

# Effective implementation of void filling in OBS networks with service differentiation

Giovanni Muretto, Carla Raffaelli, Paolo Zaffoni

*DEIS – University of Bologna*

*viale Risorgimento, 2 – 40136, Bologna – ITALY*

*{gmuretto, craffaelli, pzaffoni}@deis.unibo.it*

## Abstract

*In this paper the problem of channel reservation in Optical Burst Switching nodes is considered with the aim to reduce the scheduling processing time and, at the same time, to maintain burst loss probability as low as possible. A new implementation of the void filling problem related to a multi-class traffic environment is proposed based on the binary heap tree data structure. Fast search of the void intervals for burst scheduling is achieved by means of vector implementation of the tree that contains information about voids. The search is further optimized by reducing the search interval to obtain a good trade off between processing time and burst loss probability in relation to traffic load. Results obtained by simulation show interesting improvements of the scheduling time compared with other implementations of void filling, while, at the same time, maintaining burst loss probability low, especially for high quality traffic.*

*Keywords: Optical Burst Switching (OBS), scheduling, void filling, quality of service (QoS).*

## 1. Introduction

In recent years Optical Burst Switching (OBS) has obtained growing attention in the research community and industry [1][2]. It is claimed to combine the best of optical circuit and packet switching with the aim to dynamically exploiting the huge bandwidth made available by fibers [3][4][5]. OBS networks can be viewed as a possible solution for the implementation of high-speed optical backbone that efficiently interconnects peripheral IP networks. To this end packets are assembled into burst at ingress edge nodes and disassembled into packets at network egress. The main property of the burst switching concept is to separate the transmission of data from control information to make control processing more feasible. More specifically, the OBS paradigm provides that a control packet is sent before the transmission of each data burst for resource reservation at each intermediate node along the edge-to-edge network path, giving rise to an offset time between data and control itself. The control packet is typically delivered out-of-band by using common channel signalling, e.g. using a separate wavelength, and can be transmitted with an

extra-offset in advance to implement service differentiation techniques [6].

The performed reservation is one-way and can be based on different approaches [7], which differ for the length of the interval during which network resources are available for a given burst. An efficient exploitation of network resources is achieved by the so-called JET (Just Enough Time) algorithm [7][8][9]. In JET the reservation at each node is based on the expected burst arrival time,  $r$  and lasts for a period of time equal to the burst length,  $l$ . The arrival time is calculated taking the difference between the offset time and the processing time accumulated by the control packet at previous nodes. It is important to point out that by maintaining the second term low it is possible to limit the amount of the overall offset.

This reservation in advance causes the rising of void intervals over different wavelengths, as shown in figure 1, that represents a challenge to optimize bandwidth utilization. As a consequence a primary target in OBS networks is to implement efficient scheduling of burst transmissions in order to try to suitably fill voids with new arriving bursts, ensuring in this way an efficient use of the available bandwidth, possibly without penalizing network performance.

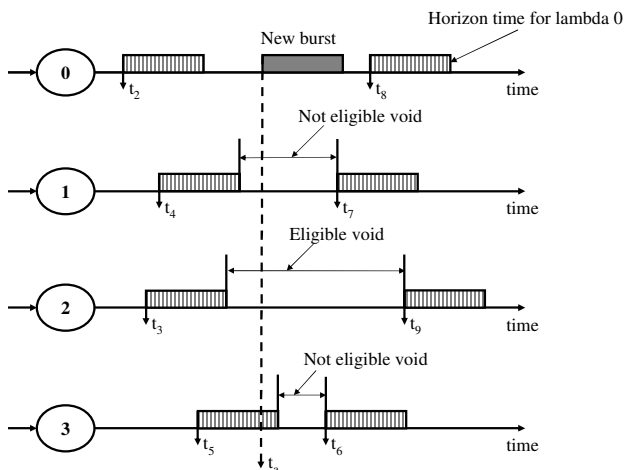
Moreover in this work a service differentiation is implemented through the usage of extra offsets assigned to different classes. In order to obtain a good class isolation long extra offset must be used. This leads to the creation of very long void intervals that must be exploited to keep loss probability as low as possible. The organization of these intervals in a proper structure is also crucial.

In order to support the reservation function, tree data structures are typically built at each node to appropriately organise all the available void intervals for each channel. So, the processing time to find a suitable channel for a burst is strictly related to the searching method used for the investigation of the tree. Nonetheless, these structures grow with the number of available intervals and require to be updated whenever either a new interval is generated or an existing interval is no more available, being it already assigned to a burst or expired. Such operations contribute to the overall processing time needed for scheduling. Moreover, it is worthwhile remarking that a scheduling algorithm is as much efficient as it is able to process a control packet and find a suitable void quickly and smartly enough to guarantee the reservation for the related

arriving burst. So both implementation and consequent void intervals play a significant role in satisfying the above-mentioned requirements.

