{\mathbf{p}}$ from s to d such that:
1. $W(\hat{\mathbf{p}}) \leq \hat{w}$,
2. $|\hat{\mathbf{p}}| \leq |\mathbf{p}|$ for every other path $\mathbf{p}$ that satisfies the restriction $W(\mathbf{p}) \leq \hat{w}$,
3. there does not exist another path $\tilde{\mathbf{p}}$, for which $|\tilde{\mathbf{p}}| = |\mathbf{p}|$ and $W(\tilde{\mathbf{p}}) < W(\hat{\mathbf{p}})$.

Here too, the last requirement was added in order to prefer, among several solutions, one that offers better performance.

As mentioned in the Introduction, QoS routing can often be considerably facilitated by means of employing a *precomputation scheme*, which performs the path search *a priori* for any possible connection request. Such a scheme comprises of two phases: the first phase (pre-)computes a suitable path for any possible QoS requirement; the second provides a (fast) solution upon each connection request[1]. The fist phase, which incurs the main computational burden, is run as a background process, which needs to be activated only upon a change in the network state. Therefore, such schemes offer a significant reduction in computational load whenever the rate of connection requests is higher than that of changes in the network state, which is the case in many practical settings. Precomputation schemes for equal link costs (i.e., minimum hops) were investigated in [2] and [9], both for bottleneck as well as additive weights. [2] indicated that the Bellman-Ford algorithm offers a simple precomputation scheme, by "inverting" the roles of the constraint (QoS requirement) and the optimization criterion (hops). This way, the Bellman-Ford scheme computes a minimum weight for each possible hop count; upon a connection request, then, one would choose the minimum hop value for which the corresponding path meets the connection's QoS requirement. Accordingly, we define a *h-hop constrained optimal path* to be a path of minimum weight among all paths from a source s to a destination d with hop count of at most h. The All Hops Optimum Path problem was then formulated in [9] as follows.

Problem AHOP (All Hops Optimal): Given are a graph $G = (V, E)$, a source node $s \in V$ and a maximum hop count H, $H < N$. Find, for each hop value h, $1 \leq h \leq H$, and each destination node $d \in V$, an h-hop constrained optimal path between s and d.

In [9] it was shown that, for general topologies and *additive* weights, it is not possible to improve upon the Bellman-Ford solution in terms of the worst-case computational complexity. For *bottleneck* weights, [9] provided an alternative scheme that does improve upon Bellman-Ford's, in terms of the worst-case bound; yet, when the topology is sparse, as is typically the case in communication networks, the solution of [9] is inferior to Bellman-Ford's. It remained an open question whether one can propose better precomputation scheme for typical network topologies; this is the subject of Section III.

We shall also consider the precomputation perspective in the context of *additive* QoS requirements and *general* path costs. Obviously, in this case precomputation of exact solutions is intractable, since so is the basic underlying (RSP) problem. Therefore, we resort to precomputing approximated, namely ε-optimal, solutions; this is the subject of Section VIII.

III. Precomputation Scheme for Problem BH-RSP in Hierarchical Networks

In [9] a precomputation scheme for Problem BH-RSP was proposed, which consisted of solving Problem AHOP during the first phase. Since the solution of Problem AHOP fully precomputes all paths (for all possible bandwidth requirements), the second phase just consisted of searching for the solution in the data base produced by the first phase, according to the QoS requirement of the incoming connection request. As a result, the computational complexity incurred by the second phase was just $O(\log H + |\mathbf{p}|)$, where $|\mathbf{p}|$ is the hop count of the identified solution. We refer to this precomputation scheme as the AHOP-based scheme.

As mentioned in [9], the Bellman-Ford shortest path algorithm provides a simple scheme for solving Problem AHOP, with a computational complexity of $O(MH)$; for a general (dense) topology, that bound can grow to be as large as $O(N^2H)$. For bottleneck weight metrics (in other words, for Problem BH-RSP), [9] provided an alternative scheme, whose computational complexity is $O(\frac{N^2}{\log N}H)$; evidently, the latter outperforms Bellman-Ford's in dense topologies, i.e., when $M > \frac{N^2}{\log N}$, but not in sparse topologies, which are the typical setting of communication networks.

It remained an open question whether one can improve upon the latter in typical network topologies. In this section we demonstrate that, by exploiting the hierarchical structure that is typical of large-scale networks, one can establish a precomputation scheme for Problem BH-RSP, which offers a significant improvement upon the above solutions.

We begin by formulating the hierarchical network model under consideration, which is inspired by the ATM PNNI recommendations [1]. Next, by exploiting the properties of that model, we establish the required precomputation

[1] More precisely, the first phase needs to prepare a data base, with which the second phase can easily retrieve the required path.

scheme.

Specifically, the rest of the section is organized as follows. First, we introduce some terminology and formulate the hierarchical model. Then, we construct an auxiliary procedure, termed Procedure CLUSTER, which is a main building block of our precomputation scheme. Next, we describe the precomputation algorithm, which constitutes the first phase of our scheme, and then present Procedure FIND, which implements its second phase.

A. Hierarchical Model Formulation

The network is represented by a graph $G = (V, E)$ and is referred to as the *actual network*, or the *layer-1 hierarchy*. We assume that the actual network has a certain *hierarchical structure*. In order to state the precise meaning of the last term, we need to introduce some additional terminology.

Suppose that we partition the (actual) network nodes into some disjoint set of *peer groups* (or *clusters*), and refer to each resulting peer group as to a *layer-2 node*. Furthermore, suppose that we repeat the above process, such that, for each $i > 1$, layer-i nodes are clustered into layer-i peer groups, each then becoming a layer-$(i+1)$ node. We repeat this process until, for some K, we end up with a single layer-K peer group. Having performed such a (K-stage) partition, we say that layer-i nodes that form a layer-$(i+1)$ node v are its *children*, and v is their *parent*; similarly, the layer-i peer group that forms a layer-$(i+1)$ node v is referred to as *the child peer group of* v. A *descendant* of a node is either its child or a descendant's child.

Next, for each layer i, $1 \leq i \leq K$, we construct *layer-i links*, in the following way. First, we classify the actual (layer-1) links into two types: *intra-cluster links*, which connect between nodes of the same peer group; and *inter-cluster* links, which connect between nodes of different peer groups. We then define the set of layer-2 links as follows. Each inter-cluster link $e = (u, v)$ (of the actual network), gives rise to a corresponding layer-2 link, which connects the parent nodes of u and v. Intra-cluster links (of the actual network) are not represented at layer 2. Following the same process as above, each layer-2 link is classified as either intra-cluster or inter-cluster, and layer-3 links are then defined; the process is repeated up to the last, K'th layer. We have thus defined, for each i, $1 \leq i \leq K$, sets of layer-i nodes and links, which effectively form a *layer-i topology*.

It should be noted that a node v of a layer $i > 1$ represents a subgraph of the actual network, to which we refer as the *source graph* of v, namely:

Definition 5: Given a layer-i node v, $i > 1$, its *source graph* $S[v]$ is defined to be the subgraph of the actual network induced by the set of v's descendants.

It is convenient to define also the source graph of a peer group, namely:

Definition 6: Given a peer group C, its *source graph* $S[C]$ is defined to be the subgraph of the actual network induced by the set of descendants of all nodes $v \in C$.

A node in $S[C]$, which has a neighbor that does not belong to $S[C]$, is called a *border* node. For convenience, we refer to border nodes of $S[C]$ also as border nodes of C. We denote by b the maximum number of border nodes in any peer group.

We are now ready to define the concept of *hierarchical structure*. Intuitively, it means that the network can be partitioned into peer groups, according to the above process, such that, at all layers, peer groups are relatively small (each comprises of at most $O(\log N)$ nodes), and, at the same time, so is the number of inter-cluster links. Formally:

Definition 7: A network $G(V, E)$ is said to posses a *hierarchical structure* if it can be iteratively clustered into some K layers of peer groups, according to the process described above, such that all the following hold:

1. The number of nodes in a peer group is at least 2 and at most d, where $d = O(\log N)$.
2. The number of border nodes is small; specifically, there is some (fixed) value b, such that the number of border nodes of each peer group is at most b.

Note that, since there are at least 2 nodes in each peer group, we have that $K = O(\log M) = O(\log N)$.

Let us illustrate the above terminology through an example. Fig. 1 depicts an actual, layer 1, topology, while Fig. 2 presents possible layer 2 and layer 3 topologies. In this example we have $d = 6$, $K = 3$, $b = 2$.

Networks that have a hierarchical structure shall be referred to as *hierarchical networks*. In this section we assume that networks belong to this class, and, furthermore, that their hierarchical structure, i.e., partition into peer groups, is given.

We can establish the following "sparsity" property of hierarchical networks:

Lemma 1: In a hierarchical network $M = O(N \log N)$.

