CCIT Report #434

Efficient Algorithms for Computing Disjoint QoS Paths

June 2003

Ariel Orda* and Alexander Sprintson†[1]
*Department of Electrical Engineering, Technion – Israel Institute of Technology, Haifa, Israel 32000
Email: ariel@ee.technion.ac.il
†Parallel and Distributed Computing Group, California Institute of Technology, Pasadena, California, USA 91125
Email: spalex@caltech.edu

Abstract

Networks are expected to meet a growing volume of requirements imposed by new applications such as multimedia streaming and video conferencing. Two essential requirements are support of Quality of Service (QoS) and resilience to failures.

In order to satisfy these requirements, a common approach is to use two disjoint paths between the source and the destination nodes, the first serving as a *primary* path and the second as a *restoration* path. Such approach, referred to as *path restoration*, has several advantages, the major one being the ability to switch promptly from one path to another in the event of a failure. A major issue in this context is how to identify two paths that satisfy the QoS constraints imposed by network applications. Since network resources, e.g., bandwidth, are allocated along both primary and restoration paths, we need to consider also the overall network performance. Accordingly, in this paper we study the fundamental problem of finding two disjoint paths that satisfy the QoS constraints at minimum cost. We present approximation algorithms with provable performance guarantees for this fundamental network problem.

Index Terms: Routing, Restoration, Disjoint Paths, Quality of Service.

I. Introduction

Networks are expected to meet *Quality of Service* (QoS) requirements imposed by new applications, such as multimedia streaming and video conferencing. This is facilitated by current efforts to provide resource reservations and explicit path routing, e.g., *MultiProtocol Label Switching* (MPLS). On the other hand, physical network infrastructures may be prone to failures. Therefore, a major challenge in this context is to develop adequate network mechanisms for establishing connections that satisfy QoS requirements and are also resilient to failures. It has been recognized that, for many practical settings, the speed and capacity of links do not allow to provision restoration paths *after* the failure. Thus, the restoration paths must be provisioned in advance, i.e., *before* a failure occurs.

This goal can be achieved by provisioning two disjoint QoS paths between the source and destination nodes. This approach is widely used because of its ability to switch promptly from one path to another in the event of a failure. The disjoint path strategy has many additional advantages. First, it allows to use various protection schemes, such as 1+1 protection or 1:1 protection [1]. With 1+1 protection, traffic is simultaneously transmitted on both paths, which allows instantaneous recovery from link failures. Alternatively, with 1:1 protection, traffic is transmitted along a *primary* path, and, upon a failure of one of its links, the traffic is switched to a *restoration* path. Second, the disjoint path strategy requires minimal network support, because failure detection and restoration can be implemented at the application level of the source. Finally, the disjoint path strategy provides a greater flexibility to application designers, as they can choose a protection scheme (e.g., 1+1 or 1:1) that is most adequate for each particular application.

To facilitate seamless recovery to a restoration path in the event of a failure, it is necessary to reserve network resources (e.g., bandwidth) on both the primary and restoration paths. Such resources should be consumed in a networkwide efficient manner. A common way for modelling the impact of such resource consumption on each link is by associating "costs" with the links. Accordingly, a major problem is to find two disjoint paths between source and destination nodes that satisfy end-to-end QoS constraints at minimum cost. This problem is the subject of this study.

QoS constraints occur naturally in a number of practical settings, involving bandwidth and delay-sensitive applications, such as voice over IP, audio and video conferencing, multimedia streaming, etc. QoS constraints can be divided into *bottleneck* constraints, such as bandwidth, and *additive* constraints, such as delay or jitter. Bottleneck QoS constraints can be efficiently handled by pruning links that do not satisfy them. The problem is then effectively reduced to finding two disjoint paths of minimum cost; this problem was extensively investigated in the literature [2]. Accordingly, in this study we focus on additive QoS constraints, which are more difficult to handle.

QoS routing has been the subject of several recent studies and proposals (see [3], [4] for comprehensive surveys). However, the problem of finding two disjoint QoS paths got little attention. Similarly, path restoration and routing over alternate paths has also attracted a large body of research (see, e.g., [5], [6]). Most of the proposed solutions, however, considered only bottleneck

[1]Part of this work was done while A. Sprintson was with the Department of Electrical Enginnering, Technion.

QoS constraints. The few studies that did consider additive constraints (e.g., [7]), focused on heuristic approaches and did not provide proven performance guarantees.

To the best of our knowledge, this is the first study to provide a solution with provable performance guarantees for this fundamental network problem. The problem is clearly $\mathcal{NP}$-hard since even the basic problem of finding a single optimal path that satisfies an additive QoS constraint is intractable [8]. Furthermore, it turned out that a special case of our problem with no cost minimization, i.e., finding two disjoint paths that (both) satisfy an additive QoS constraint, is $\mathcal{NP}$-hard. Thus, any practical scheme is necessarily sub-optimal and incurs some violation of the QoS constraint. In this paper we present solutions that incur a small violation of the QoS constraint and whose cost guaranteed to be within a certain factor away from the optimum.

Our paper makes the following contributions. First, we introduce the *minimum constrained flow (MCF)* problem, which is an elaborated variant of the minimum-cost flow problem, and relate it to our problem. Specifically, we show that, by solving the MCF problem, we can obtain a solution to our disjoint-paths problem with the smallest possible violation of the QoS constraint. Second, this relation between our problem and network flow problems allows to employ methods and techniques from the theory of network flows, such as *path augmentation* and *cycle-cancellation*. While the path augmentation method is widely used for finding disjoint paths, our study is the first to enhance it with the *cycle-cancellation* method, thus improving the performance of our disjoint QoS paths algorithms. Third, we investigate the fundamental trade-offs between the cost of the identified solution, the violation of the QoS constraint and the computational complexity of the algorithm. Finally, we present a family of algorithms that allows to find a solution that is adequate for any particular setting, such as a solution with minimum violation of the QoS constraint, minimum cost, etc.

Due to the fundamental nature of the considered problem, our results can be used in a variety of practical applications. For example, in MPLS networks, there is a requirement to protect *Label Switched Paths (LSP)* [9]. Accordingly, our methods can be used for identification of disjoint LSPs that satisfy QoS constraints. In ATM networks, alternate paths can be used upon a *crankback* [10]. In the Differential Services framework [11], a *bandwidth broker* is responsible for establishing suitable paths that satisfy service level agreements (SLA). Here too, it is desirable to compute several disjoint QoS paths in order to facilitate failure resilience and prevent congestion.

We note that disjoint paths can be used for other purposes, beyond path protection. First, sending data on disjoint paths improves network utilization and reduces congestion. In fact, sending data on diverse paths is a major tool of traffic engineering. Second, routers may use a precomputation approach in order to improve response time [12]. The key idea is to compute several QoS paths in advance and store them in a database. Upon arrival of a connection request, a suitable path is selected through a simple, fast procedure. Since network topology can change, precomputing disjoint paths increases the probability that at least one path is valid. Finally, disjoint QoS paths can be used in the context of *multipath routing*. With multipath routing, traffic is sent along multiple paths in order to increase bandwidth and the probability of delivery. Multipath routing can be useful in wired [13] and wireless (Ad hoc) Networks [14].

The remainder of the paper is organized as follows. In Section II, we present the network model and formulate the problems considered in this paper. In Section III, we present basic concepts of network flows and establish a relation between the considered problem and network flow problems. In Section IV, we present a simple approximation algorithm for our problem. In Section V, we present a cycle-cancellation approach and show how to use it in order to minimize delay violation. In Sections VI and VII, we show how to improve the computational complexity of that scheme. In Section VIII, we establish a lower bound for the problem. Finally, conclusions are presented in Section IX.

II. Model and Problem Formulation

In this section, we describe the network model and the main problem addressed in this paper. For simplicity of exposition, we use the term *delay requirements* in order to generically refer to *additive QoS constraints*.

A. The Network Model

We represent the *network* 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 $P = (s = v_0, v_1, \cdots, t = v_n)$, such that, for $0 \leq i \leq n-1$, $(v_i, v_{i+1}) \in E$. Here, $n = |P|$ is the *hop count* of P. The subpath of P that extends from v_i to v_j is denoted by $P_{(v_i, v_j)}$. A *cycle* is a path whose source and destination nodes are identical.

Each link $l \in E$ offers a delay guarantee d_l. The delay $D(P)$ of a path P is the sum of the delays of its links, i.e., $D(P) = \sum_{l \in P} d_l$. In order to satisfy QoS constraints, certain resources such as bandwidth and buffer space must be reserved along QoS paths. In order to optimize the global resource utilization, we need to identify QoS paths that consume as few network resources as possible. Accordingly, we associate with each link l a nonnegative cost c_l, which estimates the quality of the link in terms of resource utilization. The link cost may depend on various factors, e.g., the link's available bandwidth and its location. The cost $C(P)$ of a path P is defined to be the sum of the costs of its links, i.e., $C(P) = \sum_{l \in P} c_l$. We shall assume that all parameters (both delay guaranties and costs) are positive integers.

Alg.	Approx. Ratio	Complexity
2DP-1	$(1.5, 1.5(1+\varepsilon))$	$\mathcal{O}(MN(\frac{1}{\varepsilon} + \log\log N))$
2DP-2	$(1+\frac{1}{k}, k(1+\gamma))$	$\mathcal{O}(MN \cdot OPT \log k \cdot \log(CD))$
2DP-3	$(1+\frac{1}{k}, k(1+\gamma)(1+\varepsilon))$	$\mathcal{O}(\frac{MN^3}{\varepsilon}\log(CD))$
2DP-4	$(1+\frac{1}{k}, k(1+\gamma)(1+\varepsilon))$	$\mathcal{O}(\frac{MN^2k^2\log k}{\varepsilon}\log(CD))$

TABLE I

PERFORMANCE CHARACTERISTICS OF PRESENTED ALGORITHMS.

