Grant Agreement No. 619572

COSIGN

Combining Optics and SDN In next Generation data centre Networks

Programme: Information and Communication Technologies

Funding scheme: Collaborative Project – Large-Scale Integrating Project

Deliverable D4.5 – Next Generation Data Centre Resource Orchestration with COSIGN DCN

Due date of deliverable: 31. January 2017
Actual submission date: 31. January 2017

Start date of project: January 1, 2014 Duration: 39 months

Lead contractor for this deliverable: IBM

Project co-funded by the European Commission within the Seventh Framework Programme		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Executive Summary

This is the final report on the orchestrator design, implementation, and the lessons which have been learned, for both COSIGN use-cases: VDC and vApp. The document summarizes the benefits of the use-cases for the service and application layer, and for the control and data planes. It also lists several potential extensions with respect to current trends and needs in the industry.

The document is organized as follows:

Section 2 describes the COSIGN orchestrator benefits and the learnt lessons for both COSIGN use-cases: VDC and vApp. Section 2.1 describes the VDC use-case, whose benefits are demonstrated through emulated network environment by Mininet, controlled by ODL and VDC orchestrator. Section 2.2 describes the vApp use-case, whose benefits are demonstrated through Mininet emulation and network simulation.

Section 3 presents the physical implementation of the use-cases over 3 servers, 3 TU/e ToRs and a Polatis switch. The physical implementation was demonstrated in ECOC 2016.

Section 4 describes the potential extensions of the use-cases with respect to both academia and industry.

Section 5 concludes the deliverable.

Document Information

Status and Version:	D4.5_v6_Final	
Date of Issue:	31/01/2017	
Dissemination level:	Public	
Author(s):	**Name**	**Partner**
	Yaniv Ben-Itzhak	IBM
	Albert Pagès	UPC
	Fernando Agraz	UPC
	Salvatore Spadaro	UPC
	Albert Viñes	I2CAT
	José Ignacio Aznar	I2CAT
	Giada Landi	NXW
	Marco Capitani	NXW
	Domenico Gallico	IRT
Edited by:		
Reviewed by:	Oded Raz	TUE
	Fernando Agraz	UPC
Checked by :	Sarah Ruepp	DTU

Table of Contents

Executive Summary **2**

Table of Contents **4**

1 Introduction **5**

1.1 Reference Material 5
Reference Documents 5
Acronyms and Abbreviations 7

1.2 Document History 7

2 COSIGN Orchestrator Benefits and Lessons Learnt **8**

2.1 VDC use-case 8

2.1.1 Benefits for the application/service layer 8

2.1.2 Benefits for the Control Plane and Data plane layers 9

2.1.3 Potential exploitation benefits of the COSIGN orchestrator 11

2.2 vApp use-case 13

2.2.1 Benefits for the application/service layer 13

2.2.1.1 Scalable Elephant Flow Rerouting 13

2.2.1.2 Reducing OpenFlow Rule Footprint and Setup Rate 13

2.2.1.3 Mitigating OpenFlow Outbound Latency 13

2.2.2 Benefits for the Control Plane and Data plane layers 14

2.2.2.1 Emulation 14

2.2.2.2 Simulation 16

2.2.3 Potential exploitation benefits of the COSIGN orchestrator 17

2.2.3.1 vApp Data-Plane - DCN Flat Upper-Tier Topological Trend 18

2.2.3.2 vApp Control-Plane – Private / Shared Circuit Configuration 18

3 Physical Implementation **20**

3.1 VDC 20

3.2 vApp - ECOC Demo 26

3.3 vApp – Improved Demo 28

4 Potential Orchestrator Extensions **30**

4.1 VDC 30

4.1.1 New Algorithms for Long-Term Demo Technologies 30

4.1.2 Extensions on the VDC Dashboard for Enhanced Administrative Information 30

4.2 vApp 31

4.2.1 Private / Shared-Aware Optical Circuits Scheduler 31

4.2.2 Advanced Optical Circuit Sharing 31

4.2.3 Upper-Tier Topology Design 31

5 Conclusions **32**

6 Appendix **33**

1 Introduction

1.1 Reference Material

Reference Documents

[1]	OpenStack, https://www.openstack.org/software/
[2]	COSIGN, Deliverable D4.3, "COSIGN orchestrator interaction with the COSIGN SDN controller platform"
[3]	COSIGN, Deliverable D4.4, "COSIGN Orchestrator Prototype"
[4]	Albert Pagès et al. "Experimental Assessment of VDC Provisioning in SDN/OpenStack-based DC Infrastructures with Optical DCN", 42nd European Conference and Exhibition on Optical Communication (ECOC 2016), September 2016.
[5]	ETSI, "Network Functions Virtualisation (NFV); Management and Orchestration", ETSI GS NFV-MAN 001 v1.1.1, December 2014
[6]	N. Farrington, G. Porter, S. Radhakrishnan, H. H.Bazzaz, et al. "Helios: a hybrid electrical/optical switch architecture for modular data centers", ACM SIGCOMM Computer Communication Review, 2011.
[7]	Calient. "3D mems optical circuit switching for software defined data centers and metro networks".
[8]	Calient Technologies. "The Software Defined Hybrid Packet Optical Datacenter Network". 2013.
[9]	A. R. Curtis, W. Kim, and P. Yalagandula. "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection", in INFOCOM, 2011 Proceedings IEEE, pages 1629–1637. IEEE, 2011.
[10]	M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. "Hedera: Dynamic flow scheduling for data center networks", In NSDI, volume 10, pages 19–19, 2010.
[11]	T. Benson, A. Akella, and D. A. Maltz. "Network traffic characteristics of data centers in the wild", in Proceedings ACM SIGCOMM conference on Internet measurement, 2010.
[12]	S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. "The nature of data center traffic: measurements & analysis", In Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference, pages 202–208. ACM, 2009.
[13]	D. Y. Huang, K. Yocum, and A. C. Snoeren. "High-fidelity switch models for software-defined network emulation", in Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, pages 43–48. ACM, 2013.
[14]	M. Ku´zniar, P. Perešíni, and D. Kosti´c. "What you need to know about SDN flow tables", in Passive and Active Measurement, pages 347–359. Springer, 2015.
[15]	A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. "DevoFlow: scaling flow management for high-performance networks", in ACM SIGCOMM Computer Communication Review, volume 41, pages 254–265. ACM, 2011.
[16]	K. He, J. Khalid, S. Das, A. Gember-Jacobson, C. Prakash, A. Akella, L. E. Li, and M. Thottan. "Latency in software defined networks: Measurements and mitigation techniques", in Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, pages 435–436. ACM, 2015.
[17]	C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore. "OFLOPS: An open framework for OpenFlow switch evaluation", in Passive and Active Measurement, pages 85–95. Springer, 2012.
[18]	J. Kim, W. J. Dally, and D. Abts. "Flattened butterfly: a cost-efficient topology for high-radix networks", ACM SIGARCH Computer Architecture News, 35(2):126–137, 2007.
[19]	A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren. "Inside the social network's (datacenter) network", in ACM SIGCOMM Computer Communication Review, volume 45, pages 123–137. ACM, 2015.

[20]	D. Abts, M. R. Marty, P. M. Wells, P. Klausler, and H. Liu. “Energy proportional datacenter networks”, in ACM SIGARCH Computer Architecture News, volume 38, pages 338–347. ACM, 2010.
[21]	H. Liu, F. Lu, A. Forencich, R. Kapoor, M. Tewari, G. M. Voelker, G. Papen, A. C. Snoeren, and G. Porter. “Circuit switching under the radar with REACToR”, in 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), pages 1–15, 2014.
[22]	H. Liu, M. K. Mukerjee, C. Li, N. Feltman, G. Papen, S. Savage, S. Seshan, G. M. Voelker, D. G. Andersen, M. Kaminsky, et al. “Scheduling techniques for hybrid circuit/packet networks”, CoNEXT, 2015.
[23]	M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. “Data Center TCP (DCTCP)”, in ACM SIGCOMM computer communication review, volume 40, pages 63–74. ACM, 2010.
[24]	A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. “Vl2: a scalable and flexible data center network”, in ACM SIGCOMM computer communication review, volume 39, pages 51–62. ACM, 2009.
[25]	G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. Ng, M. Kozuch, and M. Ryan. “c-Through: Part-time optics in data centers”, ACM SIGCOMM Computer Communication Review, 41(4):327–338, 2011.
[26]	B. Lantz, B. Heller, and N. McKeown. “A network in a laptop: rapid prototyping for software-defined networks”, in Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, page 19. ACM, 2010.
[27]	M. Wang, B. Li, and Z. Li. “sFlow: towards resource-efficient and agile service federation in service overlay networks”, in Distributed Computing Systems, 2004. Proceedings. 24th International Conference on, pages 628–635. IEEE, 2004.
[28]	Polatis 6000n Protection Services Switch Data Sheet. http://www.polatis.com/datasheets/products/Polatis_6000n_Protection_Services_Switch_Data_Sheet.pdf
[29]	iPerf - The TCP, UDP and SCTP network bandwidth measurement tool. https://iperf.fr/.
[30]	Plexxi Data Sheet: "Big Data Fabric". http://www.plexxi.com/wp-content/uploads/2014/07/DS_PLX_BDF20140701.pdf
[31]	High-Density 25/100 Gigabit Ethernet StrataXGS Tomahawk Ethernet Switch Series. https://www.broadcom.com/products/ethernet-connectivity/switch-fabric/bcm56960
[32]	J. Kim, W. J. Dally, S. Scott, and D. Abts. “Technology-driven, highly-scalable dragonfly topology”, in ACM SIGARCH Computer Architecture News, volume 36, pages 77–88. IEEE Computer Society, 2008.
[33]	J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber. “Hyperx: topology, routing, and packaging of efficient large-scale networks”, in Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, page 41. ACM, 2009.
[34]	B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker. “Extending networking into the virtualization layer”, in Hotnets, 2009.
[35]	S. Bojja, M. Alizadeh, and P. Viswanath. “Costly circuits, submodular schedules and approximate Carathéodory theorems”, ACM Sigmetrics, 2015.
[36]	M. Chowdhury and I. Stoica. Coflow: A networking abstraction for cluster applications. In Proceedings of the 11th ACM Workshop on Hot Topics in Networks, pages 31–36. ACM, 2012.

Acronyms and Abbreviations

Most frequently used acronyms in the Deliverable are listed below. Additional acronyms may be defined and used throughout the text.

CP	Control Plane
DC	Data Centre
DCN	Data Centre Network
DoW	Description of Work
DSCP	Differentiated Services Code Point
EPS	Electrical Packet Switch
NIC	Network Interface Card
OCS	Optical Circuit Switch
ODL	OpenDaylight
OF	Open Flow
QoS	Quality of Service
RTT	Round Trip Time
SDN	Software Defined Networking
TDM	Time Division Multiplexing
ToR	Top of Rack
VDC	Virtual Data Centre
VM	Virtual Machine

1.2 Document History

Version	Date	Authors	Comment
01	5/10/2016	See the list of authors	Table of content
02	15/01/2017		1st version – merged
03	18/01/2017		2nd version – merged
04	25/01/2017		Add new section about ECOC demos
05	30/01/2017		Version for quality check
06	31/01/2017		Final version

2 COSIGN Orchestrator Benefits and Lessons Learnt

2.1 VDC use-case

The development of the COSIGN orchestrator for the Virtual Data Centre (VDC) use case aimed at providing an automated and intuitive way to create VDC instances on demand according to the requirements of the end user (i.e. the tenant) while coordinating the provisioning of the computing and network resources to achieve an optimal utilization of the data plane infrastructure. To this end, the development of the orchestrator involved the customization of some of the OpenStack services (mainly the Horizon dashboard service) while taking advantage of several others (Heat, Nova, Neutron, Keystone) [1], the creation and development of a completely new Algorithms module dedicated to the computation of the optimal mapping of VDC instances and several client modules to interact with the OpenStack software suite. Additionally, to achieve a global view of the DC network (DCN) infrastructure and be able to provide the virtual link route mapping details to the COSIGN Control Plane (CP), specific interfaces to interact with the OpenDaylight (ODL) controller have also been developed. The details of the software development, its functionalities and integration have been reported in deliverables D4.3 [2] and D4.4 [3]. The full integrated orchestrator has been publicly show cased at the COSIGN demonstrator held in ECOC 2016 in Düsseldorf, where we successfully demonstrated the automated and dynamic configuration of VDC instances employing the COSIGN architecture. In what follows below, we will elaborate on the benefits that the COSIGN orchestrator brings to the other layers besides the orchestration layer. Additionally, we will discuss on potential exploitation of the developed software beyond the scope of the COSIGN project.

2.1.1 Benefits for the application/service layer

The main benefit from the application/service layer arises from the extended Horizon dashboard service, which has been customized to support the graphical creation of VDC instances according to the tenant's needs. A VDC instance is composed by a set of virtual nodes, with each virtual node having computational capabilities in the form of a number of Virtual Machines (VMs). Then, virtual nodes are interconnected through virtual links stating the desired bit-rate between them. From OpenStack perspective, the needed resources to create a VDC instance translate to a set of Nova servers, which are the VM instances. For each VM the tenant can select the desired characteristics in terms of resources as well as the operating system. Regarding the virtual network connectivity, OpenStack allows for the creation and configuration of the associated IP network to achieve that, however, with no specific control over the desired bit-rate. In this regard, the extended VDC dashboard adds the possibility to specify for each virtual link an independent bit-rate, which then, through the collaboration of the Algorithms module and the ODL controller, is achieved by properly configuring the data plane.

