A parallel iterative scheduler for asynchronous Optical Packet Switching networks

P. Pavon-Marino, M. Bueno-Delgado, W. Cerroni, A. Campi, F. Callegati

***Abstract*—This paper presents PI-OPS (Parallel-Iterative Optical Packet Scheduler) a parallel-iterative scheduler for asynchronous Optical Packet Switching nodes with optical buffering. Optical packets are assembled by aggregating IP packets, and attaching an optical packet header. Conventional schemes process optical packet headers one by one, in a sequential form. Then, worst case algorithm response time is tightly coupled to switch size. In contrast, in PI-OPS all the optical packets received during a given time window are jointly processed to optimize the delay and output wavelength allocation, applying void filling techniques. The scheduler has a deterministic response time, independent of the traffic arrivals pattern. In addition, PI-OPS has been specifically designed to allow a parallel electronic implementation similar to the ones found in VOQ schedulers. In this respect, we evaluate the traffic loss performance of the scheduler in different settings, to dimension a set of hardware related parameters. Finally, we conduct an emulation of an FPGA implementation of a large-scale version of the scheduler. Results support the feasibility of its implementation.**

***Index Terms*—Optical Packet Switching, scheduling algorithms, performance evaluation, FPGA prototyping.**

I. Introduction

Advances in optical communications are considered among the major drivers for the development of future ICT services. In particular, switching paradigms such as *Optical Packet Switching* (OPS) and *Optical Burst Switching* (OBS) are able to take advantage of both the huge capacity provided by the photonic technology and the flexibility attainable with a statistical multiplexing approach [1]. Therefore, they are considered attractive candidates for the deployment of next generation network services requiring dynamic provisioning of large amounts of bandwidth on short time scales [2]. Recent studies have also demonstrated the advantages of OBS and *asynchronous* OPS in terms of reduced CAPEX and power consumption respectively [3][4].

Both OPS and OBS paradigms are based on the use of shared transmission and switching resources, leading to typical packet contention events that cannot be solved using traditional buffering techniques, due to the lack of optical RAMs. In general, contentions could be solved by exploiting three dimensions [1]: the wavelength domain, by using wavelength conversion each time two or more packets need to be simultaneously transmitted on the same fiber; the time domain, by delaying the transmission of some of the contending packets through a finite set of optical delay lines; the space domain, by re-routing some of the contending packets to a different output along a different path to the destination. The combined use of time and wavelength domains proved to be quite effective for both paradigms in terms of logical node performance [5][6]. Several additional contention resolution techniques have been proposed in the literature, especially for the OBS case where significant research effort has been devoted to the burst scheduling problem [2].

However, despite the latest advances in integrated photonics [7] and the significant number of optical switching test-beds available [8], several technological challenges are still to be solved to achieve the feasible deployment of packet-based optical networks. One of these issues is posed by the need for specialized hardware implementing the node control functions. In fact, due to the extremely high packet arrival rates, each node is required to process packet headers in a very short time span to be

The work described in this paper was carried out with the support of the BONE project ("Building the Future Optical Network in Europe"); a Network of Excellence funded by the European Commission through the 7th ICT-Framework Program.

This research has been partially supported by the project CALM TEC2010-21405-C02-02 (subprogram TCM) funded by the Spanish Ministerio de Innovación y Ciencia and and it is also developed in the framework of "Programa de Ayudas a Grupos de Excelencia de la R. de Murcia, F. Séneca".

Pablo Pavon Marino, Maria Victoria Bueno Delgado are with Technical University of Cartagena, Plaza Hospital 1, 30202, Cartagena, Spain (phone: +34 968325952; fax: +34 968325973; e-mail: {pablo.pavon, mvictoria.bueno}@upct.

Walter Cerroni, Aldo Campi, Franco Callegati are with the University of Bologna, DEIS Department, Via Venezia 52, 47521 Cesena (FC), Italy (phone: +39 0547 339209; fax: +39 0547 339208; e-mail: {walter.cerroni, aldo.campi, franco.callegati}@unibo.it).

able to apply a contention resolution scheme, to correctly schedule data transmission and to compute the appropriate switching device configuration.

Therefore, a suitable packet scheduling algorithm must be defined and executed when processing headers in order to allocate a contention-free output channel and the required delay, if available. The mismatch between the asynchronous packet arrival process and the finite set of optical delays available may lead to the creation of a significant amount of unused time gaps (called *voids*) between consecutive scheduled packets, which could heavily affect the node performance [9]. As a consequence, optical packet scheduling represents a key optimization problem. Solutions based on a sequential approach, where headers are processed one by one, may become the bottleneck of the control mechanism. When the switch port count increases the worst case packet header processing time grows while it should be well upper bounded to cope with the potential of massively simultaneous arrivals targeted to the same output fiber. Parallel header processing may reduce the computation time of the scheduler and decouple the scheduler response time from the switch fabric size.

A parallel iterative scheduling algorithm for optical packet networks is introduced here, named PI-OPS (Parallel Iterative Optical Packet Scheduler), suitable for a fast electronic implementation with a constant guaranteed response time almost independent of the switch size. It is built on a multiple header scheduling concept similar to the one adopted by iSLIP-like algorithms designed for Virtual Output Queuing (VOQ) router architectures [10] and extends a similar solution previously devised for OBS networks [11]. The PI-OPS algorithm is targeted to asynchronous OPS networks, where optical packets can be of variable duration, and are *not* aligned at switch inputs. The packet header is supposed to precede the payload, separated by a short guard band. However, it is interesting to note that PI-OPS could be applied also, without modifications, to OBS networks operating with the *Only Destination Delay* (ODD) reservation mechanism [12]. In OBS/ODD networks, an offset time exists between the burst header and the payload, which is constant in all the bursts, and kept constant hop-to-hop.

The contributions of this paper are as follows. First, we propose the PI-OPS algorithm which is, at the best of our knowledge, the first parallel-iterative approach to schedulers suitable for asynchronous OPS networks. Then we propose a PI-OPS design to allow an electronic implementation based on parallel architectures. We propose a payload overlapping check method based on a fast processing of bit-mask occupation registers. The dimensioning of these registers allows us to precisely define the requirements for building a PI-OPS scheduler for large-scale OPS nodes. Finally, we have emulated the FPGA implementation of the scheduler. Our results prove its practical feasibility and how physical implementation issues may affect the logical node design and the related performance.

The rest of the paper is organized as follows. Section II reviews the related work. Section III describes the PI-OPS algorithm. The performance of the algorithm is comparatively evaluated in Section IV, and a set of hardware related parameters are dimensioned. Section V elaborates on the electronic FPGA implementation of PI-OPS. Finally, Section VI concludes the paper.

II. RELATED WORK

A significant amount of research has been devoted to the scheduling problem in asynchronous OPS and OBS networks and several algorithms have been defined and studied in the last decade with the goal to provide efficient resource utilization and limited packet loss. The first proposal that was trying to exploit the wavelength domain was a burst switching paradigm based on the *time horizon* concept, defined for each output channel as the instant after which no burst is scheduled to be transmitted [13]. Any channel with a time horizon smaller than the arrival time of an incoming burst is suitable to accommodate the burst itself. The algorithm must then select the best output channel according to an optimal criterion. An option is to choose the channel with the latest horizon, in order to transmit each burst as close as possible to the last one previously sent on the same channel and then minimize the bandwidth wasted due to voids.

This technique is known in OBS as *Latest Available Unscheduled Channel* (LAUC) and can also be applied to the case of nodes equipped with delay lines, i.e. when both wavelength and time domains are combined to solve contentions [14]. When no channel has a time horizon smaller than the burst arrival time, the LAUC policy is applied to the future delayed copies of the incoming burst, starting with the smallest delay. It has been shown that, in the asynchronous OPS case, channel utilization may be improved by choosing the channel with the time horizon introducing the smallest gap overall, instead of the one available with the smallest delay [5].

A major improvement in terms of performance was achieved by adding *void filling* capabilities to the optical packet scheduling policy [15]. Already scheduled channels are now included in the search, given that they are unused for a time interval (i.e. void) starting before the (possibly delayed) packet arrival time and large enough to accommodate the entire incoming packet. The unscheduled channels are considered as a particular case of voids with infinite length. The algorithm selects the suitable void with the latest starting time, minimizing the gap left in front of the incoming packet. In the OBS literature this technique is known as *Latest Available Unused Channel with Void Filling* (LAUC-VF) [14]. Several other scheduling policies were proposed in OPS and OBS literature, among which are techniques for QoS differentiation [16][17][18], packet sequence maintenance [19] and burst segmentation [20].

