Inter-Domain Optimization and Orchestration for Optical Datacenter Networks

G. Landi, M. Capitani, A. Kretsis, K. Kontodimas, P. Kokkinos, D. Gallico, M. Biancani, K. Christodoulopoulos, and E. Varvarigos

***Abstract*—Hyperscale datacenters (DCs), spread in various locations around the globe, provide computing and storage resources for cloud and other applications. Elastic optical networks (both landline and subsea) operated by a single or multiple entities are used to form the inter-DC network and serve the constantly increasing traffic. In addition, optical switching technologies are researched for intra-DC networks as a means to achieve higher capacity and energy efficiency. Joint optimization and orchestration of inter-DC and intra-DC network infrastructures is the key for realizing the network virtualization and network as service visions. We present a hierarchical software defined networking orchestration platform that treats intra-DC and inter-DC networks as domains and cooperates with domain specific orchestrators/controllers to achieve inter-domain, end-to-end orchestration. The systems developed are evaluated both in an emulated and in a realistic testbed, showcasing dynamic end-to-end path establishment functionality with dedicated capacity and low control overhead.**

***Index Terms*—Elastic optical networks; Inter-domain orchestration; Optical datacenter networks; Software-defined networking.**

I. Introduction

The migration of applications to cloud datacenters (DCs), their size, and their world-wide placement leads to the proliferation of DC-oriented traffic. This traffic is served by inter-DC and intra-DC networking infrastructures that play a major role in the performance experienced. In particular, hyperscale cloud DCs will account for 83% of the public cloud servers in 2020, as reported in Ref. [1]. Also by 2020, more than 90% of DC traffic will be originated or terminated in a DC (intra-DC). Moreover, the inter-DC traffic is expected to account for almost 9% of total DC traffic, higher than the 7% reported at the end of 2015, and it is growing faster than the traffic to end users or the intra-DC traffic.

Several key communication use cases are inter-domain, meaning that they span across multiple DCs, from a server in one DC (intra-DC domain) to another in a different DC (intra-DC domain), crossing over the wide area transport networks that interconnect the DCs (inter-DC domain). Examples of such use cases include load balancing, redundancy, and disaster recovery operations, as well as services deployed across multiple DCs (e.g., content distribution networks), where the virtual instances running at different locations need to interact with some quality of service (QoS) guarantees [2]. Such scenarios require low-latency inter-domain connections, dynamic and dedicated capacity allocation, and isolation.

In this paper, we consider scenarios where the inter-domain demands cross management/technological domains that belong to the same administrative entity, e.g., a cloud provider owning both the DCs and the networking infrastructure interconnecting them. The interaction between administrative domains under the control of different actors (e.g., different DC providers and network operators) raises issues related to the federation of services and infrastructures as well as to business aspects that require additional considerations. For example, policy-driven mechanisms to regulate the advertisement of the resources offered to federated domains, dedicated procedures for authentication and authorization, as well as suitable approaches for service level agreement (SLA) management, would be required to address the challenges of multi-administrative-domain scenarios. This is out of the scope of this paper and it is left for future work.

The increase in volume and dynamic requirements of DC traffic brings new challenges in their design. Today, a number of technologies promise to satisfy these network requirements. In the transport network segment (inter-DC), elastic/flexible optical networks (EONs) [3] address the inefficiency problems of traditional WDM networks, providing a fine granular solution for sub- and super-wavelength capacity. Optical switching is also gaining momentum as a potential solution for intra-DC, due to its inherent speed, energy efficiency, and transparency to bitrate and protocols. In this context, optical technologies

Manuscript received February 5, 2018; revised May 22, 2018; accepted May 29, 2018; published June 28, 2018 (Doc. ID 322539).

G. Landi and M. Capitani are with Nextworks, Pisa, Italy.

A. Kretsis (e-mail: akretsis@mail.ntua.gr), K. Kontodimas, P. Kokkinos, and K. Christodoulopoulos are with the Department of Computer Engineering and Informatics, University of Patras, Patras, Greece.

D. Gallico and M. Biancani are with Interoute S.p.A., Roma, Italy.

E. Varvarigos is with the School of Electrical and Computer Engineering (ECE) of the National Technical University of Athens (NTUA), Greece. He is also with the Department of Electrical and Computer Systems Engineering, Monash University, Australia.

https://doi.org/10.1364/JOCN.10.00B140

ranging from hybrid electrical/optical switching to all-optical packet switching have also emerged. The NEPHELE EU project [4] leverages hybrid electrical/optical switching to attain the ideal combination of high capacity at reduced cost and power, compared to state-of-the art DC networks. Both inter- and intra-DC networks require optimization and orchestration mechanisms for deciding on the allocation of network resources, for the configuration of their parameters, and for performing the actual allocation.

In this paper, we present a software defined networking (SDN)-based optimization and orchestration platform for inter-domain DC networks that includes both an elastic optical inter-DC network domain and hybrid optical–electrical intra-DC network domains, as those envisaged in NEPHELE [4]. Using SDN, the forwarding decisions are taken centrally, enabling dynamic capacity allocation in the inter-domain networks, on demand adaptation of connections, and rerouting of traffic according to network and application characteristics. The developed platform, called the NEPHELE Inter-Domain network Orchestrator (NIDO), adopts a hierarchical approach. The centralized inter-domain network orchestrator NIDO cooperates through respective interfaces with single-domain-specific SDN-based orchestrators/controllers: OCEANIA [5] for intra-DC domains and JULIUS [6] for the inter-DC domain.

Multi-domain orchestration was also presented in other works, applying stateful hierarchical path computation elements (H-PCE) and/or hierarchical SDN controllers for the orchestration of heterogeneous (Ethernet, OpenFlow/GMPLS) intra-DC and inter-DC domains [7]. Approaches based on the Application Based Network Operations (ABNO) concept [8] were presented in Refs. [9,10], with orchestrators able to jointly manage cloud and networking resources, operating over multi-layer and multi-domain network controllers. However, these solutions did not focus on specific intra-DC technologies and topologies. This is the first time, to the best of our knowledge, that interconnection technologies such as hybrid electrical–optical switching, through OCEANIA orchestrator (presented in Ref. [5]) and EON, and through JULIUS orchestrator (presented in Ref. [6]), are put under a common orchestration framework, namely NIDO, providing actual end-to-end, from one server to another, inter-domain, dedicated capacity in real time. We evaluated the systems developed (NIDO, OCEANIA, JULIUS) both in an emulated and in a realistic testbed, showcasing fast and efficient orchestration by providing dynamic inter-domain path establishment and allocation of capacity with low control overhead.

The remainder of this paper is organized as follows. Section II presents NIDO and an overview of the NEPHELE approach for inter-domain orchestration, while Section III presents the workflows associated to relevant use cases. Section IV describes the NEPHELE DC architecture, the hybrid opto-electric infrastructure, and the SDN-based network control and cloud orchestration platforms. The NEPHELE intra-DC SDN controller and its software prototype, OCEANIA, are presented in Section V, while the inter-DC controller JULIUS is described in Section VI. Performance results on the emulated and the realistic testbed are reported in Section VII. Finally, the paper is concluded in Section VIII.

II. NIDO—Inter-Domain Orchestration

A. NIDO Operation

The overall architecture for the control of inter-domain network connections between servers located in different DCs is presented Fig. 1. NIDO is able to achieve end-to-end network resource allocation by coordinating the actions of the lower layer intra-domain and inter-DC SDN controllers.

