Fill the Void: Improved Scheduling for Optical Switching

Kurt Van Hautegem, Wouter Rogiest and Herwig Bruneel
SMACS Research Group
Department of Telecommunications and Information Processing (TELIN); Ghent University
St.-Pietersnieuwstraat 41; B-9000 Ghent, Belgium
Email: {kurt.vanhautegem ✉, wouter.rogiest, hb}@telin.ugent.be

***Abstract*—With ever-increasing demand for bandwidth, optical packet/burst switching is proposed to utilize more of the available capacity of optical networks in the future. In these packet-based switching techniques, packet contention on a single wavelength is resolved effectively by means of Fiber Delay Lines. The involved scheduling algorithms are typically designed to minimize packet loss and/or packet delay. By filling so-called voids, void-filling algorithms are known to outperform their non-void-filling counterparts. This however comes at a large computational cost as the void-filling algorithms have to keep track of beginnings and endings of all voids. This is opposed to the non-void-filling algorithms which only have to keep track of a single system state variable. We therefore propose a new type of algorithm that selectively creates voids that are larger than strictly needed, only when these will likely be filled. Results obtained by Monte Carlo simulation show that selective void creation can jointly reduce packet loss by 50% and packet delay by 18%, without imposing a high computational cost.**

I. Introduction

Growing trends in cloud computing and streaming media services are expected to increase the demand for bandwidth vastly. With dazzling bandwidths of up to 43 Terabit/s with a single laser [1], optical fiber seems the answer to all of our craving for data. In optical networks, however, capacity is not limited by the connections (links) but by the intersections (nodes). Currently circuit switching is used to establish a dedicated communication channel between two communicating nodes. This guarantees packet arrival but also reduces the available capacity due to inflexibility.

Promising solutions to address the rising demand in bandwidth are the packet-based switching techniques optical packet/burst switching (OPS/OBS) in which network links can be shared among communication sessions, thereby increasing the usage of the available fiber capacity. Although there has been a lot of criticism concerning the feasibility of OPS/OBS [2], the technology remains a future-proof alternative to slow and power-consuming electronic switching in the backbone [3]. In packet-based switching contention may arise in the network nodes when more than one packet heads for the same output port at the same time. As a solution temporary buffering is currently implemented with Fiber Delay Lines (FDLs) [4] in which the optical signals are sent through long pieces of coiled fiber to delay them for a certain time. As the number of FDLs is strictly limited, it is necessary to schedule arriving packets as efficiently as possible in order to reduce the probability of unresolved contention, and thus packet loss probability (LP).

In [5], [6] we proposed new cost-based scheduling algorithms, both in a related setting with wavelength converters present. There, we were able to validate the usefulness of a cost-based approach and achieved an increased performance (decreased LP). Although these algorithms can improve the performance significantly, their structure is very similar to existing algorithms [7]. Indeed, they can also be split up in two big categories: void-filling and non-void-filling algorithms. In contrast to the latter, the former allow packets to be scheduled before already scheduled packets, filling the so-called voids (unscheduled periods between already scheduled packets), thereby improving the performance (in terms of LP) significantly. As these void-filling algorithms keep track of all voids, also those that are not likely to be filled, this performance improvement comes at the cost of an increased computational complexity. As no trade-off between performance and computational complexity is possible in these algorithms, this is especially an issue in a typical setting with small FDLs (to minimize the packet delay and/or the footprint of the buffer) in which voids are unlikely to be filled or even unfillable. We therefore propose a new type of algorithm that selectively creates larger voids only when they will likely be filled in the future. This type of algorithm does not only fill the available voids, it also, based on the system conditions, controls the creation of the voids, making it more powerful, enabling better switch performance.

II. Background

A. Assumptions

Throughout the paper a continuous-time setting is supposed. Fig. 1 shows the assumed $K \times M$ optical switch configuration. Packets arrive on a finite number of incoming ports K, on c different *wavelengths* $\lambda_1, \ldots, \lambda_c$, also called *channels*. Each packet arrives on a certain wavelength and is switched (still on this wavelength) to one of the M output ports according to the packet header destination information. Each output port thus accepts packets from K ports, on c wavelengths. The output port is connected to a single fiber with the same c different wavelengths.