B. QoS paths

A fundamental problem in QoS routing is to identify a minimum cost path between a source s and a destination t that satisfies some delay and bandwidth constraints. Bottleneck QoS constraints, such as bandwidth, can be efficiently handled by simply pruning links that do not satisfy the QoS constraint. Thus, in the rest of the paper, we only consider delay (i.e., additive) constraints. Accordingly, the fundamental problem is to find a minimum cost path that satisfies a given delay constraint. This can be formulated as a *Restricted Shortest Path* problem.

Problem RSP (Restricted Shortest Path) : Given a source node s, a destination node t and a delay constraint D, find an (s,t)-path P such that

1) $D(P) \leq D$, and
2) $C(P) \leq C(\hat{P})$ for any other (s,t)-path $\hat{P}$ that satisfies $D(\hat{P}) \leq D$.

In general, Problem RSP is intractable, i.e., $\mathcal{NP}$-hard [8]. However, there exist pseudo-polynomial solutions, which give rise to fully polynomial approximation schemes[2] (FPAS), whose computational complexity is reasonable (see [15]–[17]). The most efficient scheme, presented in [17], has a computational complexity of $O(MN(\frac{1}{\varepsilon} + \log\log N))$, and computes a path with delay of at most D and cost of at most $(1+\varepsilon)$ times the optimum. We shall refer to that scheme as Algorithm RSP.

C. Problem Statement

We are now ready to formulate the problem that we consider in this study. The first problem seeks to identify two disjoint QoS paths of minimum total cost.

Problem 2DP (2-Restricted Link Disjoint Paths) : Given a source node s, a destination node t and a QoS requirement D, find two link-disjoint (s,t)-paths P_1 and P_2 such that:

1) $D(P_1) \leq D$ and $D(P_2) \leq D$;
2) $C(P_1) + C(P_2) \leq C(\hat{P}_1) + C(\hat{P}_2)$ for every other pair of link-disjoint (s,t)-paths $\hat{P}_1$ and $\hat{P}_2$ that satisfy $D(\hat{P}_1) \leq D$ and $D(\hat{P}_2) \leq D$.

We denote by OPT the cost of an optimal solution to Problem 2DP for (G, s, t, D).

Problem 2DP includes Problem RSP as a special case; hence, it is $\mathcal{NP}$-hard. In addition, as discussed below (in Section VIII), it is intractable to find a solution that does not violate the delay constraint of at least one of the paths. Furthermore, in most cases, we cannot provide an efficient solution without violating the delay constraint in both primary and restoration paths. Accordingly, we introduce the following definition of (α, β)*-approximations*.

Definition 1 ((α,β)-approximation) : Given an instance (G, s, t, D) of Problem 2DP, an (α,β)-approximate solution (P_1, P_2) to Problem 2DP is a solution for which:

1) $D(P_1) + D(P_2) \leq 2\alpha D$;
2) the total cost of two paths is at most β times more than that of the optimal solution, i.e., $C(P_1) + C(P_2) \leq \beta OPT$.

In general, the path with minimum delay among P_1 and P_2 serves as a primary path. Thus, the primary and restoration paths violate the delay constraint by factors of at most $\alpha/2$ and α, respectively, i.e., $D(P_1) \leq \alpha D$ and $D(P_2) \leq 2\alpha D$.

D. Our results

We introduce four approximation algorithms for Problem 2DP. Table I shows the approximation ratio achieved by each algorithm and its complexity.

The parameters ε and k capture the trade-off between the violation of the delay constraint, the cost of the approximation and the computational complexity of the algorithms. For example, Algorithm 2DP-2 achieves an approximation ratio of $(1+\frac{1}{k}, k(1+\gamma))$ for a positive integer k, where γ is a small value bounded by $\frac{2(\log k+1)}{k}$. Thus, choosing $k = 4$ yields a $(2.25, 5.5)$-approximation solution to Problem 2DP. In general, smaller values of k yield solutions with lower delay violation at the expense of higher costs and running times.

[2]A *Fully Polynomial Approximation scheme* (FPAS) provides a solution whose cost is at most $(1+\varepsilon)$ times more than the optimum with a time complexity that is polynomial in the size of the input and $1/\varepsilon$.

Algorithm 2DP-4 is the main contribution of this paper. The algorithm achieves, for fixed $\varepsilon > 0$ and any integer $k > 0$, the approximation ratio of $(1 + \frac{1}{k}, k(1 + \gamma)(1 + \varepsilon))$, i.e, the primary and restoration paths violate the delay constraint by a factor of $(1 + \frac{1}{k})$ and $2(1 + \frac{1}{k})$, respectively. Thus, the violation of delay by the primary path can be minimized by choosing sufficiently large values of k.

III. Preliminaries: Network Flows

In order to establish an efficient solution to Problem 2DP, we employ ideas and techniques from the theory of network flows. The solution to our problem, that is two disjoint paths, can be conveniently represented as a *flow*. Accordingly, in this section we briefly present the concept of network flow. A comprehensive survey on the theory of network flows can be found in [18].

We consider flow networks in which each link is associated with a *nonnegative* capacity. We assume that for any pair of nodes u and v, the flow network does not contain two links in opposite directions ((v, u) and (u, v)). We note that this assumption does not impose any loss of generality, because by a suitable transformation we can always define a network that is equivalent to any given network but satisfies the above assumption: the transformation splits each node v into two nodes v' and v'' corresponding to node output and input links, and replaces each original link (v, u) by a link (v', u'') with the same capacity, cost and delay; it also adds a link (v'', v') of zero cost and delay and infinite capacity to each node v.

We proceed to introduce the fundamental concept of *network flows*. We restrict ourselves to *unary flows*, i.e., flows that take the value of 0 or 1 in each of the links.

Definition 2 (Unary Flow): A *unary* (s,t)*-flow* f is a binary function $f : E \to \{0, 1\}$ that satisfies the following two properties:

1) For all $l = (u, v) \in E$, it holds that $f_l \in \{0, 1\}$;
2) For all $v \in V \setminus \{s, t\}$, it holds that

$$\sum_{w:(w,v)\in E} f_{(w,v)} = \sum_{w:(v,w)\in E} f_{(v,w)}.$$

For clarity, we say that each link $l \in G$ for which $f_l = 1$ *belongs* to the flow f and that f *includes* all links for which $f_l = 1$.

Definition 2 uses the *link* representation, i.e., the flow is described by means of a function associated with each link of the network. Alternatively, a unary flow can also be represented by a set of paths $\mathbf{P} = \{P_1, \cdots, P_k\}$ and cycles $\mathbf{W} = \{W_1, \cdots, W_x\}$, such that exactly one unit of flow is sent along each path and cycle. We refer to this representation as a *path and cycle* representation. Note that given a *path and cycle* representation of a flow f, it is easy to determine the link representation: the flow f_l on each link l that belongs to a path in $\mathbf{P}$ or to a cycle in $\mathbf{W}$ is 1, while the flow on any other link is 0. Similarly, given a link representation f of a flow, we can determine its path and cycle representation by using the flow decomposition algorithm [18].

The *value* of a flow f is defined as follows:

$$|f| = \sum_{v:(s,v)\in E} f_{(s,v)} \tag{1}$$

A flow of zero value contains only cycles and no paths. Such a flow is referred to as a *circulation*.

The cost $C(f)$ of a flow f is defined as follows:

$$C(f) = \sum_{(u,v)\in E} c_{(u,v)} \cdot f_{(u,v)} \tag{2}$$

We introduce the notion of the *delay* $D(f)$ of flow f.

$$D(f) = \sum_{(u,v)\in E} d_{(u,v)} \cdot f_{(u,v)} \tag{3}$$

Note that a flow f with $|f| = 2$ can be decomposed into two disjoint paths whose total delay and cost is at most $D(f)$ and $C(f)$, respectively. Thus, our goal is to find a flow whose delay and cost are no more than $2\alpha D$ and βOPT, respectively.

A. Minimum Constrained Flow Problem

We proceed to introduce the *minimum constrained flow* (MCF) problem. The problem seeks a minimum cost (s,t)-flow f such that $|f| = 2$ and $D(f) \leq D$, where D is a given delay constraint.

Problem MCF (Minimum Constrained Flow Problem): Given a graph G, a source node s, a destination node t and a delay requirement D, find an (s,t)-flow f such that:

1) $|f| = 2$;
2) $D(f) \leq 2D$;
3) $C(f) \leq C(\hat{f})$ for any other flow $\hat{f}$ that satisfies $|\hat{f}| = 2$ and $D(\hat{f}) \leq 2D$.

The cost of an optimal solution to Problem MCF for (G, s, t, D) is denoted by OPT_1. Note that Problem MCF is a relaxation of Problem 2DP. In particular, instead of imposing a delay constraint for each of the two paths, Problem MCF requires that the total delay of two paths be no more than $2D$. Thus, if (P_1, P_2) is a feasible solution to Problem 2DP, then the flow $f = \{P_1, P_2\}$ is a feasible solution to Problem MCF. We conclude that the cost of the optimal solution to Problem MCF is lower than that of Problem 2DP, i.e., $OPT_1 \leq OPT$.

IV. Simple Approximation Algorithm

In this section we present our first approximation algorithm, which achieves an approximation ratio of $(1.5, 1.5(1+\varepsilon))$. The computational complexity of the algorithm is $\mathcal{O}(MN(\frac{1}{\varepsilon} + \log\log N))$, which is identical to that of the approximation scheme for Problem RSP [17].

The idea of the algorithm is to identify a suitable flow f between s and t such that $|f| = 2$ and then decompose it into two disjoint paths $\hat{P}_1$ and $\hat{P}_2$. The algorithm employs the *path augmentation approach* [18], which is a standard approach for network flow and disjoint path problems.

