An Emulation Environment for SDN Enabled Flexible IP/Optical Networks

Aristotelis Kretsis, Loris Corazza, Kostas Christodoulopoulos, Panagiotis Kokkinos, and Emmanouel (Manos) Varvarigos

Department of Computer Engineering and Informatics, University of Patras, Greece

Department of Electrical and Computer Engineering, National Technical University of Athens, Greece

Tel: (+30) 210 7724731, e-mail: vmanos@central.ntua.gr

ABSTRACT

Elastic optical networks in conjunction with the Software Defined Networking (SDN) paradigm promise to serve the increasing networking requirements of current and future applications and services, providing on demand and scheduled capacity. We present an emulation environment for SDN-enabled, multi-layer and flexible IP/Optical networks, named Julius, based on Mininet, for the SDN network emulation and LINC-OE, for the optical emulation. Julius also integrates a Path Computation Element (PCE) client that communicates with any standard PCE. Julius can be used for basic and applied research on protocols (e.g., OpenFlow, PCEP) extensions and resource reservation/routing algorithms.

Keywords: elastic optical networks, multi-layer, software defined networking, PCE, emulation.

1. INTRODUCTION

Network optimization and orchestration is a growing need stemming from the dynamicity and the exponential increase of network traffic [1]. Also, the Network as a Service vision is becoming a reality, providing the missing programmability and flexibility in using the network resources, together with significant OPEX, CAPEX and quality benefits. Several evolutions support this vision.

Converging the IP and optical network layers, and taking jointly and centrally the respective decisions, is an excellent way to increase service velocity, flexibility and lower total cost of ownership for network operators [2][3]. For this reason, multi-layer (IP and optical) network planning and operation is considered a key issue for future networks [5], requiring appropriate algorithms and control plane extensions. Software Defined Networking (SDN) [4] transforms network devices to remotely programmable forwarding elements, allowing the separation of the forwarding decisions (control plane) from the devices themselves (data plane). In this way, the forwarding decisions are taken in a central entity, called SDN controller, giving it control for adapting the transfers based on the network and application characteristics. SDN can play an important role in multi-layer network planning and operation, making the optical layer more dynamic and agile, increasing its efficiency and enabling end-to-end service provisioning. In addition, fixed-grid WDM networks are evolving into flexi-grid ("elastic", "adaptive", or "tunable") networks [6][7]. Flexible networks are based on (i) flex-grid technology that enables the slicing of the spectrum according to the needs, as opposed to the rigid granularity of WDM networks, but also on (ii) flexible transponders (flex-rate), also known as bandwidth variable transponders (BVTs), which can tune their transmission parameters, trading off the transmission reach for spectrum and/or rate. Flexible optical networks solve various inefficiency problems of traditional WDM networks, providing a fine granular solution for sub- and super-wavelength capacity.

All these evolutions require an environment for the development and evaluation of protocols extensions (e.g., OpenFlow, PCEP) and optimization algorithms regarding the network resources management. We present such an emulation environment for SDN enabled, multi-layer and flexible IP/Optical networks, named Julius, based on Mininet [9], for the SDN network emulation and LINC-OE [11], for the optical emulation. Julius also integrates with the Mantis' [8] Path Computation Element (PCE) providing optimization logic.

A related software is ONOS [10], an open source SDN operating system for communication services providers that is designed for scalability, high performance and high availability. ONOS was originally released in December 2014 and it is currently supported by most major vendor and operators. One of the most notable use cases for ONOS focuses on a packet-optical SDN implementation for Wide Area Networks. In this use case ROADMs are emulated using LINC-OE [11] software switch. Julius differs from ONOS for its single-role vocation. While software like ONOS aims to be universal tools with limitless support for diverse technologies and uses, Julius is deeply focused on its role as an SDN emulation platform for multilayer IP and optical (fixed and flex-grid) optical networks. A primary purpose of Julius is to create a common benchmarking environment where users compare their algorithms under the same conditions, making in this way their results more usable and creating a connection between them. Its thin design is useful when introducing disruptive changes, minimizing development time to adopt cutting edge technologies.

The reminder of this work is organized as follows. In Section 2 we present Julius architecture. Section 3 describes Julius operation and in Section 4 we conclude the paper.