As already mentioned, the computational complexity of the scheduling algorithm is a critical aspect that affects the actual feasibility of the node control logic. Therefore, it is necessary to design an efficient and fast scheduling algorithm while keeping the throughput at an acceptable level. In particular, since void filling algorithms must maintain information on all the voids present in each output channels and perform extensive look-ups over such data set, the scheduler implementation may become

too complex and the header processing too time-consuming compared to the packet arrival rate. Proposals to significantly reduce the scheduler processing burden while achieving the same optimal loss ratios as LAUC-VF, thanks to the use of efficient data structures and smart search techniques, include *MinSV*, *MinEV*, *BestFit* [21] or *HVF* [22].

A hardware implementation of the OBS scheduler based on burst re-sequencing was recently proposed [23], which is able to achieve optimal scheduling in O(1) complexity. This scheduler does not process burst headers immediately as they arrive. Instead, it delays and reorders them according to the correct payload arrival time. Then, by applying a simple time horizon policy, it is possible to schedule bursts that would have required a void filling algorithm in sequential header processing. However, this scheduler is applicable to the bufferless OBS case only, since it is able to exploit only the voids created by different offset times and not by delays. Furthermore, due to the delayed header processing, data bursts experience additional latency.

In order to implement a scalable packet scheduler, parallel processing should be adopted, as suggested on a recent paper which re-formulates the problem towards a viable hardware implementation [24]. However, all the schemes mentioned above take their decisions based on single header processing, resulting in a greedy approach to the problem. On the contrary, the parallel iterative scheduling algorithm proposed in this paper works on a multiple header set. The approach is in principle similar to the one used by *OBS Group Scheduling* [25], although the algorithm proposed in the following sections is specifically designed for a parallel and feasible electronic implementation. As a result, we show that a constant algorithm response time below 10 μs can be obtained for large-scale switches.

III. PI-OPS ALGORITHM DESCRIPTION

A. General view and time constraints

Figure 1 shows the OPS optical node architecture, with N input and output fibers, and n data wavelengths per fiber ($\lambda_0...\lambda_{n-1}$). The optical switching fabric (OSF) transparently switches optical payloads from input ports to output ports. In this paper, we consider OSFs able to emulate output buffering, with full wavelength conversion, and D fiber delay lines (FDLs) of duration d=0,G,...,(D-1)G, where G denotes the FDL granularity. Output buffering emulation means that the OSF does not add any internal contention to the process of allocating an output wavelength, and a FDL to the arriving packets. For instance, the switching matrices proposed in [26] and [27] are able to emulate output buffering.

The electronic control unit processes the arriving packet headers, to make the appropriate scheduling decisions. In the architecture here proposed the scheduler works according to a slotted time frame, while the data plane operates asynchronously. In this section, we outline the timing relationships between the control and data plane that are necessary to guarantee safe operations. Let us assume that the payloads arrives $\delta+D_P$ microseconds after the corresponding headers, where:

- δ denotes the header offset, being $\delta=0$ in OPS networks (the guard band duration is assumed negligible), and $\delta>0$ in OBS networks applying the ODD reservation protocol;
- D_P denotes the duration of the fixed fiber delays at the switch fabric inputs (see Fig. 1).

The PI-OPS algorithm is designed as a parallel iterative algorithm, with a constant response time lasting T_A μs.

The algorithm is executed periodically, every T_I μs. [1] The algorithm execution starting at time $t=t_0$, is responsible for jointly processing the packet headers asynchronously received during the time interval $[t_0-T_I, t_0]$. We call this interval the *header arrival time window* of the algorithm execution.

At the end of a given algorithm execution, the scheduling decisions for all the headers belonging to the relative header arrival time window are available. Based on these decisions, the OSF has to be configured to guarantee that the payloads find a proper optical path from inlet to outlet. The OSF configuration will take an amount of time that may depend on the optical path and the number of related devices to configure. In this work, we assume it is upper bounded by T_{OSF} microseconds.

[1] The constraint $T_I \geq T_A$ ensures that an algorithm execution starts when the previous execution is already finished.

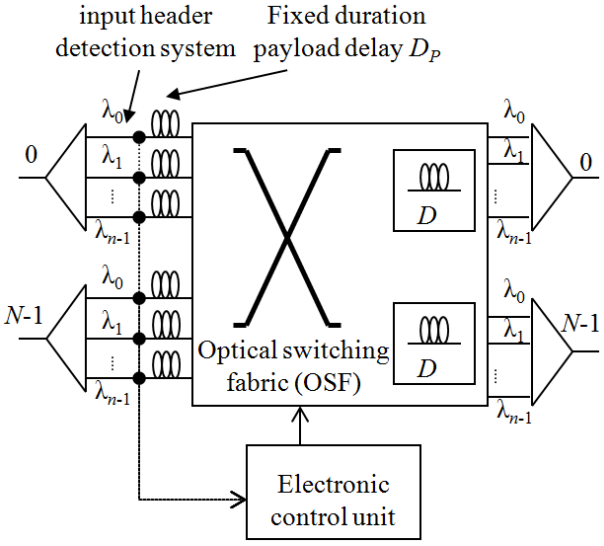

Fig. 1. OPS switching node scheme

Therefore for a header arriving at time $t=t_0$, in the worst case the switch will be configured after: (i) T_I μs because the header arrives just after the start of the header arrival time window (ii) T_A μs required for executing the algorithm, and (iii) T_{OSF} μs necessary to reconfigure the OSF. Therefore the following inequality has to be satisfied:

$$\delta + D_P \geq T_{WC} = T_I + T_A + T_{OSF} \qquad (1)$$

Note that T_{WC} represents a worst-case delay between the moment the packet header arrives to the switch and the moment an optical path is configured in the OSF. The non-negligible reconfiguration time of the optical components has one additional implication in the scheduling operation. During an OSF internal path reconfiguration time, the optical components involved can not be used. Therefore, if the idle time between two consecutive payload arrivals to the same input port was lower than T_{OSF}, the second packet signal would be destroyed by the optical components reconfiguration. As a conclusion, a network-wide agreement is required to define the so-called minimum inter-packet gap (IPG), in the same way as occurs with the inter-frame gap in Ethernet technology, or the guard bands defined for the KEOPS packet format [26]. The optical reconfiguration time of the node equipment should be below the (not yet standardized) IPG time. Schedulers must guarantee that two payloads allocated at an output wavelength in any link are head-to-tail separated by at least the IPG time.

B. Scheduler architecture

Fig. 2 depicts the main building blocks of the proposed scheduler architecture. It is based on the electronic interconnection of nN input modules (left hand side), and nN output modules (right hand side), connected by means of a crossbar interconnection for inter-module signaling. The operation of the scheduler is equal in the OPS and OBS/ODD case. For the sake of readability, we assume in the sequel an underlying OPS network.

1) Input modules description

One input module $IM(f,w)$ exists per each input fiber (f=0,..,N-1), and each input wavelength (w=0,...,n-1). Any input module stores and processes the information regarding one header, received in the associated header arrival time window. Therefore it must be guaranteed that at most one payload is associated with each input module in an algorithm execution. This implies that the minimum allowed payload length (L_{min}) plus the IPG must be greater than the period of the algorithm execution T_I.

$$L_{min} + \text{IPG} > T_I \qquad (2)$$

Both the payload arrival time and payload duration are stored altogether at the input module as a bit-mask register of length K_{in} bits. We denote as $R_{in}(f,w)$ the bit-mask input occupation register at input module $IM(f,w)$. Bit-mask registers are used as a method to permit fast overlapping check operations, as described later in this section.

2) Output modules description

One output module $OM(f,w)$ exists for each output fiber f=0,...,N-1, and output wavelength w=0,...,n-1. Each output module contains the following control registers: (i) one *output occupation bit-mask register* $R_{out}(f,w)$ of length K_{out} bits, which stores in a bit-mask form the time intervals that are occupied by previously allocated payloads in the associated output port, (ii) a register

$P(f,w)$ storing a *grant pointer*, of length $\log_2(nN)$ bits, and (iii) one bit register $CW(f,w)$ setting the scanning direction clock-wise or counter clock-wise of the grant pointer. The utilization of these registers will be made clear below.

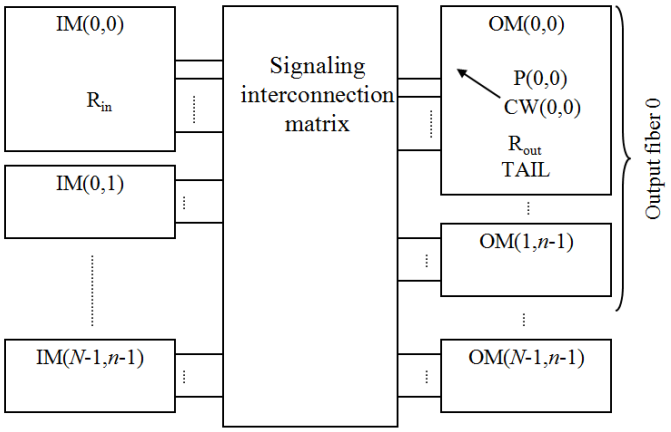

Fig. 2. Scheduler parallel architecture.