The first step of the algorithm is to compute a path P_1 between the source node and destination nodes s and t that satisfies the delay constraint D. The path P_1 is constructed by applying Algorithm RSP for $(G, s, t, D, \varepsilon)$. This path defines a flow $f = \{P_1\}$ whose value is one unit.

The next step is to augment this flow in order to increase its value to 2. To that end, we construct a *residual network* $G(f)$ imposed by the flow f. Intuitively, the residual network consists of links that can admit more flow.

Definition 3 (Residual Network): Given a network G with unit capacities and flow f, the *residual network* $G(f)$ is constructed as follows. For each link $(u, v) \in G$ for which $f_{(u,v)} = 0$, we add to $G(f)$ a link (u, v) of the same delay and cost as in G. For each link $(u, v) \in G$ for which $f_{(u,v)} = 1$, we add to $G(f)$ a *reverse* link (v, u) to $G(f)$ of zero cost and zero delay.

A flow $\hat{f}$ in the residual network $G(f)$ is referred to as an *augmenting flow*. Having identified the flow $\hat{f}$, we can *augment* the flow f along the flow $\hat{f}$ by performing the following steps:

1) Omit from f each link (v, u) whose reverse link (u, v) appears in $\hat{f}$.
2) Add to f each link $(v, u) \in \hat{f}$ whose reverse link (u, v) does not appear in f.

With the augmentation paths approach, the flow f is augmented along flows that consist of a single *augmenting path*. In particular, our algorithm identifies an augmenting path P_2 in $G(f)$ that satisfies a delay constraint of $2D$. To that end, we apply Algorithm RSP for $(G(f), s, t, 2D, \varepsilon)$. Then, we augment the flow f along the path P_2. For each link l that belongs to P_2 if $f_l = 0$ we set $f_l = 1$, otherwise we set $f_l = 0$. The value of the resulted flow f is 2.

The final step is to decompose the flow f into two paths $\hat{P}_1$ and $\hat{P}_2$, such that $D(\hat{P}_1) \leq D(\hat{P}_2)$. For this purpose we employ the following flow decomposition algorithm. We start at the source node s and select a link (s, v) for which $f_{(s,v)} = 1$. If v is a destination node, we stop; otherwise, there must be a link (v, u) for which $f_{(v,u)} = 1$. This process is repeated until we either encounter a destination node t or revisit a previously examined node. In the former case we obtain an (s, t)-path P and in the latter case we obtain a cycle W. If we obtain a directed path P, we redefine $f_l = 0$ for each link l in P. Similarly, if we obtain a cycle W, we redefine $f_l = 0$ for each link l in W. We repeat this process till we discover two paths between s and t.

The detailed description of the approximation algorithm, referred to as 2DP-1, appears in Fig. 1.

The correctness of our algorithm is based on the following lemma.

Lemma 1: Let $G(f)$ be the residual network of G imposed by $f = \{P_1\}$. Then, there exists a path $P'_2 \in G(f)$ such that $D(P'_2) \leq 2D$ and $C(P'_2) \leq OPT$.

Proof: Let $\hat{G} \subseteq G$ be a network imposed by links that belong to P_1^{opt}, P_2^{opt} and P_1. Let $\hat{G}(f)$ be the residual network of $\hat{G}$ imposed by the flow $f = \{P_1\}$. Clearly, $\hat{G}(f) \subseteq G(f)$. We prove that there exists a path $P'_2 \in \hat{G}(f)$ that satisfies the conditions of the lemma.

By way of contradiction, assume that such a path does not exist. Then, by the *Path Augmentation Theorem* [18], the flow $f = \{P_1\}$ is a maximum flow in $\hat{G}$. However, there exists a flow $f' = \{P_1^{opt}, P_2^{opt}\}$ of higher value, resulting in a contradiction.

Note that path P'_2 includes only links that belong to P_1^{opt}, P_2^{opt} as well as links originated from P_1, whose delay and cost are zero. As a result, $D(P'_2) \leq D(P_1^{opt}) + D(P_2^{opt}) \leq 2D$ and $C(P'_2) \leq C(P_1^{opt}) + C(P_2^{opt}) \leq OPT$.

Since $\hat{G}(f) \subseteq G(f)$, it holds that $P'_2 \in G(f)$ and the lemma follows. ■

Let (G, s, t, D) be an instance of Problem 2DP and let (P_1^{opt}, P_2^{opt}) be an optimal solution for this instance, i.e., $D(P_1^{opt}) \leq D$, $D(P_2^{opt}) \leq D$ and $C(P_1^{opt}) + C(P_2^{opt}) = OPT$ (see Fig. 2(a)). Note that the path P_1 computed in Step 1 might share links with the optimal paths (see Fig. 2(b)). We note also that

$$C(P_1) \leq (1+\varepsilon)\min\{C(P_1^{opt}), C(P_2^{opt})\} \leq$$
$$\leq (1+\varepsilon)\frac{OPT}{2}.$$

```
Algorithm 2DP-1 $(G, s, t, D, \varepsilon)$
  input:
      $G$ - the graph
      $s$- source node
      $t$ -destination node
      $D$- the delay constraint
      $\varepsilon$- the approximation ratio
  output:
      $(\hat{P}_1, \hat{P}_2)$- An approximate solution to Problem 2DP.

   1  Identify path $P_1$ in $G$ such that $D(P_1) \leq D$ by using
      Algorithm RSP
   2  $f \leftarrow \{P_1\}$
   3  Construct the residual network $G(f)$ of $G$ imposed by $f$:
   4        Add to $G(f)$ each link in $G$ that does not belong to $P_1$
   5        for each link $(u, v) \in P_1$ do
   6              Add a link $(v, u)$ to $G(f)$ with
                  $d_{(v,u)} = 0$ and $c_{(v,u)} = 0$
   7  Identify path $P_2$ in $G(f)$ such that $D(P_2) \leq 2D$
      by using Algorithm RSP
   8  Augment flow $f$ along path $P_2$:
   9        for each link $l(u, v) \in P_2$ do
  10              if $f_{(v,u)} = 0$ then
  11                    $f_{(v,u)} \leftarrow 1$
  12              else
  13                    $f_{(v,u)} \leftarrow 0$
  14  Decompose flow $f$ into two paths, $\hat{P}_1$ and $\hat{P}_2$
      such that $D(\hat{P}_1) \leq D(\hat{P}_2)$
  15  return $\hat{P}_1$ and $\hat{P}_2$
```

Fig. 1. Algorithm 2DP-1

Fig. 2(c) depicts the residual graph $G(f)$ imposed by the flow $f = \{P_1\}$. Each residual link $l \in G(f)$ is assigned zero delay. By Lemma 1, there exists a path $P_2' \in G(f)$ between s and t whose delay is at most $2D$ and whose cost is at most OPT (see Fig. 2(d)). Thus, Algorithm RSP, invoked for $(G(f), s, t, 2D)$, returns a path P_2, whose cost is at most $(1 + \varepsilon) \cdot OPT$ (see Fig. 2(e)). We conclude that:

$$D(P_1) + D(P_2) \leq 3D$$

and

$$C(P_1) + C(P_2) \leq 1.5(1 + \varepsilon)OPT.$$

Paths $\hat{P}_1$ and $\hat{P}_2$ include links that belong to P_1 and P_2, excluding links that were assigned zero cost and delay (see Fig. 2(f)). Hence,

$$D(\hat{P}_1) + D(\hat{P}_2) \leq D(P_1) + D(P_2) \leq 3D$$

and

$$C(\hat{P}_1) + C(\hat{P}_2) \leq C(P_1) + C(P_2) \leq 1.5(1 + \varepsilon)OPT.$$

Choosing the minimum delay path among $\hat{P}_1$ and $\hat{P}_2$ as a primary path, results in a $(1.5, 1.5(1+\varepsilon))$ approximation algorithm for Problem 2DP. The algorithm invokes Algorithm RSP twice, hence its computational complexity is $\mathcal{O}(MN(\frac{1}{\varepsilon} + \log\log N))$.

We summarize our discussion by the following theorem.

Theorem 1: Algorithm 2DP-1 computes, in $\mathcal{O}(MN(\frac{1}{\varepsilon} + \log\log N))$ time, a $(1.5, 1.5(1 + \varepsilon))$-approximate solution for Problem 2DP.

V. Minimizing the Delay Violation

In the previous section we presented Algorithm 2DP-1 that provides a $(1.5, 1.5(1+\varepsilon))$ approximate solution for Problem 2DP. Fig. 3(a) demonstrates an instance (G, s, t, D) of Problem 2DP, for which algorithm 2DP-1 has the worst-case delay violation, i.e., $\alpha = 1.5$. For $D = 2$, the optimal solution is $P_1^{opt} = \{s, v_1, v_2, t\}$ and $P_2^{opt} = \{s, v_3, v_4, t\}$ (see Fig. 3(b)). The cost OPT of the optimal solution is 2. We now apply 2DP-1 to the instance (G, s, t, D). The algorithm selects $P_1 = \{s, v_1, v_2, v_3, v_4, t\}$ because it is the minimum cost path among all paths in G that satisfy the delay constraint 2 (see Fig. 3(c)). Fig. 3(d) depicts the residual network $G(f)$ of G imposed by flow $f = \{P_1\}$. The only path between s and t in $G(f)$ is $P_2 = \{s, v_3, v_2, t\}$, with delay 4 and cost 0. The algorithm returns the paths $\hat{P}_1 = \{s, v_1, v_2, t\}$ and $\hat{P}_2 = \{s, v_3, v_4, t\}$, as depicted in Fig. 3(e).