2. JULIUS ARCHITECTURE

Julius aims to be a complete emulation platform for designing and evaluating the next generation SDN based Layer 3 - Layer 2.5 - Layer 0 networks. We have adopted an architecture that provides efficient computation resources' usage and enables the immediate extension of the overall functionality without breaking the implementation of the existing modules. Julius is organized in three layers: *the access layer, the orchestration layer, the execution layer*. Figure 1 shows Julius architecture and its main components. The access layer handles the interaction with the users through a web-based user interface. The orchestration layer contains Julius and third party components responsible for the creation and management of the emulation environment. Finally, the execution layer contains the algorithmic logic for performing the network resource management decisions.

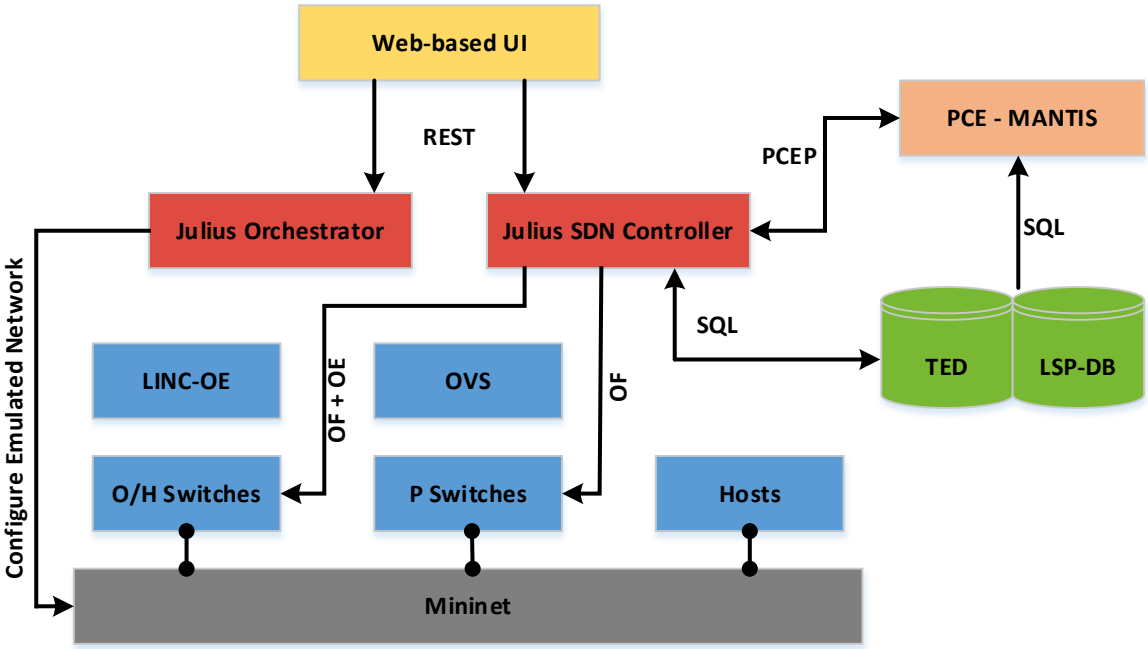

Figure 1. Julius architecture and its main components.

2.1 Access Layer

Julius comes with a simple and neat web-based user interface through which the users can have access to the provided functionalities. In the user interface there is a separation of the different steps: network topology and traffic demands creation, control and execution of emulation, results presentation and monitoring of the emulated network infrastructure. In particular, Julius user interface enables users to easily and graphically design various network topologies by defining network nodes, devices and links between them along with their characteristics, save, edit, and use them at any time. Similarly, users can define their own traffic demands, in terms of new connections to be established between source-destination pairs and setup node and link failures. Furthermore, they can check the current network status, including established lightpaths, link utilization details, devices used at the nodes, and also have access to other useful information, such as logging details for execution requests and key actions performed by the orchestration layer.

The users' actions on the web interface are translated to commands sent to the orchestration layer and from there to the execution layer for setting up the emulation: create or tear down nodes and links, for creating new connections, for tearing down established connections, or for re-optimizing the current network state. Furthermore, the user interface is dynamically updated based on the changes in the underlying network topology through the supported WebSocket mechanism. Essentially, the access layer enables users to build in a few minutes various network emulation scenarios, execute them and analyse their performance easily using the exposed interfaces.

