Crosstalk-Preventing Scheduling in Single- and Two-Stage AWG-Based Cell Switches

# Crosstalk-Preventing Scheduling in Single- and Two-Stage AWG-Based Cell Switches

Andrea Bianco, *Senior Member, IEEE*, David Hay, *Member, IEEE*, and Fabio Neri, *Senior Member, IEEE*

***Abstract*—Array waveguide grating (AWG)-based optical switching fabrics are receiving increasing attention due to their simplicity and good performance. However, AWGs are affected by coherent crosstalk that can significantly impair system operation when the same wavelength is used simultaneously on several input ports. To permit large port counts in a $N \times N$ AWG, a possible solution is to schedule data transmissions across the AWG preventing switch configurations that generate large crosstalk. We study the properties and the existence conditions of switch configurations able to control coherent crosstalk. The presented results show that, by running a properly constrained scheduling algorithm to avoid or minimize crosstalk, it is possible to operate an AWG-based switch with large port counts without significant performance degradation.**

***Index Terms*—Arrayed waveguide grating (AWG), coherent crosstalk, input-queued switches, optical switchings, scheduling algorithms.**

## I. Introduction

INTERNET traffic has been increasing at a pace faster than Moore's law, and electronic technologies may not be able to support the realization of large packet switches and IP routers in the near future. Power density and dissipation, in particular, are becoming major bottlenecks [1]. Optical technologies may help in overcoming intrinsic limitations of current switching architectures [2], [3]. Indeed, photonic technologies exhibit a number of interesting properties: a switching complexity almost independent of the data rate, very high data rates supported via large information densities on physical interconnections, no significant constraints on the physical size of the switch and on the length of internal switch interconnections (while electrical backplanes and interconnects have severe distance limitations), and very good scalability of power requirements.

All-optical packet switches are still far from being feasible due to several limitations such as the lack of optical memories, the very limited data processing capabilities, and the inherent difficulties in realizing functions in the time domain. Therefore, switching architectures in the near future will exploit both electronics and photonic technologies [4]: Packet processing and storing will likely be realized in electronics, while packet forwarding will likely rely on an optical switching fabric.

A promising approach to realize an optic switching fabric is to use a passive wavelength routing device with tunable transmitters and receivers at the device's inputs/outputs.

Given current line rates, packet switching requires very short switching times, in the nanoseconds range. Few mature optical technologies are available to build optical packet-switching fabrics and interconnects. One promising approach is to use a passive wavelength routing device with tunable transmitters at the inputs. Switching is obtained by tuning transmitters to different wavelengths. Thus, switching times are related to laser tuning times. Solid-state tunable lasers with adequate tuning times are available today. As for passive devices, arrayed waveguide gratings (AWGs) [5] are a good candidate. AWGs are wavelength routing devices that have been widely used in the commercial deployment of wavelength division multiplexing (WDM) transmission systems. They exploit well the advantages of the optical domain, using wavelength agility to harness the large available bandwidth.

AWGs are passive devices behaving as multiport interferometers. In the $1 \times N$ $(N \times 1)$ configuration, AWGs act as wavelength multiplexers (demultiplexers). In the $N \times N$ configuration, AWGs behave as wavelength routers: The information at an input port is forwarded to an output port depending on the selected wavelength. More specifically, at each input port, different wavelengths are used to reach different output ports. Since AWG devices are symmetrical, i.e., the role of input and output ports and the direction of forwarded information can be reversed, information is received at each output port from different inputs with different wavelengths. Overall, an $N \times N$ AWG can be simultaneously traversed by $N^2$ information flows, one for each input/output pair, leading to a full mesh bipartite connectivity exploiting space and wavelength separation.

The specific wavelengths used to route information through an AWG depend on the device design, but commercial devices typically exploit the ITU grid standard bands, with 100- or 50-GHz spacing. Although other wavelength assignments are possible, we assume in this paper, with no loss of generality, that: 1) the $N \times N$ AWG operates with a set of $N$ wavelengths $\Lambda = \{\lambda_0, \lambda_1, \ldots, \lambda_{N-1}\}$; and 2) at input $i$, $0 \leq i \leq N-1$, information is delivered to output $j$, $0 \leq j \leq N-1$, using wavelength $\lambda_k$, with $k = (j-i) \bmod N$ being the wavelength channel number. This cyclic behavior is typical of the interferometric nature of the AWG, whose routing behavior is replicated over the wavelength axis with a period called free spectral range (FSR). Our assumptions on the AWG behavior

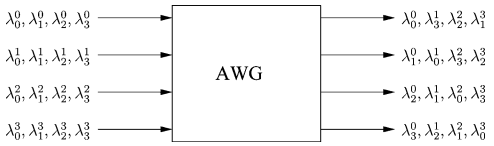


Fig. 1. Considered routing for the AWG device.

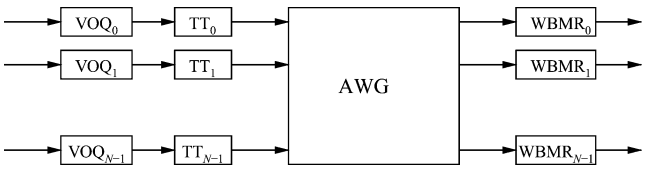


Fig. 2. Simple single-stage AWG-based switch.

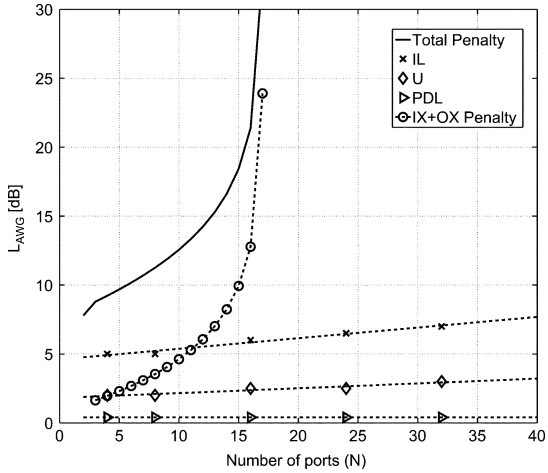


Fig. 3. Power penalties as a function of port count $N$ in an $N \times N$ AWG.

imply that only $N$ wavelengths are needed to support $N^2$ connections over $N$ input and $N$ output ports. Fig. 1 depicts, for a $4 \times 4$ AWG, the wavelength behavior considered in this paper, where superscripts refer to input port indices, and subscripts to wavelength channel numbers.

We consider the nonblocking optical interconnection among input and output ports depicted in Fig. 2. Slotted, synchronous operation is assumed: fixed-size data units, named cells, are temporarily buffered at input ports according to an input-queueing (IQ) architecture, waiting for the availability of the output line. To balance electronic and optical complexity, each input port is equipped with a single tunable transmitter (TT). Thus, at most one cell can be transmitted from each input in each time slot, using the proper wavelength to reach the chosen output. The adopted switch control (scheduling) algorithm must ensure that at most one cell is forwarded to each output at the same time to avoid output contention where buffering is not available. Current schedulers typically use virtual output queueing (VOQ) at inputs to achieve high throughput: Incoming cells at each input port are stored in $N$ separate first-in–first-out (FIFO) queues according to their destination. Since, at each output port, cells are received at a wavelength depending on the transmitting input port and, at each time slot, at most a single cell is received at each output, there is no need for tunable receivers at the outputs, and single wideband receivers suffice. Yet, since each output port receives (over time) cells from different inputs, burst-mode operation—hence, wideband burst-mode receivers (WBMR)—is necessary.

Note that in this setup, the AWG is largely underutilized, as only at most $N$ out of the possible $N^2$ input/output connections are used at a given time. This underutilization is a direct consequence of the single transceiver architecture assumed in the paper, a constraint introduced to reduce the electronic complexity of input/output line-cards. Indeed, given the increasingly high transmission speeds, it is becoming overly difficult and expensive to support more than a single data flow at each input/output port, especially when considering memory access speed. Different AWG-based optical switching architectures have been studied in the technical literature (e.g., [6]–[8]); although some of these architectures achieve better performance, they require a significantly higher electronic complexity and are not further considered in this paper.

Commercial AWGs provide uniform transfer functions and extinction ratios among adjacent channels in the order of 30–40 dB. These physical-layer characteristics are largely sufficient for multiplexers and demultiplexers, successfully used in commercial WDM systems. However, $N \times N$ AWGs are known to be affected by significant coherent crosstalk [9]: When the same wavelength is used at different inputs, the corresponding output ports receive an in-band interference from signals aimed at other output ports due to the finite extinction ratio. If the same wavelength is used at all AWG inputs, the maximum number of AWG ports $N$ is severely limited: Fig. 3 (from [10]) shows the power penalty $L_{\text{awg}}$ in dB for a typical AWG as a function of the number of ports $N$. While insertion losses (IL), nonuniformities of the transfer function (U), and polarization-dependent losses (PDL) are almost independent of the port count, in-band (or coherent) and out-of-band crosstalk (IX+OX) increase sharply and limit the port count to around 15. This would limit the possibility of using AWGs in large-size switching fabric unless the coherent crosstalk is controlled.

We remark that several proposals in the literature assume very large AWG port counts, even if this turns out to be unfeasible without countermeasuring the above-described crosstalk impairments. These large crosstalks can only marginally be reduced by improving the physical-layer behavior of the device [11]. One possibility to overcome this impairment is to exploit homologous wavelengths in several FSRs (as proposed in some studies, e.g., [12]). However, this increases the operational bandwidth of the system (possibly preventing the utilization of optical amplifiers); furthermore, the behavior of the device outside the principal FSR often degrades rapidly. The alternative approach pursued in this paper, and initially explored in [13] and [14], is to prevent coherent crosstalk by controlling AWG-based slotted switches with scheduling decisions that avoid simultaneously using the same wavelengths at too many different inputs. We show in this paper that this approach can be very effective, limiting the number of inputs using the same wavelength at the same time to few units without reducing switching fabric performance. Our new results enable the full exploitation of $N \times N$ AWGs for packet-switching applications. This novel approach is an important example of the benefits of devising control algorithms that are aware of the physical characteristics of the controlled optical devices.

In this paper, the crosstalk-constrained scheduling problem is formally formulated in Section II. We introduce the notion of $k$-legal permutations: A scheduling decision is $k$-legal if each wavelength is reused at most $k$ times in a given time-slot. By choosing a proper value for $k$, crosstalk figures can be controlled and large port counts become feasible. In Section III, we discuss basic properties of the permitted switch configurations. For 1-legal permutations, we show that a difference exists between odd and even values of the number of input and output ports $N$. In Section IV, we elaborate on the performance obtainable with single-stage AWG switching architectures, showing that uniform traffic patterns can be scheduled using 1-legal permutations with no speedup for odd $N$ and with a small speedup with even $N$. In Section V, we show that general traffic patterns can instead be scheduled with 1-legal permutations using two-stage switches using the same small speedup, VOQs between the two switching stages, and cell resequencing at outputs. We also show that 2-legal permutations permit to avoid intermediate VOQs and resequencing problems for small values of $N$. Finally, as completely original contributions, in Sections VI and VII we formally prove that a 2-stage AWG-based switch can be configured with pairs of 4-legal permutations (or 3-legal permutations in case $N$ is a prime number) with no buffering between the two stages, proposing a switch control scheme with quadratic complexity in the number of ports.

## II. Problem Statement

The considered IQ $N \times N$ switch handles fixed-size cells that arrive at input ports and leave output ports in a time-slotted manner: All the switch external lines are assumed to be synchronized. In each time slot, the scheduler defines $S$ *scheduling decisions*, where $S$ is the switch speedup. Due to the single-transceiver-per-port assumption, at each scheduling decision, at most one cell can be sent from each input port and at most one cell can be sent to each output port. Thus, each scheduling decision is a *permutation* (or a partial permutation) of port indices. We denote these permutations by vectors $\pi = [\pi[0], \pi[1], \ldots, \pi[N-1]]$, where $\pi[i]$ is the output port index to which input $i$ forwards a cell. Clearly, each output port index can appear at most once in $\pi$. If the scheduling decision creates a partial permutation, some entries in $\pi$ are "don't care." We denote by $I$ the unit permutation: $I = [0, 1, \ldots, N-1]$.

The traffic to be forwarded by the switch can be described by a traffic matrix $T = [t_{i,j}]$, where $t_{i,j}$ is the number of cells (or, alternatively, the number of cells per time unit, or the number of cells per time frame) that must be forwarded from input $i$ to output $j$. Using a matrix notation, an input/output permutation could also be described by an $N \times N$ permutation matrix, i.e., a 0–1 matrix $P = p[ij]$, where rows (columns) represent inputs (outputs), and $p[ij] = 1$ if and only if input $i$ is connected to output $j$. In a permutation matrix, at most a single "1" is present in each column and in each row.

Since we deal with an AWG passive router, cell forwarding through the switching fabric is done by assigning to each cell a wavelength out of a predetermined set of $N$ wavelengths $\Lambda = \{\lambda_0, \ldots, \lambda_{N-1}\}$, according to the following rule.

