Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

ANDREAS BLEY   MARTIN GRÖTSCHEL   ROLAND WESSÄLY

Design of Broadband Virtual Private Networks: Model and Heuristics for the B–WiN

Preprint SC 98-13 (March 1998)

Design of Broadband Virtual Private Networks: Model and Heuristics for the B–WiN *

Andreas Bley Martin Grötschel Roland Wessäly

March 30, 1998

Abstract

We investigate the problem of designing survivable broadband virtual private networks that employ the Open Shortest Path First (OSPF) routing protocol to route the packages. The capacities available for the links of the network are a minimal capacity plus multiples of a unit capacity. Given the directed communication demands between all pairs of nodes, we wish to select the capacities in a such way, that even in case of a single node or a single link failure a specified percentage of each demand can be satisfied and the costs for these capacities are minimal. We present a mixed–integer linear programming formulation of this problem and several heuristics for its solution. Furthermore, we report on computational results with real-world data.

Keywords: Telecommunication Network Design, Survivable Networks, Network Capacity Planning, OSPF Routing, Shortest Path Routing, Heuristics

Mathematical Subject Classification (1991): 90B12, 90C11, 90C27, 90C90

1 Introduction

In this article we describe a network design and bandwidth allocation problem that we developed in cooperation with our partner DFN–Verein e.V. (Registered Association for the Promotion of a German Research Network). DFN acts as the internet provider for German universities and research institutions. The backbone network of DFN, called B–WiN, is operated as a broadband virtual private network (BVPN) on the ATM cross connect network of the Deutsche Telekom AG.

Currently, the network contains ten central service switches. It is possible to rent a link at a certain capacity between each pair of switches. These links are virtual paths in the ATM cross connect network, whose structure is transparent to DFN. The capacities available for rented links are a certain minimum capacity plus multiples of a unit capacity. See [4] for a description of the technical implementation of the B–WiN.

In the network design problem considered here, we must employ the OSPF (Open Shortest Path First, see [3] and [8]) routing protocol to route the traffic demands in the network. Using this routing policy, each package is sent to its destination along a shortest path with respect to some given edge weights. In principle, the routing is performed in an embedded IP overlay network consisting of permanent virtual circuits (PVCs), which are paths in the

*This work was partially funded by the Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie (BMBF).

BVPN. However, since DFN currently identifies virtual paths and virtual circuits, we do not distinguish either.

Nowadays, a network provider must not only offer large bandwidths and high transmission rates, but also guarantee a specified quality of service. Hence, it is important for the provider to limit the impact of network failures. In this article we consider single failures of switches or links. Given the directed traffic demands between all pairs of switches, the capacities must be selected in such a way, that even in case of a single component failure at least a specified percentage of each demand can be routed in the remaining network.

The design of such survivable networks has been studied for various capacity and survivability models; see [1] for several mixed-integer linear programming models. In this article we present a model that differs in one important point from the models studied previously. We integrate the survivability and the OSPF routing policy in one model, i.e., in the non-failure case and in all single component failure cases the OSPF routing protocol must be used. In particular this means that the way the routing paths may change in a failure situation is defined by the routing protocol and that each demand must always be routed on a single path. Bifurcated routing is allowed neither in the non-failure nor in any of the failure cases.

The remainder of this article is organized as follows: The next section contains a detailed description of the problem, its input data, and its constraints. A mixed–integer linear programming model for the IP network design and routing problem is developed in Section 3. In Section 4 we briefly present several heuristics to solve this problem. Computational results for real-world data provided by DFN are reported in the last section.

2 The Problem

Formally, the problem can be described as follows: We are given a **supply graph** $\mathbf{G} = (V, E)$ consisting of the **nodes** $\mathbf{V}$ and the **edges** $\mathbf{E}$. In our practical application, the node set V corresponds to the set of central service switch locations of the B–WiN. These switches are the locations of the IP routers. The edge set E corresponds to the set of all virtual paths (in the underlying ATM–network) that may be rented from Deutsche Telekom.

For every supply edge, the installed capacity must be either zero or the **minimal capacity** $\mathbf{c_{min}} \in \mathbb{Z}_+$ plus a multiple of the **unit capacity** $\mathbf{c_u} \in \mathbb{Z}_+$. For no edge the capacity may exceed the **maximal capacity** $\mathbf{c_{max}} \in \mathbb{Z}_+$. The capacities are bidirectional, i.e., the capacity installed on an edge can be used in both directions independently. An edge with non-zero capacity is called a **chosen edge**.

We wish to design a network that is still operational if a single node or a single edge fails. Therefore, we introduce the set of **operating states** S. The considered operating states $s \in S$ are the **normal operating state** ($s = 0$), which is the state with all nodes and edges operational, and the **failure states**, which are the states with a single edge ($s = e \in E$) or a single node ($s = v \in V$) non-operational. Note that in a node failure state $s = v \in V$ all edges incident to v are non-operational, too.

For each ordered pair of nodes $(u, v) \in V \times V$ the value $\mathbf{d_{uv}} \in \mathbb{Z}_+$ is the directed **communication demand** from u to v. Clearly, in a node failure state $s = v \in V$ the demands with origin or destination v cannot be satisfied and are therefore not considered in this operating state.