The adopted hierarchical approach enables NIDO to efficiently operate over different types of domains, changing the intra-domain controller and/or the related data-plane network technology of any domain (DC, core, metro, access network, or even for 5G fronthaul), while hiding the details from the applications utilizing NIDO. This means that OCEANIA and JULIUS can be interchanged with other domain-specific orchestrators. Thus, NIDO is more robust and flexible in comparison to network/technology-specific

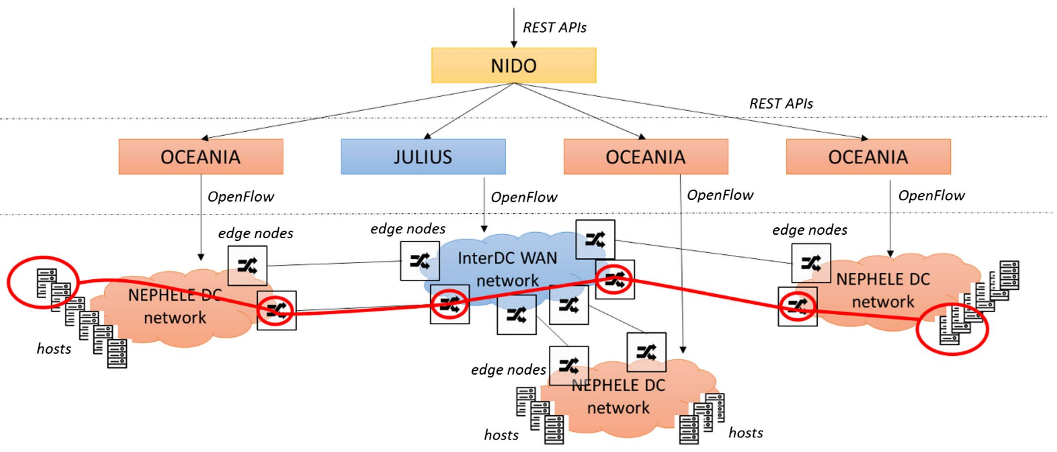


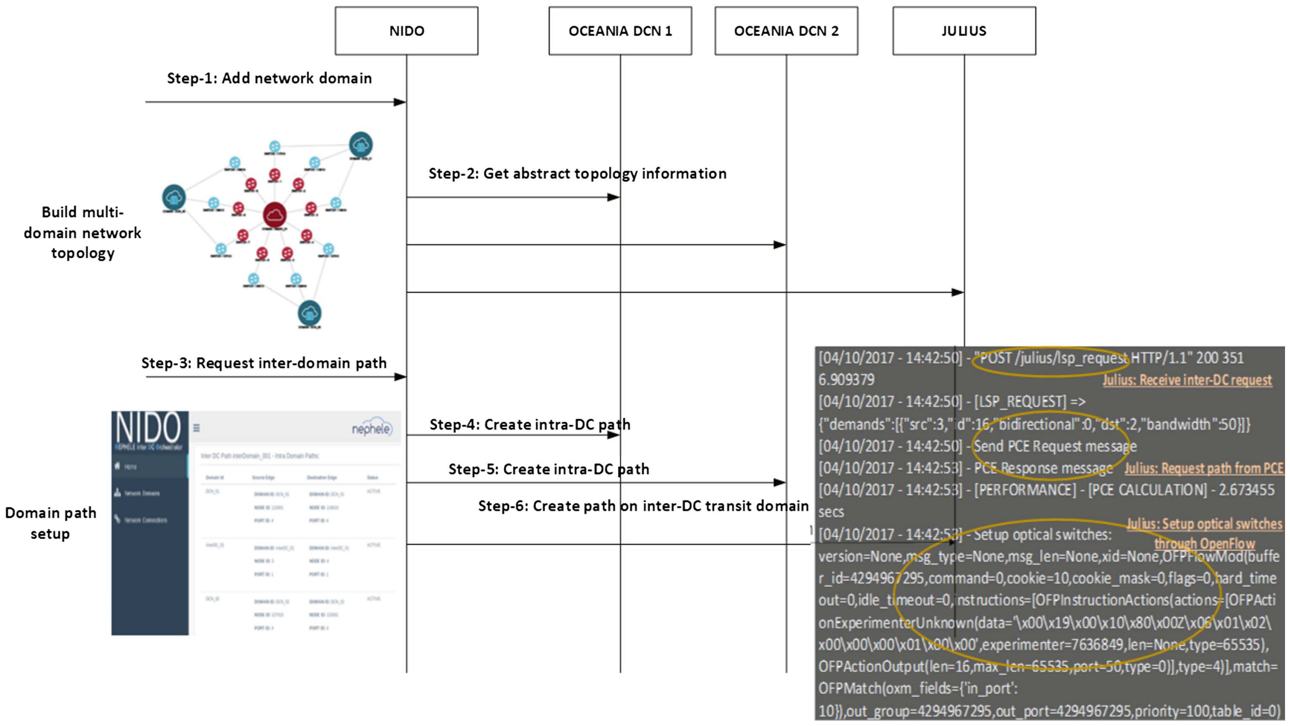
Fig. 1. NIDO inter-domain orchestration.

related systems that were presented in Ref. [11]. Details for the OCEANIA intra-DC domain orchestrator for NEPHELE-based DC networks [4] (Section IV) are presented in Section V, while the JULIUS orchestrator for inter-DC EON domains is presented in Section VI.

Cloud orchestrators, such as OpenStack, can be extended to interact with NIDO, building inter-domain networks, interconnecting their virtual computing and storage instances across multiple DCs. For this reason, NIDO exposes (Fig. 1) at its northbound interface (NBI) the appropriate REST APIs to enable the integration of such cloud orchestrators, making it possible to set up and tear down QoS guaranteed end-to-end connections, as part of the whole workflow for cloud service provisioning, termination, migration, or recovery. For example, Fig. 2 shows the workflow for the provisioning of a dedicated inter-domain transport network connection between two servers belonging to two different DCs, in order to enable a suitable interconnection between virtual instances placed in those servers. Initially, when a new domain is added through NIDO, there is a process for building the multi-domain network topology (Steps 1 and 2). The network provisioning is initiated by a cloud orchestrator that triggers the entire automated procedure (Steps 3–6) to establish an end-to-end path from a server in DC-1 to a server in DC-2, through which the virtual instances' traffic will be carried. This approach allows for hiding the internal details of the network topology, protocols, and configuration mechanisms from the cloud controller, which uses only abstract and technology-agnostic, domain-independent interfaces (based on REST-APIs) to request connections with the desired QoS (capacity and latency) parameters.

NIDO's internal procedure for establishing an end-to-end network service consists of two main steps. NIDO computes initially the loose domain path for the requested inter-domain connection, i.e., the sequence of domains that must be traversed, together with the edge nodes at each domain. In the second step, NIDO issues requests to set up the intra-domain network connection in the domains returned in the domain path, utilizing the NBI of the respective intra-domain controllers (i.e., OCEANIA for intra-DC networks and JULIUS for the inter-DC transport network). Figure 2 indicates also the actions taken by the JULIUS controller: receiving the respective inter-DC request, asking from the internal PCE for a path, and actually reserving the optical path by interacting through OpenFlow (OF) protocol [12], with the required SDN-enabled optical switches. Once all the intra-domain paths have been established and acknowledged to NIDO, the end-to-end inter-domain path becomes active.

The adopted intra-DC (NEPHELE) and inter-DC (EON) architectures are all optical, and their related controllers (OCEANIA and Julius, respectively) utilize appropriate reservation/path establishment processes that can guarantee in each individual domain the requested capacity, minimum delay (equal to the propagation delay in that domain), and ultra-low packet drop ratio. Domain to domain (inter-domain) communication is performed using a DC gateway, an IP router at the edge of the DC, which is connected to an IP router attached to the ingress ROADM of the EON. At these routers, we can reserve capacity and provide specific latency or dropping ratio QoS guarantees using well-established mechanisms, such as integrated services (IntServ) and differentiated services (DiffServ). NIDO


