

# OSHI - Open Source Hybrid IP/SDN networking and Mantoo - a set of management tools for controlling SDN/NFV experiments

Stefano Salsano<sup>(1)</sup>, Pier Luigi Ventre<sup>(2)</sup>, Francesco Lombardo<sup>(1)</sup>, Giuseppe Siracusano<sup>(1)</sup>, Matteo Gerola<sup>(3)</sup>, Elio Salvadori<sup>(3)</sup>, Michele Santuari<sup>(3)</sup>, Mauro Campanella<sup>(2)</sup>, Luca Prete<sup>(4)</sup>

(1) CNIT / Univ. of Rome Tor Vergata - (2) Consortium GARR - (3) CREATE-NET - (4) ON.Lab

Submitted paper under revision – May 2015

**Abstract** – The introduction of SDN in large-scale IP provider networks is still an open issue and different solutions have been suggested so far. In this paper we propose a hybrid approach allowing the coexistence of traditional IP routing with SDN based forwarding within the same provider domain. The solution is called OSHI – Open Source Hybrid IP/SDN networking as we have fully implemented it combining and extending Open Source software. We discuss the OSHI system architecture and the design and implementation of advanced services like Pseudo Wires and Virtual Switches. Moreover we designed and implemented a set of Open Source management tools for the emulation of the proposed solution over the Mininet emulator and over distributed testbeds. We refer to this suite of tools as Mantoo (Management tools). Mantoo includes an extensible web-based graphical topology designer providing different layered network “views” (e.g. from physical links to service relationships among nodes). The framework is able to validate a topology, to automatically deploy it over a Mininet emulator or a distributed SDN testbed, to access nodes by opening consoles directly via the web GUI. Finally, Mantoo provides tools for evaluating the performance of the deployed solution.

**Keywords** - *Software Defined Networking, Open Source, Network management tools, Emulation.*

## I. INTRODUCTION

Software Defined Networking (SDN) [1] [2] is proposed as the new paradigm for networking that may drastically change the way IP networks are run today. Important applications have been found in Data Centers and in corporate/campus scenarios and there have been several proposals considering SDN applicability in wide area IP networks of large providers. Currently, these geographical area networks are mostly run with a combination of IP and MPLS technologies. We observe that IP/MPLS control and forwarding planes have been optimized in the years and are capable to operate on large-scale carrier networks, while SDN technology has not reached the same maturity level. The advantage of introducing SDN technology in a carrier grade IP backbone is not related to performance improvements. We rather believe that the openness of the SDN approach eliminates the need of complex distributed control planes architectures with closed implementations and difficult interoperability, facilitates the development of new services and fosters innovation. The importance of Open Source in SDN is highlighted in [3] and the rising interest on white box networking [4] confirms its relevance in current and near future networking arena.

Taking the openness as the main driver for moving to SDN, the scientific and technological question “what is the best way to introduce SDN in large-scale IP Service Providers (ISP) networks?” is definitely still open and different solutions have been proposed. The OSHI (Open Source Hybrid IP/SDN) networking architecture, first introduced in [5], addresses the above question, providing an Open Source

reference implementation complemented with a rich set of services and management tools.

The introduction of SDN in wide area ISP networks requires solving critical issues, such as:

- how to provide the scalability and fault tolerance required in operators’ environments;
- how to cope with the high latency in the control plane (due to a geographically distributed deployment).

Let us consider two key aspects in the resolution of the above issues: (i) design, implementation and testing of a solution; (ii) evaluation of a solution and comparison of different ones. As for (i), the development and testing of a solution ideally requires the availability of a large-scale test network with real hardware nodes on which to deploy the solution. This is clearly an expensive approach, which is typically not feasible either for a researcher or for research entities. As for (ii), the evaluation of a solution (and the comparison among different ones) is a complex task. First of all the functional properties of the solution need to be assessed: what services can be offered to the customers and what management tools will be available to the service provider itself. Then, several non-functional properties need to be evaluated, from quantitative properties like node performances in processing packets/flows, to qualitative properties that are more difficult to assess, like “network scalability” or “easiness to deploy new services”.