The tree structures presented in literature and used for solving the channel scheduling problem are typically implemented by means of linked lists [10][11]. The crucial requirement is to resort to memory allocation functions and pointer management that are time consuming tasks. The aim of this work is to apply a new tree organization model for burst scheduling, based on the *binary heap* concept, and prove with several numerical evaluations that the proposed data structure and related major management operations permit to obtain efficient void filling scheduling. The key aspect of this approach is the implementation of the tree as a vector, whose elements are directly accessible through their related indexes. The processing time and the burst loss rate are calculated to prove the effectiveness of the proposed implementation and compare its performance with respect to other existing algorithms presented in literature.

The paper is organized as follows. Section 2 briefly reviews existing literature and states the problem, while section 3 describes the proposed solution. The implementation and the related measurement methodology together with discussions of the obtained results are presented in section 4. In section 5 conclusions are drawn.



**Figure 1. Wavelength usage fragmentation in time due to burst scheduling based on JET, with a new burst starting at  $t_a$ .**

## 2. Problem description

The main aspect of OBS networks that will be here considered is the presence of the offset time and of the related void interval that arises as a consequence of resource reservation, which resource reservation starts when the control packet arrives at the node and is kept for the whole time duration of the burst itself. Taking into account that bursts do not get to a node one right after another, void intervals in channel bandwidth usage arise. As already mentioned, one of the most-used reservation method is the JET protocol, which leads to efficient bandwidth exploitation providing that scheduling algorithms are implemented to utilize this unused time interval. In fact, the created void can be used by other

management of the data structure used for organising the bursts to achieve good bandwidth utilisation and a consequent reduction of the burst loss rate. Offset time between control packets and burst is about some microseconds. Nonetheless, JET can be used for Quality of Service (QoS) differentiation by assigning different extra offsets to diverse traffic classes of service [2]. This extra amount of time is then added to the offset time, which is normally assigned to each burst. Larger extra offsets will be assigned to higher priority classes with respect to the lower priority ones. In this way, resources for highest-priority bursts can be reserved more in advance with a consequent reduction of the corresponding drop probability, which can be improved of several orders of magnitude in comparison with the undifferentiated traffic case [2]. The problem to exploit the void intervals is usually called *void filling*. As regards, the introduction of QoS management in OBS networks leads to an increase in the number of idle time intervals (voids) available on different wavelengths of a given fiber and so, an efficient interplay between smart scheduling algorithms and the JET approach is required to properly exploit these intervals and improve the overall network performance. Different approaches have been proposed in literature to execute this task and try to achieve the best trade-off among performance, complexity and scheduling delay [12] [13].

OBS can also take advantage of the usage of Fiber Delay Lines (FDLs) settled in each node, where bursts can be stored until a channel becomes available, but this solution undoubtedly complicates the design of scheduling algorithms. In fact, by using discrete delay unit as FDL more voids are created and hence the complexity of void filling problem becomes strictly related with the dimension of the optical buffer and depends on whether the transmission is synchronous (or not) or if the length of the bursts is variable (or not) as explained in [5].

So far, some scheduling algorithms have been already studied and discussed in terms of time scheduling and performance.

The first one called HORIZON has been described in [2]. It can be seen as an extreme case where no void filling is made basically. This algorithm considers the horizon time for a given channel as the time after which no reservation is applied and so the next arriving burst can only book this channel after the horizon time. This algorithm results very simple and fast but also bad performing.

In [7] a smarter algorithm called LAUC-VF (Latest Available Unused Channel with Void Filling) is presented. In LAUC-VF the key information about voids (i.e. starting and ending time), or at least those whose ending time is greater than the current time, are stored. Among all eligible void intervals for a specific burst the latest is chosen (i.e. the one with the latest starting time). LAUC-VF is proved to perform better than Horizon but it also results a very slow scheduling solution, which may cause failed reservation. This is due to the impact of both storage and research operations on voids information on the computational time.

In [14] the LGVF (Least Gap Void Filling) is presented. This algorithm only stores information about the least void interval (the one with largest starting time)

improving LAUC-VF performance especially for scheduling time.

In [12] more accurate algorithms are investigated especially from the scheduling time point of view. In particular, Min-SV (Minimum Starting Void) and Min-EV (Minimum Ending Void) are presented in a case without FDLs. Min-SV performs as good as LAUC-VF but provides a scheduling time comparable with the Horizon time and so much better than LAUC-VF. Min-SV performs a bit better than Min-EV but it takes longer to schedule a void interval for an arriving burst. Min-SV selects a void interval between those eligible for a given burst, minimising the time gap between the starting time of the void and the arrival time of the burst. Min-EV minimises the gap between the arrival time of the last bit of the burst and the ending time of the void. For both algorithms void interval information are kept in a balanced binary search tree implemented with pointers. Within this structure burst query, interval insertion and interval deletion operations are carried out. The last two algorithms, Min-SV and MIN-EV result to be the best performing in terms of trade-off between burst loss rate and scheduling time.