NIDO
OCEANIA DCN 1
OCEANIA DCN 2
JULIUS
Step-1: Add network domain
Step-2: Get abstract topology information
Build multi-domain network topology
Step-3: Request inter-domain path
Step-4: Create intra-DC path
Step-5: Create intra-DC path
Step-6: Create path on inter-DC transit domain
Domain path setup
NIDO
nephele
Julius: Receive inter-DC request
Julius: Request path from PCE
Julius: Setup optical switches through OpenFlow

```
[04/10/2017 - 14:42:50] - "POST /julius/lsp_request HTTP/1.1" 200 351
6.909379
[04/10/2017 - 14:42:50] - [LSP_REQUEST] =>
{"demands":[{"src":3,"id":16,"bidirectional":0,"dst":2,"bandwidth":50}]}
[04/10/2017 - 14:42:50] - Send PCE Request message
[04/10/2017 - 14:42:53] - PCE Response message
[04/10/2017 - 14:42:53] - [PERFORMANCE] - [PCE CALCULATION] - 2.673455
secs
[04/10/2017 - 14:42:53] - Setup optical switches:
version=None,msg_type=None,msg_len=None,xid=None,OFPFlowMod(buffe
r_id=4294967295,command=0,cookie=10,cookie_mask=0,flags=0,hard_time
out=0,idle_timeout=0,instructions=[OFPInstructionActions(actions=[OFPActi
onExperimenterUnknown(data='\x00\x19\x00\x10\x80\x00Z\x06\x01\x02\
x00\x00\x00\x01\x00\x00',experimenter=7636849,len=None,type=65535),
OFPActionOutput(len=16,max_len=65535,port=50,type=0)],type=4)],match=
OFPMatch(oxm_fields={'in_port':
10}),out_group=4294967295,out_port=4294967295,priority=100,table_id=0)
```

Fig. 2. Workflow for provisioning of a dedicated inter-domain path through NIDO.

collects the propagation latency specifications of the different intra-domain paths, subtracts them from the end-to-end latency requirement, and enforces that to the inter-domain links. Capacity and loss are satisfied more easily, since these are deterministically provided by all related intra-domain all-optical paths. Finally, note that NIDO can use the load/latency as the criterion for selecting the inter-domain link. In the current implementation, NIDO establishes end-to-end paths and reserves the required capacity, while with the simple extensions indicated above it can enforce other QoS requirements, such as end-to-end latency and the packet loss ratio.

NIDO is responsible for maintaining the status of the different inter-domain paths, together with their relationship to the list of intra-domain paths that compose them. The abstract network topology is also kept updated accordingly, modifying the resource availability on the inter-domain links based on the active inter-domain paths. Recovery in the adopted multi-domain scenario is quite complicated. Domain specific restoration/protection processes can handle link/connection failures in each domain. If such recovery violates the QoS requirements or the failure is in an inter-domain link, the recovery could be handled by a similar process as the end-to-end connection establishment process, described above, where the failed link is removed (by NIDO or the domain controller, depending on the failure type) from the possible solution space, and the NIDO inter-domain calculation process is repeated. Such an advanced concept is left for future work.

The proposed NIDO architecture is extendable. It is possible to add and remove domains dynamically, specifying the type of plug-in that must be used to interact with the associated controller. Furthermore, it is possible to extend the architecture to support infrastructure federations, and enable the integration of services and (virtual) resources from different providers. This would require mechanisms to manage SLAs, business aspects, and architectural principles based on brokering or peer-to-peer interactions. In particular, to support the peer orchestration model, the NIDO architecture would need to be extended with discovery mechanisms, cooperative approaches for inter-domain path computation (e.g., following the backward recursive PCE-based computation procedures defined in RFC 5441 [13]) and policies for disclosure of topology information between peering domains. Finally, the adoption of NEPHELE architectures in network functions virtualization (NFV) environments would require additional interfaces where NIDO interacts with NFV orchestrators (NFVOs), for example, acting as a WAN infrastructure manager (WIM). In this context, beyond the provisioning of transport connections between DCs or NFVI point of presence (PoP)s, NIDO would also need to implement mechanisms for management of service function chains and traffic steering.

B. NIDO Architecture

Figure 3 shows the functional architecture of NIDO. The architecture follows some of the principles of the ABNO architecture, defined in IETF RFC 7491 [8], including, for example, components for path computation (equivalent to the ABNO PCE), topology management [similar to the Traffic Engineering Database (TED)], and provisioning management. However, NIDO components are focusing on the specific challenges of multi-technology domain scenarios, implementing a topology representation that is based on the concept of network domains, edge nodes, and logical inter-domain and edge-to-edge intra-domain links. Moreover, the specific functionalities of the ABNO controller that handles the network requirements from the applications are implemented through the multi-domain connectivity manager. This entity is specifically designed to operate in combination with cloud management platforms, triggering the configuration of the network based on the requirements of cloud services.

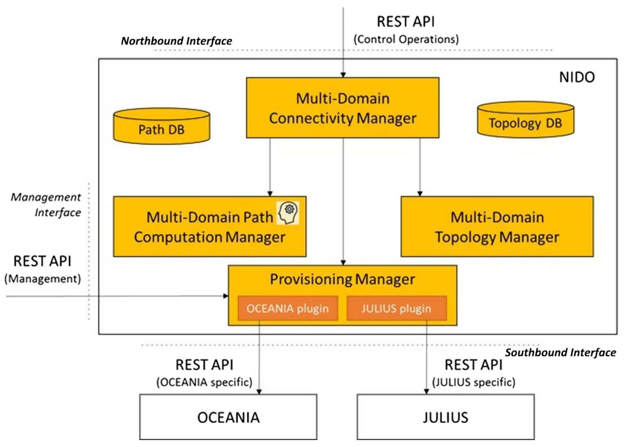


Fig. 3. NIDO functional architecture.

In the NIDO architecture, the multi-domain connectivity manager is the entry point of the system, and it handles the operational requests for the provisioning of inter-domain network connections. At its northbound, it offers the REST APIs to create, tear down, and retrieve the network paths, and it handles internally the lifecycle of all the inter-domain connections managed by NIDO.

The NIDO REST API has been specifically developed to address the needs of multi-domain orchestration while exploiting the capabilities of the intra-domain controllers OCEANIA and JULIUS. It allows the user to specify end-to-end paths with reserved capacity to be established between hosts belonging to the DCs. These APIs, while based on the abstraction approach also adopted in the transport APIs defined by the Open Networking Foundation (ONF), are designed around a NEPHELE specific information model focused on DC entities, such as DC hosts, DC gateways, and inter-DC connections. This information model fully abstracts the complexity of connecting different technological domains, as well as the details of the intra-domain paths and the interaction with different SDN controllers.

The multi-domain connectivity manager is also responsible for maintaining the status of each inter-domain connection and its relationship with the intra-domain network connections (i.e., path on the intra-DC or in the inter-DC network) that compose the end-to-end path. The persistency of inter-domain and intra-domain paths is guaranteed through the path DB. Moreover, the multi-domain connectivity manager coordinates the procedures to compute and instantiate the paths across the different domains.

The provisioning manager is the functional entity responsible for the provisioning of the single intra-domain paths at the southbound. It is triggered by the multi-domain connectivity manager, and it issues provisioning requests to the child SDN controllers handling the different domains. The provisioning manager has multiple plugins to manage the specific APIs exposed by each child controller; for example, the OCEANIA plugin includes a REST client to invoke the functions provided by the OCEANIA controller to create network connections in NEPHELE DCs (Section V). In addition, the provisioning manager offers a management interface, based on REST APIs, for the configuration of network domains, and it triggers the multi-domain topology manager to update the whole network topology according to the abstract topology gathered on each single controller.

The multi-domain topology manager is the entity responsible for building and maintaining an abstract inter-domain topology, based on the information collected about single domains from the provisioning manager plugins. The multi-domain topology includes the concepts of network domains, edge nodes, hosts, and inter-domain links (intra-DC and inter-DC), as well as resource availability in terms of capacity of the inter-domain links. The internal details of each domain are considered out of scope, assuming a flat full mesh connectivity among all the edge nodes of a single inter-DC domain and a full connectivity between each host and all the network gateways in a DC network. The topology is persisted in a database (the topology DB). In terms of network nodes and links, it is updated based on the network domains added or removed in the system, while the resource availability on the inter-domain links is modified according to the established paths.

Finally, the multi-domain path computation manager provides the functionalities for the computation of the domain chain to be traversed by the requested inter-domain paths, including the edge nodes for ingress and egress at each domain. The multi-domain path computation manager may support different algorithms, which run over the abstract network topology handled by the multi-domain topology manager. The current version of NIDO implements an algorithm that performs load balancing among the inter-domain links, distributing the traffic load across different domain ingress and egress edge nodes.

