Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

SVEN O. KRUMKE
JÖRG RAMBAU
DIANA POENSGEN
ANDREAS TUCHSCHERER

Optimization of optical networks with dynamic routing
Milestone 4

ZIB-Report 02-XX (September 2002)

OPTIMIZATION OF OPTICAL NETWORKS WITH DYNAMIC ROUTING MILESTONE 4

SVEN O. KRUMKE [1,2], DIANA POENSGEN [1], JÖRG RAMBAU [1], AND ANDREAS TUCHSCHERER [1]

1. ISSUES OF MILESTONE 4

Our work in the last six months focused on three major topics: the development of more sophisticated algorithms, the possibility to implement our algorithm CBL in a distributed fashion, and last but not least, the generation of requests with finite demands and all necessary extensions of the simulation tool.

In Section 2, we present two new algorithms. The first of these, called *Disjoint Lightpath Decrease* (DLD) strives to utilize a measure of the 'fitness' of the network which is more accurate than that of our previous algorithm CBL and which can be computed efficiently via solving a MAXIMUM FLOW PROBLEM. A small example in Section 2.1 illustrates a weakness of our previous algorithms: none of them makes use of the information of the start and stop time of a call. This is overcome by the second algorithm, *Anticipating* DLD. Moreover, we introduce an extension of our algorithms which takes into account the expected frequency of a call as indicated by an average demand matrix.

In Section 3, we show how our algorithm CBL could be implemented efficiently in a distributed fashion. All key ingredients for a distributed version of CBL are explained in detail. In particular, the discussion sheds light on the general issues which have to be addressed when designing distributed routing algorithms.

Section 4 contains a description of the simulation tool's new features. As mentioned above, the major progress in this respect is the ability of randomly generating requests with finite durations. Moreover, the algorithms had to be extended accordingly, involving a more elaborate administration of the network's resources.

The description and results of extensive simulation studies are given in Section 5. The settings were chosen in accordance with our project partner such as to enable a comparison of the experiments. In particular, we used the same procedures to generate random call sequences based on a given traffic demand matrix as well as a topology provided by our project partner. This topology is composed of the 17-nodes network introduced in previous reports and a dimensioning based on a shortest path routing of static demands. In order to compare the different algorithms also on topologies which do not favor shortest path routings, we also consider two other dimensionings of the same network.

[1]Konrad-Zuse-Zentrum für Informationstechnik Berlin, Department Optimization, Takustr. 7, D-14195 Berlin-Dahlem, Germany.
Email: {krumke,poensgen,rambau,tuchscherer}@zib.de

[2]Research supported by the German Science Foundation (DFG, grant Gr 883/5-3)

Finally, Section 6 provides conclusions with respect to the whole project: we give an overview of the results of our work and discuss which subjects provide fertile ground for future research.

2. New Algorithms for Call-Admission

2.1. **The Algorithm DLD.** The idea upon which all our practical routing algorithms are based is to route a given call in such a way that the resulting network status allows for a maximum potential profit of future calls. Since we do not know the calls to come, we can only relate potential future profit at any point in time to the current network status. Therefore, we want to assign a 'fitness' level to a given network status, and the routing of a call should lead to a smallest possible decrease in the fitness of the network. The key task is to find a good measure for network fitness.

The algorithm CBL introduced in a previous report uses the number of currently available lightpaths as a measure for the fitness of the network. Hence, for each lightpath on which a given request can be routed, CBL computes the total number of lightpaths which are currently free but would become blocked if this lightpath was realized. Then, it chooses the routing choice with the smallest number of newly blocked lightpaths.

However, these newly blocked lightpaths are in general not arc disjoint. In particular, if the actual load in the network is small, their number will be huge, and many of them will intersect. Therefore, it would not be possible to accept and implement all of them simultaneously. Hence, counting all available lightpaths might not be the best of measures for the potential future profit or the fitness of the network.

This drawback motivated the definition of a new algorithm, called *Disjoint-Lightpath-Decrease* (DLD). Instead of taking into account all available lightpaths between any two nodes, DLD only computes for each pair of distinct nodes s and t the maximum number of *arc-disjoint* (s,t)-lightpaths which are still available. More precisely, DLD gets as input a digraph $D = (V, A)$, the set of wavelengths $W := \{\lambda_1, \ldots, \lambda_\chi\}$, a subset $W(a) \subset W$ containing the first χ_a wavelengths in W for each arc $a \in A$, and a sequence $\sigma = (\sigma_1, \sigma_2, \ldots)$ of requests, where each σ_j specifies a pair of distinct nodes $u_j, v_j \in V$ to be connected from $t_j^{\texttt{start}}$ until $t_j^{\texttt{stop}}$. Given the current network status, we denote by $d(s,t,\lambda)$ be the maximum number of disjoint (s,t)-lightpaths which use wavelength λ and which are still available. For any currently free lightpath (P,λ), let $d^P(s,t,\lambda)$ be the analogous value if (P,λ) were additionally routed. Furthermore, we define for each wavelength λ and each lightpath (P,λ) using this wavelength

$$d(\lambda) := \sum_{s \in V} \sum_{t \in V \setminus \{s\}} d(s,t,\lambda)$$

and

$$d^P(\lambda) := \sum_{s \in V} \sum_{t \in V \setminus \{s\}} d^P(s,t,\lambda).$$

DLD uses $d(\lambda)$ as measure of possible future profit gained by calls routed in wavelength λ in the following way. Upon arrival of a call σ_j, it computes for each available lightpath (P,λ) connecting u_j and v_j its current cost c_{DLD} defined by

$$c_{\mathsf{DLD}}(P,\lambda) := d(\lambda) - d^P(\lambda).$$

Then, DLD chooses among all available lightpaths one with minimum cost.

Before we turn to the question how the values d and d^P can be computed efficiently, let us mention that, in general, not all of those lightpaths counted in $d(\lambda)$ are arc-disjoint since lightpaths connecting different pairs of nodes may still overlap. But much fewer than counted by CBL will intersect, since the considered (s,t)-lightpaths are arc-disjoint for each pair of distinct nodes $s,t \in V$. At the end of this section we describe an approach in which the network fitness is measured by the maximum cardinality of a set of available lightpaths such that any two lightpaths in that set are arc-disjoint.

The implementation of the algorithm DLD works as follows. At any time, for each pair of distinct nodes s and t and each wavelength λ the maximum number of currently available arc-disjoint (s,t)-lightpaths is memorized. Each of those values $d(s,t,\lambda)$ can be computed efficiently via solving an instance of the well-known MAXIMUM FLOW PROBLEM, defined by the digraph D with source s, sink t, and arc capacities

$$\kappa_a := \begin{cases} 1, & \text{if } \lambda \text{ is currently available on arc } a, \\ 0, & \text{otherwise.} \end{cases}$$

A vector $x = (x_a)_{a \in A} \in \mathbb{R}^A$ is called an (s,t)-*flow* if it fulfills the *capacity constraints*

$$0 \leq x_a \leq \kappa_a \text{ for all } a \in A$$

and the *flow conservation constraints*

$$\sum_{a \in \delta^-(v)} x_a = \sum_{a \in \delta^+(v)} x_a \text{ for all } v \in V \setminus \{s,t\},$$

where $\delta^-(v) := \{(u,v) \in A\}$ and $\delta^+(v) := \{(v,u) \in A\}$. The *value* of a flow x is defined by

$$\text{val}(x) := \sum_{a \in \delta^+(s)} x_a - \sum_{a \in \delta^-(s)} x_a.$$

The task of the MAXIMUM FLOW PROBLEM is to find an (s,t)-flow whose value is maximum.

[AMO93] contains a proof that the value $d(s,t,\lambda)$ can indeed be derived as the optimal solution of the MAXIMUM FLOW PROBLEM defined as above. Efficient algorithms to solve the MAXIMUM FLOW PROBLEM are known: for arbitrary capacities, the Goldberg-Tarjan algorithm solves the problem with running time $O(|V|^3)$ and even faster using sophisticated data structures [AMO93, Chapter 7]. If all arc capacities are either 0 or 1, the running time can be improved to $O(\min\{|V|^{2/3}|A|, |A|^{3/2}\})$ (cf. [AMO93, Chapter 8]).

This shows that the numbers $d(s,t,\lambda)$ can be computed efficiently. Recall that DLD also needs for the computation of the current cost $c_{\text{DLD}}(P,\lambda)$ of a routing choice (P,λ) the values $d^P(s,t,\lambda)$ for each pair of distinct nodes $s,t \in V$. These can be computed by solving a slightly modified instance of the MAXIMUM FLOW PROBLEM from above: the capacities are now defined as $\kappa_a := 1$ if and only if λ is currently available on arc a and $a \notin P$, and $\kappa_a := 0$ otherwise. Hence, for each routing choice, DLD has to solve $|V| \cdot (|V| - 1)$ instances of the MAXIMUM FLOW PROBLEM in order

to compute $c_{\mathsf{DLD}}(P, \lambda)$. That is, the total running time for this task is $O(|V|(|V|-1) \cdot \min\{|V|^{2/3}|A|, |A|^{3/2}\})$.

Table 3 at the end of Section 5 contains the running times of several algorithms, including DLD, on some instances generated for our simulation experiments.

The different cost functions used by CBL and DLD show different dependences of the load in the network: whereas c_{CBL} decreases with increasing network load, c_{DLD} does not show any continuous trend. Table 1 shows the development of the costs of 15 successively routed lightpaths for both algorithms. The underlying topology is our project partner's 17-nodes network with one available wavelength (see Appendix). At the beginning the network is empty. Since the costs of lightpaths usually increase with their length, we only considered single arc lightpaths in order to get comparable values.

Lightpath	c_{CBL}	c_{DLD}
$(9,5)$	3850	3
$(13,10)$	4540	48
$(12,16)$	4288	39
$(8,3)$	3699	34
$(2,1)$	1729	30
$(9,3)$	2339	30
$(3,9)$	1225	8
$(9,11)$	1562	33
$(16,12)$	541	2
$(9,4)$	962	7
$(11,10)$	509	51
$(16,11)$	431	9
$(14,12)$	170	30
$(5,11)$	314	72
$(5,16)$	156	23

TABLE 1. Development of the costs c_{CBL} and c_{DLD}

The clear trend of CBL's cost function to decrease with an increase in routed lightpaths is explained easily: if the current load in the network is small, the routing of a lightpath will destroy a big number of previously available lightpaths. If the network is heavily loaded, the number of available lightpaths is small, hence the reduction of available lightpaths caused by the routing of a call must be small, too.

On the other hand, DLD's cost values seem to vary independently of the network load. The same was observed by us when computing the different cost values of one fixed lightpath at different routing stages of a call sequence. In order to better understand c_{DLD}'s behavior, note that there might be many sets of arc-disjoint (s,t)-lightpaths of the same cardinality for a given pair s, t of network nodes. Hence, even if an arc of a newly implemented lightpath (P', λ) is contained in one of the arc-disjoint (s,t)-lightpaths of a maximum cardinality set, there can still be a different set of the same cardinality all of whose lightpaths do not intersect P'. In this case, for this pair s, t of nodes, the number $d(s,t,\lambda)$ does not change with the implementation of (P', λ).

In our opinion it is due to this 'covering' by multiple sets that the values $d(s,t,\lambda)$ and hence the values $d(\lambda)$ do not change by much, and the same holds for $d^P(s,t,\lambda)$ and $d^P(\lambda)$ for a fixed lightpath (P,λ).

Of course, the load of the network must have some influence also on these values: with a decreasing number of available lightpaths overall, also the d-values must decrease. But in our opinion, this factor becomes only dominant when the network load is very high.

The fact that the absolute values taken by DLD's cost function are much smaller than those of CBL shows that, for a given pair (s,t), a great number of the (s,t)-lightpaths counted by CBL do indeed intersect. Furthermore, it reveals that there must be a large number of different sets of arc-disjoint (s,t)-paths, and since the connectivity of the considered network is not too high, many of them must have the same cardinality.

We also want to point out another advantage of DLD's cost function when compared to that of CBL. So far, we did not consider any rejection criteria, although the algorithms are in principle allowed to reject calls even if they could be routed. A straightforward idea for a rejection criterion is to relate the profit of a call to the decrease in network fitness its implementation would cause, that is, to its cost. But if the cost is depending on the load of the network as in CBL's case, a value for the ratio of profit to cost of a call which shall be used as a rejection/acceptance threshold must also depend on the load of the network. For DLD, one such threshold value could be used independently of the load in the network.

An Extension of DLD using Multicommodity Flows. In this section, we look at a promising approach which measures the potential future profit by computing the cardinality of a set of available lightpaths in which any two lightpaths are arc-disjoint, and such that the set has maximum cardinality. Again, this value can be obtained as the optimal solution value of a flow problem. But since we do not restrict ourselves to one fixed pair of distinct nodes s and t whose lightpaths must be arc-disjoint but allow the set to contain lightpaths between different pairs of nodes, we cannot express it by an (s,t)-flow. Instead, it must be represented by flows between many pairs of nodes ('commodities') which exist in the network simultaneously (and therefore compete for the network resources). This leads to the MAXIMUM MULTICOMMODITY FLOW PROBLEM, which is defined as follows.

The input is a digraph $G=(V,A)$ with arc capacities $\kappa_a \in \mathbb{R}_+$ and a set $B \subseteq V \times V$ of pairs of nodes from V. The task is to find a vector $(x^b)_{b\in B}$, where x^b is an (s,t)-flow in G for each pair $b=(s,t) \in B$, such that in addition to the flow conservation constraints for each flow the *joint capacity constraints* $\sum_{b\in B} x^b(a) \leq \kappa_a$ for all $a \in A$ hold, and the total flow value $\sum_{b\in B} \mathrm{val}(x^b)$ is maximum.