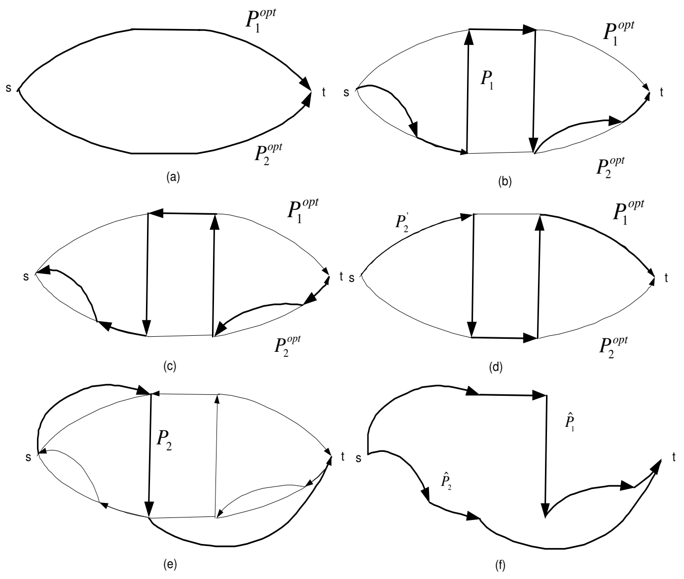

Fig. 2. Execution of Algorithm 2DP-1(a) An optimal solution (P_1^{opt}, P_2^{opt}) to Problem 2DP (b) Path P_1 (c) Residual network $G(f)$ of G imposed by flow $f = \{P_1\}$ (d) Path P_2' (e) Path P_2 (f) Paths $\hat{P}_1$ and $\hat{P}_2$.

Note that $D(\hat{P}_1) = D(\hat{P}_2) = 3$. Thus, we conclude that the path augmentation strategy alone cannot achieve a delay ratio (α) better than 1.5.

The basic idea of the algorithm is to find cycles with negative delays and to augment flow f along these cycles. This allows to reduce the delay of the solution and achieve a smaller delay ratio. For example, Fig. 3(f) shows the residual network $G(f)$ of G imposed by the flow $f = \{\hat{P}_1, \hat{P}_2\}$, constructed from G by substituting each link $l(u, v) \in f$ by a link $l'(v, u)$ and setting $d_{l'} = -d_l$ and $c_{l'} = 0$. The residual network $G(f)$ contains two negative delay cycles: the first cycle is formed by two links between s and v_1, while the second cycle is formed by two links between v_4 and t. Each cycle has delay -1 and cost 1. Thus, if we augment the flow along each of these cycles, the total delay of the flow is improved by -1 at cost 1. By identifying two cycles, we find a flow whose delay and cost are 4 and 2. This flow can be decomposed into two paths whose total delay is at most 4, achieving the approximation ratio of $(2, 1)$.

Algorithm 2DP-2 gets as input the network G, the source and destination nodes s and t, a delay constraint D and approximation parameter k. The algorithm includes the following steps. First, we invoke Algorithm 2DP-1 for $(G, s, t, D, \frac{1}{N})$, which identifies two paths P_1 and P_2. These paths impose a flow $f^0 = \{P_1, P_2\}$. If $D(f^0) \leq 2D(1 + \frac{1}{k})$, then the algorithm halts and returns paths P_1 and P_2. Otherwise, we identify a negative delay cycle in the residual graph $G(f_0)$ of f_0 and augment flow f_0 along this cycle. We repeat this process until the delay $D(f)$ of the resulted flow f is lower or equal to $2D(1 + \frac{1}{k})$. Finally, we decompose the flow f into two disjoint paths $\hat{P}_1$ and $\hat{P}_2$.

More specifically, we introduce the following Procedure IMPROVEFLOW. The procedure gets as input flow f^0 and an approximation parameter k. We begin by setting $f = f^0$ and constructing the residual graph $G(f)$ of G imposed by flow f. The residual graph is constructed according to Definition 3, with the following exception: the delay of a reverse link (v, u) is set to $-d_{(u,v)}$, where (u, v) is the link in the original network that corresponds to (v, u). Next, we find a cycle W in $G(f)$ that minimizes the delay/cost ratio, i.e., the cycle for which $\frac{D(W)}{C(W)}$ is minimal among all cycles in $G(f)$. Such a cycle is determined by using the *minimum cost-to-time ratio cycle algorithm*, presented in [18]. Next, we augment flow f along W. This process (i.e., finding cycles and circulation augmentation) terminates when the delay $D(f)$ of f is lower or equal to $2D(1 + \frac{1}{k})$. Finally, flow f is decomposed into two paths, $\hat{P}_1$ and $\hat{P}_2$, such that $D(\hat{P}_1) \leq D(\hat{P}_2)$. The formal description of Algorithm 2DP-2 appears in Fig. 4.

The following theorem establishes the main properties of Algorithm 2DP-2. Its proof appears in the next section.

Theorem 2: Algorithm 2DP-2 computes, in $\mathcal{O}(MN \cdot OPT \log k \log(CD))$ time, a $(1 + \frac{1}{k}, k(1 + \gamma))$ approximate solution for Problem 2DP, where $\gamma \leq \frac{2(\log k+1)}{k}$.

A. *Analysis of Algorithm* 2DP-2

In this section we analyze the performance of Algorithm 2DP-2 and establish its computational complexity. We use the following variant of the *Augmenting Cycle Theorem*, taken from [18].

Theorem 3: (**Augmenting Cycle Theorem**, [18]) Let f and f^0 be any two feasible network flows such that $|f^0| = |f|$. Then, f equals f^0 plus the circulation $\bar{f}$ in $G(f)$. Furthermore, $D(f) = D(f^0) + D(\bar{f})$.

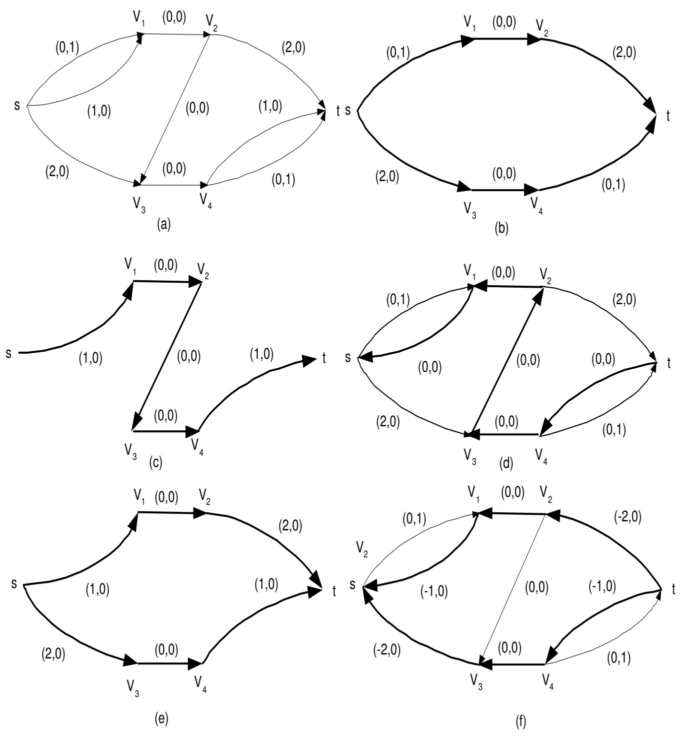
Fig. 3. Execution of Algorithm 2DP-1. Associated with each link are its delay and cost. (a) The original network (b) The optimal solution paths $P_1^{opt} = \{s, v_1, v_2, t\}$ and $P_2^{opt} = \{s, v_3, v_4, t\}$ (c) Path $P_1 = \{s, v_1, v_2, v_3, v_4, t\}$ (d) The residual network induced by flow f (e) Paths $\hat{P}_1$ and $\hat{P}_2$ returned by Algorithm 2DP-1 (f) The residual network $G(f)$ of G imposed by the flow $f = \{\hat{P}_1, \hat{P}_2\}$.

Proof: The proof follows the same lines as in [18], substituting the cost c_l and $C(f)$ by d_l and $D(f)$ for each link l and each flow f, respectively. ■

Recall that OPT_1 is the cost of an optimal solution to Problem MCF. We prove that there exists a circulation $\bar{f}$ in $G(f^0)$ such that $D(\bar{f}) \leq -(D(f) - 2D)$, $C(\bar{f}) \leq OPT_1$.

Lemma 2: Let f be an (s,t)-flow in G such that $|f| = 2$ and $D(f) \geq 2D$, and let $G(f)$ be the residual network of G imposed by f. Then, there exists a circulation $\bar{f}$ in $G(f)$, such that $D(\bar{f}) \leq -(D(f) - 2D)$, $C(\bar{f}) \leq OPT_1$.

Proof: Let f^* be the optimal solution to Problem MCF. Note that $D(f^*) \leq 2D$. Let $\hat{G}$ be a subgraph of G that includes only links l for which hold either $f_l = 1$ or $f_l^* = 1$. By Theorem 3, there exists a circulation $\bar{f}$ in the residual graph $\hat{G}(f)$ of $\hat{G}$ imposed by f such that $D(f^*) = D(f) + D(\bar{f})$. Since $D(f^*) \leq 2D$, we have $D(\bar{f}) \leq 2D - D(f) = -(D(f) - 2D)$. We observe that all links with positive cost in $\hat{G}(f)$ belong to f^*. Thus, the cost of $\bar{f}$ is at most OPT_1. We also observe that $\hat{G}(f) \subseteq G(f)$. Thus, $\bar{f}$ belongs to $G(f)$ and the lemma follows. ■

Lemma 2 implies that there exists a circulation $\bar{f}$ in $G(f)$, such that $D(\bar{f}) \leq -(D(f) - 2D)$ and $C(\bar{f}) \leq OPT_1$. Since the delay $D(\bar{f})$ of circulation $\bar{f}$ is negative, it holds that $\frac{D(\bar{f})}{C(\bar{f})} \leq \frac{D(\bar{f})}{OPT_1} \leq -\frac{D(f)-2D}{OPT_1}$. As circulation $\bar{f}$ includes a number of cycles, there must be a cycle $W \in \bar{f}$ for which it holds that $\frac{D(W)}{C(W)} \leq -\frac{(D(f)-2D)}{OPT_1}$.

