Algorithmica (2005) 43: 17–41
DOI: 10.1007/s00453-005-1156-z

Algorithmica

Building Edge-Failure Resilient Networks[1]

Chandra Chekuri,[2] A. Gupta,[3] Amit Kumar,[4] J. Naor,[5] and Danny Raz[5]

Abstract. We consider the design of resilient networks that are fault tolerant against link failures. Resilience against link failures can be built into the network by providing *backup paths*, which are used in the eventuality of an edge failure occurring on a primary path in the network. We consider several network design problems in this context; these problems are motivated by the requirements of current high-speed optical networks. In all the following problems the objective is to provide resilience in networks while minimizing the cost incurred.

The main problem under consideration in this paper is that of *backup allocation*: this problem takes as its input an already provisioned *primary* network and a parameter k, and allocates backup capacity on the edges of the underlying network so that all the demand can be routed even in the presence of k edge failures. We also consider a variant of this problem where the primary network has a tree topology, and it is required that the restored network retains a tree topology.

We then address the problem of simultaneous *primary and backup allocation*: we are given specifications of the traffic to be handled, and the goal is to provision both the primary as well as the backup network. Finally, we investigate a single-commodity problem motivated by a pragmatic scenario in which the primary network is not known in advance and demands between source–sink pairs arrive online.

Key Words. Network design, Link failure, Backup path, Restoration, Approximation algorithm.

1. Introduction.

Fault tolerance in networks is an important and well-studied topic with many applications. Telephone networks and other proprietary networks adopt a variety of techniques to provide reliability and resilience to network failures and have been in use for many years now. On the other hand data networks such as the Internet have very little centralized fault tolerance. Instead, the network relies on the routing protocols that adapt to failures by sending traffic on alternate paths. This has been acceptable till now, since there have been no guarantees on the quality of service (QoS) on the Internet. However, with the maturity of the Internet, an increasing number of applications now

[1] An extended abstract of this work appeared in the *Proceedings of the* 9*th Integer Programming and Combinatorial Optimization Conference*, Cambridge, Massachusetts, 2002. Much of this work was done while the authors were at Lucent Bell Labs, 600 Mountain Avenue, Murray Hill, NJ 07974, USA. This research was partly done while A. Kumar was at Cornell University where he was supported in part by an ONR Young Investigator Award of Jon Kleinberg. The research of J. Naor and D. Raz was supported in part by US–Israel BSF Grant 2002276, and by a foundational and strategical research grant from the Israeli Ministry of Science.

[2] Bell Labs, Lucent Technologies, 600–700 Mountain Avenue, Murray Hill, NJ 07974, USA. chekuri@research.bell-labs.com.

[3] Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA. anupamg@cs.cmu.edu.

[4] Department of Computer Science, Indian Institute of Technology, Hauz Khas, New Delhi, India 110016. amitk@cse.iitd.ernet.in.

[5] Computer Science Department, Technion, Israel Institute of Technology, Haifa 32000, Israel. {naor,danny}@cs.technion.ac.il.

Received January 2004; revised December 2004. Communicated by L. Zhang.
Online publication June 30, 2005.

require QoS guarantees. The emergence of very high capacity optical networks has enabled the move towards providing users with their own *virtual private networks* or VPNs [31]. Several virtual networks can be accommodated on the underlying (high capacity) optical network by splitting the available bandwidth among them. However, since VPNs also require us to provide QoS guarantees to applications and users, fault tolerance and network resilience become very critical issues: failure of a single high capacity link can disrupt all the VPNs that use that link.

In many cases the sheer speed and the large capacity of the links does not permit us to rely merely on the routing protocols (say, OSPF or BGP) to reroute the traffic successfully on alternate routes *after* the failure; it is now imperative to a priori provision the network to handle failures. This places two broad constraints on these networks: (1) resources for rerouting traffic should be reserved at the same time the subnetworks are provisioned, and (2) the routing protocol should be simple, both for regular routing and when a fault occurs. (See, e.g., [34] and [14] for similar survivability issues of IP over optical networks, and [24] and [25] for discussions of fast restoration of the MPLS tunnels that are often used to implement VPNs in optical networks [10].)

In light of the importance of the problem, there has been some recent interest in obtaining algorithmic solutions for problems of guaranteeing resilience against failures. A variety of failure and recovery models have been proposed, and it is not feasible to give even an overview of all the models and their intricacies; the interested reader is referred to Section 1.2 for some pointers to related work in the literature. At a high level, this paper focuses on the cases where it is guaranteed that no more than a fixed number of edge failures can occur, and the algorithm must provision a minimum-cost network to allow for "local" restoration. We expand on these assumptions in the following paragraphs; a formal specification of the model is provided in Section 1.1.

Our model of edge failures is *adversarial*; we make the assumption that there is some fixed value k such that only k edge failures can happen in the network at any given instant of time. This should be contrasted with probabilistic models, where edges are allowed to fail with some specified probabilities. This assumption is commonly used in practice and seems to work reasonably well [27]. Another pragmatic reason for this assumption is that most networks are k-connected for some small k, and hence they cannot tolerate more than k adversarial edge failures. Furthermore, it is interesting to note that the resulting optimization problems are already hard for the case of $k = 1$. In the following discussion we usually restrict our attention to the *single edge failure* case of $k = 1$; where appropriate, we indicate how the ideas for $k = 1$ extend to general k. Note that for $k \geq 2$, both primary edges as well as backup edges are allowed to fail.

Resilience against single edge failures can be built into the network by providing for each edge e, a backup path $P(e)$, which is used when the edge e fails. However, since only one edge is guaranteed to fail, making the backup paths for two *different* edges intersect each other and *share* the same amount of bandwidth results in backup networks of lower cost. This multiplexing is one of the factors that makes this problem especially difficult; we spell out some of the others as we explain the models and our results.

In all our models we assume that we are provisioning bandwidth on a given *uncapacitated* underlying network, called the *base network*. In other words, we assume that there is unlimited capacity on each edge of the base network; for each edge e, there is an associated cost of c_e per unit of bandwidth, and we can allocate any amount x_e of bandwidth

on it by paying $c_e x_e$. This assumption is clearly not true for any practical network, but we make it for two reasons. First, we believe it to be a reasonable approximation since the capacities of the underlying network are usually much larger than the capacity of any *single* VPN that is to be provisioned. Second, the capacitated versions of the problems are provably much harder, and we believe that the domain in which they are hard does not apply to real settings. For example, the disjoint paths problem is notoriously hard for small capacities, but it is much easier if the capacities of the edges are sufficiently large compared with the individual demands. Similar assumptions have often been made in the literature; see, e.g., [7], [12], and [22].

We consider several network design problems with the above assumptions. The first general problem we consider is that of **Backup Allocation**. In this problem we are given an already *provisioned* (*primary*) *network*, and we want to reserve *backup* capacity on the edges of the base network so that traffic can be routed in the primary network even in the case of an edge failure. At this point, we point out the final requirement of the network: the restoration has to be handled *locally*; i.e., if edge $e = (i, j)$ carrying $u(e)$ bandwidth fails, there must be a *single* path $P(e)$ in the backup network between i and j with capacity at least $u(e)$, which stands in for the edge e.

Local restoration is important for timing guarantees. Global or end-to-end restoration could require portions of the network far away from the endpoints of e to be aware of a failure at e. Another advantage of local restoration is simplicity of the recovery mechanism. It is imperative for our purposes that there is a *single path* between u and v that routes all of $u(e)$; merely having a backup network that is able to push the right amount of "flow" is not sufficient. This is necessary in optical networks, where splitting the traffic is not feasible. As an aside, this local restoration can be very easily implemented using MPLS, and is sketched in Section 1.1. The reader curious about MPLS and the details of the mechanisms used for efficient local restoration of paths, can refer to the relevant literature [10], [28], [29].

The second problem we consider is that of simultaneous **Primary and Backup Allocation**. We look at both offline and online settings of the problem. In the offline case we are given specifications of the traffic to be handled, and we want to provision both the primary network as well as the backup network. In the online version of the problem, demands arrive one by one. On the arrival of a pair of terminals $\{s, t\}$, we must find both a primary path and a backup path between them. Again, the online algorithm endeavors to multiplex as many backup paths as possible, and models this by allowing different costs for an edge depending on whether it is a part of a primary or a backup path.

Details of these problems are given in the next section. Though these problems have some similarities to traditional network design problems, they also differ in some salient respects. Our contributions include providing formal models and building upon existing techniques to give algorithms for these new problems. We hope that our techniques and ideas will be useful in related contexts.

1.1. *Models and Results.* We now give detailed and precise formulations of the problems studied and results obtained in this paper. In all these problems we look at undirected *base networks* $G = (V, E)$ with edge costs c_e per unit bandwidth. Recall that this is an uncapacitated network, and any amount of bandwidth x_e may be allocated on e at cost $c_e x_e$.

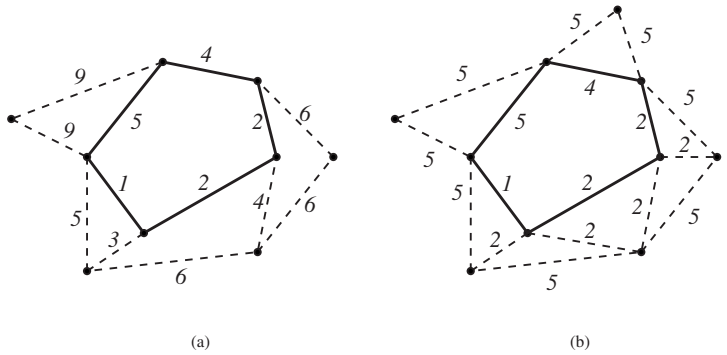

Fig. 1. Backup allocation: backup network indicated in dashed lines. (a) Backup network can handle $k = 2$ edge failures *only* in the primary network. (b) Backup network can handle $k = 2$ edge failures either in the primary or the backup networks.