### 3. Implementation of the scheduling algorithm.

This section presents the proposed void filling implementation algorithm for channel scheduling in OBS networks. The idea on which this algorithm is based is here described together with the characterisation of the data structure built to be exploited for a smart organisation of the available void intervals, which represents one of the key aspects of the effectiveness in performance of the presented approach.

#### 3.1 General assumptions

Although it has been shown that FDLs can be used in OBS networks, in this study a pure loss OBS environment is considered and no storage capabilities exist for bursts at the switching. Hence, if a burst cannot complete the reservation it is dropped. More specifically, a scenario with QoS management is considered. Three different classes of service are implemented. As mentioned above, different values of extra offset are assigned to bursts belonging to different classes. In particular, extra offset is zero for low priority class,  $3*L$  for the intermediate class and  $9*L$  for the high priority one, where  $L$  is the average burst length equally set for each class of service. This choice has been made in order to achieve a good level of class isolation as explained in [15].

The scheduling algorithm here proposed is called *HVF* (*Heap Void Filling*) and works as follows. When a setup message is received all wavelengths belonging to the

output link are investigated. If at least one wavelength whose horizon time is shorter than the arrival time of the burst is present, the burst is sent on that wavelength. If more of such wavelengths are available, the one that minimises the time gap between two consecutive packets is chosen.

If no wavelength satisfying that time constraint is present, the algorithm looks for one of the *eligible void interval* previously created and in particular the oldest one (that one with the shortest starting time) is chosen. Clearly, an *eligible void* means an interval whose starting time is shorter than the burst arrival time and with an ending time that is larger than the one of the tail of the burst itself. The burst is clearly dropped if a suitable interval is not available.

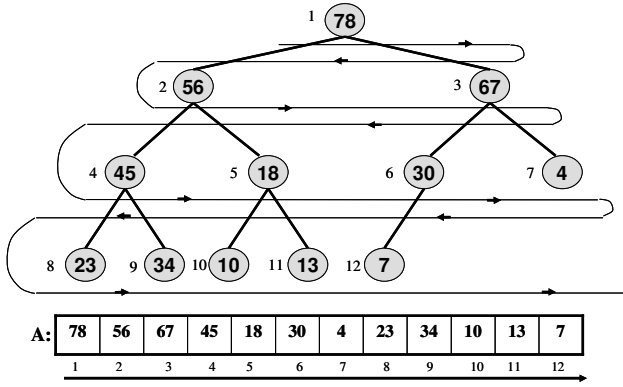
#### 3.2 Binary heap as data structure

The data structure used by HVF to treat information about void intervals is a kind of tree represented by a *binary heap* [16]. A binary heap can be intended as a balanced research tree, which allows a partial ordering among elements.

The ordering imposes the key of a child to be lower than or equal to the father. The key used to sort the tree is the starting time of the interval. In the root there is the newest interval, in the leafs the oldest. The condition of balanced tree must be kept for each new insertion and so the node has always two sub-trees with the same weight. A binary heap stores information of a single node state and each element of it represents a void interval and carries the following key parameters:

- the starting time of the void interval ( $T_a$ );
- the ending time of the void interval ( $T_e$ );
- the wavelength of the considered channel ( $w$ );
- the number of void intervals in the whole set of wavelengths belonging to the link ( $N_i$ ).

As far as the implementation is concerned, the binary heap consists of a vector (or array) whose access is proved to be less time consuming if compared to a pointers-based structure. The drawback of using a vector is that its maximum dimension has to be decided a priori in a static way. In figure 2 an example of a binary tree and its implementation on an array is shown.



**Figure 2. Example of a binary heap implemented with an array data structure.**

If  $A$  is assumed to be the used array, root occupies position  $A[0]$  and a generic element  $i$  that occupies position  $A[i]$  has its left child in position  $A[2i]$  whereas the right one is in  $A[2i+1]$ .

### 3.3 Operations within the binary heap and their complexity

The data structure is assumed to be supported by the three main operations described in the following:

- *Insertion*: a new void interval is inserted in the last position and its starting time is compared to that of his father: if it is largest the two elements are swapped. Complexity results  $O(\log n)$  being  $n$  the number of elements of the tree.
- *Refresh*: this operation is made when  $n$  reaches the maximum array dimension. All intervals whose starting time is shorter than the current time are removed from the tree. After that, a specific function is called in order to update the structure and respect the aforementioned binary heap features.  $O(n \log n)$  is the related complexity
- *Search*: among all eligible void intervals, the one with the largest starting time is chosen. As a consequence, the whole tree must be inspected since the used hierarchy only assures a partial ordering. If the interval exists then it is selected and removed from the binary heap, which is updated immediately after. In figure 2 the arrow indicates how the search would be done in that specific case. The resulting complexity is also  $O(n \log n)$ .