In this paper an arbitrary single output port is analyzed, marked by the dashed-line box in Fig. 1. We assume no wavelength converters are present at the output port. While other wavelengths may be used for packet switching within the same switch, they operate independently of each other and all packets are processed on the same wavelength upon

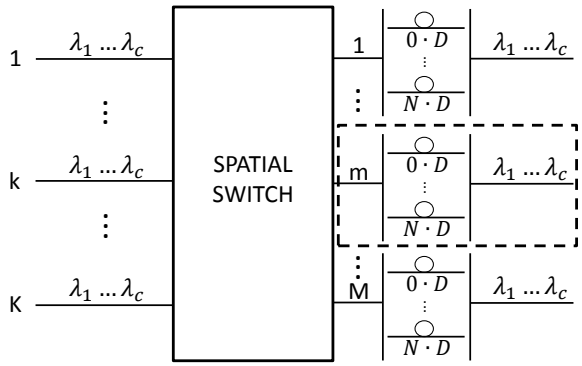


Fig. 1: The modelled output port (in dashed-line box) as part of a $K \times M$ optical switch.

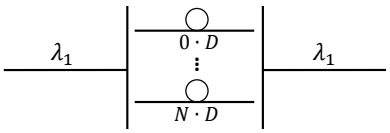


Fig. 2: The modelled output port as analyzed: with FDLs and a single wavelength λ_1.

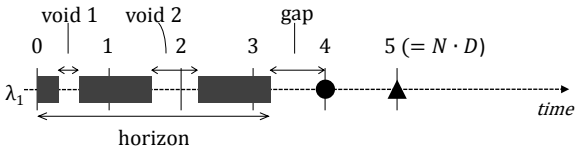


Fig. 3: An example of a provisional schedule.

which they arrive. We can thus confine our analysis of the output port to a single wavelength λ_1 (Fig. 2). Each output port has an optical buffer in which $N+1$ FDLs are available to schedule incoming packets. The lengths of the FDLs are consecutive multiples of a basic value D called the granularity. This is called a degenerate delay buffer [8], in which incoming packets sent through the j-th $(j = 0 \dots N)$ delay line encounter a delay of $j \cdot D$.

We assume packets arrive at the output port on wavelength λ_1 according to a Poisson process, with exponentially distributed inter-arrival times T and average $E[T]$. The length of the arriving packets is assumed a fixed length B equal to the granularity, $B = D$. While matching the granularity and the packet size is a natural choice in view of performance (see [9]), it also enables to devise an intuitive void-creating scheduling algorithm, as argued below. Algorithms with a similar scheduling strategy for other values of D of course can be thought of, but are considered out of the scope of this paper.

The overall incoming traffic load at the output port is fixed and given by $\rho = B/E[T]$. Further, the nature of the Poisson arrival process implies possible overlap of distinct packets at the entrance of the output port. This overlap causes contention, which has to be resolved before the packets exit the output port, also on the single wavelength λ_1.

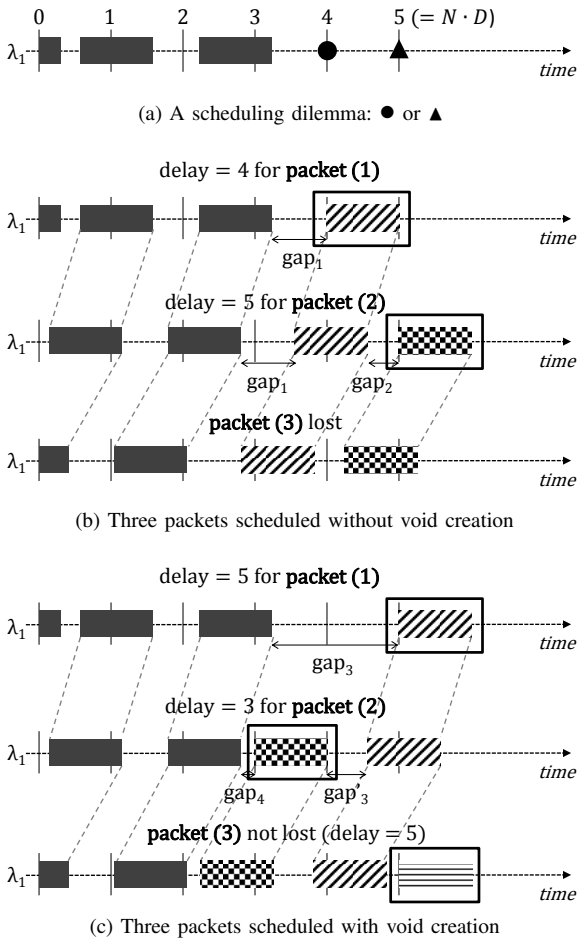
This contention is resolved by sending one of the contenting packets through one of the FDLs, if available. As the number of FDLs is always limited, a contenting packet is lost if all FDLs are occupied upon its arrival. It is therefore necessary to schedule packets as wisely as possible in the FDLs. In general, LP is the main performance measure for which the scheduling is optimized. Besides LP, we will also consider the packet delay as a second performance measure.