Additionally, the extended VDC dashboard makes the whole provisioning of a VDC instance much simpler, with less involved steps and operations. In standard OpenStack, it is required that the tenant firstly creates the IP network through the Network tab in the dashboard. In this section, the tenant can specify the characteristics of the IP network (e.g. address range, subnetwork, DHCP). However, all configuration parameters have to be input through textual menus and pop-ups. The next step involved in the creation of a VDC instance relates to the configuration and creation of the VMs. For this, the tenant has to access the computing tab of the dashboard. In this section, the characteristics of the VMs can be specified similar to the creation of the IP network (through textual menus). Thus, the overall process requires the tenant to navigate through multiple menus in the dashboard, with the addition that all resource specification is made through textual menus, making it a tedious process depending on the size and complexity of the desired VDC instance. Conversely, if the Heat service is installed in the OpenStack deployment, a tenant could opt to create all the necessary VDC resources through the orchestration tab in the dashboard. This section allows for the possibility to define a stack, that is, a collection of OpenStack resources matching the desired configuration and send it to the Heat service for its instantiation. However, this process requires that the tenant inputs, manually or in a file, the YAML template of the stack, which requires a specific knowledge about the types of resources that

OpenStack admits, their parameters and the overall template syntax. Thus, this option requires an advanced knowledge from the tenant’s side.

With the extended Horizon dashboard in the COSIGN orchestrator, it is possible to circumvent the complexities associated with the deployment of all the resources of a VDC instance. A new section on the dashboard has been developed which allows for the graphical creation of all resources associated to a VDC instance. That is, the extended dashboard allows to actually draw the desired topology for the VDC with as many virtual nodes as desired and the virtual links interconnecting them. Then, through simple menus, it is possible to specify the number of VMs and their characteristics per virtual node as well as the bit-rate for the virtual link. Once graphically specified, the details of the VDC instance are sent to the Algorithms module which, after computing the VDC mapping, will create a YAML template which contains all the OpenStack resources necessary for the instantiation of a VDC (IP network, Nova servers, etc.). In this regard, the whole interaction with the core OpenStack services is completely transparent to the tenant. Additionally, it automatizes all the necessary steps to achieve the creation of the VDC stack thus making the process a lot simpler. An additional characteristic of the extended dashboard is that it also supports the graphical update of a deployed VDC instance. That is, it allows to add new nodes or links, modify characteristics of the deployed resources or delete old ones, all from the same graphical view employed during the creation of the VDC instance. Once the VDC graphical representation has been edited according to the desired modification, its details will be passed down to the Algorithms module to start the update of the deployed stack. All this process is made completely transparent to the tenant. Lastly, the extended dashboard allows for a one click deletion of the deployed VDC instance, not requiring that the tenant interacts with the multiple tabs associated to each type of resource to completely erase the stack deployed. All these operations are enabled thanks to the collaboration between the Horizon dashboard service and the newly designed Algorithms module, which is the responsible to interact with Heat, Nova and Neutron for the configuration/update/deletion of a VDC instance. In summary, the COSIGN orchestrator brings a richer and more intuitive interface for the creation of VDC instances, benefiting the provisioning of services to tenants.

2.1.2 Benefits for the Control Plane and Data plane layers

In legacy architectures, where an orchestrator layer is not present, the configuration of the VMs and the network is not done jointly, that is, with no shared knowledge between the two operations. This usually leads to underperformance of the DC infrastructure since most software solutions for the deployment of VMs only consider the load of the computing resources (i.e. the servers), with little to none regard to the actual network load, leading to potential network congestions. In this regard, the introduction of an orchestration layer is highly beneficial since it allows for the coordination of the configuration of all VDC resources (computing and networking) to reach an overall optimized utilization of the data plane resources. With this goal in mind, the COSIGN orchestrator has introduced a new Algorithms module whose purpose is to determine the mapping of VDC instances with the aim to optimize the data plane utilization. In this regard, the Algorithms module performs all the calculations necessary to decide the placement of the VMs onto the servers and the optical paths to be configured to satisfy the connectivity requirements of the virtual links. This joint calculation brings basically two main benefits: the first and most important one is the optimal utilization of the physical resources of the DC. By optimizing the usage of the physical resources, it is possible to increase the number of supported VDC instances on top of the shared DC infrastructure. The second benefit relates to the calculation of the optical paths. In a situation where the orchestrator is not present, the path calculation and configuration is made solely by the CP, which is responsible for the provisioning of the optical connectivity between servers. With the inclusion of the orchestrator plane, all calculations are made at the COSIGN orchestrator, so it lightens the computational burden of the CP which will only be an enforcer of the decisions taken at the orchestrator level. Additionally, the orchestrator allows for an automated and dynamic configuration of VDC instances, interacting with the CP as needed. We have demonstrated the automated VDC provisioning considering the whole COSIGN architecture (data, control and orchestrator planes) in [4], where we experimentally analysed the incurred delays in all layers for the provisioning of a single VDC instance. The obtained results confirmed that seamless VDC provisioning can be efficiently achieved at the COSIGN orchestrator, with the necessary time to calculate the optimal mapping of the VDC resources being less than one second.

To further evaluate the benefits of the joint resource provisioning that the COSIGN orchestrator allows, we executed additional experiments. For this purpose, we considered an emulated data plane (with Mininet) consisting on three clusters of 7 ToRs each interconnected in a tree fashion to a central Polatis switch per cluster. Then, all Polatis switches are interconnected in a full mesh fashion. Each pair of nodes is interconnected with 24 fibre links. Additionally, all racks are equipped with 40 servers per rack. With such data plane scenario, we analysed the enhanced service acceptance due to the optimized resource utilization that the COSIGN orchestrator can bring compared with the case where the VM placement and the network path calculation is done separately (non-jointly), at the orchestrator and at the network controller level respectively.

To this purpose, we considered the deployment of 100 VDC instances. We generated the VDCs interconnecting in a full mesh fashion between 2 and 5 VMs with virtual links. The characteristics of the VMs are based on the default VM configurations in OpenStack. As for the links, we considered that their bandwidth is chosen randomly from the set {10, 100, 1000} Mb/s. To extract more in depth conclusions, we considered three different configurations of VDCs: a) 2-3 nodes; b) 3-4 nodes; and c) 4-5 nodes. With such configurations, the VDC instances are sent one by one from a VDC generator to the COSIGN orchestrator which calculates the optimal VDC mapping and proceeds with the whole resource deployment interacting with the CP for the network configuration. Figure 1 depicts a schematic of the employed experimental setup.

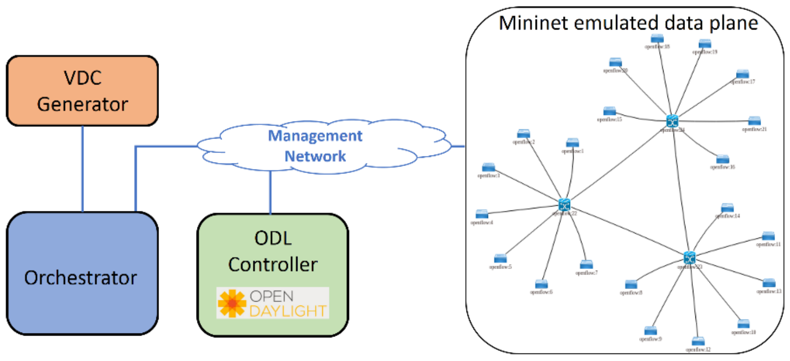


Figure 1 - Employed experimental setup

In the non-joint provisioning case, the Algorithms module present in the COSIGN orchestrator is not used. Hence, the mapping of the VMs is decided solely considering the load balancing of the servers, and the route calculation is done using the path computation module of the ODL controller, which uses a shortest path approach. With such considerations, Figure 2 reports the VDC acceptance ratio for the three different configurations for both joint and non-joint scenarios.

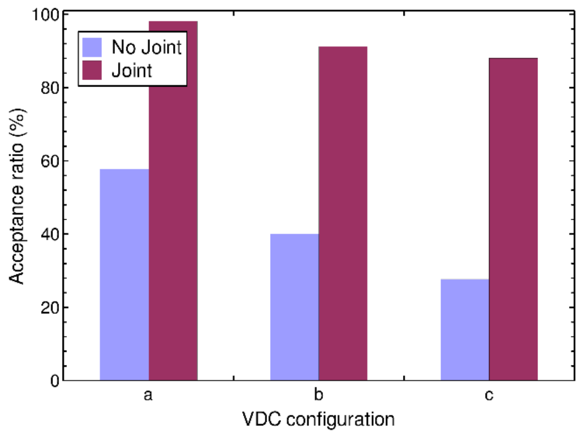


Figure 2 – VDC acceptance ratio

It can be appreciated that the use of the COSIGN orchestrator substantially increases the acceptance of VDC instances. This is due to the aforementioned joint calculation considered for the VDC mapping, which includes information regarding both computing and network resource status to decide on the most optimal placement of the VMs and the establishment of the optical paths for a given VDC instance.

2.1.3 Potential exploitation benefits of the COSIGN orchestrator

Aside from the previously demonstrated enhanced VDC provisioning, the development of the COSIGN orchestrator opens up the possibility to be employed in the testing of joint resource orchestration in scenarios where complex virtual infrastructure provisioning is needed in infrastructures encompassing several types of resources. Thanks to the open nature of the Algorithms module, it is possible to add/modify the mapping algorithms to consider the presence of any kind of physical infrastructure or service. In addition, thanks to the REST interfaces employed to interact with upper modules (such as the Horizon dashboard) and lower ones (such as the ODL controller), it could be theoretically possible to interact with other control planes and upper application layers, granted that new interfaces are designed for this purpose, thus extending the utilization of the core functionalities of the COSIGN orchestrator to other scenarios. In this regard, the COSIGN orchestrator is a very flexible tool that academic and research partners can employ in later projects or internal initiatives to determine the benefits of joint resource orchestration in multiple scenarios besides the one targeted by COSIGN.

Moreover, the flexibility of the orchestrator design as an external plugin, is key for an easy adaptation to further different scenarios involving cloud resource orchestration. An example can be found in the context of NFV MANO (Network Function Virtualization MANagement and Orchestration [5]) tools. In the NFV MANO architecture, an orchestration entity like the NFVO (NFV Orchestrator) is responsible for coordinating the allocation of computing and network resources to deploy Virtual Network Functions (VNFs) and Service Function Chains (SFCs) in a virtualized environment controlled through a VIM (Virtual Infrastructure Manager), for example OpenStack. In this scenario the COSIGN orchestrator algorithms can be integrated as a third party module in the NFVO, with the role of taking decisions about the placement of the VMs composing the tenants' VNFs over the available cloud resources, based on the specific network-related constraints of the VNFs' applications and their network graphs.

Additionally, the development of the COSIGN orchestrator has provided academic and research partners the opportunity to gain knowledge about the OpenStack cloud management software platform. Being OpenStack the de facto open source software platform for cloud infrastructures management, COSIGN partners can leverage the lessons learnt about development in OpenStack and the orchestration of complex cloud and telecom infrastructures for later research projects, either national or international ones, as well as for providing educational training or consultancy about this topic.

In terms of industrial exploitability, the concepts of the COSIGN orchestrator in the context of the VDC use case are fully applicable to cloud services offered by cloud providers like Interoute. An example is the Virtual Data Centre[1] service, an IaaS platform providing cloud resources integrated with network services built on the global fibre optic network owned and managed by Interoute. Keys of the Interoute VDC service are low-latency and high throughput at the network level, supporting distributed and highly scalable virtual applications running on Interoute infrastructures with predictable and customizable service levels. The COSIGN orchestrator approach, applied to Interoute VDC, brings several benefits from the operator's perspective (e.g. in terms of management automation and resource allocation efficiency), but also widening the service features offered to the customers (e.g. flexible QoS guarantees, specialized and customizable network configurations). In particular, the following aspects of the COSIGN orchestrator are particularly relevant to drive the evolution of Interoute VDC service:

- Automation of VDC provisioning for complex scenarios requiring particular network configuration, in order to extend the "self-service" delivery of VDC instances to a wider portfolio of VDC templates. This is a fundamental feature to simplify the interaction with the

[1] https://cloudstore.interoute.com/what_is_vdc

user as well as to reduce the operational cost of the cloud infrastructure and the manual management of service delivery.

- Enhanced dashboard for interactive VDC service requests and management for the customers. The dashboard could be further extended integrating a sort of marketplace offering pre-defined VDC templates for the most common applications and a number of options that facilitate the virtual infrastructure tuning to specific customer's needs.
- Improvements in QoS guarantees for the virtual network infrastructures offered to the customer through the adoption of virtual optical networks inside the DC. This becomes even more relevant when applied over a physical infrastructure based on COSIGN long-term scenario technologies and a Software Defined Networking (SDN)-based controller specialized to fully exploit their flexibility and allocation granularity. In fact, in this kind of DC infrastructure, the COSIGN solution as a whole is able to provide highly customizable virtual networks, which can be tailored to the requirements of different applications guaranteeing the desired service level through the delivery of per-tenant network slices with reserved resources. However, the fine tuning of the optical resources allows for efficient resource allocation at the physical DC level reducing over-provisioning and thus supporting a higher number of customers over the same shared infrastructure.