Corollary 1: Let f be an (s,t)-flow in G such that $|f| = 2$ and $D(f) \geq 2D$, and let $G(f)$ be the residual network of G imposed by f. Then, there exists a cycle W in $G(f)$ for which it holds that $\frac{D(W)}{C(W)} \leq -\frac{(D(f)-2D)}{OPT_1}$.

Proof: By Lemma 2, there exists a circulation $\bar{f}$ in $G(f)$, such that $D(\bar{f}) \leq -(D(f) - 2D)$ and $C(\bar{f}) \leq OPT_1$. Such a circulation is a set of cycles and the average ratio $\frac{D(W)}{C(W)}$ for the cycles in this set is $\frac{D(\bar{f})}{C(\bar{f})} \leq -\frac{D(f)-2D}{OPT_1}$. Hence, there exists at least one cycle such that $\frac{D(W)}{C(W)} \leq -\frac{D(f)-2D}{OPT_1}$. ■

In the next lemma we prove that the cost of each negative delay cycle identified by Procedure IMPROVEFLOW is at least 1 and at most $(2k+1)OPT_1$.

Lemma 3: Let W be a cycle identified in the line 7 of Procedure IMPROVEFLOW. Then, it holds that $1 \leq C(W) \leq (k+1)OPT_1$.

Proof: Let W be an augmenting cycle in $G(f)$. It is easy to verify that flow f contains no cycles, i.e., f includes two disjoint (s,t)-paths. It follows that any cycle in $G(f)$ must include at least one link that does not belong to f. Since all links in W have non-negative cost, it follows that the total cost of W is at least 1.

We proceed to prove that $C(W) \leq (2k+1)OPT_1$. By Corollary 1, $\frac{D(W)}{C(W)} \leq -\frac{D(f)-2D}{OPT_1}$, hence $C(W) \leq \frac{-D(W)OPT_1}{D(f)-2D}$. We note that $D(W) \geq -D(f)$, otherwise augmentation of f by W would result in an (s,t)-flow in G whose delay is negative,

```
Algorithm 2DP-2 $(G, s, t, D, k)$

1  $(P_1, P_2) \leftarrow$2DP-1$(G, s, t, D, \frac{1}{N})$
2  $f^0 \leftarrow \{P_1, P_2\}$;
3  **if** $D(f^0) \leq 2D(1 + \frac{1}{k})$ **then**
4      **return** $P_1$ and $P_2$
5  $f \leftarrow$ImproveFlow$(G, f^0, D, k)$
6  Decompose flow $f$ into two paths, $\hat{P}_1$ and $\hat{P}_2$, such that
   $D(\hat{P}_1) \leq D(\hat{P}_2)$
7  **return** $\hat{P}_1$ and $\hat{P}_2$

   Procedure ImproveFlow$(G, f^0, D, k)$
1  $f \leftarrow f^0$
2  **while** $D(f) > 2D(1 + \frac{1}{k})$ **do**
3      Construct the residual network $G(f)$ of $G$
       imposed by $f$:
4          Add to $G(f)$ each link $l$ in $G$ for which $f_l = 0$
5          **for** each link $(u, v)$ in $G$ for which $f_{(u,v)} = 1$ **do**
6              Add a link $(v, u)$ to $G(f)$ with
               $d_{(v,u)} = -d_{(u,v)}$ and $c_{(v,u)} = 0$
7      Find a cycle $W$ in $G(f)$ that minimizes $\frac{D(W)}{C(W)}$
8      Augment flow $f$ along $W$
9  **return** $f$
```

Fig. 4. Algorithm 2DP-2

which contradicts the fact that all links in G have positive delay. Thus, it follows that $C(W) \leq \frac{D(f)OPT_1}{D(f)-2D} \leq \frac{OPT_1}{1-2D/D(f)}$. Since $D(f) \geq 2D(1 + \frac{1}{k})$, we have $C(W) \leq \frac{(D+D/k)OPT_1}{D/k} = (k + 1)OPT_1$. ■

Lemma 4: Let f be a flow in G, $G(f)$ be the residual network of G imposed by f. Let W be a cycle in $G(f)$ and f' be a flow resulting from augmenting f along W. Then, $D(f') = D(f) + D(W)$ and $C(f') \leq C(f) + C(W)$.

Proof: Each link with positive delay in W is added to $\bar{f}$. For each link l in W with negative delay d_l, we delete from f a link whose delay is $|d_l|$. Hence, $D(f') = D(f) + D(W)$. Since flow f' includes links from f and W and since each link in W has positive cost, it follows that $C(f') \leq C(f) + C(W)$. ■

Lemma 4 implies that an augmentation of a flow f by a cycle W decreases the delay $D(f)$ of f by $|D(W)|$ (since $D(W)$ is negative) and increases its cost by $C(W)$. In the following, we prove that the number of iterations needed in order to achieve $D(f) \leq D(2 + \frac{1}{k})$ is at most $2OPT_1 \log k$.

Our prove uses the *Geometric Improvement Approach*. In particular, we use the following theorem, taken from [18].

Theorem 4: Suppose that an algorithm A iteratively minimizes some value z such that z^0 is the initial value of z, z^i is the value of z at the i-th iteration and z^* the minimum objective function value. Furthermore, suppose that the algorithm A guarantees that, for every iteration i,

$$z^i - z^{i+1} \geq \zeta(z^i - z^*) \tag{4}$$

for some constant ζ with $0 < \zeta < 1$. Then, within $2/\zeta$ consecutive iterations, it holds that $z \leq z^* + \frac{z^0 - z^*}{2}$. Furthermore, the algorithm A terminates after at most $\frac{2\log(z^* - z^0)}{\zeta}$ iterations.

Proof: See [18]. ■

Theorem 5: Procedure ImproveFlow returns a flow f with delay $D(f) \leq 2D(1+\frac{1}{k})$ and cost $C(f) \leq (k + 2\log k + 1 + \frac{1.5}{N})OPT_1$. The computational complexity of Procedure ImproveFlow is $\mathcal{O}(MN \cdot OPT_1 \log(CD) \log k)$.

Proof: Consider the main loop of Procedure ImproveFlow, i.e., the loop that begins at line 2. We denote by f_i the state of flow f at the beginning of iteration i (at line 7). Further, we denote by W_i the cycle that has been identified at iteration i and by C_i the cost of W_i, i.e., $C_i = C(W_i)$.

First, we prove that the procedure stops within $\frac{2\log k}{OPT_1}$ iterations, i.e., at some iteration $j \leq \frac{2\log k}{OPT}$ it holds that $D(f_j) \leq 2D(1 + \frac{1}{k})$. By Lemma 3, at each iteration i we identify a cycle W_i whose cost is at least 1. Corollary 1 implies that the delay of the cycle is at least $-\frac{(D(f_i)-2D)}{OPT_1}$. Thus, by Lemma 4, the delay of flow f is reduced by at least $\frac{(D(f_i)-2D)}{OPT_1}$ at each iteration. By Theorem 4, after at most $\frac{2\log k}{OPT_1}$ iterations, it holds that $D(f) \leq 2D + \frac{D(f^0)-2D}{k} \leq 2D(1 + \frac{1}{k})$.

Second, we prove that the procedure returns a flow f whose cost is at most $C(f^0) + (2k + 2\log k + 2.5)OPT_1$. Let j be the last iteration of the procedure. We show that the total cost of all cycles identified in the first $j - 1$ iterations is at most $2\log kOPT_1$. We assume, by way of contradiction, that $\sum_{i=1}^{j-1} C(W_i) \geq 2\log kOPT_1$. We consider two cases. In the first case all cycles $W_1, \cdots, W_{j-1}$ have cost of exactly 1. Then, at each iteration, the delay of flow f decreases by at least $\frac{D(f_i)-2D}{OPT_1}$. Thus, by Theorem 4, after $j-1$ iterations, the delay of flow f is at most $2D+\frac{D(f_0)-2D}{k} \leq 2D(1+\frac{1}{k})$, which contradicts the fact that $j-1$ is not a last iteration. The last inequality follows from the fact that $D(f^0) \leq 3D$ (by Theorem 1). Now we consider the second case, in which the cost a each W_i may be more than 1. In this case, at iteration i, the delay of flow f decreases by at least $\frac{D(f_i)-2D}{OPT_1}C(W_i)$. We note that augmenting of flow f along cycle W_i is equivalent to augmenting it along $|C(W_i)|$ cycles of cost 1 and delay $\frac{-(D(f_i)-2D)}{OPT_1}$. Thus, if $\sum_{i=1}^{j-1} C(W_i) \geq 2\log kOPT_1$, then cycles $W_1, \cdots, W_j$ can be substituted by $2\log kOPT_1$ cycles of cost 1. This implies that after iteration $j-1$ we have $D(f_{j-1}) \leq 2D(1+\frac{1}{k})$, and again we have a contradiction. By Lemma 3, the cost of the cycle W_j identified at the last iteration is at most $(k+1)OPT_1$. Thus, the total cost of flow f returned by Procedure IMPROVEFLOW is at most $(k+1+2\log k)OPT_1 + C(f^0) \leq (k+2\log k+1+\frac{1.5}{N})OPT_1$.