NIDO comes with a simple user interface through which it is possible to add new domains, to view the multi-domain network topology, and to request and view inter-domain network connections. The NIDO software prototype was developed in Java and released as open source software [14].

III. Workflows for Inter-Domain Communications

This section provides examples of workflows for the management and control of cloud services, where NIDO

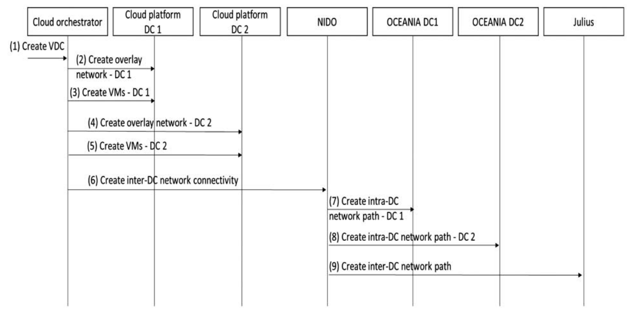

Cloud orchestrator
Cloud platform DC 1
Cloud platform DC 2
NIDO
OCEANIA DC1
OCEANIA DC2
Julius
(1) Create VDC
(2) Create overlay network - DC 1
(3) Create VMs - DC 1
(4) Create overlay network - DC 2
(5) Create VMs - DC 2
(6) Create inter-DC network connectivity
(7) Create intra-DC network path - DC 1
(8) Create intra-DC network path - DC 2
(9) Create inter-DC network path

Fig. 4. Workflow for provisioning of a distributed VDC.

is used for setting up inter-domain paths, providing dedicated capacity and isolation to the traffic flows of the respective services. In particular, in the presented workflows, it is assumed that a cloud orchestrator oversees the cloud operations of various DCs, interacting with DC-located cloud platforms. This cloud orchestrator is also responsible for interacting with NIDO. For example, in Ref. [15] a disaster recovery layer (the cloud orchestrator in our context) operates over a pair of OpenStack enabled datacenters (the cloud platforms in our context), enabling virtual machines (VMs) and volumes to be protected and recovered in another DC, in case of a disaster.

A. Provisioning of a Distributed VDC

Figure 4[1] shows the workflow for the provisioning of a distributed virtual datacenter (VDC), where the VMs composing the cloud service are placed in two different DCs and need to communicate with a given capacity. This scenario can easily be extended to other QoS metrics and several DCs just by provisioning additional network connections among all the DC servers involved in the cloud service and setting up in this way an inter-domain virtual network.

The traditional approach is based on the provisioning of VMs and overlay virtual networks in the two DCs (Steps 1–5). The creation of the overlay virtual network is typically performed through the configuration of virtual switches, such as an open virtual switch (OVS) [16], running on the servers where the VMs are placed, e.g., utilizing cloud platform components like OpenStack Neutron. However, this procedure does not allow configuration of the underlying transport network in the different domains traversed by the VMs traffic and, thus, it cannot provide QoS guarantees.

Through NIDO, the workflow is extended with an additional step triggered by the cloud orchestrator (Step 6). After the creation of the overlay virtual network and the allocation of the VMs, the cloud orchestrator invokes NIDO to request the provisioning of inter-domain network connections with specified capacity between the servers where the VMs are placed. This request triggers in turn the entire automated procedure (Steps 7–9) to establish

[1]In the figure we omit the reply messages for simplicity.

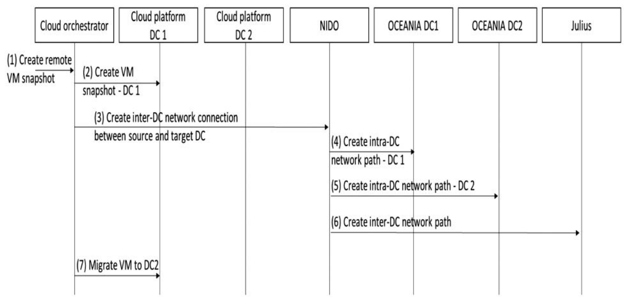

Cloud orchestrator
Cloud platform DC 1
Cloud platform DC 2
NIDO
OCEANIA DC1
OCEANIA DC2
Julius
(1) Create remote VM snapshot
(2) Create VM snapshot - DC 1
(3) Create inter-DC network connection between source and target DC
(4) Create intra-DC network path - DC 1
(5) Create intra-DC network path - DC 2
(6) Create inter-DC network path
(7) Migrate VM to DC2

Fig. 5. Workflow for migration of a VM snapshot.

an end-to-end path from a server in DC 1 to another in DC 2, through which the VMs traffic will be carried.

B. Cloud Service Recovery

Mechanisms for cloud service recovery are usually provided by cloud operators in order to offer a certain level of protection against data loss in case of infrastructure failures or disruptive actions on the VMs [2,15]. Snapshots of active VMs are captured periodically and are moved to a remote DC, so that they can be re-used to re-instantiate the service during disaster recovery procedures. The transfer of VM snapshots between DCs generates a huge amount of traffic that is currently carried through the same connections used for serving the traffic of active VMs. This may create overloading situations, impacting the performance of the running cloud services. To solve this issue, NIDO allows creating dedicated-isolated inter-domain network connections with a given amount of reserved capacity, so that they can be used exclusively to transfer the VM snapshots without interfering with other cloud service traffic.

Figure 5[2] shows the workflow for the provisioning of a dedicated network connection between DC 1, where the reference VM is running, and DC 2, i.e., the target DC where the VM snapshot will be saved. The whole procedure is coordinated by the cloud orchestrator. In Step 2, it requests from the cloud platform to create a snapshot of the VM. In Step 3, it interacts with NIDO, requesting the provisioning of the connection between the source and target server (Steps 4–6), while in Step 7 it requests the migration of the VM snapshot to the target DC. The related traffic will be routed through the end-to-end (intra-DC and inter-DC) network path established in the previous stage.

IV. NEPHELE Datacenter Architecture

This section describes the architecture of the NEPHELE [4,17] intra-datacenter network (DCN), presenting the main technologies adopted at the data plane level and the high-level mechanisms offered by the SDN control plane and the cloud platform to efficiently manage its resources. The internal details of the NEPHELE intra-DC SDN controller are provided in Section V.

[2]In the figure we omit the reply messages for simplicity.

The NEPHELE DC follows the SDN approach: the data and the control plane are decoupled and interact through open interfaces and protocols at the southbound interface (SBI) of the SDN controller. The SDN controller through its NBI can provide network customization to applications, delivering connectivity and specific capacity. On top of that, a cloud management platform orchestrates all the DC resources (computing, storage, and networking) by interacting with the related resource controllers to deliver end-to-end cloud services to upper layer applications. The NEPHELE architecture is compliant with this trend and follows a three-layer architecture:

- The DCN data plane employs hybrid optical–electrical switching to support high capacity and energy efficiency.
- The DCN network control framework is based on an SDN controller that interacts with the hybrid optical/electrical DCN through SBI, and also offers NBI to the cloud orchestrator
- The cloud orchestrator operates jointly the whole DC infrastructure, coordinating the resource allocation (computing, storage, and DCN), delegating the actual DCN configuration to the network control framework.

A. Data Plane Architecture

The NEPHELE DCN data plane (Fig. 6) is based on a flat topology with two tiers that can be easily scaled in the east–west direction without increasing traffic latency and congestion. The network is divided into *pods*, each containing W racks, and there are P pods in the network. Each rack contains a number of innovation zones, where an innovation zone is a collection of hosts, storage, memory, and other devices. A rack uses a hybrid electrical/optical top-of -rack (TOR) switch (first tier) to connect to an all-optical network (second tier) that interconnects the pods. All TORs in a pod are connected with one optical port to a POD switch, forming a star topology. The POD switch is built with arrayed waveguide gratins (AWGs), splitters, combiners, and wavelength selective switches (WSSs). P POD switches

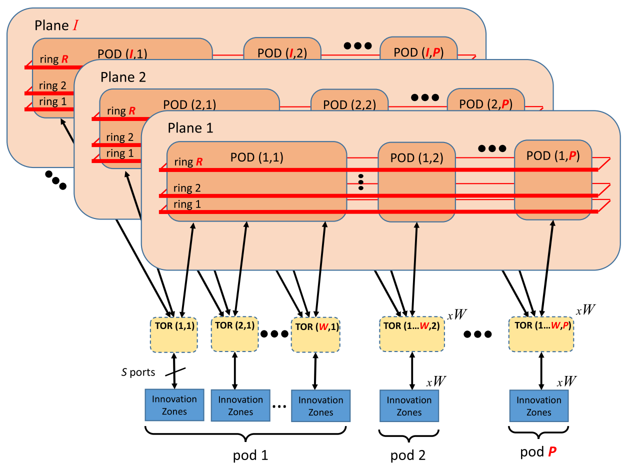


Fig. 6. NEPHELE data plane architecture.

are connected with R unidirectional rings to create an all-optical network that we call *optical plane*.