Proof: For each actual (layer-1) link $e \in E$, there exists an intra-cluster link of some peer group C, which belongs to some layer-i topology. Accordingly, let us count the number of intra-cluster links of all peer groups of all topologies. Each peer group has at most d^2 intra-cluster links, and a layer-i topology consists of $O(d^{K-i})$ peer groups. Therefore,

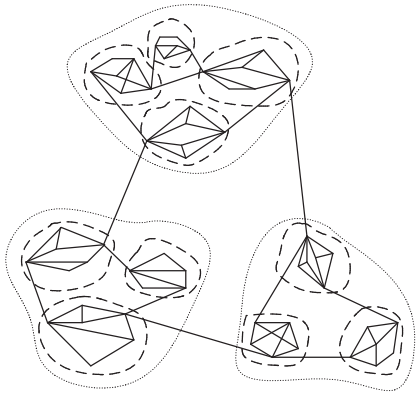

Fig. 1. An example of a hierarchical network

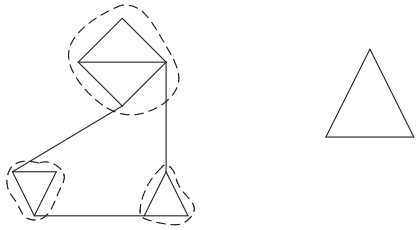

Fig. 2. Example network: layer 2 and layer 3 topologies

the total number of intra-cluster links of all topologies is at most $d^2 \cdot (d + d^2 + ... + d^{K-1}) = O(N \cdot d)$. Next, it is easy to verify that each intra-cluster link represents at most b^2 links of the actual network. As a result, the total number of links in E is at most $N \cdot b^2 \cdot d = O(N \log N)$. ■

B. Procedure CLUSTER

In general, the task of the first phase of a precomputation scheme is to considerably reduce the computational burden at the second phase. With hierarchical networks, this goal can be achieved by precomputing, per peer group, the "costs" of all connections that may be established across it. Algorithm BH-HIE which constitutes the first phase of our precomputation scheme, implements that idea, by sequentially calling to Procedure CLUSTER , which is described in this subsection.

Procedure CLUSTER receives, as input, some layer-i peer group C, and a node $\bar{s}$, which is one of C's border nodes. It then (pre-)computes, for each border node v of C and hop-count h, the minimum weight h-hop constrained path $\mathbf{p}$ that connects between $\bar{s}$ and v through the peer group; the weight of this path is stored in a 3-dimensional array T^C, namely $T^C[\bar{s}, v, h] = W(\mathbf{p})$. A main idea in Procedure CLUSTER is that, when applied on a layer-i peer group as input, it already has available the output $T^{\hat{C}}$ of previous invocations on lower layer peer groups $\hat{C}$.

For the layer-1 topology, the implementation of Procedure CLUSTER is straightforward, since it essentially solves a standard Problem AHOP. For all higher layers, however, a more elaborated process is required, since each single node represents a whole subgraph of the actual network.

Consider the invocation of Procedure CLUSTER on a layer-i peer group C, where $i > 1$. At this stage, due to the previous invocations of the procedure, we have the following information on each node $v \in C$: for each pair $(u_{1,}u_2)$ of border nodes of the child peer group C_v of v, and for each hop value $1 \leq h \leq d^{i-1}$, we have the minimum weight value of a path with at most h hops that runs between those two nodes across C_v. The procedure starts by constructing the following auxiliary graph $\bar{C}(\bar{V}, \bar{E})$. Each node $v \in C$ is substituted in $\bar{C}$ by the set of border nodes B_v of its child peer group C_v. Each such pair of border nodes is connected in $\bar{C}$ by a link; in addition, each two nodes in $\bar{C}$ which are connected by a link in the actual network, are also connected by a link in $\bar{C}$. Having constructed the topology of $\bar{C}$, the procedure produces (through its sub-procedure INITIALIZE) a set S of quadruples (u_1, u_2, h, w), such that u_1 and u_2 are two connected nodes in $\bar{C}$, h is a hop count value, and w is the minimum weight value that can be supported on a path with at most h hops between u_1 and u_2, as computed in the previous invocations of Procedure CLUSTER. The procedure then assigns "length" values $l(e)$ to the links $e \in \bar{C}$, in the following iterative way. Initially, all lengths are

considered as infinite; then, the procedure scans the set S by increasing order of the weight values: for each scanned quadruple (u_1, u_2, h, w), the procedure sets the length of $e = (u_1, u_2)$ to the value of h, and then updates the tree of minimum-length paths in $\bar{C}$ from the source node $\bar{s}$; this way, the values of $T^C[\bar{s}, \cdot, \cdot]$ are identified. More specifically, if the change in the length of e results in shortening the length between $\bar{s}$ and some border node $v \in C$ to a (smaller) value $\hat{h}$, then $T^C[\bar{s}, v, \hat{h}]$ is assigned the value of w, i.e., the weight value of the scanned quadruple. The formal specification of Procedure CLUSTER appears in Fig. 3.

We proceed to establish the following properties of the procedure.

Lemma 2: Given are a layer-k peer group C and the (correct) values of $T^{\hat{C}}$ for every lower layer peer group $\hat{C}$. Then, for each border node v of C and for each $0 \leq h \leq d^k$, Procedure CLUSTER identifies the minimum weight of a h-hop constrained path from $\bar{s}$ to v in the source graph $S[C]$ of C.

Proof: By way of contradiction, assume that the lemma does not hold. Then, for some border node v of C, there exists a path $\mathbf{p} = \{s = v_0, v_1, ..., v_m = v\} \in S[C]$, for which $|\mathbf{p}| \leq h$ and $W(\mathbf{p}) < T^C[\bar{s}, v, h]$. Denote the first node u in $\mathbf{p}$ for which $u \in \bar{C}$ by $\bar{v}_0$, the second by $\bar{v}_1$, etc., up to $\bar{v}_n = v$. The nodes $\bar{v}_i$ constitute a path in $\bar{C}$, which we denote by $\bar{\mathbf{p}} = \{s = \bar{v}_0, \bar{v}_1, ..., \bar{v}_n = v\}$. Let $\hat{\imath}$ be a lowest value of i, for which $T^C[\bar{s}, v_i, |\mathbf{p}_i|] > W(\mathbf{p_i})$, where $\mathbf{p}_i = \{s, ..., \bar{v}_i\}$ is a subpath of $\mathbf{p}$. Note that for $i = \hat{\imath} - 1$ still holds $T^C[\bar{s}, \bar{v}_i, |\mathbf{p}_i|] = W(\mathbf{p}_i)$.

Consider now a link $e = (\bar{v}_{i-1}, \bar{v}_i) \in \bar{C}$, and a subpath $\mathbf{p}_e$ of $\mathbf{p}$ that corresponds to e, $\mathbf{p}_e = \{\bar{v}_{i-1}, ..., \bar{v}_i\}$. It is easy to verify that $(\bar{v}_{i-1}, \bar{v}_i, h, w) \in S$, where $h = |\mathbf{p_e}|$ and $w \leq W(\mathbf{p}_e)$.

We need to consider two possible cases:

1. When the quadruple $(\bar{v}_{i-1}, \bar{v}_i, h, w)$ is processed at line 7 of the algorithm, it holds that $T^C[\bar{s}, \bar{v}_{i-1}, |\mathbf{p}_{i-1}|] = W(\mathbf{p}_{i-1})$. In this case, the sub-procedure **propagate** will be invoked at line 12 of Procedure CLUSTER with parameters $(\bar{v}_{i-1}, \bar{v}_i, h, w)$. After the invocation of PROPAGATE, T^C implies that $T^C[\bar{s}, v_i, |\mathbf{p}_i|] = W(\mathbf{p}_i)$, hence resulting in a contradiction.

2. Otherwise, consider the step of the algorithm in which $T^C[\bar{s}, \bar{v}_{i-1}, |\mathbf{p}_{i-1}|]$ was assigned the value $w(\mathbf{p}_{i-1})$. Since the quadruple $(\bar{v}_{i-1}, \bar{v}_i, h, w)$ was already processed by the loop at line 7, this update leads to a recursive invocation of the sub-procedure PROPAGATE (line 6) with parameters $(\bar{v}_{i-1}, \bar{v}_i, h, w)$, where $w = w(\mathbf{p}_{i-1})$ and, again, after this invocation, $T^C[\bar{s}, v_i, |\mathbf{p}_i|] = W(\mathbf{p}_i)$, resulting in a contradiction.

■

In the next lemma we analyze the complexity of Procedure CLUSTER .

Lemma 3: The computational complexity of Procedure CLUSTER for a layer-i peer group is $O(d^{i+1})$.

Proof: First, let us count the number of elements in S. For each $e \in C$ we added at most b^2 elements to S. We also added at most d^{i-1} elements for every pair of border nodes of the child peer group C_v for each $v \in C$. In total, the number of elements in S is at most $b^2 \cdot d^{\min(i,2)}$. This is also the complexity of the sub-procedure INITIALIZE and of lines 2-11 of Procedure CLUSTER .

Next, we show that sorting the elements of S consumes $O(d^i)$ running time. Note that S's elements are constructed from at most $b^2 \cdot d$ ordered sets, and an additional set of at most $b^2 \cdot d^2$ links. It is easy to verify that such a sorting can be performed by $O(d^{\min(i,2)} \log d)$ steps.

Finally, let us count the number of invocations of the sub-procedure PROPAGATE. This procedure is invoked $|\bar{E}|$ times by line 12 of the cluster procedure and also is invoked recursively. Each recursive invocation implies that h_u for some $u \in \bar{C}$ is increased by at least 1. Since h_u for each $u \in \bar{C}$ is bounded by d^i, the number of recursive invocations of **propagate** is $O(d^{i+1})$. Note that a single invocation of **propagate** requires constant time.

We conclude that the total running time of Procedure CLUSTER is y is indeed $O(d^{i+1})$. ■

C. First phase: Algorithm BH-HIE

In this subsection we describe Algorithm BH-HIE, which implements the first phase of our precomputation scheme.

