# Precomputation Schemes for QoS Routing

Ariel Orda, Senior Member, IEEE, and Alexander Sprintson, Student Member, IEEE

Abstract—Precomputation-based methods have recently been proposed as an instrument to facilitate scalability, improve response time, and reduce computation load on network elements. The key idea is to effectively reduce the time needed to handle an event by performing a certain amount of computations in advance, i.e., prior to the event's arrival. Such computations are performed as background processes, thus enabling to promptly provide a solution upon a request, through a simple, fast procedure.

In this paper, we investigate precomputation methods in the context of Quality-of-Service (QoS) routing. Precomputation is highly desirable for QoS routing schemes due to the high computation complexity of selecting QoS paths on the one hand, and the need to promptly provide a satisfactory path upon a request on the other hand. We consider two major settings of QoS routing. The first is the case where the QoS constraint is of the "bottleneck" type, e.g., a bandwidth requirement, and network optimization is sought through hop minimization. The second is the more general setting of "additive" QoS constraints (e.g., delay) and general link costs.

This paper mainly focuses on the first setting. First, we show that, by exploiting the typical hierarchical structure of large-scale networks, one can achieve a substantial improvement in terms of computational complexity. Next, we consider networks with topology aggregation. We indicate that precomputation is a necessary element for any QoS routing scheme and establish a precomputation scheme appropriate for such settings. Finally, we consider the case of additive QoS constraints (e.g., delay) and general link costs. As the routing problem becomes  $\mathcal{NP}$ -hard, we focus on  $\varepsilon$ -optimal approximations and derive a precomputation scheme that offers a major improvement over the standard approach.

Index Terms—Hierarchical networks, precomputation, Quality of Service (QoS), routing, topology aggregation.

# I. INTRODUCTION

N RECENT years, we have witnessed considerable accomplishments in the design, development, and deployment of broadband communication networks. Network capabilities, in particular those of the Internet, grow at a remarkable rate. At the same time, a phenomenal growth in data traffic and a wide range of new requirements of emerging applications call for new mechanisms for the control and management of communication networks. This poses some major challenges. Not only are many problems intrinsically difficult, but there are also additional constraints, such as limitations on the computational capabilities of network elements. In addition, any such control and management mechanism must scale well with network growth and provide fast response to internal (e.g., link failure) and external (e.g., connection request) events.

Manuscript received March 14, 2001; revised March 14, 2002; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor P. Steenkiste.

The authors are with the Department of Electrical Engineering, Technion—Israel Institute of Technology, Haifa 32000, Israel (e-mail: ariel@ee.technion.ac.il; spalex@tx.technion.ac.il).

Digital Object Identifier 10.1109/TNET.2003.815299

Precomputation-based methods have recently been proposed as an instrument for facilitating scalability, improving response time, and reducing the computational load on network elements. The key idea is to effectively reduce the time needed to handle an event by performing a certain amount of computations in advance, i.e., prior to the event's arrival. Such advance computations are performed as background processes, i.e., when a network element is idle or underutilized, thus, resulting in better utilization of the computational capabilities of network elements. In addition, when the rate of external events is high, a considerable reduction in overall computational load is achieved.

Precomputation is performed by means of a two-phase procedure, which we refer to as a precomputation scheme. The first phase is executed in advance and its purpose is to precompute solutions a priori for a wide set of possible event parameters. The computations performed at this phase are then summarized in a database for later usage. The second phase is activated when an event arrives and its purpose is to promptly provide an adequate solution for the event's parameters. The second phase either selects one of the solutions precomputed at the first phase, or, if necessary, performs a few additional computations. For instance, when handling connection requests with delay constraints, the first phase may precompute paths for a wide range of possible delay constraints, while the second phase just needs to select a suitable path from the precomputed database, i.e., one that satisfies the particular delay constraints of the connection. The execution time of the second phase has an immediate impact on network performance, hence, it is highly desirable to keep its computational complexity as low as possible. In the above example, the less time consumed in finding the proper path, the less time is consumed in establishing the new connection.

We conclude that precomputation is a highly desirable scheme and, at times, a necessary component for the efficient control and management of broadband networks. We proceed to discuss its major benefits in some more detail.

Enhancing scalability. As networks grow in size, appropriate control mechanisms must scale well with network growth. The two major strategies for achieving scalability are limiting the amount of link state information, and reducing the computational load of network elements. Precomputation methods constitute a useful tool for both strategies. Indeed, in many typical settings, where the rate of event arrivals is high, precomputation allows reduction of the overall computational load. Furthermore, as will be discussed below, precomputation methods are necessary for handling topology aggregation, a major technique for obtaining scalability.

**Improving fault tolerance.** A failure of a network element (e.g., link or node) must be handled properly, for example, by rerouting the existing connections to alternative routes. Failures

are handled much faster if some computations are performed in advance. For instance, an alternative path can be precomputed for each possible link failure.

**Improving performance in bursty conditions.** Under bursty conditions, a new connection request might arrive before the handling of a previous request has been completed. Computations performed prior to the burst reduce the time needed for handling a request.

Improving load balancing. In a precomputation scheme, a number of potential requests are processed through the same procedure. This facilitates distribution of available resources among different requests in an efficient manner. For example, consider a setting in which packets are sent along shortest (or almost-shortest) paths, determined by the source node. For such a setting, a precomputation scheme would identify a number of shortest and near-shortest paths in advance and supply different paths to different connection requests, effectively facilitating load balancing.

In fact, several existing network mechanisms employ some form of precomputation. As a straightforward example, consider standard IP routing, where each packet is forwarded by a router according to its *precomputed* routing table.

As will be demonstrated, many of the algorithmic tools that are often proposed as building blocks for network control and management were not designed with precomputation in mind, and better results can be obtained when such a scheme is considered. The problem of how to efficiently precompute a set of solutions for a wide range of parameters effectively opens a new area of research. We note that the running time of the precomputation scheme is important due to the following reasons. First, the time available for precomputation is limited because the network element has other off-line tasks. A second limitation arises from the need to invoke the precomputation scheme upon changes in the link state, because such changes may invalidate the precomputed solutions.

In this paper, we focus on the precomputation perspective of Quality-of-Service (QoS) routing. QoS routing is, undoubtedly, one of the major building blocks for supporting QoS and, hence, a necessary component of future communication networks. Indeed, it has been the subject of several recent studies and proposals (see, e.g., [6], [7], [9], [11], [16], [19], [21], [24], [25], [27], [28], and references therein).

QoS routing is, in general, a complex problem, for 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. 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 [1], [2]. Bandwidth, in turn, belongs to the class of "bottleneck" path requirements, which are much easier to handle than "additive" requirements, such as delay, loss, or jitter [11], [16], [17]. As for global network optimization, often it turns out that much can be achieved by employing the simple criterion of *hop minimization* [1], [3]; 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 component of several recent proposals for IP-oriented QoS routing protocols [9]. Luckily, hop minimization turns out to be an optimization criterion that is relatively easy to handle.

The high complexity associated with QoS routing on the one hand and the requirement of fast path selection on the other hand make precomputation highly desirable for QoS routing schemes. Accordingly, this paper mainly focuses on the problem of precomputing paths of maximal bandwidth for each possible hop-count value. This problem was initially investigated in [12], and was termed there as the *all-hops optimal path* problem (AHOP). While a trivial solution to that problem is offered by the standard Bellman–Ford algorithm [8], in [12] an algorithm with a lower worst case bound is presented; yet the improvement is achieved only in dense (highly connected) topologies, while communication networks usually have a sparse topology. In this paper, we show that, by exploiting the *hierarchical structure* typical of large-scale networks, a better solution in terms of computation complexity can be obtained.

Next, we consider QoS routing in networks with topology aggregation, which improves the scalability of link state protocols by effectively limiting the amount of link state information stored at a node. With topology aggregation, subnetworks, or routing domains, do not reveal the details of their internal structure, but rather supply the aggregated representation to the outside world. The aggregated representation summarizes traversal characteristics of a routing domain. It may include, for example, the description of paths available across the domain. The aggregated representation is not computed for a specific event parameter, such as required path delay, but for a wide range thereof; therefore, as we will see, precomputation techniques are an efficient tool for calculating aggregated representations. We indicate that precomputation is a necessary element for performing QoS routing in such settings and establish an appropriate precomputation scheme.

Finally, we demonstrate the (wide) scope of problems that can benefit from precomputation techniques by considering the harder case of *additive* QoS requirements and *general* (additive) path optimization criteria (i.e., other than hop minimization). The respective problem becomes a variant of the *restricted shortest path* (RSP) problem, which is known to be  $\mathcal{NP}$ -hard [10]. Some general approximation schemes that are  $\varepsilon$ -optimal have been proposed (see, e.g., [26] 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 this paper we establish an approximation scheme which offers both efficient solutions as well as efficient performance for precomputing "optimal" (minimum cost) paths for all possible values of an additive OoS requirement.

The rest of this paper is organized as follows. First, in Section II, we formulate the network model and formally state the considered problems. In Section III, we consider the problem of hop minimization with bottleneck QoS constraints in hierarchical networks and present and analyze our precomputation scheme. In Section IV, we extend our scheme for networks with

topology aggregation. In Section V, we consider additive QoS constraints and general (additive) path costs, and we present and analyze our precomputation scheme and demonstrate its advantages over standard alternatives. Finally, conclusions appear in Section VI. Due to space limitations, some proofs and technical details are omitted and can be found in [22].

#### II. MODEL AND PROBLEM FORMULATION

This section formulates the general model and main problems addressed in this paper.

#### A. Network Model

We begin with a definition of a *general* communication network; definitions of some specific classes, namely, *hierarchical* and *aggregated* networks, will be introduced in the following sections.

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. We denote by N and M the number of network nodes and links, respectively, i.e., N = |V| and M = |E|. An (s, t)-path is a finite sequence of distinct nodes  $\mathcal{P} = \{s = v_0, v_1, \ldots, t = v_h\}$ , such that, for  $0 \le i \le h-1$ ,  $(v_i, v_{i+1}) \in E$ ;  $h = |\mathcal{P}|$  is then said to be the number of hops (or hop count) of  $\mathcal{P}$ . The subpath of  $\mathcal{P}$  that extends from  $v_i$  to  $v_j$  is denoted by  $\mathcal{P}_{(v_i, v_j)}$ . Let  $\mathcal{P}_1$  be a  $(u_1, u_2)$ -path and  $\mathcal{P}_2$  be a  $(u_2, u_3)$ -path; then,  $\mathcal{P}_1 \circ \mathcal{P}_2$  denotes the  $(u_1, u_3)$ -path formed by concatenation of  $\mathcal{P}_1$  and  $\mathcal{P}_2$ . We denote by H the maximum possible hop count of any path  $\mathcal{P}$  in G that may be considered for routing purposes. Obviously,  $H \le N - 1$ , and it is much smaller in many typical network topologies.