Since network failures are considered to be non-permanent, we do not have to satisfy the total demands in a failure situation. For each operating state $s \in S$, we introduce a **reservation parameter $\mathbf{r^s}$** > 0 specifying how many percent of each demand must be routable in this operating state. To hedge against burstiness in traffic, we allow to "oversize" the installed capacities in the normal operating state by setting the "reservation" parameter r^0 to a value larger than one. In this case, even if all demands increase simultaneously by a factor of r^0, the capacities permit a feasible routing as long as no network component fails.

In principle, the demands are routed in a second network layer, the so called overlay network, that is embedded in the BVPN. The nodes of the overlay network are the nodes of the supply graph. Its edges, which are permanent virtual circuits (PVCs) in the BVPN, correspond to paths in the supply graph. For each PVC a capacity is reserved on its edges. With this second network layer it is easier to manage different traffic types, such as IP, X.25, etc., in the network. Usually, one sets up a separate overlay network for each traffic type and routes the corresponding demands within this overlay network. However, we assume here that there is only one traffic type in the network, namely IP traffic (Internet Protocol traffic). Furthermore, we restrict ourselves to the case where, for each rented virtual path, there is exactly one direct permanent virtual circuit set up at the full capacity of the virtual path, i.e., the IP overlay network and the BVPN are equivalent. Hence, we can pretend that the demands are routed directly in the supply graph and no second network layer exists.

The capacities chosen for the supply edges have to be large enough to allow a feasible routing with respect to the OSPF routing protocol. Assuming non-negative **routing weights** for all supply edges, the OSPF protocol implies that each demand is sent from its origin to its destination along a shortest path with respect to these weights. In each operating state only operational nodes and edges are considered in the shortest path computation. Although not required by the OSPF protocol, in our problem we have to guarantee the existence of a path between any pair of operational nodes in each operating state. This implies that the subgraph of G induced by the chosen supply edges must be 2-node-connected.

The routing weights must be chosen in such a way, that, for all operating states and all demands, the shortest path from the demand's origin to its destination is unique with respect to these weights. Otherwise, it is not determined which one of the shortest paths will be selected by the implementation of the routing protocol in the network and, therefore, it would be impossible to guarantee that the chosen capacities permit a feasible routing of the demands.

For each demand, the variation of its data package transmission times increases significantly with the number of nodes in the routing path, especially if the network is heavily loaded. For multi-media applications, for example, this might lead to unacceptable differences in the transmission times. To avoid too large variations, the number of edges in each routing path is bounded from above by $\ell \in \mathbb{Z}_+$ in the normal operating state.

DFN does not allow an arbitrary number of edges in a solution. It stipulates that a predefined number, say $\mathbf{m}$, of supply edges must be chosen. As noted above, the graph induced by these edges has to be 2-node-connected.

Finally, let us explain the (complicated) cost structure of the problem considered. The total cost of installing capacities on the supply edges is defined as follows. For installing the minimal capacity c_{min} on m edges we have the **fixed cost $\mathbf{K_{min}}$** $\in \mathbb{Z}_+$. Since we cannot affect these costs, they can be omitted in the optimization.

Each capacity unit c_u that is installed in addition to the minimal capacity on a single supply edge causes further costs. The set of feasible capacities above the minimal capacity c_{min} is divided into consecutive capacity intervals. Each interval (except maybe the last interval) contains $\mathbf{b} \in \mathbb{Z}_+$ capacity units and, thus, has a total capacity of $\mathbf{c_b} := b \cdot c_u$. The number of these intervals is $\lceil \frac{c_{max}-c_{min}}{c_b} \rceil$. For each $i \in I := \{0, \ldots, \lceil \frac{c_{max}-c_{min}}{c_b} \rceil - 1\}$, we denote the cost for one capacity unit c_u installed within the interval $[c_{min} + i \cdot c_b\, , \, c_{min} + (i+1) \cdot c_b]$ by $\mathbf{k_i} \in \mathbb{Z}_+$.

For the capacity units installed *simultaneously on all* m chosen supply edges in interval $[c_{min} + i \cdot c_b\, , \, c_{min} + (i+1) \cdot c_b]$, the sum of the unit costs of these m capacity units is reduced by the **discount** $\mathbf{K_i} \in \mathbb{Z}_+$, with $0 \leq K_i \leq m \cdot k_i$. For notational convenience, we denote the minimum of the capacities of the m chosen supply edges as **network base capacity** $\mathbf{c_{base}}$. Note that the cost discount applies exactly to those capacity units that are installed below the network base capacity.

All cost parameters depend on the number of chosen edges m. The general structure of the cost function is shown in Figure 1.