Algorithm BH-HIE computes, for each peer group of each layer, the best cost (in terms of number of links) for each weight value that can be supported through the peer group. Specifically, for each peer group C, and considering each border node as a source node, we identify the solution of the corresponding Problem AHOP in the source graph $S[C]$ of C. These solutions are then the input of Procedure FIND ,which implements the second phase of the precomputation scheme.

Algorithm BH-HIE runs across the hierarchical layers in a "bottom-up" manner. First, we process each peer group C of the actual network, in the following way. Considering each border node[2] of C as a source node, we invoke

[2]In this context, if the source graph of C includes the source node s, then s is also considered as one of C's border nodes.

Procedure* CLUSTER *($G, \bar{s}$):

parameters

$C(V, E)$- a layer-k peer group

$\bar{s}$ - a source node, which is a border node of C

variables

S- a set of "node-node-hop-weight" quadruples

$\bar{C}(\bar{V}, \bar{E})$- the auxiliary graph, i.e.:

for all $v \in \bar{V}$

$Adj(v)$ - the adjacency list for a node v.[a]

for all $v \in \bar{V}$

h_v- the minimum length of a path between $\bar{s}$ and v in $\bar{C}$

notation

C_v- the child peer group of a layer-i node v.

$B(C)$ - the set of border nodes of the peer group C.

```
1  INITIALIZE()
2  for all v ∈ V̄ do
3     h_v ← d^k + 1
4     Adj(v) ← ∅
5  h_s̄ ← 0
6  T^C[s̄, s̄, 0] ← 0
7  for each (v, u, h, w) ∈ S by increasing order of w do
8     if ((u, ·) ∈ Adj(v)) then
9        let (u, i) ∈ Adj(v)
10       Adj(v) ← Adj(v)\(u, i)
11       Adj(v) ← Adj(v) ∪ {(u, h)}
12       PROPAGATE(v, u, h, w);
```

Procedure* PROPAGATE *(v, u, h, w):

```
1  if (h_v + h) < h_u then
2     for i ← (h_v + h) to (h_u − 1) do
3        T^C[s̄, v, i] ← w
4        P^C[s̄, v, i] ← v
5        h_u ← (h_v + h)
6        for all (x, i) ∈ Adj(u) do
7           PROPAGATE(u, x, i, w)
```

Procedure* INITIALIZE *():

```
1  S ← ∅
2  V̄ ← ∅
3  for each v ∈ V do
4     Ĉ ← the child peer group of node v.
5     B ← the set of Ĉ's border nodes
6     V̄ ← V̄ ∪ B
7     for each pair (u1, u2) ∈ B do
8        for h ← 1 to d^k do
9           S ← S ∪ {(u1, u2, h, T^C[u1, u2, h])}
10 for each e = (v, u) ∈ C do
11    Ĉ ← the child peer group of node v.
12    B_v ← the set of Ĉ's border nodes
13    Ĉ ← the child peer group of node u.
14    B_u ← the set of Ĉ's border nodes
15    for each pair (u1, u2) : (u1 ∈ B_v ∧ u2 ∈ B_u) do
16    if ∃e(u1, u2) ∈ S[C] then
17       S ← S ∪ {(u1, u2, 1, w(e))}
18 return S, V̄
```

[a] $Adj(v) = \{(u, l_e)\}$ for a node v, where l_e is the length of the edge $e = (v, u)$.

Fig. 3. Procedure CLUSTER

Procedure CLUSTER described in Subsection III-B, and store the result in the array T^C. We then iteratively apply the same process to all higher layers. The formal specification of Algorithm BH-HIE appears in Fig. 4.

Algorithm* BH-HIE *(*G, s*):
parameters
G- actual network;
$s \in G$- source node.
notation
$B(C)$- the set of border nodes of a peer group C[a];
G_i- the layer-i topology;
C_i^1, C_i^2, ...- the peer groups of the layer-i topology.

```
1  i ← 1
2  while i ≤ K do
3     for each peer group C_i^j of G_i do
4        for each node v ∈ B(C_i^j) do
5           invoke Procedure CLUSTER for (C_{i-1}^k, v).
6     i ← i + 1
```

[a] In this context, if the source graph of C includes s, then s is also considered as one of C's border nodes.

Fig. 4. Algorithm BH-HIE

We proceed to establish the following properties of Algorithm BH-HIE.

Lemma 4: Algorithm BH-HIE solves Problem AHOP for all peer groups at all layers.

Proof: Straightforward by induction on topology layers and application of Lemma 2. ■

Lemma 5: The computational complexity of Algorithm BH-HIE is $O(N \log^2 N)$.

Proof: Let us count the time required to process a layer-i topology. Such a topology contains $O(\frac{N}{d^i})$ peer groups, for each of which the cluster procedure is invoked. Since he running time of Procedure CLUSTER is $O(d^{i+1})$ (by Lemma 3), a layer-i topology requires $O(N \cdot d) = O(N \log N)$ operations. As there are $K = \log(N)$ layers,the algorithm's complexity is $O(N \log^2 N)$. ■

D. Second phase: Procedure FIND

We proceed to present Procedure FIND. This procedure is invoked upon each new connection request, and identifies the corresponding path, namely a path of minimum hops among the corresponding source (s) and destination (d) that satisfy the connection's bottleneck requirement ($\tilde{w}$).

The procedure processes the hierarchical layers iteratively, starting from the first layer, i.e., the actual network, up to the last, K'th, layer. For each layer, we identify the peer group C, for which the source graph $S[C]$ includes the destination node d. Then, a minimum hop path from each border node b of C to d is identified.

For this purpose we construct the following auxiliary graph $\bar{G}$. The destination node d and the border nodes of the child peer groups of C constitute the set of $\bar{G}$'s nodes. Every pair of border nodes (v, u) of a child peer group $\tilde{C}$ of C is connected by a link, whose length is assigned to be the minimum number of hops of a $\tilde{w}$-weight constrained path between v and u in the source graph $S[\tilde{C}]$ of $\tilde{C}$; this value is provided by the array $T^{\tilde{C}}$, which was computed in the first phase. In addition, for every actual network link $e = (v, u)$ for which $w(e) \leq \tilde{w}$ and which gave rise to an intra-cluster link in C, we add in $\bar{G}$ a link between v and u, whose length is set to 1. As a result, a minimum length path in $\bar{G}$ corresponds to a minimum hop path in the source graph $S[C]$ of C.

As shall be shown below, the complexity of Procedure FIND is $O(\log^3 N + |\mathbf{p}|)$, where $|\mathbf{p}|$ in the number of links in the identified path. The formal specification of Procedure FIND appears in Fig. 5.

We proceed to prove the correctness of Procedure FIND.

Lemma 6: Suppose that Procedure FIND is invoked for a source s, destination d and (bottleneck) QoS constraint $\tilde{w}$. Then, the hop count $\tilde{h}$ of the returned path is the minimum number of hops of a path in the actual network between s and d that satisfies the QoS constraint $\tilde{w}$.

Proof: By way of contradiction, assume that the lemma does not hold. Then, there exists a path $\mathbf{p} = \{s = v_0, v_1, ..., v_m = d\}$ in the actual network, for which $W(\mathbf{p}) \leq \tilde{w}$ and $|\mathbf{p}| < \tilde{h}$. For $1 \leq i \leq K$, let C_i be a layer-i peer group for which $d \in S[C_i]$. Certainly, the path $\mathbf{p}$ includes border nodes of peer groups C_i, $1 \leq i \leq k$, for some

Procedure* FIND *($G, s, d, \tilde{w}$):

parameters

G- actual network.
$s \in G$- source node;
$d \in G$- destination node;
$\tilde{w}$- QoS (bottleneck) requirement;

notations

G_i- layer-i topology;
$S[C]$- the source graph of a peer group C;
$B(C)$- set of border nodes of peer group C.

variables

$\bar{G}(\bar{V}, \bar{E})$- the auxiliary graph;

1 let C be a layer-1 peer group, for which $d \in C$;
2 remove from C all links which weight is bigger than $\tilde{w}$;
3 for each border node $b \in B(C)$, identify a minimum hop pathin C from b to d (e.g., using a Breadth First Search algorithm[6]);
4 $\bar{V} \leftarrow \{B(C) \cup d\}$;
5 $\bar{E} \leftarrow \emptyset$;
6 **for** each $v \in B(C)$ **do**
7 add a new link $e = (v, d)$ into $\bar{E}$;
8 set $l(e)$ to be the minimum number of hops in a path from v to d
9 $i \leftarrow 2$;
10 **while** $i \leq K$ **do**
11 let C be a layer-i peer group, for which $d \in S[C]$;
12 **for** each $v \in C$ **do**
13 let C_v be a child peer group of v;
14 $\bar{V} \leftarrow \{\bar{V} \cup B(C_v)\}$;
15 **for** each ordered pair (u, w) [a] of C_v's border nodes **do**
16 **if** there exist h, for which holds $T^{C_v}[u, w, h] \leq \tilde{w}$**then**
17 find the lowest h, for which holds $T^{C_v}[u, w, h] \leq \tilde{w}$,
18 add a new link $e = (u, w), l(e) = h$ to $\bar{G}$;
19 **for** each pair of nodes $v, u \in \bar{V}$, for which $\exists e = (v, u) \in G, w(e) \leq \tilde{w}$ **do**
20 add a new link $e = (v, u)$ to $\bar{G}$ and set $l(e) = 1$,
21 using Dijkstra's algorithm, identify the shortest path fromeach $v \in B(C)$ to d in $\bar{G}$;
22 $\bar{V} \leftarrow \{B(C) \cup d\}$;
23 $\bar{E} \leftarrow \emptyset$;
24 **for** each $v \in B(C)$ **do**
25 add a new link $e = (v, d)$ into $\bar{E}$;
26 set $l(e)$ to be the length of the shortest path from v to d, asidentified in line 21;
27 let $e = (s, d) \in \bar{E}$.[b]
28 **return** $l(e)$.