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. Accordingly, the  $path\ weight\ W(\mathcal{P})$  of a path  $\mathcal{P}$  is defined differently for additive metrics, such as delay, than for bottleneck metrics, such as bandwidth. When link weights constitute an additive metric, the  $weight\ W(\mathcal{P})$  of a path  $\mathcal{P}$  is defined as the sum of weights of its links, i.e.,  $W(\mathcal{P}) = \sum_{e \in \mathcal{P}} w_e$ . When link weights constitute a bottleneck metric, the  $weight\ W(\mathcal{P})$  of a path  $\mathcal{P}$  is defined as the weight of its worst link, i.e.,  $W(\mathcal{P}) = \max_{e \in \mathcal{P}} w_e$ .

We can define the notion of a path that is "best" when only path weights are considered. A *minimum-weight* (s, t)-path is a path  $\mathcal{P} = \{s, \ldots, t\}$  whose weight is no larger than that of any other (s, t)-path.

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  $\mathcal{O}(M+N\log N)$  computational complexity [8]. 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 it may 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 its number of 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  $c_e > 0$  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  $cost\ C(\mathcal{P})$  of path  $\mathcal{P}$  is defined to be the sum of the costs of its links, i.e.,  $C(\mathcal{P}) = \sum_{e \in \mathcal{P}} c_e$ .

In this paper, we will 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. For clarity, we say that a path  $\mathcal P$  is a w-weight constrained if its weight is no more than w; similar definitions apply to h-hop constrained paths and c-cost constrained paths.

#### B. Problem Formulation

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

Problem RSP: Given a source node s, a destination node t, and a QoS requirement  $\hat{w}$ , find a minimum-cost path  $\hat{\mathcal{P}}$  between s and t such that  $W(\hat{\mathcal{P}}) \leq \hat{w}$ .

We refer to a solution of Problem RSP as a  $\hat{w}$ -weight constrained optimum (s, t)-path.

For additive weights and general costs, Problem RSP is intractable, i.e.,  $\mathcal{NP}$ -hard [10]. However, there exist pseudo-polynomial solutions, based on dynamic programming, which give rise to fully polynomial approximation schemes (FPAS), whose computational complexity is reasonable (see, e.g., [26] and references therein).

As mentioned above, many QoS routing problems consist of identifying, for each connection request, a path of minimum hop count 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 1) weights are of the bottleneck type and 2) links have equal costs; each of these two simplifications renders Problem RSP tractable. The first goal of this paper 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 a source node s, a destination node t and a bottleneck QoS requirement  $\hat{w}$ , find a path  $\hat{\mathcal{P}}$  between s and t of minimum hop count such that  $W(\hat{\mathcal{P}}) < \hat{w}$ .

As mentioned in Section I, 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 prepares a database with precomputed paths for any possible QoS requirement, and the second phase promptly retrieves the required path from the database upon a connection request.

Precomputation schemes for equal link costs (i.e., minimum hops) were investigated in [1] and [12], for bottleneck as well as additive weights. A simple precomputation scheme, which "inverts" the roles of the constraint (QoS requirement) and the optimization criterion (hops), was proposed in [1]. In particular, that 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 an h-hop constrained optimal (s, t)-path to be a path of minimum weight among all (s, t)-paths with hop count of at most h. The AHOP problem was then formulated in [12] as follows.

Problem AHOP: Given are a graph G=(V,E), a source node  $s \in V$  and a maximum hop count H. Find, for each hop value  $h, 1 \le h \le H$  and each destination node  $t \in V$ , an h-hop constrained optimal (s,t)-path.

We will contrast our precomputation scheme with schemes that are based on solving Problem AHOP. In addition, we will 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 the basic underlying (RSP) problem is intractable. Therefore, in Section V we resort to precomputing *approximated* ( $\varepsilon$ -optimal) solutions.

# III. PRECOMPUTATION SCHEME FOR PROBLEM BH-RSP IN HIERARCHICAL NETWORKS

In this section, we present and analyze our precomputation scheme for the problem of hop minimization with bottleneck QoS constraints, i.e., Problem BH-RSP, in hierarchical networks.

A possible approach for devising a precomputation scheme is to fully precompute all solutions during the first phase. With this approach, the second phase just consists of searching for the solution in the database produced by the first phase, according to the specific QoS requirement of the connection request. Such is indeed the precomputation scheme for Problem BH-RSP proposed in [12], which consists of precomputing all paths for all possible bandwidth requirements, i.e., solving Problem AHOP. The Bellman-Ford shortest path algorithm provides a simple precomputation scheme for solving Problem AHOP with a computational complexity of  $\mathcal{O}(MH)$ ; for a general (dense) topology, that bound can grow to be as large as  $\mathcal{O}(N^2H)$ . In [12], an alternative scheme is described whose computational complexity is  $\mathcal{O}(N^2H/\log N)$ ; evidently, the latter outperforms Bellman-Ford's in dense topologies, i.e., when  $M > N^2/\log N$ , but not in sparse topologies, which are the typical case of communication networks. The computational complexity incurred by the second phase of both schemes is just  $\mathcal{O}(\log H + |\hat{\mathcal{P}}|)$ , where  $|\hat{\mathcal{P}}|$  is the hop count of the identified solution.

It thus remained an open question whether one can devise a faster precomputation scheme for typical network topologies. In the following, 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 with a formulation of the hierarchical model.

# A. Hierarchical Model Formulation

A routing algorithm may be presented with a hierarchical topology due to two possible reasons. First, the (actual) topology intrinsically has a hierarchical structure, as is often the case with large-scale networks. Alternatively, the (actual) topology was hierarchically aggregated, as in the ATM PNNI recommendations [25].

We begin with the first case, assuming a *link state* routing environment, where the source node has a detailed image of the entire network. Beside being an interesting framework *per se*, it provides the required foundations for an extension that deals with the second case, which will be considered in Section IV.

We assume that the network has a certain *hierarchical structure*. In order to state the precise meaning of the last term, we need to introduce some additional terminology.

The network G=(V,E) is referred to as the actual network. Suppose that we partition the actual network nodes into some disjoint set of layer-1 peer groups (or clusters). Furthermore, suppose that we repeat the above process, such that, for each k>1, layer-k peer groups are combined into layer-(k+1) peer groups. 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 nodes that form a layer-1 peer group  $\Gamma$  are its children and  $\Gamma$  is their parent. Similarly, layer-(k-1) peer groups that form a layer-k peer group  $\Gamma$  are referred to as the children of  $\Gamma$ ;  $\Gamma$  is referred to as their parent. We denote by  $H_{\Gamma}$  the maximum hop count of a path in a peer group  $\Gamma$  that can be considered for routing purposes. As we will see,  $H_{\Gamma}$  plays an important role in our scheme.

A node in a peer group  $\Gamma$ , which has a neighbor that does not belong to  $\Gamma$ , is called a *border* node of  $\Gamma$ . For each peer group  $\Gamma$  that includes the source node s, we consider s to be a border node of  $\Gamma$ . We also assume that a path between two nodes of a peer group does not cross the peer group's boundary.

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, for all k,  $1 \leq k \leq K$ , each layer-k peer group has a small number (at most d) of children, and, at the same time, the number of border nodes in a peer group is also small (at most b). Formal definitions are given below.

Definition 1: A network G(V, E) is said to be (d, b)-hierarchical 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) For  $1 \le k \le K$ , each layer-k peer group is a union of at least two and at most d children.
- 2) The number of border nodes of each peer group is at most *b*.

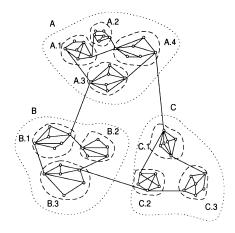


Fig. 1. Example of a hierarchical network.

Note that, since each layer-(k+1) peer group has at least two children, we have that  $K = \mathcal{O}(\log M) = \mathcal{O}(\log N)$ .

Let us illustrate the above terminology through an example. Fig. 1 depicts a (6, 2)-hierarchical network.  $A.1, \ldots, A.4, B.1, \ldots, B.3, C.1, \ldots, C.3$  are layer-2 peer groups, while A, B, C are layer-3 peer groups. In this example, K = 3.

We assume that the hierarchical structure of the network, i.e., partition into peer groups, is given. The next lemma establishes a "sparsity" property of hierarchical networks.

Lemma 1: The number of links in a (d, b)-hierarchical network is at most  $M = \mathcal{O}(b^2 dN)$ .

*Proof:* We divide the set  $\vec{E}$  of actual network links into the following subsets  $E_k$ , k = 0, ..., K - 1.

- 1)  $E_0$  includes all links in E that connect nodes of the same layer-1 peer group.
- 2)  $E_k$ , for k = 1, ..., K 1, includes all links in E that connect layer-k peer groups.

Each node v is connected to at most d nodes of its layer-1 peer group. Thus,  $|E_0| = \mathcal{O}(dN)$ . Since each layer-k peer group comprises of at least two children, the number of layer-k peer groups is  $\mathcal{O}(N/2^k)$ . Note that each layer-k peer group,  $k \geq 1$ , is connected to at most d layer-k peer groups. Note also that each two layer-k peer groups are connected by at most  $b^2$  links. Hence, for  $k \geq 1$ , it holds that  $|E_k| = \mathcal{O}(b^2dN/2^k)$ . We conclude that  $M = \sum_{k=0}^{K-1} |E_k| \leq \mathcal{O}(dN + b^2d\sum_{k=1}^{K-1} N/2^k) = \mathcal{O}(b^2dN)$  and the lemma follows.

In practical settings, d and b are small values. Typically, a network grows as a flat topology until its size reaches a certain threshold, which triggers the creation of a new peer group. As the network grows larger, more peer groups are added, but the size of each peer group remains bounded. Thus, the size of a peer group does not depend on the size of the network, hence, we can assume that  $d = \mathcal{O}(1)$ . Border nodes connect a peer group to its neighbors; in typical settings, the number of neighbors does not depend on the network size, i.e.,  $b = \mathcal{O}(1)$ . Hence, in such settings, we have  $\mathcal{O}(b^2d) = \mathcal{O}(1)$  and  $M = \mathcal{O}(N)$ .