C. Bit mask occupation registers.

The PI-OPS algorithm aims at a simple electronic parallel implementation. To this end the occupation registers in the input and output modules are bit-mask registers. In the execution which starts at time $t=t_0$, the i-th bit in the occupation register (i=0,...,K), is associated with the time interval $[t_{\text{OSF-min}}+ig,\ t_{\text{OSF-min}}+(i+1)g)$, where $t_{\text{OSF-min}}=t_0-T_I+\delta+D_P$ is the earliest arrival time to the OSF of a payload whose header is being processed in current algorithm execution. g is a time granularity value corresponding to the duration of the time interval represented by one bit in the occupation register. A 1 at the i-th bit means that there is a payload already allocated in some portion (or all) of the time interval.

The time granularity value g is a significant system parameter to tune. On one hand, a coarse granularity (a high value of g) implies a loss of precision in the position of the allocated bursts. This may degrade the system performance since non-overlapping bursts could appear as overlapping. On the other hand, higher values of g imply a lower number of bits in the occupation register, which leads to a reduction of implementation complexity and algorithm response time. A relevant contribution of this paper is the exploration of this trade-off.

The length of the occupation registers in the input and output modules are different:

- Input modules: The first bit in the occupation register corresponds to the time interval $[t_{\text{OSF-min}},t_{\text{OSF-min}}+g)$. That is, it is associated with the earliest time of arrival of a payload which is being processed in this algorithm execution. The last bit corresponds to the time interval $[t_{\text{OSF-max}}-g,\ t_{\text{OSF-max}}]$, where $t_{\text{OSF-max}}$ is the latest possible arrival time of the last bit of a payload which is being processed in this algorithm execution: $t_{\text{OSF-max}}=t_0+\delta+D_P+L_{max}$. This corresponds to a packet whose header arrived at the end of the header arrival window (t_0) and has the maximum duration (L_{max}). Then, the number K_{in} of bits of the input module occupation registers is given by:

$$K_{in} = ceil\left(\frac{t_{OSF-\max} - t_{OSF-\min}}{g}\right) = ceil\left(\frac{T_I + L_{\max}}{g}\right) \qquad (3)$$

- Output modules: The difference with the previous register is that it must consider that the packet can be allocated an FDL. Therefore, the time of the last bit of the latest payload that can be allocated equals $t_{\text{OSF-max}}'= t_{\text{OSF-max}} +maxFDL$, where $maxFDL$ is the duration of the longest FDL that can be assigned to a packet. Then, the number K_{out} of bits of the output module registers equals

$$K_{out} = ceil\left(\frac{T_I + L_{\max} + \max FDL}{g}\right) \qquad (4)$$

D. Scheduler algorithm description

An algorithm execution starts periodically every T_I μs. Let us suppose that current algorithm execution starts at time $t=t_0$. At this moment, the input modules contain the control information of the packet headers which arrived in the header arrival time window $[t_0\text{-}T_I, t_0)$. In each output module, the R_{out} registers contain the occupation of the output wavelengths by the already allocated payloads.

Every algorithm execution is composed of a sequence of D delay cycles, one for each FDL in the OSF. Each delay cycle d is devoted to find a wavelength allocation for the arriving payloads if they were delayed by the corresponding FDL d, d=0,...,D-1. Each delay cycle is itself composed of a sequence of C_I *algorithm iterations*. An iteration associated with delay cycle d=0,...,D-1 consists of the following sequence of steps:

- *Request step*: This step takes one clock cycle. It is executed in parallel, in each of the input modules. The actions taken in the request step are slightly different in the first iteration of a delay cycle and in subsequent iterations. In the first iteration of a delay cycle, each input module sends a request signal to all the output modules associated with all the output wavelengths of the target output fiber f: $OM(f,w)$, w=0,...,n-1. In subsequent iterations of the same delay cycle, if the input module accepted the allocation of an output module (f,w) (described later), a request signal is only sent to that (f,w) output module. The importance of this different behavior will be explained later in this paper.
- *Serial overlap check*. This step takes K_{in} clock cycles. Each input module serially sends through the signaling interconnection matrix its own occupation register to those output modules for which a request signal was sent. Simultaneously, each output module will check the overlap of the occupations from the requested input modules against its own occupation register R_{out}. Each clock cycle, one bit is received from each input module and an AND operation is made with the associated bit of the R_{out} register. In the d-th delay cycle, the i-th bit in the input occupation registers is AND-ed with the $(i+dG/g)$-th bit of the R_{out} register. In other words, it is assumed that the payload goes through a FDL of length $d \cdot G$ μs. If the result of an AND operation is 1, then the payload from the associated input port overlaps a payload already allocated in the output wavelength. The overlapping flag associated with that input module is then set to 1. Checking in parallel the overlapping of the output wavelength occupation register and nN input module occupation registers is a core aspect in the scheduler. This aspect will be carefully addressed in Section V.
- *Grant step*: This step takes one clock cycle. It is executed in parallel, in each of the nN output modules. An arbiter circuit is responsible of scanning the overlapping flags sending a grant signal to the first input module without overlapping. The input modules are assumed to be internally arranged according to a lexicographical order (f,w). The scanning performed by the output module $OM(f,w)$ starts with the input module pointed by the grant pointer $P(f,w)$, and follows a clock-wise or counter-clockwise direction, depending on the state of the $CW(f,w)$ bit. These operations can be performed by arbiters implemented as fast combinational circuits [28]. As a result, a grant signal is sent to an input module with a packet which does not overlap either with existing packets scheduled in previous algorithm executions, nor packets allocated in *previous delay cycles*.
- *Serial minimum void computation*: This step takes $\log_2(K_{out})$ clock cycles. Each output module, after sending the grant signal, serially sends the $\log_2(K_{out})$ bits of the TAIL register associated with the granted input module. TAIL registers reflect the length of the *void* created if the payload was allocated. That is, the distance from the head of the allocated payload to the tail of the previous payload allocated in that output port. Each input module chooses, among the granted output modules, that with the smallest void. If that holds for more than one grant, the one with the lowest wavelength index is preferred (first-fit). This decision is stored in a register at the input module. We denote this action as *accepting* an output module. As will be shown in Section V, the serial reception of the bits allows for a simple electronic implementation of the circuit which selects the accepted module.

The only information that remains in the scheduler from one iteration to the next is stored at the input modules: each input module remembers the accepted output module. In the next iteration of the same delay cycle, a request signal will be sent solely to that output module. The output modules not requested are now free to grant other input modules. This is the source of the allocation improvement method, so that the number of optical packets which are allocated an output wavelength can increase in subsequent iterations.

After the last iteration in a delay cycle is completed, the allocations accepted by the input modules are considered final. Those input modules will not enter into play in subsequent delay cycles. Also, two further steps are needed in the last iteration of each delay cycle, to allow the output modules to update their occupation registers.

- *Serial output occupation register update*: This step takes K_{in} clock cycles. The input modules with an accepted packet send their input occupation registers to the accepted output modules. Each output module uses this information to update its occupation register, by performing a bit-OR operation between the bits in the current occupation register, corresponding to payloads that go through the d-th FDL, and the information received from the input modules.

At the end of the execution of the algorithm, the information in the input and output modules is processed as follows:

- Input modules: The information in the input modules is considered a final allocation and is sent to the circuit in charge of sending the reconfiguration signals to the OSF.
- Output modules: The information in the occupation registers of the output modules is prepared for the next algorithm execution. Each occupation register is bit-shifted by T_I/g bits, to reflect a payload propagation of T_I μs.

Fig. 3 illustrates with a simple example the main ideas behind PI-OPS operation. It depicts one iteration of the first delay cycle in a scheduler of parameters N=2, n=2. Only two input and two output modules are shown. The output modules correspond to two wavelengths in output fiber 0. The input modules correspond to two input ports with a packet targeted to output fiber 0. The occupation registers are drawn inside the modules, showing in black color the occupation by the arriving packets (input modules), and the occupation of previously allocated packets (output modules). Fig. 3(a) shows the request signals sent during the first iteration of the delay cycle: the input modules send a request to the two output modules. Then, in Fig. 3(b), the occupation registers of both input modules are serially transmitted to the requested output modules, and a serial overlap check is performed. Fig. 3(c) shows the grant signal sent by the output modules. In this example, the overlap check was successful in both output modules for the same input module, which thus receives two grant signals. The tail register stores the size of the void created. Fig. 3(d) depicts the serial transmission of the tail register to the granted module. The input module 0, receiving the two tail registers, chooses the smallest one (output module 1), which corresponds to the allocation with minimum void. Finally, Fig. 3(e) describes the register update occurring in the last iteration of the delay cycle. Input module 0 sends its occupation register to output module 1, which bit-adds it to its occupation register.