B. Scheduling basics

At the output port under study, scheduling is done separately for each packet upon its arrival. In this setting this amounts to assigning a single variable (j) to the packet, corresponding to the delay line $(j = 0 \dots N)$ the packet is scheduled on. This is done by means of a provisional schedule, of which an example is given in Fig. 3, showing the already scheduled packets (grey boxes) upon arrival of an arbitrary packet that is yet to be scheduled (and is thus not displayed on the provisional schedule). The arrival instant of a packet corresponds to the zero delay reference line. The provisional schedule is represented horizontally and the vertical lines represent the delays of the FDLs $(j = 0 \dots N = 5)$. The granularity in the example is assumed as unity $(D = 1)$ to ease notation. In this representation, the provisional schedule evolution can be seen as a choppy (observed from arrival to arrival) but uniform (all packets move alike) movement of all packets to the left, with packets disappearing (because they are being transmitted) when crossing the zero delay reference line. The provisional schedule, although similar at first sight, should not be confused with a slotted arrival process. The inter-arrival times are distributed continuously and the vertical lines represent the delays of the FDLs and not some slot boundaries. A packet has to be scheduled on an FDL (● or ▲) without overlapping with any of the already present packets. If no such FDL is available, the packet is lost.

Scheduling is done according to a scheduling algorithm, of which most, as said, are designed purely to achieve minimal LP. Regardless of their exact design aim, existing scheduling algorithms can be split up in two main categories:

- *void-filling algorithms* allow packets to be scheduled on any suitable FDL. This implies the possibility of filling up *voids*, defined as unscheduled periods followed by one or more packets scheduled beyond it. Here, only voids overlapping an FDL more than a packet length to the right of this FDL on the provisional schedule are fillable.
- *non-void-filling algorithms* only allow packets to join at the back and voids can not be filled. In this way the algorithm is not obliged to keep track of all voids but merely of the *horizon*, defined as the latest time at which the output is currently scheduled to be in use. Graphically, this corresponds to the right edge of the rightmost packet. Related the *gap* is defined as the length of the void in front of a packet when it is assigned to an FDL. For the first FDL to the right of



(a) A scheduling dilemma: ● or ▲

(b) Three packets scheduled without void creation

(c) Three packets scheduled with void creation

Fig. 4: Evolution of the provisional schedule for the considered example of three arriving packets.

the horizon this becomes:

$$gap = \left\lceil \frac{horizon}{D} \right\rceil \cdot D - horizon.$$

Where $\lceil x \rceil$ is the so-called ceil of x, the smallest integer greater than or equal to x. For each FDL further to the right the gap increases with a value of D.

As $D = B$ in the example of Fig. 3 no voids equal to or larger than the packet length (*fillable voids*) are created and thus none can be filled. Only voids smaller than the packet length (*unfillable voids*) are created. In this setting of a single wavelength, and $D = B$, all void-filling and non-void-filling algorithms schedule the packet on the first FDL to the right of the horizon. This is unless the horizon is larger than $N \cdot D$, in which case the packet is lost. Performance (in terms of LP and packet delay) will thus be the same for all void-filling and non-void-filling algorithms in this setting. Needless to say that the implemented algorithm is thus always non-void-filling, as it only keeps track of the horizon.

III. VOID-CREATING SCHEDULING ALGORITHM

In order to improve the performance of this system, fillable voids are created by scheduling the packet on the second FDL (triangle in Fig. 3) instead of the first FDL (dot in Fig. 3) to the right of the horizon. As $D = B$ this creates voids larger than B (fillable voids) which, if favorably positioned with respect to an FDL, can be filled. A favorable position occurs when the void overlaps the FDL at least a packet length to the right of the FDL, i.e. the (fillable) void is *reachable*. Analogously, a (fillable) void that is not positioned favorably with respect to an FDL is called *unreachable*. Note that an unfillable void can never be reachable. When a fillable void created by a first packet can be filled by any subsequent packet the average gap as well as the average delay encountered per packet will typically be lower than in the case when a fillable void was never created.