#### B. Traversal Functions

The precomputation scheme proposed in [12] consists of precomputing all paths for all possible bandwidth requirements. Our precomputation scheme is based on a different approach. Rather than explicitly precomputing a set of paths for each

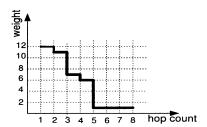


Fig. 2. Typical traversal function  $F_{(b_i,b_i)}^{\Gamma}(h)$ .

```
Algorithm BH-HIE (G, s):

input:

G- actual network;

s \in G- source node.

1 for k \leftarrow 1 to K-1
2 do for each layer-k peer group \Gamma
3 do for each border node b_i of \Gamma
4 do invoke Proc. CLUSTER for (G, \Gamma, b_i)
```

Fig. 3. Algorithm BH-HIE.

destination, our scheme computes traversal functions (defined below) for each peer group at each hierarchical layer. The traversal functions summarize the ability of a peer group to support QoS connections that may be established across it. Such an approach allows to exploit the network's hierarchical structure and yields an efficient precomputation scheme. In addition, this approach is useful in networks with topology aggregation, as will be shown in the next section.

Definition 2: Given a peer group  $\Gamma$  and two border nodes,  $b_1$  and  $b_2$  of  $\Gamma$ , the traversal function  $F^{\Gamma}_{(b_1,b_2)}(h)$  is defined to be the minimum weight of a  $(b_1,b_2)$ -path whose hop count is at most h.

A typical traversal function  $F_{(b_i,b_j)}^{\Gamma}(h)$  is depicted in Fig. 2. In this example, the minimum weight of a 5-hop constrained  $(b_i,b_j)$ -path across  $\Gamma$  is 1, while the minimum weight of a 3-hop constrained path is 7.

# C. First Phase: Algorithm BH-HIE

We proceed to describe Algorithm BH-HIE, which implements the first phase of our precomputation scheme. For each peer group  $\Gamma$  and for each pair  $(b_i, b_j)$  of  $\Gamma$ s border nodes, the algorithm computes the corresponding traversal function  $F_{(b_i,b_j)}^{\Gamma}(h)$ .

The key idea is to compute the traversal function of a peer group out of the traversal functions of its children. Accordingly, Algorithm BH-HIE runs across the hierarchical layers in a "bottom-up" manner, processing first peer groups of layer-1, then peer groups of layer-2, and so on, up the last, Ks layer. For each peer group  $\Gamma$  and for each border node  $b_0$  of  $\Gamma$ , the algorithm invokes Procedure Cluster, described in Section III-C1. The formal specification of Algorithm BH-HIE appears in Fig. 3.

1) Procedure CLUSTER: We proceed to describe Procedure CLUSTER, the main building block of Algorithm BH-HIE. Procedure CLUSTER receives as input some layer-k peer group  $\Gamma$ 

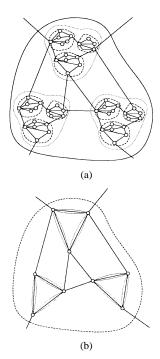


Fig. 4. Construction of auxiliary graph  $\overline{\Gamma}$ . (a) Peer group  $\Gamma$ . (b) Auxiliary graph  $\overline{\Gamma}$ .

and a node  $b_0$ , which is one of  $\Gamma$ s border nodes. It then (pre)computes the traversal function  $F^{\Gamma}_{(b_0,\,b)}(h)$  for each border node b of  $\Gamma$ . The order of processing of peer groups by Algorithm BH-HIE implies that, when Procedure CLUSTER is applied to a layer-k peer group  $\Gamma$ , it already has available the traversal functions  $\{F_{(\cdot,\cdot)}^{\Gamma_i}(h)\}$  for each child  $\Gamma_i$  of  $\Gamma$ .

Since layer-1 peer groups have flat topologies, we can employ the standard Bellman-Ford scheme. For all higher layers peer groups, however, we need more elaborate methods, in order to exploit their hierarchical structure.

The procedure starts by constructing the following auxiliary graph  $\overline{\Gamma}(\overline{V}, \overline{E})$ , whose purpose is to provide a concise representation of children and the connectivity among them. Each child  $\Gamma_i$  of  $\Gamma$  is represented in  $\overline{\Gamma}$  by the set  $\{b_{ij}\}$  of its border nodes. Each pair  $(b_{ij}, b_{ij'})$  of  $\Gamma_i$ s border nodes is connected by several links, each corresponding to a different hop count constraint. Specifically, for each  $h = 1, ..., H_{\Gamma_i}$ , we add a link e between  $b_{ij}$  and  $b_{ij'}$  in  $\overline{\Gamma}$ , with cost  $c_e = h$  and weight  $w_e = F^{\Gamma_i}_{(b_{ij},b_{ij'})}(h)$ . In addition, for each actual network link  $e(b_{ij}, b_{i'j'})$  that connects children of  $\Gamma$ , we add a link  $e'(b_{ij}, b_{i'j'})$  to  $\overline{\Gamma}$ , with weight  $w_{e'} = w_e$  and cost  $c_{e'} = 1$ . Fig. 4 illustrates the construction of the auxiliary graph for a layer-3 peer group.

The lemma below follows from the construction of  $\overline{\Gamma}$  and from the validity of traversal functions of the children  $\Gamma_i$  of  $\Gamma$ . Lemma 2: Let  $(b_i, b_j)$  be a pair of  $\Gamma$ s border nodes. Then:

- 1) for each  $(b_i, b_i)$ -path  $\mathcal{P}$  in  $\Gamma$ , there exists a  $(b_i, b_i)$ -path  $\overline{\mathcal{P}}$  in  $\overline{\Gamma}$  such that  $W(\overline{\mathcal{P}}) \leq W(\mathcal{P})$  and  $C(\overline{\mathcal{P}}) \leq |\mathcal{P}|$ ;
- 2) for each  $(b_i, b_i)$ -path in  $\overline{\Gamma}$ , there exists a  $(b_i, b_i)$ -path  $\mathcal{P}$ in  $\Gamma$  such that  $W(\mathcal{P}) = W(\overline{\mathcal{P}})$  and  $|\mathcal{P}| \leq C(\overline{\mathcal{P}})$ .

The lemma implies that we can use the auxiliary graph  $\Gamma$  for computing the traversal functions of  $\Gamma$ . Specifically, we need to find, for each border node  $b_i$  of  $\Gamma$  and for each  $1 \le c \le H_{\Gamma}$ , a minimum weight c-cost constrained  $(b_0, b_i)$ -path in  $\overline{\Gamma}$ . The key idea is to first remove (temporarily) all links from  $\overline{\Gamma}$  and then add them back to  $\overline{\Gamma}$  by increasing order of the weight values. During this process, we maintain the tree of minimum-cost paths in  $\overline{\Gamma}$ from the source node  $b_0$ ; we update the tree after each insertion of a link to  $\Gamma$ .

More specifically, for each  $v \in \overline{\Gamma}$ , we maintain the minimum cost  $c_v$  of a  $(b_0, v)$ -path in  $\overline{\Gamma}$ . In addition, we maintain an array  $T^{\Gamma}_{(b_0,\,v)}$ , such that:

- $T_{(b_0,v)}^{\Gamma}[c].w$  keeps the weight of a c-cost constrained optimum  $(b_0, v)$ -path  $\hat{\mathcal{P}}$  in  $\overline{\Gamma}$ ;
- $T^{\Gamma}_{(b_0,v)}[c].p$  keeps the predecessor u of v on  $\hat{\mathcal{P}}$ ;  $T^{\Gamma}_{(b_0,v)}[c].c$  keeps the cost of the last link (u,v) of  $\hat{\mathcal{P}}$ .

When we add a link e(u, v) to  $\overline{\Gamma}$ , we check whether the value  $c_v$  decreases; if it does, we update  $c_v$  and set

$$T^{\Gamma}_{(b_0, v)}[c_v].w \leftarrow w_e$$

$$T^{\Gamma}_{(b_0, v)}[c_v].p \leftarrow u$$

$$T^{\Gamma}_{(b_0, v)}[c_v].c \leftarrow c_e.$$

We perform a similar process for each node x for which  $c_x$  decreases as a result of adding link e. Upon completion, for each border node  $b_i$  of  $\Gamma$ , the traversal function  $F_{(b_0,b_i)}^{\Gamma}(h)$  is stored in the array  $T^{\Gamma}_{(b_0,b_i)}$ , i.e., for each h,  $1 \leq h \leq H^{\Gamma}_{\Gamma}$ 

$$F_{(b_0,b_i)}^{\Gamma}(h) = T_{(b_0,b_i)}^{\Gamma}[h].c.$$

The formal specification of Procedure CLUSTER appears in Fig. 5.

# D. Procedure FIND

We proceed to present Procedure FIND. This procedure is invoked upon each new connection request, and its goal is to identify a minimum hop  $\hat{w}$ -weight constrained (s, t)-path  $\hat{\mathcal{P}}$ .

For clarity, we denote by  $\Gamma^1$  the parent of t, by  $\Gamma^2$  the parent of  $\Gamma^1$ , etc., up to some peer group  $\Gamma^{\hat{k}}$  for which s is a border node. First, the procedure identifies, for each border node  $b_i$  of each peer group  $\Gamma^k$ , the minimum hop count  $H_{(b_i,t)}$  of a  $\hat{w}$ -weight constrained optimal  $(b_i, t)$ -path in  $\Gamma^k$ . Next, a suitable path is determined by Procedure PATH, which is described in the Appendix.

We begin with a layer-1 peer group  $\Gamma^1$ . We prune each link  $e \in \Gamma^1$  for which  $w_e > \hat{w}$ , and then apply a breadth-first search (BFS) algorithm [8] to the reversed graph, i.e., a graph in which each link appears in the reverse direction. This yields, for each border node  $b_i$  of  $\Gamma^1$ , a  $\hat{w}$ -weight constrained optimal  $(b_i, t)$ -path in  $\Gamma^1$  and its hop count  $H_{(b_i, t)}$ .