2.2 Orchestration Layer

The orchestration layer implements the application logic for the creation and management of the emulation environment and orchestrates the execution of the users' requests from the access layer to the execution layer. It consists of the Julius Orchestrator, the SDN controller and a number of auxiliary components that are responsible for the creation and management of the emulated network infrastructure. The Julius Orchestrator's primary task is the preparation and management of the emulation based on the user preferences from the access layer. Initially, when a user selects to start an emulation for a specific network setup, Julius Orchestrator creates

the necessary configuration files for the auxiliary components (Mininet, LINC-OE) that implement the emulated network, sets their characteristics according to the user preferences and bootstraps them. Furthermore, Julius Orchestrator constantly monitors all the orchestration layer components and reacts accordingly either to changes requested from the access layer or to any malfunction.

The second basic component in the orchestration layer is the SDN controller, which has the overall picture of the network and is exclusively responsible for its management. For the implementation of the SDN controller we have chosen the Ryu SDN framework [12], an open source software written in python language that supports OpenFlow enabled switches. Furthermore, Ryu offers a RESTful API with JSON data format that provides access information and configuration capabilities. In Julius, we have created an extended version of Ryu that interacts with the network devices (both Packet and Optical) via the appropriate protocols (OpenFlow and OpenFlow with Optical Extensions) and provides the end users a RESTful API through which it receives and serves requests from the access layer.

The third-party orchestration layer's components are based on open source tools, which are put together with the Julius Orchestrator and the SDN controller in order to build the emulation environment. Mininet [9] is used to instantiate all the low level components of the testbed, these are: LXC hosts, virtual switches and links between them. In this way, Mininet sets up the topology provided by the user. The Open Virtual Switches (OVS) [13], managed by Mininet, are used to emulate L3 switches. Julius orchestration layer interacts with these through the OpenFlow protocol. The LINC switch for Optical Emulation (LINC-OE) [11] is used to emulate ROADM and Hybrid switches. Julius interacts with them using OpenFlow 1.3 with Infoblox Optical Extensions, in order to emulate all peculiarities of optical switches.

Part of the orchestration layer are also the databases that contain details regarding the capabilities and the current state of the network infrastructure: the Traffic Engineering Database (TED) and the Label Switched Path Database (LSP-DB). TED contains topology information about the existing nodes, the links that connect them and traffic engineering information (such as the range of spectrum occupied per link), providing each time the current view of the network. LSP-DB contains information on the state of the LSPs (such as traversed interfaces, bit-rate, occupied frequency slot) that are provisioned and operational within the network. The contents of these two databases, which are kept up to date from the Julius Orchestrator and the SDN controller, are very important for the efficient operation of the execution layer.

2.3 Execution Layer

The execution layer consists of the Path Computation Element (PCE) that provides to the orchestration layer the required algorithmic logic for performing efficiently the various network resource management decisions. PCE communicates with the orchestration layer and more specifically with the SDN controller, using the Path Computation Element Protocol (PCEP), receiving requests for serving new connections or re-optimizing existing ones so as to improve the current network state. PCE is responsible for the selection of the appropriate algorithm, the monitoring of the execution progress, and the handling of the final results or possible failures. The SDN controller, which acts as Path Computation Client (PCC), receives the results for its requests from the PCE through the appropriate PCEP response messages.

For the PCE implementation, we have used in Julius an extended version of Mantis [8], a network planning and operation tool for designing the next generation optical networks, supporting both flexible and mixed line rate (MLR) WDM networks. Although, Mantis was originally designed as a standalone tool, it has been extended to integrate the necessary functionality to operate as a PCE server and support the PCEP protocol. It contains algorithms for a number of network resource management operations including routing and wavelength assignment, route and spectrum allocation, push-pull operations, rerouting of established connections, bandwidth on demand and calendaring operations, protection and restoration decisions. Also, its modular architecture design allows new algorithms to be easily plugged in. In the Julius operation context Mantis acts as a stateful PCE computing network paths and routes based on the current network state (information from the TED) and applying computational constraints, while it also takes into account the state of all the previously computed and established paths along with their required resources (information from the LSP-DB).