Finally, we analyze the computational complexity of Procedure IMPROVEFLOW. We proved that the procedure performs at most $2\log kOPT_1$ iterations. The running time of each iteration is dominated by the time required to identify a cycle that minimizes $\frac{D(W)}{C(W)}$. Such a cycle is identified by invoking the *minimum cost-to-time ratio cycle algorithm* [18], which incurs $\mathcal{O}(MN\log(CD))$ time. We conclude that the computational complexity of the procedure is $\mathcal{O}(MN \cdot OPT_1 \log k \log(CD))$. ■

We are ready now to prove Theorem 2.

Theorem 2: Algorithm 2DP-2 computes, in $\mathcal{O}(MN \cdot OPT \log k \log(CD))$ time, a $(1+\frac{1}{k}, k(1+\gamma))$ approximate solution for Problem 2DP, where $\gamma \leq \frac{2(\log k+1)}{k}$.

Proof: The computational complexity of Algorithm 2DP-2 is dominated by the time required to identify paths P_1 and P_2 in (line 1) and the running time of Procedure IMPROVEFLOW. By Theorem 1 The computational complexity of Algorithm 2DP-1 is $\mathcal{O}(MN^2)$ (by Theorem 1). By Theorem 5, Procedure IMPROVEFLOW requires $\mathcal{O}(MN\log k \cdot OPT_1 \log(CD))$ time. Since $N \leq OPT$ and $OPT_1 \leq OPT$, the computational complexity of Algorithm 2DP-2 is $\mathcal{O}(MN \cdot OPT \log k \cdot \log(CD))$.

By Theorem 5, Procedure IMPROVEFLOW returns a flow f with delay $D(f) \leq 2D(1+\frac{1}{k})$ and cost $C(f) \leq (k+2\log k+1+\frac{1.5}{N})OPT_1 \leq k(1+\gamma)OPT$, where $\gamma \leq \frac{2(\log k+1)}{k}$. We conclude that Algorithm 2DP-2 computes a $(1+\frac{1}{k}, k(1+\gamma))$ approximate solution for Problem 2DP. ■

VI. MINIMIZING THE COMPUTATIONAL COMPLEXITY

While Algorithm 2DP-2 provides a good approximate solution for Problem 2DP, its computational complexity is proportional to the cost OPT of the optimal solution. The algorithm can be used in settings in which the cost of each link is a relatively small value. However, for settings with high cost values, its computational complexity may be prohibitive. Accordingly, in this section we present Algorithm 2DP-3 whose computational complexity does not depend on the values of links costs. The algorithm uses the same ideas as Algorithm 2DP-2 and, in addition, employs the *cost scaling* approach [16] in order to reduce the computational complexity.

Algorithm 2DP-3 begins by invoking Algorithm 2DP-1, which identifies two paths P_1 and P_2. These paths form a flow $f^0 = \{P_1, P_2\}$ and we construct the residual network $G(f^0)$ of G imposed by flow f^0. In the previous section we showed that there exists a circulation $\bar{f}$ in $G(f^0)$, such that $D(\bar{f}) \leq -(D(f)-2D)$, $C(\bar{f}) \leq OPT_1$ (Lemma 2). Algorithm 2DP-2 uses this fact to reduce the delay of the flow f^0. In particular, it invokes Procedure IMPROVEFLOW, whose running time depends on the cost of the circulation, i.e., OPT_1.

The basic idea of Algorithm 2DP-3 is to reduce the cost of each link in G by a certain factor. The cost of the circulation $\bar{f}$ in the resulting graph, and in turn, the computational complexity of Procedure IMPROVEFLOW, are much smaller. A key requirement in the scaling approach is to get sufficiently tight upper and lower bounds L and U on the cost OPT_1 of the optimal solution to Problem MCF. We present an efficient technique for obtaining these bounds in Section VI-A.

We proceed to describe cost scaling in more detail. We scale the cost c_l of each link l in G, replacing it by c'_l, as follows:

$$c'_l = \left\lfloor \frac{c_l}{\Delta} \right\rfloor + 1, \tag{5}$$

where $\Delta = \frac{L\varepsilon}{2N}$.

Let $G'(f^0)$ be the residual graph of G imposed by flow f^0 with scaled link costs. We show that there exists a circulation $\bar{f}$ in $G'(f^0)$, such that $D(\bar{f}) \leq -(D(f)-2D)$ and $C(\bar{f}) \leq \frac{2N \cdot U}{\varepsilon \cdot L} + 2N$. Let f^{opt} be the optimal solution for Problem 2DP. It is easy to verify that flow f^{opt} contains at most $2N$ links. Thus, the cost of $\bar{f}$ with respect to the scaled link costs is at most $OPT'_1 = \frac{2N \cdot OPT_1}{\varepsilon \cdot L} + 2N$. It can be shown, in the same way as in Lemma 2, that there exists a circulation $\bar{f}$ in $G'(f^0)$, such that $D(\bar{f}) \leq -(D(f)-2D)$ and whose cost is at most $\frac{2N \cdot OPT_1}{\varepsilon \cdot L} + 2N$. Thus, with scaled link costs Procedure IMPROVEFLOW performs just $\mathcal{O}\left(\frac{N \cdot U}{\varepsilon \cdot L}\right)$ iterations. For sufficiently tight lower and upper bounds, L and U, the number of iterations is small.

Scaling allows to reduce the computational complexity of the algorithm, but it incurs some penalty in terms of the solution cost. Let $\bar{f}$ we a circulation whose cost with respect to the scaled link costs is at most OPT'_1. The cost of that circulation with respect to original link costs is at most

$$\Delta \cdot OPT'_1 \leq OPT_1 + \varepsilon L \leq (1+\varepsilon)OPT_1.$$

Lower values of ε yield better approximate solutions on the expense of the running time of the algorithm.

The detailed description of the algorithm appears in Fig. 5.

```
Algorithm 2DP-3 $(G, s, t, D, k, \varepsilon)$

1  $(P_1, P_2) \leftarrow$2DP-1$(G, s, t, D, \frac{1}{N})$
2  $f^0 \leftarrow \{P_1, P_2\}$
3  if $D(f^0) \le 2D(1+\frac{1}{k})$ then
4      return $P_1$ and $P_2$
5  $L, U \leftarrow$ BOUND$(G, s, t, f, D)$
6  $\Delta \leftarrow \frac{L\varepsilon}{2N}$
7  for each link $l \in E$ do
8      $c_l \leftarrow \lfloor \frac{c_l}{\Delta} \rfloor + 1$
9  $f \leftarrow$ IMPROVEFLOW$(G, f^0, D, k)$
10 Decompose flow $f$ into two paths, $\hat{P}_1$ and $\hat{P}_2$, such that
   $D(\hat{P}_1) \le D(\hat{P}_2)$
11 return $\hat{P}_1$ and $\hat{P}_2$

   Procedure BOUND $(G(V,E), s, t, D)$
1  let $c^1 < c^2 < \cdots < c^r$ the distinct costs values of the
    links.
2  $low \leftarrow 1$; $high \leftarrow r$
3  while $low < high - 1$
4      $j \leftarrow \lfloor (high + low)/2 \rfloor$
5      $E' \leftarrow \{l \mid c_l \le c^j\}$
6      find a minimum delay flow $f$ between $s$ and $t$ in
       $(G'(V, E')$ such that $|f| = 2$
7      if $D(f) > 2D$ then
8          $high \leftarrow j$
9      else
10         $low \leftarrow j$
11 $U \leftarrow 2N \cdot c^{high}$; $L \leftarrow c^{high}$
12 return $L$, $U$
```

Fig. 5. Algorithm 2DP-3

A. Computing Lower and Upper Bounds, L and U

In this subsection we present Procedure BOUND (see Fig. 5), which identifies lower and upper bounds L, U on OPT_1 such that $U/L \le 2N$. We use the technique presented in [17].

We denote by $c^1 < c^2 < \cdots < c^r$ the distinct costs values of the links. Our goal is to find the maximum cost value $c^* \in \{c^i\}$ such that the graph G' derived from G by omitting all links whose cost is greater than c^*, does not contain a *feasible* flow f, i.e., a flow with value $|f| = 2$ and with delay $D(f) \le 2D$. Clearly, a feasible flow contains at least one link whose cost is c^* or more, hence c^* is a lower bound on OPT_1. In addition, there exists a feasible flow that comprises links whose cost is c^* or less. Since an optimal flow f includes at most two paths, it includes at most $2N$ links. We conclude that $2N \cdot c^*$ is an upper bound on OPT_1.

Procedure BOUND performs a binary search on the values $c^1, c^2, \cdots, c^r$. At each iteration, we need to check whether $c \le c^*$, where c is the current estimate of c^*. For this purpose, we remove from G all links whose cost is more than c, and assign the unit cost to the remaining links. Then, we find a minimum delay flow in the resulting graph. For this purpose we employ the minimum cost disjoint path algorithm taken from [2], with respect to links delays. If this algorithm returns a feasible flow, then $c \ge c^*$; otherwise, $c < c^*$. Procedure BOUND performs $\mathcal{O}(\log N)$ iterations.

B. Analysis of Algorithm 2DP-3

The following theorem establishes the correctness of Algorithm 2DP-3.

Theorem 6: Algorithm 2DP-3 computes, in $\mathcal{O}(\frac{\log k \cdot M \cdot N^3}{\varepsilon}\log(CD))$ time, a $(1+\frac{1}{k}, k(1+\gamma)(1+\varepsilon))$-approximate solution for Problem 2DP, where $\gamma \le \frac{2(\log k + 1)}{k}$.