For a layer-k peer group  $\Gamma^k$ ,  $k \geq 2$ , we construct the following auxiliary graph  $\overline{\Gamma}^k$ . Each child  $\Gamma_i$  of  $\Gamma^k$  is represented in  $\overline{\Gamma}$  by the set  $\{b_{ij}\}$  of its border nodes. Each pair  $(b_{ij}, b_{ij'})$  of  $\Gamma_i$ s border nodes is connected a link e, whose cost is set to

$$c_e \leftarrow \min \left\{ h \middle| F_{(b_{ij}, b_{ij'})}^{\Gamma_i}(h) \le \hat{w} \right\}. \tag{1}$$

In addition, for each actual network link  $e(b_{ij}, b_{i'j'})$  that connects children of  $\Gamma^k$  and whose weight is no more than  $\hat{w}$ , we

```
Procedure CLUSTER (G, \Gamma, b_0):
         input:
             G- network
             \Gamma- layer-k peer group
             b_0 - \Gamma's border node
          1 \bar{\Gamma}(\bar{V}, \bar{E}) \leftarrow \text{Initialize}()
         2 for all v \in \bar{V} do
                  c_v \leftarrow H_\Gamma + 1
         4 \quad c_{b_0} \leftarrow 0
             T^{\Gamma}_{(b_0,b_0)}[0].w \leftarrow 0, T^{\Gamma}_{(b_0,b_0)}[0].p \leftarrow nil
         6 S \leftarrow \bar{E}
             for each link e(v, u) \in S by increasing order of w_e do
                  if exists a link e' between v and u in \bar{E} then
                     remove e' from \bar{E}
        10
                  add e to \bar{E}
        11
                  PROPAGATE(e(v, u))
        13 return T_{(b_0,b)}^{\Gamma} for each border node b of \Gamma
        PROPAGATE(e(u, v))
              if (c_u + c_e) < c_v then
                  for c \leftarrow (c_u + c_e) to (c_v - 1) do
                     T^{\Gamma}_{(b_0,v)}[c].w \leftarrow w_e, T^{\Gamma}_{(b_0,v)}[c].p \leftarrow u,
                                                          T^{\Gamma}_{(b_0,v)}[c].c \leftarrow c_e
                  c_v \leftarrow (c_u + c_e)
                  for each link e'(v,x) \in \bar{E} do
                      PROPAGATE(e'(v, x))
        INITIALIZE()
         1 \bar{V} \leftarrow \emptyset, \bar{E} \leftarrow \emptyset
         2 if k = 1 then
                  for each link e(v, u) \in \Gamma do
                     add a new link e' between v and u to \bar{E}
         5
                      c_{e'} \leftarrow 1, w_{e'} \leftarrow w_e.
         6
                  for each child \Gamma_i of \Gamma do
                      add each border node b_{ij} of \Gamma_i to \bar{V}.
         8
                      for each pair (b_{ij}, b_{ij'}) of \Gamma_i's border nodes do
        10
                         for c \leftarrow 1 to H_{\Gamma_i} do
                            add a new link e between b_{ij} and b_{ij'} to \bar{E}
        11
                            c_e \leftarrow c, w_e \leftarrow F^{\Gamma_i}_{(b_{ij}, b_{ij'})}(c)
        12
                  for each link e(b_{ij},b_{i'j'}) \in G that connects children \Gamma_i
        13
                                                                           and \Gamma_{i'} of \Gamma do
        14
                      add a link e' between b_{ij} and b_{i'j'} to \bar{E}
        15
                      c_{e'} \leftarrow 1, w_{e'} \leftarrow w_e
             return ar{\Gamma}(ar{V},ar{E})
```

Fig. 5. Procedure CLUSTER.

add a link  $e'(b_{ij},b_{i'j'})$  to  $\overline{\Gamma}$  with cost  $c_e=1$ . Finally, we connect by a link each border node  $b_j$  of  $\Gamma^{k-1}$  and the destination node t. The cost of such a link is set to the  $H_{(b_j,\,t)}$ ; the value of  $H_{(b_j,\,t)}$  was computed in the previous iteration. We note that a minimum-cost  $(b_i,\,b_j)$ -path in  $\overline{\Gamma}^k$  corresponds to a  $\hat{w}$ -weight constrained path in the actual network. Having constructed the auxiliary graph  $\overline{\Gamma}^k$ , we identify, for each border node  $b_i$  of  $\Gamma^k$ , the minimum-cost  $(b_i,\,t)$ -path  $\overline{\mathcal{P}}^{\Gamma^k}(b_i,\,t)$  in  $\overline{\Gamma}^k$ , by applying Dijkstra's algorithm on the reverse graph of  $\overline{\Gamma}^k$ . Then, we set  $H_{(b_i,\,t)} = C(\overline{\mathcal{P}}^{\Gamma^k}(b_i,\,t))$ . In the last step, Procedure FIND invokes Procedure PATH that identifies, for a peer group  $\Gamma^k$  and a border node  $b_i \in \Gamma^k$ , a  $\hat{w}$ -weight constrained optimal  $(b_i,\,t)$ -path. The formal specification of Procedure FIND appears in Fig. 6.

```
Procedure FIND (G, s, t, \hat{w}):
            G(V, E)- actual network
            s \in G- source node
            t \in G- destination node
            \hat{w}- (bottleneck) QoS requirement
         1 \Gamma^1 \leftarrow \text{parent of } t
             remove from \Gamma^1 all links e for which w_e \geq \hat{w};
              apply a Breadth First Search (BFS) algorithm [8] to the
              reversed graph \bar{\Gamma}^1 of \Gamma^{1a}
              for each border node b_i of \Gamma^1 store the hop count of
               (b_i, t)-path in \bar{\Gamma}^1 computed in Line 3 into H_{(b_i, t)}
              for k \leftarrow 1 to K do
                 \Gamma^k \leftarrow \text{parent of } \Gamma^{k-1}
                 for each child \Gamma_i of \Gamma^k do
                     add each border node b_{ij} of \Gamma_i to \bar{\Gamma}^k
         8
                     for each pair (b_{ij}, b_{ij'}) of \Gamma_i's border nodes do
        10
                        add a new link e between b_{ij} and b_{ij'} to \bar{E}
                        c_e \leftarrow \min\{h \mid F_{(b_{ij},b_{ij'})}^{\Gamma_i}(h) \le \hat{w}\}
        11
                 for each link e = (b_{ij}, b_{i'j'}) \in G that connects
        12
                                                     children \Gamma_i and \Gamma_{i'} of \Gamma^k do
        13
                     if w_e \leq \hat{w} then
                     add a link e' between b_{ij} and b_{i'j'} to \bar{\Gamma}^k
        14
        15
                 for each border node b_j of \Gamma^{k-1} do
        16
                     add a link e between b_i and t
        17
                     c_e \leftarrow H_{(b_i,t)}
        18
        19
                 apply Dijkstra's algorithm [8] for the reversed graph
        20
                 for each border node b_i of \Gamma^k store the cost of
                 (b_i,t)-path in \bar{\Gamma}^k computed in Line 19 into H_{(b_i,t)}
        21
                 if s is a border node of \Gamma^k then
        22
                     let \bar{P} be path returned by Dijkstra's algorithm
                     \hat{\mathcal{P}} \leftarrow \text{PATH}(\Gamma^k, \bar{\mathcal{P}})
        23
        24
                     return \hat{\mathcal{P}}
   a\bar{\Gamma}^1 includes a link \bar{e}(u,v) for each link e(v,u)\in\Gamma^1.
```

Fig. 6. Procedure FIND.

### E. Analysis of the Precomputation Scheme

Lemma 3: Suppose that the (correct) traversal function  $F^{\Gamma_i}_{(b_{ij},b_{ij'})}(h)$  is available for each pair  $(b_{ij},b_{ij'})$  of border nodes of each child  $\Gamma_i$  of  $\Gamma$ . Then, Procedure CLUSTER, applied on  $\Gamma$  and a border node  $b_0$  of  $\Gamma$ , computes the traversal function  $F^{\Gamma}_{(b_0,b_i)}(h)$  for each border node  $b_i$  of  $\Gamma$ .

*Proof:* See [22].

Theorem 1: Algorithm BH-HIE determines, for each peer group  $\Gamma$  at each hierarchical layer, the traversal function  $F_{(b_i,b_i)}^{\Gamma}(h)$  for each pair  $(b_i,b_j)$  of border nodes of  $\Gamma$ .

*Proof:* Straightforward by induction on hierarchical layers and application of Lemma 3.

Lemma 4: The computational complexity of Algorithm BH-HIE is  $\mathcal{O}(b^2d^2NK)$ .

*Proof:* See [22]. ■

Theorem 2: Procedure FIND provides an  $\mathcal{O}(b^2dK(d+\log H)+|\hat{\mathcal{P}}|)$  solution to Problem BH-RSP, i.e.,: given a connection request with source node s, destination node t, and (bottleneck) QoS constraint  $\hat{w}$ , and given the output of Algorithm BH-HIE, Procedure FIND identifies,

in  $\mathcal{O}(b^2dK(d+\log H)+|\hat{\mathcal{P}}|)$  steps, a  $\hat{w}$ -weight optimal (s,t)-path  $\hat{\mathcal{P}}$  in G.

*Proof:* See [22].

Note 1: In some settings the detailed path is not required. For example, in order to decide whether to admit a connection, we only need to know the minimum hop count of a QoS path. Then, the computational complexity of Procedure FIND is just  $\mathcal{O}(b^2dK(d+\log H))$ .

#### F. Discussion

We conclude this section with a performance comparison between our scheme and some alternatives.

Consider first the "standard" precomputation scheme proposed in [1] and [12], which was based on solving Problem AHOP through Bellman–Ford's shortest path algorithm. Lemma 1 implies that hierarchical networks are sparse, in the sense that  $M = \mathcal{O}(b^2dN)$ . This, in turn, implies that the standard scheme incurs a computational complexity of  $\mathcal{O}(b^2dNH)$  for its first phase, i.e., it is  $\Omega(H/(dK))$  times slower than ours. Since  $K = \mathcal{O}(\log N)$ , our scheme offers a significant improvement over the standard solution. The difference is particularly significant when  $d = \mathcal{O}(1)$  and  $K \ll H$ , which is a typical case.

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  $\mathcal{O}(\log H + |\hat{\mathcal{P}}|)$ , where  $\hat{\mathcal{P}}$  is the path identified by the scheme. This is somewhat less than that of our scheme, i.e.,  $\mathcal{O}(b^2dK(d+\log H) + |\hat{\mathcal{P}}|)$ . However, in typical settings, where  $b = \mathcal{O}(1)$ ,  $d = \mathcal{O}(1)$  and  $K \ll H$ , our scheme is just  $\Omega(\log H)$  times slower than the standard solution. Moreover, the difference is not significant when  $|\hat{\mathcal{P}}|$  is the dominating component.

Next, let us compare our precomputation scheme with an alternative where no precomputation is performed at all. In such a single-phase scheme, the required path can be identified by applying a BFS algorithm [8], which, for  $M = \mathcal{O}(b^2dN)$ , incurs  $\mathcal{O}(b^2dN)$  running time compared to  $\mathcal{O}(b^2dK(d+\log H)+|\hat{\mathcal{P}}|)$  of our scheme. Since  $K = \mathcal{O}(\log N)$ , our solution is  $\Omega(N/(\log N(d+\log H)+|\hat{\mathcal{P}}|))$  times faster. Typically,  $|\hat{\mathcal{P}}| \ll N$  and  $d = \mathcal{O}(1)$ , hence, the difference is significant.

It is interesting to compare 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  $\mathcal{O}(b^2dK(d+\log H))$  time (see Note 1), whereas the single-phase scheme still incurs  $\mathcal{O}(b^2dN)$  time. Thus, our solution is  $\Omega(N/(\log N(d+\log H)))$  times faster. The difference is significant in typical settings where  $d=\mathcal{O}(1)$ .

# <sup>1</sup>Recall that H is the maximum hop count of a path in G and, in the worst case, $H = \mathcal{O}(N)$ , while $K = \mathcal{O}(\log N)$ .

