Impact of dynamic wavelength assignment strategies on interference in WSON

Francesco Piccinno

Abstract

An effective lightpath routing scheme is proposed, which combines an adaptive routing mechanism [1] with an interference aware frequency band allocation. The proposed solution is able to outperform classic RWA schemes and it allows a much greater quality in optical traffic transmission with the least blocking probability as possible.

1 Introduction

Wavelength Switched Optical Networks (in short WSON) have been attracting considerable interest in the recent years. This interest is demonstrated by the great number of research projects carried out by academia and industry. This short technical paper tries to offer a new perspective in the direction of a new series of dynamic wavelength assignment strategies that account for new metrics such as low average interference level.

The final goal of this work is to present a novel yet simple modification to a classical routing algorithm jointly with a simple slot allocation heuristic, which allows an outstanding performance measured in terms of blocking probability and average interference level.

The conclusions are mainly derived by a centralized simulation study focused on a real topology, namely the European Optical Network (EON) topology depicted in Figure 1. The EON is a transparent all-optical network which interconnects major centers in Europe via high capacity pipes. It is a mesh network composed by a relatively small number of nodes interconnected by links which span several kilometers. In order to provide a worst-case estimate of our solution we treated every single node as a transparent node not capable of carrying out any wavelength conversion.

In this case a given lightpath is subjected to wavelength continuity constraint. The constraint can be expressed as follows: having wavelength w_1 been chosen as carrier for a given lightpath, the wavelength w_1 must be allocated on every link that are part of the computed lightpath.

It is worth pointing out that the proposed solution is essentially a centralized solution with source routing based on a link-state approach. Therefore the scheme can be only adopted on a WSON which adopts a link-state routing strategy, such as OSPF routing or a network employing a centralized Path Computation Element (PCE).

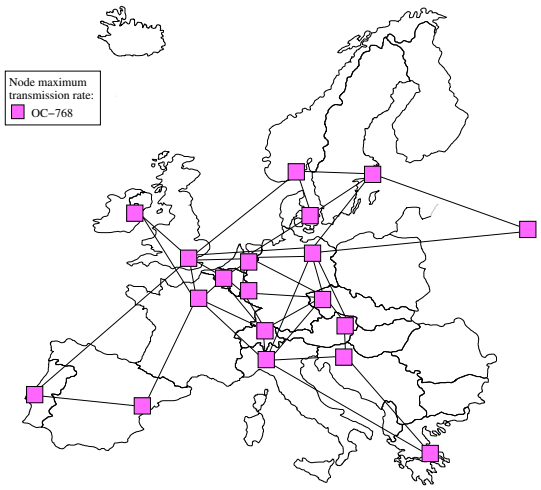


Figure 1: European Optical Network Topology

2 Routing and slot assignment

As already pointed out, a transparent WSON is considered with E nodes and V bi-directional links. Each of the link supports N slots of bandwidth B. Given a pair (S, D) where $S, D \in V$ and $S \neq D$, the routing problem consists in finding a path $(V_1, V_2, \cdots, V_n)$ connecting S to D. In case of a network with no wavelength converters, slot assignment problem consists in finding a slot N_x common to all the links, that can be allocated on the entire path returned by solving the routing problem.

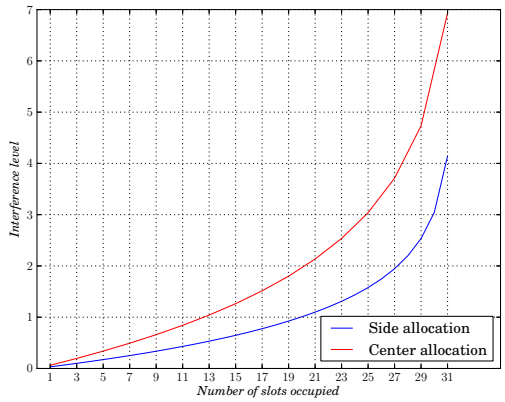
Several algorithms exist that solve these problems by trying to minimize a given metric. The most known algorithm for solving the routing problem, is *Dijkstra's shortest path algorithm*, whose aim is to minimize the cost of the resulting path. On the other hand, *FirstFit* is the simplest and most used heuristic for solving slot assignment problem, and its goal is to minimize the blocking probability.