> A cell sent from input port $i$ with wavelength $\lambda_k \in \Lambda$ is forwarded to output port $i + k \bmod N$.[1]

[1]In the remainder of the paper, the $\bmod N$ operator, denoting the remainder of the division by $N$, may be omitted to improve readability.

Given a permutation $\pi$, we call $\lambda(\pi)$ the *wavelength assignment* of $\pi$; that is, the vector of indices of the wavelengths that are needed at the input ports to realize permutation $\pi$. Note that, with our wavelength assignment rule, the wavelength used to reach output $j$ from input $i$ is $\lambda_{(j-i) \bmod N}$. Hence, $\lambda(\pi) = (\pi - I) \bmod N$ when $I$ is the identity permutation. As mentioned in Section I, other wavelength assignments are possible, depending on the design of the AWG device. These different wavelength behaviors can in some cases (for example, if output $(i-k) \bmod N$ is reached from input $i$ using $\lambda_k$) be modeled by relabeling wavelengths and ports in our formalization, so the properties outlined in the sequel hold for several AWG wavelength assignments.

Recall that performance degradation due to the coherent crosstalk arises when several different input ports in an AWG-based switch simultaneously use the same wavelength to send cells to different output ports. The impairments due to coherent crosstalk increase as the number of input ports using the same wavelength increases, up to the point in which the switch operation becomes impossible; this limit was estimated to be around 15 in [10]. We focus on avoiding such effects by restricting the switch scheduler to use only a certain type of permutation.

*Definition 1:* A permutation $\pi$ is $k$-legal if, in the vector $\lambda(\pi) = \pi - I$, no index appears more than $k$ times.

$k$ represents the maximum number of times the same wavelength is used at different input ports in a given scheduling decision. Our goal is to build a switch that can handle the incoming traffic using only $k$-legal permutations, with the smallest possible value of $k$, to minimize crosstalk.

## III. Properties of $k$-Legal Permutations

We start by investigating the properties of $k$-legal permutations. Definition 1 immediately implies that a permutation $\pi$ is 1-legal if and only if its wavelength assignment $\lambda(\pi)$ is also a permutation. Note that any $k$-legal permutation is also $m$-legal, for any $m \in \{k, \ldots, N\}$. Furthermore, we have the following lemma.

*Lemma 3.1:* Let $\chi \in \{0, \ldots, N-1\}^N$, such that all its elements are $x$. If a permutation $\pi$ is $k$-legal, then the permutation $\pi + \chi$ is also $k$-legal.

*Proof:* Assume toward a contradiction that $\pi + \chi$ is not $k$-legal, so in $\psi = \pi + \chi - I$ there exist $k+1$ indices $i_1, \ldots, i_{k+1}$ such that $\psi[i_1] = \cdots = \psi[i_{k+1}]$. This implies that $\pi[i_1] + \chi[i_1] - i_1 = \cdots = \pi[i_{k+1}] + \chi[i_{k+1}] - i_{k+1}$. Since $\chi[i_1] = \cdots = \chi[i_{k+1}] = x$, it follows that $\pi[i_1] - i_1 = \cdots = \pi[i_{k+1}] - i_{k+1}$, implying that $\pi - I$ has $k+1$ identical elements. This contradicts the assumption that $\pi$ is $k$-legal, and the claim follows. ∎

Next, we show how to build 1-legal permutations for odd values of $N$.

*Lemma 3.2:* If $N$ is odd, then there exists a 1-legal permutation $\pi_{\text{odd}}$ of $\{0, \ldots, N-1\}$.

*Proof:* Let $\pi_{\text{odd}}$ be the following permutation:

$$\pi_{\text{odd}}[i] = 2i \mod N \quad i \in \{0, \ldots, N-1\}.$$

If $i \neq j$, then $2i - 2j$ is even and does not equal 0. Since $N$ is odd, this implies that $2i - 2j \neq 0 \bmod N$, implying that $\pi_{\text{odd}}[i] \neq \pi_{\text{odd}}[j]$. Hence, $\pi_{\text{odd}}$ is a permutation.

The wavelength assignment of $\pi_{\text{odd}}$ is $\pi_{\text{odd}} - I = 2I - I = I$ which is clearly a permutation. Thus, $\pi_{\text{odd}}$ is 1-legal. ∎

Note that similar arguments prove that the *anti-diagonal permutation* $-I$, whose wavelength assignment is $-2I$, is also a 1-legal permutation.

The following lemma deals with *even* values of $N$. The proof is omitted because the result was independently proven in [13].

*Lemma 3.3:* If $N$ is *even*, there is no 1-legal permutation of $\{0,\ldots,N-1\}$.

We next deal with 2-legal permutations.

*Lemma 3.4:* For every $N$, there is a 2-legal permutation of $\{0,\ldots,N-1\}$.

*Proof:* Since every 1-legal permutation is also a 2-legal permutation, the claim follows immediately by Lemma 3.2 for odd values of $N$.

Assume that $N$ is even, and consider the following assignment permutation:

$$\pi_{\text{even}}[i] = \begin{cases} 2i \bmod N, & i < N/2 \\ 2i+1 \bmod N, & N/2 \le i \le N-1. \end{cases}$$

Clearly, $\pi_{\text{even}}$ is a permutation: Its first half covers all the even output ports, and its second half covers all the odd output ports. We now compute the wavelength assignment of $\pi_{\text{even}}$

$$\lambda(\pi_{\text{even}})[i] = \begin{cases} i \bmod N, & i < N/2 \\ i+1 \bmod N, & N/2 \le i \le N-1. \end{cases}$$

Clearly, each input port except input port 0 and $N-1$ has a different wavelength assignment, while for input ports 0 and $N-1$ we get $\lambda(\pi_{\text{even}})[0] = \lambda(\pi_{\text{even}})[N-1] = 0$, implying that $\pi_{\text{even}}$ is a 2-legal permutation. Hence, we have a 2-legal permutation with only one wavelength repetition. ■

## IV. Single-Stage AWG-Based Switches

We first consider a single-stage AWG switch. We denote by $\rho_i^I$ and $\rho_j^O$ the normalized load (i.e., the average traffic divided by the port speed) at input $i$ and output $j$, respectively, $i,j \in [0,\ldots,N-1]$. A traffic pattern is admissible if, for each $i$ and each $j$, $\rho_i^I \le 1$ and $\rho_j^O \le 1$. We start by investigating the speedup required to realize any adversarial traffic pattern.

### A. Worst-Case Traffic

Due to the AWG switch crosstalk impairments, the most difficult traffic to handle is "generalized diagonal" traffic, in which all cells from input port $i$ are directed to output port $i+x \bmod N$ ($x \in \{0,\ldots,N-1\}$), with the same value of $x$ for each input port: All inputs are forced to use the same wavelength to reach the proper output. As a consequence, under generalized diagonal traffic, if we are restricted to $k$-legal permutations, at most $k$ cells can be forwarded in any scheduling decision, implying that the required speedup is $S = N/k$. Since $k$ should be a small constant (less than 16, as shown in [10], but possibly even smaller, in case of all-optical cascades of switching stages) to avoid coherent crosstalk impairments, this implies that single-stage AWG switches with large port counts require prohibitive speedups to cope with such an adversarial traffic. In contrast, a crossbar switch, in which there is no restriction on the permutation used, can schedule generalized diagonal traffic with speedup $S = 1$.

### B. Uniform Traffic

Let us now focus on the classical uniform traffic pattern, where the uniform traffic matrix $T$ is an $N \times N$ matrix whose elements are all $\frac{\rho}{N}$, where $\rho \le 1$ is the normalized load at each input and output ($\rho = \rho_i^I = \rho_j^O, \forall i,j$). It is well known that an $N \times N$ uniform traffic matrix can be scheduled using a fixed *time-division multiplexing (TDM)* approach, in which, during a frame of $N$ time slots, each input is in turn connected to the different $N$ outputs. This can be interpreted as the decomposition of matrix $T$ in a set of $N$ "covering" permutation matrices. If we ignore the constraint of using only $k$-legal permutations, a possible decomposition of matrix $T$ is achieved through a set of $N$ generalized diagonal (switching) matrices $\Pi_{\text{TDM}} = \{\pi_0, \pi_1, \ldots, \pi_{N-1}\}$, where $\pi_i = I + \chi_i \bmod N$ and $\chi_i$ is a vector in which all elements are $i$. For example, in the case $N = 4$, we have the following decomposition:

$$\frac{\rho}{N}\begin{bmatrix}1&1&1&1\\1&1&1&1\\1&1&1&1\\1&1&1&1\end{bmatrix} = \frac{\rho}{N}\left(\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix} + \begin{bmatrix}0&1&0&0\\0&0&1&0\\0&0&0&1\\1&0&0&0\end{bmatrix} + \begin{bmatrix}0&0&1&0\\0&0&0&1\\1&0&0&0\\0&1&0&0\end{bmatrix} + \begin{bmatrix}0&0&0&1\\1&0&0&0\\0&1&0&0\\0&0&1&0\end{bmatrix}\right).$$

We look for a decomposition of a uniform traffic matrix to $k$-legal permutations, and we wish to determine the minimum number of $k$-legal subpermutations needed to decompose a uniform traffic matrix.

*1) Scheduling Uniform Traffic Using 1-Legal Permutations When N Is Odd:* If $N$ is odd, we use the following covering sequence of $N$ 1-legal permutations, in which each input/output pair is connected exactly once: $\Pi = \{\pi_{\text{odd}} + \chi_x \mid 0 \le x \le N-1\}$, where $\pi_{\text{odd}}$ is the permutation defined in Lemma 3.2 and $\chi_x$ is the $N$-vector whose elements are all equal to $x$. Being that $N$ is odd, by Lemmas 3.2 and 3.1, all permutations in $\Pi$ are 1-legal. Furthermore, each input port $i$ is connected to output port $j$ if and only if the permutation $\pi_{\text{odd}} + \chi_x \in \Pi$ with $x = (j-2i) \bmod N$ is used.

*2) Scheduling Uniform Traffic Using 1-Legal Permutations When N Is Even:* Recall that by Lemma 3.3, no 1-legal permutation exists when $N$ is even. Thus, we cannot apply the same strategy as with odd values of $N$.

A straightforward way to get around this problem is to add another port to the switch making its port count odd. The cost of adding ports relative to the entire switch is called the *spatial speedup* of the switch, and in this case, it is $1 + \frac{1}{N}$ (a single additional port should be added for $N$ existing ports). Note that the extra port will not be active in sending/receiving cells, hence it does not add complexity in term of transceiver hardware.

*3) Scheduling Uniform Traffic Using 2-Legal Permutations:* When $N$ is odd, we already found in the previous section a scheduling for uniform traffic providing 100% throughput with no speedup, i.e., relying on 1-legal permutations. When $N$ is even, and we permit 2-legal permutations, we can use the set $\Pi = \{\pi_{\text{even}} + \chi_x \mid 0 \le x \le N-1\}$, where $\pi_{\text{even}}$ is the permutation defined in Lemma 3.4 and $\chi_x$ is an $N$-vector whose all elements are $x$. The correctness of the construction is identical to the one described for 1-legal permutations.

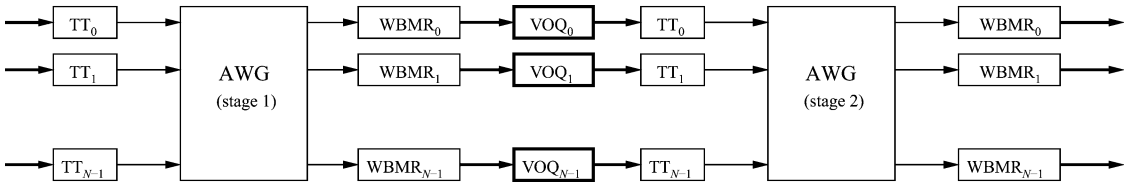


Fig. 4. Two-stage AWG-based switch. Electronic paths and components are highlighted with thick lines.

In summary, in a single-stage AWG-based switch, uniform traffic can be scheduled using 1-legal permutations with no speedup when $N$ is odd. When $N$ is even, a spatial speedup of $1 + 1/N$ is required for 1-legal permutations. Alternatively, 2-legal permutations with no time or space speedup can be used. However, the worst-case traffic scenario implies that a prohibitive time speedup of $N/k$ is required to schedule all admissible traffic patterns under the $k$-legal constraint. To cope with this issue, in the rest of the paper, we present and study two-stage architectures that are able to efficiently deal with any admissible traffic.

## V. Two-Stage Architectures

In this section, we consider an AWG-based two-stage switch architecture. We adapt the two-stage Load-Balanced Switch, introduced in [15] and briefly recalled below, and schedule the AWGs switching stages to operate with $k$-legal permutations.