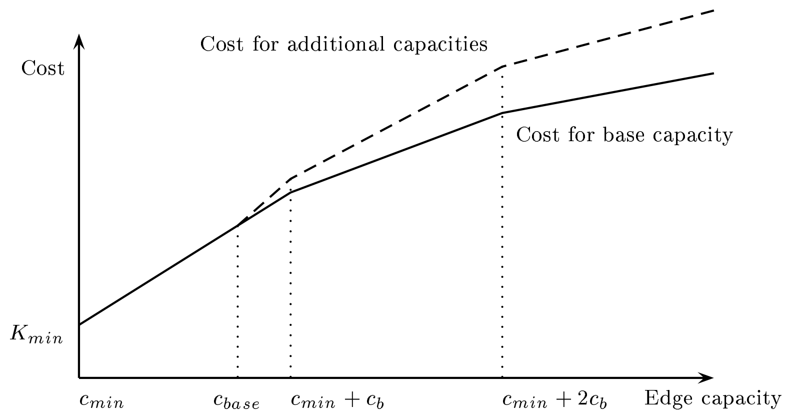

Figure 1: The cost function for a supply edge

Note that in the problem considered here the cost of a solution does not depend on the physical lengths of the chosen edges. Nevertheless, given the **physical lengths** $\lambda_\mathbf{e} \in \mathbb{R}_+$ for all supply edges $e \in E$, the average length of all chosen edges must not exceed the given **length bound** $\lambda_\mathbf{max}$.

The **IP network design problem**, called **IP-ND**, consists of two parts. In the first part, we design the overall BVPN, i.e., we decide which capacities to install on which edges. In the second part, we determine the routing weights and compute the routings for each operating state. The objective is to minimize the total costs of the network, which depend on the installed capacities.

3 The Model

In this section we present a **mixed-integer linear programming model** of the IP-ND network dimensioning and routing problem.

3.1 The Supply Graph

To describe which edges are chosen at which capacities we have so-called global and local variables in our model. The global variables are used to decide the network base capacity, whereas the local variables are employed to decide whether a supply edge will be installed in a solution, and if so, to decide the capacity installed on this particular edge. The **global variables** are

- $x_i \in \{0, \dots, b\}$, for all $i \in I$, determining the number of capacity units installed in interval $[c_{min} + i \cdot c_b\, ,\, c_{min} + (i+1) \cdot c_b]$ as base capacity.

The **local variables**, for every supply edge $e \in E$, are

- $z_e \in \{0, 1\}$, where $z_e = 1$, if we install capacity on supply edge e,
- $z_{i,e} \in \{0, 1\}$, for all $i \in I$, where $z_{i,e} = 1$ if the capacity of edge e is larger than $c_{min} + i \cdot c_b$, and
- $x_{i,e} \in \{0, \dots, b\}$, for all $i \in I$, determining the number of capacity units installed in interval $[c_{min} + i \cdot c_b\, ,\, c_{min} + (i+1) \cdot c_b]$ on edge e.

To represent feasible choices, the local variables associated with a supply edge $e \in E$ must satisfy

$$z_{i,e} - z_e \leq 0 \qquad \forall\, i \in I, \tag{1}$$

$$x_{i,e} - b \cdot z_{i,e} \leq 0 \qquad \forall\, i \in I, \text{ and} \tag{2}$$

$$x_{i,e} - b \cdot z_{i+1,e} \geq 0 \qquad \forall\, i \in I. \tag{3}$$

Due to inequality (1) the variable $z_{i,e}$ may only happen to be positive, if $z_e = 1$. Suppose we wish to choose edge e in a solution, i.e., we have $z_e = 1$. In this case it follows from (2) and (3) that the variables $x_{i,e}$ can only be positive if $z_{i,e} = 1$ and that $z_{i+1,e} = 1$ implies $x_{i,e} = b$. Hence, if capacity units are installed in interval $i \in I$, then $z_{j,e} = 1$ and $x_{j,e} = b$ for all $j < i$, i.e., all b capacity units are installed in the capacity intervals "below" i, and $z_{i,=}1$. This determines a proper setting of the local variables and the edge's capacity. If we do not choose edge e, i.e., $z_e = 0$, then the inequalities (1), (2), and (3) imply $z_{i,e} = x_{i,e} = 0$ for all $i \in I$.

Note that, if the capacity of edge e equals $c_{min} + i \cdot c_u$ for some $i \in I$, then the variable $z_{i,e}$ may be zero as well as one.

With the following constraints we guarantee, that the capacity of each chosen supply edge exceeds the base capacity:

$$x_i \leq x_{i,e} + b(1 - z_e) \qquad \forall\, i \in I,\, e \in E. \tag{4}$$

To understand (4) consider the following. If supply edge e is chosen in the network, then $z_e = 1$. In this case inequality (4) reduces to $x_i \leq x_{i,e}$, which means that the number of capacity units installed in interval $[c_{min} + i \cdot c_b\, ,\, c_{min} + (i+1) \cdot c_b]$ on edge e is at least the number of units installed for the network base capacity. In the other case $z_e = 0$ and (4) is satisfied anyway.

For notational convenience, we introduce auxiliary variables

$$y_e := c_{min} \cdot z_e + c_u \sum_{i \in I} x_{i,e} \qquad \forall\, e \in E, \tag{5}$$

representing the capacities installed on the supply edges.