In our application, the cost of a lightpath (P,λ) is defined as the difference of the optimal value of the instance of the MAXIMUM MULTICOMMODITY FLOW PROBLEM defined by the network status before realizing (P,λ) and the optimal value of the instance defined by the network status which results if (P,λ) is additionally routed. We proceed to define the instance in dependence of a given network status. Again, the capacities κ_a are defined as in the MAXIMUM FLOW PROBLEM:

$$\kappa_a := \begin{cases} 1, & \text{if } \lambda \text{ is currently available on arc } a, \\ 0, & \text{otherwise.} \end{cases}$$

However, there are different possibilities to select the set of demand arcs B. Be aware that a demand arc corresponds to a pair of nodes which are allowed to appear as end nodes of a lightpath. Choosing $B := \{(u, v) \in V \times V \mid u \neq v\}$ will always yield an optimal value equal to the number of arcs on which λ is still available (the set of single links clearly contains only arc-disjoint lightpaths and it must be maximum). Hence, the cost of a lightpath is minimum if and only if it is a shortest lightpath with regard to its number of arcs. In the case of non-uniform traffic distribution, B should be reduced to the connection pairs with positive traffic demand, or even to such pairs whose traffic demand is not too small. In doing so, it has to be taken into account that traffic is divided among all available wavelengths. We can also add constraints which allow no flow values greater than the corresponding demand. But these constraints will rarely be tight.

The MAXIMUM MULTICOMMODITY FLOW PROBLEM as defined above can still be efficiently solved, for instance using a standard LP-solver such as *CPlex*. But we neglected one major constraint necessary to ensure that the optimal solution of the MAXIMUM MULTICOMMODITY FLOW PROBLEM is indeed equal to the maximum cardinality of a lightpath set with the required properties: we must require the flow values to be integral. Unfortunately, adding these integrality constraints, the problem becomes NP-hard [GJ79]. Therefore, it is not possible to solve the MAXIMUM MULTICOMMODITY FLOW PROBLEM with integral flows efficiently. This is the major drawback of the proposed procedure. Nevertheless, using the relaxation which ignores flow integrality constraints might achieve good routing decisions if the solutions do not contain too many fractional flow values. Therefore, we strongly recommend to further pursue this approach.

2.2. **The Algorithm ADLD.** So far, all of the considered routing algorithms make only use of the information which lightpaths are *currently* available and which are not, that is, at the point in time at which a new request arrives. But each request σ_j also specifies its start and expiration times $t_j^{\texttt{start}}$ and $t_j^{\texttt{stop}}$. Therefore, a lightpath unavailable at the time at which a new call σ_j arrives will not be accounted for when computing the cost of a routing choice (P, λ) for σ_j, although it might expire soon and could then be blocked by (P, λ).

The following simple example illustrates why it makes sense to take the start and stop times of a each call into account, too.

Example 2.1. Consider the topology defined by the three-node circle shown in Figure 1, together with one wavelength. For simplicity, we only refer to paths instead of lightpaths in this example. The given network is strongly 2-arc-connected, i.e., there are two arc-disjoint paths connecting each pair of nodes. Moreover, there are no further (s, t)-paths for each pair of distinct nodes s and t. Therefore, DLD and CBL are equivalent if applied to this network. Assume that the algorithm is given a sequence of calls, beginning with $\sigma_1 = (1, 2)$ and $\sigma_2 = (2, 1)$. Let $t_i^{\texttt{stop}} = t_i^{\texttt{start}} + 1$ for $i = 1, 2$, and $t_2^{\texttt{start}} = t_1^{\texttt{stop}} - \epsilon$. That is, the first call ends very shortly after the beginning of the second call.

Upon arrival of the first call $\sigma_1 = (1, 2)$, there are two possible paths $(1, 2)$ and $(1, 3, 2)$ to route the connection request. While the first one intersects the paths $(1, 2)$, $(1, 2, 3)$, and $(3, 1, 2)$, the second intersects $(1, 3)$, $(3, 2)$, $(1, 3, 2)$, $(2, 1, 3)$, and $(3, 2, 1)$.

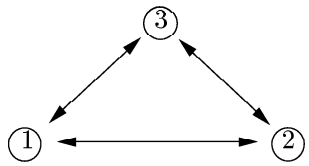


FIGURE 1. Circle with three nodes.

Hence, the costs are $c_{\mathsf{DLD}}(1,2) = 3$ and $c_{\mathsf{DLD}}(1,3,2) = 5$, and DLD realizes the path $(1,2)$. As the second call $\sigma_2 = (2,1)$ arrives, the first connection is still enabled. Let us consider the costs of the routing choices $(2,1)$ and $(2,3,1)$ at the current network status. As above, they intersect three and five paths, respectively. These five paths are $(2,3)$, $(3,1)$, $(2,3,1)$, $(1,2,3)$, and $(3,1,2)$, but two of them contain the arc $(1,2)$ already blocked by σ_1. Therefore, $c_{\mathsf{DLD}}(2,1) = c_{\mathsf{DLD}}(2,3,1) = 3$, and DLD may choose an arbitrary one of them. However, path $(2,1)$ is clearly the better choice: call σ_1 will expire soon (within the next ϵ time units), and then, routing choice $(2,3,1)$ also blocks the two lightpaths $(1,2,3)$ and $(3,1,2)$ until it expires.

There are similar examples in which DLD has only a unique routing choice but which is inferior to a more costly lightpath when considering the durations of the calls. The following variant of DLD, called *Anticipating* DLD (ADLD), aims at taking into account to which extent a routing choice blocks some lightpath during its whole existence. The input of ADLD is a digraph $D = (V, A)$, a set $W(a) = \{\lambda_1, \ldots, \lambda_{\chi_a}\}$ of available wavelengths for each arc $a \in A$, and a sequence $\sigma = (\sigma_1, \sigma_2, \ldots)$ of requests, where each σ_j specifies a pair of distinct nodes $u_j, v_j \in V$ to be connected from $t_j^{\mathtt{start}}$ until $t_j^{\mathtt{stop}}$. The idea of ADLD is as follows. When a call σ_j arrives, some of the lightpaths which are unavailable at that time will be blocked by other previously routed lightpaths until σ_j expires. But some of the currently blocked lightpaths could become available again at some point in time $t^{\mathtt{free}} \in (t_j^{\mathtt{start}}, t_j^{\mathtt{stop}})$, depending on coming routing decisions.

As DLD before, ADLD aspires to measure the availability of arc-disjoint (s,t)-lightpaths which use wavelength λ for any pair of distinct nodes $s, t \in V$. Since this measure now also depends on the starting time $t_j^{\mathtt{start}}$ and expiration time $t_j^{\mathtt{stop}}$ of the given call σ_j, it is denoted by $d_j(s,t,\lambda)$. It takes into account those lightpaths which are newly blocked as well as those which would become available again before $t_j^{\mathtt{stop}}$ if no other connections were established. The latter lightpaths are taken into account according to their fraction of availability during $(t_j^{\mathtt{start}}, t_j^{\mathtt{stop}})$. More precisely, for each arc a and each wavelength λ currently utilized on a, let $t^{\mathtt{free}}(a,\lambda)$ be the time when λ will become available on a again. We define $d_j(s,t,\lambda)$ as the optimal value of the MAXIMUM FLOW PROBLEM $(D, s, t, \kappa_a^{(j)})$ with capacities defined as follows.

$$\kappa_a^{(j)} := \begin{cases} 1, & \text{if } \lambda \text{ is currently available on arc } a, \\ \max\{0, 1 - \frac{t^{\mathtt{free}}(a,\lambda) - t_j^{\mathtt{start}}}{t_j^{\mathtt{stop}} - t_j^{\mathtt{start}}}\}, & \text{if } \lambda \text{ is currently utilized on arc } a. \end{cases}$$

Obviously, $0 \leq \kappa_a^{(j)} \leq 1$ for each arc $a \in A$. Moreover, for an arc a on which λ is currently not available, its capacity $\kappa_a^{(j)}$ is decreasing for increasing $t^{\mathtt{free}}(a,\lambda)$. If

$t^{\texttt{free}}(a,\lambda) \geq t_j^{\texttt{stop}}$, $\kappa_a^{(j)} = 0$. Notice that, different than in the definition by DLD, the above values for κ can be fractional.

Given a network status and a connection request σ_j, note that $d_j(s,t,\lambda) \geq d(s,t,\lambda)$ since $\kappa_a^{(j)} \geq \kappa_a$ for each arc $a \in A$. As for DLD, we define for each wavelength λ

$$d_j(\lambda) := \sum_{s \in V} \sum_{t \in V \setminus \{s\}} d_j(s,t,\lambda)$$

and

$$d_j^P(\lambda) := \sum_{s \in V} \sum_{t \in V \setminus \{s\}} d_j^P(s,t,\lambda).$$

Analogously, the cost function c_{ADLD} is defined by

$$c_{\mathsf{ADLD}}(P,\lambda) := d_j(\lambda) - d_j^P(\lambda).$$

This cost function reflects that the routing decision depends on how many lightpaths become blocked at $t_j^{\texttt{start}}$ and how much lightpaths are prevented from becoming available again later on. Just as for DLD, $|V| \cdot (|V|-1)$ instances of the above MAXIMUM FLOW PROBLEM have to be solved in order to determine the cost of an available lightpath, but as the capacities are no longer 0 or 1, solving each instance takes more time. Again, see Table 3 for the running time of ADLD on selected instances.

Finally, let us reconsider Example 2.1 and see what decisions ADLD would make. For the first call σ_1, ADLD and DLD compute the same costs, since the network is still empty. Recall that $t_1^{\texttt{stop}} = t_2^{\texttt{start}} + \epsilon$ and $t_i^{\texttt{stop}} - t_i^{\texttt{start}} = 1$ for $i = 1, 2$. Upon arrival of σ_2, the capacity of arc $(1,2)$ is

$$\kappa_{(1,2)}^{(2)} = 1 - \frac{t_1^{\texttt{stop}} - t_2^{\texttt{start}}}{t_2^{\texttt{stop}} - t_2^{\texttt{start}}} = 1 - \epsilon,$$

while all other arcs have capacity 1. Table 2 shows the cost defining values $d_2(s,t)$, $d_2^{(2,1)}(s,t)$ and $d_2^{(2,3,1)}(s,t)$ for each pair of distinct nodes s and t (again, the wavelength index is omitted).

(s,t)	$d_2(s,t)$	$d_2^{(2,1)}(s,t)$	$d_2^{(2,3,1)}(s,t)$
$(1,2)$	$2-\epsilon$	$2-\epsilon$	$2-\epsilon$
$(1,3)$	$2-\epsilon$	$2-\epsilon$	1
$(2,1)$	2	1	1
$(2,3)$	2	1	1
$(3,1)$	2	1	1
$(3,2)$	$2-\epsilon$	$2-\epsilon$	1

TABLE 2. Computation of the cost c_{ADLD}.

Hence, the costs of the possible lightpaths are

$$c_{\mathsf{ADLD}}(2,3,1) = 5 - 2 \cdot \epsilon \text{ and } c_{\mathsf{ADLD}}(2,1) = 3.$$

Obviously, the lightpath with minimum cost is $(2,1)$. Note also that a more general form of the cost $c_{\mathsf{ADLD}}(2,3,1)$ is $c_{\mathsf{ADLD}}(2,3,1) = 5 - 2 \cdot \frac{t_1^{\texttt{stop}} - t_2^{\texttt{start}}}{t_2^{\texttt{stop}} - t_2^{\texttt{start}}}$, which shows that

it increases with increasing duration of σ_2 and decreasing overlapping time of the two calls, as desired. On the other hand, the cost $c_{\mathsf{ADLD}}(2,1)$ remains constant.

2.3. **Consideration of the traffic distribution.** Assume that the average traffic demand between each pair of distinct nodes $s,t \in V$ is previously known. This is for instance the case if we are given a static demand matrix based on which the call sequences are generated, as described in Section 4. Clearly, lightpaths connecting two nodes with high average traffic demand are more important than those between two nodes of little demand, and should therefore be protected. In other words, the higher the traffic demand between the end nodes of a lightpath (P',λ), the more it should contribute to the cost of a lightpath (P,λ) for blocking it. This can be taken into account by our algorithms CBL, DLD, and ADLD in the following way. Let $\mathtt{demand}(s,t)$ be the static traffic demand from node s to node t.

As mentioned in the last report, for CBL, the weight of an (s,t)-path $w(P)$ (which is independent of the utilized wavelength) already consists of two factors $w_1(P)$ and $w_2(P)$. In order to adapt the weight function, we only need to introduce a third factor $w_3(P)$ reflecting whether traffic demands are being considered or not. If not, w_3 is constant, meaning that the algorithm is equal to the old version. The functions are defined as follows:

(C) $w_3(P) := 1$,
(T) $w_3(P) := \mathtt{demand}(s,t)$.

We will refer to the resulting new versions of CBL by appending a third index, e.g. the combination of (C) for w_1, (L) for w_2, and (T) for w_3 is denoted by $\mathsf{CBL}_{(\mathtt{CLT})}$.