Backup allocation. In backup allocation we are given an already provisioned *primary* network $G^p = (V^p, E^p)$, with each edge $e \in E^p$ having provisioned capacity $u^p(e)$; we are also given an upper bound k on the number of edge failures. The objective is to find an edge set $E^b \subseteq E$ and backup capacities u^b for these edges, so that given any set $F \subseteq E^p \cup E^b$ of *failed edges* with $|F| \leq k$, for each edge $e = (u, v) \in F \cap E^p$, there is a path $P_F(e) \in E^b \backslash F$ between u and v. This path $P_F(e)$ can be used to restore e locally lest the set F fails.

Of course, the backup network must have enough capacity, and given any edge $e' \in E^b$, the total capacity used by the paths in $\{P_F(e) \mid e \in F \cap E^p\}$ is less than $u^b(e')$. That is, for all $e' \in E^b$ and $F \subseteq E^p \cup E^b$ s.t. $|F| \leq k$,

$$\sum_{e: e \in F \cap E^p, e' \in P_F(e)} u^p(e) \leq u^b(e').$$

For an example, see Figure 1; the solid lines are the primary network (with the capacities indicated), and the dashed lines are one possible backup for it. (We do not require that the set of edges in the backup and primary networks be disjoint, and $E^b \cap E^p$ could certainly be non-empty.) Also note that in our model both backup and primary edges can fail. Figure 1(a) would not be a valid solution in our model, since some of the vertices on the cycle induced by the primary edges have only one backup edge adjacent to them. If, for such a vertex, both a primary edge and the backup edge adjacent to it fail, then the connection cannot be restored.

In Section 3 we describe an $O(1)$ approximation algorithm for the backup problem when $k = 1$. When then extend this to give an $O(k)$ approximation algorithm for $k > 1$. We first examine the *uniform capacity* case, i.e., when $u^p(e) = 1$ for all $e \in E^p$. This special case is similar to the Steiner network problem [16], [23], [33] where the goal is to design a minimum cost network with prescribed connectivity requirements for vertex pairs; the uniform capacity case has an additional constraint. In Section 2 we describe an algorithm to handle this uniform capacity case. In Section 3 we extend the algorithm to handle non-uniform capacities via capacity scaling.

Primary and backup allocation. In this problem we have to build both the primary network as well as the backup network. We require the specifications of the traffic that the primary network should be able to carry. A common model for specifying traffic requirement is the *point-to-point demand model*, where a *demand* matrix $D = (d_{ij})$ gives demands between each pair of terminals i and j; the objective is to output the cheapest network capable of carrying the traffic specified by D. In our setting where the base network is uncapacitated, the optimal primary network simply routes all the flow between u and v on a shortest path between the two terminals (with the "length" of an edge e being c_e).

Considering that good estimates are often not known for the pairwise demands in real networks, Duffield et al. [11] proposed an alternate way to specify traffic patterns, the so-called *VPN hose model*. In its simplest form, each terminal i is given a *threshold* $b(i)$, and a symmetric demand matrix $D = (d_{ij})$ is called *valid* if it respects all thresholds, i.e., if $\sum_j d_{ij} \leq b(i)$ for all i. The primary network is specified by two quantities: a vector u^p indicating the bandwidth allocated on the edges of the network, and for each pair of terminals ij a single path P_{ij} on which their flow will be routed. A feasible solution satisfies the following for *each* valid demand matrix $D = (d_{ij})$:

$$\sum_{i<j} d_{ij}\ \chi(P_{ij}) \leq u^p. \tag{1.1}$$

Here $\chi(P) \in \{0, 1\}^{|E|}$ is the characteristic vector of a path P, and the sum is a vector sum. Provisioning the primary network in the hose model has been studied by Gupta et al. [18], where among other results, an optimal algorithm is given when the provisioned network is required to be a tree; it is also shown that this tree provides a 2-approximation for the problem (without the tree restriction). An extension of the model includes the *asymmetric* case where both *ingress* and *egress* thresholds $b_{\text{in}}(i)$ and $b_{\text{out}}(i)$ are given, and a demand matrix is valid if $\sum_j d_{ij} \leq b_{\text{out}}(i)$ and $\sum_i d_{ij} \leq b_{\text{in}}(j)$; algorithms in this extended model that find near-optimal trees and general networks are presented in [18] and [20].

In this paper we study, based on the above, several models for designing primary and backup networks. Specific details of each model are provided in the relevant sections. We obtain the following results. We show that an α-approximation algorithm for allocating the primary network implies an $O(\alpha \log n)$ approximation for both primary and backup allocation. The simple two-stage algorithm for this first uses the α-approximation algorithm to allocate a primary network G^p, and then uses the algorithm of Section 3 to find a near-optimal backup network for G^p. We show that our analysis is tight, and give examples where $\alpha = 1$ but our algorithm outputs primary and backup networks costing $\Omega(\log n)$ times the optimum. Our algorithm works only for single edge failures ($k = 1$).

Tree networks. It is often desirable to provision networks having a tree topology; the advantages include simplicity, scalability, and the presence of good routing schemes for trees. This prompted Gupta et al. [18] to give algorithms for primary allocation in the VPN hose model which outputs the optimal tree, which they showed had a cost within a factor of 2 of the optimal network. Though we can always use the methods of Section 3 to add resilience to trees, when some edge e in a tree fails and is locally restored by $P(e)$, the new network may no longer be a tree. For some applications, and also for simplicity

of routing schemes, it is convenient that the network remains a tree at all times, even after restoration.

In Section 5 we study the problem of allocating backup to a given primary network T while ensuring that $T - e + P(e)$ is also a tree. We show that this problem is closely related to the group Steiner problem on trees [15]. From this, we show that it is hard to approximate within an $\Omega(\log^{2-\varepsilon} n)$ factor, and give a backup allocation algorithm with a pseudo-approximation guarantee of $O(\log^2 n)$.

The single commodity problem. In practical applications, the demands often appear in an online manner, i.e., new demands for resilient paths (i.e., primary and backup paths) between pairs of nodes arrive one by one. We consider the case where a demand consists of a pair of vertices, the source s and sink t, with a specified demand d to be sent between them; the goal is to construct a primary path P and a set of backup edges Q that can be used for restoration when an edge on P fails. As explained before, a backup edge can be used to back up several primary edges, and hence the edges which lie on previous backup paths may have already been paid for. We model this eventuality by allowing different *primary costs* and *backup costs* for an edge, depending on the purpose for which we will use this edge; clearly, the primary cost of an edge should be at least as large as the backup cost. We present a simple 2-approximation algorithm for the resulting problem. Our 2-approximation extends to the case of protection against k-failures. We also give two natural linear programming formulations of the problem, and show that one of these formulations dominates the other for all instances. We point out that we are considering the local optimization problem that needs to be solved each time a new demand arrives; our aim is not to perform the usual competitive analysis where the online algorithm is compared with the best offline solution.

MPLS Implementation. For the reader familiar with MPLS routing, we very briefly sketch the fact that the local restoration for the edge $e = (u, v)$ can be very easily implemented. (Readers curious about MPLS are referred to [10]; a theoretical model is given in [19].) Node u just precomputes the MPLS stack $S^b_{u,v}$ that routes to v on the backup path $P(e)$. (Symmetrically, node v has a stack $S^b_{v,u}$ to route to v.) In case the edge e fails and u receives a packet desirous of taking the edge e, it could push the stack $S^b_{u,v}$ onto the packet's MPLS stack and send it on the first vertex on $P(e)$.

1.2. *Related Work.* There have been several papers on (splittable) flow networks resilient to edge failures; see, e.g., [7], [8], and [12]. Papers [24] and [25] formulate the online restoration problem as an integer program, and give some empirical evidence in favor of their methods. Paper [22] considers backup allocation in the VPN hose model and gives a constant-factor approximation when accounting *only* for the cost of edges not used in the primary network and hence does not provide a true approximation algorithm. Further, [22] assumes that the primary network is a tree and hence the algorithm does not generalize to arbitrary networks. Paper [6] looks at the problem of limited-delay restoration; however, it does not consider the question of bandwidth allocation and its cost.

The problem of survivable network design has also been investigated extensively (see, e.g., [2] and the references therein). Most of this work has been focused on obtaining strong relaxations to be used in cutting plane methods. In fact, the linear programs

we use have been studied in these contexts, and have been found to give good empirical performance. For more details on these, and on polyhedral results related to them, see [3]–[5] and [9]. In contrast to most of these papers, we focus on worst-case approximation guarantees, and our results perhaps explain the good empirical performance of relaxations considered in the literature. Our models and assumptions also differ in some way from those in the literature. We are interested in local restoration, and not necessarily in end-to-end restoration. This allows our results to be applicable to the VPN hose model as well, in contrast to the earlier literature, which is concerned primarily with the point-to-point model. We also focus on path restoration as opposed to flow restoration. On the other hand, we do consider a simpler model and limit ourselves only to the case of uncapacitated networks.

1.3. *Paper Organization*. The rest of the paper loosely follows the structure of the Introduction: in Section 2 we define and study the **Constrained Steiner Network Problem**, giving a constant-factor approximation for the problem. This algorithm is crucially used in the algorithm for backup allocation presented in Section 3. We then go on to the problem of simultaneous primary and backup allocation in Section 4, for which we give an $O(\log n)$ approximation algorithm. Section 5 is devoted to the case of backup provisioning on tree topologies (where the restored network must also be a tree). Finally, we conclude with the single commodity backup problem in Section 6.

Throughout the paper we use OPT to denote the value of an optimal solution to the problem being discussed.

2. Constrained Steiner Network Problem. Given a primary network $G^p = (V^p, E^p)$ in the backup allocation problem with $k = 1$, the objective is to reserve capacity on a set of edges E^b and specify paths $P(e) \subseteq E^b$ for each edge $e \in E^p$ such that the backup path $P(e)$ must not contain e itself; furthermore, (1.1) requires that the capacity of any edge on $P(e)$ must be at least $u^p(e)$.

Consider a further simplification of the problem where $u^p(e) = 1$ for all $e \in E^p$; the problem is now to find a minimum cost set of edges E^b which, for each $e = (u, v) \in E^p$, contains a path $P(e) \subseteq E^b \backslash \{e\}$ between the endpoints u and v of e.