3. JULIUS OPERATION

The initiation of the emulation is performed through the web-based user interface or the exposed RESTful API. In any case, it is necessary first to create through the web-based interface a particular emulated scenario by defining details regarding the network topology, the existing devices in each node (along with their operation profiles) and possible pre-established connections. The Julius Orchestrator based on the provided information initiates and configures the auxiliary components in order to get the emulated network infrastructure, creating the described hosts and OVS or LINC-OE switches (depending on whether the node is IP or Optical). Next, it creates the links, configuring appropriately the respective switches. In the end, it populates the TED, initializes the LSP-DB database and starts the SDN controller.

When the initialization phase is completed the network is configured according to the specified characteristics and the users can execute several network operations. The SDN controller either forwards the user

requests to the execution layer (establish new connection or perform network re-optimization) or handles it internally (tear down an established connection). In the first case, the SDN controller sends the appropriate PCEP message to the PCE and when it gets back the response configures the emulated network infrastructure. In any case, the contents of the TED and LSP-DB are updated accordingly. In this way users can evaluate the way algorithms' (mantis) decisions affect the network performance or even compare different algorithms under common conditions. Furthermore, they can use various commands through the provided Mininet prompt to further inspect the provided network connectivity or to get additional details about the configuration of the network components.

Figure 2a shows the available user interface through which users have access to useful details for the emulated network and execute various network operations, regarding the provision of new connections or the management of existing ones. The access layer automatically checks that each provided request is valid before forward it to the orchestration layer. Figure 2b shows the provided details regarding an established connection and how exactly it is served in the network.

(a) *(b)*

Figure 2. Julius user interface (a): emulated network topology overview, (b) manage established LSPs.

4. CONCLUSIONS

Julius is an emulation environment for programmable, multilayer (IP and optical), fixed and flex-grid optical networks. Its thin design is useful when introducing disruptive changes, minimizing development time to adopt cutting edge technologies. Julius also integrates with Mantis [8] as PCE in order to provide optimization logic. Users can use Julius to perform basic and applied research on protocols (e.g., OpenFlow, PCEP) extensions and network resource reservation/routing algorithms.

ACKNOWLEDGEMENTS

This work was partially funded by the European Community's through the Horizon 2020 NEPHELE under grant agreement nº 645212.

REFERENCES

[1] Cisco Visual Networking Index – Forecast and Methodology, 2012– 2017.
[2] Ericsson Review, "IP-optical convergence: a complete solution," 2014, last seen Oct. 2015.
[3] O. Gonzalz de Dios *et al*., "CAPEX savings by scalable IP offloading approach," in *Proc. OFC*, 2011.
[4] Open Networking Foundation, Definition https://www.opennetworking.org, last seen Nov. 2015.
[5] Alcatel Lucent – Bell Labs, "The Cloud-Optimized MAN and WAN: Leveraging a multi-layer SDN framework to deliver scalable and agile cloud services," 2013.
[6] Evolution to flexible grid WDM, http://www.lightwaveonline.com/articles/print/volume-30/issue-6/features/evolution-to-flexible-grid-wdm.html, (last seen Jun. 2014).
[7] K. Christodoulopoulos, I. Tomkos, and E. A. Varvarigos, "Elastic bandwidth allocation in flexible OFDM-based optical networks," *Journal of Lightwave Technology*, vol. 29, no. 9, pp. 1354-1366, 2011.
[8] A. Kretsis, K. Christodoulopoulos, P. Kokkinos, and E. Varvarigos, "Planning and operating flexible optical networks: Algorithmic issues and tools," *IEEE Communications Magazine*, vol. 52, pp. 61-69, 2015.
[9] Mininet – An Instant Virtual Network on your Laptop (or other PC) : http://mininet.org/
[10] ONOS: http://onosproject.org/
[11] LINC-OE Optical Switch Emulation: https://github.com/FlowForwarding/LINC-Switch
[12] Ryu SDN framework: http://osrg.github.io/ryu
[13] Open vSwitch: http://openvswitch.org/