For DLD and ADLD, we denote the modified versions by $\mathsf{DLD}_{(\mathtt{T})}$ and $\mathsf{ADLD}_{(\mathtt{T})}$. Here, we simply need to redefine the cost determining parameters $d(\lambda)$ and $d^P(\lambda)$ for DLD by

$$d(\lambda) := \sum_{s\in V}\sum_{t\in V\setminus\{s\}} \mathtt{demand}(s,t)\cdot d(s,t,\lambda)$$

and

$$d^P(\lambda) := \sum_{s\in V}\sum_{t\in V\setminus\{s\}} \mathtt{demand}(s,t)\cdot d^P(s,t,\lambda).$$

Furthermore, we redefine

$$d_j(\lambda) := \sum_{s\in V}\sum_{t\in V\setminus\{s\}} \mathtt{demand}(s,t)\cdot d_j(s,t,\lambda),$$

and

$$d_j^P(\lambda) := \sum_{s\in V}\sum_{t\in V\setminus\{s\}} \mathtt{demand}(s,t)\cdot d_j^P(s,t,\lambda)$$

for ADLD. The corresponding costs of a lightpath are computed in the same manner as before, using these modified values. Note that (s,t)-lightpaths with $\mathtt{demand}(s,t)=0$ never yield any cost contribution.

3. Distributed Algorithms

In this section we describe a distributed implementation of the online call-admission algorithm CBL. The implementation has the property that it does not produce a substantial overhead compared to the centralized version.

For more precise statements we need some notation. Let g be a function from $\mathbb{N}$ to $\mathbb{N}$. The set $\mathcal{O}(g(n))$ contains all those functions $f : \mathbb{N} \to \mathbb{N}$ with the property that for constants $c > 0$ and $n_0 \in \mathbb{N}$: $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$. A function f belongs to the set $\Omega(g(n))$, if and only if $g(n) \in \mathcal{O}(f(n))$. The notation $f(n) \in \Theta(g(n))$ means that $f(n) \in \mathcal{O}(g(n))$ and $f(n) \in \Omega(g(n))$. Finally, we write $f(n) \in o(g(n))$, if $\lim_{n\to\infty} f(n)/g(n) = 0$.

All messages sent during the distributed algorithm are of small size (at most $\mathcal{O}(m^2)$ bits in a network with m arcs, but most of the time only $\mathcal{O}(m \log m)$ bits). Once the preprocessing (see Section 3.2) has been accomplished, the distributed information in the network nodes need only be updated if the topology of the network changes substantially. Hence, the outlined distributed algorithm seems to be practical.

We assume that our network is represented by a directed graph $G = (V, A)$. We encode by $n := |V|$ the number of nodes and by $m := |A|$ the number of arcs in the network. For an arc $a \in A$ we denote by $\alpha(a)$ the node where a initiates (the *tail* of arc a) and by $\omega(a)$ the node where a ends (the *head* of arc a).

3.1. **A Trivial Distributed CBL-Algorithm.** Before we give the outline of a distributed algorithm which uses the available parallelism in the network, we would like to note that there is an almost trivial distributed version of CBL which should also perform quite well (perhaps even better) in practice. Suppose that we want to compute a cheapest (s, t)-path in wavelength λ. The trivial version first collects information about the network topology in wavelength λ at s. This can be done by first computing a spanning tree of G rooted at s by the distributed algorithm described in "Optimization of optical networks with dynamic routing: Milestone 3" and then having each node sending the information about its incident links to s along the tree. After this collection stage, the centralized algorithm is run at vertex s. All this can be accomplished by a total number of messages in the network which is $\mathcal{O}(m)$. Each message involves only information about $\mathcal{O}(m)$ links, which can be encoded with $\mathcal{O}(m \log m)$ bits (each of the m links can be given a number in the range $1, \ldots, m$ and such a number can be encoded using $\mathcal{O}(\log m)$ bits).

3.2. **Preprocessing.** We first recall the definition of the (s, t)-prefix tree. The (s, t)-prefix tree is a unique way of representing all (s, t)-paths in the network. It is a rooted tree whose nodes correspond to partial (s, t)-paths starting in s. The root node corresponds to the empty set, its children are the arcs in the network emanating from s. Given a node in the tree representing a partial (s, t)-path $(s, v_1), (v_1, v_2), \ldots, (v_{i-1}, v_i)$, its children are obtained by attaching those arcs (v_i, v_{i+1}) of the network for which there exists a path from v_{i+1} to t and whose attachment does not yield a cycle. The leafs of the tree are exactly the (s, t)-paths in the network.

The (s, t)-prefix tree is an important tool for speeding up the computation of a cheapest (s, t)-path in the algorithm CBL. It can be observed that the prefix trees depend only on the topology of the network. Hence, given a network all $n(n-1) = \Theta(n^2)$ prefix trees can be computed in a preprocessing step. While the corresponding algorithm could in principle also be formulated as a distributed algorithm, there is no major benefit associated with this. Hence, we will assume in the sequel that all prefix trees have been computed. In our distributed setting we have node v in the network

store all $n-1$ (v,t)-prefix trees ($t \in V \setminus \{v\}$). This leads to a distributed storage and reduces the space requirements at each network node.

We note that link failures in the network change the topology. At first glance one might conjecture that a re-computation of all prefix trees will be necessary in each failure situation. However, this is not the case. A link failure on arc a can be treated in our algorithms as the case that a desired wavelength λ is no longer available on a.

3.3. **Computing a Cheapest Lightpath.** In this section we show how the computation of a cheapest (s,t)-lightpath via the (s,t)-prefix tree can be implemented in a distributed network. Our method uses the distributed algorithm COLLECT-PATHS presented in the following Section 3.4 as a subroutine.

Given the (s,t)-prefix tree, a cheapest lightpath connecting s and t can be computed fast as follows. For each wavelength, the tree is traversed in a depth-first manner, starting at the root node. A child and all its descendants need not be considered, if the last arc which has to be attached to reach that child is not available anymore in the considered wavelength. For each node which is passed in the process, its cost is computed. This can be done easily by incrementing the cost of its parent node by the cost of the attached arc.

The depth-first exploration of the (s,t)-prefix tree immediately suggests an idea to implement the branch-and-bound method in a distributed fashion. The prefix tree has a nice locality structure: The successors of node x in the tree which stores the prefix $a_1,\ldots,a_k$ are all nodes y which store a prefix $a_1,\ldots,a_k,a_{k+1}$ where $\omega(a_k) = \alpha(a_{k+1})$. It is possible to identify node x with $\omega(a_k)$ and node y with $\omega(a_{k+1})$. We can handle node x at node $v = \omega(a_k)$ of the network since only local information (the successors of v) is required.

We now describe how to handle node x with prefix $\tau = a_1,\ldots,a_k$ of the (s,t)-prefix tree at node $v = \omega(a_k)$. We will call a vertex $w \in V$ of the network a successor of $v \in V$ in the prefix tree, if node x in the prefix tree has a successor y (in the prefix tree) which stores the prefix τ, a_{k+1} and $w = \omega(a_{k+1})$.

(1) Vertex v considers its successors $v_1,\ldots,v_q$ in its locally stored (v,t)-prefix tree.
(2) If v does not have any successors, then the cost of the prefix τ is returned to the predecessor $\alpha(a_k)$ of v in the (s,t)-prefix tree.
(3) For each successor v_i with the property that the attachment of (v,v_i) to the prefix τ does not produce a cycle, the following steps are performed:
    (a) The cost $c_{\tau,\lambda}(v,v_i)$ of arc (v,v_i) is computed.
    This task is accomplished by the distributed procedure COLLECT-PATHS described in Section 3.4.
    (b) If the cost of τ plus the cost $c_{\tau,\lambda}(v,v_i)$ of the new arc does not exceed the currently best know upper bound, a message is sent to v_i. This message contains the augmented prefix $\tau,(v,v_i)$ with its corresponding cost.
    (c) As soon as v_i has returned a solution, the upper bound is updated (if the new solution value is better than the old upper bound) and communicated to the predecessor $\alpha(a_k)$ of v in the (s,t)-prefix tree.

There are implementations of Steps 3a–3c: a sequential and a parallel one.

In the sequential implementation, each successor v_i is called sequentially, i.e., v_{i+1} is only called after v_i has returned its solution. This sequential implementation has

the same running time as the centralized version of the branch-and-bound-method, if one neglects the time for communication over a link. In fact, this version performs essentially the same steps as the centralized algorithm in the same order. The communication overhead is comparatively small, since at any moment in time at most one message of size $\mathcal{O}(n \log m + nm)$ bits is sent over a link. The bound on the message size is derived by observing that each prefix consists of at most n links, each of which can be specified using an identifier of $\mathcal{O}(\log m)$ bits. The cost of a prefix is at most the length of the prefix (which in turn is bounded by n) times the maximum cost of a single arc. Since each arc intersects at most 2^m paths, the cost of a single arc is at most 2^m which can be encoded using $\mathcal{O}(\log 2^m) = \mathcal{O}(m)$ bits. This leads to the additional term nm in the message size requirements.

In the parallel implementation, each successor v_i is called immediately after the cost of the arc (v, v_i) has been computed. This version traverses *the whole* prefix tree faster than the sequential version. This, however, neglects the bounding scheme (via the best known upper bound) which in the centralized version is a key ingredient for a good performance. In order to have the parallel version take advantage of the bounding, vertex v should proceed as follows. Suppose that the cost of arc (v, v_i) has been computed.h Then, instead of calling v_i immediately, v first updates the currently best known upper bound by sending an update request to the root s of the computation (recall that we are currently searching for (s, t)-paths and that the computation initiates at s where the (s, t)-prefix tree is stored). Then, the computation need not be forwarded to v_i if the cost of the prefix plus that of arc (v, v_i) is above the upper bound. Otherwise, v_i is informed and we proceed to v_{i+1} (without waiting for the answers of v_i, of course). As soon as a child say v_j returns a new upper bound, this bound is communicated back to s.

Its is not clear in advance whether the sequential or the parallel version will actually perform superior in practice. While in principle the parallel version is faster if we are considering the whole prefix tree, it might be the case that the parallel version explores a larger portion of the prefix tree than the sequential one. This is due to the fact that an updated upper bound might arrive at a vertex only after it has already forked off a computation at some of its children.

3.4. **Computation of Arc Costs.** In this section we address the following problem: We are given a prefix $\tau = a_1, \ldots, a_k$ of arcs and a new arc a_{k+1} and we seek to compute the cost of arc a_{k+1} with respect to wavelength λ, defined as

$$c_{\tau,\lambda}(a_{k+1}) := \sum_{\substack{p: a_{k+1} \in p \\ p \cap \{a_1, \ldots, a_k\} = \emptyset \\ \lambda \text{ is free on } p}} w(p). \tag{1}$$

As noted above, the problem of computing $c_{\tau,\lambda}$ arises as a subproblem in the implementation of CBL.

For simplicity we will restrict ourselves to the case that $w(p) = 1$ for each path. Hence, the quantity referred to in (1) equals the number of simple paths which do not contain the arcs $a_1, \ldots, a_k$ and for which wavelength λ is still available.

Algorithm COLLECT-PATHS presented below is a distributed implementation to compute the cost of an arc $a = a_{k+1}$. The whole computation will be initiated as follows:

(1) The set of "forbidden arcs" is defined as $F := \{a_1, \ldots, a_k\}$.
(2) Vertex $\alpha(a)$ behaves as if it received the message "expand $[a, \text{tail}, \text{NULL}]$" from $\omega(a)$.
(3) Vertex $\omega(a)$ behaves as if it received the message "expand $[a, \text{head}, \text{NULL}]$" from itself.

At the end of the distributed computation, the cost of arc a will be available at $\omega(a)$. In the formulation of the algorithm we do not explicitly transmit the set F to the nodes in the network. However, in a real implementation this would be done by appending F to any message sent.

Before we state the algorithm in pseudo-code, we give an informal description. The idea behind the distributed algorithm COLLECT-PATHS is best explained in the most simple scenario. Suppose that we want to count the paths that start with the given arc a. Let $v := \omega(a)$ be the head of arc a and denote by $v_1, \ldots, v_k$ its successors, that is, those vertices v_i in G for which there exists an arc emanating from v and ending in v_i. The sought number is then just the sum $1 + \sum_{i=1}^{k} Q_i$, where Q_i denotes the number of paths which start with the prefix $a, (v, v_i)$. Instead of computing each quantity Q_i at vertex a, we can distribute this task to vertex v_i. This vertex splits the computation of Q_i among its successors in G in the same way (which leads to a recursive process). As soon as v_i has received the answers from its successors, v_i sends the Q_i back to v.

This simple idea forms the basis of our distributed algorithm which is more complex since we do not only have to cover head-children. We use four types of messages in the algorithm:

(1) "expand $[(e_1, \ldots, e_l), \text{head}, \text{id-old}]$"
This message is sent from node $\alpha(e_l)$ to node $v = \omega(e_l)$ and requests that v computes (recursively) all paths of the form $e_1, \ldots, e_l, p'$ where p' is a path starting at v which does not cause any loop. This siltation is illustrated in Figure 2 and corresponds to the simple basic idea of the algorithm outlined above.

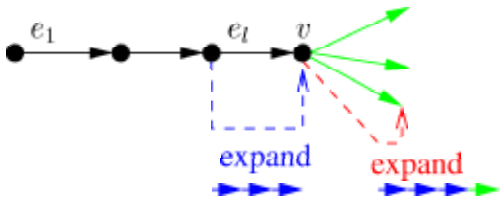


FIGURE 2. Response to an "expand,..., head" request.

(2) "expand $[(e_1, \ldots, e_l), \text{tail}, \text{id-old}]$"
This message is sent from node $\omega(e_1)$ to node $v = \alpha(e_1)$. Similarly to the first message, this is a request for v to compute all paths which can be obtained by appropriate extensions of the path $e_1, \ldots, e_l$. In contrast to the first message type, we have two types of extensions:
(a) Extensions of the form $p', e_1, \ldots, e_l$
This is essentially the symmetric situation to the "expand, ..., head" message in the first message type and is covered by sending out "expand, ..., tail" messages to predecessors.