Dropping yet another constraint, we can attempt to find a minimum cost set of edges E' which, for all $e = (u, v) \in E^p$, contains a path $P(e) \subseteq E^b$ between the u and v. (Note that now $P(e)$ may be just the edge e itself.) This turns out to be a special case of a well-studied network design problem, the *Steiner Network Design* (SND) *problem*. Formally, an instance of SND is given by an undirected graph $G = (V, E)$, a cost function $c : E \to \mathbb{R}^+$, and *requirements* $r_{ij} \in \mathbb{Z}^+$ for pairs of vertices $(i, j) \in V$. (We can assume that $r_{ij} = 0$ for pairs (i, j) for which there is no requirement.) The goal is to select a minimum cost set of edges $E' \subseteq E$ such that there are r_{ij} edge-disjoint paths between i and j in E'. A 2-approximation for this problem was given by Jain [23].

We now define the *Constrained Steiner Network Design problem*, or the CSND problem, as the SND problem with the added constraint that r_{ij} edge-disjoint paths in E' between i and j must not contain the edge (i, j). (Note that (i, j) could still lie in E' and be used to connect some other pair (i', j').) The $k = 1, u^p = 1$ case of the backup allocation problem now just corresponds to setting $r_{ij} = 1 \iff \{i, j\} \in E^p$.

2.1. *An Approximation Algorithm for CSND.* We show that an α-approximation algorithm for SND can be used to obtain a 2α-approximation algorithm for CSND. The algorithm is simple and is given below.

- Let I_1 be the instance of SND with requirement r on G. Solve I_1 approximately, and let E' be the set of edges chosen.
- Define a new requirement function r' as follows. For $(i, j) \in E'$ such that $r_{ij} > 0$, set $r'_{ij} = r_{ij} + 1$, else set $r'_{ij} = r_{ij}$.
- Let I_2 be the instance of SND on G with requirement function r' and with the cost of edges in E' reduced to zero. Let E'' be an approximate solution to I_2. Output $E'' \cup E'$.

It is easy to see that the above algorithm produces a feasible solution. Indeed, if $(i, j) \notin E'$ then $E' - \{(i, j)\}$ contains r_{ij} edge-disjoint paths between i and j. If $(i, j) \in E'$ then E'' contains $r_{ij} + 1$ edge-disjoint paths between (i, j), and hence $E'' - \{(i, j)\}$ contains r_{ij} edge-disjoint paths.

LEMMA 2.1. *The cost of the solution to CSND produced by the above algorithm is at most* $2\alpha\mathsf{OPT}$, *where* α *is the approximation ratio of the algorithm used to solve SND.*

PROOF. It is easy to see that $\mathsf{OPT}(I_1) \leq \mathsf{OPT}$, and hence $c(E') \leq \alpha\mathsf{OPT}$. We claim that $\mathsf{OPT}(I_2) \leq \mathsf{OPT}$. Indeed, if $A \subseteq E$ is an optimal solution to I, then $A \cup E'$ is feasible for requirements r'. Therefore, $c(E'' - E') \leq \alpha\mathsf{OPT}(I_2) \leq \alpha\mathsf{OPT}$, and $c(E'' \cup E') \leq 2\alpha\mathsf{OPT}$. □

Using the 2-approximation algorithm for SND due to Jain [23] gives us the following corollary.

COROLLARY 2.2. *There is a* 4*-approximation algorithm for the CSND problem.*

In Section 3 we build on this algorithm to give approximation algorithms for the general backup allocation problem.

2.2. *Integrality Gap of an LP Relaxation for CSND.* While our algorithm for CSND used the algorithm for SND as a black box, it will be useful for us to write down a natural linear programming (LP) relaxation of the CSND problem, and consider its integrality gap. This will be useful in Section 4. Consider the following integer linear programming formulation for CSND, where x_e is the indicator variable for picking edge e in the solution. For compactness we use the following notation to describe the constraints. We say that a function $\bar{x}$ on the edges *supports* a flow of f between s and t if the maximum flow between s and t in the graph with capacities on the edges given by $\bar{x}$ is at least f. This property can be easily modeled by linear constraints:

$$\begin{array}{lll} \text{(IP1)} \quad \min & \displaystyle\sum_e c_e x_e & \\ \text{s.t.} & \bar{x} \text{ supports } r_{ij} \text{ flow between } (i, j) \text{ in } E - \{(i, j)\} & \text{for all } i, j, \\ & x_e \in \{0, 1\} & \text{for all } \quad e \in E. \end{array}$$

We relax the integrality constraints to obtain the following linear program:

$$\begin{array}{lllll} \text{(LP1)} & \min & \sum_e c_e x_e & & \\ & \text{s.t.} & \bar{x} \text{ supports } r_{ij} \text{ flow between } (i,j) \text{ in } E-\{(i,j)\}, & \text{for all } i,j, \\ & x_e & \in [0,1] & \text{for all} & e \in E. \end{array}$$

LEMMA 2.3. *The integrality gap of* (LP1) *for CSND is upper bounded by* 4.

PROOF. Consider the following LP formulation (LP-flow) for SND:

$$\begin{array}{lllll} \text{(LP-flow)} & \min & \sum_e c_e x_e & & \\ & \text{s.t.} & \bar{x} \text{ supports } r_{ij} \text{ flow between } (i,j) \text{ in } E & \text{for all } \{i,j\}, \\ & x_e & \in [0,1] & \text{for all} & e \in E. \end{array}$$

Jain's result in [23] shows that the integrality gap of (LP-flow) is at most 2 for SND. Note that the optimal solution to (LP-flow) for either of the instances I_1 and I_2 created in our algorithm for CSND costs no more than an optimal solution to (LP1) for I; indeed, the solution to (LP1) for I is a feasible solution to both those LPs. This, combined with the fact that (LP-flow) has an integrality gap of at most 2, gives us the claimed result. □

3. Backup Allocation. We now use the algorithm for CSND from Section 2 to show an $O(k)$ approximation for the problem of computing the cheapest backup network for a given primary network. Let $G = (V, E)$ be the underlying base network, and let $G^p = (V^p, E^p)$ be the primary network. We are also given the primary edge capacities $u^p : E^p \to \mathbb{R}^+$. Our goal is to find an edge set $E^b \subseteq E$ (the backup edges), and a function $u^b : E^b \to \mathbb{R}^+$ (the backup bandwidth), and backup paths $P(e)$ for every edge $e \in E^p$ satisfying (1.1) for any set of at most k edge failures F^p.

We first state the quality of approximation we can obtain for the uniform capacity problem.

LEMMA 3.1. *For the uniform capacity backup allocation problem with k failures, there is a* $4k$*-approximation algorithm.*

PROOF. By scaling capacities we can assume without loss of generality that the primary capacity of each edge is 1. We solve the CSND problem induced by the backup problem: for each edge (i, j) in the primary network we set $r_{ij} = k$. The cost of the CSND problem is no more than four times the value of an optimum solution to the backup problem. On each edge e in the solution to the CSND problem we place a capacity of k. This increases the cost of the solution by a factor of k. The total flow on any backup edge is at most k since this is the maximum number of edges that can fail. Thus, the modified solution is feasible for the backup allocation problem. The cost is clearly within $4k$ times the optimum cost. □

Let $u^p_{\max} = \max_{e \in E^p} u^p(e)$. Our algorithm for backup allocation given below is based on scaling the capacities and solving the resulting uniform capacity problems separately:

- Let $E^p_i = \{e \in E^p \mid u^p(e) \in [2^i, 2^{i+1})\}$. For all $e \in E^p_i$, round up $u^p(e)$ to 2^{i+1}.
- For $1 \leq i \leq \lceil \log u^p_{\max} \rceil$, *independently* backup E^p_i.

Let E^b_i be the edges for backing up E^p_i and let u^b_i be the backup bandwidth on E^b_i. Note that rounding up the bandwidths of E^p_i causes the the backup allocation problem on E^p_i to be a uniform problem. The lemma below states that solving the problems separately does not cost much in the approximation ratio.

LEMMA 3.2. *There is an approximation algorithm for the backup allocation problem with ratio* $16k$.

PROOF. Let E^{r*} be an optimal solution for backup allocation, with u^{r*} being the bandwidth allocation function on E^{r*}. For $0 \leq i \leq \lceil \log u^p_{\max} \rceil$ construct solutions E^{r*}_i, where $e \in E^{r*}_i$ with capacity $u^{r*}_i(e) = 2^{i+1}$ if $u^{r*}(e) \geq 2^i$, and 0 otherwise. Clearly, $\sum_i u^{r*}_i(e) \leq 4u^{r*}(e)$ for each e, and hence by linearity of the cost function, $\sum_i c(E^{r*}_i) \leq 4c(E^{r*})$. Note that E^{r*}_i is a feasible solution to the CSND problem induced by the backup for E^p_i. Hence, for each i, using the approximation algorithm for the uniform case for E^p_i as described in the proof of Lemma 3.1, we obtain a solution of cost at most $4kc(E^{r*}_i)$. This completes the proof. □

The ratio of $16k$ in Lemma 3.2 can be further improved to $4ek$ by randomness: instead of grouping by powers of 2, grouping can be done by powers of e (with a randomly chosen starting point). This technique is fairly standard by now (e.g., [26] and [17]), but we give the proof for sake of completeness.

THEOREM 3.3. *Given a primary network, there is a* $4k \cdot e \simeq 10.87k$*-approximation for the* k*-failure resilient backup allocation problem with linear edge cost functions.*

PROOF. We pick a real number α from the interval $[1, e)$ randomly according to the density function $f(t) = 1/t$. Here e is the base of the natural logarithm. As before, E^p is the set of edges in the primary network, and $u^p(a)$ denotes the primary reservation on edge a. Now we group the edges in the following manner. Define E^p_i as the set of edges in E^p for which $u^p(a)$ lies in the interval $[\alpha \cdot e^i, \alpha \cdot e^{i+1})$. Round up $u^p(a)$ to $\alpha \cdot e^{i+1}$ for all edges in the set E^p_i. As before, we independently backup E^p_i for all i. Observe that the only difference from the earlier deterministic algorithm is in how we group the edges.