The pseudo-code of the HVF algorithm is presented below and describes its behaviour when a burst gets to a given node at the *current\_time* instant.

```

begin{HVF algorithm}
step 1:
for(i=0;i<=W;i++){ /*explore all the
W wavelengths of the output link */
    look_for(i); /* search for the
wavelength w that minimizes the gap
*/
}
if(w is found){
    if(class!=0) {
        insertion(new_void); /* Insert
the new void */
        if( $N_i$  == Array_ Max_Dimension)
            refresh(current_time);
    }
}
new_time_horizon(w)= $T_a$ +extra_offset+bu
rst_length; /*update time horizon*/
return channel; /* report
the selected channel */
}
else goto step 2/* channel not found
*/

```

### 3.4 Optimization of the search procedure

The *search* operation takes longer time than the other two and mostly affects the scheduling time when the ingress load is high because it is the most frequently called procedure with highest complexity. In fact, when the traffic is high (i.e. 80%) bursts belonging to the low priority class especially need to exploit a void interval to make a reservation and so the binary heap requires to be explored very often.

An attempt to reduce the overall scheduling time by reducing the search time has been done afterwards. This is possible thanks to the implementation of the binary heap on a vector, where a particular position is directly accessed knowing its index. The procedure consists in calculating off-line the sample mean  $\mu$  and the sample variance  $S^2$  of the position of the array containing the useful void interval by referring to a Gaussian distribution as a direct consequence of the Central Limit Theorem. This assumption is a quite good approximation taking into account both the very high number of bursts typically involved and searches on the vector access the scheduling algorithm has to perform. The search interval  $[L_1, L_2]$  is calculated for each class and each input load per wavelength in a way such that  $P_\mu(L_1 < \mu < L_2) = (1 - \alpha)$ . The parameter  $\alpha$  that belongs to  $[0,1]$ , determines the probability to have  $\mu$  falling in the search interval and it can be seen as an index of approximation. The scheduling algorithm refers then to an interval centered in that position, which results shorter than the one already existing and in any case it contains most of the void values of interest. At that time, the search operation is

executed with respect to this shorter interval and the last eligible void interval (if any) is chosen.

Reducing the search range of available intervals allows a significant improvement in performance of the scheduling algorithm as exposed in the next section where simulations results for different values of  $\alpha$  are presented.

#### 4. Numerical Evaluations

In this section performance of the scheduling algorithm presented in section 3 are shown. Results have been obtained by means of an ad-hoc, event-driven simulator of the OBS node and simulations have been conducted on a DELL PC with a Pentium 4 CPU (1.3 GHz). The numerical evaluations concern the different contributions to scheduling time that are compared with those of other approaches already known in literature [2][7][14]. Performance results in terms of burst loss probabilities are also presented.

A single switching node characterized by 4 input and output fibers with 32 wavelengths each has been considered. The input traffic is generated according to a Poisson distribution. Burst size is exponentially distributed with average value  $L$  equal to 5 Mbit, which corresponds to a burst duration in the range of milliseconds at Gbit/s link speeds. As previously mentioned, three different classes are considered. More specifically, the low priority class, namely *class 0*, has an extra offset equal to zero, while *class 1* and *class 2* have an extra offset equal to  $3*L$  and of  $9*L$ , respectively. In this work it has been assumed that only voids created with a positive extra offset are considered. The reason for this statement is that class 0 creates short voids compared to those created by class 1 and 2 and their possible exploitation results very modest. It follows that *class 0* doesn't create any void and consequently this class makes no insertion operations. Actually, it has been shown that this technique achieves good service differentiation in terms of burst loss probability [7]. The whole traffic is assumed to be equally divided among the three classes. The number of simulated burst considered for testing performance of the proposed algorithm is twenty million.

##### 4.1 Scheduling time for HVF

Let's now focus the attention on the scheduling time of HVF, whose average value is here indicate as  $T_{sch}$  and can be expressed as follows

$$T_{sch}^i = \delta_s^i + \delta_l^i,$$

where  $\delta_s^i$  and  $\delta_l^i$  represent the average searching time and the average insertion time for the corresponding class of service, which is distinguished with the apex "i". Figure 3 shows the average time taken by the single procedure call. For the void insertion time, only class 1 and 2 are considered since they are the only classes that create

voids. Refresh time is included in insertion time given that they are strictly related and, in any case, the contribution of the first one is negligible. The evaluations of the average time have been obtained through the function *clock()* provided by the Programming Language C - ANSI. In particular this function has been called at the starting time, *tckstart* and at the ending time, *tckend* of the procedure to be evaluated. The variable *events* stores the number of times the procedure has been called and so the average duration *tckaver* of the procedure is then given by:

