Efficient Bandwidth Allocation in the NEPHELE Optical/Electrical Datacenter Interconnect

K. Christodoulopoulos, K. Kontodimas, A. Siokis, K. Yiannopoulos, and E. Varvarigos

***Abstract*—The NEPHELE data center interconnection network relies on hybrid electro-optical top-of-rack switches to interconnect servers over multi-wavelength optical rings. The bandwidth of the rings is shared, and an efficient utilization of the infrastructure calls for coordination in the time, space, and wavelength domains. To this end, we present offline and incremental dynamic resource assignment algorithms. The algorithms are suitable for implementation in a software defined network control plane, achieving efficient, collision-free, and on demand capacity use. Our simulation results indicate that the proposed algorithms can achieve high utilization and low latency in a variety of traffic scenarios that include hot spots and/or rapidly changing traffic.**

***Index Terms*—Dynamic resource allocation; Matrix decomposition; Scheduling; Slotted and synchronous operation; Time-wavelength-space division multiplexing.**

I. Introduction

The widespread availability of cloud applications to billions of end users and the emergence of platform- and infrastructure-as-a-service models rely on concentrated computing infrastructures, the data centers (DCs). DCs typically comprise of a large number of interconnected servers running virtual machines. As traffic within the DC (east–west) is higher than incoming/outgoing traffic, and both are expected to continue to increase [1], DC networks (DCNs) play a crucial role. High throughput, scalable, and energy/cost efficient DCN networks are required to fully harness DC potential.

State-of-the-art DCNs are based on electronic switching in fat-tree topologies [2]. Fat-trees tend to underutilize resources, require a large number of cables, and suffer from poor scalability and low energy efficiency [3,4]. To reduce cost, fat-trees are typically oversubscribed (e.g., 1:4), and do not offer full bisection bandwidth (FBB) that may be needed for certain applications. Application-driven networking [5,6], an emerging trend, would benefit from a network that flexibly allocates capacity where needed.

To cope with the shortcomings of fat-trees, many recent works proposed hybrid electrical/optical DCN, a survey of which is presented in Ref. [7]. The authors of Refs. [8,9] proposed a DCN in which heavy long-lived (elephant) flows are selectively routed over an optical circuit switched (OCS) network, while the rest of traffic goes through the electronic packet switched (EPS) network. These solutions rely on the identification of elephant flows, which is rather difficult, while it was observed that such long-lived heavy flows are not typical [4], making it difficult to sustain high OCS utilization. Instead, a high connectivity degree is needed [4]. To enable higher connectivity, Ref. [10] proposed and prototyped a very dense hybrid DCN that also supports multi-hop connections, along with a custom built control stack. The authors measured the total delay, including control plane and OCS hardware reconfiguration (microelectromechanical system—MEMS—switches), to be of the order of hundreds of milliseconds. Multi-hop routing was exploited anew as *shared* optical circuits in Ref. [11], where an OpenFlow (OF)-based control plane was developed [12], showing that circuit sharing reduces the effect of slow OCS reconfigurations.

Other proposed DC interconnects completely lack electrical switches. Proteus, an all-optical DCN architecture based on a combination of wavelength selective switches (WSSs) and MEMS was presented in Ref. [13]. Again, multi-hop is used to achieve high utilization. However, it is still hard to compensate the MEMS slow reconfiguration times through sophisticated control. References [14,15] introduced hybrid OCS and optical packet/burst switching (OPS/OBS) architectures, controlled using SDN. Various other architectures based on OPS/OBS were proposed [7,16] (and references therein). However, OPS/OBS technologies are not yet mature, so the current target could be small-scale networks with limited upgradability potential.

The authors of Ref. [17] presented a hybrid DCN architecture called Mordia, which uses WSS to provide switching times of 11.5 μs. Mordia operates in a dynamic slotted manner to achieve high connectivity. However, the scalability of

Manuscript received June 26, 2017; revised October 6, 2017; accepted October 22, 2017; published November 29, 2017 (Doc. ID 300915).

K. Christodoulopoulos (e-mail: kchristo@mail.ntua.gr), K. Kontodimas, and E. Varvarigos were with the Department of Computer Engineering and Informatics, University of Patras, Greece. They are now with the School of Electrical and Computer Engineering, National Technical University of Athens, Greece.

A. Siokis is with the Department of Computer and Engineering and Informatics, University of Patras, Greece.

K. Yiannopoulos is with the Department of Informatics and Telecommunications, University of Peloponnese, Greece.

E. Varvarigos is also with the Department of Electrical and Computer Systems Engineering, Monash University, Australia.

https://doi.org/10.1364/JOCN.9.001145

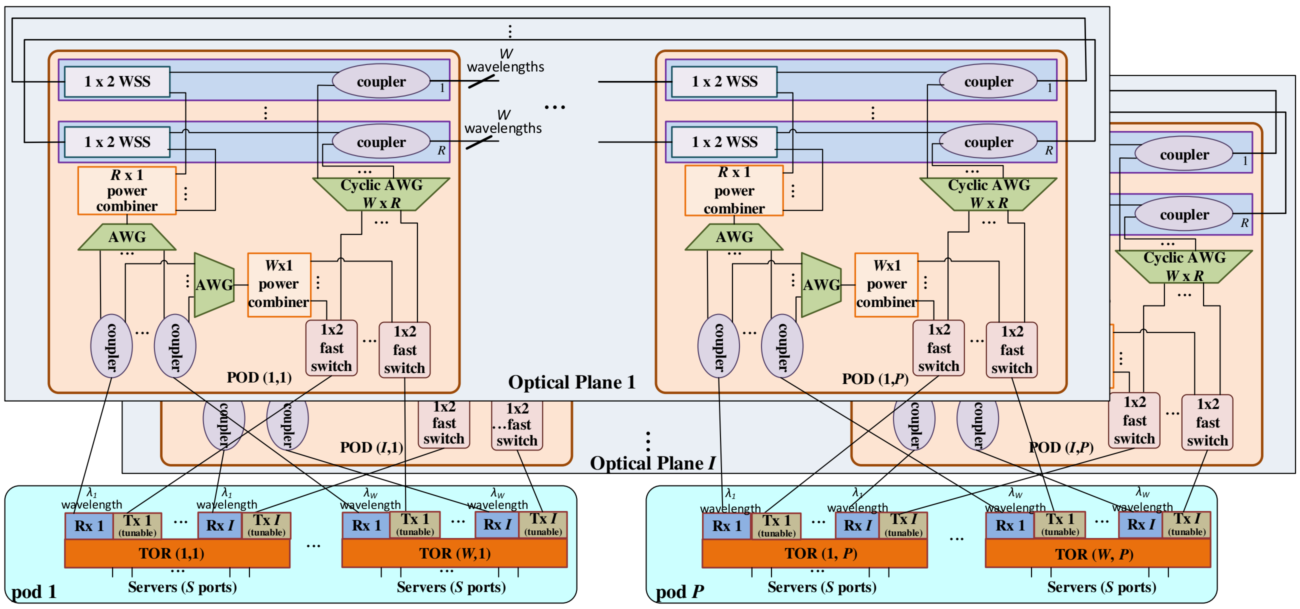

Fig. 1. NEPHELE DCN architecture.

Mordia is limited as it uses a single wavelength division multiplexing (WDM) ring whose capacity can accommodate only a few racks, while resource allocation algorithms exhibit high complexity and cannot scale to large DCs.

The European project NEPHELE is developing an optical DCN that leverages hybrid electrical/optical switching with SDN control to overcome current datacenter limitations [18]. To enable dynamic and efficient sharing of optical resources and collision-free communication, NEPHELE operates in a synchronous slotted manner. Timeslots are used for rack-to-rack communication and are assigned dynamically, on a demand basis, so as to attain efficient utilization, leading to both energy and cost savings. Moreover, multiple wavelengths and optical planes are utilized to implement a scalable and high capacity DC network infrastructure.

The NEPHELE network relies on WSSs, which are faster than the MEMS used in Refs. [8–11] and more mature than the OPS/OBS used in Refs. [14–16]. The fast switching times, along with the dynamic slotted operation, provide high and flexible connectivity. Compared to Mordia [17], which also relies on WSSs, NEPHELE is more scalable: it consists of multiple WDM rings, re-uses wavelengths, and utilizes cheap passive routing components and scalable scheduling schemes. The latter is the major contribution of this paper, which presents fast scheduling algorithms to meet NEPHELE dynamic reconfiguration requirements.

Regarding resource allocation, scheduling problems similar to those addressed in this paper were studied in the past for satellite and ATM switches [19–26]. Indeed, one can view the entire NEPHELE multi-ring DCN as a large distributed switch. The key difference of our work is that we consider huge network installations and dynamic time-division multiplexing (TDM) operation; thus strict optimality is not the objective, but we rather target low complexity. We also encounter certain internal collision constraints that are particular to the NEPHELE architecture (Section VI), and thus we need to extend previous TDM algorithms appropriately. Apart from [17], which considers dynamic TDM operation, a somehow relevant algorithmic work is [27], where the authors present an integrated optical network-on-chip (NoC) based on a ring topology and micro-ring resonators (MRs). The key difference with the NEPHELE network is that MRs target NoC and small networks, where propagation and control plane delays are negligible. Thus, scheduling does not take place in periods, as in NEPHELE, but on a per slot basis as in electronic switches [28].

The remainder is organized as follows. In Section II, we describe the NEPHELE architecture. In Section III, we describe the dynamic resource allocation problem. In Section IV we provide a set of algorithms to solve it. In Section V we analyze the resource allocation constraint induced by the NEPHELE architecture. In Section VI we evaluate the performance of the proposed algorithms. Finally, we provide our conclusions in Section VII.

II. NEPHELE Electrical/Optical Interconnect

NEPHELE is a hybrid electrical/optical DCN architecture, built out of POD and top-of-rack (TOR) switches. Figure 1 describes the NEPHELE DCN. The NEPHEL DCN is divided into P pods[1] of racks. A pod consists of I POD switches and W TOR switches, interconnected as follows: each TOR switch listens to a specific wavelength (thus, by design, the number of wavelengths equals the number W of racks in a pod) and has I ports. Each port is connected to a different one of the I POD switches. A rack

[1]The term “pod” refers to the cluster of racks, and “POD” to a NEPHELE pod switch.

consists of S (computer, storage, or memory) servers. The TOR is a hybrid electrical/optical switch and each of the S servers of the rack connects to it via a link. Thus, a TOR switch has S ports facing "south" to the servers.

POD switches are interconnected via WDM rings to form "optical planes." An optical plane consists of a single POD switch per pod (for a total of P POD switches in the DCN) connected with R fiber rings. Each fiber ring carries WDM traffic over W wavelengths (W, by design, equals the number of racks in the pod), propagating in the same direction. There are I identical and independent/parallel (in the sense that traffic entering a plane stays in it until the destination) optical planes. In total, there are $I \cdot P$ POD switches, $W \cdot P$ TOR switches, and $I \cdot R$ fiber rings.

We now explain how communication is performed in the NEPHELE DCN (Fig. 1). The key routing concept is that *each TOR switch listens to a specific wavelength* (out of W available), and *wavelengths are re-used* among pods. The NEPHELE TORs use tunable transmitters that are tuned according to the desired destination. Each TOR employs virtual output queues (VOQs) per TOR destination ($W \cdot P$ VOQ per TOR) to avoid head-of-line blocking.

Traffic in the form of an optical signal originating from a port (plane) of a TOR switch enters a POD switch and is switched through a fast 1×2 space switch according to locality: if the traffic is destined to a TOR in the pod (intra-pod), it remains within the POD switch; otherwise, it is routed to the rings and to the next POD switch. Local intra-pod traffic enters a $W \times 1$ power combiner, located after the 1×2 space switch, and then a $1 \times W$ arrayed waveguide grating (AWG). The AWG passively routes the traffic, depending on the used wavelength, to the desired destination.