To scale the network, we connect I parallel (independent) optical planes, each of which consists of P POD switches. Each TOR switch is connected to all the I POD switches of the pod, where each (north) port of the TOR switch is connected to a different POD switch. The TOR also has S (south) ports facing the innovation zones.

The NEPHELE optical network uses WDM and time division multiplexing (TDM). Regarding WDM, the key concept is that each rack listens to a specific wavelength, and there are W wavelengths used in the network, which equals, by design, the number of racks in a pod. Thus, in each pod, each TOR is reached via a different wavelength, and wavelengths are reused in different pods. Each port of the TOR is equipped with a tunable laser that tunes according to the selected destination. Each optical link from TOR to POD (or POD to TOR) switches carries a single wavelength at a time, while wavelengths are multiplexed in the fiber rings connecting the POD switches.

The network operates in a slotted manner (TDM), where T slots form a period. In the upstream direction, wavelength assignment is performed dynamically, per TDM slot, identifying uniquely the position of the destination TOR switch within the target POD switch. The slot containing the optical signal is then switched all-optically in the network of POD switches, so in the east–west and downstream directions the wavelength assignment is static.

B. SDN-Based Control and Orchestration Architecture

The goal of the NEPHELE DCN control framework is to provision intra-DC connections and optimize the usage of the DCN physical infrastructure, while guaranteeing the desired level of QoS for the applications running in the virtual environments. To do so, it makes use of the WDM and TDM concepts to achieve fine granularity and dynamicity in the assignment of network resources.

In NEPHELE, DCN resource allocation is driven from applications and the related traffic dynamics. These are captured in a traffic matrix built according to the cloud service requests. This approach makes it possible to transfer application awareness from the cloud platform to the network control plane, enabling a more tight cooperation between the cloud orchestrator and the SDN controller. This implies that the cloud orchestrator is capable of managing enriched cloud service models, with service templates describing network requirements and traffic patterns, as expected by the cloud applications. These parameters constitute the input to the network controller and feed advanced algorithms for application-aware network allocation [18].

The NEPHELE control and orchestration architecture is shown in Fig. 7. The cloud management platform (e.g., OpenStack) orchestrates the resources (computing, storage, DCN) of the entire DC and delivers virtual infrastructures to different tenants. The DCN is controlled

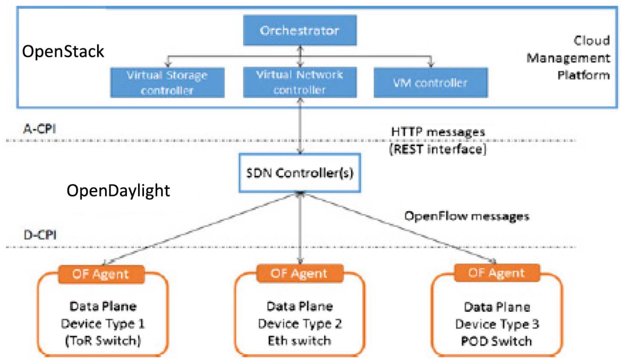

Orchestrator
OpenStack
Cloud Management Platform
Virtual Storage controller
Virtual Network controller
VM controller
A-CPI
HTTP messages (REST interface)
SDN Controller(s)
OpenDaylight
OpenFlow messages
D-CPI
OF Agent
Data Plane Device Type 1 (ToR Switch)
OF Agent
Data Plane Device Type 2 Eth switch
OF Agent
Data Plane Device Type 3 POD Switch

Fig. 7. NEPHELE control and orchestration frameworks.

through the centralized SDN controller, which implements the network logic. For scalability purposes, the logic of the SDN controller can be split across several entities following a hierarchical approach, with child controllers dedicated to optical planes or pods and an upper-layer/parent-controller responsible for coordinating them. The SDN controller prototype that was developed (see Section V) followed a strictly centralized approach.

V. OCEANIA–NEPHELE SDN Controller

The NEPHELE SDN controller prototype, called OCEANIA, was developed in the OpenDaylight (ODL) framework [19].

OCEANIA has the complete view of the NEPHELE DCN, composed of the hybrid TOR and POD switches and hosts. It offers a REST API at its NBI to receive service requests that describe the application traffic requirements, driving the creation of new or the adaptation of existing network connections. The OCEANIA REST API is an *ad hoc* development that allows for specifying both the endpoints and the capacity of the network connections to be established between DC nodes to optimize the translation of these requirements into the allocation of optical resources. Such requirements are translated appropriately to form the input of the optimization engines, which decide on the allocation of the resources. The resource allocation decisions are then translated into a set of OF [12] rules that are installed into the DCN data plane devices (TOR and POD switches), configuring their forwarding behavior. The interaction of OCEANIA with the data plane is based on extended OF messages that support advertisement, operational configuration, and monitoring.

OCEANIA is released as open source software [20] and was also publicly demonstrated [21]. In the following we present its internal mechanisms and workflows.

A. SDN Controller Functional Architecture

OCEANIA, NEPHELE's SDN controller, adopts a dual strategy for resource allocation, with real-time reactions for short-term decisions and periodic reconfiguration of the entire DCN for medium-/long-term decisions.

The short-term strategy is applicable to service requests that require the upscale or downscale of already active services, and to react to data plane failures with fast recovery. These cases require high dynamicity, and thus the SDN controller adopts fast “online” or “incremental” algorithms [18] to react quickly to such single (or few) events, even if leading to suboptimal solutions. The short-term strategy takes into account the new requests and incrementally adapts the previous DCN allocation. This strategy is well suited for on-demand provisioning and fast recovery of network connections. However, in order to maximize the DCN usage, a second, medium-/long-term strategy provides better performance. In this case, application traffic profiles are used as input to build an application-aware traffic matrix that can be updated over long periods. Offline scheduling algorithms that can take long computation time are used to optimally allocate the resources in this case.

Figure 8 shows the OCEANIA functional architecture, including the main components and interactions for short- and medium-/long-term resource allocation. The interaction between the SDN controller and the data plane is performed via the SBI; the extended OF protocol is used for configuring TOR and POD switches and the OVSDB [22] protocol for configuring the OVS instances on the servers.

The core services provide basic functions, abstracting the physical resources with unified information models. These services are invoked by higher layer SDN applications, to collect the network topology or monitoring information or to issue configuration commands through protocol- and technology-agnostic interfaces. In compliance with the ODL framework, all the interfaces of the services use YANG models [23] and can be invoked by other ODL components or by external entities via REST APIs.

The NEPHELE DCN logic is implemented in the programming applications. They employ special purpose optimization algorithms for short- and medium-/long-term resource allocation decisions at the ONLINE COMPUTATION ENGINE and at the traffic offline scheduling engine, respectively. The application affinity service coordinates the workflows for DCN resource allocation based on traffic profiles and requirements. As in core services, all the DCN programming applications expose REST APIs to enable the interaction with the cloud platform.

B. DCN Configuration Workflows

This section describes the internal workflows to allocate network resources in the NEPHELE DCN following the approach based on the periodic reconfiguration of the whole infrastructure, with the optimal allocation solution computed by the offline scheduling engine [18].

The workflow initiates from the application affinity service, when it receives a request to initiate a network connection for a particular application profile. The details of the requested connections are forwarded to the traffic matrix engine, which updates the current traffic matrix with the new data. Then, the offline engine starts to re-compute the allocation of the network resources for the entire DCN (see Fig. 9).