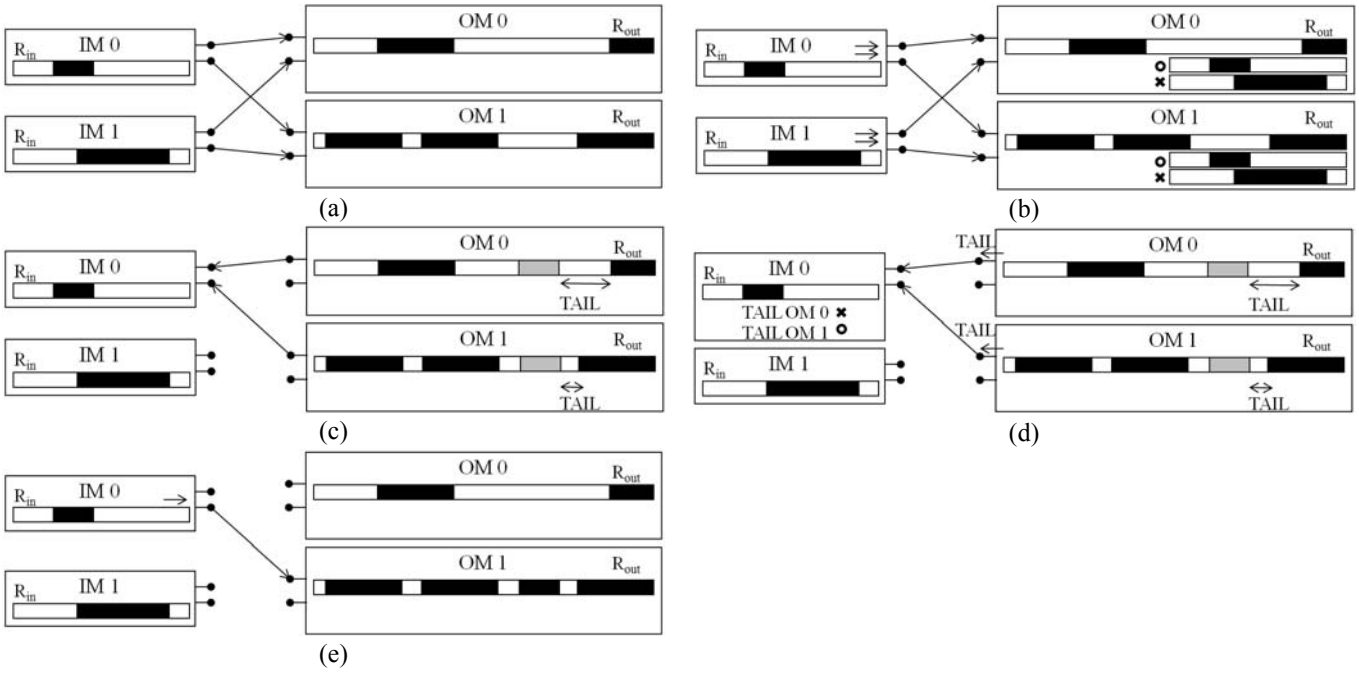

Fig. 3. PI-OPS: Example of one iteration

E. Grant pointers operation and system initialization

As it happens in the iSLIP scheduler for VOQ switches [10], the operation of grant pointers strongly affects the performance of the system. If an input module enters into a request step, it simultaneously sends a request signal to n output modules, one per output wavelength of the target output fiber. It is of interest to reduce the number of simultaneous grants an input module receives, as at most one grant can be accepted. The non-accepted grants correspond to FDL assignments not granted to other modules. Those candidate allocations will not enter into play until next iteration. Therefore, the grant pointers of output modules corresponding to the same output fiber should be *desynchronized*, in the sense that they point to input modules as separated as possible one from the other in the (f,w) lexicographical ordering. Then, we increase the chances that the grants are more uniformly spread among the input modules. Similarly to algorithm [29] for VOQ switches, and to algorithm [30] for slotted OPS switches, this can be obtained by: (i) a grant pointer initialization during system start-up which maximizes the minimum lexicographical distance between pointers, (ii) the *CW* bit is changed after every algorithm execution, switching the scanning direction of the input ports pointed by the grant pointers. This action aims to improve system fairness when packet arrivals are

not uniform across input fibers, in the same way as in [29][30]. (iii) Every two algorithm executions (with opposite scanning directions) all the pointers increase the value by one, modulo *nN*. As all the pointers update their value simultaneously, the pointers de-synchronization is maintained.

F. Algorithm convergence

We define algorithm convergence time as the number of iterations needed for the system to achieve a *stable allocation* in all the processed packets, which would not change if more iterations were performed. This prevents the variations in the allocation to persist eternally. The convergence of the PI-OPS is guaranteed by the *stability of accepted modules* property.

Proposition: If an input module *IM* receives a grant from output module *OM* and accepts it at iteration *i*, it will be granted again by the same *OM* in all the future iterations of the same delay cycle.

Proof: After accepting a grant at iteration *i*, *IM* sends a request signal solely to *OM* at iteration *i*+1. At iteration *i*+1, the *OM* will receive a subset of the requests it received in the previous iteration. Thus, if *IM* had the priority with respect to the other requests according to the grant pointer in *OM* at iteration *i*, it will be also prioritized in the subsequent iterations.

The previous proposition limits the maximum number of iterations to convergence to *nN*. This corresponds to the case in which at every iteration one new input module (not granted in the previous iterations) receives at least a grant, and thus accepts one output module. As shown in the result section, this theoretical convergence limit is not reached in normal operation, for which a moderate number of iterations (≤6 in all cases) suffices to achieve convergence.

IV. RESULTS

Figure 4 illustrates the testing scenario. The switch under evaluation S_E receives traffic from *N* source nodes and is responsible for switching it to *N* output sink nodes. Connecting fibers have *n* data wavelengths $\lambda_0,\ldots,\lambda_{n-1}$.

Two different scheduling algorithms will be evaluated in the S_E node: LAUC-VF and PI-OPS. LAUC-VF is a sequential algorithm commonly used in comparative performance evaluation studies, in optical packet/burst switching networks with asynchronous arrivals. Its operation consists of processing the headers one by one, allocating the smallest FDL possible. If more than one output wavelength can be allocated to the payload in the same minimum FDL, it is chosen the one which creates a smaller *void*. A data structure accounting with all the voids created is supposed to be available.

In our tests, the reconfiguration time of the optical equipment and the IPG time are assumed to be equal to 0.03 μs (T_O=IPG=0.03 μs). This is consistent with the reconfiguration time in SOA-based switch fabrics. Both the source nodes and the S_E node under test respect this IPG time in their assignments, so that each payload is followed by an IPG idle time.

In OPS/OBS networks, optical packets are made by assembling several IP packets, according to a defined assembling strategy. In our tests, the packet assembling is emulated by the source nodes. Each source node assembles optical packets of duration given by a truncated normal distribution. Minimum payload length is set to L_{min}=10 μs, and the maximum payload length to 100 μs. Average length is set to 55 μs. Three different variances of the payload duration are tested, corresponding to the coefficients of variation *CV*={0, 0.75, 1.5}. The CV values represent the ratio between the typical deviation and the mean of the normal distributions before being truncated. Note that *CV*=0 corresponds to fixed length payloads.

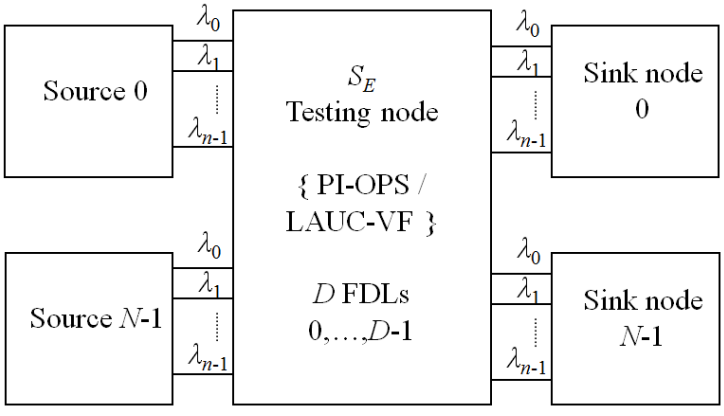

Fig. 4. Testing scenario

The target output node of the packets is selected randomly with uniform distribution. The time between the assembling of two consecutive packets is exponentially distributed. Its average is calculated to match the desired load value in the input fibers. After a packet is assembled in the source node, its transmission wavelength and injection time in the connecting fiber are selected as if the input node was a LAUC-VF node, with an infinite number of FDLs. Using LAUC-VF source nodes is considered a simple and effective way to generate more realistic traffic profiles reproducing the correlations in packet arrivals

over different wavelengths of the same fiber in an OPS/OBS network. The rationale is to make the incoming packet flows behave as if they already crossed part of an OPS/OBS network. The assumption of an infinite number of FDLs in the upstream LAUC-VF allocation allows to inject the node with the same load assumed in the packet generation process.

