Technical Report TR-RI-12-328: Introducing feedback to preemptive routing and wavelength assignment algorithms for dynamic traffic scenarios

Philip Wette, Holger Karl
University of Paderborn
33098 Paderborn, Germany
Email: {philip.wette, holger.karl}@uni-paderborn.de

***Abstract*—Preemptive Routing and Wavelength Assignment (RWA) algorithms preempt established lightpaths in case not enough resources are available to setup a new lightpath in a Wavelength Division Multiplexing (WDM) network. The selection of lightpaths to be preempted relies on internal decisions of the RWA algorithm. Thus, if dedicated properties of the network topology are required by the applications running on the network, these requirements have to be known by the RWA algorithm. Otherwise it might happen that by preempting a particular lightpath these requirements are violated. If, however, these requirements include parameters only known at the nodes running the application, the RWA algorithm cannot evaluate the requirements. For this reason a RWA algorithm is needed which involves its users in the preemption decisions.**

We present a family of preemptive RWA algorithms for WDM networks. These algorithms have two distinguishing features: a) they can handle dynamic traffic by on-the-fly reconfiguration, and b) users can give feedback for reconfiguration decisions and thus influence the preemption decision of the RWA algorithm, leading to networks which adapt directly to application needs. This is different from traffic engineering where the network is (slowly) adapted to observed traffic patterns.

Our algorithms handle various WDM network configurations including networks consisting of heterogeneous WDM hardware. To this end, we are using the layered graph approach together with a newly developed graph model that is used to determine conflicting lightpaths.

I. Introduction

Over the last years wavelength division multiplexing networks have been deployed at large scale. In most cases these networks are assembled from passive equipment which cannot adapt dynamically to traffic changes. With the development of very fast active WDM equipment [1] that can be remotely reconfigured [2], these networks can adapt to changing traffic demands.

In WDM networks a connection between two hosts is established by creating a so-called lightpath, enabling these hosts to communicate in a circuit-switched manner. For the static traffic case, a traffic matrix is given and a RWA algorithm has to find a set of lightpaths that meet the traffic demands and has no two lightpaths using the same wavelength on any shared edge. When traffic is dynamic, the lightpath assignment should adapt to the actual traffic situation. Numerous algorithms for the dynamic traffic case exist which either work non-preemptively [3], [4], [5] or preemptively [6], [7]. Preemptive RWA algorithms are advantageous because preemption gives the opportunity to exchange old, underutilized lightpaths with new, potentially higher utilized lightpaths leading to higher overall network utilization. The major drawback of existing preemptive RWA algorithms is that they have no or only a very limited notion of dependencies between lightpaths. This leads to problems especially when designing cooperative systems built on top of a dynamic network topology [8]. Cooperative systems, like peer to peer networks or multi-tier services, depend on a particular virtual network topology induced by the connections held by each participating node. Depending on the existing virtual topology, these nodes create new connections. Thus, a new connection is only useful if some other connections remain in the network. As these dependencies are known neither to the network operator nor the used RWA algorithm, the knowledge at the users of the network can be used to decide preemption better suited to the needs of the applications running on the network.

As an example, consider a distributed application requiring logical interconnections such that the shortest path between each pair of nodes running the application is 2 hops or less. Fig. 1 shows such a system where solid edges represent established lightpaths. To comply with the "2 hops or less" constraint, a new lightpath between F and C has to be set up. Now suppose that (due to capacity constraints) to set up this new lightpath either the lightpath F-E or A-D has to be preempted. If A-D is preempted the hop count from B to D rises to 3 violating the constraint. For the RWA algorithm to deal with custom constraints a utility function is required which rates the different preemption candidates and removes solutions violating the constraints. The utility function strongly depends on the required constraints of the application. As these constraints could also include parameters which can only be determined at the nodes running the application, like end-to-

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" (SFB 901).

end latency, the application and not the RWA algorithm should evaluate the utility function.

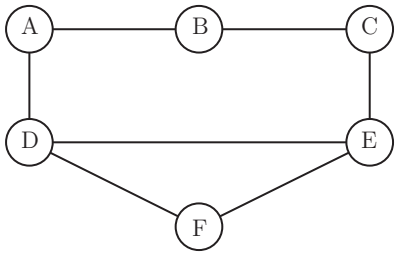


Fig. 1. Exemplary topology of a distributed application. Edges represent established lightpaths.

We present a preemptive routing and wavelength assignment algorithm for dynamic traffic which enables the applications running on the network to actively influence the preemption decisions by selecting the most feasible candidate from several choices calculated by our algorithm. This serves as a powerful building block when constructing networks which automatically adapt to the logical topology of the applications running on top of it. In addition we show how to use this building block to create a lightpath rerouting algorithm. This algorithm computes *multiple* sets of possible candidate lightpaths for rerouting in case a new lightpath demand can not directly be handled by the network. Neither the number of candidate sets nor the cardinality of each set are bounded in size which is an outstanding feature since most existing lightpath rerouting algorithms strongly restrict the number of possible rerouting candidates.

The rest of the paper is structured as follows: Section II discusses related work in both RWA algorithms and lightpath rerouting algorithms. Section III presents assumptions and definitions along with the graph model that is used throughout the paper. Section IV describes our preemptive RWA algorithm which is evaluated by simulation in Section V. Section VI concludes this paper.

II. Related Work

Existing routing and wavelength assignment algorithms can be differentiated by a) their assumption on the existence of wavelength converters in the network, b) their required knowledge about future lightpath requests, and c) their optimization criteria. In addition, they can be differentiated into algorithms which—given a new lightpath request—build new networks from scratch and algorithms trying to integrate the new lightpaths on-the-fly into the existing lightpath topology.

We are interested in preemptive RWA algorithms taking heterogeneous wavelength converters into account and handling new lightpath requests by on-the-fly network reconfiguration without knowledge about future lightpath requests. In addition to a low blocking probability, we are interested in a generic algorithm which is able to create lightpath topologies that comply with arbitrary custom constraints.

Several preemptive RWA algorithms with different strategies for choosing preemption candidates exist in the literature. In the following we will give a short overview over some of these algorithms.