We now compute the approximation ratio of this algorithm. We follow the proof of Lemma 3.2. Fix an edge a. Let $g(a) = i$ if $u^{r*}(a)$ lies in the interval $[\alpha \cdot e^i, \alpha \cdot e^{i+1})$. Note that $g(a)$ is a random variable that takes one of two values, either $\lfloor \ln u^{r*}(a) \rfloor$ or $\lfloor \ln u^{r*}(a) \rfloor - 1$. Then it can be shown that the cost of our solution is at most $4k \cdot \sum_{a \in E} \alpha \cdot e^{g(a)+1} \cdot (e/(e-1))$. Here the factor $4k$ comes from the approximation algorithm for the uniform bandwidth case, the term $\alpha \cdot e^{g(a)+1}$ comes from rounding the reservations up, and the term $e/(e-1)$ comes from the geometric sum.

Now, a routine calculation shows that the expected value of $\alpha \cdot e^{g(a)+1}$ is $(e-1)u^{r*}(a)$. Thus, in expectation, we get an approximation ratio of $4k \cdot e$. The algorithm can be easily derandomized since there are $O(|E|)$ distinct values of α that are of interest and these can be easily enumerated. □

3.1. *Integrality Gap of an LP Relaxation.* We have shown an $O(k)$ approximation for the backup allocation problem. We now analyze the integrality gap of a natural LP relaxation for the problem and show that it is $O(k \log n)$. This will allow us to analyze an algorithm for simultaneous allocation of primary and backup networks in the next section. The LP formulation uses variables y_e which indicate the backup bandwidth bought on edge e. We relax the requirement that the flow uses k edge-disjoint paths. A k-flow between a pair of vertices s and t is a flow of k units from s to t such that the flow on any edge of the graph is at most 1.

$$\text{(LP2)} \qquad \min \quad \sum_e c_e y_e$$
$$\text{s.t.} \quad \bar{y}/u_e^p \text{ supports a } k\text{-flow between } (i, j) \text{ in } E-\{e\} \qquad \text{for all } \ e \in E^p,$$
$$y_e \geq 0.$$

We now analyze the integrality gap. Recall the definition of E_i^p as the set of edges in E^p such that $u^p(e) \in [2^i, 2^{i+1})$. As before we round up the bandwidth of these edges to 2^{i+1}. Let $x_e(i) = \min\{1, y_e/2^i\}$. Note that $x_e(i) \in [0, 1]$. We claim the following.

PROPOSITION 3.4. *The variables $x_e(i)$ are feasible for the uniform bandwidth backup allocation LP relaxation induced by E_i^p where the bandwidths are scaled to* 1.

From Lemma 3.1 it follows that the integrality gap of (LP1), the LP for the uniform bandwidth problem, is at most $4k$. Hence we can find a solution that backs up the edges in E_i^p with cost at most $4k \sum_e c_e y_e$. Since we only have to look at $\lceil \log u_{\max}^p \rceil$ values of i, there is a solution that backs up all edges in E^p with cost at most $4k \log u_{\max}^p \sum_e c_e y_e$. We can make the upper bound on the integrality gap $O(k \log n)$ via a simple argument. We set $x_e(i) = 0$ if $y_e/2^i \leq 1/n^3$, otherwise we set $x_e(i) = \min\{1, (1+1/n)y_e/2^i\}$. It is straightforward to argue that Proposition 3.4 still holds for the variables $x_e(i)$ defined in this modified fashion. The cost goes up by a $(1+1/n)$ factor. Each edge e participates in the backup of at most $O(\log n)$ groups E_i^p, hence the overall cost is at most $O(k \log n)$ times the LP cost. This gives us the following theorem.

THEOREM 3.5. *The integrality gap of* (LP2) *is* $O(k \min\{\log n, \log u_{\max}^p\})$.

The following theorem shows that our analysis is tight for $k = 1$.

THEOREM 3.6. *The integrality gap of* (LP2) *is* $\Omega(\log n)$ *for* $k = 1$.

PROOF. We construct a graph G with the required gap as follows. The graph consists of a complete binary tree T rooted at r with some additional edges. The cost of each edge in T is 1. The additional edges go from leaves to the root and each of them is of cost

d, where d is the depth of T. Primary bandwidth is provisioned only on the edges of T and is given by $u^p(e)$: for an edge e at depth $d(e)$, $u^p(e) = 2^d/2^{d(e)}$. Backup bandwidth allocation defined by the following function $u^b(e)$ is feasible for (LP2): $u^b(e) = 1$ for each edge e that goes from a leaf to the root and $u^b(e) = u^p(e)$ for each edge of T. It is easy to check that the cost of this solution is $O(d2^d)$.

We claim that any path solution to the backup of T in G has a cost of $\Omega(d^2 2^d)$. Setting d to be $\log n$ gives the desired bound on the integrality gap. We now prove the claim. Let E_i be the edges of T at depth i. Let c_i be the minimum cost of backing up edges in E_i. We first note that $\sum_i c_i \leq 4\,\mathsf{OPT}$. This follows from a simple scaling argument that we used in the proof of Lemma 3.2. Thus, it is sufficient to prove that $c_i \geq d2^d$. Let e be an edge in E_i and let T_e be the subtree rooted under e. We have that $u^p(e) = 2^{d-i}$. The backup solution for E_i requires an edge from a leaf in T_e to the root with capacity $u^p(e)$. Note that for any $e, e' \in E_i$, the trees T_e and $T_{e'}$ are disjoint. Therefore $c_i \geq |E_i| d 2^{d-i} = d2^d$. □

We note that the primary network in the above proof is a feasible primary network for an instance in the point-to-point demand model as well as for an instance in the VPN hose model. Correctness of the former is clear: every edge implicitly defines a point-to-point demand between its endpoints of value equal to the primary bandwidth allocated to the edge. To see that the above primary network is feasible for an instance in the VPN hose model, consider the leaves of T as demand points, each with a bandwidth bound of 1.

3.2. *Concave Capacity Costs.* We have assumed till now that the cost per bandwidth on each edge is proportional to the bandwidth. We now demonstrate the applicability of our ideas to the case where the cost is a *concave* function of the capacity. Network design problems with concave cost functions are also referred to as *buy-at-bulk* problems [30], [1]. For each edge e in G we let $c_e : \mathcal{R} \to \mathcal{R}$ denote the concave function that defines the cost per capacity on e. The algorithm we use is the same as the one for linear capacities that we described above, the only difference is in the analysis and the performance guarantee. We first claim that Lemma 3.1 is also valid for the concave cost functions. Lemma 3.2 requires a modification as given below.

LEMMA 3.7. *Let α be the approximation ratio for the uniform capacity backup allocation problem. Then there is an approximation algorithm for the backup allocation problem with concave costs with ratio $O(\alpha \log u^p_{\max})$.*

PROOF. The proof is similar to that of Lemma 3.2. Let E^{r*} be an optimal solution for backup allocation, with u^{r*} being the bandwidth allocation function on E^{r*}. For $1 \leq i \leq \lceil \log u^p_{\max} \rceil$ construct solutions E^{r*}_i, where $e \in E^{r*}_i$ with capacity $u^{r*}_i(e) = 2^{i+1}$ if $u^{r*}(e) \geq 2^i$, and 0 otherwise. By concavity of the cost function, $c_e(u^{r*}_i(e)) \leq 2c_e(u^{r*}(e))$, hence $\sum_i c_e(u^{r*}_i(e)) \leq 2c_e(u^{r*}_i(e)) \log u^{r^*}_e(e)$. Therefore $\sum_i c(E^{r*}_i) \leq 2c(E^{r*}) \log u^p_{\max}$. Note that E^{r*}_i is a feasible backup for E^p_i, since every edge in E^{r*} of bandwidth at least 2^i lies in E^{r*}_i with bandwidth 2^{i+1}. Hence, for each i, using the approximation algorithm for the uniform case for E^p_i would give us a solution with cost at most $\alpha c(E^{r*}_i)$. Therefore the algorithm outputs a solution of cost no more than $O(\alpha \log u^p_{\max})$. □

From the above lemma we obtain the following.

THEOREM 3.8. *Given a primary network, there is an* $O(k \log u^p_{\max})$*-approximation for the k-failure resilient backup allocation problem with concave edge cost functions.*

4. Simultaneous Primary and Backup Allocation. In this section we examine the problem of simultaneously building a primary network as well as the backup network so as to minimize the overall cost. In this section we restrict ourselves to single edge failures. We have a constant-factor approximation for backup allocation when given the primary network. We use it in a natural way to provision both the primary and backup. We adopt the two-phase strategy of first building the primary network, and then building a backup network for it. If α is the approximation guarantee for the problem of building the primary network, we obtain an $O(\alpha \log n)$ approximation algorithm for the problem of primary and backup allocation. This result applies when the primary network has to support a set of point-to-point demand matrices. The set of point-to-point demands can be explicitly specified or implicitly specified as in the VPN hose model.

For the two models we use to specify the primary bandwidth requirements, namely the point-to-point demand model and the VPN hose model, we have constant-factor approximation algorithms for building the primary network. For the point-to-point model an optimal solution is obtained by shortest path routing and for the VPN model a constant factor approximation is given in [18]. We thus obtain $O(\log n)$ approximations for the combined problem for these two models.

4.1. *The* $O(\log n)$ *Approximation Algorithm.* We analyze the two-stage approach for primary and backup allocation. Let G^p be the subgraph of G that is chosen in this first step. We provide backup for this network using the algorithm described in Section 3. To analyze this algorithm we use the LP relaxation (LP2) for the backup allocation problem. In the following lemma we will be using extra capacity on the edges of the provisioned network itself. Note that this is allowed. We call a primary solution u^p *minimal*, if for any edge e with $u^p(e) > 0$, we cannot reduce $u^p(e)$ by even an arbitrarily small $\varepsilon > 0$ without violating the feasibility of u^p. It follows that for a given minimal primary network and an edge e with $u^p(e) > 0$, there is a demand matrix D_e for which e is *critical*; that is, reducing $u^p(e)$ would imply that D_e cannot be routed.