Inter-pod traffic is routed via the fast 1×2 switch toward a $W \times R$ cyclic AWG (CAWG) followed by couplers that combine the CAWG outputs into the R fiber rings. The $W \times R$ CAWG has a passive routing functionality, with the incoming signal being routed to the output port (ring):

$$r = (w_s + w_d - 1) \bmod R, \tag{1}$$

where $1 \leq w_s \leq W$ is the input port (the index of the source rack in the source pod, which equals its listening wavelength), $1 \leq w_d \leq W$ is the wavelength that has to be used to reach the specific destination (thus, also equal to the index of the destination rack in the destination pod), and "mod" denotes the modulo operation. In the simple (not cyclic) $1 \times W$ AWG, the output depends only on the used wavelength. So, the traffic enters the ring according to the CAWG function, propagates in the same ring through intermediate POD switches, and is dropped at the destination pod. These routing decisions are applied by setting appropriately the wavelength selective switches (WSSs) in the related POD switches. The WSSs can select whether traffic passes through or drops on a per-fiber, per-wavelength, and per-slot basis. Thus, each intermediate POD sets the corresponding WSS to the pass state, while at the destination the related WSS is set to the drop state. The drop ports of all the WSSs—corresponding to all the rings—are connected to a power combiner and a $1 \times W$ AWG. So again, the traffic once dropped is passively routed to the desired TOR according to the wavelength used.

Following the above, wavelengths are statically assigned to racks, to simplify optical routing, and are re-used for efficient operation. Conflicts on the WDM rings are avoided in the time and space (plane) domains. Regarding the time domain, NEPHELE operates in a synchronous slotted manner that closely resembles the operation of a single (huge and distributed) time-division multiple access (TDMA) switch. In particular, NEPHELE maintains the timeslot component of TDMA, but timeslots are not statically assigned; instead, a central scheduler dynamically assigns them based on traffic needs, enabling efficient utilization of the resources. However, making scheduling decisions on a per-timeslot basis is prohibitive, due to high communication and processing latency. Instead, it is both more efficient and less computationally demanding to perform resource allocation periodically, so that scheduling decisions are made for periods of T timeslots; this approach facilitates the aggregation and suppression of monitoring and control data and also absorbs traffic peaks.

From the control plane perspective, configurable components are the tunable transmitters (I per TOR switch), the 1×2 optical switches (W per POD switch), and the WSSs (R per POD switch). The timeslot duration is lower bounded by the slowest component, which is the WSS with a switching time of about 10 μs [17]. This is reserved as a guardband and the timeslot is taken to be 0.2 ms, so that the network exhibits 95% efficiency. The amount of data transmitted during a timeslot equals the wavelength transmission rate times the timeslot duration (i.e., $0.2\ \text{ms} \times 10\ \text{Gbps} = 2\ \text{Mbits}$) and will be referred to as a data unit (DU). This is also the switching granularity of a NEPHELE DCN. A reference number for T is 80 timeslots, corresponding to a period of 16 ms.

The existence of I parallel planes provides an additional domain, the space, to resolve conflicts: each timeslot of each plane can be independently allocated. We will refer to a timeslot/plane combination as a generic (time)slot, implying that the space and time domains are interchangeable in NEPHELE.

Variations of the above described baseline NEPHELE architecture include cases where each TOR does not listen to a specific wavelength. One such variation will be given in Section V. Still, the NEPHELE routing function remains similar: the transmitter needs to select the appropriate wavelength, which is pre-calculated based on certain parameters (the source, the destination, the plane, etc., as opposed to only the destination in the baseline architecture), while the WSSs are configured according to that wavelength mapping.

Since a CAWG is used to route the W wavelengths on R rings, we must have $W \geq R$ in order for the CAWG to be able to use all R egress ports. This is a system constraint. We can also derive the required conditions for achieving FBB assuming that the NEPHELE network is nonblocking (see Sections IV and V). We say that a DC interconnect has FBB if for any bisection of the servers in two equal partitions, each server of one partition is able to communicate at

TABLE I
FULLY FLEDGED NEPHELE NETWORK PARAMETERS

Parameter	W	P	S	I
Value	80	20	20	20

full rate with any server of the other partition. Since a TOR supports S servers, the number of PODs connected to a TOR must be at least $I \geq S$, so that all servers of a TOR can communicate with servers outside their rack. Considering the whole network, there are $P \cdot W \cdot S$ server ports, whereas the overall capacity in the POD-to-POD network is $I \cdot R \cdot W$. Thus, for FBB, we need to have $I \cdot R \cdot W \geq P \cdot W \cdot S => I \cdot R \geq P \cdot S$. Assuming $I = S$, the FBB requirement becomes $R \geq P$. More flexibility is obtained by increasing the number of planes I. In the presence of traffic locality, the FBB requirement can be relaxed to support larger DCs. Table I presents target values satisfying the above constraints (including FBB) for a *fully fledged* NEPHELE network using commodity off-the-shelf (COTS) equipment and a reference DC size (32K servers).

III. BANDWIDTH ALLOCATION IN NEPHELE

NEPHELE architecture exploits the SDN concept that decouples data and control planes through open interfaces, enabling programmability of the networking infrastructure. NEPHELE utilizes an optical network with I optical planes, R fibers/plane and W wavelengths/fiber to interconnect the TOR switches in P pods. As discussed above, the network operates in a slotted and synchronous manner. A key functionality of the NEPHELE SDN controller is the coordination of the networking resources usage, including the timeslot/plane dimension. Thus, an important building block of the SDN controller is the *scheduling engine*, which allocates resources to communicating TOR pairs in a centralized, periodic, and on-demand manner.[2]

Recall that the number of racks per pod is equal to the number of wavelengths, and each rack listens to a specific wavelength. A TOR switch s is thus defined by a unique pair $s = (p_s, w_s)$, where p_s, $1 \leq p_s \leq P$, is the index of the pod it belongs to, and w_s, $1 \leq w_s \leq W$, is the rack index within the pod (w_s is also the wavelength on which TOR s receives data). It will sometimes be convenient to represent the TOR switch by the scalar index $s = p_s \cdot (W - 1) + w_s$ instead of the pair representation (p_s, w_s); as the mapping between the two representations is one-to-one, we will use, with a slight abuse of notation, the same symbol s to stand for the TOR itself, the scalar index, and the pair representing it.

We assume that a Data period consists of T timeslots, and we denote by $Q(n)$ the *queue matrix* for period n. The queue matrix $Q(n)$ is of size $(W \cdot P) \times (W \cdot P)$, and element $Q_{sd}(n)$ corresponds to the number of DUs that are queued at the start of period n at source TOR s and have as destination TOR d, with $s = p_s \cdot (W - 1) + w_s$, $d = p_d \cdot (W - 1) + w_d$, $1 \leq w_s, w_d \leq W$, and $1 \leq p_s, p_d \leq P$. That is, $Q_{sd}(n)$ is the number of DUs in VOQ (s, d) at the start of period n. Since the scheduling problems of the different wavelengths are *not* independent, we will avoid breaking this matrix per wavelength.

Two operation modes are envisioned for the NEPHELE network: (i) application-aware and (ii) feedback-based networking. The former approach [5,6] assumes that applications communicate to the NEPHELE SDN controller (or via the DC orchestrator) their topology and traffic requirements. In that case, the queue matrix is constructed from input from the applications. The latter, feedback-based, mode assumes that the central controller collects (monitors) data from the TOR queues [9] to build the queue matrix. We can also have a hybrid application-aware and feedback-based network. In the following we focus on the feedback-based approach, which is the hardest of the two from the control and scheduling viewpoint. The analysis and the proposed algorithms are applicable with minor changes to application-aware and hybrid operation.

Recall that matrix $Q(n)$ records the queue sizes at the start of period n. We denote by $A(n)$ the matrix of arrivals at the queues during period n and by $\mathcal{S}(n)$ the schedule calculated for period n. Element $Q_{sd}(n)$ denotes the DUs in the (s, d) queue at the start of period n, element $A_{sd}(n)$ the DU arrivals during period n, and element $\mathcal{S}_{sd}(n)$ the DUs scheduled to be transferred from s to d during period n. We will describe in the next section the way schedule $\mathcal{S}(n)$ is calculated.

Under feedback-based operation, the NEPHELE network operates in *two parallel cycles*:

1) data communication cycles of T timeslots (also referred to as a *Data period*), where the actual communication between TORs takes place, and
2) resource allocation cycles of duration C (measured in Data periods of T timeslots), where control information is exchanged. If the duration of the resource allocation process is not fixed, C is an upper bound on it.

Resource allocation cycle n corresponds to Data period n, and computes the schedule $\mathcal{S}(n)$ to be used during that period. Note, however, that the schedule is computed based on information that was available C periods earlier than the Data period to which the resource allocation cycle corresponds (and is applied). Thus, $\mathcal{S}(n)$ is a function of $Q(n - C)$, i.e.,

$$\mathcal{S}(n) = f(g[Q(n - C)]), \qquad (2)$$

where $\hat{Q}(n) = g[Q(n - C)]$ is the function that creates the *estimated queue matrix* $\hat{Q}(n)$ from $Q(n - C)$ upon which the schedule is calculated, and f is the scheduling algorithm. When $C > 1$ period (control delay larger than the Data period), a new resource allocation cycle still starts every Data period. So, there are C resource allocation cycles (or virtual control planes) running in parallel. For

[2]In the following, the terms "bandwidth allocation," "resource allocation," and "scheduling" will be used interchangeably.

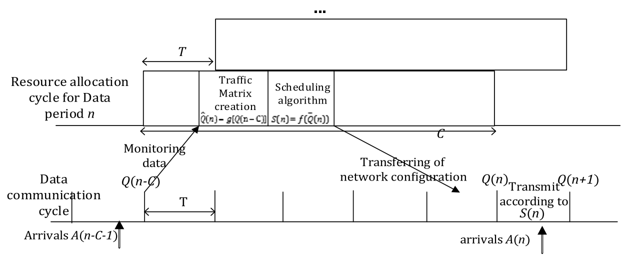

Fig. 2. NEPHELE resource allocation and data cycles.

determining the schedule $\mathcal{S}(n)$ to be used during Data period n:

a) the traffic matrix engine of the SDN controller collects the queue sizes from the TORs to build $Q(n-C)$ and runs the queue estimation algorithm g to create the *estimated queue matrix* $\hat{Q}(n) = g[Q(n-C)]$,
b) the scheduling engine of the SDN controller runs algorithm f to calculate the schedule $\mathcal{S}(n) = f(\hat{Q}(n))$, and
c) the SDN controller communicates the scheduling output $\mathcal{S}(n)$ to the NEPHELE data plane devices (POD and TOR switches) to be used during Data period n.

Figure 2 shows the NEPHELE resource allocation and data cycles (control and data plane, respectively). As discussed, there is a delay between the two cycles: schedule $\mathcal{S}(n)$ applied in Data cycle n is computed based on queue matrix $Q(n-C)$, since it takes C periods to compute and reach the data plane devices. The queue evolution is described by

$$Q(n+1) = Q(n) + A(n) - \mathcal{S}(n), \tag{3}$$

where $\mathcal{S}(n) = f(g[Q(n-C)])$. The value C does not affect the achievable throughput, as long as scheduling decisions are efficient (more on that later), but affects the traffic delay. The control plane delay C depends on many factors, on the execution time of the scheduling algorithm, and the delay of the control protocol carrying information from TORs to the SDN controller (if monitoring is assumed) and from the SDN controller to the data plane devices. Both delays depend on the network size and the choice of the Data period T.