In order to maximize the exploitation of COSIGN solution beyond the intra-DC scenario, the orchestration of DC resources should be further integrated in the whole management of a VDC service, which spans across different geographical areas (i.e. the VDC zones), and become part of a global resource allocation strategy which automates service delivery not only at the DC level but also in the inter-DC segment.

2.2 vApp use-case

2.2.1 Benefits for the application/service layer

2.2.1.1 Scalable Elephant Flow Rerouting

State-of-the-art solutions, e.g. [6], which utilize the OCS plane by elephant flows, introduce a coupled architecture in which both the detection and rerouting of elephant flows are employed over the switches that are directly connected to the OCS plane.

In particular, for OpenFlow based solutions [7], such coupling dictates the installation of an OpenFlow rule for each detected elephant flow in order to reroute it to the OCS plane [8].

Other OpenFlow-based solutions (e.g., Mahout [9] and Hedera [10]) present different elephant flow detection and load-balancing solutions; both are employed by a specific OpenFlow rule for each detected elephant flow.

This approach is referred to as *per-flow setup*.

By exploiting the elephant flows DSCP tagging and the packet metadata assignment according to their corresponding input ports, vApp results in a single OpenFlow rule for each switch that is connected to an optical circuit's end-point.

2.2.1.2 Reducing OpenFlow Rule Footprint and Setup Rate

Assuming that there are on average 1k simultaneous elephant flows [11] [12] between two packet switches at the upper tier, means that existing approaches require 1k OpenFlow rules for each of the packet switches, which might consume most of current OpenFlow switches flow table size. For instance, HP ProCurve 5400zl switches support up to 1.7K OpenFlow entries [9]; HP ProCurve J9451A supports 1.5k OpenFlow entries [13]; HP ProCurve 5406zl, Pica8 P-3290, and Dell PowerConnect 8132F support up to 1.5k, 2k and 750 rules, respectively [14]. Hence, the currently used *per-flow setup* approach results in average flow table consumption of 50%-67% for elephant flows rerouting. The significant OpenFlow entry footprint has also been observed in [15].

Since vApp requires a single OpenFlow rule for each circuit, it results in a significantly smaller OpenFlow footprint. Furthermore, OpenFlow switches have limited OpenFlow rule setup rate. For instance, [13] indicates that OpenFlow rule setup rate is limited to approximately 40 flow/sec. Clearly, vApp significantly reduces the required OpenFlow setup rate; hence, proposes feasible solution for current OpenFlow switches, as opposed to the *per-flow setup* approach, which results in a limited scale solution.

2.2.1.3 Mitigating OpenFlow Outbound Latency

Once an optical circuit is configured, subsequent ingress elephant flows arriving to the packet switches are matched by the already installed OpenFlow rule and transmitted through the optical circuit. Consequently, vApp mitigates the OpenFlow outbound latency[2] for such subsequent flows. The OpenFlow outbound latency has been measured in previous works: [16] and [14] report that the OpenFlow outbound latency can be as high as 30ms or 400ms, respectively. [17] measures the outbound latency of two switches by using OFLOPS, and report ranges of 50-1000ms and 8-2000ms depending on the number of inserted flow entries.

Per-flow setup approach dictates that each new ingress elephant flow is transmitted through the OCS plane only after the OpenFlow outbound latency, and in the meantime transmitted through the EPS plane or buffered.

[2] The latency of the switch to install/modify/delete OpenFlow rules provided by the SDN controller.

Therefore, by avoiding the OpenFlow outbound latency for each subsequent elephant flow served by an optical circuit, vApp better utilizes the OCS plane, and achieves better network throughput.

2.2.2 Benefits for the Control Plane and Data plane layers

In this section, we present the evaluation of vApp by both simulation and evaluation.

The list below provides a detail on the tools and metrics that we have used for such evaluation:

- **Topologies:** we evaluate vApp for two flat upper tier topologies: Ring and Flatted Butterfly [18]. Ring topology offers a simple wire connectivity, and is used by industrial DCNs. Facebook [19] presents a DCN architecture which uses Ring topology to connect the cluster and aggregation switches. Flattened butterfly (FBFly) takes advantage of high-radix switches to create a scalable, yet low-diameter network. Google [20] show that FBFly is a power efficient topology for high-performance DC networks. For both topologies, the bandwidth of the packet and circuit links are set for 1/10 ratio, as used by [21].
- We use two DCN traces to simulate aggregated traffic to the upper packet tier, with skewed and uniform traffic patterns.
  1. **Traces from the University of Wisconsin (*UNI1*)** are presented in [11], which contain recorded traffic among approximately 2900 servers for a one hour duration. Analysis of these traces by [22] shows mostly *sparse and skewed traffic*. We analyse *UNI1* pcap traces and extract the TCP sessions properties, and their start time. Then, in order to simulate DCNs with different number of hosts, we consolidate the hosts by subnets, and merge the traffic for each subnet to represent a node in our modified trace. The subnet sizes are chosen accordingly to meet the required number of hosts. In addition, we reduce the time intervals between the sessions to obtain moderate network load.
  2. **Synthetic Data Center Trace (*Uniform*)** is created based on traffic characteristics from [10], [23], [12], [21], such that elephant flows are 10% of the number of flows and accommodate 90% of the demand. We generate traffic with random distribution of sessions between mice flow traffic (2KB to 32KB) and elephant flows (up to 100MB) [23], [24], with *uniform traffic* distribution [10].
- **Scheduling:** We use a traditional maximal-weight-matching (MWM) approach [25], which targets the maximal demand offloading to the OCS plane. We recalculate the MWM over the estimated demand and reconfigure the OCS accordingly.

2.2.2.1 Emulation

We develop an emulated environment of vApp by using Mininet [26] version 2.2.1 running over an IBM x3550 M4 server with 196GB of RAM, 24 Xeon-E5-2630@2.3GHz CPUs (with six cores each), and Ubuntu 14.04 with Linux 3.19 kernel. We use sFlow [27] to sample the egress flows of the hosts by the *Elephant Flow Detector*, and to sample the Open vSwitches by the *Network Observer*. The OCS is emulated by a constrained Open vSwitch to employ optical circuits, such that only one input port can be configured to transmit to any given output port. Each OCS reconfiguration is emulated by first removing the colliding optical circuits, and configuring the new requested optical circuits after a 20ms delay to emulate 3D MEMS OCS typical reconfiguration penalty, e.g. [28]. We evaluate an upper tier Ring with 10 packet switches and 3-ary-3-flat FBFly (9 packet switches) with packet and circuit links of 10 and 100 Mbps, respectively. The network traffic is generated by iperf3 [29] according to *UNI1* and uniform traces configured for moderated network load without hitting the CPU-bound of the server.

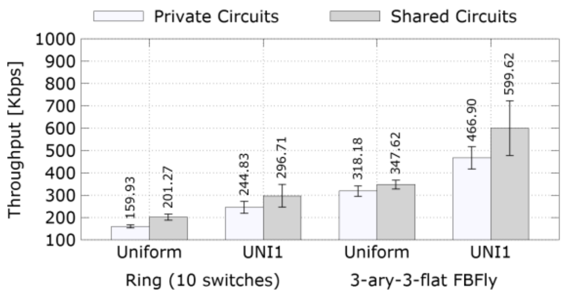
Table 1 presents the improvement of the average network throughput per flow, by adding the OCS plane and employing *shared* circuits. As can be seen, installing OCS plane can significantly improve the network throughput, mostly for uniform traffic pattern, and for ring topologies which inherently offer limited connectivity.

Table 1. Average network throughput per flow [Mbps] comparison, with and without OCS plane.

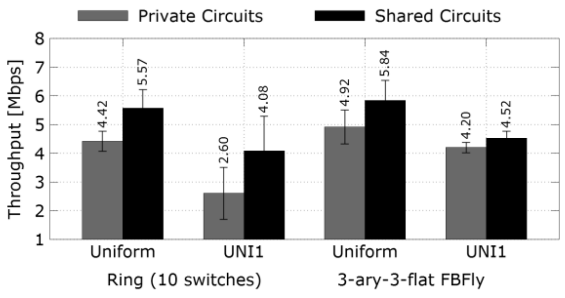
Topology	Trace	w/o OCS	w OCS	**Ratio**
Ring (10 Switches)	Uniform	0.169	0.894	**5.29**
	UNI1	0.225	0.829	**3.68**
3-ary 3-flat FBFly	Uniform	0.297	1.001	**3.37**
	UNI1	0.815	1.095	**1.34**

Below, we demonstrate the advantage of the *shared* circuits, as compared to *private* circuits.

We evaluate the average throughput improvement of the mice and elephant flows for each of the circuit configurations.



(a) Average Mice Throughput



(b) Average Elephant Throughput

Figure 3. Average throughput as reported by iperf3 for Mininet environment under moderate network load.

Figure 3 presents a comparison of the average throughput as reported by iperf3 between mice and elephant flows, for both network traces over the Ring and FBFly topologies. In general, *shared* circuits improve the throughput of both elephant and mice flows as compared to *private* circuits.

In particular, we observe that: (a) Skewed traffic (*UNI1* trace) introduces patterns which can be exploited by *shared* circuits, such as many elephant flows from different sources to the same destination. Therefore, *shared* circuits further improve the network performance of skewed traffic, for instance by 29% for mice flows over FBFly, and 57% for elephant flows over Ring; whereas, uniform traffic is improved by 9% and 26%, respectively. (b) The connectivity of Ring topology is limited, which results in degraded performance as compared to FBFly. Therefore, the connectivity and network throughput of Ring topology can be further improved by the *shared* circuits. In particular, the *shared* circuits improve the network throughput of Ring topology by 21%-57%, as compared to FBFly which is improved by 8%-28%.

Table 2 presents the OpenFlow rule footprint of *UNI1* and uniform traces, under moderate network load. vApp reduces the OpenFlow footprint by 90-95% as compared to the current approach.

Table 2. OpenFlow rule footprint for elephant flow rerouting during one minute of trace, under moderate network load. VApp significantly reduces the OpenFlow footprint.

Trace|Method	Topology|Circuit	Ring (10 Switches) Private	Ring (10 Switches) Shared	3-ary-3-flat FBFly Private	3-ary-3-flat FBFly Shared
UNI1	*Per-flow setup*	445	449	398	384
	vApp	26	31	20	17
Uniform	*Per-flow setup*	563	588	486	435
	vApp	45	52	37	31

2.2.2.2 Simulation

We use an event driven simulation to evaluate the completion time of mice coflows (An abstraction for a collection of flows with a shared completion time [36]) and elephant flows, and measure the corresponding OpenFlow rule footprint for rerouting elephant flows by the packet switches through private or shared circuits. We use the synthetic uniform traces to demonstrate the scalability of vApp under intensive network load. Specifically, we generate network traffic comprised of mice coflows and elephant flows. The mice coflows are 90% of the number of flows, and accommodate 10% of the total demand. We simulate Ring and Flattened Butterfly upper packet tier topologies, with varied number of packet switches. Each packet switch serves 40 hosts. The packet and circuit links are set to 10Gbps and 100Gbps, respectively.

Figure 4 presents the average completion run-time of mice coflows and elephant flows for private and shared circuit configurations over 60 trials. The shared circuits improve the average completion time by 20% for a Ring with 10 switches and up to 30% for a Ring with 16 switches. The Ring topology is unscalable in terms of connectivity. Therefore, the *shared* circuits can significantly increase the topology connectivity and mice/elephant flow separation, which results in increased improvement of the completion time as the Ring size increases. On the other hand, since FBFly is scalable, the improvement of the completion time by shared circuits equals 15%-20% for all FBFly sizes; hence, the shared circuits results in relatively constant mice/elephant flow separation degree. The same applies for the OpenFlow rule footprint presented in Figure 5. The required OpenFlow rules of per-flow setup for Ring remains constant and higher than 1.7k (prevalent OpenFlow table size [9], [13], [14]). On the other hand, due to the scalability of FBFly, as the size of FBFly increases, less OpenFlow rules are required for rerouting the elephant flows through private or shared circuits. However, the OpenFlow footprint of per-flow setup is still high and is significantly reduced by vApp.

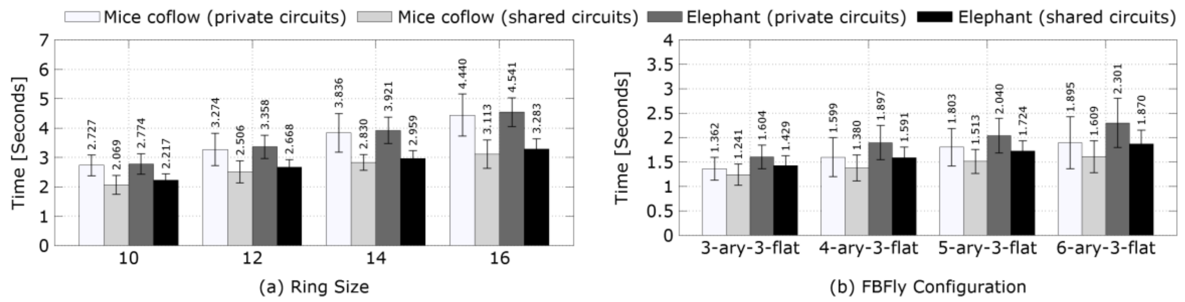


Figure 4. Average completion time of mice coflows and elephant flows under intensive network load, over two upper tier topologies: (a) Ring (10 to 16 packet switches) and (b) FBFly (9 to 36 packet switches).