$$Tckaver = Tcktotal / events$$

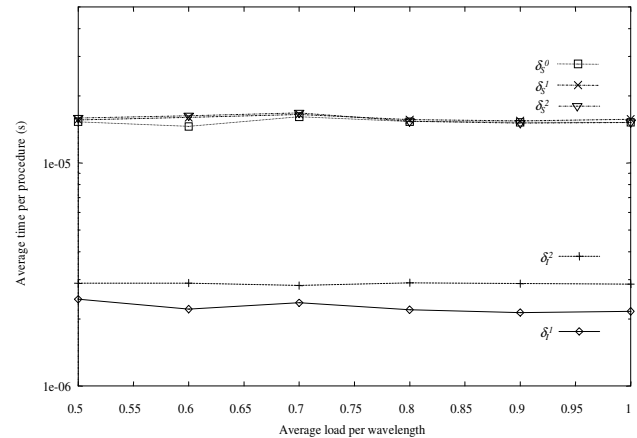
Or, if expressed in time units, by:

$$Taverage = tckaver / CLOCK\_PER\_SEC$$

where *CLOCK\_PER\_SEC* is a constant value, which contains the clock period expressed in time units (e.g. seconds).

The procedure *eval* used to perform these evaluation is shown below.

```
/* procedure eval */
{
  tckstart = clock();
  procedure();
  tckend = clock();
  tcktotal += (tckend - tckstart);
  events++;
}
```

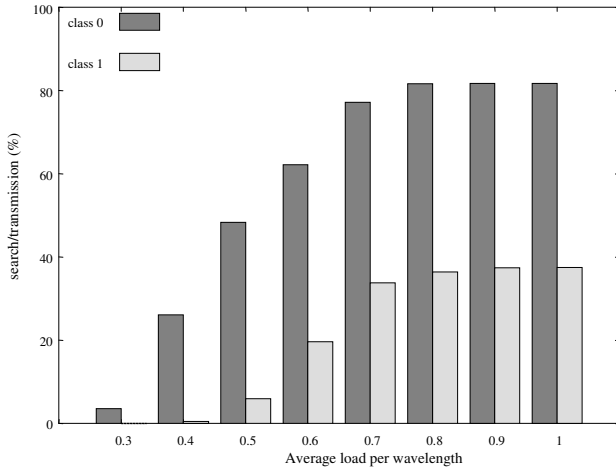


**Figure 3. Average time in seconds for single insertion and single searching procedures executed by HVF for different classes as a function of the average load per wavelength .**

By observing figure 3, it is quite clear that the search time (15-16  $\mu$ s) is dominant if compared to the insertion time (2-3  $\mu$ s) regardless of the class is considered. This result can be explained taking into account that when the searching procedure is called, the whole binary heap has to be explored independently by which class is considered. As regards the insertion, *class 2* is slower than *class 1* because its requests are made more in advance and so more steps are needed in order to find the correct position in the binary heap for the new void to be

inserted. It is also interesting to note that the trend of the curves is independent of the average traffic load and in any case the overall search time depends on how many times each class calls the procedure.

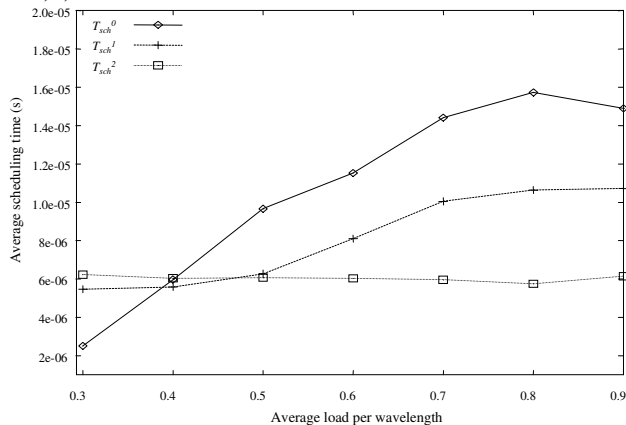
In figure 4 the percentage of search requests as a function of the amount of transmitted data is shown.



**Figure 4. Percentage of searching operations for different values of load per wavelength for class 0 and class 1.**

As expected, *class 0* exploits the search of a suitable void interval to make a reservation much more often than the other two classes and this happens at any load value. *Class 2* calls the search procedure a number of times nearly equal to zero and for this reason it has not been considered in the figure. This result is due to the fact that the large extra offset of this class allows to find a wavelength much more in advance and without the need of a void interval.

In Figure 5,  $T_{sch}^i$  experienced by HVF is shown for  $i=1,2,3$ .