It is easy to see, that, if $K_i > 0$ for all $i \in I$, in any optimal solution of our mixed-integer linear programming model the term $\sum_{i \in I} x_i$ is the network base capacity. The upper bounds for these capacities are enforced by

$$y_e \quad \leq c_{max} \qquad \forall\, e \in E. \tag{6}$$

To ensure that the number of chosen supply edges is exactly m we have the equality

$$\sum_{e \in E} z_e = m\,. \tag{7}$$

The restriction of the average (physical) length of the edges in a solution can be formulated as

$$\sum_{e \in E} z_e(\lambda_e - \lambda_{max}) \leq 0. \tag{8}$$

The objective is to minimize the total cost of installing the necessary capacities on the edges of the supply graph. This is formulated as

$$\min \quad K_{min} + \sum_{e \in E} \sum_{i \in I} k_i x_{i,e} - \sum_{i \in I} K_i x_i\,. \tag{9}$$

The term $K_{min} + \sum_{e \in E} \sum_{i \in I} k_i x_{i,e}$ describes the total cost of the installed capacity units and $\sum_{i \in I} K_i x_i$ is the discount for those capacity units, that are installed for the network base capacity.

3.2 The OSPF–Routing

In the following we present a model of the OSPF routing protocol, which is used to route the IP traffic in the network.

Let $G^s = (V^s, E^s)$ be the supply graph in operating state $s \in S$, i.e, the subgraph of G consisting of all nodes and edges that are still operational. The directed graph $D^s = (V^s, A^s)$ with

$$A^s := \bigcup_{e \in E^s} \{\overrightarrow{e}, \overleftarrow{e}\}$$

is the directed graph **associated** with G^s, where $\overrightarrow{e}$ and $\overleftarrow{e}$ denote the two orientations of edge e. By $D = (V, A) = (V^0, A^0)$ we denote the digraph associated with G.

Using the OSPF routing protocol, each package is routed to its destination on a shortest path with respect to a common edge weight function. All packages emanating from a node with the same destination must use the same route, bifurcation is not allowed. In our model we use the **path variables**

- $t^s_{uv,a} \in \{0, 1\}$, for all $s \in S$, $u, v \in V^s$, and $a \in A^s$, which are chosen to be one if arc a is contained in the routing path from u to v in operating state s.

To guarantee that, for every operating state $s \in S$ and every pair $u, v \in V^s$ of nodes, the arcs corresponding to the non-zero path variables form a path we introduce the equalities and inequalities

$$\sum_{a \in \delta^+_{D^s}(u)} t^s_{uv,a} = \sum_{a \in \delta^-_{D^s}(v)} t^s_{uv,a} = 1\,, \tag{10}$$

$$\sum_{a \in \delta^+_{D^s}(w)} t^s_{uv,a} - \sum_{a \in \delta^-_{D^s}(w)} t^s_{uv,a} = 0 \qquad \forall\, w \in V^s \setminus \{u,v\}, \text{ and} \tag{11}$$

$$\sum_{a \in A^s(W)} t^s_{uv,a} \le |W| - 1 \qquad \forall\, \emptyset \ne W \subseteq V^s, \tag{12}$$

where $A^s(W)$ denotes the set of arcs in the subgraph of D^s induced by W. Equality (10) ensures that there is exactly one arc leaving the source node u and one arc entering the destination node v. For every other node we ensure that the number of incoming arcs equals the number of leaving arcs by (11), while (12) eliminates additional cycles.

To guarantee that no routing path contains more than ℓ arcs in the normal operating state we need the inequalities

$$\sum_{a \in A^0} t^0_{uv,a} \le \ell \qquad \forall\, u,v \in V. \tag{13}$$

For each operating state $s \in S$, the installed capacities must permit a feasible routing of the specified reservation percentage r^s of each demand. This condition yields the capacity constraints

$$r^s \sum_{u,v \in V^s} t^s_{uv,a} d_{uv} \le y_e \qquad \forall\, s \in S,\, e \in E^s,\, a \in \{\overleftarrow{e}, \overrightarrow{e}\}. \tag{14}$$

Up to here, we have modeled a survivable single path routing scheme, i.e., a scheme where in each operating state there is exactly one routing path between any pair of nodes. To model the OSPF routing protocol completely, we also have to ensure that all routing paths are shortest paths with respect to a common weight function $w : E \to \mathbb{R}_+$. We introduce the following variables:

- $w_e \in \mathbb{R}_+$, for each $e \in E$, which is the **routing weight** of edge e, and
- $\pi^s_{v,u} \in \mathbb{R}_+$, for each $s \in S$ and each pair $u, v \in V^s$, the **potential variables** denoting a feasible potential of node v with respect to the edge weights w and the root node u in D^s.

The node potentials are used to decide which arcs are on shortest paths and can be used in routing paths and which are not. They correspond to the dual variables in the embedded shortest path problems (see [2] and [5]).

The feasibility of the potentials is guaranteed by the following metric inequalities:

$$\pi^s_{u,u} = 0 \qquad \forall\, u \in V^s, \tag{15}$$

$$\pi^s_{v,u} - \pi^s_{w,u} + w_e \ge 0 \qquad \forall\, u,v,w \in V^s,\, vw = e \in E^s. \tag{16}$$

It follows from (15) and (16) that in each solution of this mixed–integer program the variable $\pi^s_{v,u}$ is at most the distance from v to u in G^s with respect to w. Note that the perturbation technique that will be introduced in the end of this section to ensure uniqueness of all shortest paths implies that all routing weight variables are strictly positive.