Since the focus of this work is to propose a solution that offer the lowest average interference level as possible, while not impacting on other metrics, it is useful at this point to introduce Equation 1. The formula can be used to derive the interference level perceived by a given slot x ($x \in S$) on a link L (providing S slots).

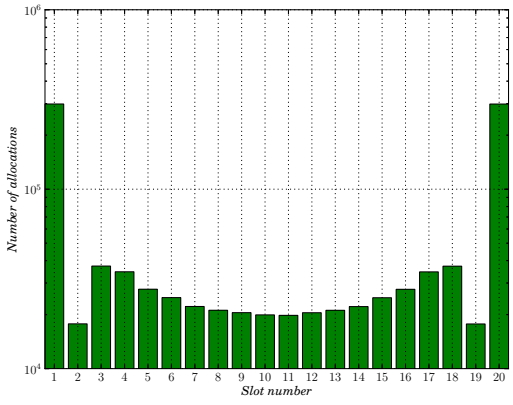
$$\sum_{i=0}^{S_{active}} \log\left(\frac{(|x-i| \times 2) + 1}{((|x-i|-1) \times 2) + 1}\right) \tag{1}$$

In order to better understand the problem and successively design proper heuristics, Equation 1 was used to derive the plots reported in Figure 2. The first figure summarizes the difference in interference level between the *Center* and *Side* allocation on a link offering 32 slots. The *Center allocation* curve was obtained by fixing as the *target slot* the 16-th slot. At each iteration, one free slot from the right (in case of even iteration number) or from the left (in case of odd iteration number) is gradually allocated until bandwidth saturation. Similarly, the *Side allocation* curve was obtained by fixing the 32-th slot as a target slot, while gradually allocating new slots from the right $(1, 2, \ldots, 31\text{-th})$.

The second figure presents an histogram realized by taking in consideration a link with 20 available slots and analyzing all the possible combinations (2^{20}). The x axis represents the optimal slot that minimize the interference level, while the y axis represents the number of times a given slot position was selected.



(a) Central allocation and side allocation



(b) Average interference

Figure 2: General considerations about interference

By analyzing the two figures, you can easily realize that the winning strategy for achieving the minimum interference level is side allocation. Moreover, an interesting fact emerging from Figure 2(b) is that in general is better to avoid the allocation of the second and the second-last slot whenever possible, at least probabilistically speaking.

With this idea in mind, a simple variation of Dijkstra shortest path algorithm was evaluated (Algorithm 1). During its operation the procedure tries to keep in consideration both the distance and the available slots. The objective of the procedure is to select the shortest path in terms of distance which has the greater number of available slots, in order to have more space to introduce optimization. This result is then used in the slot assignment procedure (Algorithm 2).

Regarding slot assignment strategies, four different heuristics were analyzed:

First Fit In this schema the first available slot, starting from the left, is allocated.

First Fit / Last Fit This heuristic allocates the first available slot that is located as farther as possible from the center.

First Fit / Last Fit 2 This heuristic is the same as FFLF with the only difference that it prefers to skip the allocation of the second and second-last slot, if possible. The choice is justified by the histogram shown in Figure 2(b).

MinCost This method allocates the slot that has the minimum cost among all the possible available slots. This method provides a lower bound for the interference level.

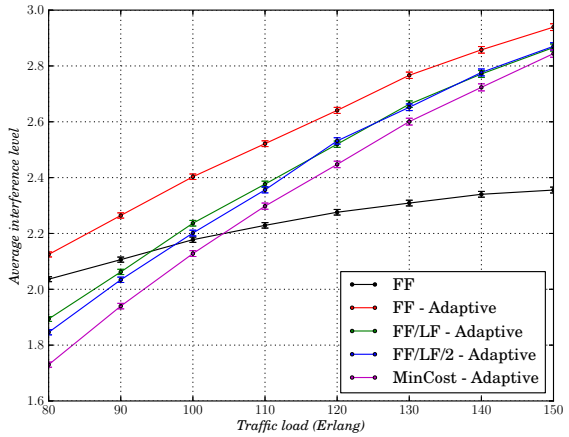
3 Implementation and Results

The performance of the proposed algorithm was evaluated by using a custom Python event-driven simulator. *Lightpath setup requests*, represented as a tuple (S, D), are generated by a Poisson process that is exponentially distributed with parameter λ. Upon reception of a request, the PCE is in charge of allocating a given slot N_x, possibly on the shortest path connecting S to D. If the lightpath allocation is successful, the simulator generates a *tear-down request* event with exponential distribution of parameter μ. A blocking situation occurs in case the setup request cannot be handled, that is all the slots on the given path are already occupied by other lightpath requests.