### A. Two-Stage Load-Balanced Switch

The Load-Balanced Switch (Fig. 4) consists of two switching stages: The first stage performs load balancing of the incoming traffic, while the second stage performs the actual switching of cells to their destination. The basic idea is to transform any generic traffic pattern at the switch input into a uniform traffic at the output of the first stage, hence at the input of the second stage. To achieve this, cells arriving at one input in the first stage are forwarded in turn to all outputs of the first switching stage, regardless of the final output port. This permits to evenly distribute the considered input load to all first-stage outputs. Since this traffic "spreading" operation is performed at all inputs, all first-stage outputs receive, on average, the same amount of traffic, under mild input traffic stationarity conditions. The traffic at the input of the second stage is therefore uniform: same load at all ports, and equal probability for any input/output pair. This uniform traffic can be easily forwarded in a load-balanced switch by a fixed TDM switch schedule in the second stage, providing 100% throughput if the traffic is stationary and weakly mixing,[2] excellent delay performance, and efficient buffer usage.

It is important to notice that no cell buffering is required at inputs, as arriving cells are immediately forwarded. VOQ buffering is instead required between the two stages (cells destined for different output ports are stored in separate FIFO buffers), in which cell queues may build up in case of congestion.

The load-balancing operation with VOQ buffering between the two switching stages has the drawback of out-of-sequence cell delivery. To avoid this, either resequencing modules must be introduced at the outputs of the second stage, or more complex queuing structures and policies must be used between the two stages [16]–[18]. Both solutions increase complexity and must be implemented in the electronic domain.

The scheduling in the two stages can be fully distributed—i.e., based on local decisions at each input port—without any coordination among different ports, apart from a switch-wide slot synchronization, provided that traffic is weakly mixing, to avoid adversarial patterns that would impair the load-balancing effect when the first stage is operated in fixed TDM. It can be easily understood that, for both switch stages, the scheduling translates into a periodic sequence of $N$ permutations, such that each input/output pair is connected exactly once in each period. This is equivalent to scheduling uniform traffic matrices in both stages; hence, the scheduling for the two-stage $N \times N$ load-balanced switch must cyclically run over a set of $N$ covering permutations for a uniform (i.e., comprising all equal values) traffic matrix. While these $N$ covering permutations can be found in several ways, we are interested, for the AWG-based switch, in a set of $N$ $k$-legal permutations, which can be obtained as described in Section IV-B1.

Note that the two $N \times N$ switching stages can be interpreted as a twofold speedup realized in the space domain: Up to $2N$ cells are simultaneously switched in every time slot.

*Implications to AWG-Based Switches:* The results from the previous sections imply that any traffic pattern in the two-stage AWG-based load-balanced switch can be scheduled with no speedup in each stage when the number of ports is odd. A spatial speedup equal to $1 + 1/N$ is needed in the case of even number of ports. If considering the two-stage architecture as a switch spatial speedup, a generic traffic matrix can be scheduled with a speedup of 2 (respectively, $2 + 2/N$) for AWGs with odd (respectively, even) number of ports. Since the spatial speedup avoids any speed increase in components and transmission lines, this architecture is well suited for the optical domain, keeping the electronic speed in the feasible domain of today's technology.

### B. Avoiding Buffers in the Middle Stage

A promising approach to circumvent the need for buffering in the middle stage and resequencing at the egress is to control the AWG in both stages simultaneously so that their combination will produce the desired permutation. In this setting, suppose we have an *oracle crossbar scheduler* that produces a sequence of permutations; our goal is to realize each of these permutations $\pi$ using two $k$-legal permutations $\pi_1$ and $\pi_2$ such that $\pi = \pi_1 \circ \pi_2$, where $\circ$ denotes function composition. We call the pair of permutations $\langle \pi_1, \pi_2 \rangle$ a *$k$-legal decomposition of* $\pi$. Since the oracle may produce any permutation of $\{0, \ldots, N-1\}$, our algorithm must be able to decompose all permutations. Fig. 5

[2]A stochastic sequence $\{\underline{a}(t), t \geq 1\}$ is weakly mixing [15] if for all $A, B \in \mathbb{R}^{(\mathbb{N}\times\mathbb{N})^\infty}$, $\lim_{t\to\infty} 1/t \sum_{s=0}^{t-1} |\Pr(\theta_s \underline{a} \in A, \underline{a} \in B) - \Pr(\underline{a} \in A)\Pr(\underline{a} \in B)| = 0$, where $\theta_s \underline{a}$ is the sequence $\underline{a}$ shifted by $s$ time slots: $\theta_s \underline{a} = \{\underline{a}(t+s), t \geq 1\}$. Note that each weakly mixing stochastic sequence is also ergodic.

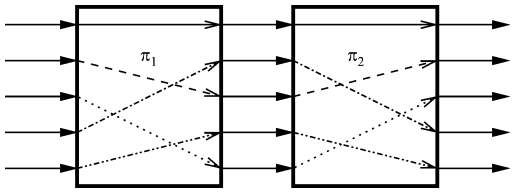


Fig. 5. Illustration of a 1-legal decomposition of the identity permutation in a $5 \times 5$ switch using permutations $\pi_1 = [0,\ 2,\ 4,\ 1,\ 3]$ and $\pi_2 = [0,\ 3,\ 1,\ 4,\ 2]$. Both $\pi_1$ and $\pi_2$ are 1-legal permutations.

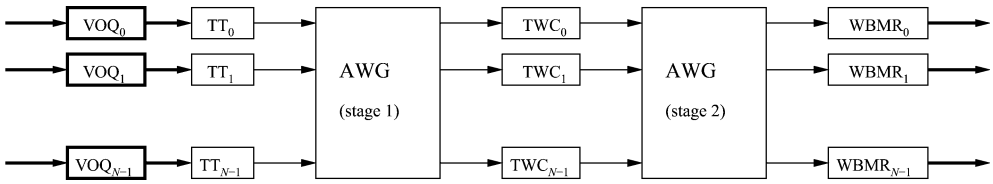


Fig. 6. Two-stage AWG-based switch with no internal buffering. Electronic paths and components are highlighted with thick lines.

depicts a 1-legal decomposition of the identity permutation in a $5 \times 5$ switch.

The resulting architecture is depicted in Fig. 6, where tunable wavelength converters (TWCs) (or equivalently, a WBMR followed, with no cell buffering stage, by a TT) are needed between the two stages to create the proper permutation. With this solution, VOQ buffering and O/E/O conversions are no longer required between the two stages, and cell out-of-sequence delivery is eliminated. VOQs are however needed in front of the first-stage, similarly to the classical IQ switch architecture depicted in Fig. 2.

Besides eliminating the need for buffering in the middle stage, our decomposition approach has other significant advantages over the two-stage Load-Balanced Switch approach. First, while the load-balanced switch provides 100% throughput on a wide range of traffic patterns, there are still pathological traffic patterns that make its throughput arbitrarily small [19, Ch. 1.3.3]. Moreover, load-balanced switches require a full switch reconfigurations at each scheduling decision; these reconfigurations may become infeasible as the line rates grow. Lastly, the two-stage load-balanced switch is only aiming at providing 100% throughput; however, there is no bound on other important performance measures such as latency, smoothness (delay jitter), or fairness. These measures are crucial to provide the stringent QoS required by contemporary applications, and therefore a thorough research was done in the last decade to devise scheduling algorithms that perform better under these metrics (see, for example, [20] for a comprehensive survey).

Our decomposition algorithms offer a modular black-box approach in which any existing (or future) scheduling algorithm can be *converted* to be a crosstalk-preventing algorithm. We indeed model the scheduling algorithm as an oracle whose output is its switching decisions (that is, a sequence of permutations). Our algorithms get these permutations (one by one) as an input and produce the necessary $k$-legal permutations needed for crosstalk-preventing scheduling (see illustration in Fig. 7). It is thus suited to operate with *any* input-queuing scheduling algorithm.

For example, to reduce the number of reconfigurations of the AWG devices one can use a scheduling algorithm that takes into account the reconfiguration delays and aims at minimizing

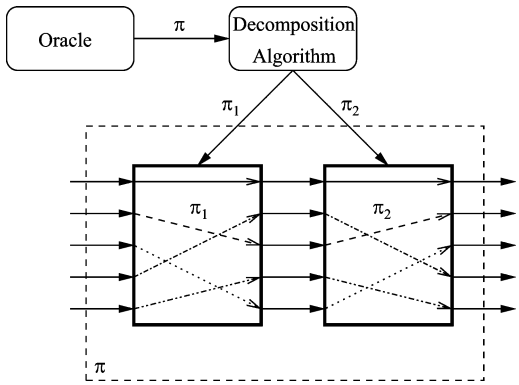


Fig. 7. Exploiting an oracle to implement a decomposition.

the total delay (e.g., [21]–[23]). Since the changes in permutations under such schedulers are not frequent, the number of needed decompositions decreases accordingly, thus facilitating the computation demands of our decomposition algorithms.

We now discuss for which values of $k$ a $k$-legal decomposition exists for all permutations.

*Impossibility of Using 1-Legal Permutations:* Recall that by Theorem 3.3, no 1-legal permutation exists for even $N$; this implies that no 1-legal decomposition exists. However, we are able to prove, by a counterexample, that no 1-legal decomposition algorithm exists for any $N$, regardless of its parity.

*Theorem 5.1:* For any $N$, the following permutation $\pi = [0, 1, 2, \ldots, N-3, N-1, N-2]$ has no 1-legal decomposition.

*Proof:* Assume that there is a 1-legal decomposition of $\pi$ into two 1-legal permutations $\pi_1$ and $\pi_2$. Let $\lambda_1 = \lambda(\pi_1)$ and $\lambda_2 = \lambda(\pi_2)$ be the wavelength assignments of $\pi_1$ and $\pi_2$, respectively. Since $\pi_1$ and $\pi_2$ are 1-legal, $\lambda_1$ and $\lambda_2$ are permutations. Since for every $i \leq N-3$, $\pi[i] = i$, then $\lambda_2[\pi_1[i]] = -\lambda_1[i]$. Thus, in the composite permutation $\pi[i] = \pi_2[\pi_1[i]] = \pi_1[i] - \lambda_1[i] = i + \lambda_1[i] - \lambda_1[i] = i$, as required.

Since $\lambda_2$ is a permutation, the remaining elements $\lambda_2[\pi_1[N-2]]$ and $\lambda_2[\pi_1[N-1]]$ must use the remaining values $-\lambda_1[N-2]$ and $-\lambda_1[N-1]$. If $\lambda_2[\pi_1[N-2]] = -\lambda_1[N-2]$, this results in $\pi[N-2] = \pi_2[\pi_1[N-2]] = N-2 \neq N-1$. Thus, we should have $\lambda_2[\pi_1[N-2]] = -\lambda_1[N-1]$ and $\lambda_2[\pi_1[N-1]] = -\lambda_1[N-2]$, implying that $\pi[N-2] = \pi_2[\pi_1[N-2]] =$

TABLE I
NUMBER OF FIRST-STAGE PERMUTATIONS NEEDED TO PROVIDE A 2-LEGAL DECOMPOSITION

| Number of ports $N$ | Number of permutations | Number of 2-legal permutations | Size of $\Pi_1$ (upper bound) |
|---|---|---|---|
| 4 | 24 | 20 | 2 |
| 5 | 120 | 65 | 4 |
| 6 | 720 | 396 | 8 |
| 7 | 5040 | 2338 | 12 |
| 8 | 40320 | 16912 | 29 |
| 9 | 362880 | 132759 | 53 |
| 10 | 3628800 | 1183200 | 107 |
| 11 | 39916800 | 11531641 | 237 |
| 12 | 479001600 | 123019776 | 543 |

$N-2+\lambda_1[N-2]-\lambda_1[N-1]$, which in turn implies that $\lambda_1[N-2]-\lambda_1[N-1]=1$ since $\pi[N-2]=N-1$. This yields that $\pi_1[N-2]=N-2+\lambda_1[N-2]=N-1+\lambda_1[N-1]=\pi_1[N-1]$, which contradicts that $\pi_1$ is a permutation, and the claim follows. ■

*Decomposition Using 2-Legal Permutations:* We continue by investigating 2-legal decomposition algorithms. First, for $N=3$, Theorem 5.1 and the fact that each permutation of $\{0,1,2\}$ is either 1-legal or 3-legal immediately implies the following.

*Corollary 5.2:* There is no 2-legal decomposition algorithm for $N=3$.

For $N=4,\ldots,12$, we verified by exhaustive search that any permutation $\pi$ of $\{0,\ldots,N-1\}$ can be 2-legally decomposed. Hence, we can state the following theorem.

*Theorem 5.3:* There exists a 2-legal decomposition for all permutations with $N\in\{4,\ldots,12\}$.

Furthermore, it is important to notice that, given a permutation $\pi$, by choosing a 2-legal first permutation $\pi_1$, one only needs to verify that $\pi_2=\pi_1^{-1}\circ\pi$ is 2-legal to decide whether the decomposition is legal. Our experiments, reported in Table I, show that, to decompose any permutation $\pi$, it is sufficient to choose $\pi_1$ from a small set $\Pi_1$ of 2-legal permutations. Since $\Pi_1$ can be precomputed and programmed directly in the scheduler of the AWG, it implies that it is feasible to implement this scheduler for these values of $N$. Table I shows the size of $\Pi_1$ for different values of $N$.