LEMMA 4.1. *Let* u^p *be any minimal solution to the primary problem. Let* u^{p*} *and* u^{r*} *be the primary and backup in some optimal solution. Then* $u^p + u^{p*} + u^{r*}$ *is a feasible solution for* (LP2)*, the LP relaxation for the backup of* u^p*.*

PROOF. Let $e = (i, j)$ be such that $u^p(e) > 0$. Let D_e be the demand matrix for which e is critical. Let f be a feasible multicommodity flow routing for D_e in u^p. It follows that $f(e) = u^p(e)$ for otherwise e is not critical for D_e. Let the flow paths that use e in a flow decomposition for f be $P_1, P_2, \ldots, P_\ell$ and let f_i be the flow on P_i. For $1 \le h \le \ell$, let x_h and y_h be the endpoints of P_h such that i occurs before j in traversing P_h from x_h to y_h. Let G' be the capacitated graph obtained from G by removing e and setting capacities equal to $u^p + u^{p*} + u^{r*}$. We need to argue that G' can support a flow of $u^p(e)$

from i to j. We do this as follows. For $1 \leq h \leq \ell$, we simultaneously send f_h units of flow from i to x_h using a capacity of u^p. Let G'' be the capacitated graph obtained from G by removing e and setting capacities equal to $u^{p*} + u^{r*}$. Since the optimum solution is resilient against single edge failures, for $1 \leq h \leq \ell$, G'' can simultaneously support a flow of f_h units from x_h to y_h. Since $\sum_h f_h = f(e) = u^p(e)$, it follows that we can route a flow of $u^p(e)$ from i to j in G'. □

THEOREM 4.2. *The two-stage approach yields an $O(\alpha \log n)$ approximation to the combined primary and backup allocation problem where α is the approximation ratio for finding the primary allocation.*

PROOF. Let P be the cost of the primary allocation and let B be the cost of backup allocation in the two-stage approach. From the approximation guarantee on finding P, we have $P \leq \alpha \mathsf{OPT}$. From Lemma 4.1, it follows that there is a feasible (LP2) relaxation for the backup allocation problem of value at most $P + \mathsf{OPT}$, hence at most $(\alpha + 1)\mathsf{OPT}$. From Theorem 3.5, the backup solution we obtain is at most $O(\log n)$ times the LP value. Hence, $B = O(\alpha \log n)\mathsf{OPT}$ and the theorem follows. □

It turns out that the two-stage approach loses an $\Omega(\log n)$ factor even if the first step obtains a primary network of optimum cost; the example in the proof of Theorem 3.6 demonstrates this. We observe that the two-stage approach cannot guarantee a good approximation for the case of k edge failures when $k \geq 2$. We briefly describe an example to show this. The underlying graph G consists of two vertices s, t with two parallel paths P_1 and P_2 between them. Each path has $\ell \gg k$ edges on it. In addition, each edge e on the two paths has k parallel copies which we refer to as the auxiliary edges associated with e. The auxiliary edges for every $e \in P_1$ are "cheap" while the ones associated with every $e \in P_2$ are "expensive." However, the cost of P_1 itself is marginally more than that of P_2. Consider a point-to-point demand between s and t. In a two-stage process an algorithm for building a primary network might choose P_2 as a solution. For $k \geq 2$, if k edges in P_2 fail, then one of the expensive auxiliary edges on P_2 will be forced to be chosen in the second stage. An optimum solution consists of P_1 and its auxiliary edges. The gap between the two solutions can be made arbitrarily large by appropriately choosing the costs of the edges.

5. Backup for Tree Networks. In this section we consider the case when the provisioned network T is a tree, and, furthermore, it is required that, when an edge e fails, the network $T - \{e\} + P(e)$ also be a tree. The objective, as before, is to minimize the cost of allocating the backup bandwidth. We prove that this problem is at least as hard as the group Steiner problem [15] on trees, which in turn is $\Omega(\log^{2-\varepsilon} n)$-hard [21].

THEOREM 5.1. *The tree backup problem is at least as hard to approximate as the group Steiner problem on trees. Hence, for any fixed $\varepsilon > 0$, the problem cannot be approximated to within a ratio of $\Omega(\log^{2-\varepsilon})$ unless* NP $\subseteq$ ZTIME$(n^{\text{polylog}(n)})$.

PROOF. The group Steiner tree problem is given by a weighted graph $G_s = (V_s, E_s)$, a root vertex $r \in V_s$, and ℓ subsets $S_1, \ldots, S_\ell$ of vertices. A solution consists of a subtree $T_s \subseteq G_s$ containing the root r which intersects every set, i.e., $V(T_s) \cap S_i \neq \emptyset$ for $1 \leq i \leq \ell$. The objective is to find a tree T_s of minimum cost. We consider an instance when G_s is restricted to being a tree. Note that in this case, a solution can be completely specified by giving, for $1 \leq i \leq \ell$, a vertex $v_i \in S_i$.

We reduce the group Steiner problem on trees to the tree backup problem. The base network $G = (V, E)$ consists of G_s, and ℓ new vertices $u_1, \ldots, u_\ell$. Each u_i is connected to all the vertices in S_i, and also to r, by an edge e_i. Finally, for each edge $e \in E_s$, E contains a parallel copy of the edge e. All the edges in E_s have unit cost, while the new edges have zero cost.

Let $\delta < 1$ be some fixed constant. The primary network $G^p = (V, E^p)$ has the tree edges E_s along with $e_1, \ldots, e_\ell$; furthermore, we set $u^p(e) = \delta$ for $e \in E_s$, and $u^p(e_i) = 1$. Since each edge $e \in E_s$ can be protected by this parallel copy at zero cost, we need only protect the edges e_i. Since the restored network is to be a tree when e_i fails, this tree must be $T - e_i + f_i$, with f_i being an edge from u_i to some $w_i \in S_i$. Hence, for $1 \leq i \leq \ell$, we must reserve one unit of backup bandwidth on the path from the vertex w_i to r in T. This shows that the cost of the optimal backup is the same as the cost of an optimal solution to the group Steiner tree problem in G_s. Thus the tree backup problem is at least as hard as the group Steiner problem on trees which is hard to approximate to within a factor of $\Omega(\log^{2-\varepsilon} n)$ [21, Theorem 1.1]. □

We also give an algorithm for the tree problem when $k = 1$. Let $T = G^p = (V^p, E^p)$ the primary network, and let E^b be the backup edges, with u^p and u^b the primary and backup bandwidth allocations. Let α be the approximation ratio for group Steiner problem on trees. It is known that $\alpha = O(\log^2 n)$ [15]. Our algorithm outputs a solution to the backup problem whose cost is $O(\alpha) \sum_e c_e(u^p(e) + u^b(e))$. Note that this does not yield a true $O(\alpha)$ approximation algorithm since we are including the cost of the primary network in the upper bound. The algorithm and the proof of the upper bound can be found in Appendix A. We note that the proof of Theorem 5.1 shows (by choosing ε appropriately) that the problem remains hard even if the algorithm is allowed to compare its cost against $\sum_e c_e(u^p(e) + u^b(e))$.

6. Single Source and Sink. In this section we consider a unit-capacity MPLS primary and backup allocation problem which is motivated by the online problem of choosing the best primary and backup paths for demands arriving one by one. Suppose that we are given source and destination vertices, denoted by s and t, respectively. The goal is to provision a primary path p from s to t and a backup set of edges q of minimum overall cost simultaneously. Since we are dealing with a single source–sink pair we can scale the bandwidth requirement to 1, hence all edges have unit capacity, i.e., the primary and backup edge sets are disjoint. We require that for any failure of k edges, $e_1, \ldots, e_k$, $(q \cup p) - \{e_1, \ldots, e_k\}$ contains a path from s to t. We call this problem SSSPR (Single Source Sink Provisioning and Restoration). Note that this requirement is slightly different from the backup model discussed earlier in the paper; here, we do not insist on local restoration. The backup edges together with the primary edges are

required to provide connectivity from s to t. This problem is in the spirit of the work of Kodialam and Lakshman [24], [25]. As explained before, we model the online nature of the problem by using two different costs. Formally, there are two non-negative cost functions associated with the edges: c_1 gives the cost of an edge when used as a primary edge, and c_2 gives the cost of an edge when used as a backup edge. We assume that $c_1(e) \geq c_2(e)$ for all edges $e \in E$.

Let p be a primary path from the source s to the destination t. The following procedure due to Suurballe [32] computes a minimum cost backup set of edges for a given primary path p. The idea is to direct the edges on the path p in the "backward" direction, i.e., from t to s and set their cost to be zero. All other edges are replaced by two anti-symmetric arcs. The costs of the two arcs a and a^- that replace an edge e are set to $c_2(e)$. We now compute a shortest path q from s to t. It can be shown that the edges of q that do not belong to p define a minimum cost local backup [32].

We first present a 2-approximation algorithm for the SSSPR problem for the case $k = 1$, and then generalize it to arbitrary values of k. First, find a shortest path p from s to t with respect to the c_1-cost function. Then use Suurballe's [32] procedure to compute an optimal backup q to the path p with respect to the c_2-cost function. We show below that p and q together induce a 2-approximate solution.

THEOREM 6.1. *The two-stage approach yields a* 2*-approximation to SSSPR.*

PROOF. Let OPT be the cost of an optimal primary and backup solution, let $P = \sum_{e \in p} c_1(e)$ be the cost of p, and let $Q = \sum_{e \in q} c_2(e)$ be the cost of q. It is clear that $P \leq \mathsf{OPT}$ since we find the cheapest primary path. We next argue that $Q \leq \mathsf{OPT}$. Consider Suurballe's [32] algorithm to find the optimum backup path for p. As described earlier, the algorithm finds a shortest path in a directed graph obtained from G and p. The main observation here is that any primary path p' and a q' that backs up p' yields a path in the directed graph created by Surballe's procedure. We omit the (straightforward) formal proof of this observation. In particular, the observation holds for the set of edges of p^* and q^*, where p^* is an optimal primary path and q^* is a set of edges that backs up p^*. Surballe's procedure finds the shortest path and therefore, $Q \leq \sum_{e \in p^* \cup q^*} c_2(e) \leq \sum_{e \in p^*} c_1(e) + \sum_{e \in q^*} c_2(e) \leq \mathsf{OPT}$. Here we use the assumption that $c_2(e) \leq c_1(e)$ for all e. □

Although we provide an approximation algorithm, we note that it is not known whether SSSPR is NP-hard or not.