The granularity of the FDLs in the source nodes and the S_E node is made equal to 55 μs, the average packet length. Previous works have shown that a FDL granularity close to the average payload duration optimizes system performance [31]. The time between two algorithm executions is set to T_I=10 μs, so the design constraint (2) is satisfied with a small margin of 30 ns. The PI-OPS algorithm response time is assumed to be also T_A=10 μs, making the constraint $T_I \geq T_A$ tight. The chosen parameters $L_{min}=T_I=T_A$=10 μs are consistent with system constraints (1) and (2), and as will be shown later, also permit a feasible implementation of the PI-OPS scheduler.

The operation of the PI-OPS algorithm in asynchronous OPS networks and in OBS networks with the ODD reservation protocol is totally equivalent. The only difference comes from the dimensioning of the per-input port fiber delay D_P. In OPS networks, packet headers are attached to packet payload without offset. The per-input-port FDL should be dimensioned as $D_P=T_I+T_A+T_O$=20.03 μs. In the OBS case $D_P=T_I+T_A$ =20 μs, and bursts have a constant offset $\delta=T_O$= 0.03 μs. Consequently, a header arriving at $t=t_0$ is in any case associated with a payload which arrives to the OSF of the node $T_I+T_A+T_O$=20.03 μs later. Given this equivalence from the scheduler point-of-view, in this paper, we show the results obtained in the OPS case, being the results in the OBS-ODD case equal.

Performance evaluation has been conducted by means of discrete event simulation. The simulation tool has been built on top of the OMNeT++ platform [32]. All the tests performed consist of 5 independent samples, with 10^8 generated packets per source node. The simulation time is adjusted to fulfill this requirement. Note that the simulation time is thus dependent on the system load and number of wavelengths per fiber. Confidence intervals are calculated for a 95% quality, using the *t*-Student method. Confidence intervals obtained validate the results, but are not shown in the figures for the sake of clarity.

The first step in our study addresses the dimensioning of the optical buffering B in the S_E switch, required for guaranteeing an average *bit loss probability* below 10^{-5} for an 80% input load. Measuring the bit loss probability means that the loss of a packet is weighted by its duration. The results shown in Table I cover the cases of N={4,8} input/output fibers, n={20,40,80} wavelengths per fiber, and coefficient of variation of packet payload CV={0,0.75,1.5}. Tests are repeated for (i) the LAUC-VF scheduler and (ii) the PI-OPS scheduler with a sufficient number of iterations, to guarantee algorithm convergence, and an infinitesimal granularity of the occupation register. This means that no loss of precision in the payload initial and end times exists because of the finite number of bits in the occupation register. Then, we compare the LAUC-VF with the upper performance limits of PI-OPS algorithm.

Results reflect the known buffering requirements reduction effect for higher number of wavelengths per fiber. This happens since both algorithms exploit the wavelength dimension to solve contention. Table I shows that PI-OPS nodes require *one* more FDL in the OSF than LAUC-VF nodes when the number of wavelengths is n={20,40} (two in the case n=20, CV=0). In its turn, the buffering needs are *the same* in the DWDM case n=80. Note that in this situation, two FDLs suffice to guarantee a respectable traffic loss probability of 10^{-5} at traffic load of 80%.

TABLE I
NUMBER OF FDLS B FOR A BIT LOSS PROBABILITY OF 10^{-5}, UNDER 80% INPUT LOAD. SCHEDULERS {PI-OPS/LAUC-VF}

N, CV \ n	n= 20	n = 40	n = 80
N =4; CV=0	6 / 4	3 / 2	2 / 2
N =4; CV=0.75	5 / 4	3 / 2	2 / 2
N =4; CV=1.5	5 / 4	3 / 2	2 / 2
N =8; CV=0	6 / 4	3 / 2	2 / 2
N =8; CV=0.75	5 / 4	3 / 2	2 / 2
N =8; CV=1.5	5 / 4	3 / 2	2 / 2

It is interesting to see that LAUC-VF exhibits a better traffic loss performance than PI-OPS algorithm, despite of its sequential operation. Fig. 5 helps us to illustrate this aspect. It compares the LAUC-V and PI-OPS traffic loss performance for the parameters n={20,40,80}, N={4}, CV={0.75}, B={2} and input load factors ρ={0.5, 0.55, ..., 0.95}. Results show that if the same buffering is used, LAUC-VF outperforms PI-OPS. Larger performance gaps are seen with a higher number of wavelengths per fiber. Similar results obtained for other buffering parameters B, not shown in this paper, confirm this effect. This suggests that PI-OPS is not able to totally exploit the potential benefits of the group scheduling. One reason for this, is that PI-OPS has been designed keeping in mind the hardware implementation complexity. Then, the payload allocation process is constrained in some steps. For instance, in PI-OPS once an input port is granted by an output module in an iteration, it will not request any different output module in further iterations. On the one hand, this method permits the improvements of the allocation in

subsequent iterations, in a manner which allows a practical hardware implementation. On the other hand, it strongly constraints how the allocations are improved.

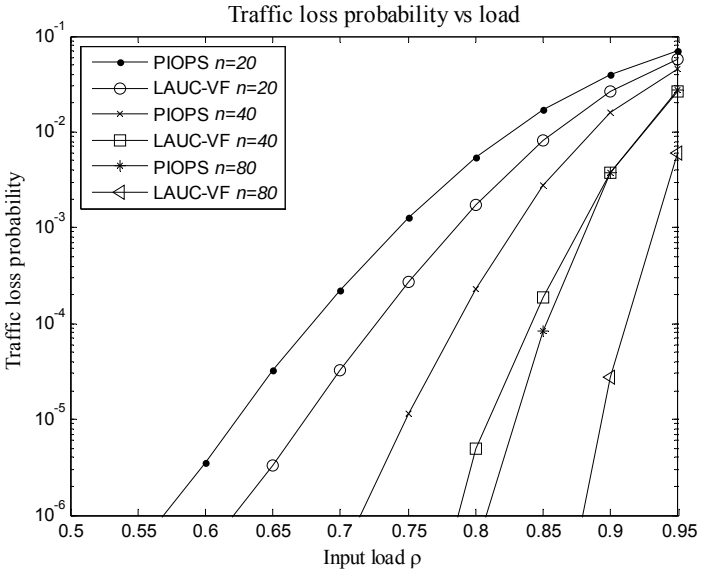

Fig. 5. PI-OPS vs LAUC-VF traffic loss performance., N=4, CV=0.75, B=2.

The data associated with PI-OPS in the Table I set an optimum performance limit for this scheduler, if infinite hardware resources were available to implement it. However, to be able to study the practical feasibility of PI-OPS we are forced to assess the sensibility of these results with respect to (i) the number of iterations in the algorithm C_I, (ii) the scheduler time granularity value g. Recall that g represents the time interval associated with one bit in the input and output bit-mask occupation registers. Finer (lower) values of g result in more bits in the occupation registers, and thus more hardware resources for building the input and output modules, and more time to complete the serial overlapping and minimum void selection steps in the algorithm.

Table II collects relevant information to evaluate this sensibility. For each of the N={4,8}, n={20,40,80}, CV={0,0.75,1.5} cases studied above, we study the coarser granularity g for which the optical buffering requirements of PI-OPS are the same as in the PI-OPS upper performance bound. That is, the best case in which using a finite precision does not degrade the optical buffering performance of PI-OPS nodes. For the sake of clarity, instead of showing the results of g (measured e.g. in nanoseconds), we plot the values 10μs/g, that is, the number of bits needed to represent an interval of 10 μs. The more the number of bits, the higher the length of the occupation registers. Next to the 10μs/g value, we also show the number of iterations C_I for which convergence was reached in the 99.9% of the occasions. That means, for which convergence is guaranteed in the practice. Note that if convergence is not reached in a delay cycle, it means that some packets will likely be assigned a higher FDL. Only if this happens in the delay cycle associated with the last iteration, some of the packets can be lost, that would not be lost if convergence was reached.