We were not able to find an algorithm to compute a 2-legal decomposition, nor we were able to prove that 2-legal decompositions exist for every $N$. We thus leave these two issues as open research problems and formulate the following conjecture, partially supported by our exhaustive searches for small values of $N$:

*Conjecture 5.1:* There exists a 2-legal decomposition for all permutations with $N\geq 4$.

The main contributions of this paper regard 4-legal and 3-legal decompositions, for which we provide constructive rules and complexity assessment in the next sections.

## VI. 4-LEGAL DECOMPOSITIONS

In this section we describe a decomposition algorithm that, for any $N$ and any permutation $\pi$ of $\{0,\ldots,N-1\}$, finds a 4-legal decomposition of $\pi$.

We start by describing a method to "*correct*" a nonlegal decomposition. This method is general for every $k$-legal decomposition, and it constitutes a single iteration of our algorithm. The procedure (see Algorithm 1 for a pseudocode description)

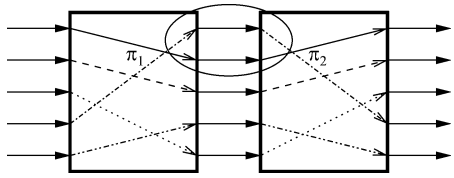


Fig. 8. Illustration of a (0, 1)-transposition of the decomposition of the identity permutation depicted in Fig. 5. The resulting decomposition uses permutations $\pi_1=[1,2,4,0,3]$ and $\pi_2=[3,0,1,4,2]$.

uses *transpositions* of middle-stage ports, which are defined as follows.

*Definition 2:* Given a decomposition $\langle\pi_1,\pi_2\rangle$ of $\pi$, the $(i,j)$-*transposition of* $\langle\pi_1,\pi_2\rangle$ is a decomposition of $\pi$ into permutations $\pi_1',\pi_2'$ such that

$$\pi_1'[\ell]=\begin{cases} j, & \ell=\pi_1^{-1}[i]\\ i, & \ell=\pi_1^{-1}[j]\\ \pi_1[\ell], & \text{otherwise.}\end{cases}$$

$$\pi_2'[\ell]=\begin{cases} \pi_2[i], & \ell=j\\ \pi_2[j], & \ell=i\\ \pi_2[\ell], & \text{otherwise.}\end{cases}$$

It is easy to verify that $\pi_1'$ and $\pi_2'$ are still permutations and that $\pi_1'\circ\pi_2'=\pi$. Fig. 8 illustrates a (0, 1)-transposition of the decomposition described in Fig. 5.

**Algorithm 1** A Correction Procedure of a Decomposition Using a Single Transposition.

$\langle\pi_1,\pi_2\rangle$ **procedure** CORRECT $(\langle\pi_1,\pi_2\rangle,k)$
  $\lambda$ is an (arbitrary) wavelength used in $\pi_2$ more than $k$ times
  $i$ is an (arbitrary) input port of $\pi_2$ using wavelength $\lambda$
  $\Lambda_1$ is the set of wavelengths used in $\pi_1$ at least $k$ times
  $\Lambda_1'$ is the set of wavelengths used in $\pi_1$ exactly $k-1$ times
  $\Lambda_2$ is the set of wavelengths used in $\pi_2$ at least $k$ times
  $\Lambda_2'$ is the set of wavelengths used in $\pi_1$ exactly $k-1$ times
  ▷ Recall that all calculations are modulo $N$
  $A_1\leftarrow\{\ell \mid (\ell-\pi_1^{-1}[i])\in\Lambda_1\}$
  $A_2\leftarrow\{\ell \mid (i-\pi_1^{-1}[\ell])\in\Lambda_1\}$
  $A_3\leftarrow\{\ell \mid (\pi_2[i]-\ell)\in\Lambda_2\}$
  $A_4\leftarrow\{\ell \mid (\pi_2[\ell]-i)\in\Lambda_2\}$
  $A_5\leftarrow\{\ell \mid \ell+\pi_1^{-1}[\ell]=i+\pi_1^{-1}[i], i-\pi_1^{-1}[\ell]\in\Lambda_1'\}$
  $A_6\leftarrow\{\ell \mid \ell+\pi_2[\ell]=i+\pi_2[i], \pi_2[i]-\ell\in\Lambda_2'\}$
  Find a middle-stage port $j$ such that $j\notin A_1\cup A_2\cup A_3\cup A_4\cup A_5\cup A_6$
  **if** no such $j$ exists **then**
    **return** $\perp$ ▷ The algorithm fails
  **else**
    $\langle\pi_1',\pi_2'\rangle\leftarrow(i,j)$ transposition of $\langle\pi_1,\pi_2\rangle$
    **return** $\langle\pi_1',\pi_2'\rangle$
  **end if**
**end procedure**

The process of correcting a permutation is captured by a potential function that counts the minimal number of input ports that should be corrected to make permutation $\pi$ $k$-legal and is formally defined as follows.

*Definition 3:* Let $\#_{\lambda(\pi)}[\lambda']$ be the number of appearances of wavelength $\lambda' \in \Lambda$ in $\lambda(\pi)$. The *$k$-potential* of $\pi$, denoted $pot_k(\pi)$, is $\sum_{\lambda' \in \Lambda} \max\{0, \#_{\lambda(\pi)}[\lambda'] - k\}$.

Clearly, the highest possible potential value is $N - k$, given to the identity permutation $I$. On the other hand, each $k$-legal permutation has a $k$-potential 0.

The correction is done by swapping a middle-stage port $i$, whose wavelength $\lambda(\pi_2)[i]$ is used more than $k$ times by $\pi_2$ (we call such a port a *violating* port), with another middle-stage port $j$. Assuming $\pi_1$ to be $k$-legal, $j$ is chosen under the following constraints.

1) In the first stage, the wavelength assigned to reach $j$ from $\pi_1^{-1}[i]$ keeps $\pi_1$ $k$-legal. Set $A_1$ (Line 8) contains all the choices of $j$ violating this constraint.
2) In the first stage, the wavelength assigned to reach $i$ from $\pi_1^{-1}[j]$ keeps $\pi_1$ $k$-legal. Set $A_2$ (Line 9) contains all the choices of $j$ violating this constraint.
3) In the second stage, the wavelength assigned to reach $\pi_2[i]$ from $j$ is not a *critical* wavelength (that is, it is not already used $k$ times or more). Set $A_3$ (Line 10) contains all the choices of $j$ violating this constraint.
4) In the second stage, the wavelength assigned to reach $\pi_2[j]$ from $i$ is not a critical wavelength. Set $A_4$ (Line 11) contains all the choices of $j$ violating this constraint.
5) Sets $A_5$ and $A_6$ (Lines 12-13) deal with the delicate situation in which the wavelength assigned to $i$ is the same as the wavelength assigned to $j$. In such a case, violating the $k$-legality of $\pi_1$ can happen even if the wavelength was used only $k-1$ times in $\pi_1$. Similarly, in such a situation a wavelength is critical in $\pi_2$ even if it is used only $k-1$ times.

These constraints yield the following lemma (see proof in Appendix A).

*Lemma 6.1:* A nonfailed execution of Algorithm 1 with a first-stage $k$-legal permutation $\pi_1$ and a second-stage permutation $\pi_2$ such that $pot_k(\pi_2) > 0$ yields a $k$-legal first-stage permutation and a second-stage permutation with strictly smaller potential.

We proceed now by describing the 4-legal decomposition algorithm that is based on this correction procedure. Algorithm 2 starts with a 4-legal permutation $\pi_1$ (either $\pi_{\text{odd}}$, which is 1-legal, or $\pi_{\text{even}}$, which is 2-legal) and computes the required $\pi_2$ to realize permutation $\pi$. Clearly, $\pi_2$ in general is not 4-legal. Then, the algorithm corrects $\pi_2$, keeping the following invariant.

$\pi_1$ is a 4-legal permutation throughout the execution of the algorithm.

The following lemma will prove that in this algorithm, the invocation of CORRECT procedure in Line 8 never fails. Specifically, we will show that as long as $\pi_2$ is not a 4-legal permutation, it is possible to choose a middle-stage port that keeps all the constraints. Then, in Theorem 6.3 , we will show that the algorithm always stops after $N-4$ iterations, thus establishing its correctness and its time complexity.

*Lemma 6.2:* At each invocation of procedure CORRECT at every execution of Algorithm 2 (Line 8), there is a valid choice of a middle-stage port $j$ in Line 14.

**Algorithm 2** A 4-Legal Decomposition Algorithm That Decomposes a Permutation $\pi$ of $\{0, \ldots, N-1\}$ to Two Permutations $\pi_1$ and $\pi_2$

1: **if** $N$ is even **then**
2: $\quad \pi_1 \leftarrow \pi_{\text{even}}$
3: **else**
4: $\quad \pi_1 \leftarrow \pi_{\text{odd}}$
5: **end if**
6: $\pi_2 \leftarrow \pi \circ \pi_1^{-1}$
7: **while** $\pi_2$ is not 4-legal **do**
8: $\quad \langle \pi_1, \pi_2 \rangle \leftarrow$CORRECT$(\langle \pi_1, \pi_2 \rangle, 4)$
9: **end while**
10: $\langle \pi_1, \pi_2 \rangle$ is a 4-legal decomposition of $\pi$

*Proof:* We first observe that fixing a middle-stage port $i$ implies that $\pi_1^{-1}[i]$ and $\pi_2[i]$ are also fixed. Thus, each wavelength $\lambda \in \Lambda_1$ adds a single port to $A_1$ and a single port to $A_2$. Similarly, each wavelength $\lambda \in \Lambda_2$ adds a single port to $A_3$ and a single port to $A_4$. Hence, the size of the sets $A_1$ and $A_2$ is bounded by $|\Lambda_1|$ and the size of $A_3$ and $A_4$ is bounded by $|\Lambda_2|$. Similarly, the size of $A_5$ is bounded by $|\Lambda_1'|$, and the size of $A_6$ is bounded by $|\Lambda_2'|$.

We focus now to $|A_1 \cup A_2 \cup A_5|$, whose size is bounded by $|A_1|+|A_2|+|A_5| \leq 2|\Lambda_1|+|\Lambda_1'|$. In addition, $4|\Lambda_1|+3|\Lambda_1'| \leq N$, since by definition $\Lambda_1 \cap \Lambda_1' = \emptyset$, and each wavelength in $\Lambda_1$ must be used by at least four different ports, while each wavelength in $\Lambda_1'$ requires three ports. (For example, if $|\Lambda_1| > N/4$, it is not possible that each of the wavelengths of $\Lambda_1$ is used at least four times.) By solving this simple linear optimization problem, we get that the largest possible size of $|A_1 \cup A_2 \cup A_5|$ is $\frac{N}{2}$, obtained when $|\Lambda_1| = N/4$ and $|\Lambda_1'| = 0$. We continue by evaluating $|A_3 \cup A_4 \cup A_6|$, which has a similar analysis except for the following fact: Since port $i$ has a wavelength that is used more than four times in $\pi_2$ (Lines 2-3), $\pi_2[i] - i \in \Lambda_2$, yielding that $i \in A_3$ and $i \in A_4$, and, obviously, $i \in A_3 \cap A_4$. Therefore, $|A_3 \cup A_4 \cup A_6| \leq 2|\Lambda_2| + |\Lambda_2'| - 1 \leq N/2 - 1$.

This implies that $|\bigcup_{r=1}^{6} A_r| \leq |A_1 \cup A_2 \cup A_5| + |A_3 \cup A_4 \cup A_6| \leq N-1$. Therefore, there is always a middle-stage port $j$ such that $j \notin \bigcup_{r=1}^{6} A_r$ and Line 14 can be executed. Note that, since $i \in \bigcup_{r=1}^{6} A_r$, the chosen middle-stage port $j$ is not equal to $i$. ∎

*Theorem 6.3:* For any $N$ and any permutation $\pi$ of $\{0, \ldots, N-1\}$, Algorithm 2 finds a 4-legal decomposition of $\pi$ in $O(N^2)$ time.[3]

*Proof:* Consider an arbitrary permutation $\pi$. Let $\pi_1$ be the initial first-stage permutation, and $\pi_2$ be the initial second-stage permutation resulting in executing Line 2 in Algorithm 2. Denote by $x = pot_4(\pi_2) \leq N-4$. By Lemma 6.2, the invocations of CORRECT never fails. Thus, Lemma 6.1 implies that each iteration of the algorithm (Lines 7-9) decreases the potential of $\pi_2$ by at least 1, implying that after $y \leq x$ iterations, the potential of the second-stage permutation $\pi_2$ is 0. Therefore, $\pi_2$

[3]Under the standard RAM model [24], in which it takes a constant time to read or write a memory word, to compare two values and to perform standard arithmetic operations.

is 4-legal. In addition, since the CORRECT procedure never fails and the initial first-stage permutation $\pi_1$ is 4-legal, Lemma 6.1 also yields that first-stage permutation $\pi_1$ is also 4-legal after iteration $y$. Since: 1) the only changes in the permutations were made by transpositions (Line 18); and 2) after the execution of Line 6, $\pi_1 \circ \pi_2 = \pi$, Definition 2 implies that, when the algorithm ends, $\pi_1 \circ \pi_2$ is still $\pi$. Therefore, $\langle \pi_1, \pi_2 \rangle$ is a 4-legal decomposition of $\pi$.