[a] Note that, since (u, v) is considered as an ordered pair, we distinguish between (v, u) and (u, v).
[b] Note that there must exist an edge $e = (s, d) \in \bar{E}$, since the source node s is a border node of C (recall that there is only one layer-K peer group).

Fig. 5. Procedure FIND

$k \leq K$.Suppose that we traverse $\mathbf{p}$ from d to s. For $1 \leq i \leq k$,denote by $\bar{v}_i$ the first node in the traversal that is a border node of C_i . Also, for each layer i, $1 \leq i \leq K$,we denote by $\bar{G}_i$ the auxiliary graph constructed for this layer. Finally, we denote by l_i the length of a shortest path from $\bar{v}_i$ to s in G_i,as identified at line 21.

It is sufficient to prove that, for each $1 \leq i \leq K$, the value l_i is at most the hop count of a subpath $\mathbf{p}_i = \{\bar{v}_i, ..., d\}$ of $\mathbf{p}$. Let $\hat{\imath}$ be the minimum i, for which this does not hold. Consider a path $\bar{\mathbf{p}} = \{\bar{v}_i, ..., \bar{v}_{i-1}\}$ in $\bar{G}$, which corresponds to the subpath $\mathbf{p}_i = \{\bar{v}_i, ..., \bar{v}_{i-1}\}$ of $\mathbf{p}$. It follows that $l(\bar{\mathbf{p}}) > |\mathbf{p_i}|$. Thus, there exists a link $e = (\bar{v}, \bar{u}) \in \bar{\mathbf{p}}$, for which $l(e)$ is greater that the hop count of the corresponding subpath $\mathbf{p}_e = \{\bar{v}, ..., \bar{u}\}$ of $\mathbf{p}$. There are two possibilities.
1. The link e corresponds to a single actual network link. In this case the link e was assigned the length 1 by line 20 of the algorithm.
2. Otherwise, $e = (\bar{v}, \bar{u})$ is a link between border nodes of a child peer group C_v for some node $v \in C$. In this case, $l_e \leq \mathbf{p}_e$ is assigned the lowest h, for which it holds that $T^{C_v}[\bar{v}, \bar{u}, h] \leq \tilde{w}$. Both cases result in a contradiction, hence

the lemma follows. ■

We proceed to analyze the computational complexity of the procedure.

Lemma 7: The computational complexity of Procedure FIND is $O(|\mathbf{p}|+\log^3 N)$.

Proof: Note that the graph $\bar{G}$ contains just $O(\log N)$ nodes and $O(\log^2 N)$ links at each hierarchical layer. The execution of all lines in the procedure, except from lines 16 and 17, require only a fixed number of steps per link, or $O(\log^2 N)$ per layer. Lines 16 and 17 may be implemented in $O(\log H)$ running time per link, by a binary search. These lines are executed $O(\log N)$ times for each layer, hence they incur $O(\log N \log H)$ steps per layer. As a result, the procedure performs $O(\log^2 N)$ operations per layer. Since the number of layers is $O(\log N)$, we need $O(\log^3 N)$ running time in total. In addition, we need $O(|\mathbf{p}|))$ time to report the output, where $\mathbf{p}$ is the path identified by the algorithm. Thus, the time complexity of the procedure is $O(|\mathbf{p}|+\log^3 N)$. ■

The above results are summarized in the following theorem.

Theorem 1: Procedure FIND provides a $O(|\mathbf{p}|+\log^3 N)$ solution to Problem BH-RSP, i.e.: given a connection request with source node s, destination node d, and (bottleneck) QoS constraint $\tilde{w}$, and given the output of Algorithm BH-HIE, Procedure FIND identifies, in $O(|\mathbf{p}|+\log^3 N)$ steps, a path with a minimum number of hops, among all paths in the actual network between s and d that satisfy the QoS constraint $\tilde{w}$.

E. Discussion

In this subsection we compare between the performance of our precomputation scheme and its alternatives.

Consider first the "standard" precomputation scheme proposed in [9], [2], which was based on solving Problem AHOP through Bellman-Ford's shortest path algorithm.

As shown above, hierarchical networks are sparse, in the sense that $M = O(N \log N)$. This implies that the standard scheme incurs a computational complexity of $O(NH \log N)$ for its first phase, i.e., it is $O(\frac{H}{\log N})$ slower than ours. Considering the second phase, the standard scheme (as well as any other which is based on fully solving Problem AHOP in the first phase) yields a computational complexity of just $O(|\mathbf{p}|+\log H)$, which is somewhat less than that of our scheme, i.e. $O(|\mathbf{p}|+\log^3 N)$, however the difference between the two figures is not significant in general, and nonexistent when $|\mathbf{p}|$ is the dominating component.

Next, let us compare between our precomputation scheme, and an alternative where no precomputation is performed at all. In such a "single-phase" scheme, the required path can be identified by applying Dijkstra's shortest path algorithm, which, for $M = O(N \log N)$, incurs $O(N \log N)$ running time. Since $|\mathbf{p}| = O(N)$, our scheme incurs a smaller computational complexity upon a connection request. The difference is particularly significant when the length of the identified path is significantly smaller than N, e.g., $|\mathbf{p}| = O(\log N)$, which is a typical case. It is interesting to compare between the two approaches also in the related context of *connection admission*, where one needs to decide whether a connection request should be admitted, based on its QoS requirement and the cost it incurs; to that end, one needs to identify the (best) cost of a path over which the connection can be established, however there is no need to explicitly specify the path itself. This means that our scheme allows to obtain an admission decision upon a connection request in just $O(\log^3 N)$ time, whereas the "single-phase" scheme still incurs $O(N \log N)$ time.

F. Model Relaxation

One of the properties of hierarchical networks is that the number of nodes in each peer group is at most d, where $d = O(\log N)$ (see III-A). This requirement may be relaxed by allowing certain peer groups to be composed from more than d nodes, provided that in all peer groups the number of links on any path is $O(\log N)$. This relaxation does not affect the computational complexity of $O(M \log N)$ for the first phase of our precomputation scheme. This follows from the fact that the computational complexity of Procedure CLUSTER applied to a peer group C is $O(M(C)\dot{d}(C))$, where $M(C)$ is the number of C's edges and $d(C)$ is an upper bound to the number of links of any path is C. This, in turn, may be easily verified in a similar way as done in the Proof of Lemma 2. Note that in the relaxed model M is not $O(N \log N)$ anymore, but can rather be as large as $O(N^2)$. The computation complexity of the second phase may be as much as $O(M)$ in the worst case. However, under under certain conditions, the running time of the second phase is same as for regular (not extended model). The condition is that neither the source node s nor the destination node d are descendants of node, which child peer group comprises more that d nodes. Note that this restriction applies to at most $2 \cdot \log N$ peer groups out of $O(N)$ peer groups in total.

IV. SOLVING ALL-TO-ALL PROBLEMS

The precomputation scheme, described above can be extended for a broad class of problems related to Problem BH-RSP. In this section we present the precomputation scheme for a variation of Problem BH-RSP, in which it is required to solve Problem BH-RSP for any two nodes in G. In other words, given a bottleneck QoS constraint $\hat{w}$ and a pair of nodes v_s and v_d, it is required to identify the minimum hop path, among all paths from v_s to v_d, which satisfy the QoS constrain $\hat{w}$. The first phase is identical to the precomputation scheme for Problem BH-RSP and is implemented by Algorithm BH-HIE. Recall that in this algorithm Procedure CLUSTER is invoked for each peer group at all layers. Since the second phase is implemented similarly to the second phase of the precomputation scheme for Problem BH-RSP, we present only a brief description. Let h be a lowest layer for which there exists a peer group C, for which $v_s \in S(C)$ and $v_d \in S(C)$ where $S(C)$ is the source graph of C. Let also C_s and C_d be layer$-(i-1)$ peer groups, for which holds $v_s \in S(C_s)$ and $v_d \in S(C_d)$, where $S(C_s)$ and $S(C_d)$ are the source graphs of C_s and C_d respectively. In order to identify the minimum hop path from v_s to v_d among all paths from v_s to v_d which satisfy a given QoS constrain $\hat{w}$, following steps are executed.

1. For each border node v of C_s, identify the minimum hop path, among all path from v_s to v, which satisfy QoS constrain $\hat{w}$.
2. For each border node v of C_d, identify the minimum hop path from v to v_d, among all paths that satisfy $\hat{w}$.
3. Construct the following auxiliary graph $\hat{G}$. The set of nodes in $\hat{G}$ includes v_s, v_d, border nodes of all peer groups whose parent node belongs to C, and border nodes of all layer$-h$ peer groups. Any two nodes v, w in $\hat{G}$ that belong to the same peer group P of G are connected by a link, whose weight is the minimum hop count of a path from v to w in P, which satisfies $\hat{w}$. In addition there are links from v_s to border nodes of C_s and from border nodes of C_d to v_d. The weight of these links is as computed in steps 1 and 2.
4. Identify the shortest path from v_s to v_d in $\hat{G}$.