In [9] a lightpath reconfiguration algorithm for IP over WDM networks is presented. The algorithm adapts the lightpath topology of a WDM network to the observed IP traffic. The WDM network is assumed to be used exclusively for IP traffic and that wavelength conversion is possible. Each lightpath is assigned two so called *watermarks*: W_L and W_H. Whenever IP traffic on a lightpath is less than W_L this lightpath is considered as underutilized and therefore the lightpath is preempted. Analogously, whenever IP traffic on a lightpath is higher than the W_H watermark, a new lightpath is inserted into the network. Routing and wavelength assignment is done by calculating a shortest path on fibers with unused wavelengths and wavelengths are assigned by first-fit. If no such shortest path can be found, existing lightpaths are neither rerouted nor preempted. Even though this algorithm uses preemption for network reconfiguration it is not possible to create lightpath topologies fulfilling any predefined constraints.

In [10] an algorithm for transforming an existing lightpath topology T into a target topology T' is proposed. The algorithm works by calculating $\Delta(T, T')$ as the set of edges being part of T but not of T'. Analogously, $\Delta(T', T)$ is defined as the set of edges to be added to T such that T' becomes a subgraph of T. The transformation process iteratively picks an edge e from the set $\Delta(T', T)$ and tries to create e by establishing a new lightpath. If this is not possible it is evaluated if there exists any lightpath creating an edge in $\Delta(T, T')$ which can be preempted to free resources used to create e. Unfortunately the algorithm does not consider any constraints during the transformation process and it is not guaranteed that the algorithm finds a transformation from T to T'. With increasing size of $\Delta(T, T')$ the success probability of the proposed method raises. However, when serving new lightpath requests on-the-fly, $\Delta(T, T')$ is constantly empty.

Ref. [6] studies the problem of service level agreement (SLA) violations in a WDM network under dynamic connection requests. SLAs specify a period in which a lightpath between two points in the network has to be active along with a availability requirement stating the amount of time the lightpath is allowed to be unavailable during that period. The given algorithm (called ERDRP) uses preemption to minimize the SLA violations over all lightpath requests. ERDRP reconfigures the network on-the-fly and does not require knowledge about future lightpath requests. Unfortunately with SLAs only very basic constraints for the lightpath topology can be defined.

When building a lightpath topology incrementally, the available network resources undergo fragmentation. One method used to deal with fragmentation is rerouting existing lightpaths in case a new lightpath request cannot be handled. Several lightpath rerouting algorithms exist, i.e. [11], [12], [13], [14], [15], assuming either the wavelength continuity constraint or exploit wavelength conversions.

In [12] the "parallel move-to-vacant wavelength retuning" (MTV-WR) algorithm is proposed. MTV-WR is a RWA algorithm that does not exploit wavelength conversion. On the arrival of a new lightpath request between two nodes u and v MTV-WR first tries to find a route for the new lightpath on a free wavelength. If this is not possible the algorithm checks if it is possible to retune existing lightpaths to other wavelengths such that a route between u and v on a continuous wavelength is free. In [13], MTV-WR is extended to MTV-OPA (move-to-vacant one-path-adjusting) for use in networks exploiting wavelength conversion. In addition to altering the wavelength of lightpaths, MTV-OPA is able to reroute existing lightpaths to different paths through the network. Unfortunately the number of different rerouting candidates per request is bounded by one. Thus many possible rerouting candidates are ignored, lowering the chances of successful rerouting.

In [14] the dynamic least congested routing (DLCG) is presented. DLCG is a rerouting technique used to equally distribute resource usage over the network. It works by periodically calculating a new least congested route for each lightpath. If this new route is significantly less congested the corresponding lightpath is rerouted from its primary route to this secondary route. Simulation results show that a) resources of the network are more evenly used and b) by distributing the load across the network the blocking probability for future lightpath requests drops significantly. Inspired from these results, in Section IV-D we present a periodic cleanup method in the context of our RWA algorithm which also tries to evenly distribute the load across the network.

Ref. [15] investigates rerouting based on calculating k shortest paths. If a lightpath request cannot be handled by the network k shortest paths between the two endpoints are calculated. In a subsequent step it is evaluated if lightpaths on these paths can be rerouted such that one of the k paths can handle the new request. Simulation results show that under light load rerouting based on the k shortest paths lowers the blocking probability significantly. But with increasing load the success rate of the presented rerouting algorithm decreases massively. Although our presented rerouting algorithm uses a related technique we use a k shortest path algorithm for calculating a set of distinct rerouting candidates instead of only calculating k alternative routes.

III. Generic Graph model

A. Assumptions and Definitions

We assume a system in which the user of the network has the possibility to actively tell the network operator about its required data rate to another point in the network. We call this a *request*. Every request has a (possibly unlimited) lifetime determining how long the corresponding lightpath has to be active. After a lightpath exceeds its lifetime or upon a release request, it is removed and its associated resources are freed for future use.

Let the physical topology of the optical network be given by the undirected graph $G_P = (V_P, E_P)$ where V_P are the nodes and E_P the fibers connecting those nodes. We assume each fiber can carry a total of W independent wavelengths, denoted by $\Lambda = \{\lambda_1, ...\lambda_W\}$, which are used for data transmission. Each of these wavelengths can be used for transmitting at a fixed data rate S.

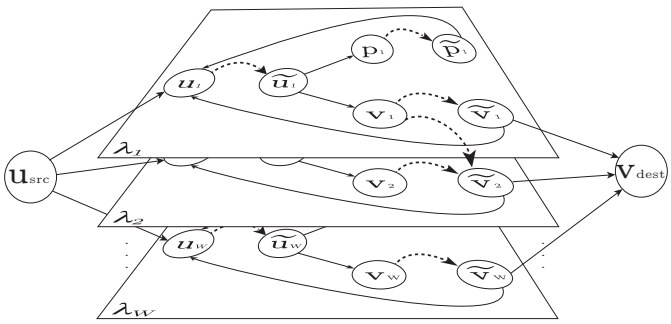


Fig. 2. Layered graph with W wavelengths. Wavelength edges dashed. In this example v is able to pass-through λ_i and to convert from λ_1 to λ_2.