(b) Extensions of the form $e_1, \ldots, e_l, p''$
These extensions are computed by sending an "expand-from-back $[(e_1, \ldots, e_l), \ldots,$" message to node $\omega(e_l)$.
The whole situation is illustrated in Figure 3.

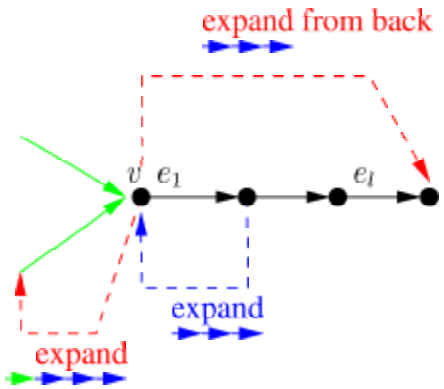


FIGURE 3. Response to an "expand,. . . ,tail" request.

(3) "expand-from-back $[(e_1, \ldots, e_l)$, head, id-old]"
As noted above this message asks node $\omega(e_l)$ to continue a recursive expansion of the path prefix (see Figure 4).

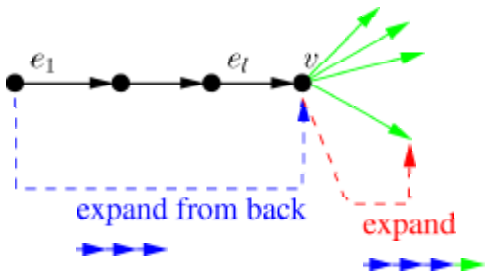


FIGURE 4. Response to an "expand-from-back,. . . " request.

(4) "result x, id"
Each vertex in the network marks each group of recursive expand requests with a unique identifier (id). Hence, it can keep track of which "subproblems" have been completely solved (by its subordinates) and which "subproblems" are still partly open. If the node itself has gathered all necessary information to answer an expand request which arrived from some other node w, it returns the result x with a "result x, id" message.

The pseudo-code of our algorithm COLLECT-PATHS is given below.

COLLECT-PATHS(λ)

1 { *Let v be the node where this algorithm is executed.*
- *λ is the wavelength in which we count available lightpaths.*
- *$E^{\rightarrow}$ denotes the arcs in G leaving v.*
- *$E^{\leftarrow}$ denotes the arcs in G entering v.*
- *$F \subseteq A$ denotes the set of "forbidden arcs".* }

2 **if** message "expand $[(e_1, \ldots, e_l)$, head, id-old]" is received **then**
3 { *in this case we have $v = \omega(e_l)$, see Figure 2* }
4 id := **new** unique id
5 `IDS` := `IDS` $\cup$ {id}
6 `paths`(id) := 1
7 `answer-id`(id) := id-old
8 `answer-to`(id) := $\alpha(e_l)$
9 **for all** $e \in E^{\rightarrow} \setminus (F \cup \{e_1, \ldots, e_l\})$ **do**
10 **if** wavelength λ is available on e **then**
11 `open-answers`(id) := `open-answers`(id) $\cup \{\omega(e)\}$
12 Send message "expand $[(e_1, \ldots, e_l, e)$, head, id]" to $\omega(e)$.
13 **end if**
14 **end for**
15 **end if**
16 **if** message "expand-from-back $[(e_1, \ldots, e_l)$, head, id-old]" is received **then**
17 { *in this case we have $v = \omega(e_l)$ and the message originates from $\alpha(e_1)$, see Figure 3* }
18 id := **new** unique id
19 `IDS` := `IDS` $\cup$ {id}
20 `paths`(id) := 0
21 `answer-id`(id) := id-old
22 `answer-to`(id) := $\alpha(e_1)$
23 **for all** $e \in E^{\rightarrow} \setminus (F \cup \{e_1, \ldots, e_l\})$ **do**
24 **if** wavelength λ is available on e **then**
25 `open-answers`(id) := `open-answers`(id) $\cup \{\omega(e)\}$
26 Send message "expand $[(e_1, \ldots, e_l, e)$, head, id]" to $\omega(e)$.
27 **end if**
28 **end for**
29 **end if**
30 **if** message "expand $[(e_1, \ldots, e_l)$, tail, id-old]" is received **then**
31 { *in this case we have $v = \alpha(e_1)$ and the message originates from $\omega(e_1)$, see Figure 4* }
32 id := **new** unique id
33 `IDS` := `IDS` $\cup$ {id}
34 `paths`(id) := 1
35 `answer-id`(id) := id-old
36 `answer-to`(id) := $\alpha(e_1)$
37 **for all** $e \in E^{\leftarrow} \setminus (F \cup \{e_1, \ldots, e_l\})$ **do**
38 **if** wavelength λ is available on e **then**
39 `open-answers`(id) := `open-answers`(id) $\cup \{\alpha(e)\}$
40 Send message "expand $[(e, e_1, \ldots, e_l)$, tail]" to $\alpha(e)$.

```
41      end if
42    end for
43    open-answers(id) := open-answers(id) ∪ {ω(e_l)}
44    Route message "expand-from-back [(e_1, ..., e_l), head, id]" to ω(e_l).
       { We use the term "route" here to indicate that the receiving node ω(e_l) is not
      a direct neighbor of v in the network. }
45  end if
46  if message "result x, id" is received from w then
47    paths(id) := paths(id) + x
48    open-answers(id) := open-answers(id) \ {w}
49  end if
50  if no message is received then
51    for all id ∈ IDS do
52      if open-answers(id) == ∅ then
53        Send/route message "result paths(id), answer-id(id)" to answer-to(id).
54        remove id from IDS
55      end if
56    end for
57  end if
```

We now address the efficiency of procedure COLLECT-PATHS. Neglecting the communication delay between vertices, COLLECT-PATHS runs at least as fast as its centralized counterpart. In fact, the distributed version should even run faster, since the computation is not only distributed but in addition also parallelized in the network. The size of any "expand" message used in the algorithm is bounded from above by $\mathcal{O}(n \log m)$ bits. The size of the "result" messages can be crudely bounded from above by $\mathcal{O}(m)$ bits. Hence, again each message is extremely small compared to the available bandwidth. The total number of messages is of the order of computed paths $c_{\tau,\lambda}$ which might be exponentially large (theoretically as large as $\Omega(2^m)$). However, these messages are well distributed over the whole network and due to the tiny size of each message the communication between the nodes should not cause any network congestion.

3.5. **Remarks about the Distributed Algorithm.** We have already stated that the distributed implementation of the call admission algorithm CBL requires only messages of small size and that the total number of messages should not cause congestion in the network.

The preprocessing (computation of all (s,t)-prefix trees) could be accomplished at a dedicated central node (for the 17 node sample network this computation was done in less than one second). The distribution of the prefix tree information to the network nodes (each of which stores its own set of prefix trees) causes the largest messages to be transferred. However, this information needs only be updated once the topology of the network changes substantially. Using this setup no computationally demanding preprocessing needs to be carried out at the network nodes.

Neither the computation of the cheapest lightpath nor the calculation of arc costs requires large computations at the network nodes. Both involve mainly the check of the local neighborhood of a node and the transfer of information to those neighboring vertices.

In our outline of a distributed CBL we have neglected the task of actually using the best computed (s,t)-lightpath, that is, the signaling to the network nodes to route the call. This is an easy task, since after the computation the information about the best (s,t)-path is at node s. The routing information can be distributed to the involved nodes along the path starting from s in a straightforward fashion.

At the current stage, it is not clear whether the distributed version of the CBL-algorithm will have major benefits compared to the centralized version. The centralized version requires only an extremely small amount of data to be transferred (see Section 3.1). Moreover, this data does not need to be distributed among *all* network nodes, but only be collected at *one single* node in the network.

On the other hand, the distributed version can take advantage of the fact that many tasks in CBL can be parallelized. In principle this should lead to a speedup. The price to be paid for the parallel execution is the distribution of messages which, though of small size, will cause a slower computation (compared to a theoretical parallel model without link delays). Depending on the link delays either a centralized or a distributed version of CBL might be superior in practice.

4. Advancement of the Simulation Environment

As mentioned earlier, the major progress in the development of the simulation environment is the possibility to generate sequences of calls whose durations are finite. Moreover, topologies of non-uniform dimensioning are now being considered as well.

Topology construction. So far, the only way to construct the static dimensioning for the network topology was to equip each arc with the same set of wavelengths. It is now possible to specify a non-uniform dimensioning by a matrix whose entries indicate for each arc the number of available wavelengths on that arc. The simulation tool is able to read in such a matrix if it has the right input format. A second possibility to obtain a dimensioning for a given network is to give as input a static demand matrix. Based on this matrix, the tool computes a dimensioning as follows: all arcs are provided with the number of wavelengths necessary to route all traffic demands simultaneously via shortest paths. The shortest-path computations are performed with the standard algorithm of *Dijkstra*.

Given for each arc a the number of wavelengths χ_a required on that arc, both new methods allocate the wavelengths in a fixed order in the following way. First, the maximum number of wavelengths needed on a single arc is determined: $\chi := \max_{a \in A} \chi_a$, and the set of wavelengths $W = \{\lambda_1, \ldots, \lambda_\chi\}$ is defined. Then, arc a is assigned the set $\{\lambda_1, \ldots, \lambda_{\chi_a}\}$ for each $a \in A$.

Generation of request sequences. We implemented the generation of random call sequences who specify finite durations in the following way. For each pair of start and end nodes (s,t), one or several *sources* generate calls between s and t using two random variables: the *inter arrival time*, and the *duration*. The inter arrival time specifies the time between the start times of two calls of the same source, whereas the duration naturally determines the difference between start and stop time of a call.

In compliance with our project partner, we use the given static demand traffic matrix to determine the number of sources as follows: for each unit of static demand between two nodes, m connection sources are considered. The parameter m is called

the *multiplex factor*. Each connection source models the arrival of calls as a Poisson input process, that is, the inter arrival times are exponentially distributed – with the restriction that no second call may start while the previous call of the same source is not expired yet. Different connection sources, even with same source s and target t, may have overlapping calls.

The rates of the exponential distributions used to compute the inter arrival times are identic for all connection sources and can be chosen by the user. He also has the choice between constant and exponentially distributed durations, again assuming that different sources use the same mean, which has to be specified. Moreover the number of lightpaths a call requires can be selected to be either constant or a uniformly distributed random variable within an interval to be specified.

Note that by definition, choosing a multiplex factor equal to 1 means that each unit of static demand corresponds to at most one call per source at any moment in time in the dynamic scenario. This means that none of the dynamically generated calls need to be rejected if the topology was dimensioned using the shortest paths routings of the static demands and the same paths are used to route the calls. Moreover, the transition to the dynamic scenario leads to free network capacities, assuming that the ratio of average inter arrival time and mean duration is greater than one, since then not all connections will overlap. Therefore, additional requests can be routed.

These additional requests are generated in a uniform way by multiplying the number of sources with the multiplex factor. As for call sequences with infinite durations, the simulation tool computes for each call sequence the ratio of rejected and generated calls as a measure for the blocking probability. Setting the blocking probability in relation to the multiplex factor therefore allows to observe the network's resilience under increasing load. The following enumeration summarizes the parameters which can be specified by the user:

(1) the number of calls to be generated,
(2) the mean of the inter arrival times,
(3) the distribution of a call's duration,
(4) the mean of a call's duration,
(5) the multiplex factor,
(6) upper and lower bounds on the number of lightpaths a call may require,
(7) a seed value which determines the request sequence uniquely depending on the other parameters.

Visualization. As we now handle connection requests with finite durations, a change of the network status also occurs whenever an implemented connection expires. By choosing a stepwise processing of the sequence of requests, the user can let the tool display each lightpath removal. Furthermore, the actual simulation time is shown at the top of the graph window whenever a new call is accepted or rejected or the connection of an already routed call is closed.

Routing algorithms. See Section 2 for a detailed description of the new algorithms implemented.

Statistical output. The output of statistical data was extended by the quantities below. I_A denotes the set of indices of the accepted calls, while I_R is the set of indices of rejected calls. The actual planning horizon (the simulation time) is denoted by T. For

each call σ_j, let u_j and v_j be the start and end node of the connection, respectively, and for each accepted call σ_j, let $\texttt{length}_j$ be the number of arcs of the chosen lightpath. Moreover, for each pair of nodes u and v, we denote by $d(u,v)$ the minimum number of arcs in a shortest (u,v)-path. Remember that $W(a)$ is the set of all wavelengths which are available on an arc a.