#### IV. PRECOMPUTATION SCHEME FOR AGGREGATED NETWORKS

In the previous section, we assumed that each node has full and unabridged information about link states, which is stored in its topology database. However, such an approach suffers from scalability problems. Indeed, as the network grows in size, significant resources are consumed for flooding and recording the changes in the link state throughout the network. With topology aggregation, subnetworks, or *routing domains*, do not reveal the details of their internal structure, but rather supply the aggregated representation to the outside world [15]. Such an approach could also be mandated by security and administrative needs. Topology aggregation is useful for ATM [25] and IP networks [20].

A key issue in topology aggregation is how to provide the routing information of a domain to the outside world. Constructing an accurate *aggregated representation* poses several complex challenges. First, there is a need to identify the *traversal characteristics* of a routing domain, i.e., its ability to support connections with different QoS requirements. Second, each domain, in turn, comprises of aggregated subdomains, whose traversal characteristics are available only through aggregated representation. Finally, each subdomain may use a different method for representing its routing information.

In this section, we establish that precomputation is a useful (virtually necessary) tool for constructing accurate aggregated representations. In particular, we show that, by using precomputation schemes, the traversal characteristics of a peer group can be computed efficiently. We employ the concept of *traversal functions*, introduced in Section III, in order to accurately represent the traversal characteristics. Further, we adapt our algorithmic techniques in order to cope with aggregated networks, i.e., networks in which each link represents an aggregated subdomain.

Also in this section, we consider a generic model for multilevel aggregated networks. The model can be used, for example, in conjunction with the ATM PNNI protocol [25], in which peer groups are represented by structures termed *complex nodes*.<sup>2</sup>

Similarly to the previous section, we focus on bottleneck QoS requirements and use hop minimization for optimizing resource utilization.

#### A. Aggregated Representation of a Peer Group

A significant body of research has been devoted to the area of topology aggregation; we proceed to quote a few relevant references. A compact  $\mathcal{O}(b)$  representation for undirected networks and a single bottleneck QoS requirement was presented in [15]. For additive QoS constraints, [23] shows that an accurate representation of a peer group requires  $\mathcal{O}(b^2)$  links in the worst case. An  $\mathcal{O}(b)$  representation that achieves a bounded distortion is presented in [4].

Devising a topology aggregation scheme that takes into account both the QoS constraints and the use of network resources is still an open research issue. A practical approach is to associate each pair of border nodes with two values: the first corresponds to a (bottleneck) weight and the second to a cost value

<sup>&</sup>lt;sup>2</sup>This requires a mild extension of the complex node structure. Specifically, we allow parallel bypass links, each link corresponding to a different value of the OoS constraint.

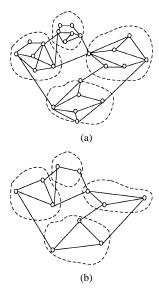


Fig. 7. (a) Actual and (b) aggregated networks.

(e.g., hop count). This approach provides a compact representation, but suffers from high distortion. In order to reduce distortion, some studies [5], [14], [18] present schemes that resemble the traversal functions introduced in Section III. Specifically, [5] proposes to use bandwidth-cost functions, i.e., functions that specify the available bandwidth for several cost values; [14] and [18] propose to approximate the bandwidth-cost functions by using curves and link segments, respectively. We note that the methods presented in the following for efficiently computing traversal functions can be used in order to compute the curves in [14] and line segments in [18], hence, resulting in a more efficient scheme.

### B. Aggregated Model

In multilevel aggregated topologies, a domain comprises aggregated subdomains. This gives rise to the following *aggregated network model*.

The actual network topology (i.e., with no aggregation) is represented by a directed graph G. The aggregated network is represented by a directed graph  $\hat{G}$ , in which each link  $e(v,u) \in \hat{G}$  represents an aggregated subdomain  $\Gamma^e$  of G. Fig. 7 depicts an example of an actual network and its corresponding aggregated topology. We assume that a node only knows the aggregated topology  $\hat{G}$ . We denote by H the maximum hop count of a path that can be considered for routing purposes in the actual network G. Each link  $e(v,u) \in \hat{G}$  is associated with a traversal function  $F_{(v,u)}(c)$ , which provides the minimum weight value that can be supported by  $\Gamma^e$  for each hop count limitation c. More specifically, for  $0 \le c \le H_{\Gamma_e}$ ,  $F_{(v,u)}(c)$  is the minimum weight of a c-hop constrained (v,u)-path across the subdomain  $\Gamma_e$ , where  $H_{\Gamma_e}$  is a maximum hop count of a path across  $\Gamma_e$  that can be considered for routing purposes.

Each path  $\hat{\mathcal{P}}$  in  $\hat{G}$  can support several QoS constraints at different costs. Accordingly, we define the cost  $C(\hat{\mathcal{P}}, \hat{w})$  of supporting the QoS constraint  $\hat{w}$  by  $\hat{\mathcal{P}}$ .

Definition 3: Given a path  $\hat{\mathcal{P}} = \{s = v_0, v_1, \dots, v_h = t\}$  in  $\hat{G}$  and a QoS constraint  $\hat{w}$ , we define, for each link  $e(v_{i-1}, v_i) \in \hat{\mathcal{P}}$ , the local cost of supporting  $\hat{w}$  to be

 $c_{(e,\,\hat{w})}=\min\{c|F_{(v_{i-1},\,v_i)}(c)\leq\hat{w}\}$ . The cost of satisfying the QoS constraint  $\hat{w}$  along the path  $\hat{\mathcal{P}}$  is then defined to be

$$C(\hat{\mathcal{P}}, \hat{w}) = \sum_{e \in \hat{\mathcal{P}}} c_{(e, \hat{w})}.$$

A traversal function in an aggregated network is defined as follows.

Definition 4: Given an aggregated network  $\hat{G}$ , a source node  $s \in \hat{G}$ , and a destination node  $t \in \hat{G}$ , we define the aggregated traversal function  $F_{(s,t)}^{\hat{G}}(c)$ ,  $1 \le c \le H$ , to be the minimum weight w of an (s,t)-path  $\mathcal{P}$  in  $\hat{G}$  for which  $C(\mathcal{P},w) \le c$ . If no such path exists,  $F_{(s,t)}^{\hat{G}}(c)$  is defined as  $\infty$ .

Intuitively, the aggregated traversal function in  $\hat{G}$  is identical to the traversal function in the actual network G. Furthermore, the traversal function  $F_{(v,\,u)}(c)$  that is associated with each link  $(v,\,u)\in\hat{G}$  is, in fact, the aggregated traversal function of the subdomain  $\Gamma^e$ . Note that  $\Gamma^e$  may, in turn, comprise aggregated subdomains.

We proceed to formulate the aggregated version of Problem AHOP (Agg-AHOP).

Problem Agg-AHOP: Let  $\hat{G}$  be an aggregated network, where each link  $e(v, u) \in \hat{G}$  is associated with a traversal function  $F_{(v,u)}(c)$ . For a source node  $s \in \hat{G}$  and each destination node  $t \in \hat{G}$ , find the aggregated traversal function  $F_{(s,t)}^{\hat{G}}(c)$ .

Problem Agg-AHOP can be solved by substituting each link  $e \in \hat{G}$  with several links, each link (v, u) being associated with a single weight w and cost value c, such that  $F_{(v,u)}(c) = w$ , and then applying the Bellman–Ford algorithm on the resulting graph. However, as the resulting graph includes  $\mathcal{O}(MH)$  links, this approach incurs a high computational complexity of  $\mathcal{O}(MH^2)$ . By using the algorithmic methods developed in the previous section, we can devise an alternative scheme for Problem Agg-AHOP, whose computational complexity is significantly lower.

#### C. Precomputation Scheme

Consider first a simple case, in which  $\hat{G}$  comprises just links (u,v) and (v,w), and our goal is to compute a traversal function  $F_{(v,w)}^{\hat{G}}(c)$ . We refer to this operation as merging the functions  $F_{(v,u)}(c)$  and  $F_{(u,w)}(c)$  into a single function  $F_{(v,w)}^{\hat{G}}(c)$ . The merge operation essentially amounts to computing, for each budget  $c,1\leq c\leq H$ , the partition  $(c^1,c^2)$  of the budget between the links (u,v) and (v,w) that minimizes the weight of a (u,w)-path in the actual network, i.e.,

$$F_{(v,w)}^{\hat{G}}(c) = \min_{c^1 + c^2 \le c} \left\{ \max \left\{ F_{(u,v)}(c^1), F_{(v,w)}(c^2) \right\} \right\}.$$

Our main observation is that, in the case of bottleneck QoS parameters, the merge operation can be performed in just  $\mathcal{O}(H)$  steps, through the following inductive process. Clearly, for budget  $c_0=2$ , the optimal partition is (1,1). Having computed the optimal partition  $(c_{i-1}^1,c_{i-1}^2)$  for a budget  $c_{i-1}$ , the optimal partition for a budget  $c_i=c_{i-1}+1$  is then either  $(c_{i-1}^1+1,c_{i-1}^2)$  or  $(c_{i-1}^1,c_{i-1}^2+1)$ :

$$F_{(u,w)}(c_i) = \min \left\{ \max \left\{ F_{(u,v)}(c_{i-1}^1 + 1), F_{(v,w)}(c_{i-1}^2) \right\}, \right.$$
$$\left. \max \left\{ F_{(u,v)}(c_{i-1}^1), F_{(v,w)}(c_{i-1}^2 + 1) \right\} \right\}.$$

The merge operation allows to solve Problem Agg-AHOP in acyclic directed graphs in  $\mathcal{O}(MH)$  time. For general directed graphs, we present a more elaborated algorithm that utilizes that same idea, and whose running time is  $\mathcal{O}(MH\log N)$ . The algorithm, referred to as Algorithm AGG-AHOP, is, in fact, an adaptation of Procedure CLUSTER (Section III) for networks with topology aggregation.