This is illustrated by means of an example in Fig. 4 in which the same scenario of three subsequent packet arrivals is analyzed. Fig. 4 compares the evolution of the provisional schedules from arrival to arrival without (Fig. 4b) and with (Fig. 4c) void creation. Packet (1) (diagonally hatched) arrives at $time = 0$ (reference time). Packet (2) (checkerboarded) and packet (3) (horizontally hatched) arrive a time of 0.44 and 1.19 respectively hereafter (assuming $D = B = 1$). These arrival instants again coincide with the zero reference time in the matching provisional schedule (second provisional schedule for packet (2) and third provisional schedule for packet (3) in both Fig. 4b and Fig. 4c), as this is how the provisional schedule works.

Without void creation, Fig. 4b: In Fig. 4b both packet (1) and (2) are scheduled on the first FDL to the right of the horizon of their specific provisional schedule, corresponding to the strategy used by the existing algorithms (both void-filling and non-void-filling) in this setting. This results in an average delay of 4.5 for packets (1) and (2) (delay of 4 for packet (1) and 5 for packet (2)). The total gap assigned to packets (1) and (2) equals $gap_1 + gap_2$. Moreover at the arrival instant of packet (3) (third provisional schedule in Fig. 4b) there is no FDL available to schedule packet (3), which thus results in a lost packet.

With void creation, Fig. 4c: In Fig. 4c packet (1) is scheduled on the second FDL to the right of the horizon, in this way creating a fillable void. Upon its arrival packet (2) is able to fill this fillable void as it is reachable (it overlaps the FDL that corresponds with a delay of 3 in a favorable way). The scenario with void creation has two clear benefits:

- Lower delays: the average delay for packets (1) and (2) equals 4 (delay of 5 for packet (1) and 3 for packet (2)) as opposed to an average delay of 4.5 for packets (1) and (2) without void creation.
- Reduced Loss: the total gap assigned to packets (1) and (2) equals $gap'_3 + gap_4$ which equals gap_1 and thus is always smaller than in the case without void creation: $(gap'_3 + gap_4 = gap_3 - D = gap_1) < (gap_1 + gap_2)$. Because of this, the stacking of packets (1) and (2) is more dense in Fig. 4c and the last FDL is available to schedule packet (3) when it arrives. In this example no packet is thus lost when void creation is used.

It is clear from the above example that filling a fillable void lowers the average delay of the packets and the average gap size. Moreover the example shows that reducing the average gap size mitigates the number of packets lost as it makes the stacking of the packets dense and reduces the chance the horizon exceeds $N \cdot D$, the only case in which a packet that arrives is lost. On the other hand creating but not filling voids is disadvantageous for the performance as the stacking becomes less dense. The chance the horizon exceeds $N \cdot D$ (and, likewise, the average packet delay) increases. Whether a void is reachable, depends on the choppy evolution of the provisional schedule and thus on the stochastic arrival process. We can however maximize the chance that a void is reachable by creating only voids that are likely to be filled. As a first condition, we only allow a single fillable void to be present in the system. As soon as the ending of the fillable void is lower than $B = D$ it is smaller than the packet length $B = D$. It will never be reachable again in the future and the fillable void *expires*. An *expired void* is thus a void that was fillable but is now unfillable. Because it is unfillable, it is omitted and a new fillable void is allowed to be created. A fillable void can of course also disappear because it is reachable the moment a new packet arrives. This packet will then fill this void, and a next arrival may be used to create a new fillable void. Besides allowing only a single fillable void in the provisional schedule we demand for two additional conditions to be met before creating a fillable void:

- A first condition relates to the size of the created fillable voids. As the void-creating algorithm chooses the second FDL to the right of the horizon, the created fillable voids will have a length between D and $2 \cdot D$. A void only slightly larger than D will have a small chance of being reachable. To fill a void, it indeed has to overlap an FDL at least a length of $D = B$ to the right of the FDL on a set of future, yet unknown, arrival instances. For a void almost equal to $2 \cdot D$ it is very likely to have a long period of reachability, although it is still possible an unfavorable arrival pattern occurs. The first condition therefore states that the created void has to be larger than a certain value $D + y_n \cdot D$, i.e. $D + y_n \cdot D < fillable\ void$. Here y_n is called the *gap threshold*, an algorithm parameter varied in simulations ($0 < y_n < 1$ in steps of 0.01) with n the horizon index (see next).
- A second condition to create a fillable void states that the horizon has to be between two specific and consecutive FDLs: $(n-1) \cdot D < horizon < n \cdot D$ in which n is called the *horizon index* and is varied in simulations $(n = 1 \ldots N-1)$. With this condition we want to investigate the effect of the position of void creation. Fillable voids created when n is closer to $N-1$ will stay longer in the system and thus given the Poisson arrival process, have a higher chance of being filled. This condition is used to create auxiliary results that allow us to obtain the optimal gap thresholds of the actual algorithm in which void creation is allowed for all horizon indexes (see next section).