- The *realized traffic flow* F_r specifies the average number of simultaneously realized connections:
$$F_r := \frac{\sum_{j \in I_A} t_j^{\texttt{stop}} - t_j^{\texttt{start}}}{T}$$
- The *offered traffic flow* F_o is defined as the traffic flow which would result if all calls were accepted:
$$F_o := \frac{\sum_{j \in I_A \cap I_R} t_j^{\texttt{stop}} - t_j^{\texttt{start}}}{T}$$
- The *blocking probability* $P_{\texttt{B}}$ is the fraction of rejected calls:
$$P_{\texttt{B}} := \frac{|I_R|}{|I_R| + |I_A|}$$
- The *mean hop distance* $D_{\texttt{mh}}$ of all chosen lightpaths is defined by
$$D_{\texttt{mh}} := \frac{\sum_{j \in I_A} \texttt{length}_j}{|I_A|}.$$
- The *mean shortest path hop distance* $D_{\texttt{msph}}$ is defined as the average distance between source and target node of all accepted calls:
$$D_{\texttt{msph}} := \frac{\sum_{j \in I_A} d(u_j, v_j)}{|I_A|}$$
- The *traffic load* L seeks to measure the capacity utilization of the network as follows:
$$L := \frac{F_r \cdot D_{\texttt{mh}}}{\sum_{a \in A} |W(a)|}$$

Among the aforementioned output quantities, the blocking probability is the main figure of interest. The value we are in fact interested in is the *expected* fraction of accepted calls, that is, the expected value of a random variable. Since it is not possible to compute the expected value of this output random variable (based on the distributions of the input random variables 'inter arrival time' and 'duration'), we can only carry out simulation runs in order to approximate the expected value of the (unknown) output random variable. The mean value of the measurements serves as estimate for the desired expected value, provided that enough measurements are used to compute it.

A natural question in this regard is how accurately the measurements approximate the desired expected value. An answer to this question is provided by the so-called *Confidence interval*. We refer to the book by Law and Kelton for detailed definitions and explanations [LK00]. Before giving a short idea of how the confidence interval should be interpreted, let us make three remarks about the computed confidence intervals. First, we computed a t confidence interval as defined by (4.12) in [LK00]. Second, the measurements were obtained by partitioning the sequence into batches of 1000 calls. And third, we computed an approximate 95% confidence interval.

The latter should be interpreted as follows: the probability that the expected value we are looking for is indeed contained in the confidence interval is at least 95 percent, provided that a sufficiently large number of experiments was carried out upon which the computation is based. Note that the number of experiments we carried out equals the number of batches.

Be aware that the mean value of all measurements, which equals the blocking probability achieved on the whole sequence, is the center of the corresponding confidence interval. Therefore, the tables in the appendix only indicate the half-length of the intervals, not their borders.

5. Simulation Studies

5.1. **Input.**

Topologies. As underlying digraph, we only used the 17-nodes network, but considered the following different dimensionings. The focus is set on the dimensioning provided by our project partner. It was obtained by a shortest path routing of the static demands as indicated by our partner's traffic matrix (see Appendix for the network and the demand matrix). The results on that topology, referred to as the *shortest path topology*, are reported on in Subsection 5.2.1. Since this topology favors the Greedy algorithms which use a shortest path routing, we also consider two other topologies. The first of these, referred to as the *ZIB topology*, was computed using the software developed at ZIB for the related project "Optimization of the static configuration of optical transport networks", and, having used the given traffic demand matrix as input, it also provides enough capacities to simultaneously route all static demands. Finally, some experiments were carried out on the *uniform topology* where all arcs are equipped with the same set of wavelengths. Figures of all three topologies can be found in the Appendix. Note that all topologies are symmetric, that is, the same set of wavelengths is available on arc (i, j) as on (j, i). This is obvious for th uniform topology, and for the other topologies it follows from the fact that all demands are symmetric, too.

We now turn to the algorithms used. Notice that all algorithms compute path lengths as number of arcs. Actual physical lengths are not considered.

Greedy-type Algorithms. In a uniform topology, the wavelength used the least and the one with the highest availability are identical, but in non-uniform topologies they may differ. This fact is taken into account in our implementation of the two Greedy-type algorithms *Pack* and *Spread*, which were originally defined to search the wavelengths in order of decreasing and increasing usage, respectively. In addition to these original versions (*Pack1* and *Spread1*), a second version for each of the two algorithms was implemented: *Pack2* considers the least available wavelength first, *Spread2* analogously starts to look for the shortest available path in the wavelength which still has the highest availability. In case that the order is not unambiguously defined by the above rules, we used as a tie-breaker the wavelength's indices: if two wavelengths utilization/availability coincide, the one with the smaller index is preferred.

For *Fixed*, we considered two different versions: *Fixed1* searches the wavelengths in increasing order of indices, *Fixed2* in decreasing order. Note that this indeed makes a difference in a non-uniform topologies, in which the wavelengths were assigned in

increasing order to an arc until the desired number of wavelengths were reached on that arc. Thus, *Fixed2* tries to assign the rarest wavelength first.

Moreover, the implementation of two different tie-breaking rules for *Exhaustive* also lead to two different versions: in *Exhaustive1*, the wavelength with the smallest index was taken among the set of wavelength which host an overall-shortest lightpath. In contrast to this, *Exhaustive2* realizes a shortest lightpath whose wavelength index is highest.

Finally, recall that *Random* searches the wavelengths in a randomly chosen order.

Versions of CBL *and* SCBL. Recall that CBL computes the cost of a routing choice as the sum of weights of newly blocked lightpaths, where the weight of a (newly blocked) lightpath is composed of three parts: a weight function w_1 which reflects the decrease in connectivity of its end nodes, a second function w_2 taking into account the lightpath's length, and finally the weight function w_3 introduced in Section 2.3 which considers the traffic demand of its end nodes. The choices for each of the weight functions as considered for our simulations are listed below, resulting in six different version of CBL.

For w_1 we have

(C) $w_1(P) := 1$,
(R) $w_1(P) := [P_{\text{all}}(s,t) - P_{\text{used}}(s,t)]^{-1}$,
(E) $w_1(P) := \mu^{[P_{\text{all}}(s,t)-P_{\text{used}}(s,t)]^{-1}} - 1$,

The parameter μ in (E) was set to be 100.

As length function w_2, we only took into account the definition which considers the relative length of a routing choice exponentially, since it proved to be superior to the other two definitions in all previous experiments:

(E) $w_2(P) := \nu^{\frac{n-|A(P)|}{n-1}}$.

Here, we chose $\nu = 100$.

Finally, the estimated average traffic demand between nodes can be taken into account by w_3 (see Section 2.3):

(C) $w_3(P) := 1$,
(T) $w_3(P) := \texttt{demand}(s,t)$.

As in the previous experiments described in the report on Milestone 3, SCBL incorporates the original version of CBL, in which w_1 is exponential (E), and all other weights are constant. Remember that SCBL allows to restrict the number of routing choices and/or the number of newly blocked lightpaths whose weight contributes to the cost of a routing choice. For our current experiments, we only restricted the number of newly blocked lightpaths: SCBL3 and SCBL5 only consider the weights of the three and five shortest among those, respectively, when computing the cost of a routing choice.

Versions of DLD *and* ADLD. Here, we tested several versions which result from the decision whether traffic demands should be taken into account as described in Section 2.3 (T as first index) or not (first index C) and their combination with two different tie breaking rules. If multiple paths incur the same cost, the versions with second index f select the first one found, whereas the second suffix s indicates that a shortest among

those is taken. Hence, $\mathsf{ADLD}_{\mathtt{Cs}}$ denotes the version of ADLD which ignores the traffic demands and selects a shortest one if several routing choices have minimum cost.

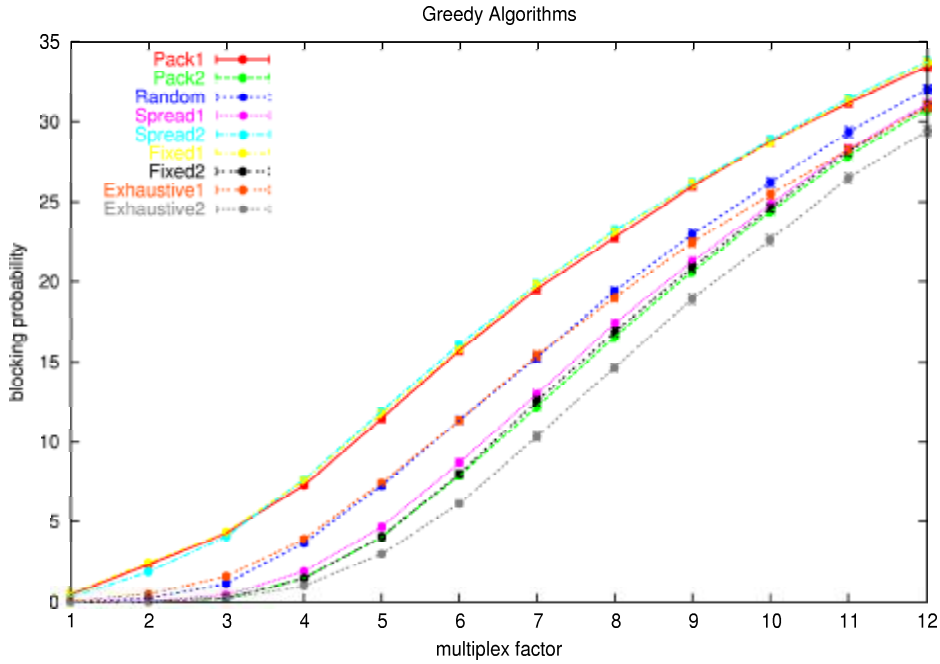
Request sequences. The parameters specifying the generation of the request sequences were chosen in accordance with our project partner as follows. All durations were fixed to 1. In order to model the arrival of one call per 12 units of time at each source on average, the mean of the corresponding exponential distribution was set to 11, and then each of the inter arrival times generated accordingly was increased by 1. Thereby it was ensured that no two calls of the same source can overlap.

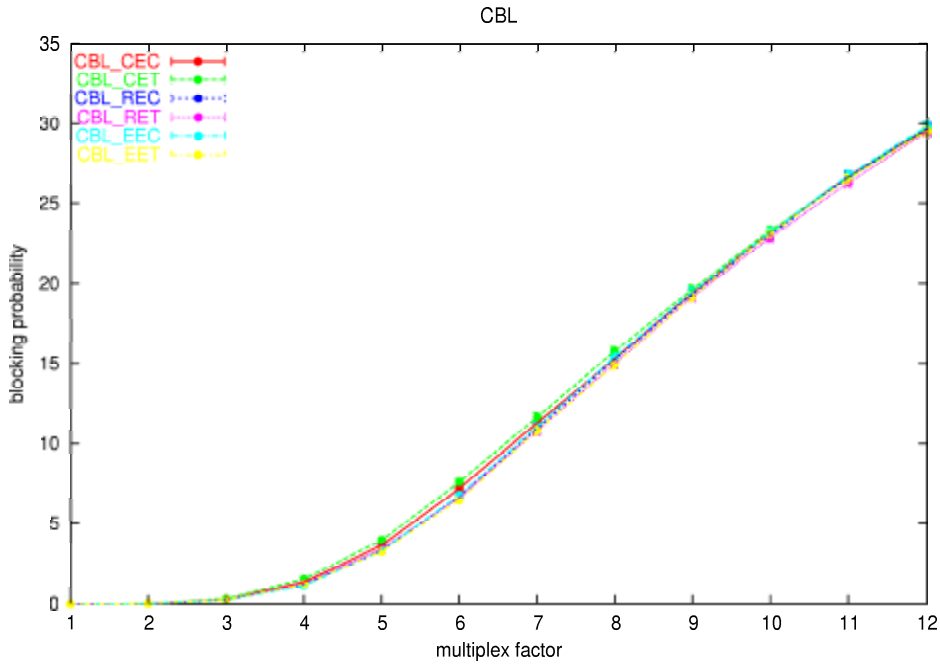
Furthermore, one request sequence was generated for each multiplex factor m between 1 and 12. For the shortest path topology, we generated two different request sequences, one with 101000 requests, the other one containing 11000 calls. The first 1000 calls in each sequence serve as onset. After testing all algorithms on the long sequence in the shortest path topology, we selected those algorithms which performed best and ran them on the shorter sequence on all three topologies. In addition, we considered one more algorithm which we hoped to perform good on those topologies which are not based on shortest path routings.

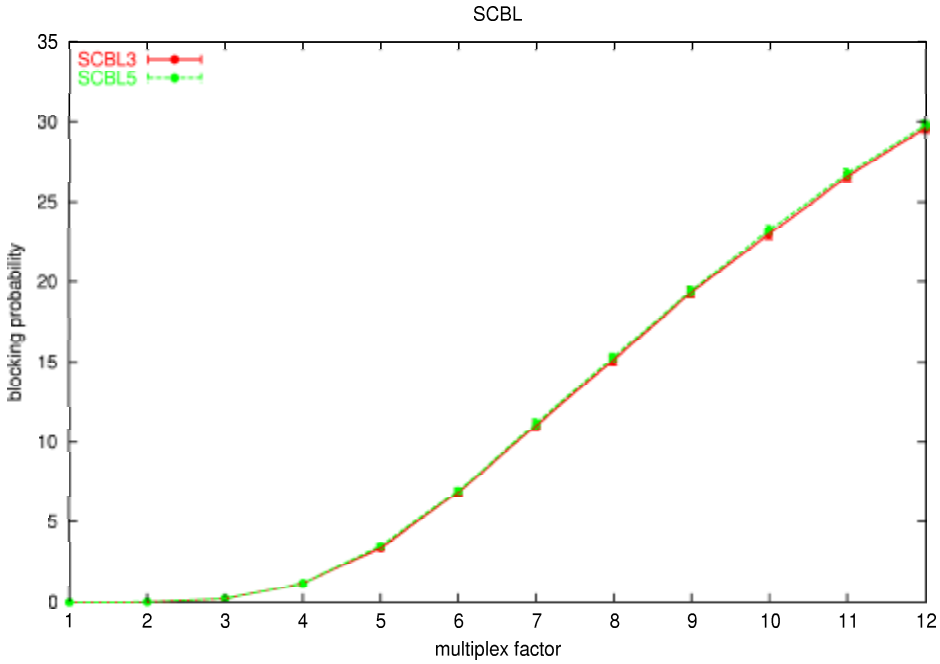
Finally, the demand of each call was restricted to be 1.