A lightpath $l = \{(e_1, ..., e_m), (\lambda_1, ..., \lambda_m)\}$, $e_i \in E_P$, $\lambda_i \in \Lambda$, consists of a path through the network and a wavelength for each edge on this path. We do not require the lightpath to use the same wavelength on each edge.

A node in the network can be seen as a switch comprising several ports. Each of these switches consists of certain light-splitters (LS) and wavelength converters (WC) [2]. An LS is used to direct a signal from one port to another port and a WC is used to convert a signal arriving at one wavelength to another wavelength. Note that not every switch has the capabilities to switch every input port to every output port, nor to convert every source wavelength to every destination wavelength [2].

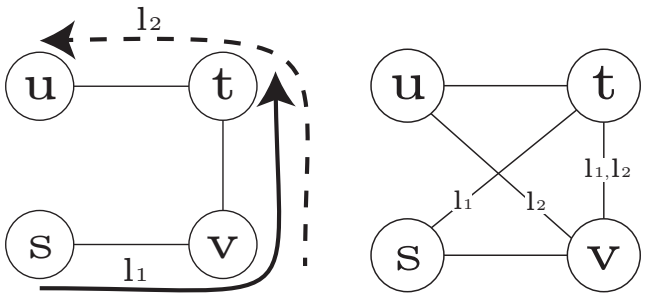
B. Graph Construction

This section presents the layered graph model [4], [5]; readers familiar with this model may want to skip to Section IV. The basic idea of the layered graph is that each of the W available wavelengths can be seen as an independent layer. This idea is visualized in Fig. 2 and the construction is discussed in the following.

We construct the layered graph $G_L = (V_L, E_L)$ of a physical topology G_P as follows: As G_L consists of W different layers, for each node $v \in G_P$ the nodes $v_1, ..., v_W$ and $\tilde{v}_1, ..., \tilde{v}_W$ are added to V_L; they represent v on each layer. For each edge in G_P, $2W$ edges are added to G_L as follows: For each undirected edge $(u, v) \in E_P$, add the edges $(\tilde{u}_i, v_i)$ and $(\tilde{v}_i, u_i)$ to E_L for $1 \leq i \leq W$. In addition, for each node $v \in V_P$, add the nodes v_{dest} and v_{src} to V_L which are used to interconnect the different layers. These nodes are connected to the layers as follows: For each $v \in V_P$, add (v_{src}, v_i) and $(\tilde{v}_i, v_{dest})$ to E_L, $1 \leq i \leq W$. We will now add the wavelength conversion capabilities of each node to V_L: If a node v is capable of converting an incoming signal on wavelength λ_i to λ_j, the edge $(v_i, \tilde{v}_j)$ is added to E_L. We call such an edge a *wavelength edge*. Note that an edge from $\tilde{u}_i$ to v_j is only inserted to represent edges induced by the topology G_P, while edges from u_i to $\tilde{v}_i$ are always wavelength

edges. This graph construction implies full LS capabilities but extending it to limited LS is straightforward to do; we ignore this limitation henceforth.

It is easy to see how to transform a path in G_L to a route and a wavelength assignment in G_P: if a lighpath between the two nodes u and v is requested, a path from u_{src} to v_{dest} in G_L has to be found. Let $p = (u_{src}, u_i, \tilde{u}_j, w_j, \tilde{w}_k, x_k, ..., \tilde{v}_l, v_{dest})$ be such a path in G_L. In addition, let $f : V_L \to \{1, ..., W\}$ be the function that maps a node to the layer it represents and $g : V_L \to V_P$ the function that given a node of the layered graph yields the corresponding physical node. Then from p a route r in G_P can be constructed by first removing every second component of p and afterwards applying f to each remaining component. The corresponding wavelength assignment w is constructed by applying g to the components of p.



(a) Simple graph with two established lightpaths l_1 and l_2 and $W = 2$ wavelengths.

(b) The corresponding conflict graph for lightpaths l_1 and l_2.

Fig. 3. Construction of the conflict graph G_C from a lightpath configuration.

IV. Routing and wavelength assignment

Our RWA algorithm consists of 3 main parts: a) a greedy lightpath selection, b) a method to find conflicts with already established lightpaths in case the greedy strategy could not find a solution, and c) a rerouting algorithm trying to reroute conflicting lightpaths.

These parts are composed to constitute our algorithm as follows: Upon a request for a lightpath between nodes u and v, at first the greedy algorithm is applied. If this algorithm could not find a free path, multiple sets of conflicting lightpaths are computed. Such a set is defined such that after removing *all* lightpaths in it, it is possible to create the requested new lightpath. Now it is checked if one of these sets can be rerouted such that the greedy algorithm is able to find a path from u to v. If this fails, these sets of lightpaths are passed to the requester who can now choose which of these sets has to be preempted. Afterwards, the new lightpath can be constructed by applying the greedy algorithm.

We will now go into detail for each of these parts individually.

A. First-Fit: Greedy shortest path routing

When considering a scenario with dynamic request patterns where new requests are submitted and old requests expire, it is important that future requests can be handled by the system. As there is no information in advance about newly arriving requests, it seems prudent to grant a request using as few resources as possible, maximizing the free resources of the network.

One possibility to use minimal resources to create a lightpath is to use a shortest path between source and destination [2]. This path can easily be found by applying a breadth-first search on the layered graph. As no two lighpaths must share the same wavelength on any edge, all edges of the layered graph have to be removed which are already in use by a lightpath. As pointed out in [4], it is even better to use the least loaded path between two points in the network, where the load of an edge is defined in terms of lightpaths travelling through the corresponding physical fiber. To find the least loaded path in the network a weighted shortest path algorithm such as Dijkstra's can be used.

B. Lightpath preemption

Establishing lightpaths between two points in the network based on the greedy algorithm works as long as there are enough free resources available. But as more and more lightpaths are established the number of free paths in the graph decreases and at some point there is a request which cannot be handled. This section presents a method to find multiple sets of already established lightpaths such that the preemption of *all* lightpaths in *any one* of these sets allows the new request to be handled. In a subsequent step these sets can be analysed to find the most feasible candidate for preemption. This can even be done in an interactive manner: Upon a new lightpath request that cannot be handled by the network, the RWA algorithm calculates multiple sets of preemption candidates. These sets are passed to the requester (which can be another program or even a human being) who is now able to analyse each candidate set. If all lightpaths in one of the sets are to be preempted this set is passed back to the RWA algorithm which preempts all lightpaths in the set and finally establishes the newly requested lightpath.