In order to support both the development/testing aspects and the evaluation/comparison it is fundamental to have a realistic emulator platform. The platform should be capable to scale up to hundreds of nodes and links, representing an IP carrier network at a reasonably large scale. Performing experiments should be affordable also for research and academic teams, not only for corporate developers. Therefore we advocate the need of an Open Source reference node implementation and of an Open Source emulation platform. On top of the platform the researcher/developer should be able to design and deploy services with a minimal effort. The management of these emulation platforms and the tools for setting up and controlling experiments are also non-trivial problems, which is why we propose an Open Source set of tools called Mantoo (Management tools).

The main contributions of this paper are:

1. Design of a hybrid IP/SDN architecture called Open Source Hybrid IP/SDN (OSHI).
2. Design and implementation of a hybrid IP/SDN node made of Open Source components.
3. Mantoo, a set of management tools to deploy and test the OSHI framework and services on Mininet emulator and on distributed SDN testbeds

#### 4. Evaluation of some Key Performance Indicators (KPIs) of the proposed architecture and its implementation.

The paper is structured as follows: in section II we describe the scenarios related to the introduction of SDN in IP Service Providers networks and the state of the art of similar solutions; in section III we define the main concepts of the proposed hybrid IP/SDN networking architecture; section IV provides a detailed description of the OSHI nodes implementation and of the services that such a solution can offer, it also identifies some limitations of current SDN ecosystem and propose the needed extensions; section V describes the Mantoo suite, that allows to design, deploy and control experimental topologies in a local emulator (Mininet) or on distributed testbeds, supporting the collection of performance measurements; section VI provides an evaluation of some performance aspects; in section VII we draw some conclusions and highlight how we are porting OSHI over white box switches, potentially stepping from experiments to production networks.

The source code of all the components of the OSHI node prototypes and of the Mantoo suite is freely available at [6]. In order to ease the initial environment setup, everything has also been packaged in a ready-to-go virtual machine, with pre-designed example topologies up to 60 nodes. To the best of our knowledge, there is no such hybrid IP/SDN node available as Open Source software, nor an emulation platform with a set of management tools like the Mantoo suite.

## II. SDN APPLICABILITY IN IP PROVIDERS NETWORKS

SDN is based on the separation of the network control and forwarding planes. An external SDN controller can (dynamically) inject rules in SDN capable nodes. According to these rules the SDN nodes perform packet inspection, manipulation and forwarding, operating on packet headers at different levels of the protocol stack.

We focus on SDN applicability in IP Service Providers networks. Figure 1 shows a reference scenario, with a single IP provider interconnected with other providers using the BGP routing protocol. Within the provider network, an intra-domain routing protocol like OSPF is used. The provider offers Internet access to its customers, as well as other transport services (e.g. layer 2 connectivity services or more in general VPNs - Virtual Private Networks). Using the terminology borrowed by IP/MPLS networks, the provider network includes a set of Core Routers (CR) and Provider Edge (PE) routers, interconnected either by point-to-point links (Packet Over Sonet, Gigabit Ethernet, 10GBE...) or by legacy switched LANs (and VLANs). The Customer Edge (CE) router is the node in the customer network connected to the provider network. Most often, an ISP integrates the IP and MPLS technologies in its backbone. MPLS creates *tunnels* (LSP - Label Switched Path) among routers. On one hand, this can be used to improve the forwarding of regular IP traffic providing: i) traffic engineering, ii) fault protection iii) no need to distribute the full BGP routing table to intra-domain transit routers. On the other hand, MPLS tunnels are used to offer VPNs and layer 2 connectivity services to customers. In any case, the commercial MPLS implementations are based on traditional (vendor-locked)

control plane architectures that do not leave space for introducing innovation in an open manner. As a matter of fact, in case of complex services involving the MPLS control plane, IP Service Providers rely on single-vendor solutions. The management of large-scale IP/MPLS network is typically based on proprietary (and expensive) management tools, which - again - constitute a barrier to the innovation.