The algorithm starts by constructing the following auxiliary graph  $\overline{G}(\overline{V},\overline{E})$ , whose purpose is to represent traversal characteristics of subdomains and the connectivity among them. Each link e(v,u) of  $\hat{G}$  is represented in  $\overline{G}$  by several links, each corresponding to a different cost constraint. Specifically, for each  $c=1,\ldots,H_{\Gamma^e}$ , we add a link e' between v and v in  $\overline{G}$ , with cost  $c_{e'}=c$  and weight  $w_{e'}=F_{(v,u)}(c)$ , where  $\Gamma^e$  is the aggregated subdomain represented by e. For each  $v\in \overline{G}$ , we maintain the minimum cost  $c_v$  of an (s,v)-path in  $\overline{G}$ . In addition, we maintain array  $T_{(s,v)}^{\hat{G}}$ , such that

- $T^{\hat{G}}_{(s,v)}[c].w$  keeps the minimum weight of a c-cost constrained (s,v)-path  $\hat{\mathcal{P}}$  in  $\overline{G}$ ;
- $T^{\hat{G}}_{(s,v)}[c].p$  keeps the predecessor of v on  $\hat{\mathcal{P}}$ ;
- $T_{(b_0,v)}^{\Gamma^{'}}[c].c$  keeps the cost of the last link (u,v) of  $\hat{\mathcal{P}}.$

The key idea is to first remove (temporarily) all links from  $\overline{G}$  and then add them back to  $\overline{G}$  by increasing order of the weight values. When we add a link e(u, v) to  $\overline{G}$ , we check whether the value  $c_v$  decreases, and if it does, we update  $c_v$  and set

$$T_{(s,v)}^{\hat{G}}[c_v].w \leftarrow w_e, \quad T_{(s,v)}^{\hat{G}}[c_v].p \leftarrow u, \quad T_{(s,v)}^{\Gamma}[c_v].c \leftarrow c_e.$$

We perform a similar process for each node x for which  $c_x$  decreases as a result of adding link e. Upon completion, for each node v of  $\Gamma$ , the traversal function  $F_{(s,\,v)}^{\hat{G}}(c)$  is stored in the array  $T_{(s,\,v)}^{\hat{G}}[c]$ . The formal specification of Algorithm AGG-AHOP appears in Fig. 8.

Given a QoS constraint  $\hat{w}$  and a destination node t, we determine a suitable path  $\hat{\mathcal{P}}$  through the following procedure. First, we determine the minimum cost  $\hat{c}$  of a (s,t)-path is  $\hat{G}$  that supports  $\hat{w}$  by setting  $\hat{c} = \min\{c|T_{(s,t)}^{\hat{G}}[c].w \leq \hat{w}\}$ . Next, path  $\hat{\mathcal{P}}$  is identified by iteratively discovering the predecessor of each node, beginning with t. The predecessor  $v_{i-1}$  of  $v_i$  is determined by setting  $v_{i-1} = T_{(s,v_i)}^{\hat{G}}[\hat{c} - C(\hat{\mathcal{P}}_{(v_i,t)},\hat{w})].p$ , where  $C(\hat{\mathcal{P}}_{(v_i,t)},\hat{w})$  is the cost of supporting  $\hat{w}$  by the subpath of  $\hat{\mathcal{P}}$  identified so far. The budget  $c_e$  allocated to link  $e(v_{i-1},v_i)$  of  $\hat{\mathcal{P}}$  is set to  $c_e = T_{(s,v_i)}^{\hat{G}}[\hat{c} - C(\hat{\mathcal{P}}_{(v_i,t)},\hat{w})].c$ . We note that the identification of  $\hat{\mathcal{P}}$  requires  $\mathcal{O}(\log H + |\hat{\mathcal{P}}|)$  time.

Theorem 3: Algorithm AGG-AHOP determines, in  $\mathcal{O}(MH\log N)$  time the aggregate traversal function  $F_{(s,\,t)}^{\hat{G}}(c)$  for each node  $t\in\hat{G}$ .

# D. Discussion

We presented an  $\mathcal{O}(MH\log N)$  algorithm for computing traversal functions in an aggregated environment. As previously noted, a straightforward approach would be to substitute each link by  $\mathcal{O}(H)$  links and execute the Bellman–Ford algorithm in the resulting graph. Since the Bellman–Ford algorithm would

```
Algorithm AGG-AHOP (G, s):
         input:
             \hat{G}- aggregated network
             s - source node
          1 \bar{G}(\bar{V}, \bar{E}) \leftarrow \text{INITIALIZE}()
              for all v \in \bar{V} do
                  c_v \leftarrow H + 1
              T_{(s,s)}^{\hat{G}}[0].w \leftarrow 0, T_{(s,s)}^{\hat{G}}[0].p \leftarrow nil
         8
              for each link e(v, u) \in S by increasing order of w_e do
                  if exists a link e' between v and u in \bar{E} then
         10
                      remove e' from \bar{E}
        11
                  add e to \bar{E}
                  PROPAGATE(e(v,u))
        13 return T_{(s,v)}^{\hat{G}} for each node v in \hat{G}
        PROPAGATE(e(u,v))
              if (c_u + c_e) < c_v then
                  for c \leftarrow (c_u + c_e) to (c_v - 1) do
                     T_{(s,v)}^{\hat{G}}[c].w \leftarrow w_e, T_{(s,v)}^{\hat{G}}[c].p \leftarrow u,T_{(s,v)}^{\hat{G}}[c].c \leftarrow c_e
                  for each e'(v, x) \in \bar{E} do
                     PROPAGATE(e'(v, x))
         6
        INITIALIZE()
             \bar{V} \leftarrow V, \bar{E} \leftarrow \emptyset
              for each e = (v, u) \in \hat{G} do
                  for c \leftarrow 1 to H_{\Gamma^e} do
                      add a new link e' between v and u to \bar{E}
                      c_{e'} \leftarrow c, w_{e'} \leftarrow F_{(v,u)}(c)
          5
          6
             return \bar{G}(\bar{V},\bar{E})
```

Fig. 8. Algorithm AGG-AHOP.

then be applied to a graph with  $\mathcal{O}(MH)$  links, its computational complexity would be  $\mathcal{O}(MH^2)$ , which is  $\Omega(H/\log N)$  times higher than that of our scheme. Recall that H is the maximum hop count in the *actual* network, whereas N is the number of nodes in the *aggregated* network, hence, our improvement is significant.

Next, let us compare our precomputation scheme with an alternative where no precomputation is performed at all. In such a single-phase scheme, the required path can be identified by computing, for each link  $e(v,u) \in \hat{G}$ , the cost  $c_e$  of supporting the QoS constraint  $\hat{w}$  (i.e.,  $c_e = \min\{c|F_{(v,u)}(c) \leq \hat{w}\}$ ), and then applying Dijkstra's shortest path algorithm [8] to a graph with link costs  $c_e$ . This scheme incurs  $\mathcal{O}(M\log H + N\log N)$  running time, compared to  $\mathcal{O}(\log H + |\hat{\mathcal{P}}|)$  in our scheme. Hence, our scheme allows to significantly reduce the time required for the identification of a suitable path.

#### V. PRECOMPUTATION SCHEMES FOR ADDITIVE METRICS

In this section, we consider the routing problem with additive QoS constraints and general links costs. We assume a link state environment, i.e., the source node has a full image of the network. We consider general networks, i.e., we do not assume that the network has a specific (e.g., hierarchical) structure. Our purpose is to devise a scheme that (pre)computes, for each cost

 $0 \le c \le C$  and for each destination node  $t \in G$ , a c-cost constrained (s,t)-path of minimum weight, where C is maximal cost of a path that can be considered for routing purposes. Accordingly, we introduce the all-costs optimal path problem (Problem ACOP), which is a generalization of Problem AHOP for general link costs.

Problem ACOP: Given are a graph G=(V,E), a source node  $s \in V$  and a maximum cost C. Find, for each cost c,  $1 \le c \le C$ , and each destination node  $t \in G$ , a c-cost constrained (s,t)-path of minimum weight.

Problem ACOP is computationally intractable since it contains Problem RSP, which is  $\mathcal{NP}$ -hard. Accordingly, we resort to precomputation schemes that offer *approximate* solutions.

Definition 5: Given an instance of Problem ACOP, with source node s, maximum cost C, and approximate ratio  $\varepsilon$ ,  $0 < \varepsilon \le 1$ , an  $\varepsilon$ -approximate solution is a set of paths S, such that, for each  $0 \le c \le C$  and  $t \in G$ , there exists an (s, t)-path  $\hat{\mathcal{P}} \in S$  that satisfies:

- 1)  $W(\hat{\mathcal{P}}) \leq W(\mathcal{P})$ , for any c-cost constrained (s, t)-path  $\mathcal{P}$ ;
- 2)  $C(\hat{\mathcal{P}}) \leq (1+\varepsilon)c$ .

We note that an approximate solution for Problem ACOP can be constructed on the basis of existing approximation algorithms for Problem RSP (e.g., [13], [26]), i.e., by sequentially executing them for various values of the cost constraint. However, as we will see, such a simplistic approach results in a (overly) high computational complexity. Therefore, in this section we propose a scheme that precomputes a set of suitable paths within  $\mathcal{O}((1/\varepsilon)HM\log C)$  computational complexity. Upon a connection request, a suitable path is chosen from a set of precomputed path within  $\mathcal{O}(\log((1/\varepsilon)H\log C))$  time.

This section is organized as follows. First, we present a simple precomputation scheme whose running time is  $\mathcal{O}(MC)$ , which is pseudo-polynomial. Next, by using a *logarithmic scaling* technique, we establish a  $\mathcal{O}((1/\varepsilon)HM\log C)$  precomputation scheme that offers an  $\varepsilon$ -approximate solution for Problem ACOP.

#### A. Pseudo-Polynomial Solution for Problem RSP

As a first step, we present a simple precomputation scheme, whose computational complexity is pseudo-polynomial. The scheme is based on dynamic programming and is an extension of the standard Bellman-Ford's algorithm. For each node  $v \in G$  we maintain array  $T_{(s,v)}[c]$  such that  $T_{(s,v)}[c].w$  keeps the minimum weight of a c-cost constrained (s, v)-path in Gand  $T_{(s,v)}[c].p$  keeps the predecessor of v in that path. The algorithm iterates over "budget" values c = 0, 1, ..., C. At each iteration, the algorithm repeatedly selects a link  $e \in G$ and relaxes it. The process of relaxing a link e(v, u) consists of testing whether the minimum weight of (v, u)-path can be improved by going through v under the current budget restriction c and, if so, updating  $T_{(s,u)}[c]$ . Since for each  $c, 1 \leq c \leq C$ , the algorithm performs  $\mathcal{O}(M)$  operations, its complexity is  $\mathcal{O}(MC)$ . A formal specification of the scheme can be found in [22].

```
Algorithm RSP-GEN (G(V, E), s, \varepsilon, C):
         input:
             G(V, E) - network
             s \in G - source node
             \varepsilon, 0 < \varepsilon \le 1- approximation ratio
             C - the maximum allowed cost of a path
          1 \delta \leftarrow (1 + \frac{\varepsilon}{6H})
             for all v \in V do
                  T_{(s,v)}[0].w \leftarrow \infty
          4 T_{(s,s)}[0].w \leftarrow 0, T_{(s,s)}[0].p \leftarrow nil
         5 i_{\max} \leftarrow \min_{i=1,2,\dots} \{i \mid C \leq \delta^i\}
               for i \leftarrow 1 to i_{\max} do
                  c_i \leftarrow \delta^i
          8
                  for all v \in V do
          9
                      T_{(s,v)}[c_i].w \leftarrow T_{(s,v)}[c_{i-1}].w
                      T_{(s,v)}[c_i].p \leftarrow T_{(s,v)}[c_{i-1}].p
        10
                  for each link e(v, u) \in E do
        11
        12
                      if (c_e \leq c_i) then
                          c \leftarrow \max_{j=1,2,\cdots} \{\delta^j \mid \delta^j \le c_i - c_e\}
        13
                         if T_{(s,v)}[c].w + w_e \le T_{(s,u)}[c_i].w then
        14
        15
                             T_{(s,u)}[c_i].w \leftarrow T_{(s,v)}[c].w + w_e
                             T_{(s,u)}[c_i].p \leftarrow v
        16
```

Fig. 9. Algorithm RSP-GEN.