When multiple applications share the same network at the same time one application might want to preempt a lightpath previously established by another application. Depending on the desired behaviour of the network this should not be possible. To implement an authorization scheme, the RWA algorithm has to keep track which application requested which lightpath. Then each preemption request can be checked against the desired scheme.

To find the preemption candidates for a given request between nodes u and v, we first construct the *conflict graph* $G_C = (V_C, E_C)$ for all established lightpaths and the layered graph G_L. G_C is an undirected graph on the nodes of the physical topology and thus we set $V_C = V_P$. The edges of this graph are labelled with sets of lightpath identifiers where the labels of an edge (u, v) are denoted by $\text{labels}(u, v)$ and are defined by the following two rules:

(1) For each unused edge $(p_i, q_j) \in E_L$, $i, j \geq 1$, create an unlabelled edge (p, q) in the conflict graph. This means that for each pair of nodes in G_P, connected by a fiber having unused wavelengths left, there is an **unlabelled** edge in the conflict graph.
(2) To model the influence of taking down an already established lightpath l, the nodes on the path of l will form *one* strongly connected component in G_C. To this end, between all pairs of nodes on l, an edge is inserted in G_C. The edges of this strongly connected component are **labelled** with the corresponding lightpath identifier but only if there does not already exist an unlabelled edge.
Thus, edges inserted by rule (1) will never get assigned a label by rule (2) and we do not allow multiple edges between two nodes. An edge (u, v) in G_C with labels $L = \{l_1, .., l_m\}$ implies that by preempting one lightpath from L it is possible for the greedy algorithm to create a new lightpath from u to v. Fig. 3 shows the conflict graph for a small graph with two established lightpaths.

In the conflict graph, a path from u to v consisting only of unlabelled edges corresponds to an opportunity to create a new lightpath without interference by other lightpaths (provided that the required wavelength conversion capabilities are available). But since the greedy search did not find such path in the layered graph, no such path in the conflict graph can exist, either. All paths from u to v in G_C hence have *at least* one edge with *at least* one label.

The labels of an edge in G_C describe the lightpaths that are forwarded over this edge. Thus, if we want to create a new lightpath between u and w we would have to remove one of the lightpaths from labels(u, v). Now suppose we have a path $p = \{(u, v), (v, w)\}$ in G_C. To create a lightpath between u and w we would have to remove one lightpath from the set of labels(u, v) and one lightpath from labels(v, w) (and not necessarily the same one). To obtain all possible combinations of lightpaths to remove, we have to calculate the cartesian product of labels(u, w) and labels(w, v). This obviously generalizes to paths with more than 2 hops.

To determine the *complete* list of all possible preemption candidates, all possible paths in G_C between the nodes u and v are required. Then, for each of these paths, the possible preemption candidates can be computed by taking the cartesian product of the labels of each edge on the path and afterwards remove all combinations which are not feasible due to wavelength conversion limitations.

As it is not practical to compute all paths between two nodes in a network [16] we concentrate on finding a fixed number c of paths. These paths are chosen such that there are no two paths p_1 and p_2 with labels$(p_1) \subseteq$ labels(p_2) (where labels$(p) = \bigcup_{(u,v)\in p}$labels$(u, v)$) because otherwise, if we were to preempt p_2, more lightpaths are preempted than necessary. To find these c paths we use a slightly modified version of Dijkstra's shortest path algorithm which has the following properties:

- The algorithm stops if c different paths between a source s and a destination d are found.

```
1: q.enqueue(u);
2: for each node n do
3:    onWay(n) ← {}
4: end for
5: while not q.isEmpty() and |onWay(v)| < c do
6:    s ← q.dequeue()
7:    for each n ∈ neighbors(s) do
8:       newLabels ← onWay(s) × labels(s,n)
9:       newLabels ← newLabels ∪ onWay(n)
10:      newLabels ← reduce(newLabels)
11:      if |newLabels| > |onWay(n)| then
12:         q.enqueue(n)
13:         onWay(n) ← newLabels
14:         if |onWay(v)| ≥ c then
15:            return onWay(v)
16:         end if
17:      end if
18:   end for
19: end while
20: return onWay(v)
```

Fig. 4. Algorithm FINDPREEMPTIONCANDIDATES(u, v)

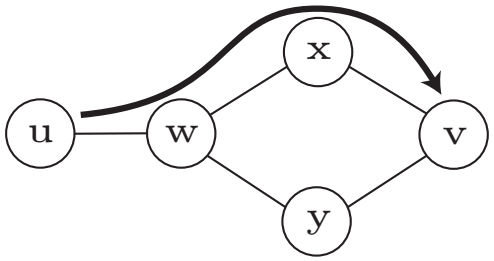


Fig. 5. To grant a request from x to v (with $W = 1$) the lightpath between u and v can be rerouted over y.

- During the computation for each node $n \neq d$ an unlimited number of paths from s to n are stored.
- If there are two paths p_1 and p_2 and it holds that labels$(p_1) \subset$ labels(p_2), then p_1 is shorter than p_2.
- There is no other notion of the length of a path.

This modified Dijkstra's algorithm can be seen in Fig. 4. The function `reduce` is used to remove obsolete elements from the set `newLabels`. To this end, it only keeps the smallest sets of labels where "smaller than" is defined by set inclusion. In addition, this function removes all lightpath combinations which cannot be merged due to WC limitations.

C. Rerouting before preemption

Before any established lightpath is preempted it should be checked if it is possible to reroute one of the preemption candidate sets. Fig. 5 shows a simple example where rerouting is possible. In the example there is a lightpath from u to v via w and x. Now we want to add a new lightpath from x to v. As there are no more free resources to grant this request, we would have to preempt the lightpath from u to v. But as rerouting this lightpath over w and y yields free resources on the edge (x, v),

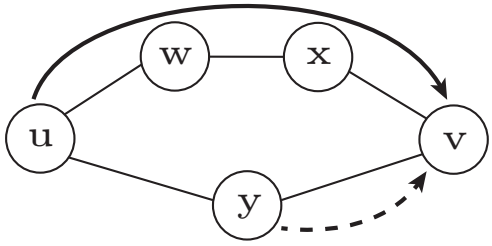


Fig. 6. If the solid shaped lightpath is older than the dashed one, cleanup will not be successful.

the new lightpath request can be granted without preempting the existing one.