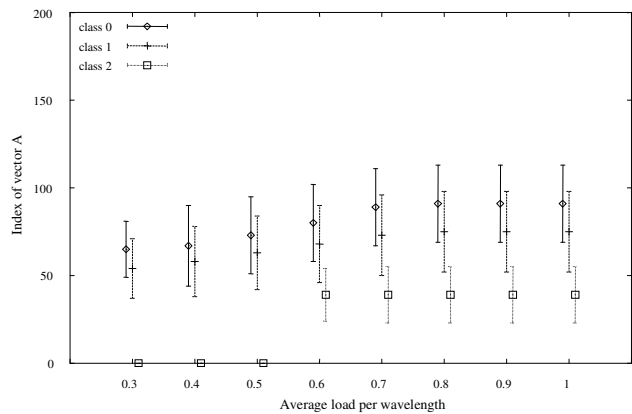


**Figure 5. Average total scheduling time in seconds for the different classes as a function of the average load per wavelength.**

The scheduling time for *class 0* coincides with the search time since this class makes no insertions, as could be obtained by multiplying corresponding values of curves depicted in figure 3 and 4. The graph shows that for *class 0* and for low values of traffic load, the average scheduling time is driven by the insertion time, whereas for higher traffic load values the contribution given by the searching operation becomes dominant. This suggests that for low traffic load the usage of the search procedure is limited. Nonetheless, the decreasing trend of the curve for very high loads (e.g.: 0.8 and 0.9) is due to the fact that in this interval losses for *class 1* and *class 2* become relevant. As a consequence a smaller number of voids is created and thus a smaller binary heap has to be explored. Moreover, the searching time for *class 1* becomes the most important contribution to the average scheduling time when load gets higher. *Class 2* experiences an average scheduling time, which remains basically constant, proving again that this class needs to call the searching procedure very rarely.

#### 4.2 Reducing scheduling time for HVF

This results confirm that to further reduce the overall scheduling time it can be useful to limit the time for the running of the search procedure, as discussed in section 3. This target can be met by limiting the searching interval thus reducing the portion of the binary heap that is explored to find a suitable void. Figure 6 presents the evaluation of the width of the search interval obtained with  $(1 - \alpha) = 68\%$  and the related sample mean of the index containing the void to be used, calculated for different values of the average traffic load per wavelength, with a maximum dimension of the array set at 200.



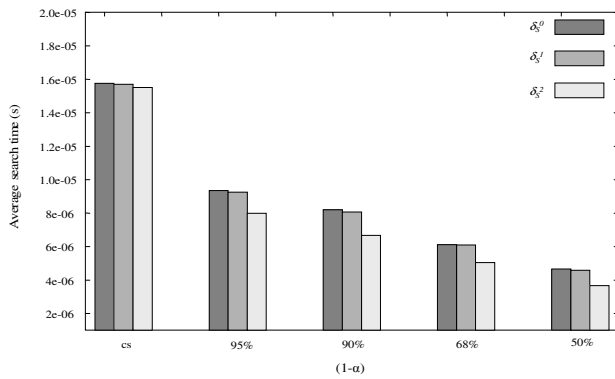
**Figure 6. Sample mean values and related search interval widths for the three classes of service.**

As expected the more the load grows, the higher is the position of the sample mean (central value) and the wider is the search interval. For instance, when load is equal to 0.7, *class 0* has a search interval centred at the 89th

position with width equal to 44. So, the search would start at the  $(89 - 22) = 67$ th position of the array and would take the time to explore 44 positions of the array. Simulation results prove that *class 2* has always narrowed search intervals and thus shorter searching time if compared to the other classes of service. It is important to remark that the searching time was previously the same for each class since the whole tree had to be explored independently by the class as shown in figure 4.

Numerical evaluations presented in the following have been obtained with experiments conducted by applying these new concepts in order to test how burst loss probability and scheduling time may change. It is foreseeable that the loss rate would increase whereas scheduling time would clearly improve, since the searching procedure does not explore the whole tree (and so all available information). Different values of  $(1-\alpha)$  are considered (50%, 68%, 90% and 95%) and the resulting performance is compared with the previous case, called *complete search (cs)* in the following figures. The average load for wavelength is set to 0.8.

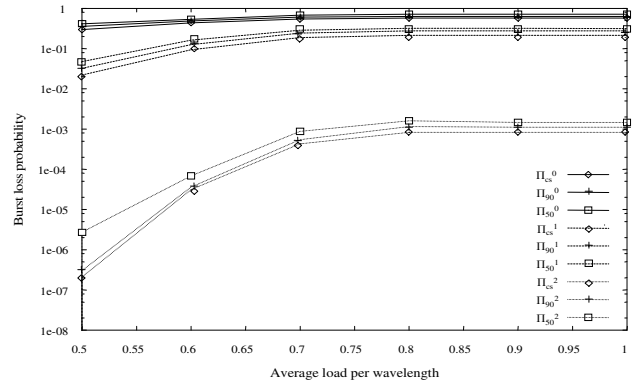
In figure 7 the trend of the overall searching time  $\delta_s^i$  as a function of  $\alpha$  is depicted.



**Figure 7. Average search time in seconds as a function of  $\alpha$  for load per wavelength equal to 0.8.**

The improvement of the average search time involves all classes significantly. *Class 2* experiences a further improvement compared to the other two classes because it benefits of a narrower search interval.