For scheduling decisions to be efficient, the scheduling matrix $\mathcal{S}(n)$, computed based on an estimated queue matrix $\hat{Q}(n)$, which in turn is calculated by $Q(n-C)$, should be a "good" scheduling to be used during Data interval n. This is true when $\hat{Q}(n)$ is a good approximation of $Q(n)$. For slowly and medium changing traffic, we expect calculations made for previous periods to be valid. In estimating $\hat{Q}(n)$ from $Q(n-C)$, it is possible to also use statistical predictions, filters, and other (notably application-aware) methods to improve performance. Moreover, it is possible for the scheduler to fill unallocated resources in $\mathcal{S}(n)$ by opportunistic transmissions, which can have collisions or be collision free (e.g., nodes agree to use slots in lexicographic order, mimicking static TDM, which under heavy load is efficient). Finally, the overall scheme is "self-correcting": if some queues are not served for some periods due to poor scheduling and their size grows due to new arrivals, this will be communicated with some delay to the controller, and the queues will eventually be served. A study of the effect of the control plane delay and ways to mitigate it is part of our future plans.

In the following we will focus on the scheduling problem in the NEPHELE network. We start from the estimated queue matrix $\hat{Q}(n)$ and devise fast algorithms to calculate the schedule $\mathcal{S}(n)$ [function f in Eq. (2)]. For reference we can assume that we calculate the estimated queue matrix [function g in Eq. (2)] as $\hat{Q}(n) = A(n-C-1) + \hat{Q}(n-1) - \mathcal{S}(n-1)$, where we acknowledge that due to control plane delay C, the central scheduler has access to (delayed) arrival information $A(n-C-1)$ instead of $A(n)$. This corresponds to the case where the schedule $\mathcal{S}(n)$ calculated on $\hat{Q}(n)$ serves the arrived traffic $A(n-C-1)$, plus a correction equal to traffic not served in the previous period $\hat{Q}(n-1) - \mathcal{S}(n-1)$.

We now describe the form of the schedule $\mathcal{S}(n)$. The scheduling engine provides the TOR pairs that communicate *during each timeslot and for each optical plane* within the upcoming Data period. Note that wavelengths and rings are dependent resources; the selected wavelength is determined by the destination, and the ring depends on the source and destination according to Eq. (1). Thus, in NEPHELE the allocated resources are the timeslots and the optical planes ($I \cdot T$ in total), or the *generic slots*, as stated previously.

The scheduling algorithm takes the estimated queue matrix $\hat{Q}(n)$ and decomposes it (fully or, if not possible, partially) into a sum of $I \cdot T$ *permutation matrices* $P(n,g)$, $g = 1, \ldots, I \cdot T$, each corresponding to a generic slot. A permutation matrix is binary of size $(W \cdot P) \times (W \cdot P)$; an entry $P_{sd}(n,g)$ equals "1" if a DU is to be transferred from TOR s to TOR d during the gth generic slot of period n, and "0" otherwise. In other words, $P_{sd}(n,g)$ identifies if one DU at the d-VOQ of TOR s will be transmitted during the gth generic slot of period n.

A permutation matrix determines a configuration of the network for a specific generic slot. For the communication to be contention free, the *scheduling constraints* SC_1, SC_2, and SC_3 that are summarized in Table II should be satisfied. In particular, the first two constraints, SC_1 and SC_2, ensure that each TOR transmits to and receives from at most one TOR per generic slot. Constraints SC_1 and SC_2

TABLE II
SCHEDULING CONSTRAINTS (SC)[a]

Constraint ID	Description
SC_1	$\sum_s P_{sd}(n,g) \leq 1$
SC_2	$\sum_d P_{sd}(n,g) \leq 1$
SC_3	$P_{sd}(n,g) + P_{s'd'}(n,g) \leq 1$, for $p_s < p_{s'} < p_d$ or $p_s < p_{d'} < p_d$, and $(w_{s'} - w_s) \bmod R = 0$

[a] $P_{sd}(n,g) = 1$, $s = p_s(W-1) + w_s$, and $d = p_d \cdot (W-1) + w_d$ indicate that one DU is scheduled for transfer from source TOR (w_s, p_s) to destination TOR (w_d, p_d) in the gth generic slot of period n.

are relevant to all TDMA-like architectures and are readily enforced by the decomposition process.

The third constraint, SC_3, is related to the (not nonblocking character of the) architecture, and particularly, it is a result of the usage of static routed CAWGs as opposed to dynamically configured components. To better illustrate SC_3, assume that a source TOR (w_s, p_s) communicates with a destination TOR (w_d, p_d). This communication takes place over the optical ring that is calculated from Eq. (1), and it occupies wavelength w_d on the ring segment between p_s and p_d. If another source TOR $(w_s, p_{s'})$ within the aforementioned ring segment (i.e., $p_s < p'_s < p_d$) concurrently communicates with destination TOR $(w_d, p_{d'})$, a collision will occur irrespective of the destination pod $(p_{d'})$, since it occupies the same ring and wavelength. A similar contention will occur if the destination pod lies in the initial ring segment (i.e., $p_s < p'_d < p_d$), irrespective of the source pod. Note that SC_3 is alleviated for $R \geq W$, which, however, leads to underutilization of rings. Moreover, the effect of the lack of the nonblocking property for the architecture (when seen as a huge switch), or equivalently the existence of SC_3, is small, and will be discussed in Sections V and VI.

The set $P(n,g), g = 1, 2, \ldots, I \cdot T$, of permutation matrices comprise schedule $\mathcal{S}(n)$, which records information for all generic slots of period n. The permutation matrices $P(n,g)$ are stored as sparse matrices, each with $W \cdot P$ entries. Similarly, $\mathcal{S}(n)$ is sparse with $I \cdot T \cdot W \cdot P$ entries.

IV. Scheduling Algorithms

Having described NEPHELE DCN operation, we now proceed to present a set of NEPHELE scheduling algorithms. We assume that we start with the estimated queue matrix $\hat{Q}(n)$ and calculate the schedule $\mathcal{S}(n)$ [function f in Eq. (2)]. To target both static and dynamic resource allocation scenarios, we developed two classes of scheduling algorithms: (i) *offline* and (ii) *incremental*. Offline algorithms, given in Subsection IV.A, take the estimated queue matrix $\hat{Q}(n)$ and compute schedule $\mathcal{S}(n)$ "from scratch." Incremental algorithms, given in Subsection IV.C, use the previous schedule $\mathcal{S}(n-1)$ and update it based on traffic changes to obtain $\mathcal{S}(n)$. Offline algorithms are better suited for semi-static traffic, take longer to execute, and achieve better utilization; incremental algorithms are faster and can handle dynamic scenarios.

A. *Offline Scheduling*

As discussed above, offline scheduling decomposes the matrix $\hat{Q}(n)$ into a set of permutation matrices $\mathcal{S}(n) = \{P(n,g)\}$, $g = 1, 2, \ldots, I \cdot T$, without taking into account the previous decomposition. We start by presenting the optimal offline scheduling algorithm.

The decomposition of $\hat{Q}(n)$ can be performed in an optimal manner following the well-known Hall's theorem (an integer version of the Birkhoff–Von Neumann theorem [24]). We define the *critical sum* $[[H\hat{Q}(n)] = h$ of matrix $\hat{Q}(n)$ as the maximum of its row sums and column sums. Then the following theorem holds:

Hall's Theorem: An integer matrix of critical sum h can be written as the sum of h permutation matrices.

The following algorithm calculates the optimal decomposition of matrix $\hat{Q}(n)$:

1. Find a matrix of nonnegative integers $E(n)$ so that matrix $M(n) = \hat{Q}(n) + E(n)$ is a *perfect matrix* with critical sum $H[M(n)] = H[\hat{Q}(n)] = h$. A perfect matrix has the sum of each row and each column equal to the critical sum. An algorithm to obtain $E(n)$ is found in Ref. [21].
2. Treat $M(n)$ as a (bipartite) graph adjacency matrix and obtain a maximum matching $j \to p(j), j = 1, 2, \ldots, P \cdot W$. This matching can then be represented as a permutation matrix $P(n,i)$, whose $[j, p(j)]$ entries are equal to 1, and all other entries are 0.
3. Find the weight c_i as the smallest element of $M(n)$ that corresponds to a nonzero entry in $P(n,i)$.
4. Repeat $P(n,i)$ for c_i times in the schedule and update $M(n) = M(n) - c_i \cdot P(n,i)$.
5. If $M(n)$ is not equal to zero, repeat steps 2–4. Otherwise, an optimal decomposition for $M(n)$ has been found.
6. Set the entries of the dummy matrix $E(n)$ to zero.

Steps 2–4 are repeated h times at most and we have that $\sum_i c_i = h$. Note that the decomposition of an integer matrix as a sum of h permutation matrices is not unique and that the permutation matrices in the decomposition of $M(n)$ are full rank (corresponding to full utilization of the $I \cdot T$ generic slots), while those in the decomposition of $\hat{Q}(n) = M(n) - E(n)$ may not be full rank [leaving some generic slots unused, namely, the entries of $E(n)$, and available for opportunistic transmissions]. In general, decompositions that use a limited number of permutations, each carrying a considerable amount of traffic c_i, are preferable as they result in fewer reconfigurations in the NEPHELE switches.

The preceding algorithm assumes that the critical (row or column) sum is constrained, but this will not always be the case. The arrival matrix $A(n)$ corresponds to traffic created by the servers and aggregated at the related TOR switches in period n. Since one link connects a server to the TOR, the server sends to its TOR switch at most 1 DU during a timeslot. Therefore, the row sums of $A(n)$ are at most $S \cdot T$. Some of $A(n)$'s column sums, however, may be larger than that, e.g., in the presence of hotspot destinations. Note that the capacity connecting a TOR to the destination servers can transfer $S \cdot T$ DU, and this is the same for all DCNs. So hotspot problems, where traffic toward some TORs (columns of A) exceeds the available capacity, are present in all DCNs and not only in NEPHELE.

We could, in principle, devise flow control mechanisms to guarantee that the critical sum of $A(n)$ satisfies $H[A(n)] \leq S \cdot T$. Using an entry flow control mechanism between servers and source TORs, like the "stop and go" queueing proposed in Ref. [29], which limits (smoothens) the entry of DUs toward the destinations, we can enforce the column sum to be less than $S \cdot T$. In particular, each source TOR can check the destination TOR d of the packets forwarded to it by the source servers and, through a backpressure mechanism, guarantee that packets equivalent to at most $(S \cdot T)/(W \cdot P)$ DUs are destined for each destination during a period of duration T. Such a source flow control mechanism, however, may be too restrictive, unnecessarily, and introduce large entry delays, as packets are queued at the servers, outside the interconnection network. To relax somewhat the constraint, a credit-based flow control mechanism can be used at the pod level, where each source POD is given $W_d = (S \cdot T)/P$ credits for each destination TOR d per period, which it can distribute to the TORs below it that can, in turn, distribute them to the servers. This would relax considerably the input flow control constraints and the corresponding delays at the servers, but requires a clever mechanism for distributing credits.

Even if a flow control mechanism is not present, the column sums will be on average $\leq S \cdot T$, assuming the destinations of packets are uniformly distributed on average. Actually, the critical sum will be $\leq S \cdot T$ not only on average but also with high probability, if the network operates at less than full load. Finally, note that TCP flow control smoothens the traffic to a given destination. Since the downstream links from a TOR to the servers can support up to $S \cdot T$ DU per Data period, the previous condition will tend to hold with high probability in a DC network that employs TCP.

Based on the previous discussion we conclude that in the "typical case" the column sums of the arrival matrix $A(n)$ will be $\leq S \cdot T$ and so will also be its critical sum (since the row sums are always $\leq S \cdot T$). In that case, the schedule $\mathcal{S}(n)$, that is calculated based on $\hat{Q}(n) = A(n - C)$, assuming $S \leq I$, can be chosen so as to completely serve all the arrivals in $A(n - C)$ in the available $I \cdot T$ generic slots. Note that in the reference FBB network scenario we assume $S = I$ and so we will interchangeably use S and I in the following. Thus, in this case, all packets generated in a Data period will be served C periods later, emptying the queue from such packets. So the delay in the NEPHELLE network is upper bounded by C periods when appropriate input flow control is used, or with high probability when the load is far enough from full load. Thus, in the typical case, NEPHELLE provides both *full throughput and delay guarantees*.