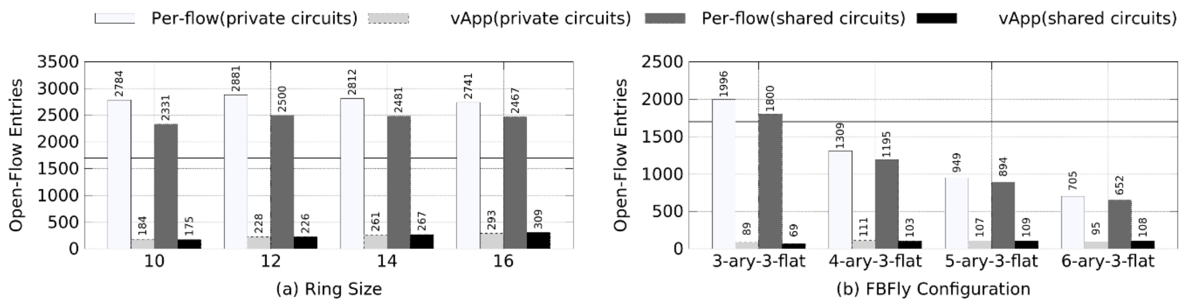


Figure 5. OpenFlow rule footprint per switch during one minute of network trace under intensive network load,over two upper tier topologies: (a) Ring (10 to 16 packet switches) and (b) FBFly (9 to 36 packet switches). The horizontal line indicates 1.7k OpenFlow rule entries. Any OpenFlow entries count above it might be an unfeasible scenario.

2.2.3 Potential exploitation benefits of the COSIGN orchestrator

The characterization and profiling of network traffic plays an essential role for an efficient management of a generic data network. In the context of an intra-DC environment the majority of TCP flows tend to be short because they are mostly related to bursty network traffic or to latency-sensitive applications, whereas the majority of packets are related to a few large flows (namely elephant flows). The characteristics of these kinds of flows can be summarized in three macro categories based on their duration and data rate:

- Short duration flows with high data rate
- Long duration flows with low data rate
- Long duration flows with high data rate.

The elephant flows are generally related to large data transfer such as backups or data migrations that can be generated both by DC's customers and by DC's owner for different kind of back-end operations. The creation of these flows has an obvious impact on the overall DC network performance, since they tend to congest end-to-end network buffers thus introducing significant delay to all the latency-sensitive mice flows sharing the same buffer, leading to performance degradation of the network.

The efficient management of elephant flows in a legacy DC network is affected by the lack of a centralized, high-level point of view on the whole network infrastructure. Even in a SDN-compliant DC network the physical switches are equipped with powerful dedicated ASICs for data plane switching, but weak CPUs for control plane or any other SDN defined task. Furthermore, as shown in the previous section, there is an upper bound limit in the number of OpenFlow rule entries that can be stored and managed by an SDN-compliant device, thus reducing the performance in an intensive network workload scenario.

In this context, the features provided by the COSIGN orchestrator for the vApp scenario bring a remarkable contribution to the optimization of elephant flow management. The optimization process starts from the interaction between the data plane through the physical and virtual observer that guarantees a continuous monitoring of the topology and of the flows crossing the physical and virtual layer network. The information retrieved by these two orchestrator components is provided to the Orchestrator Algorithms module which is in charge of configuring the optical DCN's circuits. This centralized approach together with the high configurability of the COSIGN orchestrator, allows DC owner and operator to fully exploit the network capabilities, increasing its usability and overall performance.

Being more practical, an effective management of elephant flows has an immediate and positive impact from the operational, and thus business, point of view. For example many different data intensive applications e.g. database synchronization, backups, data transfer for back-end operations, data migration or VMs migration can be delivered minimizing the impact on all the other traffic flows and preventing the possibility of SLA breach.

vApp requires a specific data-plane topology, and minor changes in the control plane. In turn, it offers substantial improvement in the network performance, both in terms of throughput and RTT, while reducing the OpenFlow foot-print over the OpenFlow switches at the upper flat topology.

In the following, we discuss the current trends which support the usage of such upper flat topology which enables the use of vApp, and describe the method which are required to employ vApp by the control plane.

In overall, the data-plane flat topology is already employed by several industrial data-centers, and the required control plane modifications are minor. Therefore, exploitation of vApp is feasible, practical, and can significantly improve the network performance.

2.2.3.1 vApp Data-Plane - DCN Flat Upper-Tier Topological Trend

Current production DCN topologies are based on a multi-tier EPS with a flat upper-tier. For instance, Facebook [19] and Plexxi [30] connect the switches at the upper-tier by a ring topology and use a hierarchical structure at the lower tiers.

In addition, current DCN switches offer up to 128 ports of 25 Gbps [31]. In the near future, switches with 256 ports of 25 Gbps are expected and apparently will be followed by switches with 256 ports of 50 Gbps. As the port density increases, data-center networks become flatter with flat upper tier topology; i.e., the upper-tier switches are intra-connected, thus omitting the need for an additional network tier. There are several well-known topologies, such as, multi-dimensional torus or mesh, Flattened Butterfly (FBFly) [18], Dragonfly [32], and HyperX [33] that offer flat and scalable topology. For instance, FBFly can support 20k nodes with radix-32 switches, and DragonFly scales to over 256k nodes with radix-64 switches.

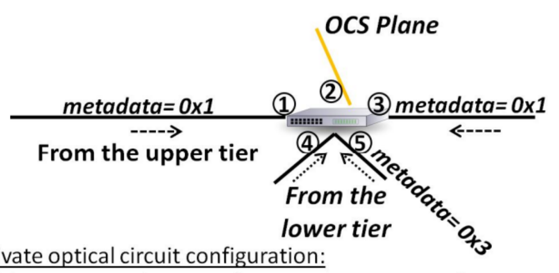
2.2.3.2 vApp Control-Plane – Private / Shared Circuit Configuration

Private and *shared* optical circuits are differed by setting which of the switch's input ports are matched by the rerouting rule of elephant flows through the optical circuit. Therefore, different metadata values are assigned to packets from input ports connected to the lower and the upper packet tiers. Then, by mask matching on the metadata value of an ingress packet, one can configure the switch either to use the optical circuit as *private* by serving only packets from the lower tier, or *shared* by serving packets also from the upper tier.

Figure 6 demonstrates Open vSwitch [34] configuration for *private* and *shared* circuits. At initialization, metadata values of 0b01 and 0b11 are assigned to packets arriving from the upper and lower tier, respectively.

For a *private* circuit, a *single* OpenFlow rule is set to match packets with metadata values of 0b1* by using 0b10 mask. Therefore, only packets from *all* input ports connected to the lower tier are matched and transmitted though the circuit. Similarly, for a *shared* circuit, packets with metadata of 0b*1 are matched by using 0b01 mask. Hence, packets arriving from *all* input ports connected to both upper and lower tiers are matched and transmitted through the circuit.

As described above, the OpenFlow rule is also set to capture the $DSCP_e$ value (nw_{tos}), and the lower tier subnet destination (nw_{dst}) of the switch connected to the other end of the optical circuit.



Private optical circuit configuration:
*table=1,ip,**metadata=0x3/0x2**,nw_tos=252,nw_dst=10.0.0.0/24,*
actions=output:2

Shared optical circuit configuration:
*table=1,ip,**metadata=0x3/0x1**,nw_tos=252,nw_dst=10.0.0.0/24,*
actions=output:2

Figure 6. Open vSwitch example – Private/Shared circuits configuration. A single OpenFlow rule matches packets arriving from all input ports of the switch connected to either lower or upper tier, by using a predefined assignment of packet metadata.

3 Physical Implementation

In this section, we described the use-cases demonstrators, which have been demonstrated in ECOC 2016.

3.1 VDC

The main focus of the VDC demonstrator was to showcase the on-demand provisioning of VDC instances employing the developed orchestrator, which is based on OpenStack, that included the extensions to the Horizon dashboard service and the new developed Algorithms module. This module handles all the interactions with the OpenStack services, the ODL controller for topological and resource availability information from the data plane and the VDC resource mapping onto physical resource, both computing and networking. Additionally, we showcased the integration of the whole software stack, which includes the abovementioned OpenStack-based orchestrator and the ODL controller, enabling the dynamic configuration of optical paths to achieve connectivity between the VMs of the provisioned VDC. The scenario employed to perform the demonstrator is depicted in Figure 7.

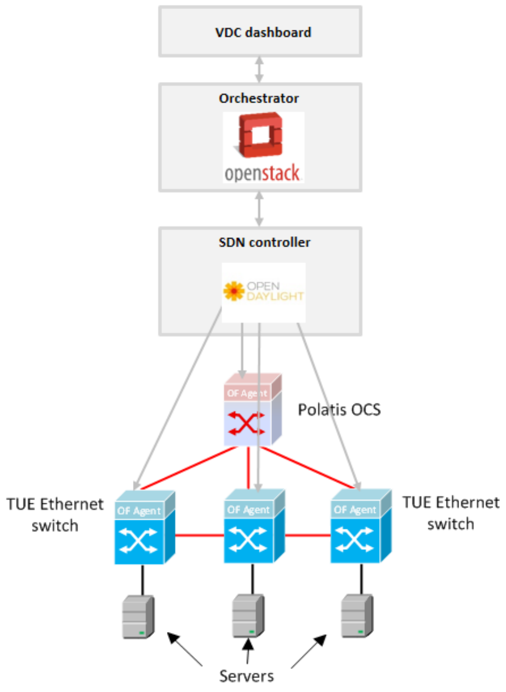


Figure 7. Physical implementation of the ECOC demonstrator for the VDC use case.

Essentially, the physical demonstrator consisted of three servers connected to the optical data plane following the topology depicted in figure 7. In each one of the servers, an OpenStack compute instance was deployed, which enabled the creation of VMs on that particular server following the specifications of the VDC instances. Then, a third server (not shown in the figure) was hosting all the control and orchestration software suites. This server, as well as the optical switches (both TUE ToRs and Polatis switch), where connected by means of a parallel management network to enable the configuration of the optical equipment as desired.

Using this configuration of the demonstrator, we proceeded on showcasing the VDC provisioning. As seen in the Dlux GUI from ODL (Figure 8), the data plane of the demonstrator is properly detected and advertised (the three physical servers, the three TUE switches, the Polatis and the OVS instances of the compute nodes in OpenStack). From this step on, all the involved network entities can be configured to install the desired dataflows in order to achieve end-to-end connectivity between VMs of the VDC.

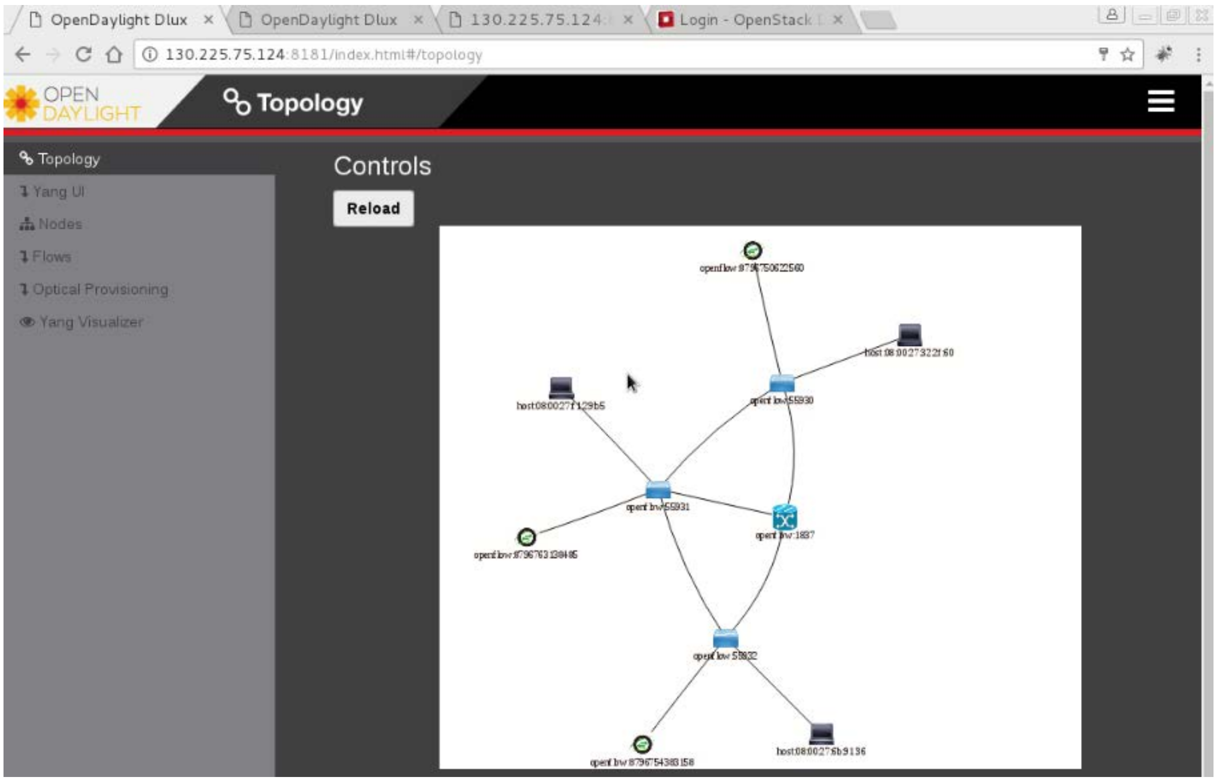
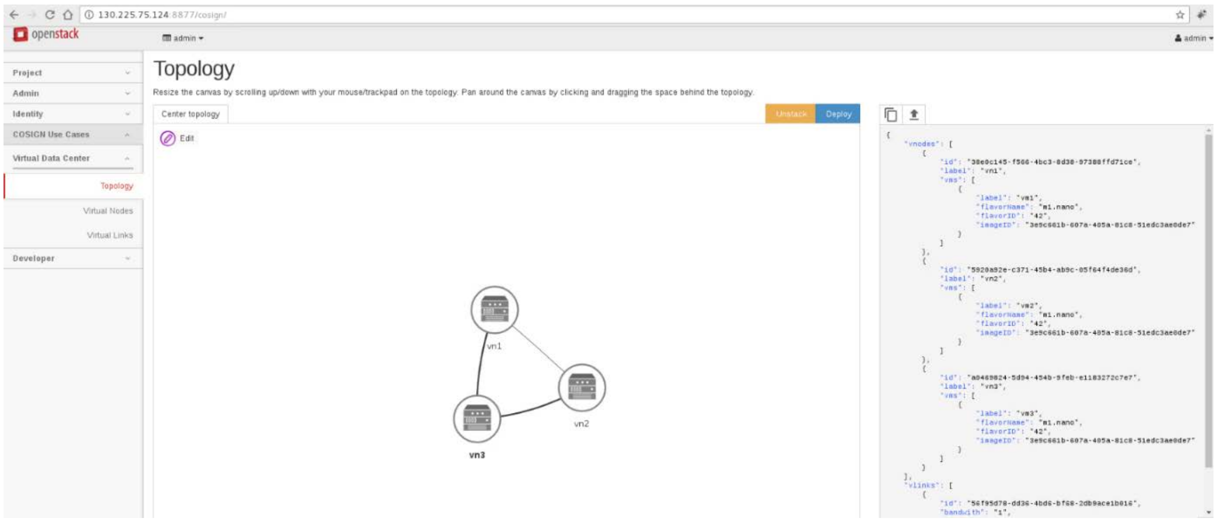


Figure 8. Dlux GUI in ODL – Physical infrastructure topology.