TABLE II
PI-OPS GRANULARITY AND CONVERGENCE TIME PRESERVING FDL DIMENSIONING IN TABLE I. VALUES (10μS/G) / C_I

n / *N,CV*	*n*= 20	*n* = 40	*n* = 80
N =4; *CV*=0	5 / 4	6 / 4	5 / 6
N =4; *CV*=0.75	7 / 4	4 / 4	4 / 6
N =4; *CV*=1.5	6 / 4	4 / 4	4 / 6
N =8; *CV*=0	7 / 4	8 / 5	5 / 6
N =8; *CV*=0.75	4 / 4	4 / 4	4 / 6
N =8; *CV*=1.5	4 / 4	4 / 4	4 / 6

Results show that the number of bits needed to represent 10μs ranges from 4 to 8, corresponding to g=2.5μs and g=1.25μs respectively. The coarser values g=2.5μs are found for n=40 and n=80 which are also the most interesting cases from the optical buffering point of view. For both n={40,80} a finer granularity g seems to be required when fixed length packets are used (CV=0). The case n=20 shows worse results, especially in nodes with degree N=4.

The values of the number of iterations to convergence C_I observed are practically independent from the CV and N parameters. Four iterations are needed for a number of wavelengths per fiber n={20,40}, and six in the case n=80. This slight increase in the

number of iterations for the DWDM case is caused by the operation during the first iterations of the algorithm. In those iterations, not granted input modules persist sending a request to all output modules of the target output fiber. As PI-OPS progresses, output modules receive less and less requests, since granted input modules reduce its request signals to 1. Intuitively, the more the number of output modules, the higher the chances of this process to take more iterations.

V. Electronic implementation

Actual implementations of optical packet schedulers are not common in literature. An example is given in [33], which describes a system design for implementing an OBS scheduler based on void filling using associative memories and parallel computation to find the optimal channel assignment. However, burst headers are still processed serially and there is no evaluation of the system processing time.

This section is aimed at providing an insight on how the physical design of an optical packet scheduler may affect the logical performance achievable in theory, referring to the proposed solution. What will be described next is a possible FPGA implementation of the scheduling sub-modules that perform the most critical tasks from the processing point of view, followed by the emulation of a corresponding board design. This is the first step towards a more comprehensive feasibility study, which would require the development of a complete test-bed and, therefore, is out of the scope of this paper.

According to the algorithm description, the number of clock cycles required to complete a scheduling time window depends on the number of bits in the input and output occupation registers and on the number of iterations needed to reach convergence. Table III shows the dimensioning of the input and output contention registers of the PI-OPS scheduler, associated with the results obtained in Table II, according to (3) and (4). The third column displays the resulting maximum clock period (in nanoseconds) allowed to maintain a constant scheduler response time T_A = 10μs. The focus here is on the most demanding algorithm steps, which consist in *Serial Overlap Check* and *Serial Minimum Void Computation* according to the discussion in Section III-D.

The functional blocks used in the implementation of the *Serial Overlap Check* and *Grant* steps are shown in Fig. 5 for a generic output module. Each output module *OM*(*f,w*) is connected to each input module *IM*(*f,w*) with a serial line D_k and to its arbiter with *nN* signals OV_0, … , OV_{Nn-1} representing the overlapping flags. A global *Scheduler Controller*, which is in charge of coordinating the different processing steps, sets the C signal to on when the *Serial Overlap Check* step starts, telling the output module to read the bits incoming on the serial lines connected to the input modules which sent a request signal in the previous phase.

TABLE III
HARDWARE DIMENSIONING FOR THE SCHEDULERS BUILT ACCORDING TO TABLE II. {K_{IN}, K_{OUT}, CLOCK PERIOD (NS)}

N \ *n*	*n*= 20	*n* = 40	*n* = 80
N =4; *CV*=0	55 / 193 / 5.3	66 / 132 / 9	55 / 83 / 11.4
N =4; *CV*=0.75	77 /231 / 4.7	44 / 88 / 13	44 / 66 / 13.8
N =4; *CV*=1.5	66 /198 / 5.4	44 / 88 / 13	44 / 66 / 13.8
N =8; *CV*=0	77 / 270 / 3.9	88 / 176 / 5.8	55 / 83 / 11.4
N =8; *CV*=0.75	44 / 132 / 7.7	44 / 88 / 13	44 / 66 / 13.8
N =8; *CV*=1.5	44 / 132 / 7.7	44 / 88 / 13	44 / 66 / 13.8

What happens inside each output module is shown in Fig. 6. Using the serial line OOR, the content of the output occupation register is compared bit-by-bit with the content of the input module registers received on the serial lines D_k to detect any overlapping between the incoming packet and those already scheduled on the output channel. If an overlapping is detected with the packet incoming on D_i, then the overlapping signal OV_i becomes active and the arbiter is able to exclude the corresponding input module from the following *Grant* step. At the same time, the position of the last bit set to one in the output occupation register preceding the first bit set to one received on line D_j is stored in the tail register T_j. In other words, each tail register stores the position of the tail of the already scheduled packet closest to the head of the packet incoming on the corresponding input module.

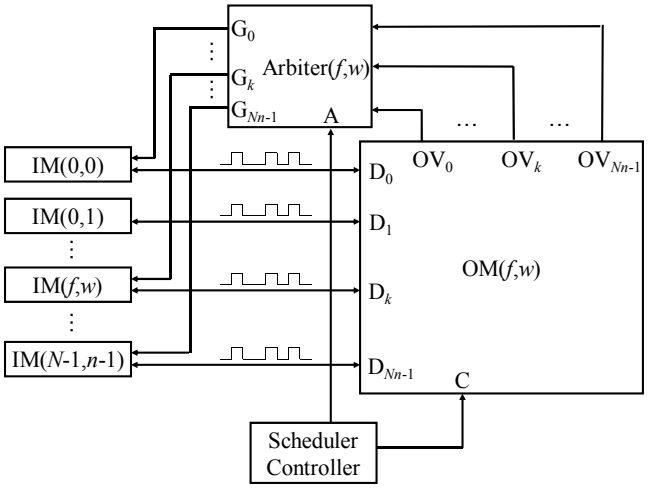

Fig. 5. Output module and arbiter used in *Serial Overlap Check* and *Grant* steps implementation

An example of how the output module works during the *Serial Overlap Check* step is shown in Fig. 7: the packet coming from input module *i* overlaps with the content of the output occupation register, whereas the packets coming from input modules *j* and *k* can be successfully scheduled within two different voids, resulting in two different values of the tail register.

Once the *Serial Overlap Check* step is completed, the global Scheduler Controller sets the C signal to off and starts the *Grant* step by activating the A signal, that tells the arbiter to check the overlapping flags and select the input module towards which the grant signal G_k should be made active.

FPGA implementation of the functional blocks of Figs. 5 and 6 has been simulated by means of Quartus II software v9.0, assuming Stratix III boards and obtaining a realistic timing analysis without any fitter optimization. The waveforms resulting from the emulation of the *Serial Overlap Check* step are shown in Fig. 8 with reference to a single input-output module pair. The signals are defined as follows: *clk* is the clock signal; *count_en* is an enabling signal representing C; *Input_bus* is a streaming input line representing serial line D_k; *Occupation_reg* is a streaming input line representing serial line OOR; *out_data* is the value of the tail register T_k; *out_en* is the overlapping signal OV_k.

The waveforms on the top of the figure refer to the case when a packet contention was emulated, whereas those on the bottom have been obtained considering the case when no overlapping takes place. A 128-bit output occupation register was used and the circuit works correctly with a clock period of 10 ns. However, the emulated circuit was able to tolerate a maximum clock frequency of 236.52 MHz, corresponding to a clock period of 4.228 ns. The *Serial Overlap Check* step has also been emulated with a full-featured output module connected to 64 input modules, measuring the timings of the hardware implementation of the functional blocks of Fig. 5 with $nN = 64$. The maximum clock frequency obtained in this case was 179.50 MHz, leading to a minimum clock period of 5.571 ns. This is quite a good result if compared with the requirements in Table III, although in some cases the clock is not fast enough. It is then reasonable to assume that the implementation of *Serial Overlap Check* step does not heavily affect the logical performance.