All this steps are implemented in a similar way as it done in Procedure FIND . It is easy to verify that the computation complexity of this solution is the same as that of Procedure FIND . To conclude, we presented a precomputation scheme for a variation of Problem BH-RSP problem, in which it is required to identify paths from any source to any destination; the computation complexity of our solution is $O(N \log N)$ for the first phase and $O(\log^3 N)$ for the second.

V. TOPOLOGY AGGREGATION

Our discussion so far concentrated on link state protocols, which assume that a complete and accurate image of the network is available for a network node. However this approach suffers from scalability problems. In particular, as a network grows in size, a significant part of network bandwidth is consumed for maintaining topology image on every node. As a solution, the ATM forum PNNI standard [1] is designed to provide a scalable representation of hierarchical topologies. According to this standard, a cluster does not reveal its internal structure to outside nodes. Instead, it supplies a summary of cost and availabilities of connections that run through that cluster. This approach is often referred to as topology aggregation. In this section we discuss a variation of our precomputation scheme for networks with topology aggregation.

A. Network topology as seen by a node

A (proper) aggregated image is simpler than the real topology, yet it still captures its structure in the way that makes it suitable for QoS purposes. Following the ATM PNNI recommendations [1], we describe the aggregated image of a network at some node v. All other peer groups are omitted from the network image. We proceed define the set A_v of peer groups that are included by aggregated image of the network for a node v. A_v includes any peer group C which source graph $S(G)$ includes v. Since there exists only one such a peer group for each layer,we conclude that the cardinality of A_v is at most $O(\log N)$. Fig. 6 depicts aggregated image of the topology depicted on Fig. 1.

Though an aggregated image that is comprised of the set A_v is sufficient for identifying a route from v to a destination, it does not contain enough data for QoS routing. Consequently, we need some additional information concerning the peer groups not included in A_v. This information includes a summary of costs and availabilities of connections that run through certain peer groups. These peer groups form a set denoted by B_v. Set B_v includes every peer group C, whose parent node belongs to a peer group in A_v (C itself does not belongs to A_v). The summary for a peer group C is in the form of the output that would be obtained by Procedure CLUSTER if it were applied to C. In Fig. 6, all peer groups belonging to B_v are marked as "clouds". It is easy to verify that the space complexity of the aggregated image is $O(N)$, as compared with $O(NlogN)$ for a non-aggregated image.

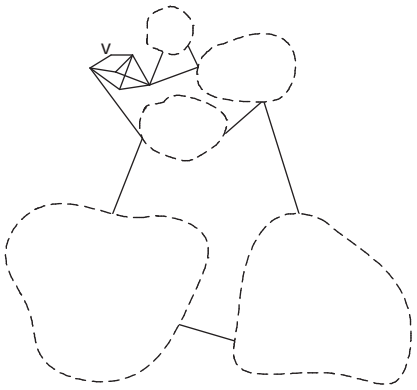

Fig. 6. An aggregated image of network for a node v

B. A revised precomputation scheme

Only minor changes should be introduced to our precomputation scheme in order to adapt it to networks with topology aggregation. Since an aggregated image already includes precomputation results for a number of peer groups, the precomputation phase becomes easier: Procedure CLUSTER is applied only for one peer group at each layer, and not for each peer group as in Algorithm BH-HIE. As a result, its computational complexity is just $O(N \log N)$ for networks with topology aggregation, as compared with $O(N\log^2 N)$ for hierarchical networks without aggregation. The aggregated image of a peer group $C \in B_v$ can be delivered to v from one of C's border nodes. With network aggregation, establishing a new connection with a QoS constraint $\hat{w}$ requires additional steps, which include data exchange between source and destination nodes. Indeed, a destination node may be located in a "cloud" in the aggregated image of a source node, which corresponds to some peer group $C \in B_v$. The only information that a source node needs for computing the optimal route to d is the lowest cost of a path from each border node b_v of C to d which satisfies the QoS constraint $\hat{w}$. This data is also calculated at the precomputation phase, with no penalty in terms of computational complexity[3]. The data is sent from the source to destination and its size is constant (does not depend on N). Upon arrival of this data to the source node, a variant of Algorithm FIND , which is a straightforward simplification of Algorithm FIND , is executed; it identifies a lowest cost path from s to d that satisfies $\hat{w}$. This procedure requires $O(\log^3 N)$ time. We conclude that, compared to the regular approach, the topology aggregation scheme requires less space and fewer messages, and it gives raise to a faster precomputation algorithm. Its disadvantage is that the source and destination nodes are required to exchange data before establishing the connection.

VI. RESTRICTED BUDGET PROBLEMS

In this section we consider a class of bottleneck problems where there is a cost assigned to each network edge. Given a cost budget, it is required to identify a *minimum weight* path, among those that obey the budget. Certainly, this is a variation of the Restricted Shorted Path (RSP) problem, defined in Section II. The formal definition is as follows.

Problem GB-RSP (General weight Bottleneck cost RSP): Given are a network G, a source node s, a destination node d and a *budget* B. Find a path $\hat{\mathbf{p}}$ from s to d such that:
1. $C(\hat{\mathbf{p}}) \leq B$,
2. $W(\hat{\mathbf{p}}) \leq W(\mathbf{p})$ for every other path $\mathbf{p}$ that satisfies the restriction $C(\mathbf{p}) \leq B$,
3. there does not exist another path $\tilde{\mathbf{p}}$, for which $W(\tilde{\mathbf{p}}) = W(\mathbf{p})$ and $C(\tilde{\mathbf{p}}) < C(\hat{\mathbf{p}})$.

We begin by noting the following straightforward, yet computationally expensive, scheme to solve this problem: for each possible weight $\hat{w}$ in G, we delete any each edge e for which $w(e) > \hat{w}$, and then execute a shortest path algorithm; with Dijkstra's shortest path algorithm, the computational complexity is $O(M(M + N \log N))$. In this section we present a more efficient solution that requires just $O(logN)$ invocations of a shortest path algorithm. Our algorithm takes advantage of the fact that, for bottleneck metrics, a minimum cost of a path from s to d whose weight is at most $\hat{w}$ can be found by means of single invocation of a shortest path algorithm. Thus, by performing a binary search on the range of weight values, a minimum weight path among all paths from s to d that obey B may be found.

The formal specification of the algorithm is presented on Fig. 7.

Lemma 8: The running time of Algorithm GB-RSP is $O((M + N \log N) \log N)$.

[3] It requires only a straightforward addition for precomputation algorithm.

Algorithm* GB-RSP *(*G, s, d, C*):

parameters:
$G(V, E)$- network,
s - a source node
d- a destination node
C- budget
variables:
r_l, r_h-integers from 1 to U, where U is an upper bound of the cost of a path from s to d.

```
1  r_l ← 1, r_h ← U
2  i ← ∞
3  repeat forever do
4     î ← i
5     i ← ⌊(r_l + r_h)/2⌋
6     if (î = i) then
7        return {ĥ, ∞}
8     Let E* = {e ∈ Ê | w(e) ≤ ŵ}
9     determine for each v ∈ Ĝ, the minimal distance l(v) from s to v ( by applying AlgorithmDijkstra on Ĝ = (V, E*))
10    set W(v) ← ∞ for all v ∈ V\s and W(s) ← 0
11    for all v ∈ V in increasing order of l(v) do
12       for all e = (v, u) ∈ E* do
13          if (level(u) − level(v) = 1) then
14             W(u) ← min(W(u), max(W(v), w(e)))
15    if l(d) = ĥ then
16       return {l(d), W(d)}.
17    if l(d) < ĥ then
18       r_h ← i
19       else r_l ← i
```

Fig. 7. Algorithm GB-RSP

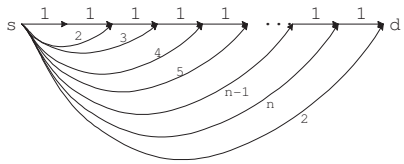

Fig. 8. An example of a special topology

Proof: Initially, the search range is at most M. After the first iteration it shrinks to $M/2$, etc. As a result, the total number of iterations is $O(logN)$. Since the computational complexity of each iteration is $O(M + N \log N)$, the total running time of the algorithm is $O(M + N \log N) \log N)$. ■

The correctness of the algorithm follows from the fact that the cost of the solution is monotonically nonincreasing with the allowed weight. The above results are summarized in the following theorem.

Theorem 2: Algorithm GB-RSP solves GB-RSP with a computational complexity of $O(M + N \log N)(\log N)$.

VII. Precomputation Scheme for Problem BH-RSP in special topologies

In this section we discuss special topologies for which there exist efficient precomputation schemes for Problem BH-RSP. Our discussion is limited to the class of "one-to-one" problems, i.e., the problems with a single or limited number of destinations. We describe first the properties of such topologies that facilitate the efficiency of the precomputation scheme. Let $\mathbf{p}$ be a h-hop constrained optimal path from s to d and $W(\mathbf{p}) = w$. Suppose that, for some $\hat{h} > h$, it holds that the weight $\hat{w}$ of a $\hat{h}$-constrained optimal path equals to w. In other words, relaxing the constraint to $\hat{h}$ links does not yield a better path. Furthermore, there are only a limited number of h values, for which the link count of h-constrained optimal path from s to d is h. We denote the set of such values by $S_{s,d} = \{h_1, h_2, ...h_k\}$. In this section we consider topologies for which the cardinality of $S_{s,d}$ is bounded by some small (fixed) value k. An example of such topology appears on Fig. 8. For this topology the value of k is 2. We proceed to present the precomputation scheme for

special topologies. In the first phase of this scheme, implemented by Algorithm BH-SPEC, we precompute for each $v \in V$ the h-hop constrained optimal path from s to v for each $h \in S_{s,v}$. The weights of that path are stored in an array $WS_{s,d} = \{w_1, w_2, ...w_k\}$, where w_i is the weight of a h_i-hop constrained optimal path from s to v. The second phase identifies for a given QoS constrain $\hat{w}$, the smallest $w_i \in WC_{s,d}$, for which $w_i \leq \hat{w}$ and outputs the corresponding path. The computational complexity of the first phase is $O(M \cdot k)$ or $O(M)$ for fixed values of k, and is $O(\log N)$ for the second phase.