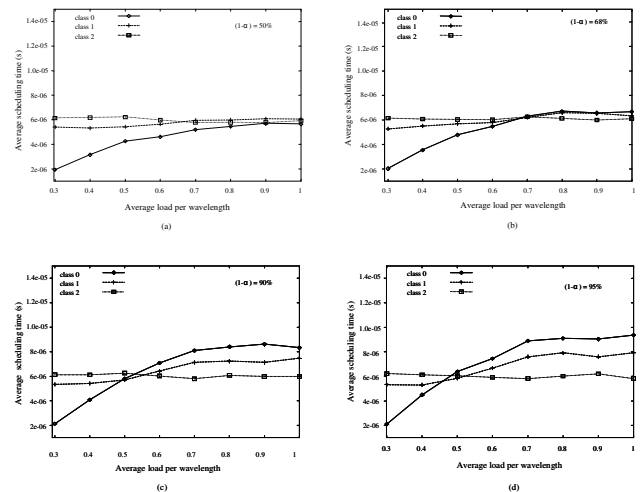
By denoting with  $\Pi_{1-\alpha}^i$  the burst loss probability for *class i* at  $(1-\alpha)$ , the effects of different values of  $(1-\alpha)$  on this probability are presented in figure 8 as a function of the average load per wavelength, being  $\Pi_{cs}^i$  the notation for the complete search performed throughout the array.



**Figure 8. Burst loss probability as a function of load varying  $\alpha$  as a parameter.**

As expected the curves of the complete search represents the lower bound. It is possible to observe how loss probability increases very slightly maintaining the same order of magnitude independently of  $\alpha$ . Actually, this represents an appreciable result together with the gain obtained for the scheduling time by considering that the only price to pay is in terms of a limited worsening for the loss probability.

Figure 9 shows the overall average scheduling time for the new search time for different values of  $(1-\alpha)$ . It is possible to note the improvement that can be obtained by reducing the running time for the search procedure. This improvement involves basically only *class 0* and *class 1* since, as already said, *class 2* nearly does not use search procedure at all.

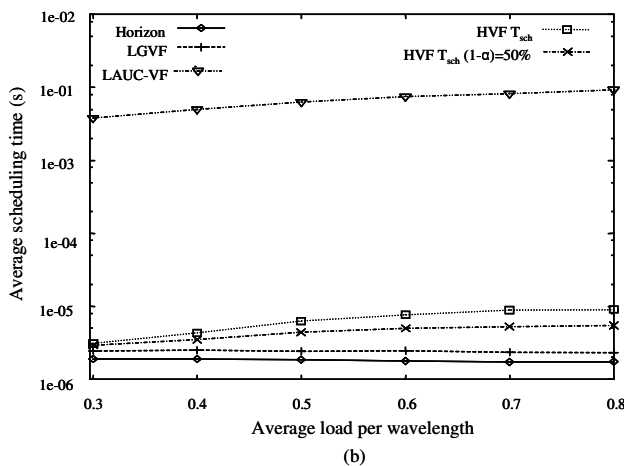
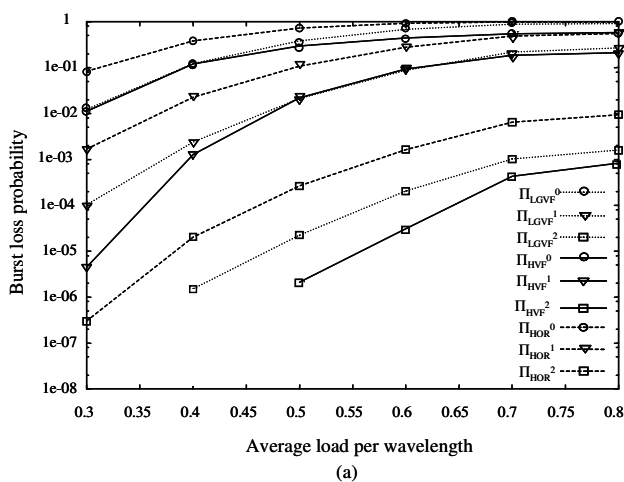


**Figure 9. Average scheduling time in seconds as a function of the load per wavelength and different search interval widths (a,b,c,d).**

### 4.3 HVF compared to others solutions

Last results make the comparison of HVF with other algorithms already presented in literature and mentioned in section 2. All these algorithms have been tested on the

node configuration taken in consideration in this study. In figure 10 a, burst loss probability of HVF compared with Horizon and LGVF algorithms is shown. For a matter of clarity LAUC has not been included in the figure and however it is outperformed by LGVF. HVF actually outperforms the others for each class in the range of the load value considered for the evaluation. In figure 10 b the scheduling time  $T_{sch}$  averaged over all classes is plotted for different algorithms confirming the good achievement of HVF in terms of trade off between processing time and burst loss performance. As it can be seen HVF gets closer to the fastest scheduling algorithms LGVF and Horizon (especially HVF with  $(1-\alpha) = 50\%$ ) but performing much better in terms of burst loss probability.