Proof: Let f^{opt} be an optimum solution to Problem MCF, i.e., $C(f^{opt}) = OPT_1$. Note that f^{opt} can be represented by two disjoint (s,t)-paths. Indeed, if f^{opt} contains a cycle, this cycle can be eliminated, resulting in a flow whose cost is less than that of f^{opt}. We conclude that f^{opt} includes at most $2N$ links. For each link $l \in G$ it holds that $\frac{c_l}{\Delta} \le c'_l \le \frac{c_l}{\Delta} + 1$. Thus, the cost of f^{opt} with respect to scaled link costs is at most $OPT'_1 = \frac{OPT_1}{\Delta} + 2N$. By Theorem 5, Procedure IMPROVEFLOW returns a flow f with delay $D(f) \le 2D(1+\frac{1}{k})$ and cost $C(f) \le (k + 2\log k + 1 + \frac{1.5}{N})OPT'_1 = k(1+\gamma)OPT'_1$, where $\gamma \le \frac{2(\log k+1)}{k}$. The cost of flow f with respect to original link costs is

$$C(f) \le k(1+\gamma)\Delta OPT'_1 \le k(1+\gamma)(OPT_1 + L\varepsilon)$$
$$\le k(1+\gamma)(1+\varepsilon)OPT.$$

We proceed to analyze the computational complexity of Algorithm 2DP-3. We begin with Procedure BOUND. As discussed above, the procedure executes $\mathcal{O}(\log N)$ iterations of the main loop (i.e., the loop that begins at line 3 of the procedure). At each iteration, the procedure invokes the minimum cost disjoint path algorithm [2], whose running time is $\mathcal{O}(M + N \log N)$. We conclude that the computational complexity of Procedure BOUND is $\mathcal{O}((M + N \log N) \log N)$.

Since Procedure IMPROVEFLOW is invoked on the graph with scaled link costs, its computational complexity is $\mathcal{O}(MN \cdot OPT'_1 \log(CD) \log k)$. Since

$$OPT'_1 \leq \frac{OPT_1}{\Delta} + 2N \leq \frac{2N \cdot U}{\varepsilon L} \leq \frac{2N^2}{\varepsilon}$$

the computational complexity of Algorithm 2DP-3 is $\mathcal{O}(\frac{\log k \cdot M \cdot N^3}{\varepsilon} \log(CD))$. ■

VII. FURTHER IMPROVEMENTS

In this section we further improve the computational complexity of our solution. To that end we extend Algorithm 2DP-3, by introducing the following changes. First, we modify Procedure IMPROVEFLOW, in order to ensure that its running time is bounded by a certain value. Second, we introduce an additional procedure, referred to as Procedure IMPROVEBOUNDS, whose purpose is to obtain tighter lower and upper bounds on the cost of the optimal solution OPT_1 to Problem MCF.

More specifically, we add a new parameter $\hat{U}$ to Procedure IMPROVEFLOW in order to ensure that the cost of the returned flow f is no more than $2k\hat{U}$. With this modification, Procedure IMPROVEFLOW has the following properties.

Lemma 5:

1) If $\hat{U} \geq OPT_1$, then Procedure IMPROVEFLOW returns a flow f with delay $D(f) \leq 2D(1 + \frac{1}{k})$.
2) If Procedure IMPROVEFLOW does not fail, it returns a flow f with delay $D(f) \leq 2D(1 + \frac{1}{k})$ and cost $C(f) \leq \min\{2k\hat{U}, (k + 2\log k + 1 + \frac{1.5}{N})OPT_1\}$.
3) The computational complexity of Procedure IMPROVEFLOW is $\mathcal{O}(MN \cdot \min\{OPT_1, \hat{U}\} \log(CD) \log k)$.

Proof: The proof follows the same lines as that of Theorem 5. ■

We proceed to describe Procedure IMPROVEBOUNDS. The procedure obtains upper and lower bounds, L and U, such that:

- L is a lower bound on the cost of an optimal solution OPT_1 to Problem MCF, i.e., for each flow in G with delay $D(f) \leq 2D$, it holds that $C(f) \geq L$.
- U is an upper bound on the cost of a flow in G with delay $D(f) \leq 2D(1 + \frac{1}{k})$, i.e., for each flow in G with delay $D(f) \leq 2D(1 + \frac{1}{k})$, it holds that $C(f) \leq U$.

Lower and upper bounds L and U for which it holds that $\frac{U}{L} \leq 2N$ can be obtained through Procedure BOUND. The goal of Procedure IMPROVEFLOW is to improve these bounds such that $\frac{U}{L} \leq 16k^2$. At each iteration we compare a test value $B = \sqrt{L \cdot U}$ with the cost of the optimal solution and update the bounds L and U accordingly. More specifically, we scale the link costs by a factor $\Delta = \frac{B}{2N}$ and invoke Procedure IMPROVEFLOW with $\hat{U} = 4N$ on the graph with scaled link costs. As we show in Lemma 6, if Procedure IMPROVEFLOW fails, then $OPT_1 > B$, hence we set $L = B$. Otherwise, Procedure IMPROVEFLOW returns a flow whose delay and cost are at most $2D(1+\frac{1}{k})$ and $4kB$, respectively, hence $U \leq 4kB$. Accordingly, in this case we set $U = 4kB$.

Note that, if the ratio U/L is equal to x at the beginning of an iteration, then at the end of the iteration we have $\frac{U}{L} \leq 4k\sqrt{x}$. Thus, it can be shown that, after $\mathcal{O}(\log \log N)$ iterations, the ratio U/L is at most $16k^2$. We then use these tight upper and lower bounds in order to efficiently find an approximate solution to Problem 2DP. The detailed description of the algorithm, referred to as Algorithm 2DP-4, appears in Fig. 5.

A. *Analysis of Algorithm* 2DP-4

Lemma 6: Procedure IMPROVEBOUNDS returns valid upper and lower bounds L and U.

Proof: We note that Procedure BOUND returns valid upper and lower bounds L and U. This follows from the fact that the minimum cost of a flow whose delay is at most $2D$ is lower than the minimum cost of a flow whose delay is at most $2D(1 + \frac{1}{k})$.

We show that the lower and upper bounds remain valid during the execution of the procedure. First, we prove that if Procedure IMPROVEFLOW fails then $OPT_1 > B$. By way of contradiction, suppose that this is not the case, i.e., $OPT_1 \leq B$. Procedure IMPROVEFLOW is applied to the graph whose costs were scaled by factor of $\Delta = \frac{B}{2N}$. In the resulting graph, the cost OPT'_1 of the optimal solution to Problem MCF is at most $\frac{2N \cdot OPT_1}{B} + 2N$, which is less than $4N$. Thus, by Lemma 5, part 1, Procedure IMPROVEFLOW returns a flow f with delay $D(f) \leq 2D(1+\frac{1}{k})$, which contradicts the fact the the procedure fails. Second, by Lemma 5, part 2, if Procedure IMPROVEFLOW does not fail, it returns a flow f whose cost with respect to the scaled link costs is at most $2k\hat{U}$ and delay is at most $2D(1 + \frac{1}{k})$. The cost of flow f with respect to the original costs is at most $2k\hat{U}\Delta = 4kB$. In this case, the minimum cost of a flow with delay at most $2D(1 + \frac{1}{k})$ is bounded by $4kB$, hence U remains to be a valid upper bound. ■

Theorem 7: Algorithm 2DP-4 computes, in $\mathcal{O}\left(\frac{MN^2k^2 \log k}{\varepsilon} \log(CD)\right)$ time, a $(1+\frac{1}{k}, k(1+\gamma)(1+\varepsilon))$-approximate solution for Problem 2DP, where $\gamma \leq \frac{2(\log k+1)}{k}$.

```
Algorithm 2DP-4 $(G, s, t, D, k, \varepsilon)$

1   $(P_1, P_2) \leftarrow$2DP-1$(G, s, t, D, \frac{1}{N})$
2   $f^0 \leftarrow \{P_1, P_2\}$
3   if $D(f^0) \leq 2D(1 + \frac{1}{k})$ then
4       return $P_1$ and $P_2$
5   $L, U \leftarrow$ Bound$(G, s, t, D)$
6   $L, U \leftarrow$ ImproveBounds$(G, s, t, f^0, D, L, U)$
7   $\Delta \leftarrow \frac{L\varepsilon}{2N}$
8   for each link $l \in E$ do
9       $c_l \leftarrow \lfloor \frac{c_l}{\Delta} \rfloor + 1$
10  $\hat{U} \leftarrow \left\lfloor \frac{U}{\Delta} \right\rfloor + 2N + 1$
11  $f \leftarrow$ ImproveFlow$(G, f^0, D, k, \hat{U})$
12  Decompose flow $f$ into two paths, $\hat{P}_1$ and $\hat{P}_2$,
    such that $D(\hat{P}_1) \leq D(\hat{P}_2)$
13  return $\hat{P}_1$ and $\hat{P}_2$

    Procedure ImproveBounds $(G(V, E), s, t, f, D, L, U)$
1   while $U/L > 16k^2$ do
2       $B \leftarrow \sqrt{L \cdot U}$
3       $\Delta \leftarrow \frac{B}{2N}$
4       for each link $l \in E$ do
5           $c_l \leftarrow \lfloor \frac{c_l}{\Delta} \rfloor + 1$
6       $\hat{U} \leftarrow 4N$
7       $f \leftarrow$ ImproveFlow$(G, f^0, D, k, \hat{U})$
8       if Procedure ImproveFlow returned FAIL then
9           $L \leftarrow B$
10      else
11          $U \leftarrow 4kB$
12  return $L$, $U$

    Procedure ImproveFlow$(G, f^0, D, k, \hat{U})$
1   $f \leftarrow f^0$
2   while $C(f) \leq 2k\hat{U}$ do
3       $G(f) \leftarrow$ residual network of $G$ imposed by $f$
4       find a cycle $W$ in $G(f)$ that minimizes $\frac{D(W)}{C(W)}$
5       augment flow $f$ along $W$
6       if $D(f) \leq D(2 + \frac{1}{k})$ then
7           return $f$
8   return FAIL
```

Fig. 6. Algorithm 2DP-4

Proof: As we proved in Lemma 6, Procedure IMPROVEBOUNDS returns valid lower and upper bounds L and U. Thus, it can be shown that Algorithm 2DP-4 computes a $(1 + \frac{1}{k}, k(1 + \gamma)(1 + \varepsilon))$-approximate solution for Problem 2DP, where $\gamma \leq \frac{2(\log k+1)}{k}$ (The proof follows the same lines as the one for Theorem 6).