Each iteration of Algorithm BH-SPEC builds an auxiliary graph $\hat{G}$. $\hat{G}$ is identical to G, but includes only these links which weight is less or equal to $w(i)$. Then, a Breadth First Search Algorithm BFS ([6]) is applied on $\hat{G}$, which determines, for each $v \in V$ the minimum hop distance $d_{s,v}$ from s to v. Let $\hat{h}$ be the minimum hop count of a path from s to d in $\hat{G}$ and let $\hat{w}$ be the minimum weight of a $h-$hop path from s to d. Then, the value of $w(i)$ for the next iteration is set to the maximum value of weight in $\hat{G}$, among all weight that are less than $\hat{w}$.

The formal specification of Algorithm BH-SPEC appears in Fig. 9.

Algorithm* BH-SPEC *(G, s):
parameters
G- actual network;
$s \in G$- source node.
$d \in G$- destination node
variables
$w(i)$- iteration parameter

```
1  i ← 1
2  w(i) ← ∞
3  while do
4     delete all links e ∈ E,for which w(e) > w(i)
5     invoke Algorithm BFS on G, determine for each v ∈ G, the minimal hop distance l(v) from s to v
6     for each v ∈ V do
7        W(v) ← ∞
8     W(s) ← 0
9     for each v ∈ V in increasing order of l(v) do
10       for each e = (v, u) ∈ E do
11          if l(u) − l(v) = 1 then
12             W(u) = min{W(u), max{W(v), w(e)}}
13    h_i = l(d)
14    w_i = W(d)
15    i ← i + 1
16    let ŵ = max_{e∈E}{w(e) < w_i}
17    w(i) ← ŵ
```

Fig. 9. Algorithm BH-SPEC

Lemma 9: The computational complexity of Algorithm BH-SPEC is $O(M \cdot d)$, where k is an upper bound to cardinality of the set $S_{s,d}$.

Proof: It is easy to verify that each iteration incurs $O(M)$, and that for each element in $S_{s,d}$ only a single iteration is performed. ■

As stated above, the second part of the precomputation scheme identifies, for a given requirement $\hat{w}$, a $\hat{w}$-constrained optimal path from s to d. We first find the smallest $w_i \in WC_{s,d}$, for which $w_i \leq \hat{w}$. This procedure requires $O(\log N)$ time. The output of the second phase is a path which weight is w_i and link count is h_i.

In the next lemma we prove that this path is a $\hat{w}$-constrained optimum path from s to d.

Lemma 10: Let $\hat{w}$ be a bottleneck constrain and $\mathbf{p}$ be a path returned by the algorithm. Then, $\mathbf{p}$ is the minimal link path among all paths from s to d that satisfy the constrain $\hat{w}$.

Proof: Consider the values of $w(i)$ over the various iterations of Algorithm BH-SPEC. Let j be an iteration, in which the value of $w(j)$ is minimal, but still greater than $\hat{w}$. The invocation of Algorithm BFS guarantees that h_j is a minimum hop count of a path from s to d that satisfies the constrain $\hat{w}$. ■

We conclude our discussion in the following theorem.

Theorem 3: For the special topologies described above, there exists a precomputation scheme whose complexity is $O(M \cdot k)$ for the first phase and $O(logN)$ for the second.

VIII. Precomputation Scheme for Additive Metrics

In this section we consider the routing problem with *additive* QoS constraints and *general* links costs. As mentioned in Section II, this problem is in fact the restricted shortest path Problem RSP, which, in general, is known to be NP-hard. Accordingly, we resort to precomputation schemes that offer *approximate* solutions to Problem RSP.

We note that a precomputation scheme can be constructed on the base of existing approximation algorithms for Problem RSP (e.g. [12]), i.e., by sequentially executing them for various cost values. However, as we shall see, such a simplistic approach results in a (overly) high computational complexity. Therefore, in this section we propose a precomputation scheme that finds an ε-optimal solution to Problem RSP, for all possible QoS constraint values, within $O((M+N\log N)\cdot H\cdot\frac{1}{\varepsilon}\cdot\log C)$ computational complexity for the first phase and $O(\log(\frac{1}{\varepsilon})+\log(H)+\log\log(C)+|\mathbf{p}|)$ for the second phase, where C is an upper bound on the (additive) cost of a path, and $|\mathbf{p}|$ is the hop count of the identified path.

The section is organized as follows. First, we present a pseudo-polynomial solution for Problem RSP in the special case of directed acyclic graphs (DAGs). Next, based on that solution, we establish an $O((M+N\log(N))\cdot H\cdot\frac{1}{\varepsilon}\cdot\log C)$ precomputation scheme that provides an ε-optimal solution for general topologies.

A. Pseudo-polynomial Solution for Problem RSP

As a first step, we present a (computationally inefficient) pseudo-polynomial solution, Algorithm PP-RSP, which is based on a generalization of Bellman-Ford's algorithm. For the sake of simplicity, we assume that the underlying graph is a DAG; an extension to general graphs is straightforward.

The algorithm is based on dynamic programming and assumes integer costs. Given a (additive) QoS constraint $\hat{w}$, the algorithm starts with a zero "budget" $c=0$ and increments it by a value of 1 on each iteration, until a $\hat{w}$-weight constrained path from s to d is discovered. At each iteration, the algorithm repeatedly selects the destination node $u\in V$ according to a topologically sorted order[4] and relaxes all links leaving u. The process of relaxing a link (u,v) consists of testing whether the best path to v found so far can be improved by going through u under the current budget restriction c and, if so, updating the best path for node v.

Since for each $c, 1\le c\le\hat{c}$, the algorithm performs $O(M)$ operations, its complexity is $O(M\cdot\hat{c})$, where $\hat{c}$ is an upper bound on the cost of a ($\hat{w}$-weight constrained optimum) path from s to any $v\in V$. The formal specification of Algorithm PP-RSP appears in Fig. 10.

B. Polynomial Precomputation (Approximation) Scheme

We proceed to present a precomputation scheme that provides ε-optimal solutions for Problem RSP. First, we present a solution for DAGs, whose complexity (for the first phase) is $O(MH\log C/\varepsilon)$, where C is an upper bound on the cost of a path, and then extend it in order to obtain an $O((M+N\log N)H\log C/\varepsilon)$ solution for general topologies.

B.1. Algorithm for directed acyclic graphs

The following algorithm is based on Algorithm PP-RSP, and it uses a *cost quantization* approach. Specifically, it considers only a limited number of budget values, namely $1, c_1, c_2, ...$, where $c_i=\delta^i$ for some $\delta>1$. For each node $v\in V$ and for each c_i, the algorithm outputs a near-minimum weight $W_v[c_i]$ of a path from s to v, whose cost is at most c_i.

For a fixed value of δ, the number of iteration is polynomial on the input size. On the other hand, this approach does not provide an exact solution, and the approximation ratio ε depends on the choice of δ.
The formal specification of Algorithm RSP-DAG is presented in Fig. 11.

Lemma 11: Given are a DAG G, a source node s and an approximation parameter ε. For a (arbitrary) value $\tilde{w}$, let c^{opt} be the cost of a $\tilde{w}$-weight constrained optimal path from s to a (arbitrary) node $v\in V$, and let $\tilde{c}=\min_{i=1,2,...\lfloor\log_\delta C\rfloor}\{c_i|W_v[c_i]\le\tilde{w}\}$, where the values $W_v[c_i]$ are the output of Algorithm RSP-DAG for G, s,ε and v. Then $\frac{\tilde{c}-c^{opt}}{c^{opt}}\le\varepsilon$.

Proof: Let $\mathbf{p}^{opt}$=$\{v_0=s,v_1,...,v_h=d\}$ be a $\tilde{w}$-weight constrained optimal path from s to d. Note that $C(\mathbf{p}^{opt})=c^{opt}$, $W(\mathbf{p}^{opt})\le\tilde{w}$. For $1\le j\le h$, we denote by $\mathbf{p}_j^{opt}=\{v_0,...,v_j\}$ a subpath of $\mathbf{p}^{opt}$,

[4] A topological sort of a DAG G is a linear ordering of its vertices such that, if G contains an edge (u,v), then u appears before v in the ordering. A topologically sorted order may be computed by a DFS algorithm [6].