If, for some feasible potentials, inequality (16) holds with equality for all edges of a path with destination u, then this path is a shortest path from its origin to u in G^s (see [2]). Furthermore, there exist feasible potentials, such that edge e is contained in a shortest path with destination u in G^s if and only if inequality (16) is tight. Hence, we can ensure that all routing paths are shortest paths from their origins to their destinations, by introducing for each operating state $s \in S$ and each pair of operational nodes $u_1, u_2 \in V^s$ the inequalities

$$\pi^s_{v,u_2} - \pi^s_{w,u_2} + w_e + M(t^s_{u_1u_2,a} - 1) \leq 0 \quad \forall\, (v,w) = a \in A^s,\, e = vw, \tag{17}$$

where M is a large constant. It is sufficient to choose M to be two times the maximum of the routing weights.

Finally, we have to ensure that all routing paths are *unique* shortest paths with respect to w. We apply a perturbation technique that guarantees the uniqueness of the lengths of *all* paths in G. Let $\sigma : E \to \{1, \ldots, |E|\}$ be an arbitrary permutation of the edges of G. We define the **perturbation** of the routing weights as

$$\epsilon_e : E \to \mathbb{R}_+, \quad \epsilon_e := 2^{\sigma(e)-|E|-1}. \tag{18}$$

It is easy to see that $\sum_{e\in E} \epsilon_e < 1$. Furthermore, for all subsets $E_1, E_2 \subseteq E$ we have $\sum_{e\in E_1} \epsilon_e \neq \sum_{e\in E_2} \epsilon_e$ if and only if $E_1 \neq E_2$.

Let $w'_e : E \to \mathbb{Z}_+$ be *integer* weights on the edges of G. Then with

$$w : E \to \mathbb{R}_+, \quad w_e := w'_e + \epsilon_e \tag{19}$$

different paths have different lengths and, thus, every shortest path is unique. This follows directly from $\sum_{e\in P} \epsilon_e < 1$ for all paths $P \subseteq E$, $\sum_{e\in P_1} \epsilon_e \neq \sum_{e\in P_2} \epsilon_e$ for all distinct paths $P_1, P_2 \subseteq E$, and the integrality of w'. Note that the integer weights w' are the variables in this mixed–integer programming model. The perturbed weights w are "only" artifical variables.

Given the perturbed routing weights for the edges of the supply graph, we can easily determine whether, for these weights, there exists a feasible solution satisfying all side constraints. From inequalities (10)–(12), (15), and (16) we obtain the routing paths and with (14) a lower bound for the capacity of each supply edge. We have to validate four constraints: The routing paths in the normal operating state must not contain more that ℓ edges (13), the number of chosen supply edges must be exactly m (7), the average length of these edges may not exceed λ_{max} (8), and, finally, the lower bound for the capacity of each supply edge may not exceed c_{max} (6). If these constraints are satisfied there exists a solution with these routing weights. The capacities of a cheapest such solution can easily be computed from the routing paths.

4 Algorithmic Approach

In this section we describe several starting and improvement heuristics that utilize the observation made at the end of the previous section. Given (perturbed) routing weights for the supply edges, it is easy to decide whether these weights, together with the induced routing paths and capacities, define a feasible solution. In the starting heuristics we try to assign such weights to the supply edges from scratch, while in the improvement heuristics we iteratively exchange or modify these weights to reduce the induced cost.

4.1 Starting Heuristics

We employ a two step approach to compute an initial feasible solution. First we choose m supply edges with average length less or equal to λ_{max} that induce a 2-node-connected subgraph containing all nodes of the supply graph. Thereafter, in the second step, we assign weights to the supply edges in such a way, that only these m edges are used to route the demands. If the induced routing paths and capacities satisfy all constraints we have found a feasible solution. Otherwise, the starting heuristic fails.

Algorithm 1 Starting Heuristic

```
choose initial topology
set initial routing weights
compute routing paths and capacities
if all conditions satisfied then
  return solution
else
  starting heuristic failed
```

4.1.1 The initial topology

To compute an initial topology, we implemented three methods, one randomized and two deterministic ones.

Random topology In the first method we iteratively select m supply edges at random until we obtain a 2-node-connected subgraph containing all nodes of G, whose average edge length is at most λ_{max}. Applying this method several times, we compute in very short running time many different topologies to continue with.

Tour based topology The idea, here, is to compute heuristically a Hamiltonian cycle with a double tree heuristic or with Christofides' heuristic (see [7]) and to add further edges until the induced subgraph contains m edges. Since we start with a Hamiltonian cycle, the final subgraph is 2-node-connected and contains all supply nodes. To increase the probability that an edge $e \in E$ is chosen if there is a high demand between its end-nodes u and v we define artificial edge costs $c_e := 1/\max(d_{uv} + d_{vu}, 1)$ for the Hamiltonian cycle computation. In the second step we add the cheapest $m - |V|$ remaining edges with respect to these costs. If the final subgraph does not satisfy the average length restriction we re-start this method using the physical edge lengths as artificial costs.