We proceed to analyze the computational complexity of the algorithm. As discussed above, Procedure IMPROVEBOUNDS executes $\mathcal{O}(\log\log N)$ iterations of its main loop (i.e., the loop that begins in line 1). At each iteration we invoke Procedure IMPROVEFLOW, whose complexity is $\mathcal{O}(MN^2 \log(CD) \log k)$. We conclude that the running time of Procedure IMPROVEBOUNDS is $\mathcal{O}(MN^2 \log\log N \log(CD) \log k)$. By Lemma 5, the computational complexity of Procedure IMPROVEFLOW, invoked at line 11 is $\mathcal{O}(MN \cdot \hat{U} \log(CD) \log k)$. Since the procedure is invoked with $\hat{U} = \lfloor \frac{U}{\Delta} \rfloor + 2N + 1 = \mathcal{O}(\frac{k^2 N}{\varepsilon})$, the computational complexity of the procedure is $\mathcal{O}(\frac{MN^2 k^2 \log k}{\varepsilon} \log(CD))$. We assume that $\frac{1}{\varepsilon} \geq \log\log N$, hence the computational complexity of the algorithm is dominated by the time incurred by invocation of Procedure IMPROVEFLOW in line 11, i.e., $\mathcal{O}(\frac{MN^2 k^2 \log k}{\varepsilon} \log(CD))$. ■

VIII. Lower Bound

In this section we prove that finding two disjoint paths that satisfy a given delay constraint D is an $\mathcal{NP}$-hard problem. Furthermore, we show that no polynomial algorithm can approximate this problem by a factor α such that $1 \leq \alpha < 2$. That is, finding two disjoint paths such that the delay of each path is at most αD is an intractable problem for any $1 \leq \alpha < 2$. Specifically, we show a reduction from the problem of finding two link disjoint paths in a directed network, one between s_1 and t_1 and the other between s_2 and t_2. This problem is known to by $\mathcal{NP}$-hard [19].

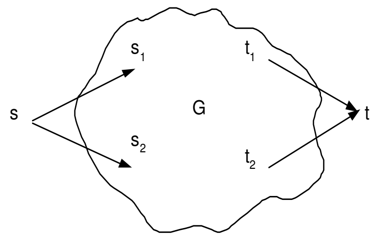

Fig. 7. Construction of auxiliary graph $\hat{G}$

Our proof follows the same lines as the proof presented in [20] for a related problem. By way of contradiction, assume that there exists an algorithm, say Algorithm A, which identifies, in polynomial time, two disjoint (s,t)-paths such that the delay of each path is at most αD. We show that this algorithm can find two disjoint paths between s_1 and t_1 and between s_2 and t_2 in a directed graph G. We build an auxiliary graph $\hat{G}$ formed from G by adding two nodes s and t and four links (s,s_1), (s,s_2), (t_1,t) and (t_2,t). We assign delays to these four links as follows: $d_{(s,s_1)} = d_{(t_2,t)} = 1$ and $d_{(s,s_2)} = d_{(t_1,t)} = 0$. All other links in $\hat{G}$ are assigned zero delay. Fig. 7 depicts the construction of an auxiliary graph $\hat{G}$. By employing Algorithm A (for $D = 1$) we can find two disjoint paths P_1 and P_2 between s and t such that $D(P_1) \leq \alpha$ and $D(P_2) \leq \alpha$. However, if $\alpha < 2$, then one of the paths must connect s_1 and t_1 and the other must connect s_2 and t_2. This contradicts the fact that the problem of finding such two disjoint paths in a directed network is $\mathcal{NP}$-hard.

IX. Conclusion

In this paper we investigated the fundamental problem of provisioning disjoint QoS paths. We presented a comprehensive analysis of the problem, by using the framework of *network flows*. We showed that any polynomial algorithm for this problem violates the delay constraint by at least a factor of 2. In addition, we indicated trade-offs between violation of the delay constraint, cost and computational complexity.

More specifically, the major contribution of our study are four approximation algorithms for the considered problem. The first algorithm is conceptually simple and has low computational complexity. This algorithm identifies a solution that violates the delay constraint by factors of 1.5 and 3 for the primary and restoration paths, respectively. The second algorithm reduces the delay violation at the expense of higher cost and computational complexity. The third and fourth algorithms, achieve similar cost and delay ratios as the second, but have significantly lower computational complexity. In particular, the algorithms compute, for any fixed $\varepsilon > 0$ and integer $k > 0$ a solution that violates the delay constraint by factors of at most $1 + \frac{1}{k}$ and $2(1 + \frac{1}{k})$ for the primary and restoration paths, respectively, and whose cost is at most $k(1+\gamma)(1+\varepsilon)$ times more than the optimum, where γ is a small value bounded by $\frac{2(\log k+1)}{k}$.

We have indications that our results can be extended to a broader class of network restoration and network design problems. In particular, our methods, especially the cycle-cancellation approach, can be used in order to solve h-disjoint path problems for any $h > 2$. In addition, the techniques established in the study can be used also in the domain of *local restoration* [21], where a restoration topology comprises of a primary path and several bridges, each protecting a portion of the primary path.

References

[1] E. Mannie and D. Papadimitriou (editors), "Recovery (Protection and Restoration) Terminology for Generalized Multi-Protocol Label Switching (GMPLS)," Internet draft, Internet Engineering Task Force, May 2003.
[2] J. Suurballe and R. Tarjan, "A Quick Method for Finding Shortest Pairs of Disjoint Paths," *Networks*, vol. 14, pp. 325–336, 1984.
[3] S. Chen and K. Nahrstedt, "An Overview of Quality-of-Service Routing for the Next Generation High-Speed Networks: Problems and Solutions," *IEEE Network, Special Issue on Transmission and Distribution of Digital Video*, vol. 12, no. 6, pp. 64–79, November/December 1998.
[4] F.A. Kuipers, T. Korkmaz, M. Krunz, and P. Van Mieghem, "A Review of Constraint-Based Routing Algorithms," in *Technical Report*, Lausanne, Switzerland, June 2002.
[5] K. Kar, M. Kodialam, and T. V. Lakshman, "Routing Restorable Bandwidth Guaranteed Connections using Maximum 2-Route Flows," in *Proceedings of IEEE INFOCOM'2002*, New York, NY, USA, June 2002.
[6] G. Li, D. Wang, C. Kalmanek, and R. Doverspike, "Efficient Distributed Path Selection for Shared Restoration Connections," in *Proceedings of IEEE INFOCOM'2002*, New York, NY, USA, June 2002.
[7] N. Taft-Plotkin, B. Bellur, and R.G. Ogier, "Quality-of-Service Routing Using Maximally Disjoint Paths," in *Proceedings IEEE/IFIP IWQoS*, London, UK, June 1999.
[8] M.R. Garey and D.S. Johnson, *Computers and Intractability*, Freeman, San Francisco, CA, USA, 1979.
[9] J.P. Lang and B. Rajagopalan (editors), "Generalized MPLS Recovery Functional Specification," Internet draft, Internet Engineering Task Force, January 2003.
[10] "Private Network-Network Interface Specification v1.0 (PNNI)," ATM Forum Technical Committee, March 1996.
[11] S. Blake, "An architecture for Differentiated Services," RFC No. 2475, Internet Engineering Task Force, December 1998.
[12] A. Orda and A. Sprintson, "Precomputation Schemes for QoS Routing," *IEEE/ACM Transactions on Networking*, vol. 11, no. 4, pp. 578–591, August 2003.
[13] I. Cidon, R. Rom, and Y. Shavitt, "Analysis of Multi-path Routing," *IEEE/ACM Transactions on Networking*, vol. 7, no. 6, pp. 885–896, 1999.
[14] S. Lee and M. Gerla, "Split Multipath Routing with Maximally Disjoint Paths in Ad Hoc Networks," in *Proceedings of the IEEE ICC'2001*, 2001.

[15] F. Ergun, R. Sinha, and L. Zhang, “An Improved FPTAS for Restricted Shortest Path,” *Information Processing Letters*, vol. 83, no. 5, pp. 237–293, September 2002.
[16] R. Hassin, “Approximation Schemes for the Restricted Shortest Path Problem,” *Mathematics of Operations Research*, vol. 17, no. 1, pp. 36–42, February 1992.
[17] D.H. Lorenz and D. Raz, “A Simple Efficient Approximation Scheme for the Restricted Shortest Path Problem,” *Operations Research Letters*, vol. 28, no. 5, pp. 213–219, June 2001.
[18] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Networks Flows*, Prentice-Hall, NJ, USA, 1993.
[19] S. Fortune, J. Hopcroft, and J. Wyllie, “The Directed Subgraph Homeomorphism Problem,” *Theoretical Computer Science*, vol. 10, no. 2, pp. 111–121, 1980.
[20] C.L. Li, T. McCormick, and D. Simchi-Levi, “The Complexity of Finding Two Disjoint Paths with Min-Max Objective Function,” *Discrete Applied Mathematics*, vol. 26, pp. 105–115, 1990.
[21] Y. Bejerano, Y. Breitbart, A. Orda, R. Rastogi, and A. Sprintson, “Algorithms for Computing QoS Paths with Restoration,” in *Proceedings of IEEE INFOCOM’2003*, San Francisco, CA, USA, April 2003.