Extension to k failures. We note that the above algorithm can be easily extended to the case where at most k edges can fail. First, find a shortest path p from s to t with respect to the c_1-cost function. Then, similar to Suurballe's [32] procedure, direct the edges on the path p in the "backward" direction, i.e., from t to s and set their cost to be zero. All other edges are replaced by two anti-symmetric arcs. The costs of the two arcs a and a^- that replace an edge e are set to $c_2(e)$. We set the capacity of each arc to 1 and compute a minimum cost flow of k units from s to t. Let q be the support of the flow in G. We claim that the edges of q that do not belong to p define a minimum cost local backup which is resilient to k edge failures, since any cut separating s from t contains at least $k + 1$ edges from $p \cup q$.

We observe that p and q together induce a 2-approximate solution. Clearly, the cost of p is upper bounded by the cost of an optimal solution. As in the case where $k = 1$, any primary path p' and backup q' (resilient to k failures) define a flow of k units from s to t in the directed graph obtained from G and p (as above). Therefore, the cost of q is also upper bounded by the cost of an optimal solution, yielding that the solution computed is a 2-approximation.

In Appendix B we provide two linear programming formulations for SSSPR. Although we show that the worst-case integrality gap for both formulations is 2, we nevertheless believe that the formulations are of interest from the point of view of mathematical programming. They could be of potential use in a branch and bound scheme to obtain good solutions in practice.

7. Conclusions. In this paper we explored models and algorithms for designing networks that are resilient to failures. Our focus was on local restoration and on uncapacitated networks. With these two assumptions we were able to provide good approximation algorithms under a fairly general model. Our main result on providing backup networks can be extended to capacitated networks, however, the algorithm might violate the capacities by $u^p_{\max}$. Can we obtain a constant factor approximation without violating the capacities? Can we obtain improved algorithms for designing primary and backup networks simultaneously? Can we obtain provably good algorithms in the end-to-end restoration model? We leave these for future work.

Acknowledgments. We thank Rajeev Rastogi for providing us with a copy of [22], and for useful discussions. We are grateful to Bruce Shepherd for sharing his knowledge of the area with us, his enthusiasm for our work, and for pointing out relevant literature. We thank Sudipto Guha for discussions that led to the integrality gap example in Section 6. We also thank one of the anonymous reviewers for extensive comments that improved the presentation of the paper.

Appendix A. Algorithm for Backup for Tree Networks. Let $T = (V, E)$ be the already provisioned tree. When an edge e fails, it splits T into two components, and $P(e)$ must be a path between these two components which is internally node disjoint from the tree T. We must reserve enough bandwidth on the edges in the graph such that the tree formed thus can support traffic between the demand nodes.

Our basic strategy is the same as in Section 3: Let E_i be the set of edges in T on which the bandwidth u^p lies in the interval $[2^i, 2^{i+1})$. Let $u^p_{\max}$ lie in the interval $[2^s, 2^{s+1})$. Our algorithm proceeds in stages—in the ith stage, we "protect" the edges in E_{s+1-i}. When we have already protected the edges in $E_{i+1}, \cdots, E_s$ by reserving bandwidth on some edges, we contract edges in $E_{i+1} \cup \cdots \cup E_s$. This will not affect our performance by more than a constant, since the bandwidth we may later reserve on some edge e in this set will be at most $\sum_{j \le i} 2^{i+1} \le 4u^p(e)$. Let T_i and G_i be the resulting tree and base graph after the contraction. We now consider protecting the edges in E_i, using the edges of G_i. The root r_i of G_i is the node which contains the root r of the original graph.

The algorithm has a few conceptual steps, which we proceed to describe next.

Structure of E_i. It can be seen that the edges of E_i form a "spider"; i.e., there is a root r, and a collection of paths $\{P_i\}_{i=1}^{k}$ which meet at r but are otherwise node-disjoint. This is because of the structure of the VPN trees as given in [18].

LEMMA A.1. *The edges of E_i form a spider in G_i.*

PROOF. As is shown in [18], the reservation $u^p(e)$ on an edge e is the total demand of the set of nodes in the component of $T_i - e$ which does not contain the root r_i. Therefore, the allocated bandwidth on the edges from the root to the leaves is non-increasing. This shows that E_i is a connected subtree of T_i. Suppose a non-root node $v \in T_i$ has two children u and w such that (v, u) and (v, w) are in E_i. Let the parent of v be $p(v)$. Clearly, $e = (v, p(v)) \in E_i$ as well. However, the reservation on e is at least the sum of the reservations on (v, u) and (v, w). So, $u^p(e) \geq 2^{i+1}$, which is a contradiction. This proves the lemma. □

Transforming G_i. Now we transform G_i into a graph G' such that $T_i T_i$ is a spanning tree of G'_i. All non-T_i edges in G'_i go between vertices of T_i. This can be done so that the backup solutions for T_i in G_i and in G'_i can be translated between each other with only a constant factor difference in cost. We prove this fact now.

LEMMA A.2. *We can transform G_i into a graph G'_i in polynomial time such that the vertex set of G'_i is same as that of T_i and T_i is a spanning tree of G'_i. Further, we have the following properties*:

- *Any backup solution for T_i in G'_i can be transformed to that in G_i without any increase in cost.*
- *Any backup solution for T_i in G_i can be transformed to that in G'_i with a constant factor loss in cost.*

PROOF. We construct the graph G'_i as follows. G'_i contains T_i. For every pair of vertices u, v in T_i, we add an edge e_{uv} between them. The cost of e_{uv} is the cost of the shortest path in G_i between u and v which does not have an internal vertex of T_i.

We now use the standard Eulerian tour idea to show equivalence between solutions in the two graphs. Indeed, a simple proof will go along the following lines. Consider the backup edges for E_i which do not belong to T_i; these must form a tree (else we could drop one of the edges). Take an Eulerian tour of this tree and consider the subpaths between consecutive nodes in T_i; these can be replaced by new edges with the appropriate weight, and all other vertices can be disposed off. Note that the optimal solution in this new instance will be at most twice the optimal solution before, since the Eulerian tour counted every edge twice.

However, we need to be careful about one fact—any solution (in G_i or G'_i) which protects the edges in T_i may need to reserve some bandwidth on the edges of the tree T_i itself. So, we must show that when we transform a solution from one graph to the other, the reservation on the tree edges need not change. This makes the proof more involved.

Let $\mathcal{S}_i$ be an optimum backup solution in G_i. Let S_i be the set of edges in $G_i - T_i$ used by $\mathcal{S}_i$—we need to reserve at least 2^i units of bandwidth on the edges of S_i. Further, if v is a vertex in T_i which is an endpoint of an edge in S_i, then it must be the case that at least 2^i units of bandwidth is reserved on the path between v and the root of the tree T_i.

Now, consider an Eulerian tour of S_i. Decompose this into paths such that no path has a vertex of T_i as an internal node. Each of these paths now corresponds to an edge in $G'_i - T_i$. Our solution $\mathcal{S}'_i$ in G'_i reserves 2^i units on these edges. Further, it reserves 2^{i+1} units on each of the paths which join an endpoint of such an edge to the root of the tree T_i—as mentioned above, at least 2^i units are reserved on these edges of T_i in the solution $\mathcal{S}_i$ also. The Eulerian tour counts each edge of S_i twice only. Thus, the cost of $\mathcal{S}'_i$ is at most twice that of $\mathcal{S}$.

The other direction can be shown similarly. □

Finding the backup cost in G'_i. We now have a simpler problem: a graph G'_i, with a spanning tree T_i in which we need to find a tree backup for the edges of E_i, which form a spider. Let r_i be the root of T_i, and let P_j's be the paths of the spider. Also let $T_{i,j}$ be the subtree of T_i hanging off P_j. (See Figure 2 for a picture.) Call a non-tree edge a *back* edge if both its endpoints belong to the same tree $T_{i,j}$, and a *cross edge* otherwise. For example, the edge e_b is a back edge, and e_c a cross edge in the figure. Now each edge e of E_i has a *savior edge* $sav(e)$ which is used to connect the two components formed if e fails. A crucial fact is that if a cross edge from $T_{i,j}$ to $T_{i,j'}$ is a savior for some edge e in P_j, then it is a savior for all edges on P_j which are above e. Hence, fixing the lowest edge e in P_j whose savior is a cross edge implies that all edges above it are also saved by that same savior edge, and all edges below it on P_j must be saved by back edges. The cost $Q(e)$ of saving the rest by back edges depends on the portion of T_i attached to these edges and the back edges between this portion; note that this is entirely independent of the rest of the problem.

Suppose we know, for each edge $e \in \bigcup_j P_j$, the cost $Q(e)$. (We shall discharge this assumption later.) Then the cost of backing up all the edges in E_i consists of the following: for each P_j, picking the single cross edge (say going to $T_{i,j'}$) which is going to be savior (and reserving 2^{i+1} capacity on it), and reserving 2^{i+1} capacity on the edges in $T_{i,j'}$ from the other end of this edge to the root r_i. Of course, we have to add the cost of saving the edges that were not saved by these cross edges to the solution.

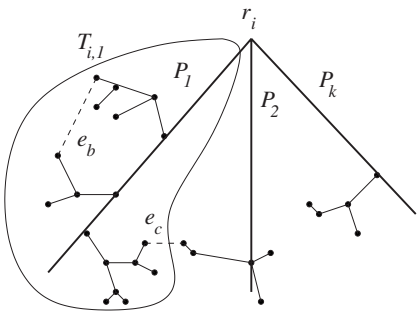

Fig. 2. The tree T_i, with tree edges shown in solid lines and non-tree edges in dashed ones.

We now claim that this can be modeled as a minor variant of the group Steiner tree problem with vertex costs. Each vertex $v \in T_i$ which is the endpoint of some cross edge e from $T_{i,j}$ is belongs to the group S_j and has a "cost" $2^{i+1}c_e + Q(e')$, where e' is the lowest edge in P_j that can be saved by e. (Note that S_j must be a multiset, with the vertex v occurring several times in S_j with various costs, if there are several such cross edges.) As a pedantic aside, there may be no such cross edge, and so r_i also belongs to S_j with cost equal to saving P_j entirely with back edges. This is done for every vertex and every value of j. Now the objective is to find the minimum cost subtree of T_i that contains the root and hits every group S_j at least once, where we also have to pay for the vertex of S_j picked in the tree. It is fairly easy to see that this can be transformed into a regular group Steiner tree problem, and the algorithm of Garg et al. [15] then gives us an $O(\log^2 n)$ approximation.