If one of these conditions is not met or another fillable void is already present in the provisional schedule, the packet is used to fill this void or otherwise is scheduled on the first FDL to the right of the horizon, creating an unfillable void.

Similar to regular void-filling algorithms, this void-creating algorithm allows packets to be scheduled out of order. The consequences of this for the upper network layers however is out of the scope of this paper.

IV. SIMULATION: ITERATIVE APPROACH

To evaluate the performance (LP and packet delay) of the void-creating algorithm proposed in Sect. III, we employ Monte Carlo simulation. Specifically, the algorithm is programmed in Matlab using a discrete event simulation (DES) in a similar way as in [5], [6]. In a DES, the system is modelled as a sequence of events marked by their particular instant in time, i.e. the simulation is event-based. The system state changes from one event to the next and does not change in-between events. This is as opposed to continuous simulation in which time is broken into small pieces called *time slices*. At each ending of a time slice the system state is (possibly) changed based on the events happened in the last time slice. Because DES simulations do not simulate every time slice, they are far more efficient in terms of computational resources.

For the void-creating algorithms the system state includes the value of the horizon and the beginning and ending of the fillable void (if present). This is opposed to the non-void-filling algorithm, that only has to keep track of the horizon. As the regular void-filling algorithm (i.e. without creating voids) has the same behavior as the non-void-filling algorithm in this setting of $D = B$, it is not simulated separately. Note that allowing more than one fillable void in the system (in a smart way) could result in even better performance. This however is outside the scope of this paper and part of future work.

To obtain the LP and packet delay of the void-creating algorithm, the following method is used. A first and auxiliary set of simulations only allows to create a void when both the conditions are met, i.e. $D + y_n \cdot D < fillable\ void$ and $(n-1) \cdot D < horizon < n \cdot D$. By varying y_n ($0 < y_n < 1$ in steps of 0.01) for a fixed n we determine the optimal gap threshold when voids are only allowed to be created on horizon index n. This is done separately for each value of n $(n = 1 \ldots N-1)$ and gives us two sets (one for LP and one for packet delay) of both $N-1$ optimal gap threshold values for which either LP or packet delay is minimized. These are called the *single creation point thresholds*.

The actual void-creating algorithm allows voids to be created on all horizon indexes but still optimizes the gap threshold y_n for each horizon index: i.e. depending on the position of the horizon the condition on the gap size can either be more or less strict. Note that the restriction of only one fillable void is maintained. Allowing void creation for each horizon index results in a very large parameter space for optimization. With $0 < y_n < 1$ in steps of 0.01 and $N-1$ different horizon indexes the parameter space contains

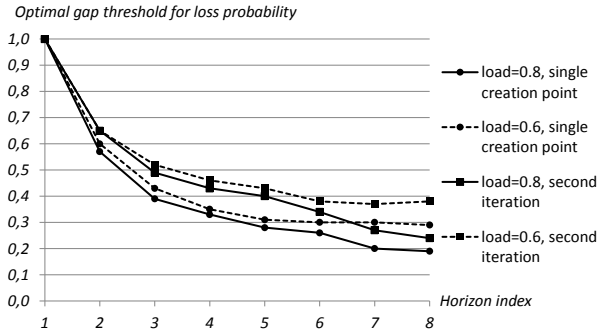


Fig. 5: Gap threshold optimized for loss probability.

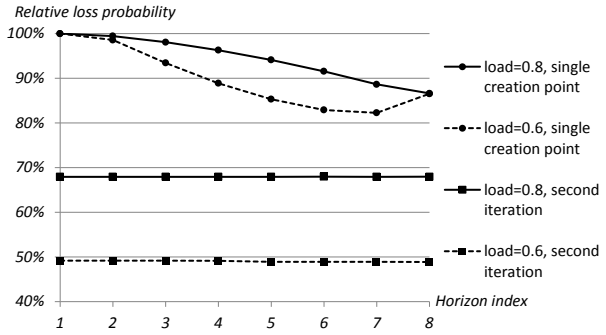


Fig. 6: The optimized relative loss probability.

100^{N-1} combinations. Even for a small value of N this results in an unreasonable long simulation time.