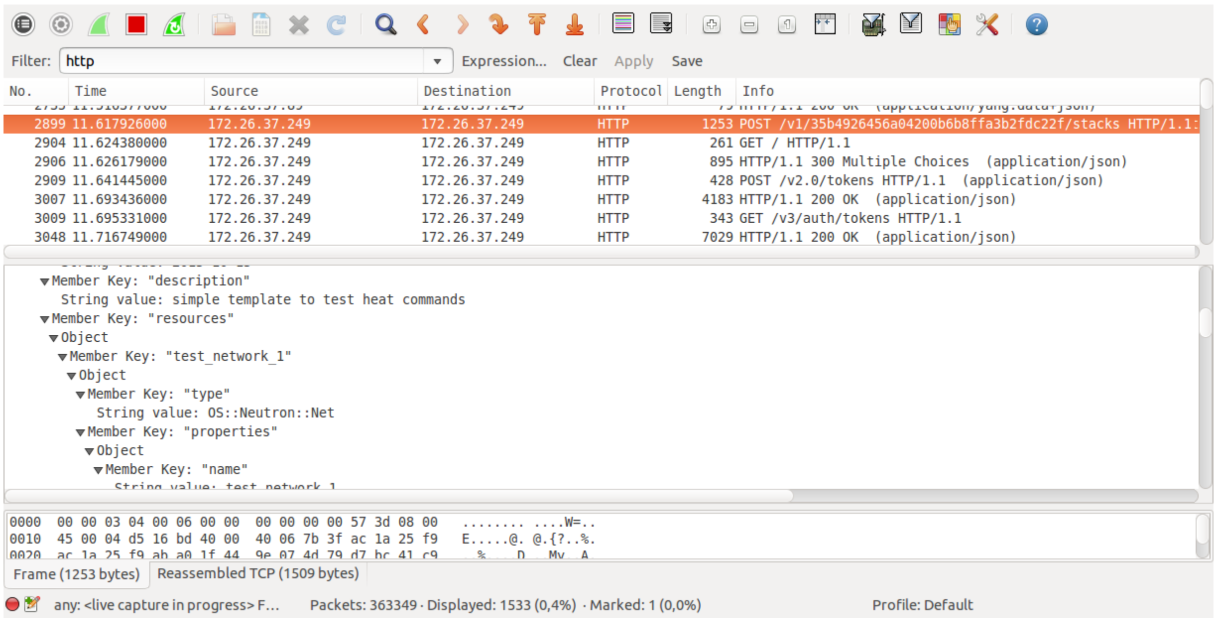
The first step to provision a VDC starts at the graphical interfaces of OpenStack, the Horizon dashboard service, extended to support the graphical creation of VDC instances. Figure 9 exemplifies a particular VDC creation, consisting on three virtual nodes, with one VM each, interconnected in a full mesh fashion through virtual links.


130.225.75.124:8877/cosign/
openstack
admin
admin
Project
Admin
Identity
COSIGN Use Cases
Virtual Data Center
Topology
Virtual Nodes
Virtual Links
Developer
Topology
Resize the canvas by scrolling up/down with your mouse/trackpad on the topology. Pan around the canvas by clicking and dragging the space behind the topology.
Center topology
Unstack
Deploy
Edit
vn1
vn2
vn3

```
{
    "vnodes": [
        {
            "id": "38e0c145-f566-4bc3-8d38-97388ffd71ce",
            "label": "vn1",
            "vms": [
                {
                    "label": "vm1",
                    "flavorName": "m1.nano",
                    "flavorID": "42",
                    "imageID": "3e9c661b-607a-405a-81c8-51edc3ae0de7"
                }
            ]
        },
        {
            "id": "5920a92e-c371-45b4-ab9c-05f64f4de36d",
            "label": "vn2",
            "vms": [
                {
                    "label": "vm2",
                    "flavorName": "m1.nano",
                    "flavorID": "42",
                    "imageID": "3e9c661b-607a-405a-81c8-51edc3ae0de7"
                }
            ]
        },
        {
            "id": "a0469824-5d94-454b-9feb-e1183272c7e7",
            "label": "vn3",
            "vms": [
                {
                    "label": "vm3",
                    "flavorName": "m1.nano",
                    "flavorID": "42",
                    "imageID": "3e9c661b-607a-405a-81c8-51edc3ae0de7"
                }
            ]
        }
    ],
    "vlinks": [
        {
            "id": "56f95d78-dd36-4bd6-bf68-2db9ace1b016",
            "bandwith": "1",
```

Figure 9. Graphical creation of a VDC through the extended Horizon dashboard service.

Once specified, the VDC is submitted for deployment through the blue deploy button at the dashboard. After confirmation of the operation, a JSON file is sent (the exact one seen at the right column of the above figure) to the Algorithms module, which listens for create, read, update or delete (CRUD) operations regarding VDC instances. In the case of a VDC creation, the Algorithms module decodes the details in terms of requested resources by the VDC from the received JSON file. Then, it collects information about computing and networking resources availability by contacting Nova and ODL, respectively. With this information, and the details of the VDC, a mapping algorithm is executed, which decides the placement of the VMs of the virtual nodes and the optical paths for the virtual links. If a valid mapping solution is found, a Heat YAML template is produced, which contains the full description of all OpenStack resources needed to be deployed (see Figure 10). This template is then sent to the Heat service in order to start the stack deployment. Note that the template also includes the compute nodes where the VMs will be deployed. In the particular example, the three VMs will be deployed to different compute nodes, that is, each compute node will host exactly one VM, since they belong to different virtual nodes of the VDC. Moreover, the details of the calculated optical paths to interconnect each pair of VMs in the VDC not deployed in the same compute node, which correspond to sequences of physical links in the ODL topology, are persistently stored at a database created for its purpose, since they will be needed when configuring the optical connectivity at the data plane.



```
No.    Time          Source          Destination     Protocol Length  Info
  2899 11.617926000  172.26.37.249   172.26.37.249   HTTP       1253 POST /v1/35b4926456a04200b6b8ffa3b2fdc22f/stacks HTTP/1.1:
  2904 11.624380000  172.26.37.249   172.26.37.249   HTTP        261 GET / HTTP/1.1
  2906 11.626179000  172.26.37.249   172.26.37.249   HTTP        895 HTTP/1.1 300 Multiple Choices  (application/json)
  2909 11.641445000  172.26.37.249   172.26.37.249   HTTP        428 POST /v2.0/tokens HTTP/1.1  (application/json)
  3007 11.693436000  172.26.37.249   172.26.37.249   HTTP       4183 HTTP/1.1 200 OK  (application/json)
  3009 11.695331000  172.26.37.249   172.26.37.249   HTTP        343 GET /v3/auth/tokens HTTP/1.1
  3048 11.716749000  172.26.37.249   172.26.37.249   HTTP       7029 HTTP/1.1 200 OK  (application/json)
```

```
    ▾Member Key: "description"
       String value: simple template to test heat commands
    ▾Member Key: "resources"
     ▾Object
      ▾Member Key: "test_network_1"
       ▾Object
        ▾Member Key: "type"
           String value: OS::Neutron::Net
        ▾Member Key: "properties"
         ▾Object
          ▾Member Key: "name"
```

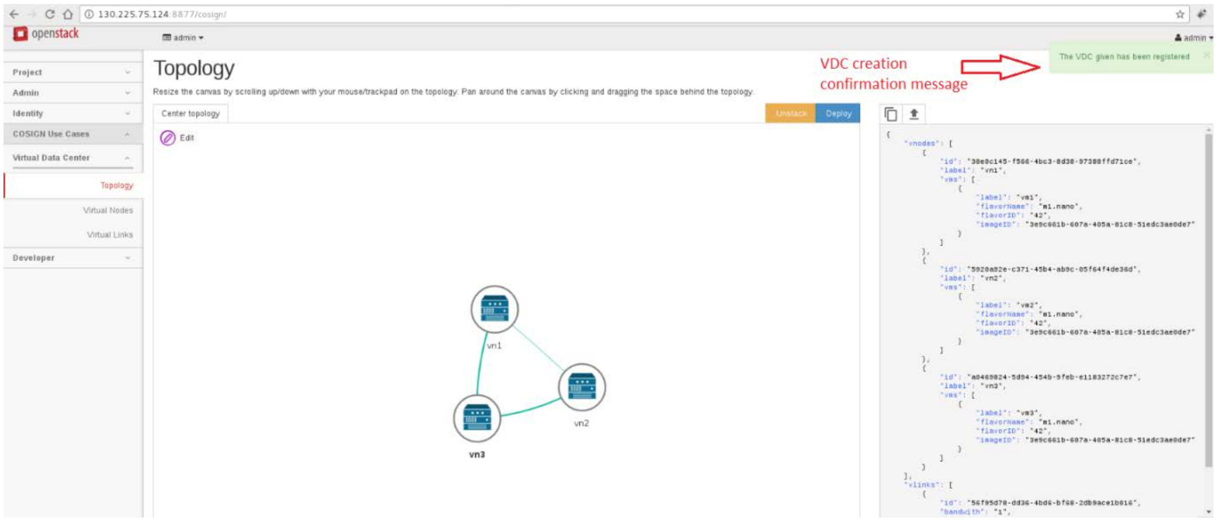
```
0000  00 00 03 04 00 06 00 00  00 00 00 00 57 3d 08 00   ........ ....W=..
0010  45 00 04 d5 16 bd 40 00  40 06 7b 3f ac 1a 25 f9   E.....@. @.{?..%.
```

Frame (1253 bytes) | Reassembled TCP (1509 bytes)

any: <live capture in progress> F... Packets: 363349 · Displayed: 1533 (0,4%) · Marked: 1 (0,0%) Profile: Default

Figure 10. Wirseshark capture containing the YAML template with the OpenStack resources of the VDC instance.

Once Heat confirms the correct creation of the stack specified in the YAML template, the Algorithms module informs back to the Horizon dashboard service and the VDC is registered (see Figure 11). The proper creation of the full stack is assessed and the topological representation of the created stack can be viewed in the orchestration tab of Horizon (Figure 12), were we have highlighted the element corresponding to a Nova server (i.e. a VM) and the created Neutron network, which consist of an IP network and subnetwork with three attached Neutron ports, one per Nova server, which are the virtual network interfaces of the VMs.



Topology

Resize the canvas by scrolling up/down with your mouse/trackpad on the topology. Pan around the canvas by clicking and dragging the space behind the topology.

The VDC given has been registered

```
{
    "vnodes": [
        {
            "id": "38e0c145-f566-4bc3-8d38-97388ffd71ce",
            "label": "vn1",
            "vms": [
                {
                    "label": "vm1",
                    "flavorName": "m1.nano",
                    "flavorID": "42",
                    "imageID": "3e9c661b-607a-405a-81c8-51edc3ae0de7"
                }
            ]
        },
        {
            "id": "5920a92e-c371-45b4-ab9c-05f64f4de36d",
            "label": "vn2",
            "vms": [
                {
                    "label": "vm2",
                    "flavorName": "m1.nano",
                    "flavorID": "42",
                    "imageID": "3e9c661b-607a-405a-81c8-51edc3ae0de7"
                }
            ]
        },
        {
            "id": "a0469824-5d94-454b-9feb-e1183272c7e7",
            "label": "vn3",
            "vms": [
                {
                    "label": "vm3",
                    "flavorName": "m1.nano",
                    "flavorID": "42",
                    "imageID": "3e9c661b-607a-405a-81c8-51edc3ae0de7"
                }
            ]
        }
    ],
    "vlinks": [
        {
            "id": "56f95d78-dd36-4bd6-bf68-2db9ace1b016",
            "bandwith": "1",
```

Figure 11. Confirmation of VDC creation.