Computing $Q(e)$. There is one more assumption to be discharged: we have to show how to compute all the $Q(e)$. We will not be able to do this optimally, but we give a constant factor approximation for this as well. Since these are independent problems, let us consider the case when we want to find the cost of backing up P_1 using only back edges.

LEMMA A.3. *Given an edge $e \in P_1$, we can compute a 4-approximation to $Q(e)$ in polynomial time.*

PROOF. We would like to use just the Eulerian trick done above to reduce the problem to edges between vertices of P_1 only, and then find the least cost augmentation. The technical problem that arises is that the optimal solution could be using edges in $T_{i,1} - P_1$, and doing this trick naively could result in our paying several times for this reservation, when paying once would have sufficed. However, we can avoid this problem. Let $P_1(e)$ be the part of P_1 below e.

Let $G'_{i,1}$ be the subgraph of G'_i induced by the vertices in $T_{i,1}$. We construct a graph $G'' = (V'', E'')$ as follows: the vertex set is the set of vertices in $P_1(e)$. Further, it contains all the edges in $P_1(e)$. For any two vertices $u, v \in P_1(e)$, we add an edge $e_{u,v}$ between them in G''. The cost of $e_{u,v}$ is the cost of the shortest path between u and v in $G'_{i,1} - P_1(e)$. We now define an instance of the *minimum augmentation problem*—the instance is the graph G'', and a solution needs to find a set of edges $S \subseteq E'' - P_1(e)$ such that for every edge $e' = (u, v) \in P_1(e)$, $F'' - e'$ contains a path between u and v. The goal is to find such a set S of smallest cost. There exists a polynomial time 2-approximation algorithm for this problem [13].

We claim that the cost of an optimal solution to the minimal augmentation problem and the value of $Q(e)$ are within a factor of 2 of each other.

Suppose we are given a set of edges S' such that the S' protects the edges in $P_1(e)$ in the graph $G'_{i,1}$. Clearly, S' is acyclic. So by taking an Eulerian tour of S', we can transform it into a solution to the above instance of the minimum augmentation problem by doubling its cost.

Let us prove the converse now. Suppose we are given a solution to the minimum augmentation problem in G''. Let the set of edges protecting $P_1(e)$ be S. Note that no vertex of $P_1(e)$ can be incident with three edges of S—indeed, this would imply that

one of these edges is redundant. So, assuming this is the case, we can map it back to a solution for protecting $P_1(e)$ in the graph $G'_{i,1}$ whose cost is at least half the cost of S. This proves the lemma. □

Thus we have shown the following theorem.

THEOREM A.4. *Given a polynomial time α-approximation algorithm for the group Steiner tree problem, there exists a polynomial time algorithm for the backup problem for tree networks that produces a solution of cost $O(\alpha)\sum_e c_e(u^p(e) + u^b(e))$ where u^b is a minimum cost feasible backup solution to the given primary network u^p.*

Appendix B. Linear Programming Formulations for SSSPR. We provide two linear programming relaxations of SSSPR. The first formulation is based on cuts and the second formulation is based on flows. We show that the second formulation dominates the first one on all instances.

A *cut* in a graph G is a partition of V into two disjoint sets V_1 and V_2. The edges of the cut are those edges that have precisely one endpoint in both V_1 and V_2. Let T be a subgraph of G which is a tree. A cut (V_1, V_2) of G is a *canonical cut* of G with respect to T if there exists an edge $e \in T$, decomposing T into T_1 and T_2, such that $T_1 \subseteq V_1$ and $T_2 \subseteq V_2$.

Let p be a primary path from the source s to the destination t. It follows from Suurballe's [32] procedure that a set of edges q is a backup to a path p if it covers all the canonical cuts of p. This leads us to the following linear programming formulation which is based on covering cuts. For an edge e, let $x(e)$ denote the primary indicator variable and let $y(e)$ denote the backup indicator variable.

$$
\begin{array}{lll}
\text{(Cut-LP)} \quad \min & \displaystyle\sum_{e\in E} c_1(e)\cdot x(e) + c_2(e)\cdot y(e) & \\
\text{s.t.} & \displaystyle\sum_{e\in C}(x(e)+y(e)) \geq 2 & \text{for all } \{s,t\}\text{-cuts } C,\\
& \displaystyle\sum_{e\in C} x(e) \geq 1 & \text{for all } \{s,t\}\text{-cuts } C,\\
& x(e)+y(e) \leq 1 & \text{for all } \quad e\in E,\\
& x(e), y(e) \geq 0 & \text{for all } \quad e\in E.
\end{array}
$$

It is not hard to see that the value of an optimal (fractional) solution to (Cut-LP) is a lower bound on the value of an optimal integral solution to SSSPR. We now present a second linear programming formulation of SSSPR which is based on flows. Our formulation relies on the following lemma.

LEMMA B.1. *Let p be a primary path from s to t and let q be a set of backup edges. Replace each edge from p and q by two parallel anti-symmetric unit capacity arcs. Then two units of flow can be sent from s to t.*

PROOF. Let path p be $s = v_1, \ldots, v_k = t$. We send the first unit of flow from s to t through p. For the second unit of flow we proceed as follows. There has to be a backup

edge (s, v_i), $1 \le i \le k$. Let a_1 be the arc (s, v_i) that maximizes i. There has to be a backup edge $(v_j, v_{i'})$ where $1 \le j \le i$ and $i < i' \le k$. Let a_2 be the arc $(v_j, v_{i'})$ that maximizes i'. We continue this way until we reach the sink, i.e., the sink is the tail of arc a_ℓ. The second unit of flow is sent first on arc a_1, then it goes back in p until it reaches the head of arc a_2, then it is sent on arc a_2, then it goes back in p until it reaches the head of arc a_3, and so on till it reaches the sink t. □

This leads us to the following bidirected flow relaxation. We replace each edge by two parallel anti-symmetric unit capacity arcs. Denote by $D = (V, A)$ the directed graph obtained. The goal is to send two units of flow in D from s to t, one from each commodity, while minimizing the cost. Denote the two commodities by blue and red, corresponding to primary and backup edges, respectively. The cost of the blue commodity on an arc a (obtained from edge e) is equal to $c_1(e)$. The cost of the red commodity on an arc a (obtained from edge e) is defined as follows. Suppose there is a blue flow on arc a^- of value f. Then the red flow on a up to the value of f is *free*. Beyond f, the cost of the red flow is $c_2(e)$.

$$
\begin{array}{lll}
\text{(Flow-LP)} & \min & \displaystyle\sum_{e\in E} c_1(e)\cdot f_1(e) + c_2(e)\cdot f_2(e) \\
& \text{s.t.} & \bar{x} \text{ supports a unit flow } (f_1) \text{ between } s \text{ and } t, \\
& & \bar{y} \text{ supports a unit flow } (f_2) \text{ between } s \text{ and } t, \\
& & f_1(e) \ge \max(f_1(a),\, f_1(a^-)) \qquad \text{for all} \quad e = (a, a^-), \\
& & f_2(e) \ge \max((f_2(a) - f_1(a^-)), 0) + \max((f_2(a^-) - f_1(a)), 0) \\
& & \qquad\qquad \text{for all} \quad e = (a, a^-), \\
& & x(a) + y(a) \le 1 \qquad \text{for all} \quad a \in A, \\
& & x(a),\, y(a) \quad \ge 0 \qquad \text{for all} \quad a \in A.
\end{array}
$$

Given a solution to the SSSPR problem, Lemma B.1 tells us how to obtain a two-commodity flow solution from it. We claim that the cost of the two-commodity flow solution is equal to the cost of the solution to the SSSPR problem. Notice that the blue flow costs the same as the blue edges in the SSSPR solution. The cost of the red flow is zero on arcs which are obtained from blue edges. On other edges, the cost of the red flow and the cost of the SSSPR solution are the same. Therefore, the value of an optimal (fractional) solution to (Flow-LP) is a lower bound on the value of an optimal integral solution. We now prove that (Flow-LP) dominates (Cut-LP).

THEOREM B.2. *For any instance of the SSSPR problem, the cost of the optimal solution produced by (Flow-LP) is at least as high as the cost of the optimal solution produced by (Cut-LP).*

PROOF. We show that given a feasible solution to (Flow-LP), we can generate a feasible solution to (Cut-LP) without increasing the cost. Consider edge $e \in E$ which is replaced by two anti-parallel arcs a and a^- in (Flow-LP). Without loss of generality, we can assume that at most one of $\{f_1(a),\, f_1(a^-)\}$ is non-zero and at most one of $\{f_2(a),\, f_2(a^-)\}$ is non-zero. Define $x(e) = f_1(e)$ (or $x(e) = f_1(a) + f_1(a^-)$) and $y(e) = \min(f_2(e), 1 - f_1(e))$. We show that $\{x(e),\, y(e)\}_{e\in E}$ defines a feasible solution for (Cut-LP). Let the x-capacity

(y-capacity) of a cut be the sum of the variables $x(e)$ ($y(e)$) taken over the edges e belonging to the cut. Clearly, the x-capacity of all $\{s,t\}$-cuts is at least one, since flow f_1 in D sends one unit of flow from s to t. It remains to show that the x-capacity together with the y-capacity of all $\{s,t\}$-cuts is at least 2.