In the more general case where the critical sum of $\hat{Q}(n)$ is not bounded by $I \cdot T$, we stop when we find the first $I \cdot T$ permutations, while the traffic $Q(n) - \mathcal{S}(n)$ that is not served is fed to produce the estimated matrix for next period $\hat{Q}(n+1)$. Fairness and priority issues can also be handled with small extensions to the above process without a requirement for additional flow control.

B. Complexity of Offline Scheduling and Stability

For general traffic, we define the load intensity between source destination TOR pair (s, d) as

$$\rho_{sd}(A) = E(A_{sd})/(I \cdot T), \tag{4}$$

where $E()$ stands for expected value and $0 \leq \rho_{sd}(A) \leq 1$ for a FBB NEPHELE network $(S = I)$. The *load intensity matrix* $P(A)$ is defined as the matrix with $\rho_{sd}(A)$ entries. The row sums of $P(A)$ are always less than or equal to 1, while for a stable network (finite queues), the column sums should also be less than or equal to 1.

Necessary condition for stability: For the NEPHELE network to be stable, the load intensity matrix $P(A)$ should be at most a double stochastic matrix.

When the previous condition does not hold, it is impossible to find a schedule to serve the queues of NEPHELE in a stable manner. It is thus up to the DC orchestrator to allocate tasks to servers so that their communication requirements meet this constraint. Our target is to provide schedules that can serve any (long-term) stable matrix $P(A)$.

We define the *average* network load $\rho(A)$ (also represented by ρ) for arrival matrices A as the scalar

$$\rho(A) = \rho = \sum_{sd} \rho_{sd}(A)/(P \cdot W) = \sum_{sd} E(A_{sd})/(I \cdot T \cdot P \cdot W), \tag{5}$$

and $0 \leq \rho(A) \leq 1$. The quantity $\rho \cdot P \cdot W \cdot I \cdot T$ equals the average of the entries of arrival matrix A during a period (or, equivalently, $\rho \cdot P \cdot W \cdot I$ is the average number of arrivals per timeslot and TOR-to-TOR pair).

Besides the load, another parameter that is important in characterizing the arrival process and the algorithmic complexity is the *arrival matrix density* $\delta(A)$, which is complementary to the sparsity of A. In particular, if we define the indicator function $1()$, as $1(x) = 1$, when $x > 0$ and 1, otherwise, then the density $\delta(A)$ of matrix A is defined as

$$\delta(A) = E\left[\sum_{sd} 1(A_{sd})\right]/(W \cdot P)^2, \tag{6}$$

where $E[\sum_{sd} 1(A_{sd})]$ is the average number of nonzero entries of A and, clearly, $0 \leq \delta(A) \leq 1$. In other words, $\delta(A)$ is the fraction of nonzero entries of A. Then, the number of nonzero entries $M(A)$ is given by $M(A) = \delta(A) \cdot (W \cdot P)^2$.

In the worst case, the optimal algorithm described earlier executes a maximum matching algorithm $I \cdot T$ times (uniform traffic). Finding a maximum matching can be time consuming, and even the well-known Hopcroft–Karp bipartite graph matching algorithm [25] exhibits complexity of $O(M(A) \cdot \sqrt{W \cdot P})$, where $M(A)$ is the number of nonzero elements in A. The number of different matches is $\rho \cdot I \cdot T$, and thus the complexity of the optimal offline algorithm is $O(\rho \cdot \delta \cdot I \cdot T \cdot (W \cdot P)^{\frac{5}{2}})$.

An indicative example of the execution time required for optimal decomposition with the Birkhoff–Von Neumann and Hopcroft–Karp algorithms is shown in Fig. 3, for a fully fledged NEPHELE network (parameters listed in Table I). The algorithm was developed in MATLAB and the simulations were performed on an Intel Core i5 laptop. Figure 3 plots the average execution time of the optimal decomposition algorithm against the load ρ and density δ, which are shown to range from tens of seconds to minutes.

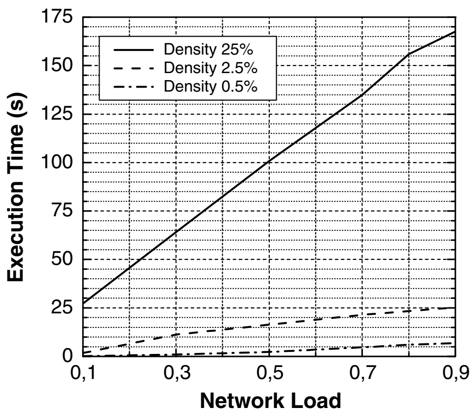

Fig. 3. Average execution time of optimal decomposition algorithm as a function of load ρ and arrival matrix density δ.

In these simulations, the traffic was created as follows: at each period, each source TOR communicated with $\delta \cdot W \cdot P$ uniformly chosen TOR destinations by transmitting a total number of $\rho \cdot I \cdot T$ DUs.

Based on the above result, and given the size of a fully fledged FBB NEPHELE DCN (Table I), we deduce that, even with an optimized software and hardware environment, the optimal algorithm would only be viable under a static resource allocation scenario, where traffic remains unchanged for prolonged periods. The requirement for dynamic resource allocation can be pursued via non-optimal algorithms that employ maximal rather than maximum matchings, at the expense of blocking at high loads. To this end, we also developed faster offline heuristics of reduced complexity and performance quite close to the optimal. In particular, we developed a greedy offline algorithm of complexity $O(\rho \cdot (W \cdot P)^2 \cdot I \cdot T)$, which is linear in the size of the problem [note that the number of DUs to be scheduled is $O(\rho \cdot (W \cdot P)^2)$ and the number of resources is $O(I \cdot T)$]. For brevity, we do not discuss this algorithm, as it still cannot meet dynamic resource allocation requirements, but describe a variation of it in the next subsection. To further reduce scheduling complexity, we have to exploit the variations (temporal and spatial) of traffic, as is done in the incremental scheduling algorithms of the next subsection.

C. Incremental Scheduling Algorithms for Locality Persistent Traffic

It is evident from the previous results that offline scheduling is not suitable for bursty traffic. Measurements in commercial DCs indicate that application traffic can be relatively bursty, with flows activating/deactivating within milliseconds [4]. Although traffic can be bursty, it tends to be highly *locally persistent*: a server tends to communicate with a set of destinations that are located in the same rack or the same cluster/pod [4]. This is due to the way applications are placed in DCs, each occupying only a small fraction of the DC.

TOR switches in NEPHELE aggregate the flows of the servers in a rack, smoothening out the burstiness of individual flows, especially considering locality persistent traffic. To formally define locality persistency, we define the *arrival matrix difference* as $D_A(n) = A(n) - A(n-1)$, the load $\rho(|D_A(n)|)$, and the density $\delta(|D_A(n)|)$ of the difference by replacing A with $|D_A|$ in Eqs. (5) and (6), where $|\cdot|$ stands for the entrywise absolute value.

Locality Persistency Property: holds if

$$\delta(|D_A(n)|) \ll 1. \tag{7}$$

We also define the *estimated queue matrix difference* as $D_{\hat{Q}}(n) = \hat{Q}(n) - \hat{Q}(n-1)$. Note that when arrivals have the locality persistency property [i.e., Eq. (6) holds], then, in view of the Section III discussion, we also expect $\delta(|D_{\hat{Q}}(n)|) \ll 1$. For example, in the typical case where $\hat{Q}(n) = A(n-C-1) + \hat{Q}(n-1) - \mathcal{S}(n-1)$, the persistency property of A also holds for the estimated matrix $\hat{Q}$.

Motivated from this observation, we propose and investigate *incremental scheduling*, i.e., rely on the previous schedule to calculate the new one. The expected benefit is that we need to update only specific elements of the permutation matrices of the decomposition of $\hat{Q}(n+1)$, corresponding to traffic that has changed, but there is no need to modify the rest of the elements.

To be more specific, let $\hat{Q}(n)$ be the estimated queue matrix and $\mathcal{S}(n)$ be the schedule produced at period n. To compute schedule $\mathcal{S}(n+1)$ for the next period $n+1$ with estimated queue matrix $\hat{Q}(n+1)$, we perform the following:

1. Compute $D_{\hat{Q}}(n) = \hat{Q}(n+1) - \hat{Q}(n)$ [Fig. 4(b)].
2. Split $D_{\hat{Q}}(n)$ into $D^+(n)$ and $D^-(n)$, where

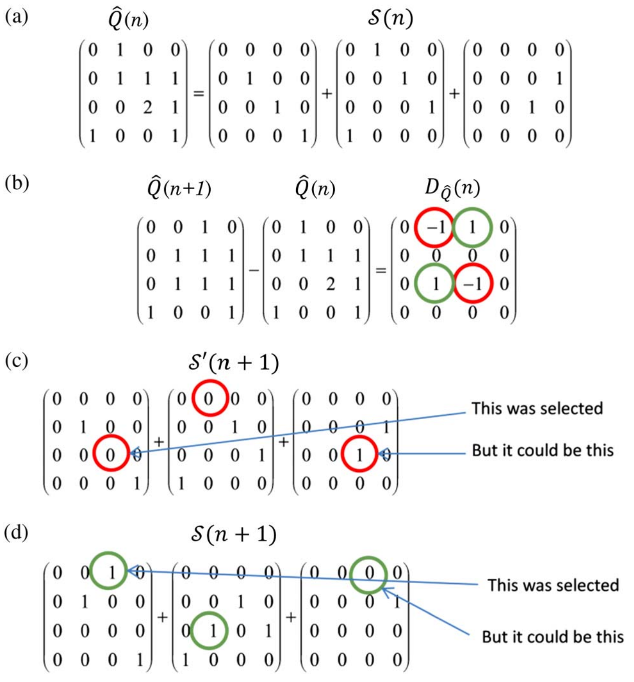
(a)
$\hat{Q}(n)$
$\mathcal{S}(n)$
$\begin{pmatrix}0&1&0&0\\0&1&1&1\\0&0&2&1\\1&0&0&1\end{pmatrix} = \begin{pmatrix}0&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix} + \begin{pmatrix}0&1&0&0\\0&0&1&0\\0&0&0&1\\1&0&0&0\end{pmatrix} + \begin{pmatrix}0&0&0&0\\0&0&0&1\\0&0&1&0\\0&0&0&0\end{pmatrix}$
(b)
$\hat{Q}(n+1)$
$\hat{Q}(n)$
$D_{\hat{Q}}(n)$
$\begin{pmatrix}0&0&1&0\\0&1&1&1\\0&1&1&1\\1&0&0&1\end{pmatrix} - \begin{pmatrix}0&1&0&0\\0&1&1&1\\0&0&2&1\\1&0&0&1\end{pmatrix} = \begin{pmatrix}0&-1&1&0\\0&0&0&0\\0&1&-1&0\\0&0&0&0\end{pmatrix}$
(c)
$\mathcal{S}'(n+1)$
This was selected
But it could be this
(d)
$\mathcal{S}(n+1)$
This was selected
But it could be this

Fig. 4. Concept of incremental scheduling.

- $D^+(n)$ denotes the matrix consisting only of the positive entries of difference matrix $D_{\hat{Q}}(n)$, and
- $D^-(n)$ denotes the matrix consisting only of the negative entries of difference matrix $D_{\hat{Q}}(n)$.

3. Use algorithm A_1 to free entries of $\mathcal{S}(n)$ according to matrix $D^-(n)$ and obtain the half-filled schedule, denoted as $\mathcal{S}'(n+1)$ [Fig. 4(c)].
4. Use algorithm A_2 to add entries in $\mathcal{S}'(n+1)$ (half-filled schedule) according to $D^+(n)$ to obtain the current period's schedule $\mathcal{S}(n)$ [Fig. 4(d)].