Depending on the DCN dimension, the offline engine computation may take time, so the application affinity service periodically requests the engine until the result is available (see Fig. 10). The network allocation solution provides the timeslots and destinations (wavelengths) to be used for the communication of each source, taking into account the architecture constraints (use single Tx and Rx at each timeslot, and no wavelength or timeslot conversion). As soon as the computation terminates, the application affinity service forwards the solution to the flow manager, which de-aggregates the data contained in the matrix and

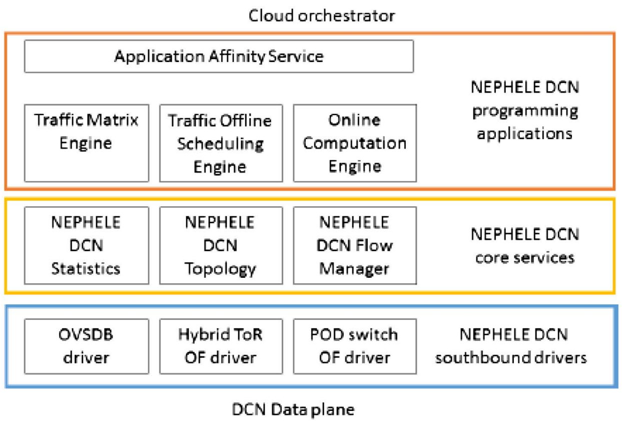

Cloud orchestrator
Application Affinity Service
NEPHELE DCN programming applications
Traffic Matrix Engine
Traffic Offline Scheduling Engine
Online Computation Engine
NEPHELE DCN Statistics
NEPHELE DCN Topology
NEPHELE DCN Flow Manager
NEPHELE DCN core services
OVSDB driver
Hybrid ToR OF driver
POD switch OF driver
NEPHELE DCN southbound drivers
DCN Data plane

Fig. 8. NEPHELE SDN controller: functional architecture.

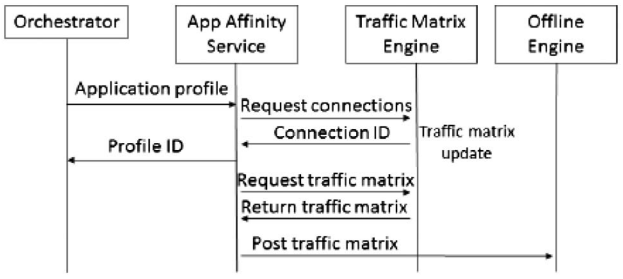


Fig. 9. Workflow for creation of a new application profile.

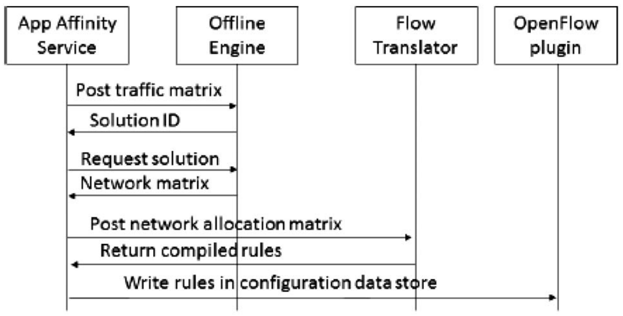


Fig. 10. Workflow for updating DCN resource allocation.

returns the list of the flow rules to be installed on the physical devices.

The flow rules defined in NEPHELE extend the traditional rules of the OF protocol. In particular, the flow match structure defines a wavelength and the timeslots (described in a bitmap) to classify the incoming traffic on optical ports, while the same parameters are defined in the flow action structure to specify a cross connection between two WDM ports for the given set of timeslots. The flow rules resulting from flow manager elaboration are then written in the ODL configuration data store, triggering the procedures at the OF plugin to send the associated flow mod messages. At the data plane level, the OF messages are intercepted by device-specific agents, which handle the translation to configuration commands toward the FPGA controlling the data plane hardware.

C. NEPHELE Controller Prototype

The proof-of-concept prototype of OCEANIA [20] is based on the ODL controller, lithium version, with extended internal components and a set of SDN applications developed from scratch. In particular, regarding the controller internal modules, the OF ODL plugin was enhanced to support the definition of wavelengths and timeslots in OF rules at the SBI. Moreover, the traffic matrix engine and the (online and offline) scheduling engine were developed as external SDN applications that make use of the controller's REST APIs. In particular, the scheduling engine is a standalone application written in C, and the algorithm implementation is a translation of MATLAB code converted using MATLAB coder. The other SDN applications are Java applications based on the Spring MVC framework.

The prototype provides mechanisms for (i) accepting requests that specify connectivity requirements between innovation zones (Section IV) with a specific capacity; (ii) aggregating these requests into a global DCN traffic matrix; (iii) computing a network-wide resource allocation solution for the specified DCN traffic load; (iv) translating the allocation solution in the set of extended OF rules; and (v) requesting the OF plugin to install these rules for the configuration of the DCN data plane devices, the POD, and TOR switches.

OCEANIA provides a NBI REST API through the application affinity service, enabling its integration with cloud management platforms, like OpenStack. Moreover, the ODL DLUX graphical user interface (GUI) was extended for OCEANIA. Through the GUI we can request new DCN connections, and visualize the traffic matrix and the installed flows. The GUI also provides a monitoring and diagnostic tool for DCN administration purposes.

The OCEANIA prototype was demonstrated with a mix of emulated and physical devices in Ref. [21]. The demonstration (Fig. 11) included the entire DCN configuration workflow, from the specification of new application-based connections via the GUI, to the calculation of the resource allocation solution and its elaboration, up to the OF-based interaction with the optical data plane through the OF agents.

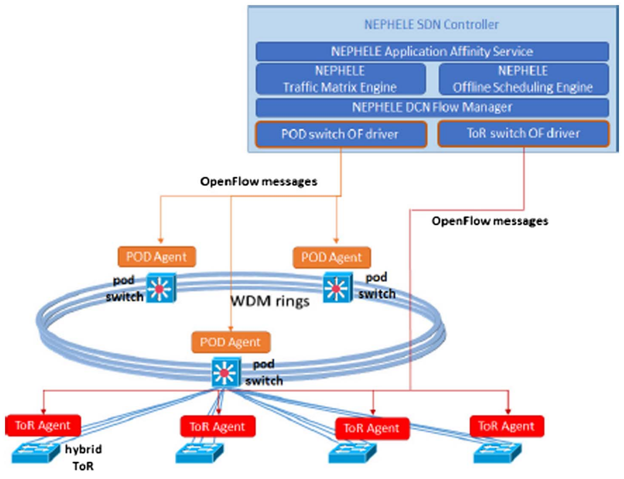


Fig. 11. Prototype of NEPHELE controller.

VI. JULIUS—Inter-DC Orchestrator and Emulation Platform

JULIUS [6] is the orchestrator for inter-DC EON. It utilizes Mantis' [24] path computation element (PCE), providing optimization logic in multi-layer IP/optical networks. JULIUS also incorporates a complete emulation platform for SDN-based, multi-layer IP/optical networks, enabling the development and evaluation of protocol extensions [e.g., OF, path computation element protocol (PCEP)] and resource optimization algorithms. The network emulation platform is based on Mininet [25]. The emulation environment can be used as a proof-of-concept demonstration of the maturity of the JULIUS orchestrator and support its adoption in real SDN-enabled IP/optical networks. JULIUS is organized in three layers (Fig. 12): the access layer, the orchestration layer, and the execution layer.