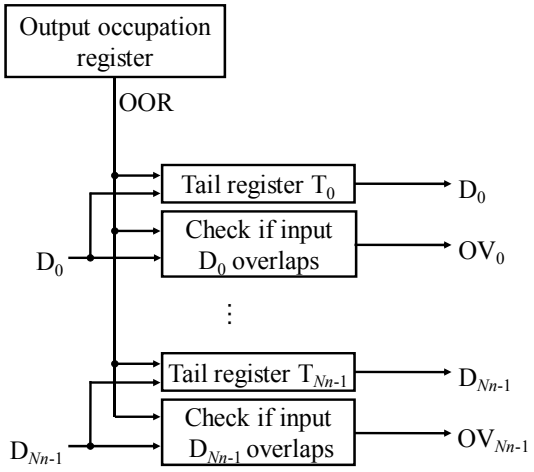

Fig. 6. Functional blocks used in the output module implementation

OOR	0	1	1	0	0	0	1	1	0	0	0	0	0
$\mathbf{D_i}$	0	0	0	0	0	1	1	1	1	1	0	0	0
$\mathbf{OV_i}$	0	0	0	0	0	0	1	1	1	1	1	1	1
$\mathbf{T_i}$	0	2	3	3	3	3	–	–	–	–	–	–	–
$\mathbf{D_j}$	0	0	0	0	0	0	0	0	0	0	1	1	1
$\mathbf{OV_j}$	0	0	0	0	0	0	0	0	0	0	0	0	0
$\mathbf{T_j}$	0	2	3	3	3	3	7	8	8	8	8	8	8
$\mathbf{D_k}$	0	0	0	0	1	1	0	0	0	0	0	0	0
$\mathbf{OV_k}$	0	0	0	0	0	0	0	0	0	0	0	0	0
$\mathbf{T_k}$	0	2	3	3	3	3	3	3	3	3	3	3	3

Fig. 7. Example of output module operations during the *Serial Overlap Check* step

When the generic input module *IM(f,w)* receives one or more grant signals from the relevant output modules, the final *Serial Minimum Void Computation* step begins. The implementation details of the generic input module operations during such step are shown in Fig. 9.

The global Scheduler Controller activates the S signal to let the input module know that the *Serial Minimum Void Computation* step has begun and that the content of the tail registers is being received on the serial lines D_0, …, D_{Nn-1} coming from the output modules. Obviously, only the lines corresponding to the output modules which sent a grant signal in the previous step are considered valid. The input module performs a bit-wise comparison of the value of the tail registers received and selects the highest one, corresponding to the channel with the minimum starting void as dictated by the PI-OPS algorithm.

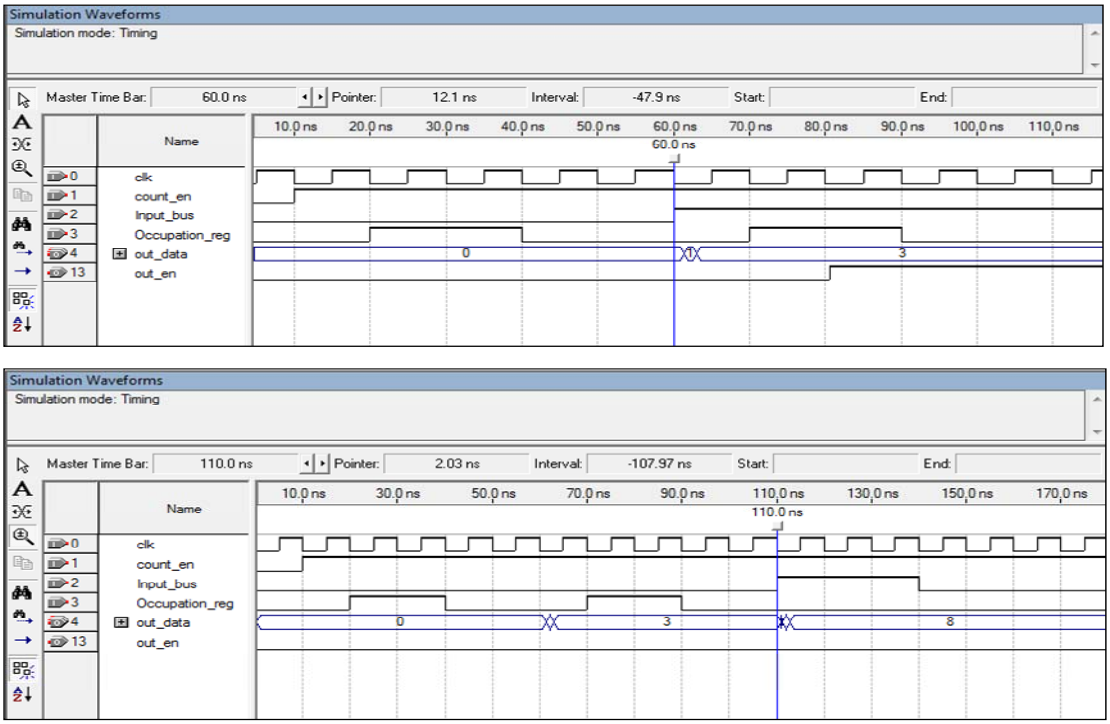

Fig. 8. Waveforms of the Serial Overlap Check step implementation. The one at the top shows the case of an overlapping packet, whereas in the one at the bottom the incoming packet does not overlap with the content of the output occupation register

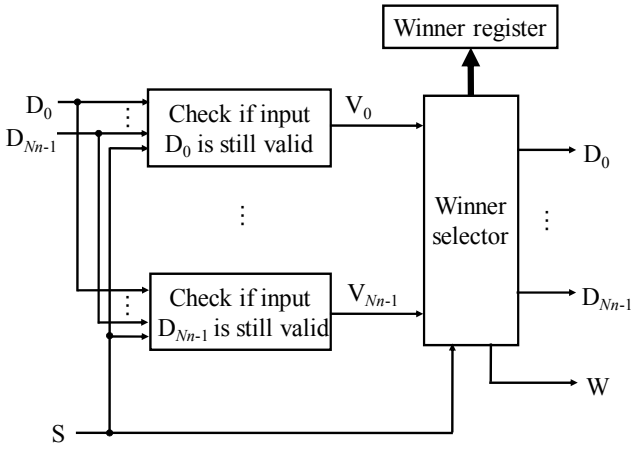

Fig. 9. Functional blocks used in the input module implementation

In particular, the circuit compares all incoming bits to check whether the value of a given output module tail register must still be considered as a potential maximum, i.e. it is still valid. The generic tail register being received on serial line D_k is still valid (i) if its current bit value is set to one or (ii) if it is set to zero and all the other bit values of valid tail registers are also set to zero. This operation is easily implemented with fast combinational logic operating on the serial bit stream. The validity signals V_0, …, V_{Nn-1} remain active as long as the corresponding tail register is still valid. The maximum value is found either when only one validity signal is left active or when all the tail register bits have been received, in which case multiple output modules provide the same optimum void, as measured by the current time granularity *g*.

The winner selector block is then in charge of storing the index of the best output channel in the winner register (making a choice in case of multiple valid tail registers) and setting the W signal to on, which tells the Scheduler Controller that at least a winner was found. The Scheduler Controller then sets the S signal to off, meaning that the *Serial Minimum Void Computation* step is over. In case of the last iteration of the scheduling algorithm, the W signal is also received by the output modules to alert them to update the output occupation register.

Two examples of how the input module works during the *Serial Minimum Void Computation* step are shown in Fig. 10: in the top figure, the optimum channel among *i*, *j* and *k* is found before the complete tail registers are received, whereas the bottom figure shows the worst case when the winner is found only at the end of the step.

The emulation of an FPGA implementation of the *Serial Minimum Void Computation* step for the case of 4 tail registers proved that the circuit works correctly with a clock period of 10 ns. Furthermore, the emulated circuit was able to tolerate a maximum clock frequency of 350.02 MHz, corresponding to a clock period of 2.857 ns. However, when the scheduler design requires several tens of such input modules, the blocks implementing bit-wise operations need larger fan-in/fan-out levels, resulting in a cascade of multiple logical gates and a significant increase of the signal propagation times across the FPGA board. This, of course,limits the minimum clock period attainable and affect the logical performance. Therefore, although the single input module looks feasible as it is, more advanced electronic design optimization techniques, such as the adoption of pipelined processing or the use of a parallel logic to compute the maximum over a set of values similar to the one described in [24], are needed to reduce the impact on the logical performance.

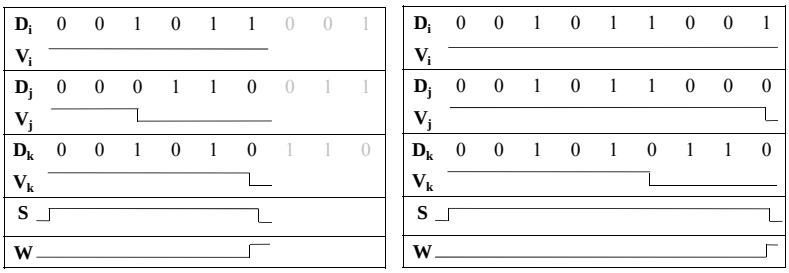

Fig. 10. Two examples of input module operations during *the Serial Minimum Void Computation* step

VI. Conclusion

This paper presents an optical packet scheduling algorithm based on a parallel, iterative mechanism similar to the one successfully implemented in routers with virtual output queuing. The proposed approach is suitable for OPS nodes with asynchronous arrivals and OBS networks with Only Destination Delay reservation mechanism. The algorithm details have been

described and simulations have been performed in order to evaluate the logical performance and provide general node design guidelines to keep the total scheduling time below 10 μs.