#### B. Polynomial Precomputation (Approximation) Scheme

We proceed to present an efficient precomputation scheme that provides an  $\varepsilon$ -approximate solution to Problem ACOP. The scheme is based on the pseudo-polynomial solution and uses a logarithmic scaling approach. Specifically, it considers only a limited number of budget values, namely,  $1, c_1, c_2, \ldots, c_{i_{max}}$ , where  $c_i = \delta^i$ ,  $i_{\text{max}} = \min\{i | \delta^i \ge C\}$  and  $\delta = (1 + (\varepsilon/6H))$ . For each node  $v \in G$  we maintain array  $T_{(s,\,v)}[c]$  such that, for  $0 \leq i \leq i_{\max}$ ,  $T_{(s,v)}[c_i].w$  keeps the minimum weight of a  $c_i$ -cost constrained (s, v)-path in G and  $T_{(s, v)}[c_i]$ . p keeps the predecessor of v in that path. The algorithm iterates over "budget" values  $c_i=1,\,c_1,\,c_2,\,\ldots,\,c_{i_{\max}}$ . At each iteration, the algorithm repeatedly selects a link  $e \in G$  and relaxes it. The process of relaxing a link (v, u) consists of testing whether the minimum weight of (v, u)-path can be improved by going through v under the current budget restriction  $c_i$  and, if so, updating  $T_{(s,u)}[c_i]$ . As will be shown below, the set of such paths constitutes a  $\varepsilon$ -optimal solution for Problem ACOP. The formal specification of Algorithm RSP-GEN appears in Fig. 9.

We will demonstrate the precomputation process by using the network G depicted in Fig. 10. Algorithm RSP-GEN is invoked for G, s,  $\varepsilon=1$  and H=5. Thus,  $\delta=1.1$ . We consider a request for a (s,t)-path that satisfies a QoS constraint  $\hat{w}=15$ . For this request,  $\hat{\mathcal{P}}=\{s,v_1,v_2,t\}\in G$  is an optimal path. We show that the algorithm identifies a path whose weight is at most  $\hat{w}$  and whose cost is at most  $(1+\varepsilon)C(\hat{\mathcal{P}})$ . First, after executing line 4, we have  $T_{(s,s)}[0].w=0$ . Next, consider the execution of the main loop, i.e., the loop that begins at line 6, for i=12. Since  $\delta^i=3.138>3$ , the condition of line 12 is satisfied, hence, upon completion of the iteration, it holds that  $T_{(s,v_1)}[3.138].w\leq 2$ . Next, consider the iteration of the main

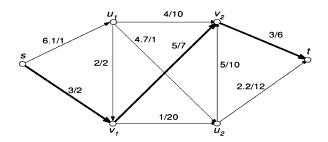


Fig. 10. Execution of Algorithm RSP-GEN. For each link e the upper number shows  $c_e$  and the lower number shows  $w_e$ .

loop for i=23 and the iteration of the subloop at line 11 for  $e=(v_1,v_2)$ . In line 13 we set c=3.797, which is the highest degree of  $\delta$  that is lower than  $c_i-c_e=3.954$ , where  $c_i=\delta^i=8.945$ . In the next lines we check whether  $T_{(s,v_1)}[3.797].w+w_e\leq T_{(s,v_2)}[8.954].w$  and, if so, we assign  $T_{(s,v_2)}[8.954].w=T_{(s,v_1)}[3.945].w+w_e$ . Thus, after completion of the iteration of the main loop for i=23, we have  $T_{(s,v_2)}[8.954].w\leq 9$ . Finally, after the completion of the iteration for i=27 we have  $T_{(s,t)}[c_i].w\leq 15$ , where  $c_i=\delta^i=13.11$ . We conclude that the algorithm identifies a path whose weight is at most  $\hat{w}=15$ , and whose cost is at most  $13.11\leq (1+\varepsilon)C(\hat{\mathcal{P}})=22$ . In fact, Algorithm RSP-GEN applied for  $G,s,\varepsilon=2$  and  $\hat{w}=15$ , yields a path  $\{s,u_1,u_2,t\}$  whose cost is 13, which is 1.18 times more than the optimum (11).

Algorithm RSP-GEN constitutes the first phase of our precomputation scheme, and its output, i.e., the arrays  $T_{(s,\,v)}[c]$ , is used by the second phase. That phase is invoked upon a connection request between s and a destination node  $t\in V$ , with a QoS requirement  $\hat{w}$ .

Upon arrival of a connection request for an (s,t)-path  $\hat{\mathcal{P}}$  with a QoS requirement  $\hat{w}$ , we first find the cost  $\hat{c}$  of  $\hat{\mathcal{P}}$  by setting  $\hat{c} = \min\{c_i|T_{(s,t)}[c_i].w \leq \hat{w}\}$ , where  $c_i = \delta^i$ . This operation is performed through a binary search on  $\mathcal{O}((1/\varepsilon)H\log C)$  values of  $c_i$  and requires  $\mathcal{O}(\log((1/\varepsilon)H\log C))$  time. The running time can be improved by considering only  $\mathcal{O}((1/\varepsilon)\log C)$  values of  $c_i$ , namely,  $c_i = \min\{\delta^j|\delta_1^i \leq \delta^j\}$ , where  $\delta_1 = (1+\varepsilon/3)$ . This improvement yields a running time of  $\mathcal{O}(\log((1/\varepsilon)\log C))$ , and, as we prove in Theorem 4, does not introduce a penalty in terms of approximation's accuracy.

Next, we identify a suitable path  $\hat{\mathcal{P}}=\{s=v_0,\ldots,v_h=t\}$  by using the information stored in the arrays  $\{T_{(s,\,v)}[c]|v\in G\}$ . Specifically, the predecessor  $v_{h-1}$  of t in  $\hat{\mathcal{P}}$  is determined by setting  $v_h=T_{(s,\,t)}[\hat{c}].p$ . Generally, the predecessor  $v_{i-1}$  of  $v_i$  is determined by setting  $v_{i-1}=T_{(s,\,v_i)}[x_i].p$ , where  $x_i=\max_{j=1,\,2,\,\ldots,\,i_{\max}}\{\delta^j|\delta^j\leq\hat{c}-C(\hat{\mathcal{P}}_{(v_i,\,t)})\}$  and  $C(\hat{\mathcal{P}}_{(v_i,\,t)})$  is the cost of the subpath  $\hat{\mathcal{P}}_{(v_i,\,t)}$  of  $\hat{\mathcal{P}}$  discovered so far.

Theorem 4: Algorithm RSP-GEN computes, in  $\mathcal{O}((1/\varepsilon)MH\log C)$  time, an  $\varepsilon$ -approximate solution of Problem ACOP.

#### C. Discussion

We established a precomputation scheme for Problem RSP that provides  $\varepsilon$ -optimal solutions within a computational complexity of  $\mathcal{O}((1/\varepsilon)HM\log C)$  for the first phase and  $\mathcal{O}(\log(1/\varepsilon) + \log\log C)$  for the second phase.

Compared to 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, Problem RSP should be solved for each connection request, through an  $\varepsilon$ -optimal approximation to Problem RSP [26], which incurs a computational complexity of  $\mathcal{O}(MH((1/\varepsilon) + \log\log H))$ . We conclude that the second phase of our scheme allows to identify an  $\varepsilon$ -optimal path upon a connection request  $\Omega(MH/(\varepsilon(\log(1/\varepsilon) + \log\log C)))$  times faster.

As previously noted, a precomputation scheme can be trivially constructed on the basis of existing approximation algorithms for Problem RSP, such as [26], by sequentially executing them for various weight values. In order to perform the precomputation for Problem RSP, this algorithm should be invoked  $\mathcal{O}((1/\varepsilon)\log C)$  times *per destination*, with a total complexity of  $\mathcal{O}((1/\varepsilon^2)NHM\log C)$  for all destinations, which is significantly  $(\Omega(N/\varepsilon)$  times) higher than that of our solution.

#### VI. CONCLUSION

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 database, 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 for QoS routing, the precomputation perspective received little attention. As was demonstrated in this paper, simplistic adaptations of standard algorithms are usually inefficient.

Accordingly, this paper investigated the precomputation perspective, considering two major settings of QoS routing. First, we focused on 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 improvement in terms of computational complexity.

Next, we considered networks with topology aggregation, which is an inevitable tool for providing scalable routing. We indicated that precomputation is an inherent component of QoS routing schemes in aggregated environments. Accordingly, we extended our precomputation scheme for bottleneck QoS requirements in a way that is suitable for topology aggregation. This specific extension indicates how our precomputation techniques can be adapted to aggregated environments in general.

Then we considered the second setting, namely, additive QoS constraints (i.e., delay) and general link costs. As the related routing problem is  $\mathcal{NP}$ -hard, we focused on  $\varepsilon$ -optimal approx-

imations and derived a precomputation scheme that offers a major improvement over the "standard" approach.

Finally, we note that the precomputation concept is applicable to various areas of network control and management, hence, offering a rich ground for future research.

# APPENDIX DETAILED DESCRIPTION OF PROCEDURE PATH

We begin by presenting Procedure GET-PATH, which retrieves the paths (pre)computed by Procedure CLUSTER. Next, we present Procedure PATH, which identifies the required QoS path by concatenating paths returned by Procedure GET-PATH. Formal specifications of the two procedures can be found in [22].

#### A. Procedure GET-PATH

This procedure receives as input a layer-k peer group  $\Gamma$ , a pair of  $\Gamma$ 's border nodes  $b_i, b_j$ , and a hop constraint  $\hat{h}$ . The procedure uses the output of Procedure CLUSTER to identify a  $(b_i, b_j)$ -path  $\hat{\mathcal{P}}$  such that  $|\hat{\mathcal{P}}| \leq \hat{h}$  and  $W(\hat{\mathcal{P}}) = F_{(b_i,b_j)}^{\Gamma}(\hat{h})$ .