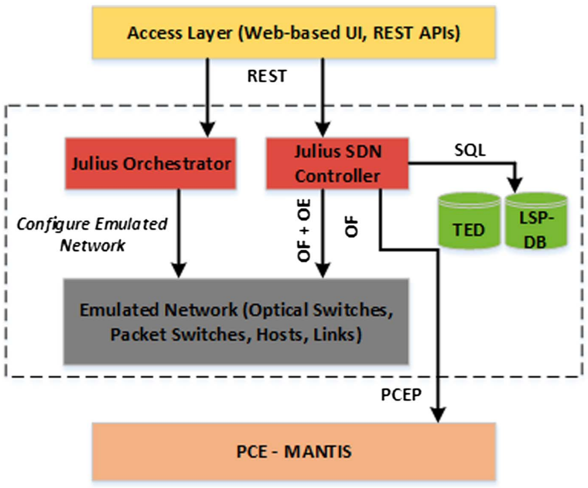

Access Layer (Web-based UI, REST APIs)
REST
Julius Orchestrator
Julius SDN Controller
SQL
TED
LSP-DB
Configure Emulated Network
OF + OE
OF
Emulated Network (Optical Switches, Packet Switches, Hosts, Links)
PCEP
PCE - MANTIS

Fig. 12. JULIUS architecture and its main modules.

A. Access Layer

The access layer handles the interaction with users either through a web-based GUI (Fig. 13) or appropriate REST APIs. The user requests that arrive at the access layer are translated into commands and sent to the orchestration layer. The access layer enables users to build various network emulation scenarios, execute them, and analyze their performance easily using the exposed interfaces.

B. Orchestration Layer

The orchestration layer creates and manages the emulation environment and coordinates the execution of the users' requests from the access layer to the execution layer. It consists of the JULIUS orchestrator, the JULIUS SDN controller, and a number of auxiliary modules that are responsible for the emulated network infrastructure. The JULIUS orchestrator's primary task is the preparation and management of the emulation based on the user preferences coming from the access layer. Furthermore, the JULIUS orchestrator constantly monitors all the modules and reacts accordingly either to changes requested from the access layer or to any malfunction.

The second basic module is the JULIUS SDN controller, which is responsible for the control of the IP/optical network. For the implementation of the SDN controller, we extended the Ryu SDN framework [26]. The interaction of the JULIUS controller with the devices (packet and optical switches) is done via SBI using appropriate protocols (OF and OF with Optical Extensions). The JULIUS controller also provides a RESTful API, through which it receives and serves requests from the access layer.

In addition to the above, the orchestration layer also includes third-party open source tools, which are put together with the JULIUS orchestrator and the SDN controller in order to build the emulation environment. We used the Mininet emulator, the OVSs for L3 switches, and LINC switch for Optical Emulation (LINC-OE) [27] to emulate optical switches (ROADM) and hybrid switches. The JULIUS controller interacts with these using OF 1.3 with Infoblox Optical Extensions, to emulate the peculiarities of the optical switches.

The orchestration layer also includes the databases that contain information regarding the capabilities and the current state of the network: the TED and the label switched path database (LSP-DB). These databases, which are kept up to date by the JULIUS orchestrator and the SDN controller, are used in the computations carried out in the execution layer.

C. Execution Layer

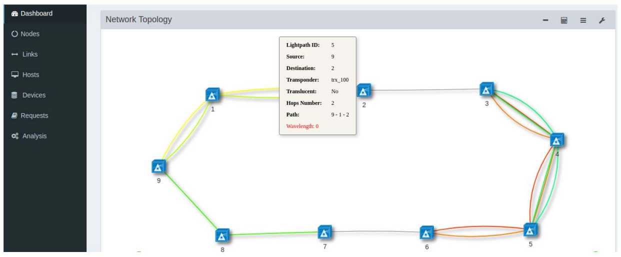
The execution layer consists of the PCE that provides to the orchestration layer the algorithmic logic for performing efficiently the various network resource allocation decisions. PCE interacts with the JULIUS SDN controller, using the PCEP, receiving requests for serving new connections or re-optimizing existing ones. For the PCE implementation in JULIUS we used an extended version of Mantis [24], which contains algorithms for a number of network resource allocation operations including routing and wavelength assignment (RWA), route and spectrum allocation (RSA), network re-optimization (including spectrum defragmentation), capacity on demand and calendaring operations, and restoration decisions. Mantis acts as a stateful PCE taking its decisions based on the current network topology (information from the TED) and the state of all the previously computed and established paths along with their required resources (information from the LSP-DB).

VII. Evaluation—Performance

We evaluated the proposed hierarchical SDN-based inter-domain orchestration platform both in an emulated and a realistic testbed. The purpose was to showcase dynamic inter-domain path establishment and allocation of capacity with low control overhead. As discussed in Section II.A, the proposed architecture assumes all-optical network (Intra-DC and inter-DC) domains and resource allocation mechanisms through the related domain controllers that provide dedicated capacity, latency equal to the propagation delay, and negligible losses in each domain. NIDO establishes end-to-end paths reserving the required capacity, while with simple extensions it can enforce other QoS requirements such as end-to-end latency and the packet loss ratio.

Initially, the performance of the OCEANIA prototype was validated in a Mininet-based emulated network environment, to evaluate the system scalability when operating with different DCN sizes. In particular, the number of pods varies from 2 to 6, representing sample topologies for mini to medium DCNs (from 3200 servers up to 9600 servers), with a variable number of total network nodes (i.e., TOR and POD switches) controlled by OCEANIA from 164 up to 492. The controller was tested by requesting and terminating flows with randomized back-off on the emulated data plane, with around 240 requests. The average of the time required to re-configure the entire DCN when a new flow is instantiated is between 3.6 and 4.2 s, depending on the size of the reference DC. Similarly, JULIUS design and implementation was thoroughly validated by using it for the emulation of various fixed-grid and flex-grid optical networks, which differ in size and available optical devices.

Next, the overall hierarchical SDN-based inter-domain orchestration platform (NIDO), along with OCEANIA and JULIUS, was deployed and functionally validated in a testbed using OpenStack as a cloud platform, with a Mininet-based emulated, inter-domain network [Fig. 14(a)], with three intra-DC and one inter-DC network domains of variable sizes. The internal topology of each DCN includes three optical planes, each of them with 10 POD switches. Each POD switch is in turn connected to 10 TOR switches. In each DC, the three TOR switches act as gateways toward the inter-DC network. The inter-DC network includes nine core nodes, while three edge nodes connect to each DCN.



Lightpath ID: 5
Source: 9
Destination: 2
Transponder: trx_100
Translucent: No
Hops Number: 2
Path: 9 - 1 - 2
Wavelength: 0

Fig. 13. JULIUS user interface: emulated network topology and established optical connections.

The experiments were based on multiple inter-domain requests to set up and tear down end-to-end connections between hosts located in different DCs, triggered through the NIDO NBI. We executed five different sets of tests depending on the rate (number of requests per second) used to trigger the NIDO orchestrator, with a duration set at 600 s for all the tests. The request rates were: 0.2, 0.5, 1, 1.5, and 2 requests/s. The network orchestrator decomposes the requests in the inter-DC network request part (source and destination DCs) and inter-DC network requests, triggering requests to the NBI of each respective controller.

In Fig. 14(b) we report the average time required for NIDO to perform each sub-task of the orchestration process as a function of the different request rates. We can see that in all cases the computation time is negligible, as NIDO operates on an abstract view of the network that consists of only DCNs/inter-DC networks as nodes and border links as edges. Furthermore, most of the path instantiation time (time between the initial setup request sent to a domain controller and the last intra-domain path activation) is spent establishing the paths in the single domains. The average time between the request's arrival and the path activation is between 4 and 4.5 s, which includes also the overhead intrinsic in the orchestration process, i.e., polling the controllers to check the instantiation results and the resource allocation on the devices.