Let us consider the migration of a Service Provider network, based on IP/MPLS, to SDN. CR and PE routers could be replaced by SDN capable switches, on top of which the provider can realize advanced and innovative services. The migration paths should foresee the coexistence of IP and SDN based services, in a hybrid IP/SDN scenario (resembling the current coexistence of IP and MPLS). According to the taxonomy defined in [7], this approach can be classified as “Service-Based” or “Class-Based” Hybrid SDN (depending on how the IP and SDN based services are combined). In this migration scenario a set of hybrid IP/SDN nodes are capable of acting as plain IP routers (running the legacy IP routing protocols), as well as SDN capable nodes, under the control of SDN controllers.

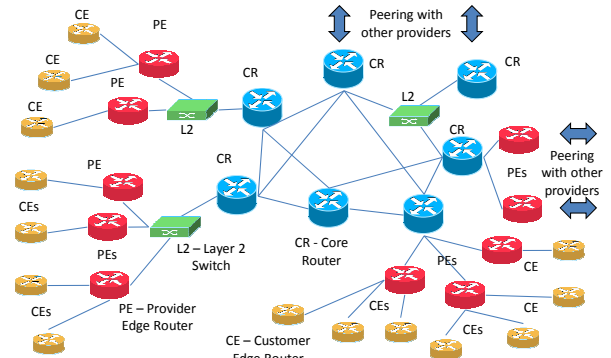


Figure 1. Reference scenario: an IP provider network

### A. State of the art

Pure SDN solutions based on L2 switches inter-connected with a centralized controller have been demonstrated both in data-centers and in geographically distributed research networks, such as OFELIA [8] in EU, GENI [9] and Internet2 [10][11] in US. To the best of our knowledge, these solutions do not integrate L3 routing within the SDN capable L2 switches. We argue that an ISP network requires a more sophisticated approach that can natively interwork with legacy IP routers and IP routing protocols. As stated in [7], a hybrid SDN model that combines SDN and traditional architectures may “sum their benefits while mitigating their respective challenges”. Some recent works address the hybrid IP/SDN networking from different perspectives.

In [12] the authors presented an Open Source Label Switching Router that generates OSPF and LDP packets using Quagga. The node computes the MPLS labels that are then installed in the switches using the OpenFlow (OF) protocol. This architecture does not exploit a logically centralized controller. Instead, it considers a traditional distributed control plane, while it uses OF only locally in a node to synchronize the FIBs and to program the data plane.

RouteFlow [13] creates a simulated network made of virtual routers at the top of a SDN controller. The simulated

network is a copy of the physical one. The controller uses the BGP protocol to interact with routers of neighbor domains and it simulates intra domain protocols (OSPF, IS-IS) between the virtual routers. A traditional IP routing engine (Quagga [14]) computes the routing tables that are eventually installed into the physical nodes via the OF protocol. The Cardigan project [15] is based on a fork of RouteFlow. Cardigan realized a distributed router based on RouteFlow concepts and deployed it in a public Internet exchange, showing the applicability of SDN/OpenFlow in a production context. The “SDN-IP” solution proposed in [16] follows similar principles. It is based on the ONOS SDN controller [17] and it also interacts with external domains using BGP. Differently from RouteFlow, the controller does not instantiates virtual routers to simulate the exchange of intra domain routing protocols, but it centralizes the routing logic for better efficiency.