Delete heuristic topology In the third method to compute an initial topology we start with the entire supply graph and iteratively delete edges until there are exactly m edges left. In every iteration we try to delete one of the remaining edges. If, after its deletion, the edges left over do not induce a 2-node-connected subgraph containing all nodes, we put this edge back and label it as unremovable. Whenever we cannot remove any further edge we start a backtracking procedure that re-inserts the edge deleted last and reverts all labels set after its deletion. We also invoke the backtracking procedure if we end up with m edges that do not satisfy the average length restriction.

The final topology depends on the order the edges are considered for deletion. In the beginning and after each successful deletion we (re-)compute the capacities that are induced by (non-perturbed) unit routing weights on the remaining edges (and "infinite" weights on the deleted edges). In each iteration we try to delete the edge with the smallest of these capacities.

4.1.2 The initial routing weights

Given a feasible network topology, we initially assign a perturbation of the unit weights to its edges. This approach has the advantage that every demand is always routed on a shortest path with respect to the number of edges, a property, which is appreciated by network operators.

With the perturbation technique described in Section 3, such routing weights only depend on the permutation of the supply edges (see (18) and (19)). Given a set $F \subseteq E$ of m supply edges whose average length is less or equal to λ_{max} and that induce a 2-node-connected subgraph of G containing all nodes, we first choose a permutation $\sigma : F \to \{1, \dots, m\}$ of these edges. Then we extend this permutation canonically to a permutation $\sigma' : E \to \{1, \dots, |E|\}$ of all supply edges, with $\sigma(e) = \sigma'(e)$ for all $e \in F$, and set the routing weights as follows:

$$w_e := \begin{cases} 1 + 2^{\sigma'(e)-|E|-1} & e \in F, \\ m + 2^{\sigma'(e)-|E|-1} & \text{otherwise.} \end{cases}$$

Since $w_{e'} \geq m + 2^{m-|E|} > \sum_{e \in F} w_e$ for all $e' \in E \setminus F$ and (V, F) is a 2-node-connected spanning subgraph of G, no edge in $E \setminus F$ is contained in any routing path in any operating state. Hence, the subgraph of G induced by the routing paths with respect to these weights is indeed (V, F). Although in practice only the edges in F are considered in the routing path computations, we set the routing weights for the other edges here according to the model we developed in Section 3.

In our implementation we use the following three methods to choose a permutation of the edges in F and to define the routing weights.

Random permutation In the first method we choose a permutation of the edges in F at random. If the induced capacities do not exceed the capacity bound c_{max}, we have found a feasible solution. We repeat this method several times and choose the cheapest solution found.

Demand based permutation To reduce the probability, that the capacities induced by the initial routing weights are larger than c_{max}, we order the edges in F in increasing order of the demands between their end-nodes. Using this permutation, edges with higher demands between their end-nodes get larger routing weights than those with smaller demands. Hence, if there is more than one path with the same minimal number of edges between the end-nodes of a demand, we select the path, for whose edges the largest of these demands is minimal.

Capacity based permutation Similar to the previous perturbation method, we again try to balance the flow on the chosen edges. First, we choose for each operating state and for each demand one of the shortest paths with respect to the number of edges in (V, F). Using these paths to route the demands, we obtain capacities, by which we order the edges in F increasingly.

4.2 Improvement Heuristics

In this subsection we present several improvement heuristics devised to solve the IP-ND problem, which are based on local search techniques. Given a feasible initial solution, we iteratively exchange or modify the routing weights to reduce the costs caused by the induced capacities.

In principle, a local search algorithm is given by a neighborhood and a rating function. The neighborhood function assigns to each solution a set of solutions, that, usually, can be obtained from this solution by a simple modification. The rating function, which assigns some (maybe artificial) costs to each solution, is used as a measure of quality of the solutions. Given a neighborhood and a rating function, a generic local search algorithm works as follows: Starting with the initial solution, in each iteration the algorithm scans the neighborhood of the current solution for a neighboring solution with the best rating, which becomes the current solution in the next iteration. This process is repeated until there is no solution with a better rating than the current solution in the neighborhood.

Algorithm 2 Improvement Heuristic

Require: N — neighborhood function
Require: c — rating function
Require: sol_{init} — initial solution
  $sol := sol_{init}$
  repeat
    $sol_{best} := sol_{last} := s$
    for all $sol' \in N(s)$ **do**
      if $c(sol') < c(sol_{best})$ **then**
        $sol_{best} := sol'$
    $sol := sol'_{best}$
  until $sol_{best} = sol_{last}$ **or** $time\_limit\_exceeded$

To solve the IP-ND problem, we implemented two classes of neighborhood functions and three different rating functions.

4.2.1 The neighborhood functions

The switching neighborhoods Given a solution, we define its k-switching neighborhood as the set of all feasible solutions that can be obtained by exchanging the routing weights of at most k supply edges. Note that exchanging the weights assigned to chosen and non-chosen edges may change the topology of the induced solution. Since a small number of changes to the routing weights causes only few routing paths to change and these changes are easy to compute, the switching neighborhoods can be implemented efficiently.