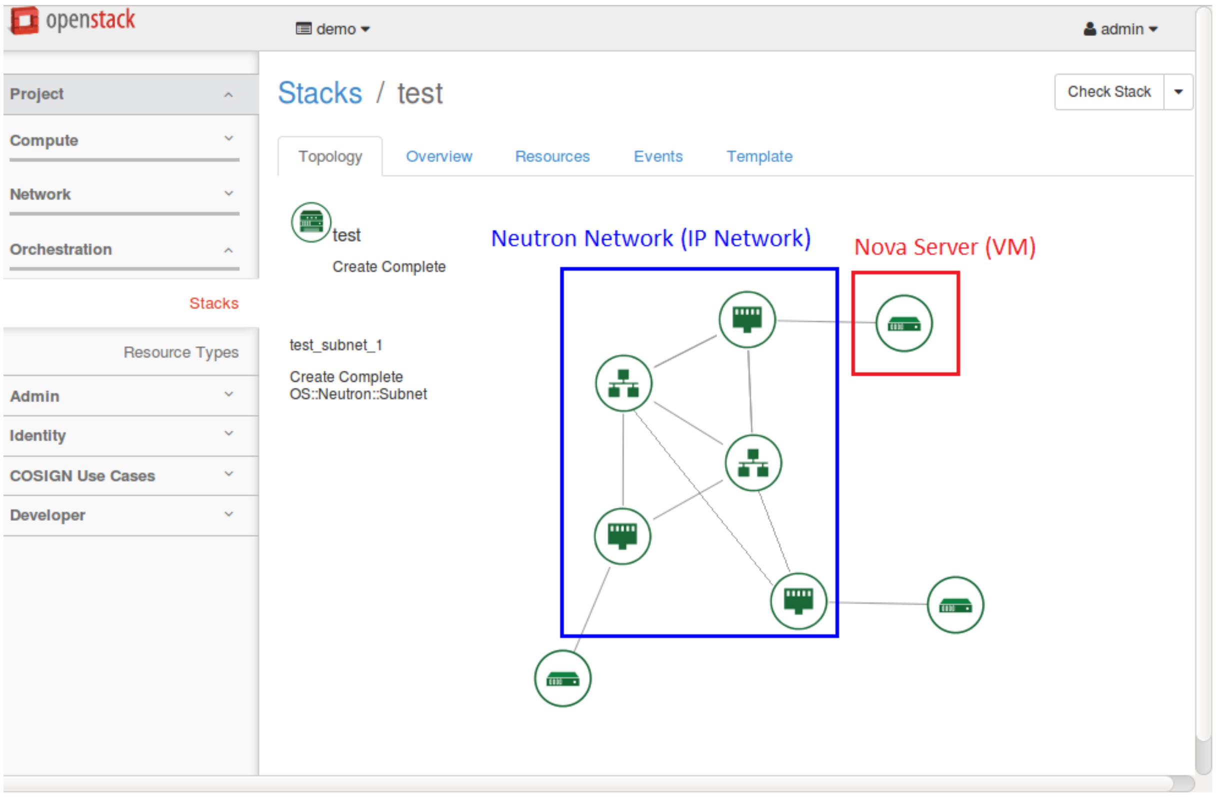
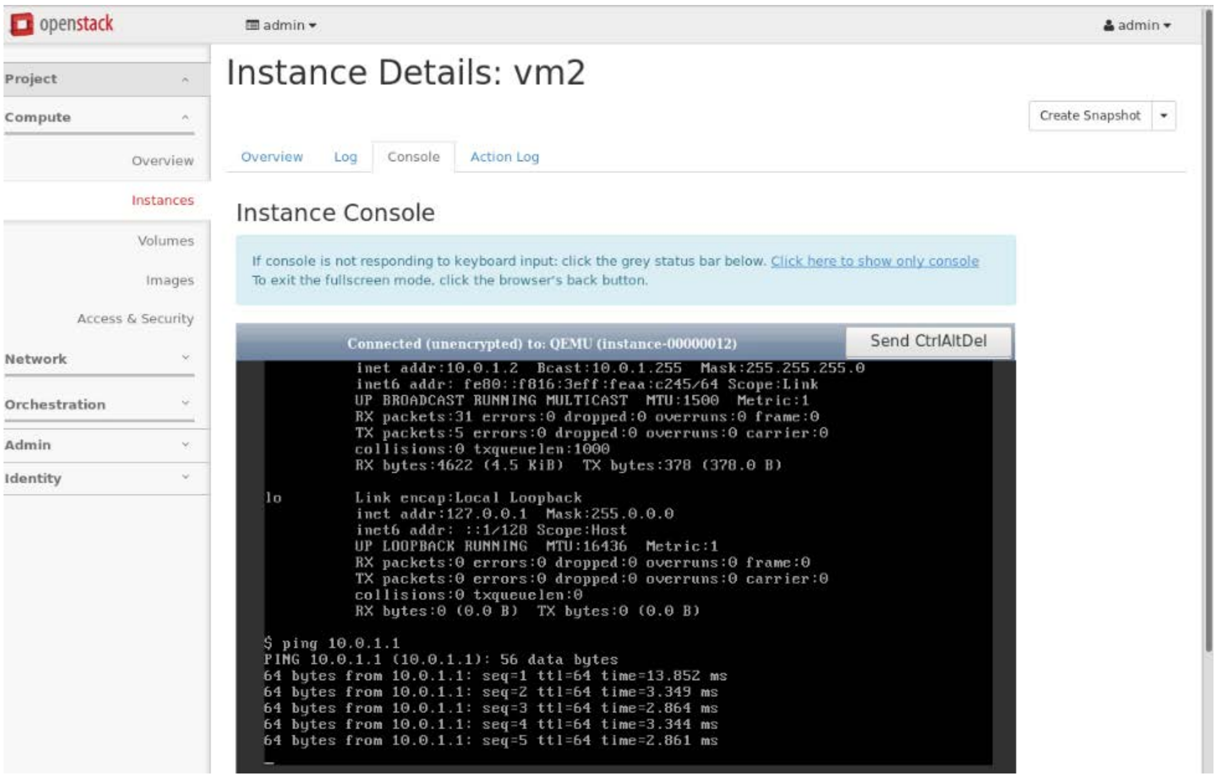


Figure 12. VDC creation – Created stack.

The final step to assess a proper deployment of the VDC and to insure that network connectivity, at the data plane, has been properly established was to perform some connectivity tests between virtual machines to insure that data transfers between the computing resources of the VDC instances can be achieved. To this end we performed simple ping operations. Through the Horizon dashboard, it is possible to access to the VMs consoles, log in to them and operate them as if the tenant had direct access to the deployed VM instances. Once logged inside the VMs, we performed the corresponding pings, establishing that the end-points can communicate with each other. Figure 13 depicts the ping traces for a particular pair of VMs in the deployed VDC of the example. These traces indicate that proper installation of the flows at the TUE switches (Figure 14) as well as the creation of a cross-connection at the Polatis switch in case that the deployed VDC employs optical paths that include this switch (Figure 15). With this, we demonstrated the on-demand and automatic creation of VDC instances, with no further manual interaction from the tenant than the graphical specification of the desired VDC instance. From that point, the VDC is ready to be operated as the tenant sees fit.



If console is not responding to keyboard input: click the grey status bar below. Click here to show only console
To exit the fullscreen mode, click the browser's back button.

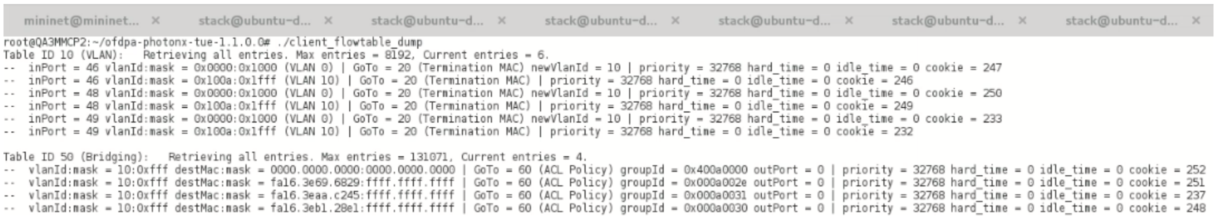
Connected (unencrypted) to: QEMU (instance-00000012)

```
             inet addr:10.0.1.2  Bcast:10.0.1.255  Mask:255.255.255.0
             inet6 addr: fe80::f816:3eff:feaa:c245/64 Scope:Link
             UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
             RX packets:31 errors:0 dropped:0 overruns:0 frame:0
             TX packets:5 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:1000
             RX bytes:4622 (4.5 KiB)  TX bytes:378 (378.0 B)

lo           Link encap:Local Loopback
             inet addr:127.0.0.1  Mask:255.0.0.0
             inet6 addr: ::1/128 Scope:Host
             UP LOOPBACK RUNNING  MTU:16436  Metric:1
             RX packets:0 errors:0 dropped:0 overruns:0 frame:0
             TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:0
             RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

$ ping 10.0.1.1
PING 10.0.1.1 (10.0.1.1): 56 data bytes
64 bytes from 10.0.1.1: seq=1 ttl=64 time=13.852 ms
64 bytes from 10.0.1.1: seq=2 ttl=64 time=3.349 ms
64 bytes from 10.0.1.1: seq=3 ttl=64 time=2.864 ms
64 bytes from 10.0.1.1: seq=4 ttl=64 time=3.344 ms
64 bytes from 10.0.1.1: seq=5 ttl=64 time=2.861 ms
_
```

Figure 13. Ping test between VMs.



```
root@QA3MMCP2:~/ofdpa-photonx-tue-1.1.0.0# ./client_flowtable_dump
Table ID 10 (VLAN):   Retrieving all entries. Max entries = 8192, Current entries = 6.
--  inPort = 46 vlanId:mask = 0x0000:0x1000 (VLAN 0) | GoTo = 20 (Termination MAC) newVlanId = 10 | priority = 32768 hard_time = 0 idle_time = 0 cookie = 247
--  inPort = 46 vlanId:mask = 0x100a:0x1fff (VLAN 10) | GoTo = 20 (Termination MAC) | priority = 32768 hard_time = 0 idle_time = 0 cookie = 246
--  inPort = 48 vlanId:mask = 0x0000:0x1000 (VLAN 0) | GoTo = 20 (Termination MAC) newVlanId = 10 | priority = 32768 hard_time = 0 idle_time = 0 cookie = 250
--  inPort = 48 vlanId:mask = 0x100a:0x1fff (VLAN 10) | GoTo = 20 (Termination MAC) | priority = 32768 hard_time = 0 idle_time = 0 cookie = 249
--  inPort = 49 vlanId:mask = 0x0000:0x1000 (VLAN 0) | GoTo = 20 (Termination MAC) newVlanId = 10 | priority = 32768 hard_time = 0 idle_time = 0 cookie = 233
--  inPort = 49 vlanId:mask = 0x100a:0x1fff (VLAN 10) | GoTo = 20 (Termination MAC) | priority = 32768 hard_time = 0 idle_time = 0 cookie = 232

Table ID 50 (Bridging):   Retrieving all entries. Max entries = 131071, Current entries = 4.
--  vlanId:mask = 10:0xfff destMac:mask = 0000.0000.0000:0000.0000.0000 | GoTo = 60 (ACL Policy) groupId = 0x400a0000 outPort = 0 | priority = 32768 hard_time = 0 idle_time = 0 cookie = 252
--  vlanId:mask = 10:0xfff destMac:mask = fa16.3e69.6829:ffff.ffff.ffff | GoTo = 60 (ACL Policy) groupId = 0x000a002e outPort = 0 | priority = 32768 hard_time = 0 idle_time = 0 cookie = 251
--  vlanId:mask = 10:0xfff destMac:mask = fa16.3eaa.c245:ffff.ffff.ffff | GoTo = 60 (ACL Policy) groupId = 0x000a0031 outPort = 0 | priority = 32768 hard_time = 0 idle_time = 0 cookie = 237
--  vlanId:mask = 10:0xfff destMac:mask = fa16.3eb1.28e1:ffff.ffff.ffff | GoTo = 60 (ACL Policy) groupId = 0x000a0030 outPort = 0 | priority = 32768 hard_time = 0 idle_time = 0 cookie = 248
```

Figure 14. Installed flows at a TUE switch.