Compared with these works, our solution assumes that the physical nodes still deal with basic IP routing, thus achieving fault-tolerance based on standard IP routing and easier interoperability with non-OF devices in the core network. On top of the basic routing, the SDN/OpenFlow controller can instruct the hybrid IP/SDN nodes to perform SDN based forwarding for specific traffic flows. This idea of supporting such hybrid nodes is already included in the OpenFlow specifications since the first version of the protocol. Two types of devices are considered: OF-only and OF-hybrid which can support both OF processing and standard L2/L3 functionalities (in OpenFlow version 1.0 the term “OF-enabled” was used instead of “OF-hybrid”). Currently, only proprietary hardware switches implement the hybrid approach offering also L3 standard routing capabilities. OSHI represents a fully Open Source OF-hybrid solution designed to be flexible and scalable, so as to facilitate experimentation on hybrid IP/SDN networks at large scale.

The Google B4 WAN [18] is an integrated hybrid IP SDN solution, and it has likely been the first application of the SDN approach to a large-scale WAN scenario. In the B4 solution the traditional distributed routing protocols coexist with an SDN/OpenFlow approach. In particular, the B4 WAN sites are interconnected with traditional routing and the SDN-based centralized Traffic Engineering solution is deployed as an overlay on top of basic routing. Google B4 solution is proprietary and it is highly tailored to the needs of their specific scenario. On the other hand, OSHI is designed as a generic and open solution for hybrid IP/SDN networks.

### III. PROPOSED HYBRID IP/SDN ARCHITECTURE

In the IP/MPLS architecture there is a clear notion of the MPLS tunnels, called Label Switched Paths (LSPs). In a SDN network several types of tunnels or - more generically - *network paths* can be created, leveraging on the ability of SDN capable nodes to classify traffic based on various fields such as MAC or IP addresses, VLAN tags and MPLS labels. Since there is not a standard established terminology for such concept, we will refer to these paths as *SDN Based Paths* (SBP). A SBP is a “virtual circuit” setup using SDN technology to forward a specific *packet flow* between two end-points across a set of SDN capable nodes. The notion of

packet flow is very broad and it can range from a *micro-flow* i.e. a specific TCP connection between two hosts, to a *macro-flow* e.g. all the traffic directed towards a given IP subnet. As highlighted before, a flow can be classified looking at the headers at different protocol levels.

We address the definition of the hybrid IP/SDN network by considering: i) mechanisms for the coexistence of regular IP traffic and SBPs; ii) the set of services that can be offered using the SBPs; iii) ingress traffic classification and tunneling mechanisms.

Let us consider the coexistence of regular IP traffic and SDN based paths on the links between hybrid IP/SDN nodes. A SDN approach offers a great flexibility, enabling the classification of the packets through a “cross-layer” approach, by considering packet headers at different protocol levels (MPLS, VLANs, Q-in-Q, Mac-in-Mac and so on). Therefore it is possible to specify a set of conditions to differentiate the packets to be delivered to the IP forwarding engine from the ones that belong to SBPs. In general, these conditions can refer to different protocol headers and can be in the form of white- or black- lists, changing dynamically, interface by interface. This flexibility may turn into high complexity. Therefore, the risk of misconfigurations and routing errors should be properly taken into account (see [19]). Without preventing the possibility to operate additional mechanisms for the coexistence of IP and SDN services in a hybrid IP/SDN network, we propose MPLS tagging as the preferred choice and VLAN tagging as a sub-optimal choice. Both solutions have been implemented in our platform.

Let us now consider the services and the features that can be offered by a hybrid IP/SDN network. As primary requirements we assume three main services/functionalities: (i) virtual private networks (Layer 2 and Layer 3), (ii) traffic engineering, (iii) fast restoration mechanisms. Moreover, the architecture should facilitate the realization of new services and the development of new forwarding paradigms (for example Segment Routing [20]) without the need of introducing complex and proprietary control planes.

As for the traffic classification, the ingress PEs need to classify incoming packets and decide if they need to be forwarded using regular IP routing or if they belong to the SBPs. The egress edge router will extract the traffic from the SBPs and forward it to the appropriate destination. We considered (and implemented in our platform) two approaches for the ingress classification: i) classification based on physical access ports; ii) classification based on VLAN tags. As the ingress classification is not a specific feature of our architecture, we did not consider other traffic classification, e.g. based on MAC or IP source/destination addresses. However, these approaches can be easily implemented without changing the other components.