**Figure 10. Burst loss probability for HVF, LGVF and Horizon algorithms as function of the average load per wavelength (a) and scheduling time in seconds for HVF, HVF  $(1-\alpha)=50$ , LAUC-VF, LGVF and Horizon algorithms (b).**

## 5. Conclusions

In this paper the problem of channel reservation in an OBS node has been investigated. A new implementation of the void filling scheduling algorithm, HVF, that tries to optimize performance in terms of both burst loss probability and scheduling time has been presented. To achieve the target a binary heap implemented by means of an array data structure is exploited to store the information related to void intervals created in the presence of QoS differentiation. Several numerical results presented in the paper evidence the improvement obtained with respect to other solutions. In particular the proposed implementation allows void filling to be performed within the temporal target of typical offset values.

## Acknowledgements

This work is partially funded by the Italian Ministry of Education and University (MIUR) under the projects "INTREPIDO - End-to-end Traffic Engineering and Protection for IP over DWDM Optical Networks". The authors wish to thank both Prof. Vincenzo Eramo and Prof. Maurizio Casoni for their initial inputs, which greatly contributed to the development of this study and Mr. Donato Trisciuzzi for his help in the development of the simulation program.

## References

- [1] C. Qiao, M. Yoo, "Choices Features and Issues in Optical Burst Switching", *Optical Networks Magazine*, 1(2), pp. 36-44, May 2000.
- [2] J. Turner, "Terabit Burst Switching", *Journal of High Speed Networks*, Vol. 8, No. 1, pp. 3-16, January 1999.
- [3] K. Sriram, D.W. Griffith, SuKyoung Lee, N.T. Golmie, "Optical Burst Switching: Benefits and Challenge", *Proceedings of the First International Workshop on Optical Burst Switching (WOBS 2003)*, Dallas/TX, October 2003
- [4] J. Xu, H. Perros, G. Rouskas, "Techniques for Optical Packet Switching and Optical Burst Switching", *IEEE Communication Magazine*, Vol. 39, no. 1, pp. 136-142, January 2001.
- [5] L. Tancevski, A. Ge, G. Castanon, L. Tamil, "A new scheduling algorithm for asynchronous variable length IP traffic", *Proceedings of Optical Fiber Communication (OFC'99) Conference*, San Diego CA, February 1999.
- [6] M. Yoo, C. Qiao, S. Dixit, "Optical Burst Switching for service differentiation in the next generation optical internet", *IEEE Communication Magazine*, pp. 98-104, February 2001.
- [7] Y. Xiong, M. Vandenhouste, H. C. Cankaya, "Control architecture in optical burst-switched wdm networks," *IEEE Journal on Selected Areas in Communications*, vol. 18, pp 1838-1851, 2000.
- [8] Myungsik Yoo, Chunming Qiao, "Just-Enough-Time (JET): a high speed protocol for bursty traffic in optical networks", *Digest of the IEEE/LEOS Summer Topical Meetings*, pp. 26 - 27, 11-15 Aug. 1997.

- [9] J. Teng, G.N. Rouskas, "A comparison of the JIT, JET, and Horizon wavelength reservation schemes on a single OBS node", Proceedings of the First International Workshop on Optical Burst Switching (WOBS 2003), Dallas/TX, October 2003 .
- [10] T.H. Cormen, C.E. Leiserson, and R.L.Rivest, "Introduction to algorithms," McGraw-Hill, MIT Press, 1990.
- [11] E. McCreight, "Priority search trees," SIAM J. Computing, vol. 14, No.2, pp. 257–276, 1985.
- [12] J. Xu, C. Qiao, J. Li, G. Xu, "Efficient channel scheduling algorithm in Optical Burst Switched networks", Proc. of INFOCOM 2003 - 22<sup>nd</sup> Annual Joint Conference of the IEEE Computer and Communications Societies, Vol.3, pp. 2268 – 2278, 30 March-3 April 2003.
- [13] K. Dolzer, C. Gauger, J. Spath and S. Bodamer, "Evaluation of reservation mechanisms for optical burst switching", AEU Int. J. of Electron. and Commun., vol. 55, no. 1, 2001.
- [14] K. Long, R. S. Tucker and Se-yoon Oh, "Fairness scheduling algorithms for supporting QoS in optical burst switching networks", Proceedings of the SPIE's International Symposium: Aia-Pacific Optical and Wireless Communications (APOC), Shanghai, CHINA, 2002.
- [15] M.Yoo, C. Qiao, "A new optical burst switching protocol for supporting quality of service", in SPIE Proceedings, All Optical Networking: Architecture, Control and Management Issue, vol.3531, pp. 396-405, Nov. 1998.
- [16] M.L. Fredman, R.E. Tarjan, "Fibonacci Heaps And Their Uses In Improved Network Optimization Algorithms", Foundations of Computer Science, 25th Annual Symposium on, pp. 338-3461, October 24-26, 1984.