If the routing weights of the initial solution are perturbed unit weights, which implies that all routing paths are minimal with respect to the number of edges in the current topology, so are the weights of the solutions considered by these neighborhoods.

The modification neighborhoods In contrast to the switching neighborhood, we modify, here, the edge weights depending on the routing paths and the capacities of the current solution.

The l, k-modification neighborhood of a given solution is defined as follows: In the first step, we compute l assignments of routing weights by modifying the weights of the current solutions. Thereafter, in the second step, for each of these l assignments we consider all weight exchanges defined by the k-switching neighborhood. The l, k-modification neighborhood consists of all feasible solutions induced by the weights obtained this way. Practical experiments revealed that the performance of our algorithms improved significantly when the weight modification methods were combined with the weight switching neighborhoods.

In our implementation we use three different kinds of weight modification methods:

Load — reduces the routing weight on edges with small load (average flow over all operating states / capacity) to use free capacities,

Capacity — reduces the routing weight on edges with small capacity to increase the usage of low-capacity edges and decrease the usage of high-capacity edges, and

Violation — increases the routing weight on edges whose current capacity is larger than the capacity in a given reference network.

In contrast to the switching neighborhoods, it may happen that in the solutions computed with these neighborhoods the routing paths are not shortest paths with respect to the number of edges if we have perturbed unit routing weights in the initial solution.

4.2.2 The rating functions

To decide, which of the solutions in the current neighborhood is chosen for the next iteration in the local search heuristic, the following three rating functions have been implemented:

Cost — $cost(solution)$, i.e., the solutions are rated by their cost,

Capacity — $\sum_{e \in E} y_e$, i.e., the solutions are rated by their total capacity, and

Violation — $\sum_{e \in E} max\{y_e - c_e, 0\}$ for given capacities $c_e \in \mathbb{R}_+$, i.e., the solutions are rated by their total violation of the capacities of a given reference network.

The first two rating functions can be applied to reduce the cost of the solution—they evaluate the solutions directly by their cost or indirectly via the total capacity. The last rating function, together with a violation based modification neighborhood, is used in routing heuristics, which try to find routing weights that allow a feasible routing in a given network.

5 Computational results

In this section we report on some computational experiments with real-world data provided by our partner DFN. The current backbone network of DFN contains ten central service switches defining the node set of the complete supply graph. We wish to select either 12 or 13 of its 45 supply edges as cheap as possible for different parameter settings. The directed demands between the central service switches are peak demands obtained from IP–accounting over three consecutive months. Over these months the demands' ranges are $[0.05, 23.75]$, $[0.07, 25.02]$, and $[0.07, 27.14]$ MBit/s, respectively. The capacities available for the supply edges are 34 Mbit/s plus a multiple of the unit capacity of 2 Mbit/s, the upper capacity bound is 155 MBit/s. In the reported computations the length restriction for the routing paths in the normal operating state is set to $\ell = 4$.

Topology	Weights	$m = 12$		$m = 13$	
		Cost	Time (sec)	Cost	Time (sec)
Random	Random	14.02	09.39	13.64	05.47
(50×)	Demand	14.36	08.57	12.70	05.50
	Capacity	13.31	12.54	12.11	09.24
Tour	Random	16.58	00.34	13.50	00.32
	Demand	14.00	00.34	14.35	00.33
	Capacity	13.47	00.42	13.27	00.41
Delete	Random	**12.47**	**02.96**	**11.69**	**02.84**
	Demand	**12.50**	**02.98**	**11.44**	**02.86**
	Capacity	**12.50**	**03.01**	**11.55**	**02.88**

Table 1: Starting heuristics (month 3, $r^0 = 2.0$, $r^s = 1.0 \; \forall s \in S \setminus \{0\}$), best three solutions in bold face

The algorithms described in the previous section have been implemented in C++ using the library LEDA [6]. The computations were performed on a Sun Ultra Enterprise 3000 (Ultra-SPARC processor at 168 MHz), the peak memory usage was 15 MB.

In Table 1 we compare the solution values and running times for the starting heuristics applied to the last demand set (month). We report the results only for one setting of the reservation parameters. In the normal operating state we oversize the network capacities by a factor of two ($r^0 = 2.0$) and in the failure states we wish to route at least 100 percent of each demand ($r^s = 1.0$ for all $s \in S \setminus \{0\}$). The experiments revealed that the delete method to compute an initial topology clearly outperforms the others, no matter which routing weight assignment we employ in the second phase of the starting heuristics. This result was obtained for all other settings of the reservation parameters and all other demand sets, too.

We run several local search heuristics to improve the best solution found by each of the starting heuristics in order to find the best combination. Results for different neighborhoods and demand sets are shown in Table 2. In contrast to the results obtained for the starting heuristics, no combination clearly outperformed the others. As one might expect, it is more difficult to improve initial solutions based on a random topology, probably because these do not take advantage of the demand structure.