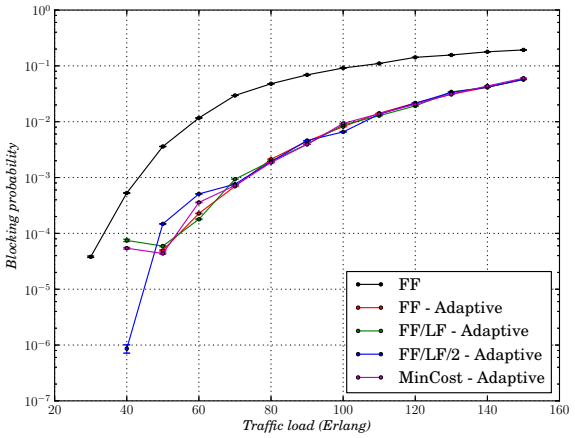
As you can see, the simulator is acting as a centralized PCE, which has all the needed link-state information representing the network state at a given time. The PCE is responsible of maintaining the traffic engineering database with detailed and updated slot availability information.

The results of our simulation are presented in Figure 3 and Figure 3. All the points of the graphs were evaluated with a confidence level of 95%, except for the last two graphs were no confidence interval was considered. The curves were obtained keeping the value of μ fixed to 1 while varying the value of λ in the interval 80 - 150. We analyzed five different strategies:

- Classic routing - First Fit
- Adaptive routing - First Fit
- Adaptive routing - First Fit / Last Fit
- Adaptive routing - First Fit / Last Fit - Skip 2
- Adaptive routing - MinCost

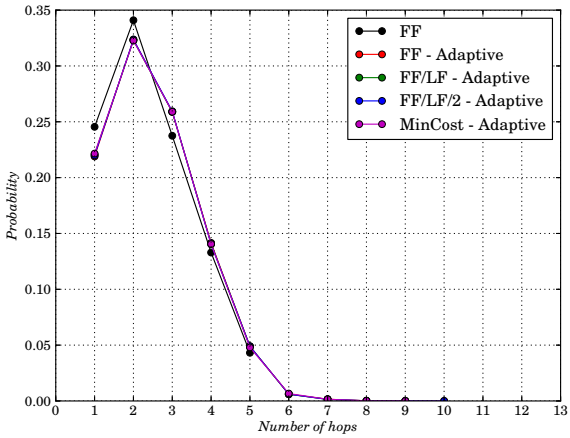


(a) Average Interference Level impact

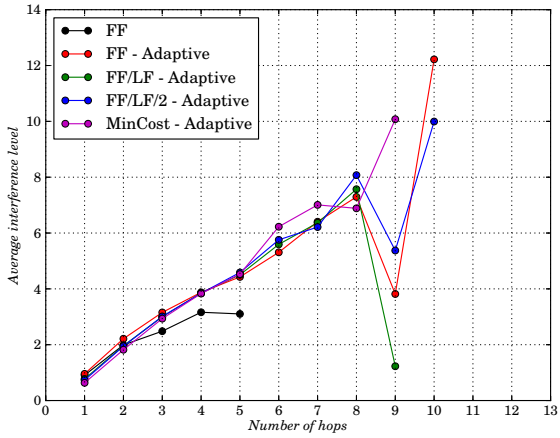


(b) Blocking Probability impact

Figure 3: Average Interference and Blocking Probability



(a) Pdf relative to the number of hops



(b) Average interference level in function of the number of hops

Figure 4: Averaged metrics (80-150 Erlang)

Figure 3(a) shows the Average Interference Level in function of the Traffic Load. By just taking in consideration the adaptive solutions you can see that FF is the worse, while the MinCost constitutes the theoretical lower bound for this metric. Both FFLF and FFLF2 performs almost the same, with a slight advantage on lower load for the latter. In general all the adaptive solutions converges at higher load. The subtle thing to point out in this graph is that although the classic FF solution seems to be best, it is actually the worst solution possible. Indeed it has a lower interference level curve but this result is paid back by an higher blocking probability.