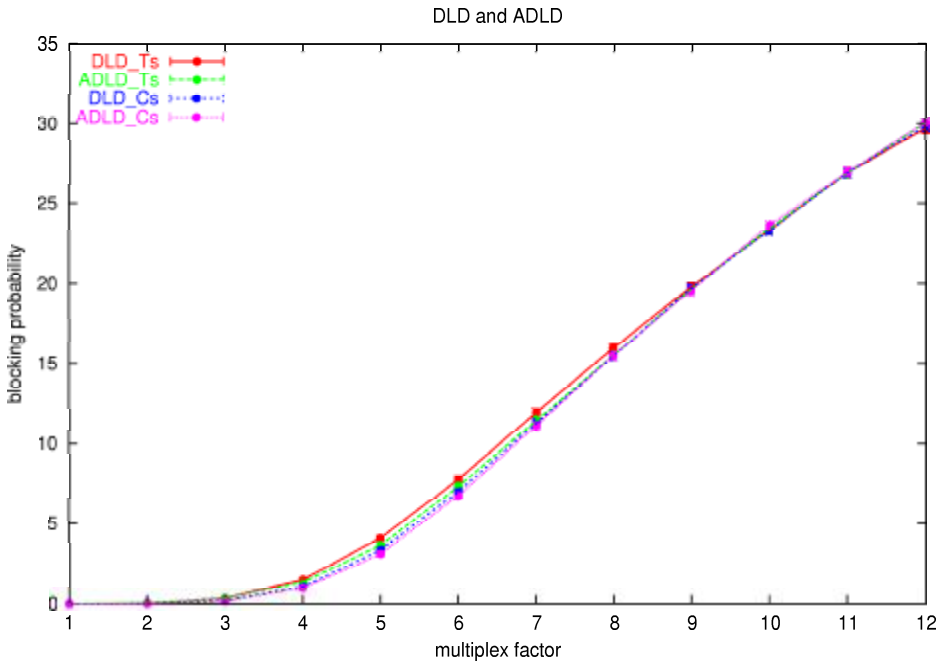
5.2. **Results.** The key figure of interest is the blocking probability $P_{\mathtt{B}}$. The diagrams on the following section show the blocking probabilities in relation to the 12 multiplex factors as achieved by the different algorithms. In addition to the blocking probabilities, the diagrams display the borders of the corresponding confidence intervals. For their computation, the sequences were partitioned into batches of 1000 calls. Since the long sequence (used only for a simulation run on the shortest path topology) contains a large number of batches, the confidence intervals here are very small and hardly visible. For the sequence with 11000 calls, the confidence intervals are much bigger and hence better visible. Tables with the corresponding values of the blocking probability and the half-lengths of the confidence intervals (for the best algorithms on the short sequence) can be found in the Appendix.

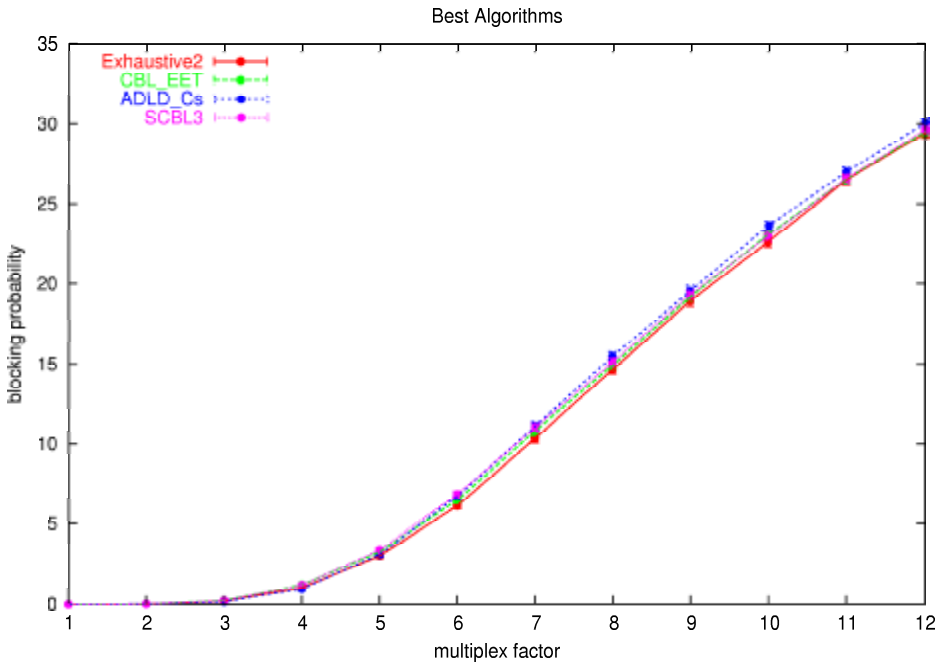
5.2.1. *Results on the shortest-path topology.* Following are six figures which show the blocking probabilities in relation to the 12 multiplex factors as achieved by the different algorithms on the sequence containing 101000 calls. The first figure contains the results of the Greedy-type algorithms, the second and third those of the considered CBL and SCBL versions, respectively, and the fourth picture displays the blocking probabilities achieved by DLD and ADLD. The fifth figure shows a combination of the best among all algorithms: *Exhaustive2*, $\mathsf{CBL}_{\mathtt{EET}}$, $\mathsf{ADLD}_{\mathtt{Cs}}$, and SCBL3. The last figure is a clipping of the fifth figure, showing the blocking probabilities of the best algorithms for multiplex factor 1 to 5 in more detail.

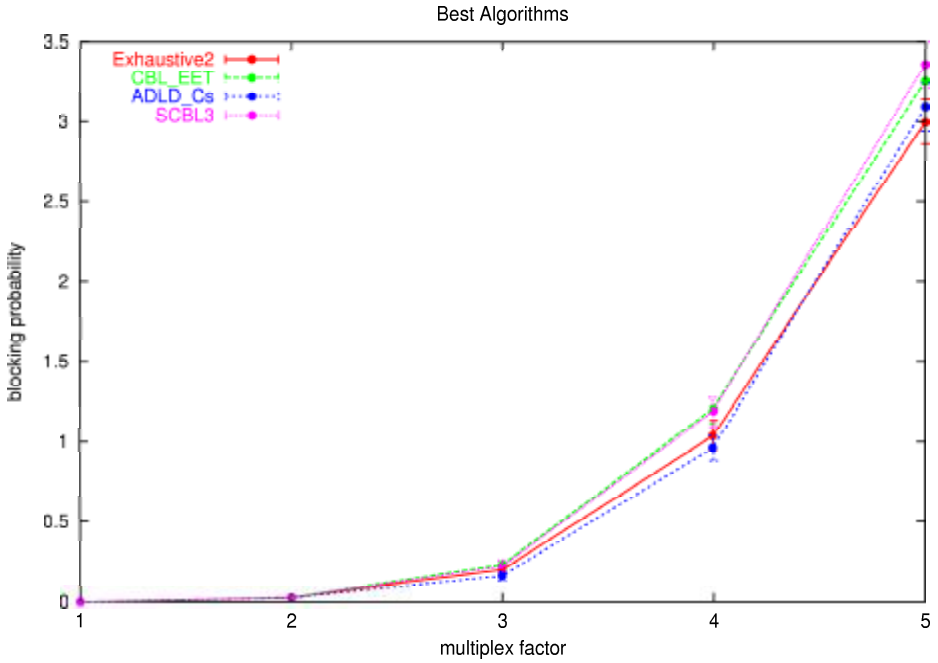

Greedy Algorithms
Pack1
Pack2
Random
Spread1
Spread2
Fixed1
Fixed2
Exhaustive1
Exhaustive2
blocking probability
multiplex factor
0
5
10
15
20
25
30
35
1
2
3
4
5
6
7
8
9
10
11
12


CBL
CBL_CEC
CBL_CET
CBL_REC
CBL_RET
CBL_EEC
CBL_EET
blocking probability
multiplex factor
0
5
10
15
20
25
30
35
1
2
3
4
5
6
7
8
9
10
11
12


SCBL
SCBL3
SCBL5
blocking probability
multiplex factor
0
5
10
15
20
25
30
35
1
2
3
4
5
6
7
8
9
10
11
12


DLD and ADLD
DLD_Ts
ADLD_Ts
DLD_Cs
ADLD_Cs
blocking probability
multiplex factor
0
5
10
15
20
25
30
35
1
2
3
4
5
6
7
8
9
10
11
12


Best Algorithms
Exhaustive2
CBL_EET
ADLD_Cs
SCBL3
blocking probability
multiplex factor
0
5
10
15
20
25
30
35
1
2
3
4
5
6
7
8
9
10
11
12

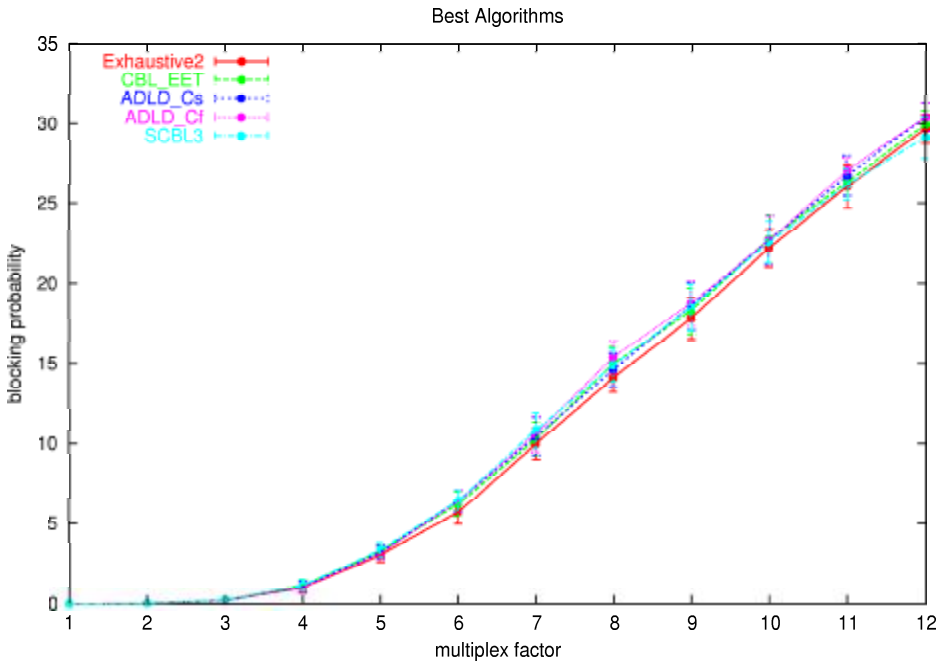

Best Algorithms
Exhaustive2
CBL_EET
ADLD_Cs
SCBL3
blocking probability
multiplex factor
0
0.5
1
1.5
2
2.5
3
3.5
1
2
3
4
5

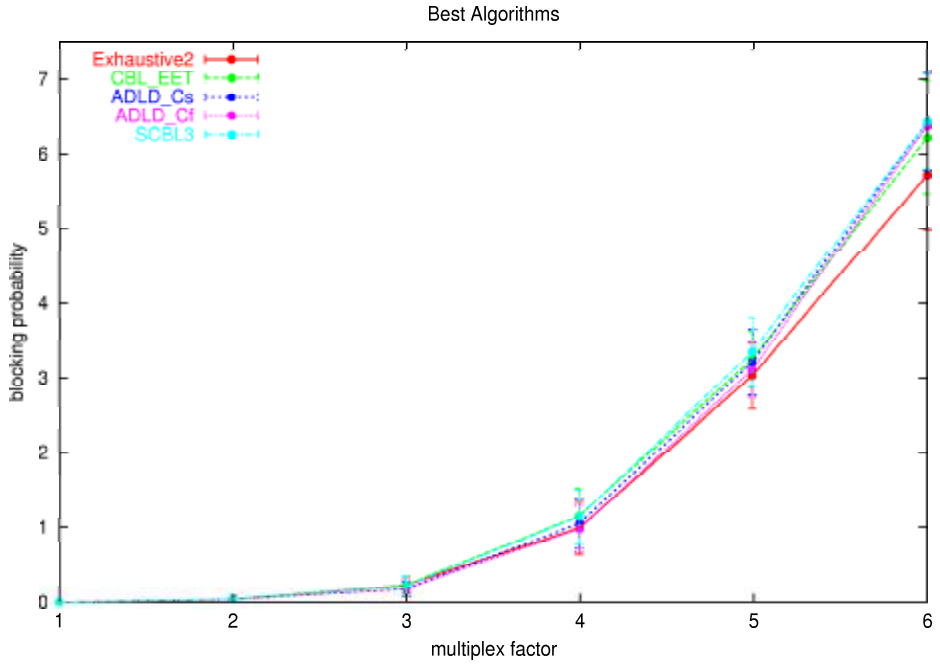
Clear differences can be observed among the algorithms of Greedy type: *Exhaustive2* is superior to all the others, followed by *Pack2* and *Fixed2*. Another interesting observation is that a great influence is exerted by the tie breaking rules and the order in which the wavelengths are searched: *Fixed2*, *Exhaustive2*, and *Pack2* perform far better than their respective counterparts *Fixed1*, *Exhaustive1*, and *Pack1*. This must be due to the way the wavelengths are considered in the tie breaking rules and the search orders, respectively: it is advantageous for *Exhaustive* and *Fixed* to prefer the wavelengths with higher index, that is, those wavelengths which are less available (in the empty network). The same holds for *Pack*; and *Spread1*, considering the currently least available wavelength first, outperforms its counterpart, too.