We therefore use the following iterative approach to approximate the behavior of the actual void-creating algorithm. In a first iteration the fillable void (only one allowed) can be created on all horizon indexes, with all gap thresholds but one fixed to the optimal single creation point thresholds (auxiliary set of simulations). The one gap threshold that is not fixed is varied and optimized ($0 < y_n < 1$ in steps of 0.01). Consecutively this is done once for each horizon index $(n = 1 \dots N-1)$ in a different simulation trace. In this way the gap threshold of each horizon index is optimized, with the other thresholds fixed. This gives us a set of $N-1$ new optimal gap thresholds which are called the *first iteration thresholds*. In a second iteration we fix all gap thresholds but one to these first iteration thresholds and again optimize the one that is not fixed. Again only the gap threshold that is optimized is varied $(n = 1 \dots N-1)$. This results in a set of $N-1$ *second iteration thresholds*. These iterations can be repeated as many times as desired until an acceptable convergence is noticed in the iteration thresholds and their corresponding performance, marking a good approximation of the actual void-creating algorithm. This iterative approach is used for both the LP and delay thresholds.

V. PERFORMANCE RESULTS

To compare the implementation complexity of the algorithms we simulate the arrival of $10 \cdot 10^4$ packets using stripped-down versions of the algorithms, i.e. without keeping track of any performance measures as LP or packet delay. The algorithms thus only decide on which FDL each packet is scheduled, just like a switch-level implementation would.

TABLE I: Comparison of simulation times of different algorithms under equal conditions.

Type of algorithm	Average simulation time for 10^4 arrivals (s)
Non-void-filling	0.1921
Void-creating	0.2974
Void-filling	3.8644

All simulations were carried out on the same PC (Intel Core i7, CPU @ 2.40 GHz) under the same circumstances and with all algorithm parameters equal (i.e. the parameters used for Fig. 5-6, load=0.8, second iteration). Table I clearly shows that keeping track of all the voids in the void-filling algorithm is, apart from useless (since no void can be filled), also very costly in terms of computation. The void-creating algorithm on the other hand is nearly as fast as the non-void-filling algorithm as it only keeps track of one extra void besides the horizon.

As in the given setting of $D = B$ the void-filling algorithms perform exactly the same as the non-void-filling algorithm, we only use the latter as a comparison for the performance of the void-creating algorithm. We thus first simulate the system without void creation, i.e. all packets are scheduled on the first FDL to the right of the horizon. If the horizon is larger than $N \cdot D$ on arrival of a packet, it is lost. The number of FDLs was assumed ten ($N + 1 = 10$) and $D = B = 1$. For a fixed load ρ of $80\,\%$ this gives an LP of $14.46\,\%$ of the arriving packets. When the load is fixed to $60\,\%$ the LP is $2.08\,\%$. The average packet delay under this algorithm is 6.69 time units for a load of $80\,\%$ and 2.98 time units for a load of $60\,\%$. The number of simulated packets is $10 \cdot 10^6$ for a load of $80\,\%$ and $10 \cdot 10^7$ for a load of $60\,\%$. For all simulations this yields confidence intervals too narrow to be displayed on the figures.

A. Optimized for loss probability

The results of the auxiliary set of simulations, i.e. the single creation point thresholds and their corresponding performance, are shown in Fig. 5 and 6. The curves marked as 'single creation point' in Fig. 5 show the optimal gap thresholds for the different horizon indexes, when void creation is only allowed for that index and the LP is minimized. For example when a fillable void is allowed to be created if (and only if) $D < horizon < 2 \cdot D$, then the void has to be larger than $1.6 \cdot D$ ($y > 0.6$) for a load of $60\,\%$ to achieve the largest reduction in LP. The LPs with the threshold values of Fig. 5 are shown in Fig. 6. The LPs are shown relative to the LPs when no void creation is allowed, i.e. $2.08\,\%$ and $14.46\,\%$ for a load of $60\,\%$ and $80\,\%$ respectively. The graphs marked as 'second iteration' on Fig 5 and 6 show the second iteration thresholds and the corresponding relative LPs obtained in the second iteration when the LP is optimized. Again the LPs are shown relative to the LPs when no void creation is allowed.

Looking at Fig. 5 we can see that for equal horizon index, the single creation point threshold is larger, and thus stricter, for the lower load. This is because for lower load the number of expected packets to arrive, before the void disappears by crossing the zero delay line, is lower. This lack of high arrival

density thus has to be compensated by a larger size of the created void. When the fillable void is only allowed to be created when $n = 1$ (i.e. the horizon is between 0 and D on the provisional schedule), no improvement in LP is achieved, and this for both loads. This is indicated by a single creation point threshold of 1 for $n = 1$, as this never occurs. For both loads the single creation point threshold decreases as the horizon index increases. As the horizon index on which fillable voids are allowed to be created increases, the created void stays longer in the system and thus given the Poisson arrival process, has a higher chance of being filled. For equal chance of being filled, smaller voids may be created as the horizon index increases.