To this end you can take a look at Figure 3(b), which summarizes the blocking probabilities of the various solutions on a larger load range. The plot clearly shows the difference between the adaptive and the classic routing approach and suggests a convergence condition in case of high load traffic conditions. For low load the interpretation is more difficult to draw general conclusions since the results are mainly dependent on the initial random seed and from the network topology.

The conclusions are similar also regarding the PDF graph relative to the number of hops, which is shown in Figure 4(a). As you can see, the classic routing experiences a an abnormal interruption at hop number 5, due to traffic saturation and the relative impossibility of serving any further lightpath setup request.

The last graph, Figure 4(b), shows the behavior of the strategies in function of the number of hops and interference levels. The more you look outside the 0-4 hops interval the less precise is the observation. This is confirmed also by the PDF of the distance previously discussed. As you can see the algorithms presented perform almost the same on this interval. Out of it any conclusion is just speculation, since also the number of samples per point decreases quickly. To draw some kind of conclusion a more long simulation with a very large number of iterations is needed.

Algorithm 1 Slot aware Dijkstra's shortest path algorithm

Precondition: G is the graph while S and D are vertexes.

```
1  function ShortestPath(G, S, D)
2      Distances ← Hash()
3      Paths ← Hash()
4      Seen ← Hash()
5      Queue ← Queue()
6      Paths.set(S, List(S))
7      Seen.set(S, 0)
8      if S = D then
9          return ⟨0, 0, paths⟩
10     Queue.enqueue(⟨0, S, 0⟩)
11     while not Queue.IsEmpty() do
12         ⟨V_distance, V, V_bandmask⟩ ← Queue.ExtractMin()
13         if Distances.Contains(V) then
14             continue
15         Distances.Set(V, V_distance)
16         if V = D then
17             break
18         for ⟨X, E_vx⟩ ← V.AdjacencyList do          ▷ E_vx is the edge object connecting V to X
19             distance_vx ← Distances.Get(V) + E_vx.GetDistance()
20             bandmask_vx ← E_vx.GetBandMask()
21             slots_occupied ←GetActiveBits(bandmask)
22             band ← bandmask_vx ⊕ V_bandmask          ▷ ⊕ is the bitmask OR operation
23             if ¬band ≠ 0 and not Seen.Contains(X) then
24                 Seen.Set(X, distance_vx)
25                 Queue.Enqueue(⟨distance_vx, slots_occupied, V, band⟩)
26                 path_vx ← Paths.Get(V)
27                 path_vx.Append(X)
28                 Paths.Set(X, path_vx)
29     return ⟨Distances, Paths⟩
```

Algorithm 2 Interference aware allocation strategy

Precondition: G is the graph while S and D are respectively source and destination vertexes.

```
1  function SlotAssign(G, S, D)
2      ⟨Distances, Paths⟩ ← ShortestPath(G, S, D)
3      Path ← Paths.Get(D)
4      Distance ← Distances.Get(D)
5      BitMask ← 0
6      for Link ∈ Path do
7          BitMask ← BitMask ⊕ Link.GetBandMask()
8      Position ← Allocate(BitMask)          ▷ Invoke FirstFit, FirstFitLastFit, FirstFitLastFitSkip2 or MinCost
9      for Link ∈ Path do
10         Link.Allocate(Position)
11 function FirstFit(BitMask)
12     return the first free bit in BitMask
13 function FirstFitLastFit(BitMask)
14     return the first free bit, either from the left or from the right, of BitMask
15 function FirstFitLastFitSkip2(BitMask)
16     return the first free bit, either from the left or from the right, of BitMask avoiding second slot if possible
17 function MinCost(BitMask)
18     return the free bit with least possible interference cost
```

References

[1] Bin Zhou, MostafaA. Bassiouni, and Guifang Li, "Routing and wavelength assignment in optical networks using logical link representation and efficient bitwise computation", *Photonic Network Communications*, vol. 10, pp. 333–346, 2005.