Finally, we considered a VLAN based tunneling mechanisms and two MPLS based tunneling mechanisms: plain IP over MPLS ([21], here referred to as IPoMPLS) and Ethernet over MPLS (EoMPLS [22] [23]). Different services are supported by these tunneling mechanisms: the IPoMPLS and the VLAN based tunneling only support the relay of IP and ARP packets between two IP end points. We refer to this service as IP Virtual Leased Line (IP VLL). The EoMPLS

tunneling mechanism can support the relaying of arbitrary layer 2 packets, providing the so called Pseudo-Wire (PW) service [22].

#### IV. DETAILED DESIGN OF THE HYBRID IP/SDN SOLUTION

In this section we present the detailed design and the implementation of the proposed architecture. We describe the Open Source tools that we have integrated and how their practical limitations have been taken into account to deliver a working prototype. We will first introduce the high level architecture of an OSHI node (IV.A) and the basic services we provide (IP Virtual Leased Line and Pseudo-wires, IV.B). Then we describe the two different options of using VLAN tags or MPLS labels to realize SDN Based Paths (SBPs) and to support the coexistence between IP based forwarding and SBP forwarding. As we will show, the design of the MPLS based option has been more challenging and the resulting implementation is more complex. In part this is due to inherent limitation of OpenFlow architecture, in part to the shortcomings of the Open Source tools that we have integrated.

##### A. OSHI High Level Node Architecture

We designed our Open Source Hybrid IP/SDN (OSHI) node combining an OpenFlow Capable Switch (OFCS), an IP forwarding engine and an IP routing daemon. The OpenFlow Capable Switch is connected to the set of physical network interfaces belonging to the integrated IP/SDN network, while the IP forwarding engine is connected to a set of OFCS virtual ports, as shown in Figure 2. In our OSHI node, the OFCS component is implemented using Open vSwitch (OVS) [27], the IP forwarding engine is the Linux kernel IP networking and Quagga [14] acts as the routing daemon.

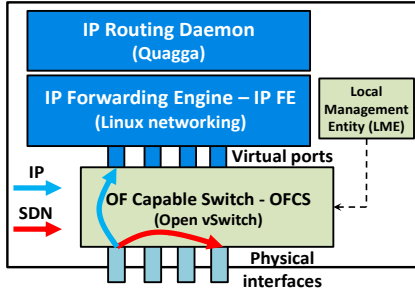


Figure 2. OSHI Hybrid IP/SDN node architecture

The internal virtual ports that interconnect the OFCS with the IP forwarding engine are realized using the “Virtual Port” feature offered by Open vSwitch. Each virtual port is connected to a physical port of the IP/SDN network, so that the IP routing engine can reason in term of the virtual ports, ignoring the physical ones. The OFCS differentiates among regular IP packets and packets belonging to SDN Based Paths. By default, it forwards the regular IP packets from the physical ports to the virtual ports, so that they can be processed by the IP forwarding engine, controlled by the IP routing daemon. This approach avoids the need of translating the IP routing table into SDN rules to be pushed in the OFCS table, at the price of a small performance degradation for the packets that needs to be forwarded at IP level. In fact, these packets cross the OFCS switch twice. It is possible to extend

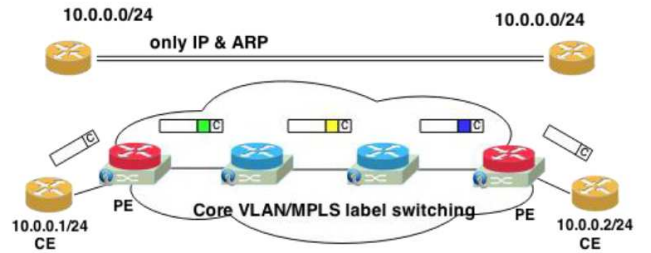
our implementation to consider the mirroring of the IP routing table into the OFCS table, but this is left out as future work.