Still on Fig. 5 we can see that for the second iteration the optimal gap thresholds are higher, and thus stricter, than for the corresponding 'single creation point' graphs. As void creation is also allowed for other positions of the horizon, a stricter policy can indeed be applied. In general these second iteration thresholds also decrease with increasing horizon index (for the same reason as above). For a horizon index $n = 8$ and a load of 60 % however an inversion is spotted in the second iteration threshold. This inversion possibly relates to the inversion found in the LP reduction for a single creation point and a load of 60 % (discussed next).

Fig. 6 shows that for a single creation point the lower load allows for a bigger relative reduction in LP. For a load of 80 % the optimal LP monotonically decreases with an increasing horizon index. Opposed to this, for a load of 60 % less improvement in LP is possible when fillable voids are created when $n = 8$ than when $n = 7$. A possible explanation is that when the load is 'low' (60 %), the probability of finding a horizon index $n = 8$ is smaller than for higher load (80 %), thus reducing the number of instances fillable voids can be created and the LP is reduced. The load for which this inversion is observed may be linked to the critical load of 69.3 % for which the infinite system without void creation becomes unstable ($\rho = ln(2)$, see [9]), i.e. for load values higher than this critical load the number of packets in the system with an infinite numbers of FDLs grows unboundedly.

As the 'second iteration' curves in Fig. 6 illustrate, a lower load also allows for a bigger reduction in LP when voids are allowed to be created on all horizon indexes. For a load of 60 % there is reduction in LP of more than 50 %. For a load of 80 % a reduction of about 32 % is achieved. This reduction is practically the same for all values of n as for all these simulation points void creation is allowed for all positions of the horizon, each with their distinct gap threshold.

B. Optimized for packet delay

Similarly to Fig. 5 the curves marked as 'single creation point' in Fig. 7 show the optimal gap thresholds for the different horizon indexes, when void creation is only allowed for that index and the packet delay is minimized. Fig. 8 shows the corresponding packet delays relative to the delay when no void creation is allowed, i.e. 2.98 and 6.69 for a load of 60 % and 80 % respectively.

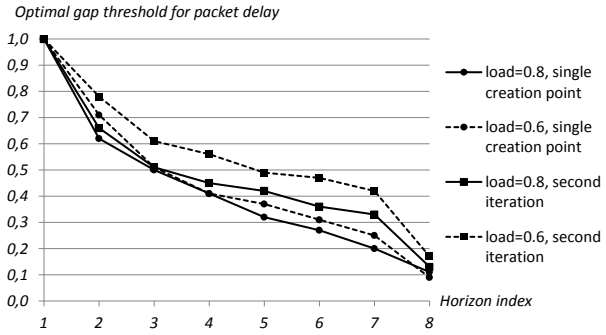


Fig. 7: Gap threshold optimized for packet delay.

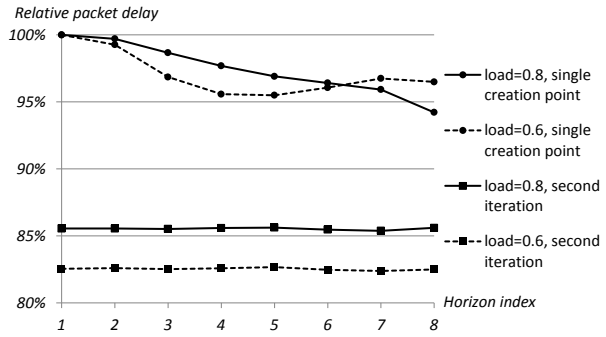


Fig. 8: The optimized relative packet delay.

Comparing Fig. 5 and Fig. 7 it is clear that the optimal thresholds are impacted by the optimization criterion (LP or delay), for both single creation point and second iteration. Despite this, similarities can be seen: the optimal threshold decreases with both increasing horizon index and increasing load. Also for a horizon index $n = 1$, no improvement in packet delay is possible by creating a void. This again is indicated by a single creation point threshold of 1 for $n = 1$. The inversion spotted for the second iteration thresholds for a load of 60 % when $n = 8$ no longer occurs when optimizing for packet delay, as all optimal thresholds decrease with increasing horizon.