We now turn to derive the time complexity of the algorithm by first showing that each invocation of Algorithm 1 has a linear time complexity. Since Algorithm 1 has a constant number of instructions (that is, there are no loops), it suffices to show that each instruction takes at most linear time.

We assume that each permutation is stored as an array of size $N$, and that it takes a constant time to read or to write an element of this array. Counting the number of appearances of each wavelength in a permutation can be done in a single pass (that is, in a linear time) on the permutation array: Let $\pi$ be the permutation array and $\#_{\Lambda(\pi)}$ be an array of $N$ integers (in which the results are stored); then, when reading $\pi[i]$, one should increment $\#_{\Lambda(\pi)}[\pi[i] - i \bmod N]$ by one (constant number of arithmetic operations). This implies that Lines 2, 6, and 7 of Algorithm 1 take linear time (single pass on $\#_{\Lambda(\pi_2)}$), and similarly Lines 4 and 5 (single pass on $\#_{\Lambda(\pi_1)}$); we assume that the results of Lines 4–7 are stored in an $N$-bit array in which bits are set if and only if the corresponding wavelength is in the set. Line 3 requires a single pass on $\pi_2$ (checking whether $\pi_2[i] - i \bmod N = \lambda$), thus also taking a linear time.

Notice also that for each permutation $\pi$, the inverse permutation $\pi^{-1}$ can be computed in a single pass on $\pi$ since $\pi^{-1}[\pi[i]] = i$. $A_1$, which we assume is stored also in an $N$-bit array, can be computed in linear time by considering $x = \pi_1^{-1}[i]$ and checking for each $\ell \in \{0, \ldots, N-1\}$ if $\Lambda_1[\ell - x] = 1$. Similarly, Lines 9–13 also can be computed in linear time. Line 14 may take at most $6\,\mathrm{N}$ comparison operations (by comparing, for each $j \in \{0, \ldots, N-1\}$ and $k \in \{1, \ldots, 6\}$, whether $A_k[j] = 0$ for all $k$). Line 18 is a simple swap operation that takes a constant time.

Thus, each iteration is linear in the number of ports, and the number of iterations is bounded by $N-4$, implying a time complexity of $O(N^2)$ (notice that Line 6 of Algorithm 2, which is executed only once, also has a linear time complexity). ■

Note that our algorithm can be used also for $k < 4$, but we could not prove its convergence in this case, except for the special case $k = 3$ and $N$ prime, which is discussed in the next section.

## VII. 3-Legal Decompositions

In this section, we assume that $N$ is a prime number and describe how to modify Algorithm 2 to achieve a 3-legal decomposition. For nonprime numbers, this implies that the device needs some spatial speedup, which can be bounded by 1.375. The maximum speedup is needed for $N = 8$, when three additional ports are required.[4]

The key insight behind the algorithm for prime $N$ is that by carefully choosing the first pair of permutations, one can decrease the number of iterations needed by the algorithm to complete. In addition, a finer analysis shows that the size of sets $\Lambda_1$ and $\Lambda_1'$ (that is, the set of wavelengths that are used three or two times in the first-stage permutation) in some iteration $x$ depends on $x$. Thus, for small $x$, one can always find a middle-stage port $j$ to decrease the potential of the second-stage permutation.

We start by characterizing a set of $N-2$ candidate permutations for the initial first-stage permutations. Specifically, we show in Lemma 7.1 that all permutations are 1-legal and that if a pair of middle-stage ports share the same wavelength in a second-stage permutation induced by one of the candidates, they cannot share the same wavelength in any of the second-stage permutations induced by all other candidates. Lemma 7.1 appears in Appendix B to enhance readability.

*Lemma 7.1:* If $N$ is a prime number, then for every $r \in \{2, \ldots, N-1\}$, the permutation $\pi_r$ defined as follows:

$$\pi_r[i] = r \cdot i \mod N \quad i \in \{0, \ldots, N-1\}$$

is a 1-legal permutation. Furthermore, for any permutation $\pi$ of $\{0, \ldots, N-1\}$, for every $r_1, r_2 \in \{2, \ldots, N-1\}$, and for every $i, j$, if $\lambda(\pi \circ \pi_{r_1}^{-1})[i] = \lambda(\pi \circ \pi_{r_1}^{-1})[j]$ and $\lambda(\pi \circ \pi_{r_2}^{-1})[i] = \lambda(\pi \circ \pi_{r_2}^{-1})[j]$, then $i = j$.

We next show that choosing the best candidate among all these permutations significantly improves the worst-case potential of the induced second-stage permutation.

*Lemma 7.2:* The 3-potential of $\pi_2$ in Line 2 of Algorithm 3 is at most $\frac{N}{8}$.

**Algorithm 3** A 3-Legal Decomposition Algorithm That Decomposes a Permutation $\pi$ of to Two Permutations $\pi_1$ and $\pi_2$, Assuming That $N$ is a Prime Number.

1: $\pi_1 \leftarrow \arg\min_{\{\pi_r | r \in 2, N-1\}} pot_3(\pi \circ \pi_r^{-1})$
   ▷ $\pi_r$ as defined in Definition 7.1
2: $\pi_2 \leftarrow \pi \circ \pi_1^{-1}$
3: **while** $\pi_2$ is not 3-legal **do**
4: $\quad \langle \pi_1, \pi_2 \rangle \leftarrow$CORRECT$(\langle \pi_1, \pi_2 \rangle, 3)$
5: **end while**
6: $\langle \pi_1, \pi_2 \rangle$ is a 3-legal decomposition of $\pi$

*Proof:* Assume toward a contradiction that $pot_3(\pi_2) > \frac{N}{8}$. Note that by Line 2 of Algorithm 3 and by the assumption, for each $r \in \{2, \ldots, N-1\}$, $pot_3(\pi_{2,r}) \geq pot_3(\pi_2) > N/8$, where $\pi_{2,r} = \pi \circ \pi_r^{-1}$. For each such permutation $\pi_{2,r}$, we count the number of input/output pairs that use the same wavelength under the permutation and show that to realize a potential of at least $N/8$, we need to have at least $5\,N/8$ such pairs.

Recall that by Definition 3, $pot_3(\pi_{2,r}) = \sum_{\lambda \in \Lambda} \max\{0, \#_{\lambda(\pi_{2,r})}[\lambda] - 3\}$. We focus on the set $\Lambda' \subseteq \Lambda$ of wavelengths in which $\#_{\lambda(\pi_{2,r})}[\lambda] > 3$; notice that, for each such wavelength, the number of pairs we count is $\frac{\#_{\lambda(\pi_{2,r})}[\lambda](\#_{\lambda(\pi_{2,r})}[\lambda]-1)}{2}$, while the potential it contributes is $\#_{\lambda(\pi_{2,r})} - 3$. Thus, the number of pairs per potential unit is

$$\frac{\#_{\lambda(\pi_{2,r})}[\lambda](\#_{\lambda(\pi_{2,r})}[\lambda] - 1)}{2(\#_{\lambda(\pi_{2,r})} - 3)}.$$

This term is minimized when $\#_{\lambda(\pi_{2,r})}[\lambda] = 3 + \sqrt{6} = 5.45$. Since $\#_{\lambda(\pi_{2,r})}[\lambda]$ is an integer, it implies that the number of pairs per potential unit is minimized when $\#_{\lambda(\pi_{2,r})}[\lambda]$ is either

[4]For $N > 28$, this claim follows by applying Rosser and Schoenfeld's bounds on the prime-counting function [25], showing that its value for $N$ is strictly less than its value for $1.375\ N$. For $N \leq 28$, the statement can be verified manually.

5 or 6, which in both cases yields five pairs per potential unit, and at least $5\ N/8$ pairs for a potential of $N/8$.

Lemma 7.1 implies that two input ports that use the same wavelength in a permutation $\pi_{2,r}$ cannot use the same wavelength in another permutation. This implies that we cannot add the same pair twice. Thus, to realize a potential of $N/8$ in all $N-2$ permutations, we need $5\ N(N-2)/8$ distinct pairs. However, for every $N > 6$, $5\,N(N-2)/8 > N(N-1)/2$, which is the maximum number of pairs and, hence, a contradiction. For $N = 3$ and $N = 2$, all permutations are 3-legal. Therefore, trivially the 3-potential of $\pi_2$ is 0. For $N = 5$, we verified by exhaustive search that the claim holds, and that the potential of $\pi_2$ is always 0. ∎

It is important to notice that Lemma 6.1 holds for any value of $k$. In particular, it implies that the number of iterations required for Algorithm 3 to stop is at most $\frac{N}{8}$.

It is left to prove that the algorithm can proceed in each iteration (see Appendix C).

*Lemma 7.3:* At each iteration of every execution of Algorithm 3, the procedure CORRECT does not fail.

Finally, the correctness of our algorithm is given by the following theorem.

*Theorem 7.4:* For any prime $N$ and any permutation $\pi$ of $\{0,\ldots,N-1\}$, Algorithm 3 finds a 3-legal decomposition of $\pi$ in $O(N^2)$ time.

*Proof:* Consider an arbitrary permutation $\pi$. Let $\pi_1$ be the initial first-stage permutation chosen in Line 1 and $\pi_2$ the initial second-stage permutation resulting in executing Line 2. Denote by $x = pot_3(\pi_2)$. By Lemma 7.2, $x \leq \frac{N}{8}$. By Lemma 7.3, the invocations of CORRECT never fails. Thus, Lemma 6.1 implies that each iteration of the algorithm (Lines 3-5) decreases the potential of $\pi_2$ by at least 1, implying that after $y \leq x$ iterations the potential of the second-stage permutation $\pi_2$ is 0. Therefore, $\pi_2$ is 3-legal. In addition, since the CORRECT procedure never fails and the initial first-stage permutation $\pi_1$ is 3-legal, Lemma 6.1 also implies that the first-stage permutation $\pi_1$ is also 3-legal after iteration $y$. Since: 1) the only changes in the permutations were made by transpositions (Algorithm 1, Line 18); and 2) after the execution of Line 2 $\pi_1 \circ \pi_2 = \pi$, Definition 2 implies that when the algorithm ends, $\pi_1 \circ \pi_2$ is still $\pi$. Therefore, $\langle \pi_1, \pi_2 \rangle$ is a 3-legal decomposition of $\pi$.

The running time of the algorithm is derived by the fact that each iteration is linear in the number of ports (see details in the proof of Theorem 6.3) and the number of iterations is bounded by $\frac{N}{8}$. ∎

## VIII. Hardware Considerations

In this section, we take a closer look at our 4-legal decomposition algorithms (Algorithms 1 and 2) and present a *linear time parallel* implementation, assuming that the following primitives are available and that they operate in constant time (independent of $N$).

1) Bit-wise operations of width $N$, including circular shifts (bit-wise rotations). We denote circular shifts by $\gg_{\text{circ}}$ (e.g., $101000101 \gg_{\text{circ}} 2 = 011010001$). Note that circular shifts can be implemented by two arithmetic shifts and a bit-wise OR.
2) $N$-bit priority encoder, which given a vector of $N$ bits returns the index of the left-most bit set to 1.

TABLE II
Data Structures Maintained When Implementing Algorithms 1 and 2

| | |
|---|---|
| $\pi_1$, $\pi_1^{-1}$ | Defined in Algorithm 2, Lines 1-4 |
| $\pi_2$, $\pi_2^{-1}$ | Defined in Algorithm 2, Line 5 |
| $\Psi$ | The set of all middle-stage ports using a wavelength which is used in $\pi_2$ more than 4 times (3 times for Algorithm 3) |
| $M_1$ | Mapping between a wavelength and the middle-stage ports using it in $\pi_1$ |
| $M_2$ | Mapping between a wavelength and the middle-stage ports using it in $\pi_2$ |
| $\Lambda_1$ | The set defined in Algorithm 1, Line 4 |
| $\Lambda_1'$ | The set defined in Algorithm 1, Line 5 |
| $\Lambda_2$ | The set defined in Algorithm 1, Line 6 |
| $\Lambda_2'$ | The set defined in Algorithm 1, Line 7 |
| $-\Lambda_1$ | $\{N-\ell \mid \ell \in \Lambda_1\}$ |
| $-\Lambda_2$ | $\{N-\ell \mid \ell \in \Lambda_2\}$ |
| $-\Lambda_1'$ | $\{N-\ell \mid \ell \in \Lambda_1'\}$ |
| $-\Lambda_2'$ | $\{N-\ell \mid \ell \in \Lambda_2'\}$ |
| $Q_1$ | An $N \times N$ bit matrix whose $(x,\ell)$ element is set if and only if $\ell + \pi_1^{-1}[\ell] = x$; we refer to $k$-th row of $Q_1[k]$ as a bit array of size $N$. |
| $Q_2$ | An $N \times N$ bit matrix whose $(x,\ell)$ element is set if and only if $\ell + \pi_2[\ell] = x$; we refer to $k$-th row of $Q_2[k]$ as a bit array of size $N$. |

3) Applying a permutation $\pi$ on a bit-array $A$ of length $N$; the resulting bit-array $A'$ will have the following property: $A'[\pi[\ell]] = 1$ if and only if $A[\ell] = 1$. Since $\pi$ is a permutation, this can be computed without conflicts using $N$ parallel operations.