The different versions of CBL as well as those of SCBL do all achieve very similar results. Also for DLD and ADLD, little deviation can be observed. For multiplex factors $m \leq 9$, $\mathsf{ADLD}_{\mathtt{Cs}}$ is marginally better than the other algorithms, but $\mathsf{ADLD}_{\mathtt{Cs}}$ and $\mathsf{ADLD}_{\mathtt{Cf}}$ being very close.

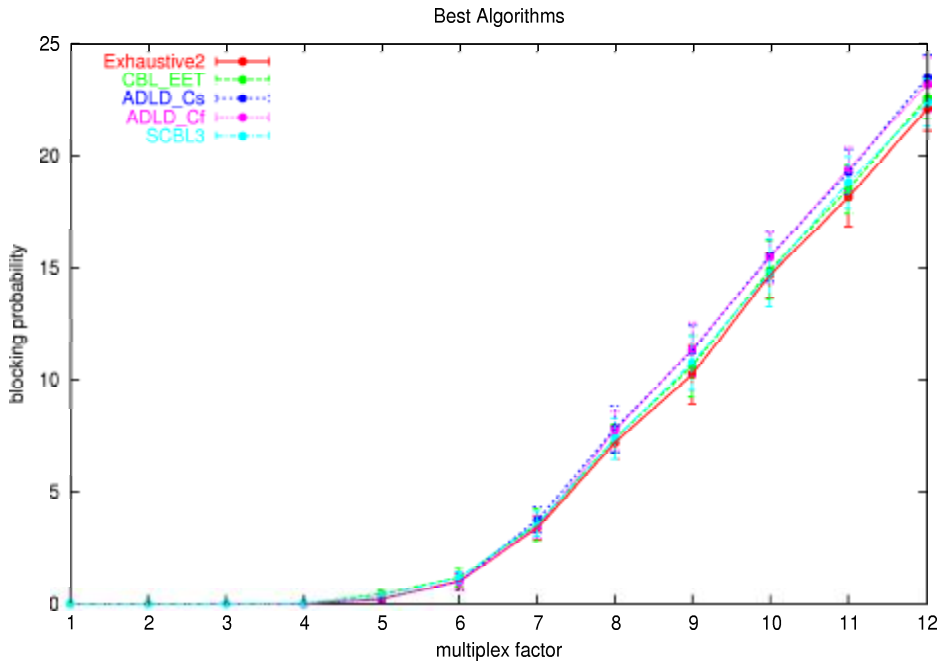
When comparing the best algorithms, it shows that all of them are at close quarters, $\mathsf{ADLD}_{\mathtt{Cs}}$ and *Exhaustive2* slightly outperforming the other two. The clipping reveals that $\mathsf{ADLD}_{\mathtt{Cs}}$ is better than *Exhaustive2* up to multiplex factor 4, at which the blocking probability still lies below 1 percent.

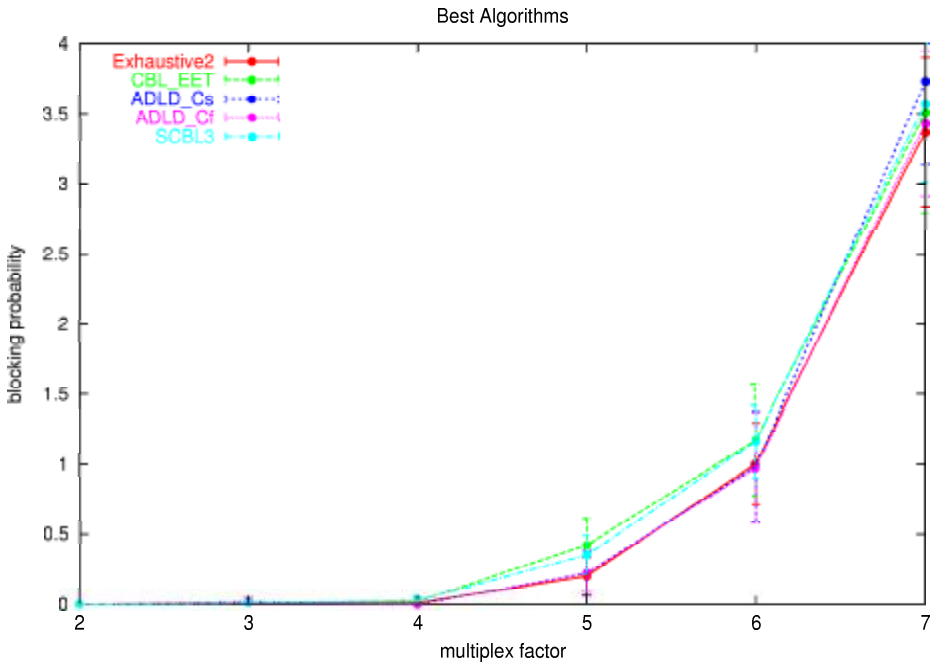
The following two figures shows the results of the selection of best algorithms on the shorter sequence. Here, we additionally considered $\mathsf{ADLD}_{\mathtt{Cf}}$.





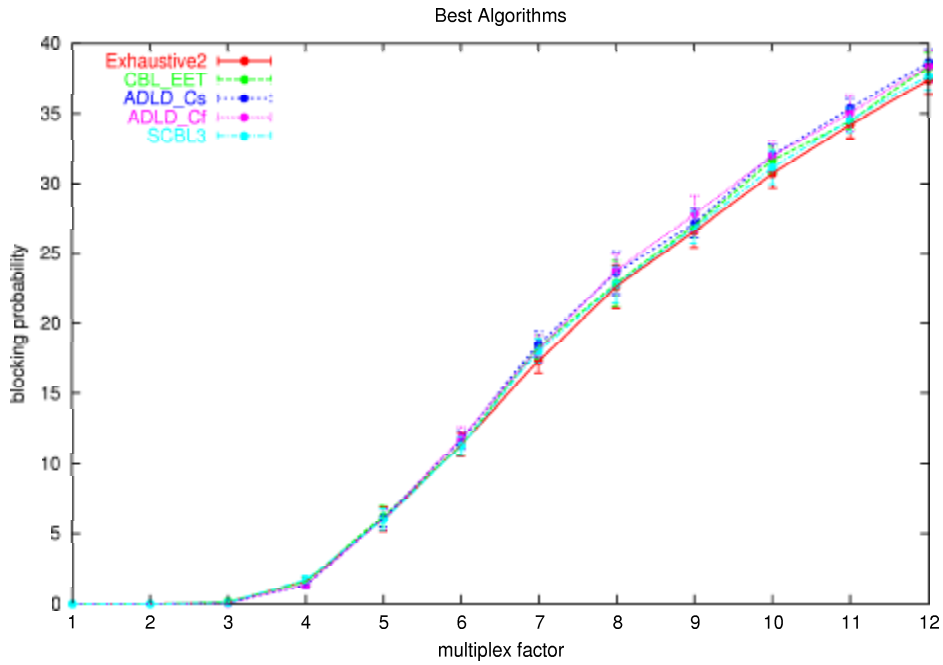
5.2.2. *Results on the ZIB topology.* Here, due to lack of time, we only tested the algorithms which performed best on the shortest path topology, plus $\texttt{ADLD}_{\texttt{Cf}}$.

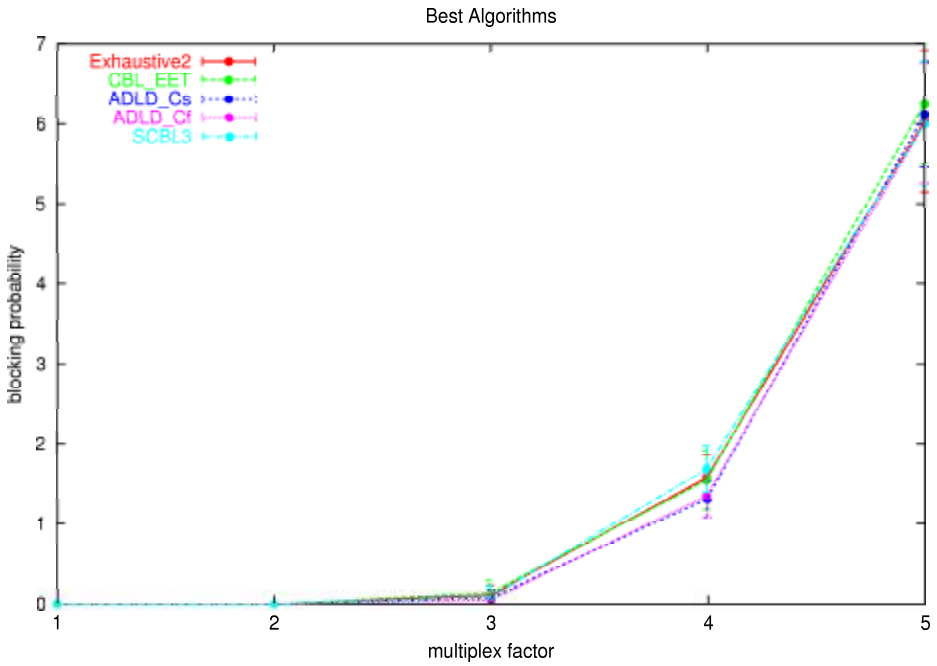




Again, all algorithms achieve very similar results. The clipping shows that $\mathsf{ADLD}_{\mathtt{Cf}}$ and *Exhaustive2* perform almost equally well up to multiplex factor 4, *Exhaustive2* being the clear winner on those scenarios with more strain on the network.

5.2.3. *Results on the uniform topology.* Again, we only compared the algorithms *Exhaustive2*, $\mathsf{CBL}_{\mathtt{EET}}$, $\mathsf{ADLD}_{\mathtt{Cs}}$, SCBL3, and $\mathsf{ADLD}_{\mathtt{Cf}}$.





Also on the uniform topology (which certainly does not favor the Greedy-type algorithms which route on shortest paths), ADLD and *Exhaustive2* are competing for the best results. As before, *Exhaustive2* outperforms the other algorithms for scenarios with multiplex factor $m \geq 5$, whereas $\mathsf{ADLD}_{\mathtt{Cs}}$ and $\mathsf{ADLD}_{\mathtt{Cf}}$ achieve the best values for the blocking probability for the smaller multiplex factors.

Let us point out that the different dimensionings also lead to different values for the blocking probability: whereas for multiplex factor $m = 5$, the best algorithms achieved a blocking probability of around 3 percent on the shortest path and the uniform topology, they seem to perform much better on the ZIB topology with values around 0.5 percent. This suggests that the dimensioning used in the ZIB topology is more generous than the other two. In fact, this is confirmed when comparing the number of wavelength hops of the different topologies (that is, the number of wavelengths per link summed over all links): for the shortest path topology, it is 328, for the ZIB topology 380, and 312 for the uniform dimensioning.

5.3. **A note on rejection criteria.** In order to investigate whether the blocking probability can be reduced by rejecting calls which could be accepted, we implemented two simple rejection criteria for the algorithm $\mathsf{DLD}_{\mathtt{Cf}}$. The first criterion relates the profit p_j of a call σ_j to the current minimum cost of a routing choice for the call, denoted by $\mathtt{min}_{\mathtt{j}}$. If p_j is too small in comparison with $\mathtt{min}_{\mathtt{j}}$, the call is rejected. Since profits of calls are identic in our simulation model, this amounts to rejecting σ_j if $\mathtt{min}_{\mathtt{j}}$ exceeds a given threshold value δ. Obviously, for decreasing δ, the number of calls rejected 'voluntarily' by the algorithm tends to increase. We carried out experiments on this version of $\mathsf{DLD}_{\mathtt{Cf}}$ using for δ the values $110, 120, 130, 140$, and 150.

The idea exploited by the second rejection criterion is to restrict the set of routing lightpaths. Recall that $d(u, v)$ denotes the minimum number of arcs in a shortest (u, v)-path for a pair of nodes $u, v \in V$. If a call σ_j with start node u_j and end node v_j can not be routed on a lightpath whose number of arcs is bounded from above by $d(u_j, v_j) + r$, where r is some constant, σ_j is rejected. We considered $2, 3, 4, 5, 6, 7, 8$, and 9 as values for r. This rejection criterion becomes more effective with decreasing value of r.

Of course, for both algorithms their rejection parameters δ and r should be chosen depending on the network topology and the average traffic load. Computational results show that, for small average load, both algorithms are not superior to the version of $\mathsf{DLD}_{\mathtt{Cf}}$ which does not reject without necessity. We conjecture that the described rejection criteria will only bring some if the network load is very high.

5.4. **Running times.** The following table shows the running times of various algorithms on two test instances in seconds. In both instances, the shortest path topology was considered and the time needed to process 1000 calls was measured, these calls not belonging to the onset of the sequence. The two sequences differ in the value chosen for the multiplex factor: sequence 1 was generated setting $m = 1$, for sequence 2, we set $m := 10$. It can be observed that the Greedy algorithms are much faster

Algorithm	Multiplex factor 1	Multiplex factor 10
Pack2	1-2	1-2
Exhaustive2	1-2	1-2
CBL	106	3
SCBL3	585	126
DLD	284	15
ADLD	623	61

TABLE 3. Running times of selected algorithms on 1000 calls in the shortest-path topology in seconds.

than our algorithms. This is due to the fact that our algorithms need to compute the cost function values. Whereas we could implement the branch-and-bound procedure described in the previous report to speed up CBL this cannot be used anymore for SCBL3, which clearly shows in the running time. As for DLD and ADLD which both have to solve instances of the MAXIMUM FLOW PROBLEM the anticipating version is much slower. This is due to the fractional capacity values in the MAXIMUM FLOW PROBLEM defined by ADLD : the $0/1$-values used by DLD allow for a faster solving algorithm, as also theoretically proven (see Section 2.1).