The running times of our heuristics are fairly good, considering the size of the problems. The starting heuristics compute feasible solutions within a couple of seconds. Keeping in mind that the network is re-designed every 2–3 months only, the running times reported for the combinations of starting and improvement heuristics are also satisfactory, even for larger neighborhoods.

Finally, to evaluate the impact of the reservation parameter on the best solution value we run further test series. We see from Table 3 that there is a clear trade-off between the cost and the quality of the resulting solution. Whether we allow arbitrary or only perturbed unit routing weights makes no big difference. It is up to the network designer to compare the different scenarios and to choose a solution that fits additional design requirements best.

Heuristic	Demands					
	Month 1		Month 2		Month 3	
	Cost	Time	Cost	Time	Cost	Time
Random (50×)						
+ 2-switch	10.69	32	11.22	48	11.69	42
+ 2,10-mod. (load)	9.95	13:29	11.22	22:38	11.07	20:41
+ 2,10-mod. (cap)	9.96	10:22	11.22	21:43	10.88	13:28
+ 3-switch	10.20	21:50	11.00	12:08	11.57	17:09
+ 3,10-mod. (load)	10.20	2:16:25	**10.35**	**6:35:27**	11.05	6:24:04
+ 3,10-mod. (cap)	9.88	3:54:17	10.57	2:12:34	11.18	3:55:02
Tour						
+ 2-switch	10.47	57	10.74	42	11.67	42
+ 2,10-mod. (load)	**9.85**	**23:19**	10.74	13:02	11.08	19:36
+ 2,10-mod. (cap)	10.47	5:22	10.71	6:42	10.88	13:28
+ 3-switch	9.95	19:28	10.64	13:45	10.84	21:40
+ 3,10-mod. (load)	9.90	2:10:44	10.57	3:48:52	**10.77**	**2:58:27**
+ 3,10-mod. (cap)	9.95	4:48:12	10.64	2:07:11	**10.61**	**6:38:29**
Delete						
+ 2-switch	10.10	37	11.00	47	11.37	48
+ 2,10-mod. (load)	**9.78**	**22:36**	**10.35**	**25:58**	10.86	13:37
+ 2,10-mod. (cap)	9.90	18:28	10.81	06:03	11.01	09:32
+ 3-switch	10.15	14:03	10.56	13:15	11.37	12:30
+ 3,10-mod. (load)	**9.88**	**4:49:08**	**10.35**	**4:33:49**	10.83	6:21:17
+ 3,10-mod. (cap)	9.95	3:24:09	10.56	1:59:05	**10.73**	**5:53:42**

Table 2: Starting + improvement heuristics ($m = 13$, $r^0 = 2.0$, $r^s = 1.0\ \forall s \in S \setminus \{0\}$), times in h:min:sec, best three solutions in bold face

6 Conclusions

In this article we introduced a mixed-integer linear programming formulation for a survivable capacitated network design problem. To our knowledge this is the first time that topology planning, capacity selection, survivability, and the OSPF routing protocol have been integrated and solved within one model.

We have shown that our heuristics produce solutions that satisfy all constraints in reasonable running times. However, to evaluate the quality of the solutions, we still need a method to compute good lower bounds to the optimal solution value.

From a practical point of view, there are at least two interesting extensions to our model. First, a virtual circuit layer that does not necessarily identify virtual paths and virtual circuits and, second, the incooperation of different traffic types or IP–service classes in this model.

7 Acknowledgments

We would like to thank DFN and BMBF for funding this project and our colleagues at DFN, in particular Dr. G. Hoffmann, K. Ullmann, and S. Schweizer, for fruitful discussions and

r^0	r^s	Best solution	
		pert. unit weights	arb. weights
2.0	1.0	11.53	11.31
2.0	0.8	10.94	10.79
2.0	0.0	10.33	9.72
1.0	1.0	10.74	10.66
1.0	0.8	9.17	9.17
1.0	0.0	8.14	8.07

Table 3: Impact of the reservation parameter setting (month 3, $m = 12$)

excellent cooperation.

References

[1] D. Alevras, M. Grötschel, and R. Wessäly, *Capacity and survivability models for telecommunications networks*, Tech. Report SC 97-24, Konrad-Zuse-Zentrum für Informationstechnik, Berlin, 1997.

[2] A. Frank, Handbook of Combinatorics, ch. 2, Connectivity and network flows, North-Holland, 1995.

[3] M. Gerla, Protocols and Techniques for Data Communication Networks, ch. 4, Routing and Flow Control, Prentice-Hall, 1981.

[4] G. Hoffmann, *B-WiN – the ATM-based high-speed network for the DFN community*, Computer networks and ISDN Systems (1996), no. 28, 1953–1960.

[5] U. Huckenbeck, *Extremal paths in graphs*, Akademie Verlag, 1997.

[6] *LEDA—Library of Efficient Data types and Algorithms*, developed at the Max-Plank-Institut für Informatik Saarbrücken, `http://www.mpi-sb.mpg.de/LEDA/leda.html`.

[7] G. Reinelt, *The traveling salesman*, Springer, 1994.

[8] A.S. Tanenbaum, *Computer networks*, 3 ed., Prentice-Hall, 1996.