Algorithm 1: Linear Greedy Algorithm

```
Given: $D^+(n)$, $\mathcal{S}'(n+1)$, $TC(n)$, $RC(n)$, $P$, $W$, $T$, $I$
Return $\mathcal{S}(n+1)$, $TC(n+1)$, $RC(n+1)$
1:  $\mathcal{S}(n+1) = \{P(n+1,g)\} \leftarrow \mathcal{S}'(n+1)$;
2:  $TC(n+1) \leftarrow TC(n)$, $RC(n+1) \leftarrow RC(n)$;
3:  for $s \leftarrow 1$ to $P \cdot W$ do
4:    for $d \leftarrow 1$ to $P \cdot W$
5:      slots $\leftarrow D^+_{sd}(n)$;
6:      $g \leftarrow 1$;
7:      while $g \leq T \cdot I$ and slots $> 0$
8:        if $TC_s(n+1,g) = 0$ and $RC_d(n+1,g) = 0$ then
9:            $P_{sd}(n+1,g) = 1$;
10:           $TC_s(n+1,g) = 1$; $RC_d(n+1,g) = 1$;
11:           slots $\leftarrow$ slots – 1;
12:       end if
13:       $g \leftarrow g + 1$;
14:     end while
15:   end for
16: end for
```

The complexity of incremental scheduling is $O(\delta(|D_{\hat{Q}}|) \cdot \rho \cdot I \cdot T \cdot (W \cdot P)^2)$, where $\delta(|D_{\hat{Q}}|) \ll 1$ in view of the persistency property of Eq. (6) and the related discussion.

The above describes the core of the incremental algorithms. In the first two algorithms that we will present, we used a greedy algorithm A_1 in Step 3 to free entries that works as follows: by iterating each element of $D^-(n)$, we find the last permutation matrix of $\mathcal{S}(n)$ that serves that element, and we free that entry (set it to zero). This algorithm frees sequentially the scheduled resources for the demands whose traffic was reduced, leaving the entries that satisfy the current traffic. Regarding step 3, we present three A_2 schemes, each corresponding to a different incremental scheduling algorithm.

1) Linear-Time Greedy Incremental Heuristic: The greedy heuristic is a non-optimal algorithm running in linear time to the size of the problem and the number of generic slots per period. The greedy heuristic can be used as an offline or as an incremental algorithm. In the following we focus on the incremental case. The algorithm takes as input the difference traffic matrix $D_{\hat{Q}}(n)$ [or $\hat{Q}(n+1)$ in offline]. It follows steps 1–3 described above, so that it finds the half-filled schedule, denoted as $\mathcal{S}'(n+1)$ and the positive difference matrix $D^+(n)$. By iterating on each non-zero element of $D^+(n)$, it greedily finds the available generic slots to use. This is done by taking into account constraints SC_1 and SC_2, of Table II, which ensure that at each generic slot a TOR can send to or receive from only one other TOR. Data structures $TC(n)$ and $RC(n)$ are used to keep track of these two constraints. In particular, element $TC_s(n,g)$ [or $RC_d(n,g)$] records whether the transmitter (or receiver) at source s (or destination d, respectively) and generic slot g is active or not. The pseudo-code of the incremental greedy algorithm is given in Algorithm 1.

2) Sublinear Greedy Incremental Heuristic: The sublinear greedy algorithm is a variation of the linear greedy heuristic, but it schedules blocks of DUs instead of DUs. In particular, an integer $k = O(I)$ is chosen and used to calculate the *block estimated queue matrix* $\hat{Q}^k(n) = \frac{\hat{Q}(n)}{k}$ (in our implementation we chose $k = 5$, and I was a multiple of 5). The purpose of this procedure is to reduce the amount of load to be scheduled, within a span of $T \cdot \frac{I}{k}$ generic slots, speeding up the scheduling process roughly by a factor of k. The block estimated queue matrix is treated as the estimated queue matrix, while applying the previous greedy algorithm. The schedule produced by the greedy algorithm is reproduced k times, in order to cover the initial traffic. As expected, the speedup obtained comes at a cost: dummy DUs are introduced when the ceiling function is applied, which are allocated some generic slots, reducing the resource usage. In particular the load overhead introduced is

$$\text{Number of dummy DUs} = \sum_{s,d}\left(k \cdot \left(\frac{\hat{Q}_{s,d}(n)}{k}\right) - \hat{Q}_{s,d}(n)\right). \quad (8)$$

In order for the algorithm to run in sublinear time (a speedup of roughly k is expected), some filtering has to be applied to $\hat{Q}(n)$ in such a way that its critical sum is at most $\frac{T \cdot I}{k}$ after the division, rather than $T \cdot I$. This process takes place in the estimated queue matrix creation module and requires at least linear time to complete. These two operations, however, namely, the estimated queue matrix creation and the scheduling, are performed by different modules. The queue matrix creation module can start executing while receiving monitoring information; once the block estimated queue matrix is created, the scheduling algorithm is executed in sublinear time. We consider this to be technically feasible for the NEPHELE's architecture.

3) Randomized Heuristic: A randomized variation of the greedy heuristic was also implemented for an incremental resource assignment. Randomized operation avoids the greedy first find approach, aiming to increase (on average) the traffic that is served [30]. The algorithm follows an approach similar to the four steps presented at the start of this subsection: it receives as input the previous period's schedule $\mathcal{S}(n)$, the estimated queue matrix $\hat{Q}(n+1)$, and calculates the schedule $\mathcal{S}(n+1)$. In the first phase, it examines the previous period's permutation matrices $P(n,g)$ against the traffic they can carry in the new period and discards any $P(n,g)$ that carries less traffic than $\sum_{sd}\hat{Q}_{sd}(n+1)/(I \cdot T)$, expecting that a new randomized allocation could provide a better solution for the corresponding generic slot. The $P(n,g)$ that carry their fair share of the traffic load are then subtracted from $\hat{Q}(n+1)$:

1. If the subtraction of a $P(n,g)$ leaves no negative entries, then the $P(n,g)$ is kept unaltered in $\mathcal{S}'(n+1)$.
2. Whenever negative entries occur, the corresponding entries on both $P(n,g)$ and $\hat{Q}(n+1)$ are set to zero, and the updated $P(n,g)$ is used in $\mathcal{S}'(n+1)$.

The previous steps calculate (i) the updated set of permutations $\mathcal{S}'(n+1)$, by skipping the calculation of $D^-(n)$, and (ii) the positive change matrix $D^+(n)$, which is the $\hat{Q}(n+1)$ matrix after the subtractions. In this case, $D^+(n)$ includes the new connections, the old connections with increased traffic, and the old connections that belonged to discarded permutations. Then the entries of $D^+(n)$ are distributed randomly on $\mathcal{S}'(n+1)$ following the algorithm below:

1. Select a random destination TOR (column) d of $D^+(n)$.
2. Find the m active source TORs for destination d, corresponding to rows $\{s_1, s_2, \ldots, s_m\}$ of the non-zero entries in column d, and re-arrange them randomly.
3. For each row s_k in the randomized arrangement:
   a. Find the existing $P(n+1,g)$ that are available for the (s_k,d) communication (by checking the related scheduling constraints—using the data structures $TC(n)$ and $RC(n)$, as discussed in Subsection IV.C.1).
   b. If the number of available $P(n+1,g)$ is greater than the $D^+_{s_k,d}(n)$ entry (i.e., more resources are available than those required), randomly select the required number; otherwise select all of them.
4. Repeat steps 1–3 for all columns of $D^+(n)$.

Finally, if any traffic remained in $D^+(n)$ and not all the $I \cdot T$ permutations are utilized, then the algorithm performs a final round where it repeats steps 1–4, with the only difference being that new permutations are considered to be initially available to all connections.

V. Architecture-Related Constraint

The resource allocation problem at hand is quite similar to scheduling problems for TDM satellite or ATM crossbar switches [19–23]. Scheduling constraints SC$_1$ and SC$_2$ are common, but constraint SC$_3$ (Table II) is new and is a result of specific architecture choices, and particularly of using static routed (C)AWGs instead of reconfigurable components. This design choice, which was decided to keep the cost and routing complexity low, results in a NEPHELE DCN (when seen as a huge switch connecting TORs) losing its nonblocking character even for $I = S$. In the previous section, we described algorithms that operate without taking into account SC$_3$, whose effect is studied here.

To evaluate the performance under the additional constraint SC$_3$, we extended the incremental greedy heuristic Subsection IV.C.1) to account for SC$_3$. The algorithm to be described is referred to as the *ring-segment greedy heuristic*. To be more specific, consider a transmission from source TOR $s = (w_s, p_s)$ to destination TOR $d = (w_d, p_d)$ at generic slot g (timeslot t over optical plane i), where $p_s < p_d$ without loss of generality. Such a communication is represented in the schedule by $P_{sd}(n,g) = 1$. Under the baseline architecture of Section II that uses $W \times R$ CAWGs at the input of the rings, this communication uses wavelength w_d and ring $r_{sd} = [(w_s + w_d - 1) \bmod R]$, according to Eq. (1). So, the communication from s to d captures the ring-wavelength resource, indexed

$$l_{sd} = [(w_s + w_d - 1) \bmod R] \cdot W + w_d. \tag{9}$$

Resource l_{sd} is actually captured only for the segment of the ring that is between pods p_s and p_d and can be used by other connections if they use non-overlapping segments of the ring. SC$_3$ constrains that s to d communication cannot take place simultaneously with communication from $s' = (w_{s'}, p_{s'})$ to $d' = (w_{d'}, p_{d'})$, with $p_s < p_{s'} < p_d$ or $p_s < p_{d'} < p_d$ and $w_{d'} = w_d$ and $(w_{s'} - w_s) \bmod R = 0$ (see Table II).

The ring-segment greedy heuristic algorithm keeps track of the utilization of the ring-wavelength resources and the specific ring segments utilized. To accommodate the communication from s to d at generic slot g, we need to check whether ring-wavelength resource l_{sd} is used between pods p_s and p_d. If it is not used, we reserve it to block any future conflicting communication. The data structure records for each generic slot $g = 1, 2, \ldots, I \cdot T$ the ring-wavelength resource $l = 1, 2, \ldots, R \cdot W$ and the specific ring segment it uses (P ring segments in the worst case), resulting in size $O(P \cdot R \cdot W \cdot I \cdot T)$. This data structure can be similar to $TC(n)$ and $RC(n)$ used to keep track of SC$_1$ and SC$_2$ (Subsection IV.C.1), which, however, are of size $O(P \cdot W \cdot I \cdot T)$. Specifically, line 8 of the pseudo-code of Algorithm 1, should also search for maximum P ring segments, which increases the complexity.

The worst case traffic pattern is obtained when we have the maximum number of conflicting communication pairs defined by SC$_3$, and all of them carry maximum traffic. Regarding the constraint on the overlapping of ring segments, there are P such conflicting (s,d) pairs for unidirectional traffic (p_1 to p_P, p_2 to $p_1, \ldots, p_{P-1}$ to p_P, p_P to p_{P-1}), and since they are in different pods they can have maximum traffic equal to $\hat{Q}_{sd}(n) = S \cdot T$. In this case, we require $I = P \cdot S$ planes to fully serve the worst case traffic. Such worst case traffic is, of course, highly improbable to occur. Still, our simulations show that the throughput is affected even in the average case when considering SC$_3$, while the execution time increases, since we need to account for the ring segment utilization.

We developed two solutions to address this problem: the first extends the incremental greedy algorithm of Subsection IV.C.1, considering in a more coarse way the utilization of the ring-wavelength resources, while the second relies on a variation of the architecture that uses spectrum-shifted optical planes.

A. Full-Ring Greedy Heuristic

In the first solution, called the full-ring greedy heuristic algorithm, communication from s to d is taken to occupy the entire ring-wavelength resource l_{sd}, i.e., the whole ring and not only the segment between pods p_s and p_d. This reduces the size of the data structure needed to $O(R \cdot W \cdot I \cdot T)$ and improves the execution time over the ring-segment greedy heuristic discussed above, sacrificing somewhat the (already lower) throughput performance.