The parallel and iterative processing introduced by the proposed solution is the key that allows to define a guaranteed maximum execution time. This is a significant step towards the actual implementation of efficient control logic for OPS nodes, one of the research issues that still need to be investigated with appropriate attention.

Emulations of possible FPGA implementations of the sub-modules in charge of the most critical tasks of the scheduling process have been performed, showing how the actual implementation of the scheduler may affect the expected logical performance, requiring to either find a sub-optimal scheduling solution (e.g. by reducing the number of bits in the registers or the number of iterations to fit the target total scheduling time) or to increase the expected scheduling time. Although the real impact of the physical design can be evaluated only with a complete implementation of the control logic, the FPGA emulations shown here allow to understand in which direction the optimization effort should be targeted.

In authors' opinion, PI-OPS should be seen as a first approach in the design of parallel-iterative schedulers in asynchronous OPS networks. The studies conducted support the practical feasibility of this strategy. The field is open for further investigations exploring e.g. the multiple cost vs performance trade-offs that appear in the design of this type of schedulers.

REFERENCES

[1] S. J. Ben Yoo, Optical Packet and Burst Switching Technologies for the Future Photonic Internet, IEEE/OSA Journal of Lightwave Technology,24(12) (2006) 4468-4492.

[2] J. P. Jue, W.-H. Yang, Y.-C. Kim, Q. Zhang, Optical Packet and Burst Switched Networks: A Review, IET Communications, 3(3) (2009) 334-352.

[3] R. Parthiban, C. Leckie, A. Zalesky, M. Zukerman, R. S. Tucker, Cost Comparison of Optical Circuit-Switched and Burst-Switched Networks, IEEE/OSA Journal of Lightwave Technology, 27(13) (2009) 2315-2329.

[4] V. Eramo, Comparison in Power Consumption of Synchronous and Asynchronous Optical Packet Switches, IEEE/OSA Journal of Lightwave Technology, 28(5) (2010) 847-857.

[5] F. Callegati, W. Cerroni, G. Corazza, Optimization of Wavelength Allocation in WDM Optical Buffers, Optical Networks Magazine, SPIE/Kluwer Academic Publishers, 2(6) (2001) 66-72.

[6] C. Gauger, Dimensioning of FDL Buffers for Optical Burst Switching Nodes, in: Proceedings of ONDM 2002, Torino, Italy, February 2002.

[7] G. Grasso, P. Galli, M. Romagnoli, E. Iannone, A. Bogoni, Role of Integrated Photonics Technologies in the Realization of Terabit Nodes, IEEE/OSA Journal of Optical Communications and Networking, 1(3) (2009) B111-B119.

[8] M. Maier, A. Hamad, M. Herzog, A Global Overview of Optical Switching Network Testbed Activities, IEEE/OSA Journal of Lightwave Technology, 27(19) (2009) 4377-4384.

[9] F. Callegati, Optical Buffers for Variable Length Packets, IEEE Communications Letters, 4(9) (2000) 292-294.

[10] N. McKeown, iSLIP: A Scheduling Algorithm for Input-Queued Switches, IEEE/ACM Transactions on Networking, 7(2) (1999) 188-201.

[11] P. Pavon-Mariño, J. Veiga-Gontan, A. Ortuño-Manzanera, W. Cerroni, J. Garcia-Haro, PI-OBS: a Parallel Iterative Optical Burst Scheduler for OBS networks, in: Proceedings of HPSR 2009, Paris, France, June 2009.

[12] L. Xu, H. G. Perros, G. N. Rouskas, A Simulation Study of Optical Burst Switching and Access Protocols for WDM Ring Networks, Computer Networks, Elsevier, 41(2) (2003) 143-160.

[13] J. S. Turner, Terabit Burst Switching, Journal of High-Speed Networks, 8(1) (1999) 3-16.

[14] Y. Xiong, M. Vandenhoute, H. C. Cankaya, Control Architecture in Optical Burst-Switched WDM Networks, IEEE Journal on Selected Areas in Communications, 18(10) (2000) 1838-1851.

[15] L. Tancevski, S. Yegnanarayanan, G. Castanon, L. Tamil, F. Masetti, T. McDermott, Optical Routing of Asynchronous, Variable Length Packets, IEEE Journal on Selected Areas in Communications, 18(10) (2000) 2084-2093.

[16] F. Callegati, W. Cerroni, C. Raffaelli, P. Zaffoni, Wavelength and Time Domain Exploitation for QoS Management in Optical Packet Switches, Computer Networks, Elsevier, 44(4) (2004) 569-582.

[17] M. Yang, S. Q. Zheng, D. Verchere, A QoS Supporting Scheduling Algorithm for Optical Burst Switching DWDM Networks, in: Proceedings of IEEE Globecom 2001, San Antonio, TX, December 2001.

[18] N. Akar, E. Karasan, K. Vlachos, M. Varvarigos, D. Careglio, M. Klinkowski, J. Solé-Pareta, A Survey of Quality of Service Differentiation Mechanisms for Optical Burst Switching Networks, Optical Switching and Networking, Elsevier, 7(1) (2010) 1-11.

[19] F. Callegati, D. Careglio, W. Cerroni, G. Muretto, C. Raffaelli, J. Solé-Pareta, P. Zaffoni, Keeping the Packet Sequence in Optical Packet-Switched Networks, Optical Switching And Networking, Elsevier, 2(3) (2005) 137-147.

[20] V. M. Vokkarane, J. P. Jue, Segmentation-Based Nonpreemptive Channel Scheduling Algorithms for Optical Burst-Switched Networks, IEEE/OSA Journal of Lightwave Technology, 23(10) (2005) 3125-3137.

[21]J. Xu, C. Qiao, J. Li, G. Xu, Efficient Burst Scheduling Algorithms in Optical Burst-Switched Networks Using Geometric Techniques, IEEE Journal on Selected Areas in Communications, 22(9) (2004) 1796-1811.
[22]G. Muretto, C. Raffaelli, P. Zaffoni, Effective Implementation of Void Filling in OBS Networks With Service Differentiation, in: Proceedings of WOBS 2004, San José, CA, USA, October 2004.
[23]Y. Chen, J. Turner, P. F. Mo, Optimal Burst Scheduling in Optical Burst Switched Networks, IEEE/OSA Journal of Lightwave Technology, 25(8) (2007) 1883-1894.
[24]F. Callegati, A. Campi, W. Cerroni, A Practical Approach to Scheduler Implementation for Optical Burst/Packet Switching, in: Proceedings of ONDM 2010, Kyoto, Japan, February 2010.
[25]S. Charcranoon, T. S. El-Bawab, H. C. Cankaya, J.-D. Shin, Group-Scheduling for Optical Burst Switched (OBS) Networks, in: Proceedings of IEEE Globecom 2003, San Francisco, CA, December 2003.
[26]C. Guillemot, *et al.*, Transparent optical packet switching: the European ACTS KEOPS project approach, IEEE Journal of Lightwave Technology, 16(12) (1998) 2117-2134.
[27]S. L. Danielsen, C. Joergensen, B. Mikkelsen and K. Stubkjaer, Analysis of a WDM packet switch with improved performance under bursty traffic conditions due to tunable wavelength converters, IEEE Journal of Lightwave Technology, 16(5) (1998) 729-735.
[28]C. K. Hung, et al., Design and implementation of high-speed arbiter for large-scale VOQ crossbar switches, in: Proceedings of ISCAS 2003 Conference, Bangkok, Thailand, May 2003.
[29]Y. Jiang and M. Hamdi, A fully Desynchronized Round-Robin Matching Scheduler for a VOQ Packet Switch Architecture, in: Proceedings of IEEE Workshop on High Performance Switching and Routing, Dallas, 2001, 407-411.
[30]P. Pavon-Marino, J. Garcia-Haro, A. Jajszczyk, Parallel Desynchronized Block Matching: A Feasible Scheduling Algorithm for the Input-Buffered Wavelength-Routed Switch, Computer Networks, 51(15) (2007) 4270-4283.
[31]D. Careglio, J. Solé-Pareta, S. Spadaro, Novel contention resolution technique for QoS support in connection-oriented optical packet switching, in: Proceedings of ICC 2005, Seoul, Korea, May 2005.
[32]OMNET++. Available: http://www.omnetpp.org
[33]S. Q. Zheng, Y. Xiong, M. L. J. Vandenhoute, Hardware implementation of channel scheduling algorithms for optical routers with FLD buffers, U.S. Patent no. 6804255, October 2004.