An initial configuration of the OFCS tables is needed to connect the physical interfaces and the virtual interfaces, in order to support the OFCS-to-SDN-controller communication and some specific SDN procedures (for example to perform Layer 2 topology discovery in the SDN controller). A Local Management Entity (LME) in the OSHI node takes care of these tasks. In our setup, it is possible to use an “in-band” approach for OFCS-to-SDN-controller communication, i.e. using the regular IP routing/forwarding and avoiding the need of a separate out-of-band network. Further details and the block diagram of the control plane architecture of OSHI nodes are reported in [25].

##### B. OSHI basic services: IP VLL and PW

We designed and implemented two basic services to be offered by the hybrid IP/SDN networks: the IP “Virtual Leased Line” (IP VLL) and the Layer 2 “Pseudo-wire” (PW) see Figure 3. Both services are offered between end-points in Provider Edge routers, the end-points can be a physical or logical port (i.e. a VLAN on a physical port) of the PE router connected to a Customer Edge (CE). The interconnection is realized in the core hybrid IP/SDN network with an SBP using VLAN tags or MPLS labels.

###### IP Virtual Leased Line



###### L2 Pseudo Wire

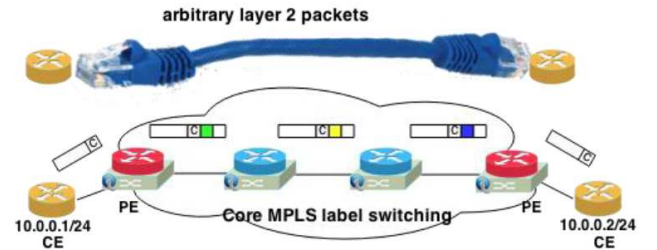


Figure 3. IP VLL and PW services

The IP VLL service guarantees to the IP end-points to be directly interconnected as if they were in the same Ethernet LAN and sending each other IP and ARP packets. It is not meant to allow the served SBP end-points to send packets with arbitrary Ethertype (e.g. including VLAN packets). We implemented the IP VLL service both using VLAN tags and MPLS tags. In both cases the original source and destination MAC addresses, shown as “C” (for Customer) in the headers of the packets in Figure 3, are preserved in the transit along the network core. This may cause problems if legacy L2 switches are used to interconnect OSHI nodes, therefore our implementation of IP VLL service can only work if all edge and core nodes are OSHI capable and are connected to each

other, without legacy intermediate switches in between. In principle, one could implement MAC address rewriting replacing the customer addresses with the addresses of the ingress and egress PEs or on a hop-by-hop case. This is rather complex to realize and to manage, because the egress node should restore the original addresses, needing state information per each SBP, so we did not follow this option.

The PW service is also known as “Pseudowire Emulation Edge to Edge” (PWE3), described in RFC 3985 [22]. The PW service provides a fully transparent cable replacement service: the endpoints can send packets with an arbitrary Ethertype (e.g. including VLAN, Q-in-Q). We implemented the PW service only using MPLS tags. The customer Ethernet packet is tunneled into an MPLS packet as shown in Figure 3. This approach solves the interworking issues with legacy L2 networks related to customer MAC addresses exposure in the core.