Looking at Fig. 8 we see that the achievable improvements in packet delay are smaller than the corresponding achievable improvements in LP in Fig. 6. Similar to Fig. 6, for a load of 80 % the optimal delay monotonically decreases with an increasing horizon index. The inversion spotted for a load of 60 % is spotted for a lower horizon index than in Fig. 6, i.e. $n = 6$ in Fig. 8 and $n = 8$ in Fig. 6. For $n = 8$ in Fig. 8 the inversion stops, and the achievable delay for $n = 8$ is smaller again than the achievable delay for $n = 7$. Despite this inversion stop, for both $n = 7$ and $n = 8$ the optimal delay is higher for a load of 60 % than for a load of 80 %. Again this inversion might be linked to the critical load of 69.3 % of the infinite system. When void creation is allowed for all horizon indexes a lower load still allows for a bigger reduction in packet delay. For a load of 60 % there is an achievable reduction in delay of about 18 %. For a load of 80 % the achievable reduction is limited to 14 %.

VI. Conclusions

In this paper we proposed a new type of algorithm called void-creating algorithm which, based on the current system state, decides to create a fillable void or not. Using Monte Carlo simulation, this algorithm was validated for a single-wavelength, fixed packet length and fixed load setting. It was shown that by selectively creating larger voids than strictly necessary, this algorithm can achieve performance improvements in both loss probability and packet delay. Improvements in loss probability of up to $30\,\%$ for a high load ($80\,\%$) and $50\,\%$ for a lower load ($60\,\%$) are achievable. The achievable improvements for packet delay are smaller with an achievable reduction of $14\,\%$ for a load of $80\,\%$ and $18\,\%$ for a load of $60\,\%$. The optimal algorithm parameter values for which these improvements in loss probability and packet delay are achieved, though not identical, are in close proximity and can be determined by an iterative approach.

The improvements in these performance measures are achieved without a large increase in computational complexity, as in our implementation only one void is part of the system state. As opposed to existing algorithms, in this algorithm a simple trade-off between performance and computational complexity could be possible by keeping track of more voids. These results open opportunities to analyze a generalization of this approach with more fillable voids and in more complex settings with, e.g., variable packet length and other values of the granularity. Besides this, analyzing the optimal algorithm parameters in a more mathematical way can provide more insight in the void creating mechanism.

Acknowledgment

Part of this research has been funded by the Interuniversity Attraction Poles Programme initiated by the Belgian Science Policy Office. The second author is Postdoctoral Fellow with the Research Foundation Flanders (FWO-Vlaanderen).

References

[1] "New world record in data transfer," 2014. [Online]. Available: http://www.technologist.eu/new-world-record-in-data-transfer/

[2] R. Tucker, "Scalability and energy consumption of optical and electronic packet switching," *Journal of Lightwave Technology*, vol. 29, no. 16, pp. 2410–2421, Aug 2011.

[3] H.-L. To, S.-H. Lee, and W.-J. Hwang, "A burst loss probability model with impatient customer feature for optical burst switching networks," *International Journal of Communication Systems*, 2014.

[4] E. Burmeister, D. Blumenthal, and J. Bowers, "A comparison of optical buffering technologies," *Optical Switching and Networking*, vol. 5, no. 1, pp. 10 – 18, 2008.

[5] K. Van Hautegem, W. Rogiest, and H. Bruneel, "OPS/OBS scheduling algorithms: Incorporating a wavelength conversion cost in the performance analysis," in *Proceedings of the 32nd IEEE International Performance, Computing, and Communication Conference (IPCCC)*, San Diego,California,USA, December 2013.

[6] ——, "Scheduling in optical switching: deploying shared wavelength converters more effectively," in *Proceedings of the 2014 IEEE International Conference on Communications (ICC)*, Sydney,Australia, june 2014.

[7] F. Callegati, W. Cerroni, and G. S. Pavani, "Key parameters for contention resolution in multi-fiber optical burst/packet switching nodes," in *Proceedings of Broadnets 07*, Raleigh, North Carolina, USA, September 2007.

[8] L. Tancevski, L. Tamil, and F. Callegati, "Nondegenerate buffers: an approach for building large optical memories," *IEEE Photonics Technology Letters*, vol. 11, pp. 1072–1074, Aug. 1999.

[9] W. Rogiest, J. Lambert, D. Fiems, B. V. Houdt, H. Bruneel, and C. Blondia, "A unified model for synchronous and asynchronous FDL buffers allowing closed-form solution," *Performance Evaluation*, vol. 66, no. 7, pp. 343 – 355, 2009.