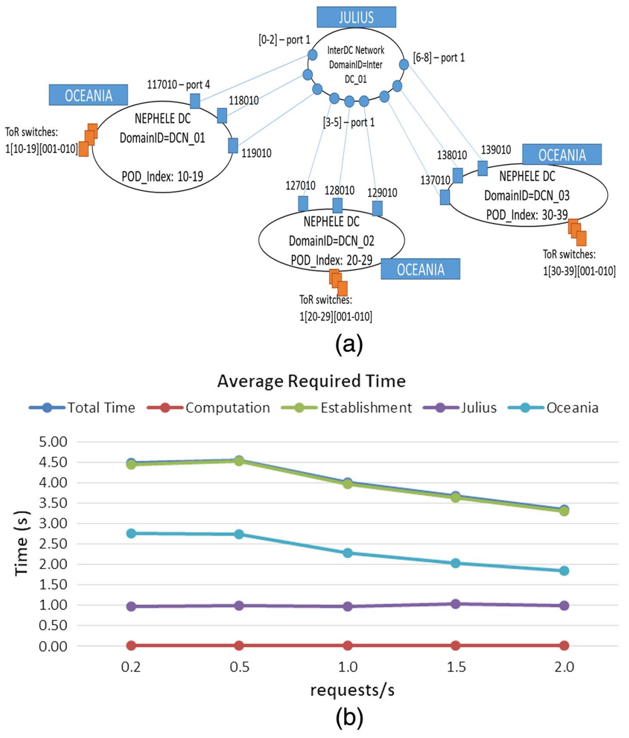


Fig. 14. (a) Virtual network topology for inter-DC testbed and (b) average required time for each sub-task for end-to-end path establishment.

VIII. Conclusion

We present a hierarchical orchestration platform for inter-domain DC networks that includes hybrid electrical/optical intra-DC networks with WDM and TDMA technologies and an elastic optical inter-DC network. The platform utilizes a hierarchy of controllers, where child controllers are responsible for resource allocation in single DCNs and in inter-DC network domains (e.g., based on flex-grid optical technologies), while a parent controller (NIDO) coordinates the end-to-end service provisioning. The platform was functionally validated by demonstrating dynamic and end-to-end allocation of capacity in an emulated inter-domain testbed.

The presented work can be extended in several directions. The introduction of other technological domains at the DC or at the inter-DC network would require the adoption of additional specialized child controllers that would need to interoperate with NIDO. Another interesting topic is to extend the architecture toward infrastructure federations, enabling the integration of services and (virtual) resources from different providers. Finally, the adoption of NEPHELE architectures in NFV environments would require NIDO to interface with NFVOs, for example, acting as a WAN infrastructure manager (WIM).

Acknowledgment

This work received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No. 645212 (NEPHELE).

References

[1] "Cisco Global Cloud Index: Forecast and Methodology, 2015–2020," Cisco White Paper, Oct., 2017.

[2] P. Kokkinos, D. Kalogeras, A. Levin, and E. Varvarigos, "Survey: Live migration and disaster recovery over long-distance networks," *ACM Comput. Surv.*, vol. 49, 26, 2016.

[3] K. Christodoulopoulos, I. Tomkos, and E. A. Varvarigos, "Elastic bandwidth allocation in flexible OFDM-based optical networks," *J. Lightwave Technol.*, vol. 29, no. 9, pp. 1354–1366, 2011.

[4] NEPHELE EU H2020 project [Online]. Available: http://www.nepheleproject.eu/.

[5] G. Landi, M. Capitani, D. Gallico, M. Biancani, and K. Christodoulopoulos, "An application-aware SDN controller for hybrid optical-electrical DC networks," in *Int. Conf. Networks (ICN)*, 2017.

[6] A. Kretsis, L. Corazza, K. Christodoulopoulos, P. Kokkinos, and E. Varvarigos, “An emulation environment for SDN enabled flexible IP/optical networks,” in *Int. Conf. Transparent Optical Networks (ICTON)*, 2016.

[7] R. Casellas, R. Muñoz, R. Martínez, R. Vilalta, L. Liu, T. Tsuritani, I. Morita, V. López, O. de Dios González, and J. P. Fernández-Palacios, “SDN based provisioning orchestration of OpenFlow/GMPLS flexi-grid networks with a stateful hierarchical PCE,” in *Optical Fiber Communication Conf. (OFC)*, 2014.

[8] D. King and A. Farrel, “A PCE-based architecture for application-based network operations,” IETF RFC 7491, Mar. 2015.

[9] R. Muñoz, R. Vilalta, R. Casellas, R. Martinez, T. Szyrkowiec, A. Autenrieth, V. Lopez, and D. Lopez, “Integrated SDN/NFV management and orchestration architecture for dynamic deployment of virtual SDN control instances for virtual tenant networks [Invited],” *J. Opt. Commun. Netw.*, vol. 7, no. 11, pp. B62–B70, 2015.

[10] R. Vilalta, A. Mayoral, R. Muñoz, R. Casellas, and R. Martınez, “Multitenant transport networks with SDN/NFV,” *J. Lightwave Technol.*, vol. 34, no. 6, pp. 1509–1515, 2016.

[11] H. Yang, J. Zhang, Y. Zhao, J. Han, Y. Lin, and Y. Lee, “SUDOI: Software defined networking for ubiquitous data center optical interconnection,” *IEEE Commun. Mag.*, vol. 54, no. 2, pp. 86–95, 2016.

[12] Open Networking Foundation, “OpenFlow switch specification, version 1.3.1,” ONF TS-007, Sept. 2012.

[13] J. P. Vasseur, R. Zhang, N. Bitar, and J. L. Le Roux, “A backward-recursive PCE-based computation (BRPC) procedure to compute shortest constrained inter-domain traffic engineering label switched paths,” IETF RFC 5441, Apr. 2009.

[14] NEPHELE-NIDO [Online]. Available: https://github.com/nextworks-it/nephele-nido.

[15] L. Tomás, P. Kokkinos, V. Anagnostopoulos, O. Feder, D. Kyriazis, K. Meth, E. Varvarigos, and T. Varvarigou, “Disaster recovery layer for distributed OpenStack deployments,” *IEEE Trans. Cloud Comput.*, 2017.

[16] Open vSwitch [Online]. Available: http://openvswitch.org/.

[17] P. Bakopoulos, K. Christodoulopoulos, G. Landi, M. Aziz, E. Zahavi, D. Gallico, R. Pitwon, K. Tokas, I. Patronas, M. Capitani, C. Spatharakis, K. Yiannopoulos, K. Wang, K. Kontodimas, I. Lazarou, P. Wieder, D. I. Reisis, E. M. Varvarigos, M. Biancani, and H. Avramopoulos, “NEPHELE: An end-to-end scalable and dynamically reconfigurable optical architecture for application-aware SDN cloud datacenters,” *IEEE Commun. Mag.*, vol. 56, no. 2, pp. 178–188, 2018.

[18] K. Christodoulopoulos, K. Kontodimas, A. Siokis, K. Yiannopoulos, and E. Varvarigos, “Efficient bandwidth allocation in the NEPHELE optical/electrical datacenter interconnect,” *J. Opt. Commun. Netw.*, vol. 9, no. 12, pp. 1145–1160, Dec. 2017.

[19] OpenDaylight, Mar. 2017 [Online]. Available: https://www.opendaylight.org/.

[20] OCEANIA—NEPHELE SDN controller code, Mar. 2017 [Online]. Available: https://github.com/nextworks-it/oceania-dcn-controller.

[21] G. Landi, I. Patronas, K. Kontodimas, M. Aziz, K. Christodoulopoulos, A. Kyriakos, M. Capitani, A. F. Hamedani, D. Reisis, E. Varvarigos, P. Bakopoulos, and H. Avramopoulos, “SDN control framework with dynamic resource assignment for slotted optical datacenter networks,” in *Optical Fiber Communication Conf. (OFC)*, 2017.

[22] B. Pfaff and B. Davie, “The open vSwitch database management protocol,” IETF RFC 7047, Dec. 2013.

[23] M. Bjorklund, “YANG—A data modeling language for the network configuration protocol (NETCONF),” IETF RFC 6020, Oct. 2010.

[24] A. Kretsis, K. Christodoulopoulos, P. Kokkinos, and E. Varvarigos, “Planning and operating flexible optical networks: Algorithmic issues and tools,” *IEEE Commun. Mag.*, vol. 52, no. 1, pp. 61–69, 2014.

[25] Mininet—An Instant Virtual Network on your Laptop (or other PC) [Online]. Available: http://mininet.org/.

[26] Ryu SDN framework [Online]. Available: http://osrg.github.io/ryu.

[27] LINC-OE Optical Switch Emulation [Online]. Available: https://github.com/FlowForwarding/LINC-Switch.

147.102.22.135 {ts '2018-07-02 04:39:11'}