We divide the operations of the algorithm into three phases. The first phase is executed only once at the algorithm setup and takes linear time. The second and third phases are executed in at most $N-4$ iterations, each taking constant time.

**Setup Phase:** This phase consists of computing the initial value of the data structures depicted in Table II. We assume that all sets are represented by bit arrays of width $N$ such that a bit $i$ $(0 \leq i < N)$ is 1 if and only if element $i$ is in the set. Each permutation $\pi$ is stored by an array of $N\ \lceil \log N \rceil$ bits, such that its $i$th element holds (explicitly) the value $\pi[i]$ in $\lceil \log N \rceil$ bits. The mappings $M_1$ and $M_2$ are arrays of sets: $M_1[i]$ $(M_2[i])$ holds the set (represented as a bitmap) of middle-stage ports using wavelength $\lambda_i$ in $\pi_1$ $(\pi_2)$; moreover, we assume there is an additional counter that counts the number of middle-stage ports using each wavelength in each permutation.

Clearly, all values can be computed in linear time (notice that there are exactly $N$ 1-bits in matrices $Q_1$ and $Q_2$ and in mappings $M_1$ and $M_2$; furthermore, the sum of all the counters in each mapping is also $N$).

**Body Phase:** This phase is computing the value of the middle-stage ports $i$ and $j$, which are used for the $(i,j)$-transposition (Algorithm 1, Line 18).

First, we note that choosing the input $i$, which we need to correct (Line 3, Algorithm 1), can be done by applying a priority encoder on set $\Psi$. Given $i$, one can compute in constant time the values $\pi_1^{-1}[i]$ and $\pi_2[i]$.

Second, we notice that sets $A_1,\ldots,A_4$ can be computed in constant time using the primitives: $A_1 = \Lambda_1 \gg_{\text{circ}} \pi_1^{-1}[i]$, $A_3 = -\Lambda_2 \gg_{\text{circ}} \pi_2[i]$, $A_2$ is obtained by applying permutation $\pi_1$ on $-\Lambda_1 \gg_{\text{circ}} i$, and $A_4$ is obtained by applying permutation $\pi_2^{-1}$ on $\Lambda_2 \gg_{\text{circ}} i$. Set $A_5$ is computed in two steps: First we compute the set $\{\ell \mid i - \pi_1^{-1}[\ell] \in \Lambda_1'\} \supseteq A_5$ by applying $\pi_1$ on $-\Lambda_1' \gg_{\text{circ}} i$, then we intersect it with set $Q_1[i+\pi_1^{-1}[i]]$ using a bit-wise AND. Similarly, set $A_6$ is computed by intersecting set $\{\ell | \pi_2[i] - \ell \in \Lambda_2'\} = -\Lambda_2' \gg_{\text{circ}} \pi_2[i]$ with set $Q_2[i+\pi_2[i]]$.

Finally, the set $A_1 \cup A_2 \cup A_3 \cup A_4 \cup A_5 \cup A_6$ is obtained by a bit-wise OR on the corresponding sets, and the middle-stage port $j$ (Algorithm 1, Line 14) is obtained by applying a priority encoder on the negation of this result.

**Update Phase:** In this phase, we update our data structures to reflect the $(i, j)$ transposition of $\langle \pi_1, \pi_2 \rangle$ (Algorithm 1, Line 18). We note that this transposition involves only a constant number of changes. Namely, only two middle-stage ports and at most eight wavelengths: four added wavelengths $\pi_2[i] - j, \pi_2[j] - i, j - \pi_1^{-1}[i], i - \pi_1^{-1}[j]$ and four removed wavelengths $\pi_2[j] - j, \pi_2[i] - i, j - \pi_1^{-1}[j], i - \pi_1^{-1}[i]$. Note that the number of middle-stage ports using a specific wavelength can be checked in constant time by reading the corresponding counter in mappings $M_1$ and $M_2$. The algorithm terminates when $\Psi = \emptyset$ after the update phase.

A detailed example of a complete run of the algorithm is worked out in Appendix D.

Finally, we note that this implementation works also for our 3-decomposition algorithm (Algorithms 1 and 3) with the following change in the setup phase: When computing $\pi_1$ (Algorithm 3, Line 1), one should compute in parallel each value of $pot_3(\pi \circ \pi_r^{-1})$ for each $2 \leq r \leq N - 1$, and then choose the permutation with minimum potential. Each such computation takes linear time; thus, with parallelism, the setup phase works in linear time also in this case.

## IX. Conclusion

In this paper, we studied ways to overcome coherent crosstalk impairments in AWG-based optical switching fabrics. The notion of $k$-legal permutations was introduced, in which each wavelength is reused at most $k$ times. We first found properties of 1-legal permutations, showing that a difference exists between odd and even values of the number of input and output ports $N$. We then showed that uniform traffic patterns can be scheduled in input-queued cell switches using 1-legal permutations with no speedup for odd $N$ and with a small speedup with even $N$. General traffic patterns can instead be scheduled with 1-legal permutations using two-stage load-balanced switches using the same small speedup, no input queues, VOQs between the two switching stages, and cell resequencing at outputs.

Finally, the major contribution of our paper is that we were able to formally prove that a two-stage switch can be configured with pairs of 4-legal permutations with no buffering between the two stages, and a quadratic decomposition algorithm is presented. 3-legal permutation pairs can be algorithmically found when the number of ports is a prime number, or when a small spatial speedup is introduced for arbitrary number of ports.

In summary, our results show that by using proper hardware configurations and scheduling algorithms, the physical-layer impairments due to coherent crosstalk can be practically neglected in AWG-based optical switches with arbitrary number of ports.

## Appendix A
## Proof of Lemma 6.1

We first focus on the first-stage permutation. The proof follows from the construction of the algorithm.

Assume $\pi_1$ is $k$-legal in the beginning of the execution, and denote by $\pi_1'$ the first-stage permutation resulting by executing Line 18. Assume by the way of contradiction that $\pi_1'$ is not $k$-legal, implying that at $\pi_1'$ a wavelength $\lambda$ is used (strictly) more than $k$ times. The only wavelengths that were introduced to first-stage permutation are $\lambda_1 = j - \pi_1^{-1}[i]$ and $\lambda_2 = i - \pi_1^{-1}[j]$, implying that $\lambda$ is either $\lambda_1$ or $\lambda_2$.

If $\lambda = \lambda_1$ and $\lambda_1 \neq \lambda_2$, $\lambda_1$ appears $k$ times in $\pi_1$, and therefore it belongs to $\Lambda_1$; thus, $j \in A_1$, contradicting Line 14. If $\lambda = \lambda_2$ and $\lambda_1 \neq \lambda_2$, it implies that $\lambda_2$ appears $k$ times in $\pi_1$ and therefore belongs to $\Lambda_1$; thus, $j \in A_2$, contradicting Line 14.

We consider now the case that $\lambda = \lambda_1 = \lambda_2$, which implies that $j + \pi_1^{-1}[j] = i + \pi_1^{-1}[i]$. If $\lambda$ appears $k$ times in $\pi_1$ it belongs to $\Lambda_1$ and $j \in A_1$, contradicting Line 14; otherwise, if $\lambda_1$ appears $k - 1$ times in $\pi_1$ it belongs to $\Lambda_1'$ and $j \in A_5$, also contradicting Line 14.

Thus, we showed that all cases contradict the choice of $j$ at Line 14, hence either the algorithm fails (no valid choice of $j$ exists) or $\pi_1'$ is $k$-legal and the claim follows.

We now turn to show that if the algorithm does not fail, the potential of the second permutation strictly decreases. Since the algorithm succeeds, the transposition in Line 18 is executed; let $i$ and $j$ be the ports used for this transposition. Furthermore, denote by $\pi_2$ the second-stage permutation at the beginning of the execution, and let $\pi_2'$ be the second-stage permutation after executing Line 18.

We will consider the following four wavelengths: $\lambda_1 = \pi_2[i] - i$, $\lambda_2 = \pi_2[j] - j$, $\lambda_3 = \pi_2[i] - j$, $\lambda_4 = \pi_2[j] - i$.

We notice that the only changes in the number of appearances was in these four wavelengths, therefore

$$
\begin{aligned}
pot_k(\pi_2) - pot(\pi_2') &= \sum_{\lambda \in \Lambda} \max\left\{0, \#_{\lambda(\pi_2)}[\lambda] - k\right\} \\
&\quad - \sum_{\lambda \in \Lambda} \max\left\{0, \#_{\lambda(\pi_2')}[\lambda] - k\right\} && (1) \\
&= \sum_{\lambda \in \{\lambda_1, \ldots, \lambda_4\}} \Big( \max\left\{0, \#_{\lambda(\pi_2)}[\lambda] - k\right\} \\
&\qquad - \max\left\{0, \#_{\lambda(\pi_2')}[\lambda] - k\right\} \Big) && (2) \\
&= \sum_{\lambda \in \{\lambda_3, \lambda_4\}} \max\left\{0, \#_{\lambda(\pi_2)}[\lambda] - k\right\} \\
&\quad + \sum_{\lambda \in \{\lambda_1, \lambda_2\}} \Big( \max\left\{0, \#_{\lambda(\pi_2)}[\lambda] - k\right\} \\
&\qquad - \max\left\{0, \#_{\lambda(\pi_2')}[\lambda] - k\right\} \Big) && (3) \\
&\geq \sum_{\lambda \in \{\lambda_1, \lambda_2\}} \Big( \max\left\{0, \#_{\lambda(\pi_2)}[\lambda] - k\right\} \\
&\qquad - \max\left\{0, \#_{\lambda(\pi_2')}[\lambda] - k\right\} \Big) && (4) \\
&\geq \max\left\{0, \#_{\lambda(\pi_2)}[\lambda_1] - k\right\} \\
&\qquad - \max\left\{0, \#_{\lambda(\pi_2')}[\lambda_1] - k\right\} && (5) \\
&\geq 1 && (6)
\end{aligned}
$$

where (1) holds by Definition 3 and (2) is since the only changes are related to wavelengths $\lambda_1, \ldots, \lambda_4$.

The gist of the proof lies in inequality (3) and is due to the fact that for each $\lambda \in \{\lambda_3, \lambda_4\}$, $\max\left\{0, \#_{\lambda(\pi_2')}[\lambda] - k\right\} = 0$: Assume otherwise, and consider first the case where $\lambda_3 \neq \lambda_4$. Thus, if $\#_{\lambda(\pi_2')}[\lambda_3] > k$, it implies that $\#_{\lambda(\pi_2)}[\lambda_3] \geq k$, hence $\lambda_3 \in \Lambda_2$, and therefore $j \in A_3$, contradicting Line 14; similarly, $\#_{\lambda(\pi_2')}[\lambda_4] > k$ implies $j \in A_4$, also contradicting Line 14. In case $\lambda_3 = \lambda_4$, $\#_{\lambda(\pi_2')}[\lambda_3] > k$ implies that $\#_{\lambda(\pi_2)}[\lambda_3] \geq k-1$, hence $\lambda_3 \in \Lambda_2'$ and $j + \pi_2[j] = i + \pi_2[i]$, and therefore $j \in A_6$, which contradicts Line 14 as well.

Inequality (4) holds because each term in the first summation is nonnegative by definition. Inequality (5) stems from the fact that the number of appearances of $\lambda_2$ decreases between $\pi_2$ and $\pi_2'$ unless $\lambda_2 = \lambda_3$ or $\lambda_2 = \lambda_4$; in the latter case, with the same reasoning of (3), we get that $\max\left\{0, \#_{\lambda(\pi_2')}[\lambda_2] - k\right\} = 0$, and inequality (5) follows as well. Finally, inequality (6) holds since by the choice of $i$ in Line 3, $\#_{\lambda(\pi_2)}[\lambda_1] > k$, and therefore

$$\max\left\{0, \#_{\lambda(\pi_2)}[\lambda_1] - k\right\} = \#_{\lambda(\pi_2)}[\lambda_1] - k \tag{7}$$

and $\lambda_1 \in \Lambda_2$. Therefore, $\lambda_1$ is not equal to $\lambda_3$ or $\lambda_4$, which immediately implies that

$$\#_{\lambda(\pi_2)}[\lambda_1] - \#_{\lambda(\pi_2')}[\lambda_1] \geq 1. \tag{8}$$

Combining (7), (8), and the fact that $\#_{\lambda(\pi_2')}[\lambda_1] - k \leq \max\left\{0, \#_{\lambda(\pi_2')}[\lambda_1] - k\right\}$ immediately yields inequality (6).