6. CONCLUSIONS

Goal. The goal of the project was the review, design, implementation, and evaluation of algorithms for the dynamic configuration of circuit-switched, transparent optical networks. To this end, we have formulated Online Optimization problems that comprehensively model the specifications of the dynamic configuration of optical networks. Moreover, we have developed a new network simulator (based on existing software for

graph manipulation) which allows to compare the performance of different algorithms. Concerning the algorithms, the following steps have been undertaken.

Review: We have reviewed the methods proposed in the literature so far. Most of these are greedy methods which do not reject calls that could be routed and which use shortest paths for the routings of calls.

Design: We have developed several algorithms which evaluate the fitness of the network at a certain point in time by various optimization methods. The common underlying principle is to choose that routing decision that yields the smallest decrease in network fitness among all possible decisions. These "fitness-algorithms" involve a higher degree of mathematical sophistication.

Implementation: We have implemented both the known and our new algorithms in the framework of our new network simulator. To this end, we used state-of-the-art algorithmic knowledge and developed some new ideas. All algorithms are fast enough to be used in the 17-node network provided by our cooperation partner.

Evaluation: We have validated our simulation system by comparison to an existing system of our cooperation partner. The key figure of interest observable in the experiments is the blocking probability for various increases in network load as characterized by the multiplex factor.

6.1. **Results.**

- The simulation results suggest that the observed blocking probability depends on the choice of algorithms. In particular, different tie breaking rules for *Exhaustive* as well as different variants of the wavelength orders used by *Pack* and *Spread* lead to diverging blocking probabilities. Therefore, these results are important for the planner.
- The performance of an algorithm also depends on how the static network configuration was obtained.
- All proposed algorithms can be used in real-time in the 17-node network of our cooperation partner, though greedy algorithms are fastest.
- Worst-case performance guarantees in terms of competitive analysis can be given only for a few algorithms. Unfortunately, these can not be employed in practice.
- All proposed algorithms can be implemented in a distributed fashion without serious degradation in performance.
- The fitness algorithms like CBL or, even more so, DLD help to avoid bad behavior in unlucky situations (see Example provided by Figures 3 and 4 in the report on Milestone 1). If a very regular distribution is chosen for the inter arrival times (Poisson type), then their performance is only slightly better compared to greedy type algorithms. Nevertheless, choosing one of the fitness-algorithms means
  - (+) the possibility to take into account further specifications of a call, such as higher demands, different service classes, non-constant profits, etc.,
  - (+) a natural way to estimate the value of blocking a call that could be routed,
  - (-) a higher implementation effort.
- Sharing resources by dynamic configuration leads (on the shortest path topology) to an acceptable blocking probability of 3.5% whenever the multiplex

factor does not exceed 5. This could be predictably achieved by algorithms *Exhaustive2*, $\mathsf{ADLD}_{\mathtt{Cs}}$, SCBL3, and $\mathsf{CBL}_{\mathtt{EET}}$.

Recommendation. Dynamic configuration yields a gain in resource utilization whenever calls vary a lot over time. For the configuration, there is an abundant number of algorithms to choose from. Out of the greedy algorithms, *Exhaustive2* performs best, for small blocking probability only beaten by the best of the fitness-algorithms $\mathsf{ADLD}_{\mathtt{Cs}}$. The results suggest that is is favorable to prefer those wavelengths with the least availability. The strategy of the dynamic algorithm should resemble the way the static network configuration was carried out.

Some issues remain unrevealed as of now, for instance, recovery strategies after node/arc failures. Here, offline algorithms for routing and wavelength assignment come into play, and the network fitness again provides a valuable decision guidance. Another important topic is the way the call sequences are generated: in our opinion, one should consider peaked demand distributions, or, even more important, simulation runs should be carried out on real data. Other field of interest are profit functions and rejection criteria as well as randomly generated call durations and demands. Therefore, a follow-up project could provide valuable additional information.

REFERENCES

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Networks flows*, Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[GJ79] M. R. Garey and D. S. Johnson, *Computers and intractability (a guide to the theory of* NP*-completeness)*, W.H. Freeman and Company, New York, 1979.

[LK00] Averill M. Law and W. David Kelton, *Simulation modeling and analysis*, McGraw-Hill, Boston, 2000.

APPENDIX

The following table contains the values of the best algorithms' blocking probabilities on the 10000 calls processed on the shortest path topology (first entry). Moreover, the second value in each entry is the half length of the confidence interval.

Mux factor	*Exh2*	CBL$_{\texttt{EET}}$	SCBL3	ADLD$_{\texttt{Cs}}$
1	0.0/0.0	0.0/0.0	0.0/0.0	0.0/0.0
2	0.026/0.01	0.026/0.01	0.025/0.005	0.025/0.005
3	0.199/0.03	0.231/0.03	0.221/0.035	0.161/0.03
4	1.037/0.09	1.205/0.095	1.187/0.095	0.963/0.08
5	2.996/0.14	3.253/0.15	3.357/0.145	3.091/0.15
6	6.143/0.195	6.485/0.185	6.859/0.195	6.736/0.195
7	10.362/0.285	10.809/0.28	11.016/0.265	11.102/0.28
8	14.655/0.275	14.925/0.265	15.115/0.295	15.481/0.29
9	18.933/0.33	19.185/0.315	19.327/0.305	19.552/0.33
10	22.639/0.31	23.1/0.28	22.999/0.3	23.641/0.27
11	26.491/0.32	26.552/0.295	26.574/0.295	26.995/0.295
12	29.452/0.315	29.537/0.31	29.635/0.295	30.058/0.29

TABLE 4. Blocking probability / Half-length of the confidence interval for the best algorithms on the shortest path topology.

The following three tables contain the same two values for each of the algorithms *Exhaustive2*, CBL$_{\texttt{EET}}$, ADLD$_{\texttt{Cs}}$, SCBL3, and ADLD$_{\texttt{Cf}}$ as obtained on the 10000 calls. The first table refers to the shortest path topology, the second one to the ZIB topology, and the last one to the uniform topology.

Mux factor	*Exh2*	CBL$_{\texttt{EET}}$	SCBL3	ADLD$_{\texttt{Cf}}$	ADLD$_{\texttt{Cs}}$
1	0.0/0.0	0.0/0.0	0.0/0.0	0.0/0.0	0.0/0.0
2	0.04/0.05	0.04/0.05	0.04/0.05	0.04/0.05	0.04/0.05
3	0.23/0.11	0.23/0.12	0.22/0.13	0.18/0.11	0.18/0.1
4	0.99/0.35	1.15/0.37	1.14/0.36	0.98/0.31	1.05/0.33
5	3.03/0.44	3.25/0.37	3.35/0.46	3.11/0.37	3.21/0.44
6	5.71/0.73	6.22/0.76	6.43/0.64	6.38/0.7	6.43/0.66
7	9.97/0.98	10.27/1.04	10.83/1.08	10.6/1.1	10.43/1.19
8	14.12/0.88	14.99/1.05	14.9/1.03	15.4/1.02	14.62/1.05
9	17.82/1.31	18.28/1.45	18.51/1.46	18.75/1.42	18.59/1.49
10	22.22/1.16	22.77/1.4	22.58/1.3	22.71/1.51	22.71/1.51
11	26.07/1.36	26.36/1.88	26.19/1.96	27.08/0.82	26.76/1.27
12	29.67/0.9	29.93/0.88	29.13/1.3	30.38/0.86	30.38/0.86

TABLE 5. Blocking probability / Half-length of the confidence interval for the best algorithms on the shortest path topology and 10000 calls.

Mux factor	*Exh2*	CBL$_{\texttt{EET}}$	SCBL3	ADLD$_{\texttt{Cf}}$	ADLD$_{\texttt{Cs}}$
1	0.0/0.0	0.0/0.0	0.0/0.0	0.0/0.0	0.0/0.0
2	0.0/0.0	0.0/0.0	0.0/0.0	0.0/0.0	0.0/0.0
3	0.01/0.02	0.01/0.02	0.01/0.02	0.01/0.03	0.01/0.02
4	0.01/0.02	0.02/0.03	0.03/0.03	0.0/0.0	0.0/0.0
5	0.2/0.13	0.42/0.19	0.35/0.14	0.22/0.13	0.22/0.16
6	1.0/0.29	1.17/0.4	1.16/0.26	0.97/0.29	0.98/0.29
7	3.37/0.53	3.51/0.72	3.57/0.56	3.43/0.52	3.73/0.59
8	7.19/0.74	7.37/0.64	7.36/0.91	7.72/0.85	7.8/1.04
9	10.23/1.3	10.6/1.36	10.75/1.2	11.35/1.17	11.2/1.11
10	14.67/1.0	14.89/1.33	14.76/1.51	15.45/1.15	15.52/1.12
11	18.13/1.29	18.5/1.11	18.8/1.17	19.4/0.97	19.26/1.0
12	22.09/0.97	22.58/0.9	22.36/1.05	23.17/1.21	23.47/1.02

TABLE 6. Blocking probability / Half-length of the confidence interval for the best algorithms on the ZIB topology and 10000 calls.

Mux factor	*Exh2*	CBL$_{\texttt{EET}}$	SCBL3	ADLD$_{\texttt{Cf}}$	ADLD$_{\texttt{Cs}}$
1	0.0/0.0	0.0/0.0	0.0/0.0	0.0/0.0	0.0/0.0
2	0.0/0.0	0.0/0.0	0.0/0.0	0.0/0.0	0.0/0.0
3	0.11/0.13	0.14/0.16	0.09/0.14	0.05/0.11	0.08/0.11
4	1.58/0.29	1.55/0.37	1.69/0.29	1.35/0.27	1.31/0.23
5	6.03/0.89	6.25/0.75	6.0/0.78	6.01/0.75	6.12/0.66
6	11.34/0.82	11.28/0.6	11.26/0.74	11.82/0.76	11.68/0.66
7	17.32/0.89	18.31/0.84	18.05/0.86	18.17/0.98	18.55/0.9
8	22.62/1.54	22.85/1.62	22.76/1.29	23.77/1.2	23.6/1.53
9	26.58/1.17	27.01/0.91	26.84/1.22	27.82/1.28	27.18/1.04
10	30.7/1.03	31.64/0.96	31.12/1.2	31.95/1.09	32.0/0.81
11	34.2/0.97	34.46/0.6	34.53/0.87	35.03/1.28	35.36/0.7
12	37.35/1.0	38.29/1.02	37.66/1.02	38.37/1.12	38.61/0.92

TABLE 7. Blocking probability / Half-length of the confidence interval for the best algorithms on the uniform topology and 10000 calls.

The following three figures show the three topologies used. Note that they are symmetric, that is, if the five first wavelengths are available on arc $(5, 16)$, then the same wavelengths are available on arc $(16, 5)$ (the arrows being hardly visible to the naked eye). Therefore, we indicated the number of eligible wavelengths only once per direction. Finally, the last table is the demand matrix. Recall that all demands are symmetric, therefore it suffices to indicate the demands for arcs (i, j) with $i < j$.

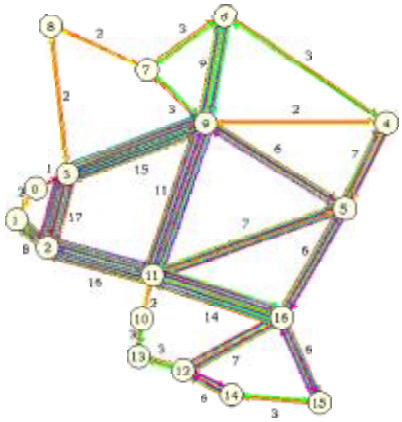


FIGURE 5. The shortest path topology.

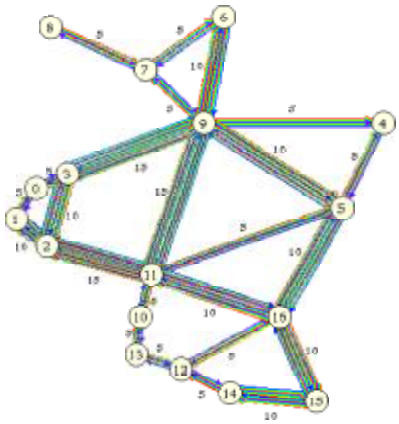


FIGURE 6. The ZIB topology.

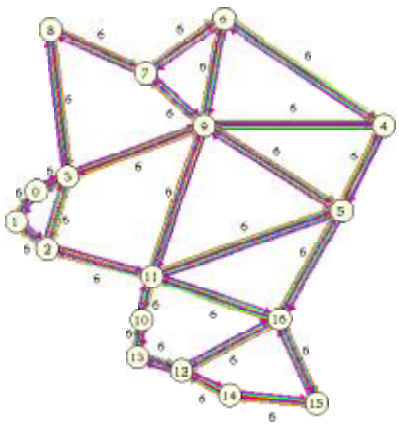


FIGURE 7. The uniform topology.

	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10	N11	N12	N13	N14	N15	N16	N17
		1	1	1													
			2	1			1			1		1	1				
				4		3	1			4		5	1				
						3	1			4		3					
						2	1			2		1				1	1
							2			3		3	1		1	1	1
								1	2	3		1	1			1	
										1							
												2					
												5	1				2
												1	1	1			
													2	1	2	2	1
														1	1	1	1
																1	
																1	
																	1