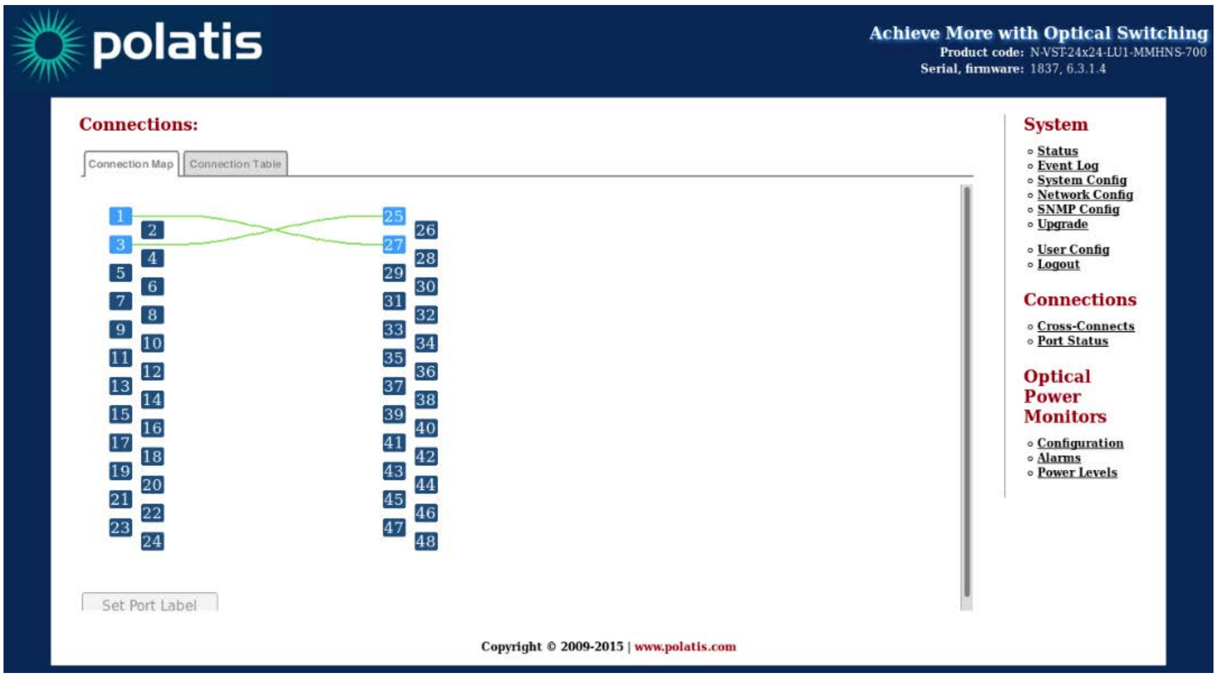


Figure 15. Created cross-connection at the Polatis switch.

We also demonstrated the on-demand and automatic deletion of already deployed VDC instances. This can be performed by selecting the orange Unstack button at the VDC tab of the Horizon dashboard, which triggers a delete operation towards the algorithms module. At its turn, this triggers a delete of the deployed stack, which is sent to Heat. Through the coordination of Nova and Neutron, Heat destroys all the deployed resources, including VMs, IP network and so on. As for the optical connectivity between the end-points (i.e. the VMs), once the ODL controller detects that the end-points have been erased, it triggers the deletion of the dataflows, eliminating the installed flows in the OVS nodes, the TUE switches and the cross-connection at the Polatis switch. For the sake of space, we will not depict all the steps and operation involved in the overall elimination of a VDC instance.

All in all, the ECOC demonstrator showed the overall interaction of the orchestrator with all the other layers, achieving the creation/deletion of VDC instances according to the tenants' specifications and needs.

3.2 vApp - ECOC Demo

Figure 16 depicts the HW setup for vApp demo, which includes: 3 servers (IBM x3690 X5), 3 TU/e ToR switches, Polatis switch, Xena Ethernet traffic generator/tester – all connected by 10G optical links; and a controller with a separate control network.

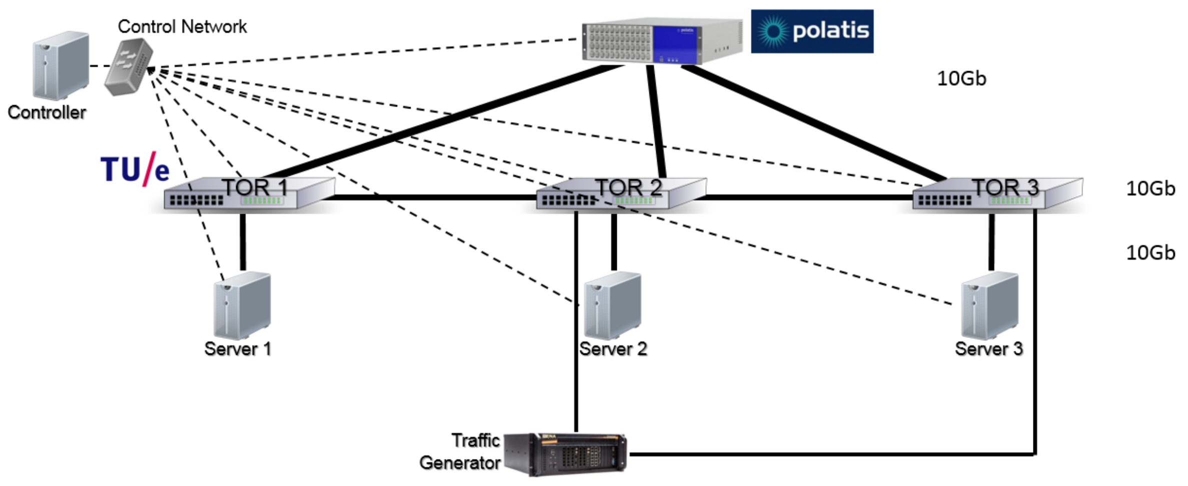


Figure 16 - ECOC demo HW setup

vApp demo demonstrates the benefits of the shared optical circuit as compared to private circuit. The demonstrated scenario includes:

- Two elephant flows between server 1 to server 3, and server 2 to server 3.
- Optical circuit between TOR 2 and TOR 3.
- Mice flows between TOR 2 and TOR 3 through the direct packet plane link, by the traffic generator.

The elephant flows are sampled by sFlow, detected, and DSCP tagged by the virtual observer employed on each server.

Private Circuit:

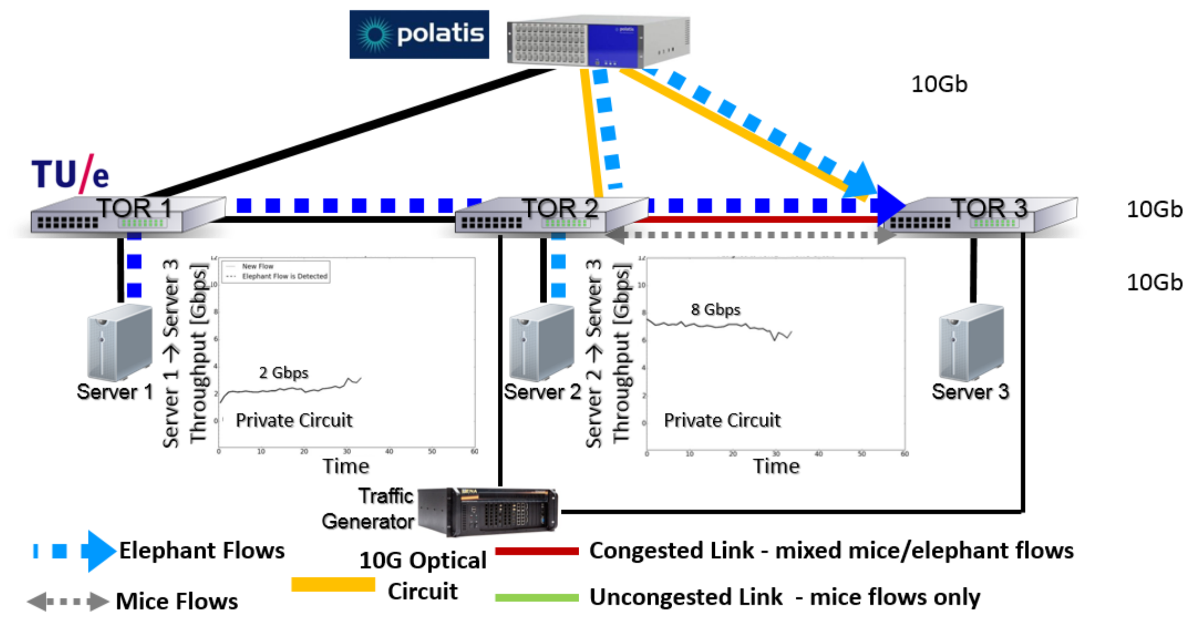


Figure 17 – Private circuit scenario

Figure 17 presents the private optical circuit scenario, in which the elephant flow between server 2 and 3 is transmitted through the established optical circuit with high throughput (~8Gbps). Whereas the elephant flows between server1 and 3 is transmitted through the packet plane, along with the mice flows over the congested link between TOR 2 and TOR 3; therefore, its throughput is much lower (~2 Gbps).

Shared Circuit:

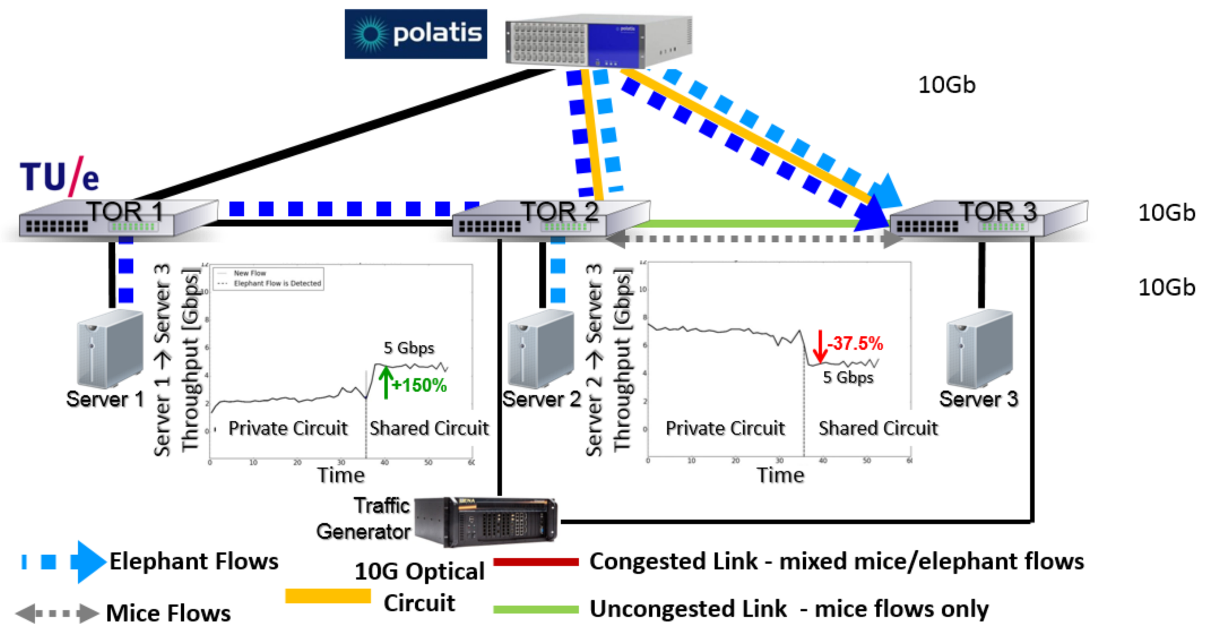


Figure 18 – Shared circuit scenario

Figure 18 presents the shared optical circuit scenario. Both elephant flows are transmitted through the established optical circuit, Therefore, the elephant flows fairly share the optical circuit bandwidth. In this scenario, the throughput of the elephant flow between server 1 and 3 is increased by 150%, whereas the throughput of the elephant flow between server 2 and 3 is decreased by 37.5%. Furthermore, the optical circuit is fully utilized, and improved by 20% as compared to private circuit.

3.3 vApp – Improved Demo

In ECOC demo, we presented how the shared optical circuit can offer fairness in terms of elephant flows throughput. However, due to the limited 10Gb optical circuit. We were unable to demonstrate the full potential of vApp. In the improved demo at TU/e we use the same topology but with 40Gb optical links between the TORs and the Polatis switch. Furthermore, we also measured the RTT of the mice flows over the packet network plane.

Throughput Comparison:

Figure 19 presents the elephant flows throughput for both private and shared optical circuit (which are alternately altering). The throughput of the elephant flow between server 2 and 3 isn't affected by optical circuit type. On the other hand, the throughput of the elephant flow between server 1 and 3 is greatly improved by the shared optical circuit (by 400%). This improvement is accomplished due to higher link bandwidth of the optical plane.

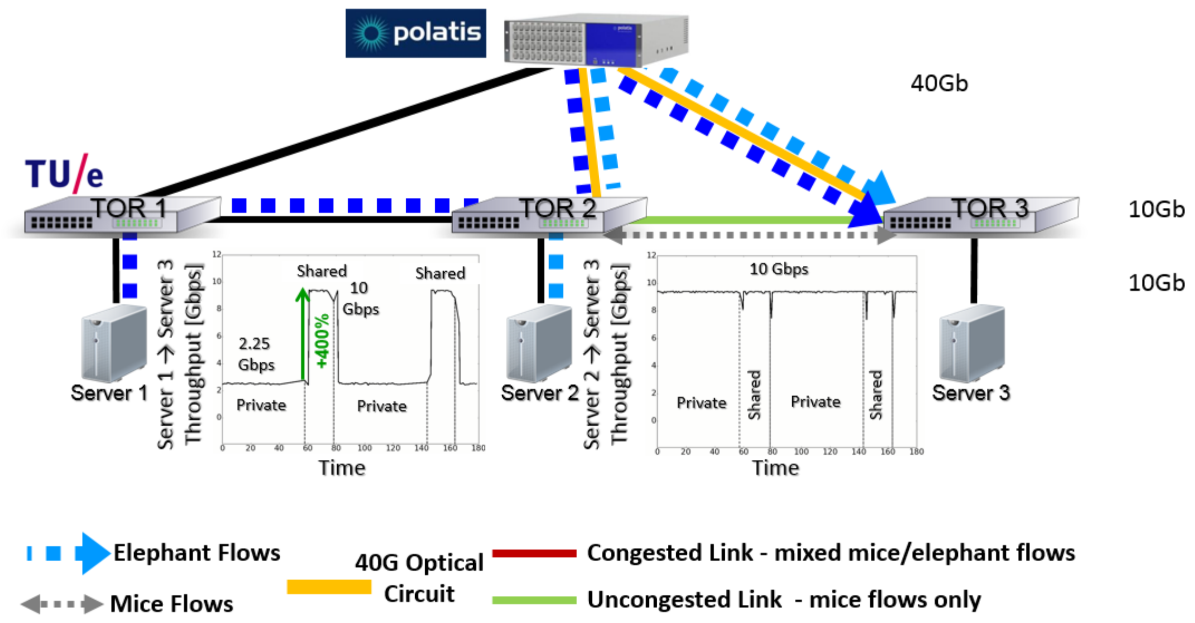


Figure 19 – Throughput comparison between private and shared optical circuits

RTT Comparison:

Figure 20 presents the mice flows RTT through the packet network plane, for both private and shared optical circuit (which are alternately altering). The RTT of both mice flows is decreased by approximately 50% when using the shared circuit. Furthermore, the RTT of the mice flows between server 2 and 3 fluctuates more, since that it interacts with two TCP sessions (elephant flow between server 1 and 3, and between server 2 and 3); whereas, the mice flow between server 1 and 3 interacts only with single TCP session.