Our rerouting works as follows: On the arrival of a request for a lightpath from u to v which cannot be handled, use the algorithm stated in Fig. 4 to calculate the set L of preemption candidate sets. Now it is checked if it is possible to remove all lightpaths of one set $C \in L$, create a new lightpath from u to v, and afterwards reinsert all previously removed lightpaths again in order of their arrival into the network. If the last step fails, we could not find a rerouting for this particular C. But, if this step succeeds for some $C \in L$, then we found a way to insert a new lightpath from u to v without preempting any existing lightpaths. Note that the success of this operation is dependant on the order in which lightpaths are reinserted and that there exist $|C|!$ different insert sequences. As for large C it would be too time consuming to try every sequence we concentrate on the order of arrival only.

D. Periodic Cleanup

After successively granting new requests and removing expired lightpaths from the network, the resources of the network undergo fragmentation. As fragmentation leads to uneven resource utilization and uneven resource utilization yields a higher blocking probability for new requests, we try to counteract fragmentation. This is done by periodically recalculating paths and wavelength assignments for all established lightpath. To this end, we first hypothetically remove all lightpaths from the network. Now we try to reinsert all lightpaths again in order of their arrival onto the empty graph. Note that, again, the success of this operation is dependant on the order in which the lightpaths are added to the network and thus this operation need not always be successful. See Fig. 6 for a configuration where cleanup fails. Only if this hypothetical experiment succeeds, we apply the changes to the real network.

E. Complexity analysis

Our proposed RWA algorithm consists of the three phases greedy path selection, conflicting lightpath computation, and rerouting. We will give a complexity analysis for each of these phases and will then compose them to end up with the overall run time.

For the greedy path selection with "least loaded paths first" Dijkstra's shortest path algorithm is run on the layered graph. The layered graph $G_L = (V_L, E_L)$ of a physical topology $G_P = (V_P, E_P)$, $N = |V_P|$, $E = |E_P|$, with W different wavelengths, consists of $|V_L| = 2W(N+1)$ nodes and, in case of full WC capabilities, of $|E_L| = NW^2 + EW + 2NE$ edges. As W is a constant, it follows that $|E_L| \in \mathcal{O}(NE)$ and $|V_L| \in \mathcal{O}(N)$. The runtime of the greedy algorithm is thus $\mathcal{O}(N \log N + NE)$.

To compute conflicting lightpaths, first the conflict graph is built. This takes time $\mathcal{O}(NE + RN^2)$ where R is the number of established lightpaths in the graph. The conflict graph has N nodes and at most $\frac{(N-1)^2}{2}$ undirected edges, where each edge can have at most R ($\leq WE$) labels. Finding c preemption candidate sets based on the conflict graph is the same as finding c shortest paths on a graph where each single label is represented by one edge. This graph would consist of N Nodes and $\frac{(N-1)^2}{2} \cdot R \in \mathcal{O}(N^2WE)$ edges. Thus, the runtime of finding c sets of preemption candidates is $\mathcal{O}(c(N^3WE + N^2 \log N))$ [16].

The runtime of the rerouting algorithm is cL times the run time of the greedy path selection, where L is the maximum size of a set of preemption candidates. L can be upper bounded by R.

Cleaning up the network consists of two steps: a) remove all lightpaths and b) add all lightpaths with the greedy algorithm. This can be done in $\mathcal{O}(R \cdot (N \log N + NE))$.

The worst case in terms of run time is when neither the greedy algorithm can find any free path nor the rerouting algorithm finds a set of lightpaths that can be rerouted. As finding the preemption candidates is the dominant task the overall run time is $\mathcal{O}(c(N^3WE + N^2 \log N)$.

From the preceding discussion we know that the space required to store the layered graph is $\mathcal{O}(NE)$ and for the conflict graph $\mathcal{O}(N^2WE)$. The space required for calculating c preemption candidates based on the c shortest paths algorithm is $\mathcal{O}(N^2WE)$ [16]. Thus, the space complexity of our proposed algorithm is $\mathcal{O}(N^2WE)$.

V. Simulation

A. Model

For the simulations of our proposed algorithm we used the janos-us-ca network (accessible at http://sndlib.zib.de) which consists of 26 nodes and 42 bidirectional links. The janos-us-ca network is a (fictional) wide-area network which is comparable to large provider backbone networks and is illustrated in Fig. 7. We assume W = 16 and that each node is equipped with full WC and LS capabilities.

We assume dynamic traffic in a circuit-switched network and perform a round-based simulation. At the beginning of each round, a source node chooses a destination node from all the other nodes uniformly at random and generates a bandwidth request with a holding time t measured in rounds. If this request is granted, it will use the allocated lightpath for t rounds.

For the generation of requests, we are using a Poisson process which is a widely used assumption for traffic in WDM networks [4], [5], [17]. We use a Poisson distribution with parameter λ_p to determine the arrival of new requests. The

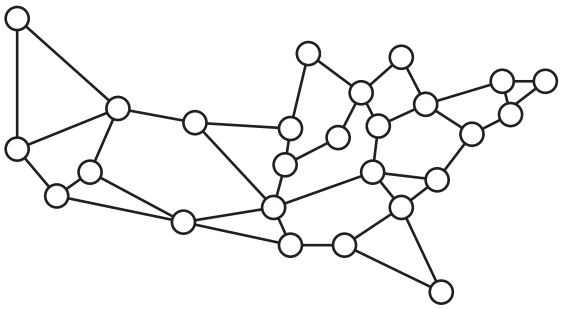


Fig. 7. The janos-us-ca network with 26 nodes and 42 edges.

TABLE I
SIX DIFFERENT CONFIGURATIONS OF OUR ALGORITHM.