B. Spectrum-Shifted Optical Planes

A problem of the baseline NEPHELE architecture is that, if two communicating source–destination pairs, (s, d) and (s', d'), conflict over an optical plane, by using the same ring-wavelength resource $l_{sd} = l_{s'd'} = l$, they will use the same resource l and conflict over all planes. This problem affects all planes, so we have available only the time domain (T) to resolve such conflicts, as opposed to having both the plane and time dimensions (all $I \cdot T$ generic slots), resulting in lower performance. To address this, we developed an architecture variation where the optical planes are *spectrum shifted*. To be more specific, in the architecture of Fig. 1, traffic for destination TOR $d = (w_d, p_d)$ always uses wavelength w_d. The main idea of spectrum-shifted optical planes is to make the ring-wavelength in Eq. (9) depend on plane i and on other source/destination location parameters. This proposed variation uses the desired passive components, i.e., (C)AWGs, instead of replacing them by reconfigurable ones that would significantly increase the cost, due to their high radix.

The goal is to design *all-pair conflict-free optical planes*, so that TOR pairs conflicting on some optical plane do not conflict on another one. There are various ways to achieve that, such as permuting the rings between pods, or varying the CAWG routing function by changing the way CAWGs are coupled/added to the rings. One such efficient solution is to replace the $1 \times W$ AWG connected to the drop ports of the WSSs with an $P \times W$ CAWG connected as follows: We connect the drop ports of all the WSSs of plane i and pod p through the $R \times 1$ power combiner to input port $z(i, p)$, $1 \leq z(i, p) \leq P$ of the $P \times W$ CAWG. The W output ports of the $P \times W$ CAWG are connected to the TORs as before. We make the wavelength $w_{sd}(i)$, used for communication between source $s = (w_s, p_s)$ and destination $d = (w_d, p_d)$ over plane i depend on s, d, and i, as opposed to the baseline architecture where this was fixed and equal to w_d. Considering the routing function of the CAWG, $w_{sd}(i)$ should satisfy the following condition in order to reach the desired destination:

$$(w_{sd}(i) + z(i, p_d) - 1) \bmod W = w_d, \tag{10}$$

where w_d in this equation indicates only the location of the destination TOR in the related pod (and not, as previously, the receiving wavelength), and $z(i, p_d)$ is the input port of the CAWG. The routing function of the CAWG that adds the traffic to the rings at the source gives the ring used:

$$r_{sd}(i) = (w_s + w_{sd}(i) - 1) \bmod R. \tag{11}$$

Then, the ring-wavelenth resource of plane i that is used is

$$l_{sd}(i) = [(w_s + w_{sd}(i) - 1) \bmod R] \cdot W + w_{sd}(i). \tag{12}$$

Consider now another TOR pair communication $s' = (w_{s'}, p_{s'}) \to d' = (w_{d'}, p_{d'})$ on the same plane i. To create conflict, this communication has to use the same wavelength and the same ring with the $s \to d$ communication, i.e.,

$$\begin{aligned} w_{sd}(i) &= w_{s'd'} \wedge (i)(w_s + w_{sd}(i)) \bmod R \\ &= (w_{s'} + w_{s'd'}(i)) \bmod R, \end{aligned} \tag{13}$$

or, equivalently,

$$z(i, p_d) - z(i, p_{d'}) = (w_d - w_{d'}) \bmod W (w_{s'} = w_s) \bmod R. \tag{14}$$

Our goal is to avoid pairs $s \to d$ and $s' \to d'$ to conflict in any other plane. This can be satisfied if $|z(i, p_d) - z(i, p_{d'})| \neq |z(i', p_d) - z(i', p_{d'})|$, for all $1 \leq i\prime < I$, $i\prime \neq i$. Generally, we want that to hold for any conflicting pair of any plane, i.e., we need the following to hold for all $i, (i'i' \neq i)$, all $p_d, p_{d'}$:

$$|z(i, p_d) - z(i, p_{d'})| \neq |z(i', p_d) - z(i', p_{d'})|. \tag{15}$$

Remember that $1 \leq z(i, p) \leq P$, since $z(i, p)$ corresponds to the input port of the $P \times W$ CAWG that the WSSs of pod p at plane i are connected. For a prime number of pods P, one choice (along with others) that satisfies Eq. (14) is

$$z(i, p) = (1 + (p - 1) \cdot (i - 1)) \bmod P. \tag{16}$$

For prime P, with the above function we construct P all-pair conflict-free planes. The number of planes I required to serve any pattern is then $I \geq P$. To see this, assume that we have several conficting pairs on a plane (P is the maximum number of pairs, as discussed previously), and each requires the full capacity (all the timeslots) of the plane. This plane can serve any of those, but the remaining pairs conflicting on that plane are not conflicting on the other $I - 1$ planes. Thus, if $I \geq P$ (which also holds for the reference NEPHELE architecture—Table I), conflicts can be solved using the plane dimension in addition to the timeslot dimension. In that case, the entire NEPHELE network is actually a *nonblocking time–wavelength–space switch*.

If P is not prime (in the reference $P = 20$), the above function constructs all-pair conflict-free planes equal to the smallest divisor (= 2 for the reference architecture). However, even in this case, the conflicts are reduced substantially. The average performance improves when the number of conflicting pairs among the planes is small, and the proposed solution reduces this number. All-pair conflict-free planes mean that this number is zero, which results in the best worst case and average performance.

We rely on simulations to evaluate the performance of our solution for average traffic.

The extensions needed in the scheduling algorithm to account for spectrum-shifted planes are straightforward, and require the calculation of the wavelength based on the source, destination, and plane. This can be done with pre-calculated tables and does not affect the complexity. We also need to use either the ring-segment or the full-ring heuristic algorithm to keep track of ring-wavelength resource utilization. We decided to use the faster full-ring greedy heuristic in the performance evaluation section.

VI. Performance Evaluation

A. Evaluation Without Architecture Constraint SC_3

The proposed incremental scheduling algorithms were evaluated via simulations for various traffic scenarios. We assumed a NEPHELE network with $W = 80$ racks/pod, $P = 20$ pods, $S = 20$ server ports/rack, and $I = S = 20$ optical planes (see Table I), and set $T = 80$ timeslots. We used a custom traffic matrix generator where we could control the following parameters [31]:

1. the average network load $\rho(A)$, defined from Eq. (4) as the ratio of the total traffic over the total capacity. The individual TOR loads $\rho_{sd}(A)$ were generated as independent Gaussian random variables, assuming that a TOR aggregates a large number of TCP/UDP flows. The distribution mean was set equal to the desired load, while its variance was correlated to the load dynamicity $\rho(|D_A|)$;
2. the load dynamicity $\rho(|D_A|)$, defined as the average change in traffic between successive periods;
3. the connection density $\delta(A)$, defined from Eq. (6). Low connection density corresponds to a small number of destinations per source, thus an increased number of traffic hotspots. To accommodate the description of traffic patterns of previous works [4], where TORs systematically prefer to communicate with peers in specific pods, or even the same pod, we further distinguished between intra-POD density $\delta_{\text{in}}(A)$ and inter-POD density $\delta_{\text{out}}(A)$. A *locality* parameter is then defined as the traffic percentage that is destined within the same pod over the total load:

$$l = \frac{\delta_{\text{in}} \cdot W}{\delta_{\text{in}} \cdot W + \delta_{\text{out}} \cdot W \cdot (P - 1)},$$

   given that the local POD comprises W TORs out of $W \cdot P$ that are available in total; and
4. the locality dynamicity $\delta(|D_A|)$, defined as the average number of connections that change from active to inactive and *vice versa* at each period. Traffic exhibits locality persistency [Eq. (7)] when $\delta(|D_A|)$ is low.

To evaluate the proposed algorithms, we developed a simulator in MATLAB. For each simulation instance, we chose to vary one parameter, while the rest of the parameters were set to their default values (Table III). To focus on the performance of the scheduling algorithms, we assumed a resource cycle with $C = 1$, which corresponds to the schedule being calculated within a Data period. We also assumed the reference case where the estimated queue matrix on which the schedule is calculated based on the arrivals: $\hat{Q}(n+1) = A(n-C) + \hat{Q}(n) - \mathcal{S}(n)$. As discussed in Subsection IV.C, this ensures that the persistency property of A is also true for the estimated queue matrix $\hat{Q}$.

TABLE III
Networking Parameters

Parameter	Symbol	Value	Default
Network load	$\rho(A)$	0.1–0.9	—
Intra-POD connection density	$\delta_{\text{in}}(A)$	100%, 25%, 2.5%	25%
Inter-POD connection density	$\delta_{\text{out}}(A)$	25%, 2.5%, 0.5%	2.5%
Load dynamicity	$\rho(\lvert D_A\rvert)$	10%, 1%, 0.1%	1%
Locality dynamicity	$\delta(\lvert D_A\rvert)$	10%, 1%, 0.1%	1%

For each parameter set, we measured a) the additional average queuing latency, i.e., the average number of periods a packet remains buffered in addition to the $C = 1$ period that it takes for the schedule to be calculated, so as to focus on the efficiency of the algorithm and not of the whole control cycle, and b) the schedulimg algorithm's execution time (s) against the network load. We also measured the *maximum network throughput*, defined as the maximum offered load at which the queues and the latency are finite. Thus, the maximum throughput indicates the load that can be transferred by the network under stable operation. Note that maximum throughput is identified in the latency/load graphs as the load at which the latency becomes (assymptotically) infinite.

1) Queuing Latency: Initially, we present the results on the latency. In the first set of simulations, the examined parameter is intra-POD density, which is set to 100% for the results of Fig. 5(a) and to 2.5% for Fig. 5(b); the other parameters were set to their default values (Table III). Figure 5(a) shows that the sublinear greedy heuristic clearly underperforms, as expected, the other two algorithms, resulting in average latency that increases at load 0.7 and becomes (asymptotically) infinite at load 0.8 (= maximum network throughput). The linear greedy heuristic comes next, followed by the randomized heuristic with slightly better performance. In the results of the second set of simulations, shown in Fig. 5(b), the density of intra-POD connections is set very low to 2.5%. The queuing latency of all three algorithms start to increase at load around 0.7. The increase is steeper for the sublinear greedy heuristic, followed by the linear greedy, and finally by the randomized heuristic. The latter two algorithms have very similar performance regarding latency and stability.

Locality considerably impacts the performance. The greedy and the random algorithms are more efficient when the matrix is concentrated in small blocks [δ_{in} high, heavy intra-POD traffic—Fig. 5(a)] than when it is spread out [Fig. 5(b)]. In contrast, the sublinear algorithm underperforms when locality is high; introducing several dummy

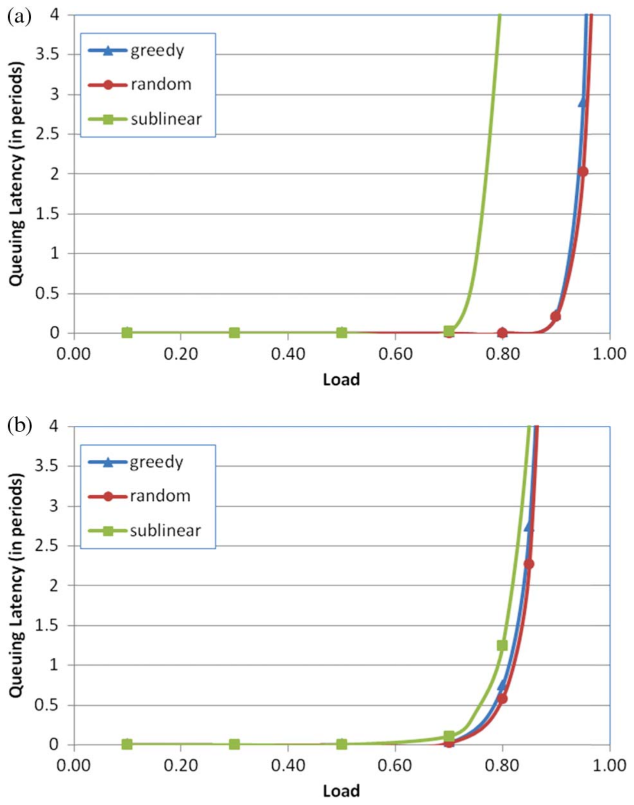

Fig. 5. Average queuing latency resulting from the examined scheduling algorithms, measured in Data periods additional to the control cycle, for intra-pod density δ_{in} equal to (a) 100% (locality 68%) and (b) 2.5% (locality 5%).