Consider a particular $\{s,t\}$-cut C. Decompose flow function f_1 in D into flow paths, each of flow value ε. Let $n(k)$ denote the number of flow paths in the decomposition that use precisely $2k+1$ edges from C. Clearly, $\sum_{k=0}^{\infty} n(k)\cdot\varepsilon = 1$, and so the contribution of flow f_1 in D to the x-capacity of C is

$$\begin{aligned}\sum_{k=0}^{\infty}(2k+1)\cdot n(k)\cdot\varepsilon &= 2\sum_{k=0}^{\infty} k\cdot n(k)\cdot\varepsilon + \sum_{k=0}^{\infty} n(k)\cdot\varepsilon \\ &= 2\sum_{k=0}^{\infty} k\cdot n(k)\cdot\varepsilon + 1.\end{aligned}$$

Suppose $\sum_{k=0}^{\infty} k\cdot n(k)\cdot\varepsilon < \frac{1}{2}$, otherwise we are done. The red flow in D, f_2, can send for "free" flow of value at most $\sum_{k=0}^{\infty} k\cdot n(k)\cdot\varepsilon$ using arcs belonging to cut C. (For each arc a carrying a blue flow of value ε, the red flow of value ε can be sent on a^- for free.) Therefore, the red flow must send a flow of value at least $1-\sum_{k=0}^{\infty} k\cdot n(k)\cdot\varepsilon$ using capacity "paid" for by f_2. (Note that for this flow we have $y(e) = f_2(e)$ for all edges e.) Hence, the y-capacity of C is at least $1-\sum_{k=0}^{\infty} k\cdot n(k)\cdot\varepsilon$, yielding that the capacity of cut C (x-capacity and y-capacity) is $1+2\sum_{k=0}^{\infty} k\cdot n(k)\cdot\varepsilon+1-\sum_{k=0}^{\infty} k\cdot n(k)\cdot\varepsilon \geq 2$, thus completing the proof. □

Integrality gap. It is not hard to show that a fractional solution to both formulations can be rounded to an integral solution while increasing the cost by at most a factor of 2. The proof is along the lines of the proof for the combinatorial 2-approximation algorithm presented earlier. Unfortunately, this is the best approximation factor that can be obtained by rounding a fractional solution to (Flow-LP). Consider the instance depicted in Figure 3. The cost of the primary and backup flows on each edge e in the graph is denoted by $(c_1(e), c_2(e))$. In the figure C is a constant that is at least 2. In an

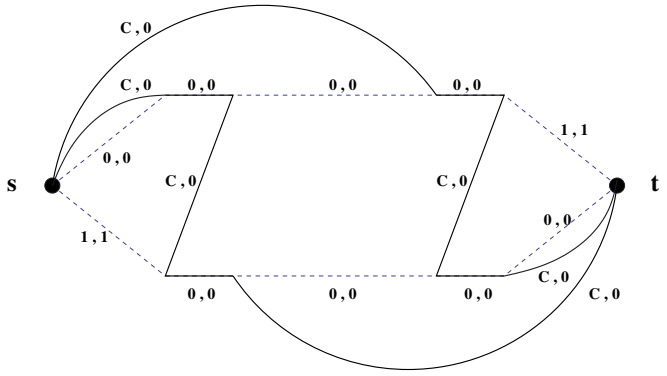

Fig. 3. The gap between the optimal fractional and the integral solutions is 2.

optimal fractional solution, the primary flow is equally split between the two dashed paths from s to t and its cost is 1. The backup flow is also equally split between the two solid paths and its cost is 0. The total fractional cost is thus 1. Any integral solution has to choose one of the two dashed paths for the primary flow and pay a cost of 1, and then choose a solid path which is disjoint from the chosen primary path, and pay a cost of 1. Thus, the gap between the optimal fractional and the integral solutions is 2.

References

[1] B. Awerbuch and Y. Azar. Buy-at-bulk network design. In *Proceedings of IEEE FOCS*, pages 542–547, 1997.

[2] A. Balakrishnan, T. Magnanti, and P. Mirchandani. Network design. In *Annotated Bibliographies in Combinatorial Optimization*, M. Dell'Amico, F. Maffioli, and S. Martello (eds.), pages 311–334. Wiley, New York, 1997.

[3] A. Balakrishnan, T. Magnanti, J. Sokol, and Y. Wang. Telecommunication link restoration planning with multiple facility types. *Annals of Operations Research* (volume "Topological Network Design in Telecommunications," edited by P. Kubat and J. M. Smith), 106(1–4):127–154, 2001.

[4] A. Balakrishnan, T. Magnanti, J. Sokol, and Y. Wang. Spare-capacity assignment for line restoration using a single-facility type. *Operations Research*, 50(4):617–635, 2002.

[5] D. Bienstock and G. Muratore. Strong inequalities for capacitated survivable network design problems. *Mathematical Programming*, 89:127–147, 2001.

[6] A. Bremler-Barr, Y. Afek, E. Cohen, H. Kaplan, and M. Merritt. Restoration by path concatenation: fast recovery of MPLS paths. In *Proceedings of ACM PODC* '01, pages 43–52; also in *Proceedings of the ACM SIGMETRICS* '01 *Conference* (2-*Page Poster*), 2001.

[7] G. Brightwell, G. Oriolo, and F. B. Shepherd. Reserving resilient capacity in a network. *SIAM Journal on Discrete Mathematics*, 14(4):524–539, 2001.

[8] G. Brightwell, G. Oriolo, and F. B. Shepherd. Reserving resilient capacity with upper bound constraints. *Networks*, 41(2):87–96, 2003.

[9] G. Dahl and M. Stoer. A cutting plane algorithm for multicommodity survivable network design problems. *INFORMS Journal on Computing*, 10:1–11, 1998.

[10] B. Davie and Y. Rekhter. *MPLS*: *Technology and Applications*. Morgan Kaufmann, San Mateo, CA, 2000.

[11] N. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. Ramakrishnan, and J. van der Merwe. A flexible model for resource management in virtual private networks. In *Proceedings of ACM SIGCOMM*, Computer Communication Review, volume 29, pages 95–108, 1999.

[12] L. Fleischer, A. Meyerson, I. Saniee, F. B. Shepherd, and A. Srinivasan. Near-optimal design of MPλS tunnels with shared recovery. DIMACS Mini-Workshop on Quality of Service Issues in the Internet, 2001.

[13] G. N. Frederickson and J. JáJá. Approximation algorithms for several graph augmentation problems. *SIAM Journal on Computing*, 10(2):270–283, 1981.

[14] A. Fumagalli and L. Valcarenghi. IP restoration vs. WDM protection: is there an optimal choice? *IEEE Network*, 14(6):34–41, November/December 2000.

[15] N. Garg, G. Konjevod, and R. Ravi. A polylogarithmic approximation algorithm for the group Steiner tree problem. *Journal of Algorithms*, 37(1):66–84, 2000. (Preliminary version in *Proceedings of the* 9*th Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 253–259, 1998.)

[16] M. Goemans, A. Goldberg, S. Plotkin, D. Shmoys, E. Tardos, and D. Williamson. Improved approximation algorithms for network design problems. In *Proceedings of the* 5*th Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 223–232, 1994.

[17] M. Goemans and J. Kleinberg. An improved approximation ratio for the minimum latency problem. In *Proceedings of* 7*th ACM–SIAM SODA*, pages 152–157, 1996.

[18] A. Gupta, A. Kumar, J. Kleinberg, R. Rastogi, and B. Yener. Provisioning a virtual private network: a network design problem for multicommodity flow. In *Proceedings of the* 33*rd Annual ACM Symposium on Theory of Computing*, pages 389–398, 2001.

[19] A. Gupta, A. Kumar, and R. Rastogi. Traveling with a Pez dispenser (or, routing issues in MPLS). In *Proceedings of the* 42*nd IEEE Symposium on Foundations of Computer Science* (*Las Vegas*, *NV*, 2001), pages 148–157. IEEE Computer Society Press, Los Alamitos, CA, 2001.

[20] A. Gupta, A. Kumar, and T. Roughgarden. Simpler and better approximation algorithms for network design. In *Proceedings of the* 35*th Annual ACM Symposium on Theory of Computing*, pages 365–372, 2003.

[21] E. Halperin and R. Krauthgamer. Polylogarithmic inapproximability. In *Proceedings of the* 35*th ACM Symposium on Theory of Computing*, pages 585–594. ACM Press, New York, 2003.

[22] G. Italiano, R. Rastogi, and B. Yener. Restoration algorithms for virutal private networks in the hose model. *Proceedings of Infocom* 2002.

[23] K. Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21(1):39–60, 2001. (Preliminary version in *Proceedings of the* 39*th Annual Symposium on Foundations of Computer Science*, pages 448–457, 1998.)

[24] M. Kodialam and T. V. Lakshman. Minimum interference routing with applications to MPLS traffic engineering. *Proceedings of Infocom* 2000, pages 884–893, 2000.

[25] M. Kodialam and T. V. Lakshman. Dynamic routing of bandwidth guaranteed tunnels with restoration. *Proceedings of Infocom* 2000, pages 902–911, 2000.

[26] R. Motwani, S. Phillips, and E. Torng. Non-clairvoyant scheduling. *Theoretical Computer Science*, 130:17–47, 1994.

[27] R. Ramaswami and K. Sivarajan. *Optical Networks*. 2nd edition. Morgan Kaufmann, San Mateo, CA, October 2001.

[28] E. C. Rosen, D. Tappan, Y. Rekhter, G. Federkow, D. Farinacci, T. Li, and A. Conta. MPLS label stack encoding (RFC 3032). `http://www.ietf.org/rfc/rfc3032.txt`, January 2001.

[29] E. C. Rosen, A. Viswanathan, and R. Callon. MultiProtocol Label Switching architecture (RFC 3031). `http://www.ietf.org/rfc/rfc3031.txt`, January 2001.

[30] F. S. Salman, J. Cheriyan, R. Ravi, and S. Subramanian. Buy-at-bulk network design: approximating the single-sink edge installation problem. In *Proceedings of* 8*th ACM–SIAM SODA*, pages 619–628, 1997.

[31] C. Scott, P. Wolfe, M. Erwin, and A. Oram. *Virtual Private Networks*. O'Reilly, Cambridge, MA, 1998.

[32] J. W. Suurballe. Disjoint paths in a network. *Networks*, 4:125–145, 1974.

[33] D. Williamson, M. Goemans, M. Mihail, and V. Vazirani. A primal–dual approximation algorithm for generalized Steiner network problems. *Combinatorica*, 15(3):435–454, 1995. (Preliminary version in *Proceedings of the* 25*th Annual ACM Symposium on Theory of Computing*, pages 708–717, 1993.)

[34] D. Zhou and S. Subramaniam. Survivability in optical network. *IEEE Network*, 14(6):16–23, 2000.