Name	c	Cleanup	Path	Time [ms]
BFS	0	∞	shortest	2.15
Dijkstra	0	∞	least loaded	2.66
C BFS	0	5	shortest	1.79
C Dijkstra	0	5	least loaded	2.26
CR BFS	7	5	shortest	19.30
CR Dijkstra	7	5	least loaded	19.96

holding time of each request is exponentially distributed with rate λ_e. All arrivals and holding times are pairwise independent of each other. To create a given load l (in Erlangs) in the network with a mean holding time λ_e^{-1}, we set $\lambda_p = l \cdot \lambda_e$.

B. Results

The metric used to determine the performance of an RWA algorithm is the blocking probability: the probability of denying an incoming lightpath request. In our simulation we did *not* make use of the preemption feature. The given results thus show the performance of our RWA algorithm, where the blocking probability can equally be seen as the probability of having to make preemption decisions. We decided for this kind of evaluation because otherwise, the results in terms of blocking probability would *massively* depend on the kind of application that runs on the network and therefore neither be comparable with other works nor be intuitively interpretable. Future work will show how to create applications that benefit from influencing the preemption decisions but this is out of scope here.

Fig. 8 shows the influence of the number of different preemption candidates c on the blocking rate. As it can be seen, higher values for c are leading to a lower blocking probability. This is no surprise since c is the number of different rerouting candidate sets considered in case there is no free path in the network. The frequency at which the periodic cleanup is executed has another impact on the blocking rate. It is expected that by increasing the frequency the blocking probability decreases. Fig. 9 shows the influence of the number of rounds between two cleanups on the blocking probability, confirming our expectation.

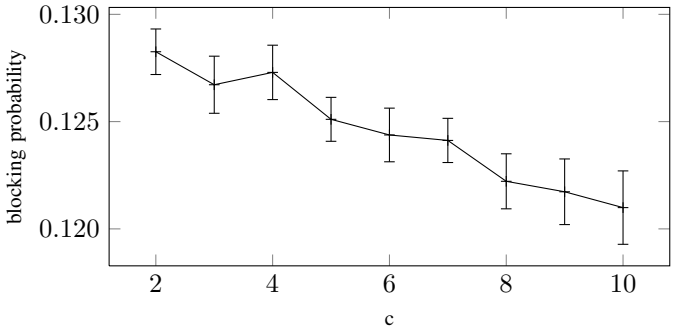


Fig. 8. Influence of number of used labels on the blocking probability. Algorithm run at 127 Erlangs, holding time 20 rounds and 5 rounds between two cleanups. Error bars illustrate the confidence interval for a confidence coefficient of 95%.

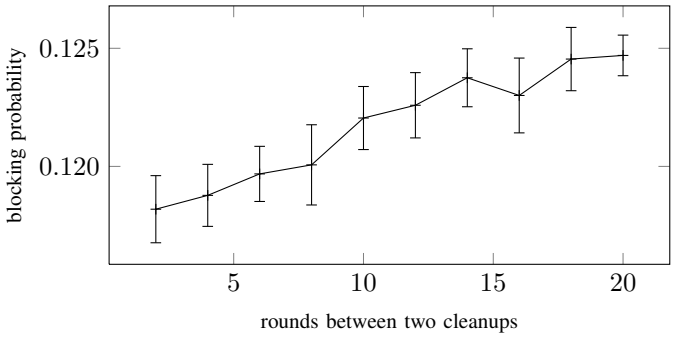


Fig. 9. Influence of the number of rounds between two cleanups on the blocking probability. Algorithm run at 127 Erlangs, holding time 20 rounds and c = 7. Error bars illustrate the confidence interval for a confidence coefficient of 95%.

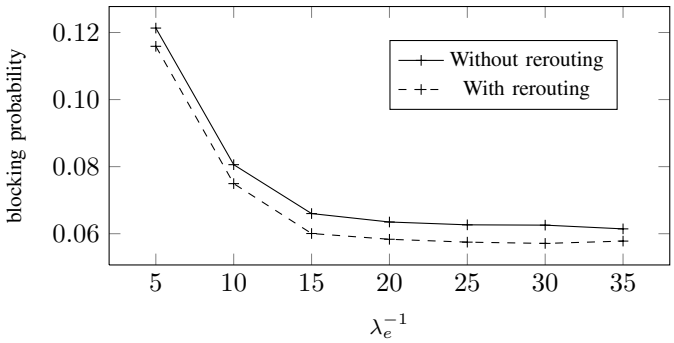


Fig. 10. Influence of the holding time on the blocking probability. Algorithm run at 113 Erlangs, holding time 20 rounds, c = 7 and $\frac{1}{4\lambda_e}$ rounds between two cleanups.

To study the impact of holding time on the blocking probability, we set the rounds between two cleanups to $\frac{1}{4\lambda_e}$ and c to 7. We fix the average load (in Erlangs) and vary the holding time. Fig. 10 shows the results of this experiment with and without rerouting: For $\lambda_e^{-1} > 15$ the blocking probability does not change significantly by increasing the holding time. Thus the holding time does not have any significant influence on the quality of our presented algorithm.

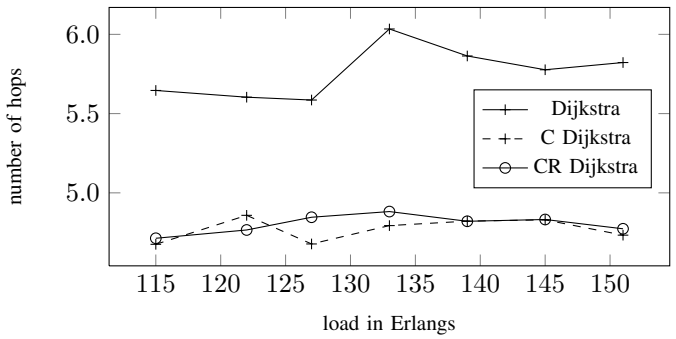


Fig. 11. Average hop count of established lightpaths. Algorithm configurations can be seen in Table I. Holding time 20 rounds.