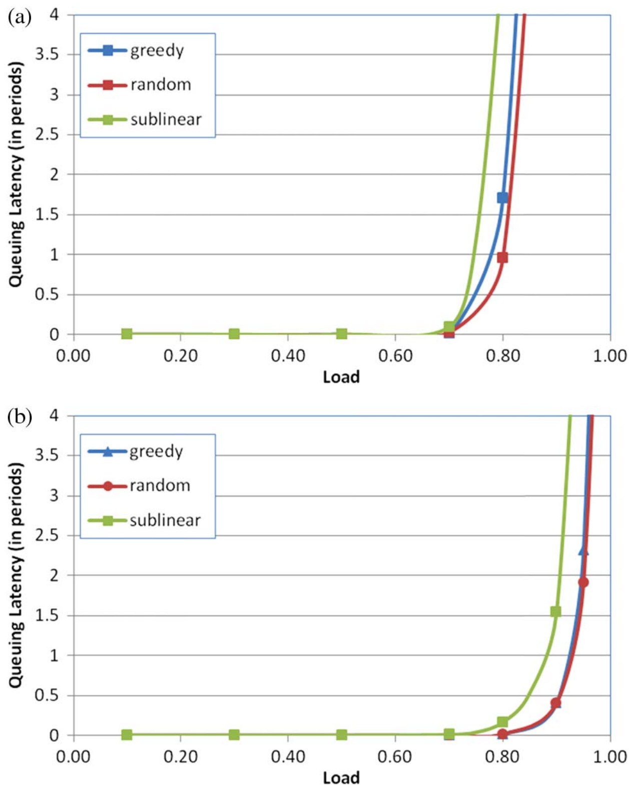

Fig. 6. Average queuing latency resulting from the examined scheduling algorithms, measured in Data periods additional to the control cycle, for locality dynamicity $\delta(|D_A|)$ equal to (a) 0.1% and (b) 10%.

DUs in a small block increases the column sum more than when traffic and the locations of the dummy DUs are spread out.

We next examine the effect of the locality dynamicity parameter $\delta(|D_A|)$. When $\delta(|D_A|) = 0.1\%$ [Fig. 6(a)], all three heuristic algorithms start to induce high latency at network load of about 0.7. As in the previous cases, the queuing latency increase with network load is steeper for the case of the sublinear heuristic, followed by the linear greedy, and then by the randomized heuristic. This is more clear at load 0.8, where the sublinear greedy heuristic is already in the unstable region, while the linear greedy and the randomized heuristic remain stable until load 0.85.

When the locality dynamicity parameter $\delta(|D_A|) = 10\%$ [Fig. 6(b)], all three algorithms improve their results by increasing their maximum throughput (latency asymptote moves to the right). Higher dynamicity reduces the persistency of bad scheduling matrices, improving the performance, but as expected, has negative effects on execution times, as will be discussed in the following.

2) Scheduling Algorithms Execution Times: Next, we present results on the execution times of the considered algorithms. We provide four plots for the same parameters examined in Subsection VI.A.1.

As shown in Fig. 7(a), the algorithms' performance in order of increasing execution times is randomized, linear greedy, and sublinear greedy heuristic. As expected, the average execution times increase with the load. At load 0.8, the randomized heuristic needs an average of 1.5 s to complete. Next comes the linear greedy heuristic with an execution time (at 0.8 load) of about 0.7 s, and last comes the sublinear greedy heuristic with about 0.5 s. These results were expected from the theoretical complexity analysis given in Section IV. The relative order of the algorithms with respect to their execution times remains the same when intra-POD connection density is set to 2.5% [Fig. 7(b)]. The decrease in the execution times for low intra-POD density is due to the fewer connections, each of higher load, which reduces the complexity of all three algorithms. The execution times for different values of locality dynamicity parameter $\delta(|D_A|)$ are depicted in Fig. 8. As expected, by complexity analysis, execution time increases as load and locality dynamicity $\delta(|D_A|)$ increases.

3) Maximum Network Throughput: We now focus on the maximum network throughput achieved by the scheduling algorithms, defined as the load at which the queues and the latency become (asymptotically) infinite and the system becomes unstable. The throughput is examined with respect to two parameters that were not discussed above: (i) the inter-POD connection density δ_{out} and (ii) the load dynamicity $\rho(|D_A|)$. The results are shown in Table IV. We see that the impact of inter-POD connection density δ_{out} is quite significant, since for dense traffic ($\delta_{\text{out}} = 50\%$), the throughput

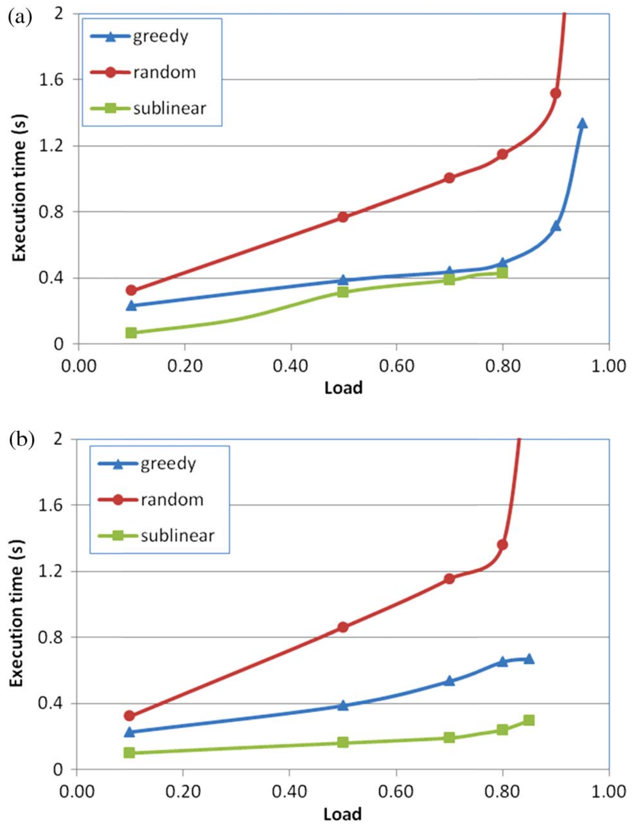

Fig. 7. Execution times of the algorithms for intra-pod density δ_{in} equal to (a) 100% (locality $l = 68\%$) and (b) 2.5% (locality $l = 5\%$).

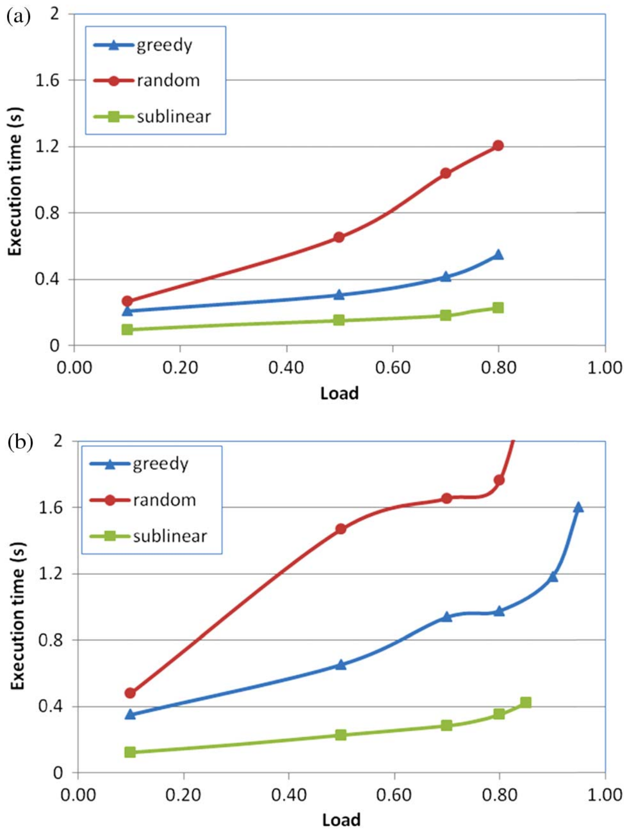

Fig. 8. Execution times of the algorithms considered for locality dynamicity $\delta(|D_A|)$ equal to (a) 0.1% and (b) 10%.

reaches about 0.97, while for sparse traffic, it drops to 0.85 at most. The reason is similar to the one discussed for the role of intra-POD density. It should be noted that, for dense inter-POD connections ($\delta_{out} = 0.5\%$), the sublinear greedy heuristic is unstable even at low traffic loads, since it wastes too much capacity. This should be expected, as small and spread demands result in many entries that create many dummy DUs, thus wasting network capacity. Regarding load dynamicity, we consider the cases $\rho(|D_A|) = 0.1\%$ and $\rho(|D_A|) = 10\%$. We observe that this parameter does not affect substantially the throughput, nor the execution time. The throughput performance of all the algorithms was similar, with the sublinear greedy heuristic being slightly worse and faster (lower than 0.4 s in almost all cases).

B. Evaluating the Effect of the SC$_3$ Constraint

We evaluated the performance of the NEPHELE network under the architecture constraint SC$_3$ and also for the architecture variation that uses the spectrum-shifted planes. In particular, we assessed the performance for

(a) reference architecture/greedy (no SC$_3$),
(b) reference architecture/segment-ring greedy,
(c) reference architecture/full-ring greedy, and
(d) spectrum-shifted planes/segment-ring greedy.

In all examined cases, the number of planes was the same ($I = 20$). Case (a) was examined in the previous subsections and is used here as a reference. The network of case (a) can achieve maximum throughput; that is, it can accommodate any traffic if an optimal algorithm is used. The network of cases (b) and (c) has worst-case traffic that requires more (20 times) planes, while case (d) also requires more planes than the I available, but lower than those of cases (b) and (c). The probability of generating the worst-case traffic is extremely low, but cases (b) and (c) have several traffic instances that require more than I planes, while for case (d) this probability is low. Note, however, that we use a heuristic (incremental greedy) and thus blocking is expected even for case (a).

Figure 9(a) shows the latency for density between pods $\delta_{out} = 50\%$, corresponding to $l = 2.5\%$ locality (default $\delta_{in} = 25\%$). Such a low locality results in heavy utilization

TABLE IV
MAXIMUM THROUGHPUT OF ALGORITHMS CONSIDERED AS A FUNCTION OF INTER-POD CONNECTION DENSITY δ_{out} AND LOAD DYNAMICITY $\rho(|D_A|)$

Parameter	Symbol	Value	Linear	Randomized	Sublinear
Inter-POD	δ_{out}	50%	0.97	0.97	0.4
connection density		0.5%	0.85	0.85	0.82
Load dynamicity	$\rho(|D_A|)$	0.1%	0.92	0.93	0.9
		10%	0.88	0.88	0.87

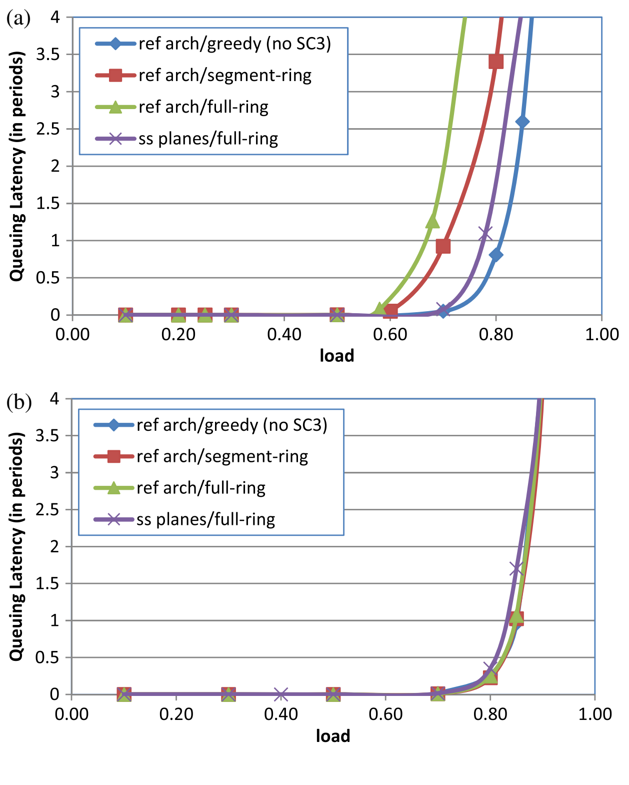

Fig. 9. Latency (in periods) as a function of load for density between pods (a) $\delta_{\text{out}} = 50\%$ ($l = 2.5\%$) and (b) $\delta_{\text{out}} = 0.5\%$ ($l = 70\%$).