## Appendix B
## Proof of Lemma 7.1

We first prove that for every $r \in \{1, \ldots, N-1\}$, $\pi_r$ is a permutation. Assume $\pi_r$ is not a permutation, thus there are $i, j$ such that $i > j$ and $\pi_r[i] = \pi_r[j]$, implying that $r(i-j) = 0 \bmod N$. Note that since $N$ is a prime and $0 < r, (i-j) < N$, $r(i-j) = 0 \bmod N$ implies the existence of a zero-divisor in the field $\mathrm{GF}(N)$, which is a contradiction.

Furthermore, for every $r \in \{2, \ldots, N-1\}$, $\pi_r$ is 1-legal since the wavelength assignment $\lambda(\pi_r) = \pi_r - I = \pi_{r-1}$ is also a permutation because $r - 1 \in \{1, \ldots, N-2\}$.

We conclude by showing the claim on the induced second-stage permutation. Assume, without loss of generality, that $r_1 > r_2$. We first observe that, for each $a \in \{i, j\}$ and $b \in \{r_1, r_2\}$, $\lambda(\pi \circ \pi_b^{-1})[a] = \pi[a] - ab$. Thus, we can state that

$$\pi[i] - r_1 i = \pi[j] - r_1 j \tag{9}$$
$$\pi[i] - r_2 i = \pi[j] - r_2 j. \tag{10}$$

By subtracting (9) from (10) we get $(r_1 - r_2)i = (r_1 - r_2)j \bmod N$. Note that $r_1 - r_2 \in \{1, \ldots, N-2\}$, thus $i = j$ since $\pi_{r_1 - r_2}$ is a permutation.

## Appendix C
## Proof of Lemma 7.3

In order to prove Lemma 7.3, we first prove the following auxiliary lemma.

*Lemma C.1:* After each iteration $x$ of Algorithm 3, $2|\Lambda_1| + |\Lambda_1'| \leq 2x$.

*Proof:* The proof is by induction on the iteration number $x$. We denote by $\Lambda_1(x)$ ($\Lambda_1'(x)$) the set $\Lambda_1$ ($\Lambda_1'$) after the iteration $x$, and by $\pi_{1,x}$ the first-stage permutation obtained after iteration $x$.

When $x = 0$ (that is, before the first iteration), $\Lambda_1(0) = \Lambda_1'(0) = \emptyset$ since the initial first-stage iteration is 1-legal, and therefore the base case holds.

We now assume that the claim holds after iteration $x-1$ and prove that it holds for iteration $x$. In each iteration, the algorithm introduces only a single transposition, implying that at most two input ports, $i$ and $j$ changed their wavelength assignment in iteration $x$. Denote by $\lambda_i$ and $\lambda_j$ the new wavelengths that ports $i$ and $j$ use after iteration $x$. Without loss of generality, assume that $\#_{\lambda(\pi_{1,x-1})}[\lambda_i] \geq \#_{\lambda(\pi_{1,x-1})}[\lambda_j]$. Furthermore, note that by Algorithm 1, $\lambda_i, \lambda_j \notin \Lambda_1(x-1)$, and if $\lambda_i = \lambda_j$, then $\lambda_i, \lambda_j \notin \Lambda_1'(x-1)$ as well.

We proceed by considering the following four cases, establishing our induction step. In each case, we consider a worst-case situation where as many possible wavelengths are added to $\Lambda_1$ or $\Lambda_1'$, thus increasing the term $2|\Lambda_1| + |\Lambda_1'|$.

- $\lambda_i \neq \lambda_j$, $\lambda_i \notin \Lambda_1'(x-1)$. In this case, in the worst case, both $\lambda_i$ and $\lambda_j$ appeared $k-2$ times in $\pi_{1,x-1}$ and are therefore added to $\Lambda_1'(x)$. Hence, $2|\Lambda_1(x)| + |\Lambda_1'(x)| \leq 2|\Lambda_1(x-1)| + |\Lambda_1'(x-1)| + 2 \leq 2(x-1) + 2 = 2x$, where the last inequality holds by the induction hypothesis. Note that situations in which only one of the wavelengths is added to $\Lambda_1'$ and/or other wavelengths are omitted either from $\Lambda_1$ or $\Lambda_1'$ trivially hold as well.
- $\lambda_i = \lambda_j$, $\lambda_i \notin \Lambda_1'(x-1)$. In this case, in the worst case, $\lambda_i \in \Lambda_1(x)$ implying that $2|\Lambda_1(x)| + |\Lambda_1'(x)| \leq 2(|\Lambda_1(x-1)| + 1) + |\Lambda_1'(x-1)| \leq 2(x-1) + 2 = 2x$ as well. Note that if $\lambda_i$ is added to $\Lambda_1'(x)$, then $2|\Lambda_1| + |\Lambda_1'| \leq 2x - 1 \leq 2x$.
- $\lambda_i \neq \lambda_j$, $\lambda_i \in \Lambda_1'(x-1)$, $\lambda_j \notin \Lambda_1'(x-1)$. In this case, in the worst case, $\lambda_i \in \Lambda_1(x)$ and $\lambda_j \in \Lambda_1'(x)$. However, since $\lambda_i \neq \lambda_j$, then $\Lambda_1'(x) = (\Lambda_1'(x-1) \setminus \{\lambda_i\}) \cup \{\lambda_j\}$, and therefore the size of $\Lambda_1'$ does not change. Hence, $2|\Lambda_1(x)| + |\Lambda_1'(x)| \leq 2(|\Lambda_1(x-1)| + 1) + |\Lambda_1'(x-1)| \leq 2(x-1) + 2 = 2x$.
- $\lambda_i \neq \lambda_j$, $\lambda_i \in \Lambda_1'(x-1)$, $\lambda_j \in \Lambda_1'(x-1)$. In this case, in the worst case, $\lambda_i, \lambda_j \in \Lambda_1(x)$, and therefore $\lambda_i, \lambda_j \notin \Lambda_1'(x)$. This implies that the size of $\Lambda_1'$ decreases by 2 while the size of $\Lambda_1$ increases by 2. Thus, $2|\Lambda_1(x)| + |\Lambda_1'(x)| \leq 2(|\Lambda_1(x-1)| + 2) + (|\Lambda_1'(x-1)| - 2) \leq 2(x-1) + 2 = 2x$.

■

We now turn to prove that the procedure CORRECT never fails. It is sufficient to show that there is a valid choice of a middle-stage port $j$ in Line 14 of Algorithm 1.

Consider the $x$th iteration ($x \leq \frac{N}{8}$). By Lemma C.1, at the beginning of this iteration, $|A_1 \cup A_2 \cup A_5| \leq |A_1| + |A_2| + |A_5| \leq 2|\Lambda_1| + |\Lambda_1'| \leq 2x$. Similarly to the proof in Lemma 6.2, the size of $|A_3 \cup A_4 \cup A_6| \leq 2|\Lambda_2| + |\Lambda_2'| - 1$, which is bounded by $\frac{2N}{3} - 1$, since $3|\Lambda_2| + 2|\Lambda_2'| \leq N$. Thus, $|\bigcup_{r=1}^{6} A_r| \leq |A_1 \cup A_2 \cup A_5| + |A_3 \cup A_4 \cup A_6| \leq 2x + 2(N/3) - 1 \leq N - 1$, where the last inequality is since $x \leq \frac{N}{8}$.

## Appendix D
## Detailed Example of a 4-Legal Decomposition

This section provides a detailed example of a complete run of our 4-legal decomposition algorithm and, more specifically, of its hardware implementation.

We consider an $11 \times 11$ switch and aim at decomposing the permutation $\pi = [0, 2, 4, 7, 9, 5, 1, 3, 6, 8, 10]$. Since $N$ is odd, the permutation $\pi_{\text{odd}} = [0, 2, 4, 6, 8, 10, 1, 3, 5, 7, 9]$ is chosen as the first-stage permutation $\pi_1$; this implies that

the second-stage permutation $\pi_2$, which is not 4-legal, is $[0,1,2,3,4,6,7,8,9,10,5]$. We also compute and store the inverse permutations $\pi_1^{-1} = [0,6,1,7,2,8,3,9,4,10,5]$ and $\pi_2^{-1} = [0,1,2,3,4,10,5,6,7,8,9]$.

Two wavelengths are used five times in $\pi_2$: Wavelength $\lambda_0$ is used by middle-stage ports $0,\ldots,4$, and wavelength $\lambda_1$ is used by middle-stage ports $5,\ldots,9$. Thus, $\Psi$ is the bit-vector representation of the set of ports $\{0,\ldots,9\}$; namely, $\Psi = 11111111110$.

We continue by computing the mappings $M_1$ and $M_2$, which link a wavelength with the middle-stage port using it. Specifically

$$M_1 = \begin{bmatrix} 10000000000 \\ 01000000000 \\ 00100000000 \\ 00010000000 \\ 00001000000 \\ 00000100000 \\ 00000010000 \\ 00000001000 \\ 00000000100 \\ 00000000010 \\ 00000000001 \end{bmatrix}; \quad M_2 = \begin{bmatrix} 11111000000 \\ 00000111110 \\ 00000000000 \\ 00000000000 \\ 00000000000 \\ 00000000000 \\ 00000000001 \\ 00000000000 \\ 00000000000 \\ 00000000000 \\ 00000000000 \end{bmatrix}.$$

Note also that we attach a counter to each row of the mapping: All the 11 counters of $M_1$ are set to 1, while the first two counters of $M_2$ (corresponding to $\lambda_0$ and $\lambda_1$) are set to 5, the seventh counter (corresponding to $\lambda_6$) is set to 1, and all other counters are set to 0. Set $\Lambda_1$ ($\Lambda_1'$) consists of the set of wavelengths used in $\pi_1$ at least four (exactly three) times and are both empty. This implies that $\Lambda_1 = \Lambda_1' = -\Lambda_1 = -\Lambda_1' = 00000000000$. $\Lambda_2 = 11000000000$ because wavelengths $\lambda_0$ and $\lambda_1$ are used in $\pi_2$ at least four times. Since only wavelengths $\lambda_0$ and $\lambda_1$ are in $\Lambda_2$, then $-\Lambda_2 = \{11-0, 11-1\} = \{11,10\} = \{0,10\} = 10000000001$. Sets $\Lambda_2'$ and $-\Lambda_2'$ are both 00000000000 (empty sets).

We conclude the setup phase by computing $Q_1$ and $Q_2$. For ease of explanation, consider the following vectors: $\ell + \pi_1^{-1}[\ell] = [0,7,3,10,6,2,9,5,1,8,4]$ and $\ell + \pi_2[\ell] = [0,2,4,6,8,0,2,4,6,8,4]$. Thus, for example, in $Q_1$, bit (4, 10) is set to 1, corresponding to the last element of $\ell + \pi_1^{-1}[\ell]$. Specifically

$$Q_1 = \begin{bmatrix} 10000000000 \\ 00000000100 \\ 00000100000 \\ 00100000000 \\ 00000000001 \\ 00000001000 \\ 00001000000 \\ 01000000000 \\ 00000000010 \\ 00000010000 \\ 00010000000 \end{bmatrix}; \quad Q_2 = \begin{bmatrix} 10000100000 \\ 00000000000 \\ 01000010000 \\ 00000000000 \\ 00100001001 \\ 00000000000 \\ 00010000100 \\ 00000000000 \\ 00001000010 \\ 00000000000 \\ 00000000000 \end{bmatrix}.$$

The first iteration starts by applying a priority encoder on $\Psi$, resulting in $i = 0$. Note that $\pi_1^{-1}[0] = 0$ and $\pi_2[0] = 0$. We proceed by computing the six sets $A_1,\ldots,A_6$.

- $A_1 = \Lambda_1 \gg_{\text{circ}} \pi_1^{-1}[i] = 00000000000 \gg_{\text{circ}} 0 = 00000000000$.
- $A_2 = \pi_1(-\Lambda_1 \gg_{\text{circ}} i) = \pi_1(00000000000 \gg_{\text{circ}} 0) = \pi_1(00000000000) = 00000000000$.
- $A_3 = -\Lambda_2 \gg_{\text{circ}} \pi_2[i] = 10000000001 \gg_{\text{circ}} 0 = 10000000001$.
- $A_4 = \pi_2^{-1}(\Lambda_2 \gg_{\text{circ}} i) = \pi_2^{-1}(11000000000 \gg_{\text{circ}} 0) = \pi_2^{-1}(1100000000) = 1100000000$.
- $A_5 = Q_1[0]$ and $\pi_1(-\Lambda_1' \gg_{\text{circ}} i) = 00000000000$.
- $A_6 = Q_2[0]$ and $-\Lambda_2' \gg_{\text{circ}} \pi_2[i] = 10000100000$ and $00000000000 = 00000000000$.