If  $\Gamma$  is a layer-1 peer group, then we use the following procedure. We first discover the predecessor  $v_1$  of  $b_j$ , then the predecessor  $v_2$  of  $v_1$ , etc. The predecessor  $v_{l+1}$  of  $v_l$  is determined by setting  $v_{l+1} = T^{\Gamma}_{(b_i,v_l)}[\hat{h}-l].p$ . The procedure returns the path  $\hat{\mathcal{P}} = \{b_i = v_{\hat{h}}, \ldots, v_1, b_j\}$ .

If  $\Gamma$  is a layer-k peer group, then we need a more elaborated procedure, because  $\hat{\mathcal{P}}$  runs through children of  $\Gamma$ . We first identify the children through which the path  $\hat{\mathcal{P}}$  runs. Next, we recursively determine the detailed path through each (layer-(k-1)) child crossed by  $\hat{\mathcal{P}}$ . Specifically, beginning with  $b_j$ , we iteratively discover the predecessor u of each node v, such that  $u \in \overline{G}$ . This is done by setting  $u = T^{\Gamma}_{(b_i,v)}[\hat{h} - C(\hat{\mathcal{P}}_{(v,b_j)})].p$ , where  $\hat{\mathcal{P}}_{(v,b_j)}$  is the subpath of  $\hat{\mathcal{P}}$  identified so far. If u and v are border nodes of some child  $\Gamma'$  of G, then the subpath  $\hat{\mathcal{P}}_{(u,v)}$  of  $\hat{\mathcal{P}}$  is determined by invoking Procedure GET-PATH on  $\Gamma'$ . Otherwise,  $\hat{\mathcal{P}}_{(u,v)}$  comprises of the link  $(u,v) \in \Gamma$ . We continue this process till we reach  $b_i$ .

#### B. Procedure PATH

This procedure identifies, for a peer group  $\Gamma^k$  and a border node  $b_i \in \Gamma^k$ , a  $\hat{w}$ -weight constrained optimal  $(b_i, t)$ -path. If k=1, i.e.,  $\Gamma^1$  is a layer-1 peer group, such a path was identified by the BFS algorithm. For k>1, Procedure FIND yields the minimum-cost  $(b_i, t)$ -path  $\overline{\mathcal{P}}^{\Gamma^k}(b_i, t)$  in the auxiliary graph  $\overline{\Gamma^k}$ . Procedure PATH identifies a path in  $\Gamma^k$  that corresponds to  $\overline{\mathcal{P}}^{\Gamma^k}(b_i, t)$ . For each link  $e \in \overline{\mathcal{P}}^{\Gamma^k}(b_i, t)$ , one of the following cases applies.

- 1) e connects border nodes  $b_{ij}$  and  $b_{i'j'}$  of different children of  $\Gamma^k$ . In this case, link e is substituted by a link  $(b_{ij}, b_{i'j'}) \in \Gamma^k$ .
- 2) e connects border nodes  $b_{ij}$  and  $b_{ij'}$  of the same child  $\Gamma_i$  of  $\Gamma^k$ . In this case, we substitute e with the  $(b_{ij}, b_{ij'})$ -path across  $\Gamma_i$ , which is identified by Procedure GET-PATH.
- 3) e connects a border node  $b_j$  of  $\Gamma^{k-1}$  and the destination t. In this case, Procedure PATH is applied recursively for

 $\Gamma^{k-1}$  and border node  $b_j \in \Gamma^{k-1}$ . The link e is then substituted by a path returned by the recursive invocation of Procedure PATH.

#### REFERENCES

- G. Apostolopoulos, R. Guérin, S. Kamat, A. Orda, T. Przygienda, and D. Williams, "QoS routing mechanisms and OSPF extensions," Internet Engineering Task Force, RFC 2676, Aug. 1999.
- [2] G. Apostolopoulos, R. Guérin, S. Kamat, A. Orda, and S. K. Tripathi, "Intra-domain QoS routing in IP networks: A feasibility and cost/benefit analysis," *IEEE Network—Special Issue on Integrated and Differ*entiated Services for the Internet, vol. 13, pp. 42–54, Sept.–Oct. 1999.
- [3] G. Apostolopoulos, R. Guérin, S. Kamat, and S. Tripathi, "Quality of service based routing: A performance perspective," in *Proc. ACM SIG-COMM*, Vancouver, BC, Canada, Sept. 1998, pp. 17–28.
- [4] B. Awerbuch and Y. Shavitt, "Topology aggregation for directed graphs," *IEEE/ACM Trans. Networking*, vol. 9, pp. 82–90, Feb. 2001.
- [5] D. Bauer, J. N. Daigle, I. Iliadis, and P. Scotton, "Efficient frontier formulation for additive and restrictive metrics in hierarchical routing," in *Proc. IEEE ICC*, New Orleans, LA, June 2000.
- [6] A. Bestavros and I. Matta, "Load profiling for efficient route selection in multi-class networks," in *Proc. IEEE Int. Conf. Network Protocols*, Atlanta, GA, Oct. 1997.
- [7] J.-Y. Le Boudec and T. Przygienda, "A route pre-computation algorithm for integrated services networks," *J. Network Syst. Manage.*, vol. 3, no. 4, pp. 427–449, Dec. 1995.
- [8] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1990.
- [9] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick, "A framework for QoS-based routing in the Internet," Internet Engineering Task Force, RFC 2386, Aug. 1998.
- [10] M. R. Garey and D. S. Johnson, Computers and Intractability. San Francisco, CA: Freeman, 1979.
- [11] R. Guérin and A. Orda, "QoS-based routing in networks with inaccurate state and metrics information: Theory and algorithms," *IEEE/ACM Trans. Networking*, vol. 7, pp. 350–364, June 1999.
- [12] —, "Computing shortest paths for any number of hops," *IEEE/ACM Trans. Networking*, vol. 10, pp. 613–620, Oct. 2002.
- [13] R. Hassin, "Approximation schemes for the restricted shortest path problem," *Math. Oper. Res.*, vol. 17, no. 1, pp. 36–42, Feb. 1992.
- [14] T. Korkmas and M. Krunz, "Source-oriented topology aggregation with multiple QoS parameters in hierarchical networks," ACM Trans. Modeling Comput. Simulation, vol. 10, no. 4, pp. 295–325, Oct. 2000.
- [15] W. C. Lee, "Topology aggregation for hierarchical routing in ATM networks," in *Proc. ACM SIGCOMM*, MA, Apr. 1995.
- [16] D. H. Lorenz and A. Orda, "QoS routing in networks with uncertain parameters," *IEEE/ACM Trans. Networking*, vol. 6, pp. 768–778, Dec. 1998
- [17] —, "Optimal partition of QoS requirements on unicast paths and multicast trees," *IEEE/ACM Trans. Networking*, vol. 10, pp. 102–114, Feb. 2002.
- [18] K. S. Lui and K. Nahrstedt, "Topology aggregation and routing in bandwidth-delay sensitive networks," in *Proc. IEEE Globecom*, San Francisco, CA, Nov.–Dec. 2000, pp. 410–414.
- [19] Q. Ma and P. Steenkiste, "Quality of service routing for traffic with performance guarantees," in *Proc. IFIP IWQoS*, New York, NY, May 1997, pp. 115–126.
- [20] J. Moy, "OSPF Version 2," Internet Engineering Task Force, RFC 2328, Apr. 1998.
- [21] A. Orda, "Routing with end to end QoS guarantees in broadband networks," *IEEE/ACM Trans. Networking*, vol. 7, pp. 365–374, June 1999.
- [22] A. Orda and A. Sprintson. (2003, Mar.) Precomputation schemes for QoS routing. CCIT Rep. 417, Dept. Electr. Eng., Technion, Haifa, Israel. [Online]. Available: ftp://ftp.technion.ac.il/pub/supported/ee/Network/os03.pdf
- [23] D. Peleg and A. A. Schaffer, "Graph spanners," J. Graph Theory, vol. 13, no. 1, pp. 99–116, 1989.
- [24] C. Pornavalai, G. Chakraborty, and N. Shiratori, "QoS based routing algorithm in integrated services packet networks," in *Proc. IEEE Int. Conf. Network Protocols*, Atlanta, GA, Oct. 1997, p. 167.
- [25] "Private Network-Network Interface Specification v1.0 (PNNI)," ATM Forum, Mar. 1996.
- [26] D. Raz and D. H. Lorenz, "A simple efficient approximation scheme for the restricted shortest path problem," *Oper. Res. Lett.*, vol. 28, no. 5, pp. 213–219, June 2001.

- [27] A. Shaikh, J. Rexford, and K. Shin, "Efficient precomputation of quality-of-service routes," in *Proc. Workshop Network and Operating Systems Support for Audio and Video (NOSSDAV)*, Cambridge, U.K., 1998, pp. 15–27.
- [28] Z. Wang and J. Crowcroft, "Quality-of-service routing for supporting multimedia applications," *IEEE J. Select. Areas Commun.*, vol. 14, pp. 1228–1234, Sept. 1996.



Ariel Orda (S'84–M'86–SM'97) received the B.Sc. (summa cum laude), M.Sc., and D.Sc. degrees in electrical engineering from the Technion—Israel Institute of Technology, Haifa, Israel, in 1983, 1985, and 1991, respectively.

Since 1994, he has been with the Department of Electrical Engineering, Technion, where he is currently an Associate Professor and the Academic Head of the Computer Networking Laboratory. He has held visiting and research positions at the Center for Telecommunication Research, Columbia

University, New York, NY, Bell Laboratories, Murray Hill, NJ, and IBM T. J. Watson Research Center, Yorktown Heights, NY. He has also held several consulting positions with Israeli industry. His current research interests include network routing, QoS provisioning, the application of game theory to computer networking, network pricing, and distributed network algorithms.

Dr. Orda received the Award of the Chief Scientist in the Ministry of Communication in Israel, the Gutwirth Award for Outstanding Distinction, the Research Award of the Association of Computer and Electronic Industries in Israel, and the Jacknow Award for Excellence in Teaching. He served as Technical Program Co-Chair of the IEEE INFOCOM 2002. He is an editor for the IEEE/ACM TRANSACTIONS ON NETWORKING.



Alexander Sprintson (S'00) received the B.Sc. degree (summa cum laude) in electrical and computer engineering from the Technion—Israel Institute of Technology, Haifa, Israel. Since 2000, he has been working toward the Ph.D. degree in the Department of Electrical Engineering at the Technion.

During the summers of 2001 and 2002, he was with the Internet Management Research Department, Bell Laboratories, Murray Hill, NJ. His research interests are in designing scalable network algorithms. He has worked mainly on QoS routing and

restoration schemes for unicast and multicast.

Mr. Sprintson received the Wolf Award for Distinguished Ph.D. Students, the Gutwirth Award for Outstanding Distinction, and the Knesset (Israeli Parliament) Award for Distinguished Students.