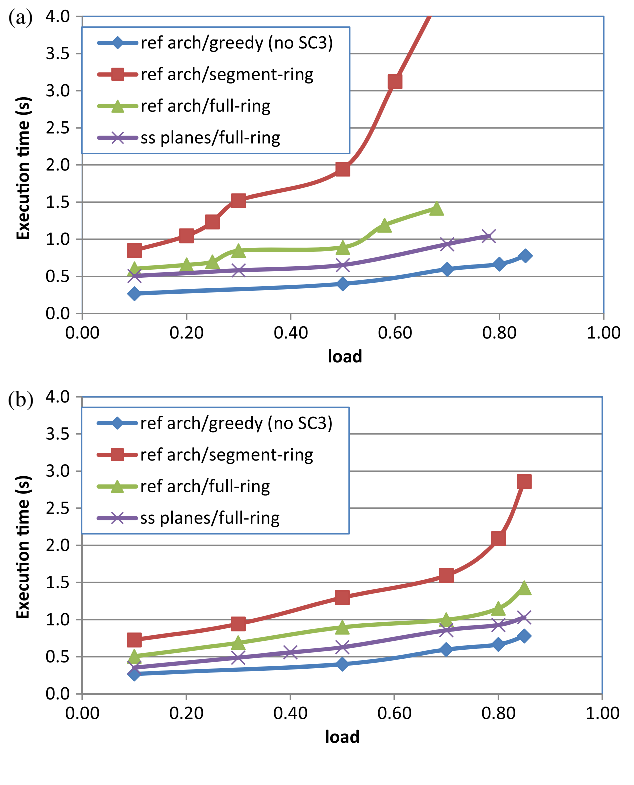

Fig. 10. Execution time as a function of load for density between pods (a) $\delta_{\text{out}} = 50\%$ ($l = 2.5\%$) and (b) $\delta_{\text{out}} = 0.5\%$ ($l = 70\%$).

of the inter-pod WDM rings and creates SC_3 conflicts. We observe that the asymptotic throughput of the reference architecture/segment-ring greedy reduces to 0.8 compared to 0.9 of the reference architecture/greedy, where SC_3 is neglected. The reference architecture/full-ring greedy has even lower throughput, measured to be 0.7, but exhibits lower execution times (see the following). The spectrum-shifted planes architecture resolves conflicts in one plane by serving in another plane and thus improves the throughput. The achieved throughput was 0.85, which is close to the case where SC_3 is neglected, as shown by the reference architecture/greedy (no SC_3).

As locality increases, inter-pod traffic decreases, and eventually, at high locality, the performance of all algorithms converges. For example, in Fig. 9(c), where the density between pods is $\delta_{\text{out}} = 0.5\%$ (or $l = 70\%$ locality), we observe that the reference architecture/greedy (no SC_3) achieves throughput close to 0.95, very close to the rest of the cases examined. Note that, according to [4], locality is very high in a Facebook DC, higher than 50% for typical DC applications, such as web and map-reduce. Figure 10 shows the related execution times. We observe that the reference architecture/segment-ring greedy has the highest running time, well above 1 s. Keeping track of ring segments yields higher complexity. Execution time is reduced in the reference architecture/full-ring greedy (but it wastes resources—has lower throughput, as seen in Fig. 9). The spectrum-shifted planes/full-ring greedy case has quite low execution time, similar to the reference architecture/full-ring greedy. Thus, it combines the execution time benefits of the full-ring algorithm while achieving throughput close to the case without SC_3 (by reducing the conflicting sets). As locality increases, the execution times of the reference architecture/full-ring and spectrum-shifted planes/full-ring converge to that of the reference architecture/greedy (no SC_3).

VII. Conclusions

We proposed and evaluated a set of scheduling algorithms for the NEPHELE DCN. In NEPHELE, resources are dynamically allocated based on traffic requirements. To avoid contention, a centralized allocation process enforces three scheduling constraints. We described in detail the NEPHELE control cycle, outlined its requirements, and presented an algorithm to optimally allocate resources. We also proposed three incremental heuristic scheduling algorithms that reduce the execution times of allocation, and evaluated their performance through simulations. The randomized and greedy heuristics exhibited normalized throughput higher than 0.85 for all examined traffic scenarios. The execution time of the greedy heuristic was measured in hundreds of milliseconds, while the sublinear greedy heuristic was faster, sacrificing some throughput. The parallel implementations of the proposed algorithms on specialized hardware (field-programmable gate array) to further reduce execution time is ongoing. We also studied the effect on performance of the third scheduling constraint (SC_3), which is specific to the NEPHELE architecture. To cope with the resulting reduction of throughput and increase of execution time, we proposed an architecture variation that employs spectrum-shifted optical planes and extended the greedy heuristic to function in such a network. Simulations showed that the throughput and execution time performance approaches that of a network without SC_3. The proposed incremental heuristic algorithms achieve high throughput and low execution time, asserting the dynamic and efficient operation of NEPHELE.

Acknowledgment

This work was partially funded by the EC through the Horizon NEPHELE project under grant agreement no. 645212.

References

[1] "Cisco Global Cloud Index: Forecast and Methodology, 2014–2019," Cisco White Paper.

[2] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM*, 2008, pp. 63–74.

[3] T. Benson, A. Akella, and D. Maltz, "Network traffic characteristics of data centers in the wild," in *ACM SIGCOMM*, 2010, pp. 267–280.

[4] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. Snoeren, "Inside the social network's (datacenter) network," in *ACM SIGCOMM*, 2015, pp. 123–137.

[5] J. Follows and D. Straeten, *Application Driven Networking: Concepts and Architecture for Policy-Based Systems*, IBM Corporation, 1999.

[6] M. Jarschel, F. Wamser, T. Hohn, T. Zinner, and P. Tran-Gia, "SDN-based application-aware networking on the example of YouTube video streaming," in *IEEE European Workshop on Software Defined Networks*, 2013, pp. 87–92.

[7] C. Kachris and I. Tomkos, "A survey on optical interconnects for data centers," *IEEE Commun. Surv. Tutorials*, vol. 14, no. 4, pp. 1021–1036, 2012.

[8] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, "Helios: A hybrid electrical/optical switch architecture for modular data centers," in *ACM SIGCOMM*, 2010, pp. 339–350.

[9] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. S. E. Ng, M. Kozuch, and M. Ryan, "c-Through: Part-time optics in data centers," in *ACM SIGCOMM*, 2010, pp. 327–338.

[10] K. Christodoulopoulos, D. Lugones, K. Katrinis, M. Ruffini, and D. O'Mahony, "Performance evaluation of a hybrid optical/electrical interconnect," *J. Opt. Commun. Netw.*, vol. 7, pp. 193–204, 2015.

[11] Y. Ben-Itzhak, C. Caba, L. Schour, and S. Vargaftik, "C-share: Optical circuits sharing for software-defined data-centers," arXiv:1609.04521, 2016.

[12] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[13] A. Singla, A. Singh, K. Ramachandran, L. Xu, and Y. Zhang, "Proteus: A topology malleable data center network," in *ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, paper 8.

[14] S. Peng, B. Guo, C. Jackson, R. Nejabati, F. Agraz, S. Spadaro, G. Bernini, N. Ciulli, and D. Simeonidou, "Multi-tenant software-defined hybrid optical switched data centre," *J. Lightwave Technol.*, vol. 33, pp. 3224–3233, 2015.

[15] G. Saridis, S. Peng, Y. Yan, A. Aguado, B. Guo, M. Arslan, C. Jackson, W. Miao, N. Calabretta, F. Agraz, S. Spadaro, G. Bernini, N. Ciulli, G. Zervas, R. Nejabati, and D. Simeonidou, "LIGHTNESS: A function-virtualizable software defined data center network with all-optical circuit/packet switching," *J. Lightwave Technol.*, vol. 34, pp. 1618–1627, 2016.

[16] N. Calabretta and W. Miao, "Optical switching in data centers: Architectures based on optical packet/burst switching," in *Optical Switching in Next Generation Data Centers*, Springer, 2017, pp. 45–69.

[17] G. Porter, R. Strong, N. Farrington, A. Forencich, C.-S. Pang, T. Rosing, Y. Fainman, G. Papen, and A. Vahdat, "Integrating microsecond circuit switching into the data center," in *ACM SIGCOMM*, 2013, pp. 447–458.

[18] http://www.nepheleproject.eu/.

[19] T. Inukai, "An efficient SS/TDMA time slot assignment algorithm," *IEEE Trans. Commun.*, vol. 27, no. 10, pp. 1449–1455, 1979.

[20] K. L. Yeung, "Efficient time slot assignment algorithms for TDM hierarchical and nonhierarchical switching systems," *IEEE Trans. Commun.*, vol. 49, no. 2, pp. 351–359, 2001.

[21] G. Bongiovanni, D. Coppersmith, and C. Wong, "An optimum time slot assignment algorithm for an SS/TDMA system with variable number of transponders," *IEEE Trans. Commun.*, vol. 29, no. 5, pp. 721–726, 1981.

[22] D. Serpanos and P. Antoniadis, "FIRM: A class of distributed scheduling algorithms for high-speed ATM switches with multiple input queues," in *IEEE INFOCOM*, 2000.

[23] K. Y. Eng and A. S. Acampora, "Fundamental conditions governing TDM switching assignments in terrestrial and satellite networks," *IEEE Trans. Commun.*, vol. 35, pp. 755–761, 1987.

[24] G. Birkhoff, "Tres observaciones sobre el algebra lineal," *Univ. Nac. Tucuman Rev. Ser. A*, vol. 5, pp. 147–151, 1946.

[25] J. Hopcroft and R. Karp, "An n5/2 algorithm for maximum matchings in bipartite graphs," *SIAM J. Comput.*, vol. 2, pp. 225–231, 1973.

[26] T. E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker, "High-speed switch scheduling for local-area networks," *ACM Trans. Comput. Syst.*, vol. 11, no. 4, pp. 319–352, 1993.

[27] I. Cerutti, N. Andriolli, P. Pintus, S. Faralli, F. Gambini, O. Liboiron-Ladouceur, and P. Castoldi, "Fast scheduling based on iterative parallel wavelength matching for a multi-wavelength ring network-on-chip," in *Int. Conf. on Optical Network Design and Modeling (ONDM)*, 2015.

[28] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Trans. Netw.*, vol. 7, pp. 188–201, 1999.

[29] S. Golestani, "A framing strategy for congestion management," *IEEE J. Sel. Areas Commun.*, vol. 9, no. 7, pp. 1064–1077, 1991.

[30] L. Tassiulas, "Linear complexity algorithms for maximum throughput in radio networks and input queued switches," in *IEEE INFOCOM*, 1998, pp. 533–539.

[31] https://github.com/kchristodou/Datacenter-network-traffic-generator.