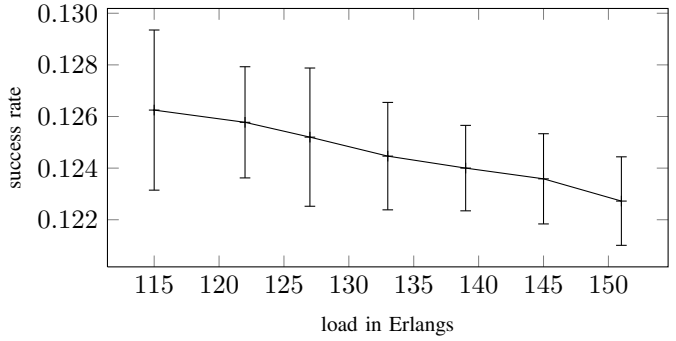


Fig. 12. Success rate of the rerouting operation. Algorithm run with c = 7, 5 rounds between two cleanups, and holding time 20 rounds. Error bars illustrate the confidence interval for a confidence coefficient of 95%.

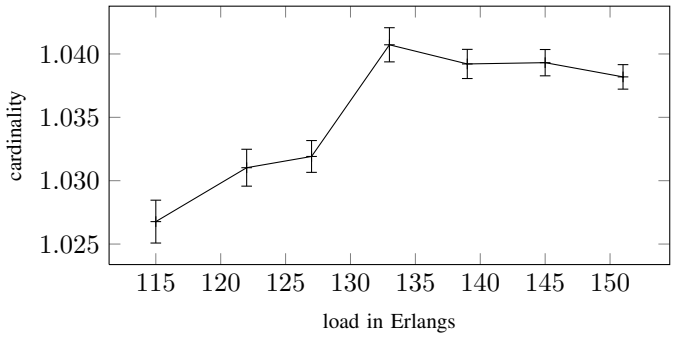


Fig. 13. Cardinality of each set of rerouting candidates. Algorithm run with c = 7, 5 rounds between two cleanups, and holding time 20 rounds. Error bars illustrate the confidence interval for a confidence coefficient of 95%.

Fig. 11 plots the average hop count of the established lightpaths for the considered load levels. As it can be seen the cleanup routine has a significant impact on the average hop count of the established lightpaths since it decreases by up to 17% by periodically cleaning up the lightpaths. When using the rerouting feature in addition to the cleanup routine the average hop count does not increase significantly which is important because this implies that end-to-end latency is not affected by using the proposed rerouting feature of our algorithm.

Fig. 12 illustrates the success rate of the rerouting operation for different load levels. Even in a highly loaded scenario

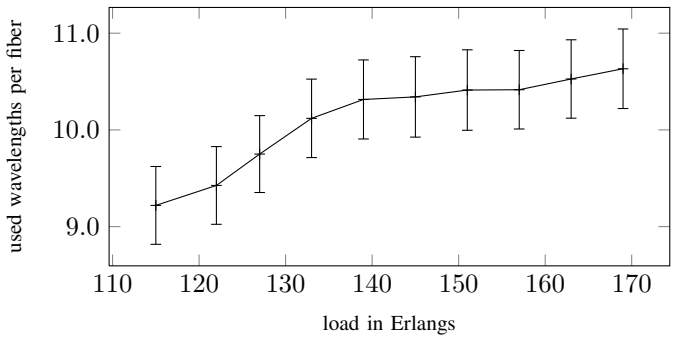


Fig. 14. Average number of used wavelengths per fiber. Algorithm run with c = 7, 5 rounds between two cleanups, and holding time 20 rounds. Error bars illustrate the confidence interval for a confidence coefficient of 95%.

rerouting is possible as the success rate of our algorithm decreases very slow with increasing load. In Fig. 13 the average cardinality of each rerouting candidate is plotted. It can be concluded that (for the janos-us-ca network) even in a highly loaded scenario in most cases it is sufficient to preempt only *one* existing lightpath to grant a new lightpath request.

Fig. 14 shows the spectral efficiency of our RWA algorithm. With increasing load the average allocated wavelengths per fiber increase too. Note that this is mainly because with increasing load the probability of receiving a lightpath request fitting into free wavelengths is also increasing.

For the performance analysis of our proposed method we considered six different configurations of our algorithm. These configurations can be seen in Table I, where c corresponds to the number of considered rerouting candidates (see line 5 in Fig. 4), "Cleanup" is the number of rounds between two cleanups (where ∞ corresponds to no cleanups at all) and "Path" is the method used to find free paths in the network. Note that the first two configurations correspond to the plain greedy algorithm using either the first found shortest path or a least loaded path to set up a lightpath. The next two configurations additionally use periodic cleanup, which is executed every 5 rounds. The last two configurations include the full functionality by adding the rerouting algorithm with up to 7 different candidates. For the simulations we set the average holding time λ_e^{-1} of a lightpath to 20 rounds and simulated 100000 rounds for each of the given loads. The simulation results are plotted in Fig. 15.

It can be seen that with a higher load in the network the blocking probability increases and there are clear differences in the performance of the considered algorithms. The worst blocking probability is achieved by the greedy algorithms. Their results can be significantly improved by using the cleanup routines as the plots of C BFS and C Dijkstra illustrate. Best results are achieved by CR Dijkstra, which indicates that the rerouting feature can be used to lower the blocking probability confirming our conclusion derived from Fig. 12.

The last column titled "Time" of Table I shows the run time of the algorithms serving one request in a scenario with mean holding time 20 rounds and 127 Erlangs load. The results