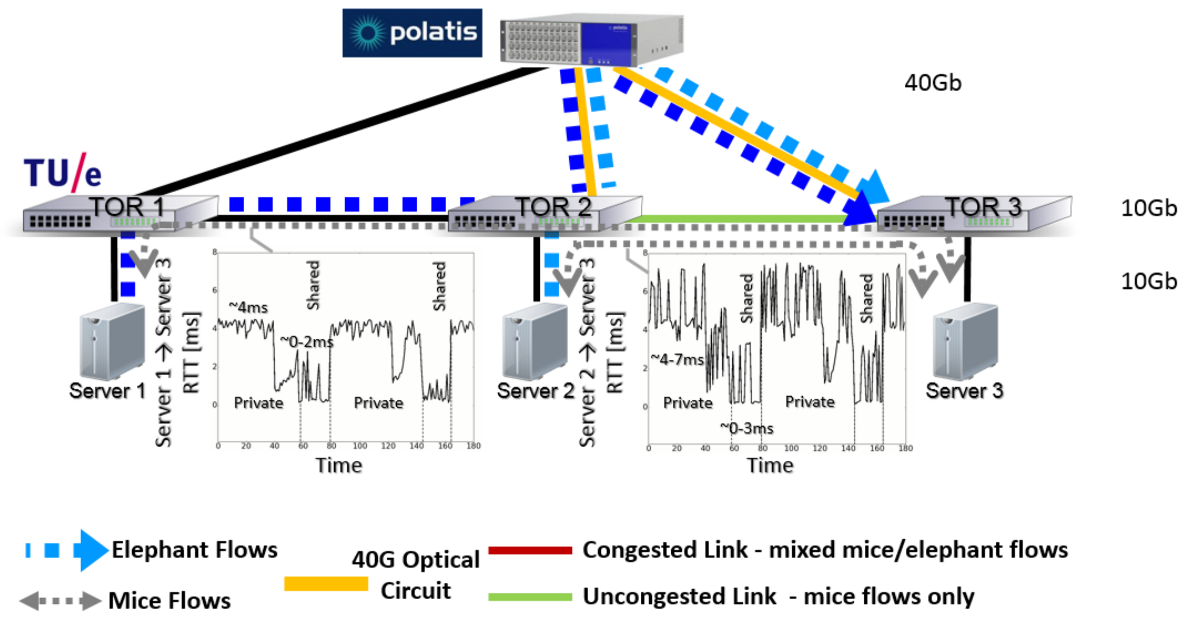


Figure 20 - RTT comparison between private and shared optical circuits

4 Potential Orchestrator Extensions

4.1 VDC

4.1.1 New Algorithms for Long-Term Demo Technologies

The newly introduced Algorithms module is the responsible for abstracting the requirements of an incoming VDC request onto actual infrastructure needs, both in terms of computing resources and networking. Then, with this information and status information collected from the Nova service and the ODL controller, it calculates the optimal mapping of the VDC instance on top of the physical resource. To this purpose, specialized algorithms taking into account the particularities of the COSIGN data plane (namely the presence of the TUE ToR switches and the Polatis fibre switches) have been designed and implemented.

As an evolution towards all-optical DCN architectures, COSIGN is focusing on the development of novel optical devices (i.e., programmable optical NIC, fast switch) to enable the migration of the data plane towards this long-term scenario where communication between servers will be done purely in the optical domain. One of the key aspects of this long term scenario is the utilization of Time Division Multiplexing (TDM)-based transmission to realize the network connectivity between servers. Hence, connection requests will have to be allocated in the form of a set of time slots which are tailored to the bandwidth needs of the said connections. Because the Algorithms module purpose is to optimize the mapping of the VDC instances considering the peculiarities of the DC data plane, in order to support the new bandwidth allocation expected for the long-term scenario, new mapping algorithms will have to be designed. The purpose of these algorithms will be to determine the necessary time slots to satisfy the bandwidth requirements of the virtual links requested in the VDC instance while, at the same time, guarantee that no time slot collision happens for the connections associated to different virtual links. To achieve this purpose, besides the new algorithms, it will be necessary to develop new interfaces with the CP in order to dynamically poll the status of the available time slots at the different network hardware elements. Additionally, in order to enforce the optimal path configuration according to the decided VDC mapping, it will be necessary to extend the already implemented interface to pass down the details of the path of the connections to the ODL controller to also include the details of the chosen time slots to be allocated for the connections.

With these modifications, the COSIGN orchestrator will be capable to support the dynamic and automated VDC provisioning considering a TDM-based all-optical data plane in the same way that has been developed for the medium term data plane scenario.

4.1.2 Extensions on the VDC Dashboard for Enhanced Administrative Information

As explained in previous sections, the developed COSIGN orchestrator allows for the graphical definition of VDC instances, harnessing the capabilities of the Horizon dashboard service. In this regard, an interesting feature that Horizon provides is the possibility to know in which server a VM has been instantiated as well as the current occupation of the different compute nodes, thus being a valuable feature for DC administrators, since it allows to monitor the utilization of the computing resources and tracking potential misbehaviours of the VM mapping process which could indicate some malfunctioning of the hardware or the software involved in the provisioning. Nevertheless, Horizon does not provide an equivalent feature regarding the network resources, thus hindering the monitoring of the DCN fabric.

To this purpose, a potential extension of the VDC dashboard service is the possibility to track the utilization of the network resources and the mapping of the multiple virtual links of the different VDC instances, so the DC administrator can have a global view of the network utilization in regards of VDC mapping. For this, it would be necessary that, after sending a VDC request to the Algorithms module, the details regarding the virtual link mapping are reported back to the Horizon dashboard, so they can be collected and displayed properly. Thus, modifications to the implemented interface for the communication between the dashboard and the Algorithms module to include the details of the VDC mapping on the responses would be necessary. Then, this information should be displayed in a new section available exclusively to the DC administrator, since it is sensitive information that should not be publicly disclosed to all tenants having access to the dashboard service. The details of the virtual link mapping could be collected in the form of a table where the identifiers of the virtual link, the VDC they belong to and the tenant owning the VDC are displayed, so the utilization of the optical resources can be tracked down for administrative purposes. Conversely, it could be possible to depict this information in a graphical way, displaying the topology of the DC and how the different physical network resources are being employed by the virtual links. Note, however, that such approach would require extensive modifications to both the Algorithms module and the Horizon dashboard in order to allow the exchange of information associated to the topological and status information of the DCN. Nevertheless, it could be though as an additional enhancement to allow a more attractive and intuitive administration interface.

With this new information, the DC administrator could monitor the utilization of the optical network resources and control the health of the overall VDC provisioning process, including both the computing and network resources.

4.2 vApp

4.2.1 Private / Shared-Aware Optical Circuits Scheduler

Recently, scheduling techniques optimized for hybrid networks, e.g., [35], [22] have been proposed. Unfortunately, these techniques cannot be directly applied for vApp. In hybrid architectures, the scheduler can first decide about the demand that should be served by the OCS, where the rest of the traffic is served by the EPS in an *independent* manner. In vApp, on the other hand, elephant flows can be routed through both planes, which creates a strong dependency between them. A scheduler that takes into account the new ability of circuits sharing, and consider this dependency may lead to even better network performance.

4.2.2 Advanced Optical Circuit Sharing

We presented *shared* circuits only for *last hop routing*. Namely, the sharing is employed for elephant flows delivered to one of the circuit's endpoints. By advanced configuration, we can enable circuit sharing with elephant flows at any hop along their routes. Furthermore, the orchestrator can also take into account the number of saved electrical hops by establishing a given optical circuit, rather than only considering the traffic demand, as commonly used.

4.2.3 Upper-Tier Topology Design

vApp is evaluated for Ring and FBFly upper tiers, which are commonly used by industry. Other topologies might offer better network throughput by exploiting the circuit sharing. For instance, hypercube network topologies are basically a multidimensional mesh network with two nodes in each dimension. Due to similarity, such topologies are usually grouped into a k-ary d-dimensional mesh topology family where d represents the number of dimensions and k represents the number of nodes in each dimension. Therefore, such a topology can be designed to also meet the FBFly properties.

5 Conclusions

This deliverable summarizes WP4 activity, and lists the benefits and potential extensions for both COSIGN use-cases, namely VDC and vApp. To this end, we have reviewed the main developments made for the COSIGN orchestrator for both use cases, namely, the Virtual Data Centre (VDC) and the Virtualized Application (vApp) use cases. Next, we highlighted how the development and implementation of the COSIGN orchestrator brings benefits to the application/service layer as well as to the control and data planes of data centre infrastructures, enriching the overall service provisioning and infrastructure management in future data centre architectures. Then, we presented a physical implementation which was demonstrated in ECOC 2016. Finally, we pointed out several potential extensions.

Focusing on the VDC use case, the inclusion of the COSIGN orchestrator has allowed for the flexible and dynamic adoption of CRUD methods in regards of complex Infrastructure as a Service (IaaS) provisioning which encompass both networking and computing resources. The presence of a newly designed algorithms module allows for a better exploitation of the underlying physical resources, which, through the tight collaboration with the control plane technologies, coordinates the placement of network and cloud DC resources to provide them to upper service and application layers to be exploited by tenants.

Focusing on the vApp use case, its architecture and orchestrator have allowed for presenting a new concept of optical circuit sharing in hybrid DCN. Such circuit sharing results in better optical circuits utilization, which increases the elephant flows' throughput in the network. Also, it results in better mice/elephant flow separation, which mitigates the filled network buffers by elephant flows over the electrical packet plane. Therefore, the mice flows experience much lower RTT and lower packet losses, since the buffers are not as full as compared to the case when more elephant flows are transmitted through the electrical packet plane. Finally, vApp orchestrator allows for efficient and scalable OpenFlow technique for optical circuit sharing.

All in all, the COSIGN orchestrator brings a rich and powerful tool that eases the provisioning of services to third party entities. Moreover, the COSIGN orchestrator opens up many opportunities for exploitation by the academic and research partners to leverage the lessons learnt in future research initiatives/projects while industrial partners could incorporate the new concepts developed to the next generation of their cloud service portfolio.

6 Appendix

Software module	**Category**	**Software baseline**	**License**	**URL**	**Partner**
vApp - Physical Observer	Orchestration	sFlow tool	sFlow tool license: http://www.inmon.com/ technology/sflowlicense.txt	stash.i2cat.net/scm/cosign/network_observers.git	IBM
vApp - Virtual Observer	Orchestration	sFlow tool	sFlow tool license: http://www.inmon.com/ technology/sflowlicense.txt	stash.i2cat.net/scm/cosign_d33/ibm_overlay_network_component_d33.git	IBM
vApp - Orchestration algorithms	Orchestration	From Scratch		stash.i2cat.net/scm/cosign/network_observers.git	IBM & DTU
VDC Algorithms Module	Orchestration	Google Gson library, Java Secure Channel (SCh) library, MySQL and IBM ILOG CPLEX Optimization Studio	Apache License v2, Berkeley Software Distribution (BSD) License , General Public License (GPL) and CPLEX v12.5 license: http://www-03.ibm.com/software/sla/sladb.nsf/lilookup/B4B61E90A8AE3DB785257D900042D5FB?OpenDocument	stash.i2cat.net/scm/cosign/algorithms_vdc_dev.git	UPC
OpenStack Neutron extensions for VDC	Cloud Platform Management	OpenStack	Apache 2.0	http://git.i2cat.net/projects/COSIGN/repos/neutron_vdc_dev/browse	NXW
OpenStack Heat extensions for VDC	Cloud Platform Management	OpenStack	Apache 2.0	http://git.i2cat.net/projects/COSIGN/repos/heat_vdc_dev/browse	NXW
OpenStack Horizon extensions for VDC	Cloud Platform Management	OpenStack	Apache 2.0 and MIT	http://stash.i2cat.net/projects/COSIGN/repos/horizon_vdc/browse	i2CAT
OpenStack Horizon extensios for vApp	Cloud Platform Management	OpenStack	Apache 2.0 and MIT	http://stash.i2cat.net/projects/COSIGN/repos/horizon_vapp/browse	i2CAT