Algorithm* PP-RSP *($G(V,E), \hat{w}$***):***

parameters:
$G(V,E)$ - network
$s \in G$ - source node
$\hat{w}$- weight (additive QoS) constraint
variables:
c - the "budget"
for all $v \in V$
$W_v[c]$ - the minimum weight of a path between s and v whose cost is at most c

```
1  for all v ∈ V do
2     W_v[0] ← ∞
3  W_s[0] ← 0
4  c ← 1
5  while W_v[c] > ŵ for some v ∈ V do
6     W_s[c] ← 0
7     for all v ∈ V do
8        W_v[c] ← W_v[c − 1]
9     for each node u taken in topologically sorted order do
10       for each node v ∈ Adj[u] do
11          let e = (u, v)
12          if (w(e) ≤ w) then
13                W_v[c] ← min[W_v[c], W_u[c − c(e)] + w(e)]
14    c ← c + 1
```

Fig. 10. Algorithm PP-RSP

Algorithm* RSP-DAG *($G(V,E), s, \varepsilon$***):***

parameters:
$G(V,E)$ - network
$s \in G$ - source node
ε- approximation parameter
variables:
c_i - the "budget"
for all $v \in V$
$W_v[c_i]$ - the approximated minimum weight of a path between s and v whose cost is at most c_i
notation
C- an upper bound to the cost of a path in G
$\delta = (1/(1 - \frac{\varepsilon}{(1+\varepsilon)\cdot H})$
$\Downarrow c \Downarrow = \lfloor \delta^{\lfloor \log_\delta c \rfloor} \rfloor$

```
1  for all v ∈ V
2     do W_v[0] ← ∞
3  W_s[0] ← 0
4  c_0 ← 0
5  i ← 1
6  c_i ← 1
7  while c_i ≤ C do
8     W_s[c_i] ← 0
9     for all v ∈ V do
10       W_v[c_i] ← W_v[c_{i−1}]
11    for each node u taken in topologically sorted order do
12       for each node v ∈ Adj[u]
13       let e = (u, v)
14       if(c(e) ≤ c_i) then
15          W_v[c_i] ← min[W_v[c_i], W_u[⇓c_i − c(e)⇓] + w(e)]
16    i ← (i + 1)
17    c_i ← ⌊δ^i⌋
18 return {W_v[c_i] | i = 1, 2, ...⌊log_δ C⌋} for each v ∈ V
```

Fig. 11. Algorithm RSP-DAG

$w_j^{opt} = W(\mathbf{p}_j^{opt})$, $c_j^{opt} = C(\mathbf{p}_j^{opt})$. For a node $v_j, j = 1, ..., h$ let $\tilde{c}_j = \min_{i=1,2,...\lfloor \log_\delta C \rfloor} \{c_i | W_{v_j}[c_i] \leq w_j^{opt}\}$, where $c_i = \lfloor \delta^i \rfloor$.

We prove by induction on j that $\tilde{c}_j \leq c_j^{opt} \cdot \delta^j$. As the base step, we consider the execution of the loop of line 7 for $i = \lceil \log_\delta c(e) \rceil$, where $e = (s, v_1)$. Line 15 assures that $W_{v_1}(c_i) \leq w_1$. Therefore $\tilde{c}_1 \leq c_i \leq c(e) \cdot \delta \leq c_1^{opt} \cdot \delta$. Thus, we proved that $\tilde{c}_1 \leq c_1^{opt} \cdot \delta$.

For the inductive step, we assume that $\tilde{c}_j \leq c_j^{opt} \cdot \delta^j$ holds for $1, 2, .., j-1$ and prove that it holds for j. Let us consider the execution of the loop of line 7 for $i = \lceil \log_\delta(\tilde{c}_{j-1} + c(e)) \rceil$, where $e = (v_{j-1}, v_j)$. Since $\tilde{c}_{j-1} < c_i$, the value of $W_{v_{j-1}}[\tilde{c}_{j-1}]$ was fixed in the loop of line 7 at either the current or a previous iteration. In both cases the value of $W_{v_{j-1}}[\hat{c}_{j-1}]$ does not change after node v_j is processed. As a result, and since $W_{v_{j-1}}[\tilde{c}_{j-1}] \leq w_{j-1}^{opt}$, line 15 assures that $W_{v_j}[c_i] \leq w_j^{opt}$. Therefore, $\tilde{c}_j \leq c_i \leq \delta \cdot (\tilde{c}_{j-1} + c(e)) \leq \delta \cdot (c_{j-1}^{opt} \cdot \delta^{j-1} + c(e))$, for $e = (v_{j-1}, v_j)$, where the last inequality follows from the inductive assumption. Hence, $\tilde{c}_j \leq c_j^{opt} \cdot \delta^j$, since $c_j^{opt} = c_{j-1}^{opt} + c(e)$, where $e = (v_{j-1}, v_j)$.

Let $\hat{\varepsilon} = \frac{\varepsilon}{1+\varepsilon}$. We have proved that $\tilde{c}_j \leq \frac{c_j^{opt}}{(1-\frac{\hat{\varepsilon}}{H})^j}$ for $1 \leq j \leq h$. This result implies that $\tilde{c} = \tilde{c}_h \leq \frac{c_{opt}}{(1-\frac{\hat{\varepsilon}}{H})^h}$. Moreover, since for $h \leq H$ it holds that $(1 - \frac{\hat{\varepsilon}}{H})^h \geq (1 - \frac{\hat{\varepsilon}}{H})^H$, and since $(1 - \frac{\hat{\varepsilon}}{H})^H$ is an increasing function of H, we conclude that $(1 - \frac{\hat{\varepsilon}}{H})^h \geq 1 - \hat{\varepsilon}$ and $\tilde{c} \leq \frac{c_{opt}}{1-\hat{\varepsilon}}$. Therefore $\frac{\tilde{c}-c^{opt}}{\tilde{c}} \leq \hat{\varepsilon}$, i.e., $\frac{\tilde{c}-c^{opt}}{c^{opt}} \leq \frac{\hat{\varepsilon}}{1-\hat{\varepsilon}}$. Since $\hat{\varepsilon} = \frac{\varepsilon}{1+\varepsilon}$, it holds that $\frac{\tilde{c}-c^{opt}}{c^{opt}} \leq \varepsilon$ and the lemma follows. ■

Lemma 12: The computational complexity of Algorithm RSP-DAG is $O(\frac{1}{\varepsilon} \cdot MH \log C)$, where C is an upper bound on the cost of a path in G.

Proof: Let us count the number of iterations k of the algorithm's main loop (i.e.the loop beginning on line 7). Let $\hat{\varepsilon} = \frac{\varepsilon}{1+\varepsilon}$. Clearly, $\frac{1}{(1-\frac{\hat{\varepsilon}}{H})^k} \leq C$, thus $k \leq \frac{-\ln C}{\ln(1-\frac{\hat{\varepsilon}}{H})}$. Since for all $x > -1$ it holds that $\ln(1 + x) \leq x$, we have that $k \leq \frac{H \ln C}{\hat{\varepsilon}}$. Each iteration of the main loop requires $O(M)$ time,hence the complexity of the algorithm is $O(1/\hat{\varepsilon} \cdot MH \ln C)$. Since $\hat{\varepsilon} \geq \frac{\varepsilon}{2}$ for $\varepsilon \leq 1$, it follows that the algorithm's complexity is $O(1/\varepsilon \cdot MH \ln C)$. ■

The next section extends Algorithm RSP-DAG to general graphs. This requires only minor changes to the algorithm.

B.2. Extension to general graphs

Recall that, in each iteration of Algorithm RSP-DAG, the graph nodes were visited in a topologically sorted order. Since such order does not exist in graphs with cycles, we process nodes according to their minimum weights from the source, using an idea similar to that of Dijkstra's shortest path algorithm. The algorithm is presented in Fig. 12.

Theorem 4: Given are a general graph G, a source node s and an approximation parameter ε. For a (arbitrary) value $\tilde{w}$, let c^{opt} be the cost of a $\tilde{w}$-weight constrained optimal path from s to a (arbitrary) node $v \in V$, and let $\tilde{c} = \min_{i=1,2,...\lfloor \log_\delta C \rfloor} \{c_i | W_v[c_i] \leq \tilde{w}\}$, where the values $W_v[c_i]$ are the output of Algorithm RSP-GEN for G, s, ε and v. Then $\frac{\tilde{c}-c^{opt}}{c^{opt}} \leq \varepsilon$.

Proof: Straightforward,since the algorithm is essentially similar to Algorithm RSP-DAG, except for the order by which nodes are visited during an iteration of the loop at line 7. The correctness of the algorithm follows from the fact that nodes with lower values of $W_v[c_i]$ are visited first, as in Algorithm RSP-DAG. ■

Theorem 5: The computational complexity of Algorithm RSP-GEN is $O(\frac{1}{\varepsilon} \cdot (M + N \log N)H \log C)$, where C is an upper bound on the cost of a path in G.

Proof: The algorithm performs the same number of iterations as Algorithm RSP-DAG, i.e., $O(\frac{1}{\varepsilon} \cdot H \log C)$. It can be easily verified that each iteration incurs $O(M + N \log N)$ computational complexity. We thus conclude that the computational complexity of the algorithm is $O(\frac{1}{\varepsilon} \cdot (M + N \log N)H \log C)$. ■

B.3. The Second Phase

Algorithm RSP-GEN constitutes the first phase of our precomputation scheme, and its output, i.e. the values $W_v[c_i]$, is used by the second phase of the precomputation scheme.