This implies that $\neg(A_1 \mid A_2 \mid A_3 \mid A_4 \mid A_5 \mid A_6) = 00111111110$, and middle port $j = 2$ is selected for the transposition. Note that if we had selected middle port 1 or middle port 10, it would have resulted in increasing the number of middle-stage ports using $\lambda_1$ to 6.

We conclude by the update phase, in which we update a constant number of elements after the $(0,2)$-transpositions. Table III depicts the updated data structures, where changes are underlined. Notice that the set $\Psi$ now shows only middle-stage ports that use wavelength $\lambda_1$; the update is done in constant time by applying $\Psi\&(\neg M_2[0])$, where $M_2[0]$ is the first row of mapping $M_2$ of the first iteration.

In the second iteration, middle port $i = 5$ is selected. Notice that $\pi_1^{-1}[5] = 8$ and $\pi_2[5] = 6$. Thus, the six sets are as follows.

- $A_1 = \Lambda_1 \gg_{\text{circ}} \pi_1^{-1}[i] = 00000000000 \gg_{\text{circ}} 8 = 00000000000$.
- $A_2 = \pi_1(-\Lambda_1 \gg_{\text{circ}} i) = \pi_1(00000000000 \gg_{\text{circ}} 5) = \pi_1(00000000000) = 00000000000$.
- $A_3 = -\Lambda_2 \gg_{\text{circ}} \pi_2[i] = 00000000001 \gg_{\text{circ}} 6 = 00000100000$.
- $A_4 = \pi_2^{-1}(\Lambda_2 \gg_{\text{circ}} i) = \pi_2^{-1}(01000000000 \gg_{\text{circ}} 5) = \pi_2^{-1}(0000001000) = 0000010000$.
- $A_5 = Q_1[2]$ & $\pi_1(-\Lambda_1' \gg_{\text{circ}} i) = 00000000000$.
- $A_6 = Q_2[0]$ and $-\Lambda_2' \gg_{\text{circ}} \pi_2[i] = 00000100000$ and $00000010000 = 00000000000$.

TABLE III
VALUES OF THE DATA STRUCTURES AFTER THE FIRST UPDATE PHASE. UNDERLINED ELEMENTS MARK CHANGES FROM THE INITIAL VALUES

$\pi_1 = [\underline{2},\underline{0},4,6,8,10,1,3,5,7,9]$ $\quad \pi_1^{-1} = [\underline{1},6,\underline{0},7,2,8,3,9,4,10,5]$
$\pi_2 = [\underline{2},1,\underline{0},3,4,6,7,8,9,10,5]$ $\quad \pi_2^{-1} = [\underline{2},1,\underline{0},3,4,10,5,6,7,8,9]$

$\Psi = \underline{00000}111110$

$$M_1 = \begin{bmatrix} \underline{0}0000000000 \\ \underline{0}0000000000 \\ \underline{1}0100000000 \\ 00010000000 \\ 00001000000 \\ 00000100000 \\ 00000010000 \\ 00000001000 \\ 00000000100 \\ 00000000010 \\ 0\underline{1}000000001 \end{bmatrix} \quad M_2 = \begin{bmatrix} \underline{0}\underline{1}011000000 \\ 00000111110 \\ \underline{1}0000000000 \\ 00000000000 \\ 00000000000 \\ 00000000000 \\ 00000000001 \\ 00000000000 \\ 00000000000 \\ 00\underline{1}00000000 \\ 00000000000 \end{bmatrix}$$

$\Lambda_1 = 00000000000$ $\quad -\Lambda_1 = 00000000000$
$\Lambda_1' = 00000000000$ $\quad -\Lambda_1' = 00000000000$
$\Lambda_2 = \underline{0}1000000000$ $\quad -\Lambda_2 = \underline{0}0000000001$
$\Lambda_2' = \underline{1}0000000000$ $\quad -\Lambda_2' = \underline{1}0000000000$

$$Q_1 = \begin{bmatrix} \underline{0}0000000000 \\ \underline{1}0000000100 \\ 00\underline{1}00100000 \\ 00\underline{0}00000000 \\ 00000000001 \\ 00000001000 \\ 00001000000 \\ 01000000000 \\ 00000000010 \\ 00000010000 \\ 00010000000 \end{bmatrix} \quad Q_2 = \begin{bmatrix} \underline{0}0000100000 \\ 00000000000 \\ \underline{11}\underline{1}00010000 \\ 00000000000 \\ 00\underline{0}00001001 \\ 00000000000 \\ 00010000100 \\ 00000000000 \\ 00001000010 \\ 00000000000 \\ 00000000000 \end{bmatrix}$$

Thus, $\neg(A_1 \mid A_2 \mid A_3 \mid A_4 \mid A_5 \mid A_6) = 11111011111$, and middle port $j = 0$ is selected for the transposition. This results in $\pi_1 = [2,5,4,6,8,10,1,3,0,7,9]$ and $\pi_2 = [6,1,0,3,4,2,7,8,9,10,5]$, which are 4-legal decompositions of $\pi$.

## REFERENCES


[1] N. McKeown, "Optics inside routers," in *Proc. ECOC*, 2003.

[2] R. Ramaswami and K. Sivarajan, *Optical Networks: A Practical Perspective*. San Mateo, CA: Morgan Kaufmann, 1998.

[3] J. Gripp, M. Dulek, J. Simsarian, A. Bhardwaj, P. Bernasconi, O. Laznicka, and M. Zirngibl, "Optical switch fabrics for ultra-high-capacity IP routers," *J. Lightw. Technol.*, vol. 21, no. 11, pp. 2839–2850, Nov. 2003.

[4] R. Tucker, "The role of optics and electronics in high-capacity routers," *J. Lightw. Technol.*, vol. 24, no. 12, pp. 4655–4673, Dec. 2006.

[5] C. Dragone, "An $N \times N$ optical multiplexer using a planar arrangement of two star couplers," *IEEE Photon. Technol. Lett.*, vol. 3, no. 9, pp. 812–815, Sep. 1991.

[6] N. Caponio, A. Hill, F. Neri, and R. Sabella, "Single layer optical platform based on WDM/TDM multiple access for large scale 'switchless' networks," *Eur. Trans. Telecommun.*, vol. 11, no. 1, pp. 73–82, 2000.

[7] A. Bhardwaj, J. Gripp, J. Simsarian, and M. Zirngibl, "Demonstration of stable wavelength switching on a fast tunable laser transmitter," *IEEE Photon. Technol. Lett.*, vol. 15, no. 7, pp. 1014–1016, Jul. 2003.

[8] M. Maier and M. Reisslein, "AWG-based metro WDM networking," *IEEE Commun. Mag.*, vol. 42, no. 11, pp. S19–S26, Nov. 2004.

[9] H. Takahashi, K. Oda, and H. Toba, "Impact of crosstalk in an arrayed-waveguide multiplexer on $N \times N$ optical interconnection," *J. Lightw. Technol.*, vol. 14, no. 6, pp. 1097–110, Jun. 1996.

[10] R. Gaudino, G. G. Castillo, F. Neri, and J. Finochietto, "Simple optical fabrics for scalable terabit packet switches," in *Proc. IEEE ICC*, May 2008, pp. 5331–5337.

[11] V. Mikhailov, C. Doerr, and P. Bayvel, "Ultra low coherent crosstalk, high port-count free-space wavelength router," in *Proc. OFC*, Mar. 2003, vol. 1, pp. 257–258.

[12] M. Maier, M. Reisslein, and A. Wolisz, "High-performance switchless WDM network using multiple free spectral ranges of an arrayed-waveguide grating," *SPIE Terabit Opt. Netw., Archit,, Control, Manage. Issues*, vol. 4213, pp. 101–112, 2000.

[13] M. Rodelgo-Lacruz, C. López-Bravo, F. J. González-Castano, and H. J. Chao, "Practical scalability of wavelength routing switches," in *Proc. IEEE ICC*, 2009, pp. 1–5.

[14] A. Bianco, D. Hay, and F. Neri, "Crosstalk-preventing scheduling in AWG-based cell switches," in *Proc. IEEE GLOBECOM*, 2009, pp. 1–7.

[15] C. Chang, D. Lee, and Y. Jou, "Load balanced Birkhoff-Von Neumann switches, part I: One-stage buffering," *Comput. Commun.*, vol. 25, no. 6, pp. 611–622, 2002.

[16] I. Keslassy and N. McKeown, "Maintaining packet order in two-stage switches," in *Proc. IEEE INFOCOM*, 2002, pp. 1032–1041.

[17] C. Chang, D. Lee, and Y. Shih, "Mailbox switch: A scalable two-stage switch architecture for conflict resolution of ordered packets," in *Proc. IEEE INFOCOM*, 2004, pp. 1995–2006.

[18] H. Lee, "A two-stage switch with load balancing scheme maintaining packet sequence," *IEEE Commun. Lett.*, vol. 10, no. 4, pp. 290–292, Apr. 2006.

[19] I. Keslassy, "Load balanced router," Ph.D. dissertation, Stanford University, Stanford, CA, 2004.

[20] H. Chao and B. Liu, *High Performance Switches and Routers*. Hoboken, NJ: Wiley, 2007.

[21] X. Li and M. Hamdi, "On scheduling optical packet switches with reconfiguration delay," *IEEE J. Sel. Areas Commun.*, vol. 21, no. 7, pp. 1156–1164, Sep. 2003.

[22] V. Alaria, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri, "Design of switches with reconfiguration latency," in *Proc. IEEE ICC*, Jun. 2006, vol. 6, pp. 2599–2605.

[23] B. Towles and W. Dally, "Guaranteed scheduling for switches with configuration overhead," *IEEE/ACM Trans. Netw.*, vol. 11, no. 5, pp. 835–847, Oct. 2003.

[24] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge, MA: MIT Press, 2001.

[25] J. Rosser and L. Schoenfeld, "Approximate formulas for some functions of prime numbers," *Illinois J. Math*, vol. 6, pp. 64–97, 1962.


**Andrea Bianco** (M'98–SM'09) received the Dr.Ing. degree in electronics engineering and the Ph.D. degree in telecommunications engineering from Politecnico di Torino, Torino, Italy, in 1986 and 1993, respectively.

He is an Associate Professor with the Electronics Department, Politecnico di Torino. He has coauthored over 130 papers published in international journals and presented in leading international conferences in the area of telecommunication networks. His current main research interests are in the fields of protocols and architectures for all-optical networks and switch architectures for high-speed networks.

Dr. Bianco has been Guest or Co-Guest Editor of several special issues in international journals, including the *IEEE Communications Magazine* and *Computer Networks*. He was Technical Program Co-Chair of High Performance Switching and Routing (HPSR) 2003 and 2008, Design of Reliable Communication Networks (DRCN) 2005, and the ICC 2010 ONS Symposium. He was a TPC member of several conferences, including IEEE INFOCOM, IEEE GLOBECOM, and IEEE ICC.

**David Hay** (M'09) received the B.A. (summa cum laude) and Ph.D. degrees in computer science from the Technion—Israel Institute of Technology, Haifa, Israel, in 2001 and 2007, respectively.

He is currently a Post-Doctoral Research Scientist with the Department of Electrical Engineering, Columbia University, New York, NY. From 1999 to 2002, he was with IBM Haifa Research Labs, Haifa, Israel. During the summer of 2006, he interned with the Data Center Business Unit of Cisco Systems, San Jose, CA. He was also a Post-Doctoral Fellow with the Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel, from 2007 to 2008, and with the Department of Electrical Engineering, Politecnico di Torino, Torino, Italy, from 2008 to 2009. His main research interests are algorithmic aspects of high-performance switches and routers—in particular, QoS provisioning, competitive analysis, and packet classification.

**Fabio Neri** (M'98–SM'08) received the Dr.Ing. and Ph.D. degrees in electrical engineering from Politecnico di Torino, Torino, Italy, in 1981 and 1987, respectively.

He is a Full Professor with the Electronics Department, Politecnico di Torino, leading research activities on optical networks and switching architectures. He coordinated the participation of his research group to several national Italian research projects. He was involved in several European projects on WDM networks and was the coordinator of the FP6 Network of Excellence e-Photon/ONe on optical networks, which involved 40 European institutions. He has coauthored over 200 papers published in international journals and presented in leading international conferences. His current research interests are in the fields of performance evaluation of communication networks, high-speed and all-optical networks, packet-switching architectures, discrete event simulation, and queuing theory.

Prof. Neri served in the Editorial Board of the IEEE/ACM TRANSACTIONS ON NETWORKING and is Co-Editor-in-Chief of the *Optical Switching and Networking Journal*. He was General Co-Chair of the 2001 IEEE Local and Metropolitan Area Networks (IEEE LANMAN) Workshop and the 2002 and 2007 IFIP Working Conference on Optical Network Design and Modeling (ONDM). He was a member of the technical program committees of several conferences, including IEEE INFOCOM, IEEE GLOBECOM, and IEEE ICC.