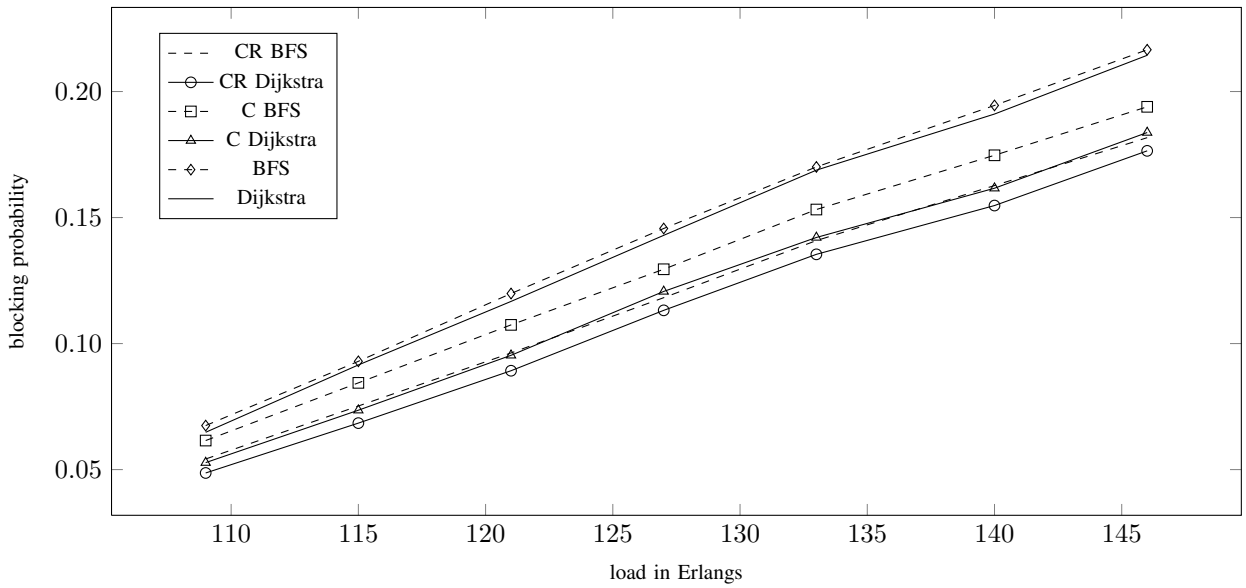


Fig. 15. Simulation results obtained by running 100000 rounds of simulation on the janos-us-ca network.

where obtained on a 2.2 GHz Intel i7 QM with 8 GB of DDR3 memory. As it can be seen in Fig. 15, the performance of C Dijkstra and CR BFS do not differ very much. But as the runtime of C Dijkstra is about 8.5 times lower, it is recommended to prefer C Dijkstra over CR BFS. Even though the presented simulation study was made using the janos-us-ca network simulation on other network topologies (NSFNET and the German National Research and Education Network (DFN)) showed similar results.

VI. Conclusion

Simulation results clearly show that our proposed algorithms can be used to lower the blocking probability in large area backbone WDM networks. We have shown that rerouting of paths can be done at a good balance between algorithmic complexity, practical runtime for relevant example scenarios, and significant improvement of the blocking probabilities. In addition we studied the effect of periodically cleaning up the network on the resulting blocking probability in a dynamic traffic scenario: By recomputing the paths and wavelength assignments of all lightpaths periodically, the blocking probability for future lightpath requests drops significantly.

This work is a step towards an interactive relationship between optical networks and the applications running on top of these networks. Through this relationship it will be possible for the network to adapt to the logical topology of the applications, possibly leading to higher network utilization and improved quality of experience of the applications.

References

[1] C. Meuer, C. Schmidt-Langhorst, R. Bonk, H. Schmeckebier, D. Arsenijević, G. Fiol, A. Galperin, J. Leuthold, C. Schubert, and D. Bimberg, "80 gb/s wavelength conversion using a quantum-dot semiconductor optical amplifier and optical filtering," *Optical Express*, 2011.

[2] R. Ramaswami, K. Sivarajan, and G. Sasaki, *Optical Networks: A Practical Perspective, 3rd Edition*. Morgan Kaufmann Publishers Inc., 2009.

[3] X. Wan, L. Wang, N. Hua, H. Zhang, and X. Zheng, "Dynamic routing and spectrum assignment in flexible optical path networks," in *Optical Fiber Communication Conference and Exposition*, 2011, pp. 1 –3.

[4] G. Shen, S. Bose, T. H. Cheng, and C. Lu, "Efficient heuristic algorithms for light-path routing and wavelength assignment in WDM networks under dynamically varying loads," *Computer Communications*, 2001.

[5] Y. Zhou, G.-S. Poo, S. Chen, P. Shum, and L. Zhang, "Dynamic multicast routing and wavelength assignment using generic graph model for wavelength-division-multiplexing networks," *IET Communications*, 2008.

[6] M. Xia, L. Song, M. Batayneh, and B. Mukherjee, "Event-Triggered Reprovisioning with Resource Preemption in WDM Mesh Networks: A Traffic Engineering Approach," in *Optical Fiber communication/National Fiber Optic Engineers Conference*, 2008.

[7] X. Wei, L. Li, H. Yu, and D. Xu, "Dynamic Preemptive Multi-class Routing Scheme Under Dynamic Traffic in Survivable WDM Mesh Networks," in *High Performance Computing and Communications*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2007, vol. 4782.

[8] G. Lee and T. Choi, "Improving the Interaction between Overlay Routing and Traffic Engineering," in *NETWORKING 2008 Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2008, vol. 4982.

[9] A. Gencata and B. Mukherjee, "Virtual-Topology Adaptation for WDM Mesh Networks Under Dynamic Traffic," *IEEE/ACM Transactions on Networking*, vol. 11, 2002.

[10] S. Sinha, N. Rammohan, and C. R. Murthy, "Dynamic virtual topology reconfiguration algorithms for groomed WDM networks," *Photonic Network Communications*, 2005.

[11] C. Xie, H. Alazemi, and N. Ghani, "Rerouting in advance reservation networks," *Computer Communications*, 2012.

[12] K. Lee and V. Li, "A wavelength rerouting algorithm in wide-area all-optical networks," *Journal of Lightwave Technology*, 1996.

[13] X. Chu, T. Bu, and X. yang Li, "A Study of Lightpath Rerouting Schemes in Wavelength-Routed WDM Networks," in *International Conference on Comunications*. IEEE, 2007.

[14] X. Chu and J. Liu, "DLCR: a new adaptive routing scheme in WDM mesh networks," in *IEEE International Conference on Communications*, 2005.

[15] W. Yao and B. Ramamurthy, "Rerouting schemes for dynamic traffic grooming in optical WDM mesh networks," in *Global Telecommunications Conference*, 2004.
[16] J. Y. Yen, "Finding the K Shortest Loopless Paths in a Network," *Management Science*, 1971.
[17] R. Ramaswami and K. N. Sivarajan, "Routing and wavelength assignment in all-optical networks," *IEEE/ACM Transactions on Networking*, 1995.