Algorithm* RSP-GEN *(*$G(V,E), s, \varepsilon$*):

parameters:
$G(V,E)$ - network
$s \in G$ - source node
ε- approximation parameter

variables:
c_i - the "budget"
for all $v \in V$
$W_v[c_i]$ - the approximated minimum weight of a path between s and v whose cost is at most c_i
Q - priority queue[a]

notation
C- an upper bound on the cost of a path in G
$\delta = (1/(1 - \frac{\varepsilon}{(1+\varepsilon)\cdot H})$
$\Downarrow c \Downarrow = \lfloor \delta^{\lfloor \log_\delta c \rfloor} \rfloor$

1 **for** all $v \in V$ **do**
2 $W_v[0] \leftarrow \infty$
3 $W_s[0] \leftarrow 0$
4 $c_0 \leftarrow 0$
5 $i \leftarrow 0$
6 $c_i \leftarrow 1$
7 **while** $c_i \leq C$ **do**
8 $W_s[c_i] \leftarrow 0$
9 $Q \leftarrow \emptyset$
10 **for all** $v \in V$ **do**
11 $W_v[c_i] \leftarrow W_v[c_{i-1}]$
12 **Add**$(Q, v, W_v[c_i])$;
13 **while**$(Q \neq \emptyset)$ **do**
14 $u \leftarrow$**Extract_Min**(Q)[b];
15 **for** each node $v \in Adj[u]$ **do**
16 let $e = (u, v)$
17 **if**$(c(e) \leq c_i)$ **then**
18 $W_v[c_i] \leftarrow \min[W_v[c_i], W_u[\Downarrow c_i - c(e) \Downarrow] + w(e)]$
19 $i \leftarrow (i+1)$
20 $c_i \leftarrow \lfloor \delta^i \rfloor$
21 **return** $\{W_v[c_i] \mid i = 1, 2, ... \lfloor \log_\delta C \rfloor\}$ for each $v \in V$

[a] The priority queue Q is implemented with a Fibbonacci heap [6]. Two operations are supported:**Add**(Q, v, w) and **Extract_Min**(Q).
[b] With Fibonacci Heaps, the amortized cost of each priority tree operation is $O(\log V)$ [6].

Fig. 12. Algorithm RSP-GEN

The second phase is invoked at a source node s, upon a connection request between s and a destination node $d \in V$, with a QoS requirement $\hat{w}$. The scheme then determines the minimum c_i for which $W_d[c_i] \leq \hat{w}$. This operation can be performed in $O(\log(\frac{1}{\varepsilon}) + \log(H) + \log\log(C))$ time by means of a binary search on the values of $W_d[c_i]$. The scheme reports the path between s and d that corresponds to $W_d[c_i]$, which, by Theorem 4, is a $\hat{w}$-weight constrained path between s and d with an ε-optimal cost. This path is not determined by the first phase explicitly, but can be derived from its output, in $O(|\mathbf{p}|)$ operations[5]. Therefore, the second phase incurs a total computational complexity of $O(\log(\frac{1}{\varepsilon}) + \log(H) + \log\log(C) + |\mathbf{p}|)$.

C. Discussion

We described a precomputation scheme for Problem RSP that provides ε-optimal solutions within a computational complexity of $O((M+N\log N)\cdot\frac{H}{\varepsilon}\log C)$ for the first phase and $O(\log(\frac{1}{\varepsilon})+\log(H)+\log\log(C)+|\mathbf{p}|)$ for the second phase.

Compared with an alternative single-phase (i.e., "no precomputation") scheme, our scheme allows to (significantly) reduce the time required for establishing a new connection. Indeed, in a single-phase scheme

[5] This requires a mild and straightforward modification of Algorithm RSP-GEN. For simplicity of exposition, the details are omitted here.

Problem RSP should be solved for each connection request, through a standard ε-optimal approximation to Problem RSP [12], which incurs a computational complexity of $O(M\frac{H}{\varepsilon}\log\log C)$ for DAGs, and $O((M + N\log N)\frac{H}{\varepsilon}\log\log C)$ for general graphs[6]. Hence, the second phase of our scheme allows to identify an ε-optimal path upon a connection request $(\log N \cdot \frac{H}{\varepsilon} \cdot \log\log C)$ times faster.

As previously noted, a precomputation scheme can be trivially constructed on the base of existing approximation algorithms for Problem RSP, such as [12], by sequentially executing them for various cost values. The computational complexity of this solution, for a *single* destination, is $O((M + N\log N)\frac{H}{\varepsilon}\log\log C)$ for general graphs. In order to perform the precomputation for Problem **RSP** , this algorithm should be invoked $O(\frac{\log C}{\varepsilon})$ times *per destination*, with a total complexity of $O((M + N\log N) \cdot \frac{H}{\varepsilon}\log\log C\frac{\log C}{\varepsilon}N)$ for all destinations, which is significantly ($\frac{\log C}{\varepsilon}N$ times) higher than that of our solution.

IX. Conclusions

QoS routing poses major challenges in terms of algorithmic design. On one hand, the path selection process is a complex task, due to the need to concurrently deal with the connection's QoS requirements, as well as with the global utilization of network resources; on the other hand, connection requests need to be handled promptly upon their arrival, hence there is limited time to spend on path selection. In many practical cases, a precomputation scheme offers a suitable solution to the problem: a background process (the "first phase") prepares a data base, which enables to identify a suitable path upon each connection request, through a simple, fast, procedure (the "second phase").

While much work has been done in terms of path selection algorithms, the precomputation perspective received little attention. As was demonstrated in this paper, simplistic adaptations of standard algorithms are usually inefficient.

Accordingly, this study considered the precomputation perspective, focusing on two major settings of QoS routing. First, we considered the (practically important) special case where the QoS constraint is of the "bottleneck" type, e.g. a bandwidth requirement, and network optimization is sought through hop minimization. For this setting, the standard Bellman-Ford algorithm offers a straightforward precomputation scheme. However, we showed that, by exploiting the typical hierarchical structure of large-scale networks, one can achieve a substantial ($O(\frac{H}{\log N})$) improvement in terms of computational complexity. Then, we turned to consider the more general setting of "additive" QoS constraints (e.g., delay) and general link costs. As the routing problem is NP-hard, we focused on ϵ-optimal approximations, and derived a precomputation scheme that offers a major ($\frac{\log C}{\varepsilon}N$) improvement over a "standard" approach.

Some topics are the subject of ongoing research. These include: (i) precomputation schemes for the (NP-hard) Problem RSP, which are based on *Lagrangian relaxation* techniques; (ii) precomputation schemes for Problem RSP in *hierarchical* networks; (iii) establishing an algorithmic technique for (automatically) partitioning a hierarchical network into the corresponding peer groups.

Finally, we note that, except for unicast path selection, there are many other networking problems, such as flow optimization, spanning tree minimization, multicast tree optimization, etc., for which the precomputation perspective offers a rich ground for future research.

References

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

[2] G. Apostolopoulos, R. Guérin, S. Kamat, A. Orda, T. Przygienda, and D. Williams. QoS routing mechanisms and OSPF extensions. Internet Draft, December 1998.

[3] G. Apostolopoulos, R. Guérin, S. Kamat, A. Orda, and S. Tripathi. Intra-Domain QoS Routing in IP Networks: A Feasibility and Cost/Benefit Analysis. *IEEE Network Magazine*, 1999. To appear.

[4] G. Apostolopoulos, R. Guérin, S. Kamat, and S. Tripathi. Quality of service based routing: A performance perspective. In *Proceedings of SIGCOMM*, pages 17–28, Vancouver, Ontario, CANADA, September 1998.

[5] A. Bestavros and I. Matta. Load Profiling for Efficient Route Selection in Multi-Class Networks. In *Proceedings of IEEE ICNP'97*, Atlanta, GA, October 1997.

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

[7] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick. A Framework for QoS-based Routing in the Internet – RFC No. 2386. Internet RFC, August 1998.

[6] The last statement, regarding general graphs, does not appear in [12], but can be easily verified.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability*. Freeman, San Francisco, 1979.
[9] R. Guerin and A. Orda. Computing Shortest Path for Any Number of Hops. 1998. Unpublished manuscript.
[10] R. Guérin and A. Orda. QoS-based routing in networks with inaccurate state and metrics information. *IEEE/ACM Transactions on Networking*, 1999. To appear.
[11] R. Guérin, A. Orda, and D. Williams. QoS routing mechanisms and OSPF extensions. Internet Draft, December 1996. Also in Proceedings of the 2nd IEEE Global Internet Mini-Conference, Phoenix, AZ, November 1997.
[12] R. Hassin. Approximation schemes for the restricted shortest path problem. *Mathematics of Operations Research*, 17(1):36–42, February 1992.
[13] D. H. Lorenz and A. Orda. QoS Routing in Networks with Uncertain Parameters. *IEEE/ACM Transactions on Networking*, 6(6):768–778, December 1998.
[14] D. H. Lorenz and A. Orda. Optimal Partition of QoS Requirements on Unicast Paths and Multicast Trees. In *Proceedings of IEEE INFOCOM'99*, New York, NY, March 1999.
[15] Q. Ma and P. Steenkiste. Quality of Service Routing for Traffic with Performance Guarantees. In *Proceedings of IWQoS'97*, Columbia University, New York, NY, May 1997.
[16] A. Orda. Routing with End to End QoS Guarantees in Broadband Networks. *IEEE/ACM Transactions on Networking*, 1999. To appear.
[17] C. Pornavalai, G. Chakraborty, and N. Shiratori. QoS Based Routing Algorithm in Integrated Services Packet Networks. In *Proceedings of IEEE ICNP'97*, Atlanta, GA, October 1997.
[18] Z. Wang and J. Crowcroft. Quality-of-Service Routing for Supporting Multimedia Applications. *IEEE JSAC*, 14(7):1288–1234, September 1996.