### C. OSHI - VLAN based approach

#### 1) Coexistence mechanisms

Two coexistence mechanisms have been designed: i) *Tagged coexistence*, in which the IP traffic is transported with a specific VLAN tag, while SBPs use different VLAN tags; ii) *Untagged coexistence*, in which the IP traffic travels “untagged” and SBPs use VLAN tags. Both mechanisms can interwork with existing legacy VLANs, as long as a set of tags is reserved for our use on each physical subnet/link (and this is needed only if the link between two routers already runs on a VLAN). As for the implementation of coexistence mechanisms we used the multiple tables functionality. It has been introduced with OF v1.1, but it is possible to emulate it using OF v1.0 with the so-called “Nicira extensions”. We have initially leveraged the Nicira extensions implementation in Open vSwitch and then we migrated to standard multiple tables when their support has been included in Open vSwitch. For *Untagged coexistence*, the incoming untagged traffic will be directed to a virtual port (toward the IP forwarding engine), while incoming tagged traffic will be switched to a physical outgoing port, according to the configuration of the SBPs (VLAN tag can be switched appropriately). For *Tagged coexistence*, incoming traffic tagged with a specific VLAN tag for regular IP traffic will be directed to a virtual port (toward the IP forwarding engine), while the traffic tagged with the other VLAN tags will be switched to a physical port, according to the configuration of the SBPs (VLAN tag can be switched appropriately). The details are reported in [25].

#### 2) Ingress classification and tunneling mechanisms

As for the ingress classification functionality in a Provider Edge router, we use the input port to classify untagged traffic as *regular IP* traffic or as belonging to the end-point of a VLL. For the VLAN tagged traffic entering in a physical port of a PE router, each VLAN tag can be individually mapped to a VLL end point or assigned to regular IP traffic. For the untagged traffic, the implementation of the ingress classification is realized within the OFCS of the OSHI Provider Edge nodes. By configuring rules in the OFCS, it is possible to map the untagged traffic on an ingress physical port to a virtual port (for regular IP) or to an SBP (for a VLL

end-point). For the tagged traffic, the incoming classification relies on the VLAN handling of the Linux networking: each VLAN tag  $x$  can be mapped to a virtual interface `eth0.x` that will simply appear as an additional physical port of the OFCS.

The IP VLL service is realized with an SBP that switches VLAN tags between two end-points (in both directions). This has a severe scalability issue, as the number of SBPs on a link is limited to 4096. Moreover, if legacy VLAN services are implemented on the links among the OSHI nodes, the VLAN label space needs to be partitioned between the SBPs and the legacy VLAN services. This reduces the maximum number of SBPs available on a link may drastically complicate the service management process.

We have implemented the IP VLL service using a VLAN based approach on top of the Floodlight controller [26]. In particular, the SBPs are setup using a python script called VLLPusher. The script uses the Topology REST API of the Floodlight controller in order to retrieve the route that interconnects the VLL end-points. The script allocates the VLAN tags and then uses the Static Flow Pusher REST API of Floodlight in order to set the rules for packet forwarding and VLAN tag switching.

#### 3) Requirements on protocol and tools versions

The proposed implementation of the VLAN based approach only requires OpenFlow 1.0 compliance in the controller. On the OFCS side, it requires the support for multiple tables, which have only been introduced in OF 1.1 but can be emulated using the proprietary “Nicira extensions”. In fact, the Local Management Entity uses the multiple table features for the initial configuration of the OFCS, while the run-time rules injected by the controller use OF v1.0. In our first implementation we relied on the Nicira extensions implemented in Open vSwitch, hence we were able to release a working prototype of a hybrid IP/SDN network using Open Source components in May 2014.

### D. OSHI - MPLS based approach

In this subsection we illustrate the MPLS based approach. With respect to the VLAN based solution, this approach extends the functionality and the services offered by the OSHI networks, solving the scalability issues (in terms of number of SBPs that can be supported on each link) coming from the former solution. The use of MPLS labels enables the establishment of up to  $2^{20}$  (more than  $10^6$ ) SBPs on each link. We have implemented both IP VLL and PW services with MPLS, in both cases the MPLS solution does not interfere with VLANs that can potentially be used in the links between OSHI nodes.

#### 1) Coexistence mechanisms

For the coexistence of regular IP service (best effort traffic) and SDN services (using SDN Based Paths) we use the Ethertype field of the L2 protocol. This corresponds to one of the mechanisms that can be used in the IP/MPLS model: regular IP traffic is carried with IP Ethertype (0x0800), while SBPs are carried with MPLS Ethernets (0x8847 and 0x8848). Using OpenFlow multi-table functionality, our solution support the coexistence of IP and MPLS traffic types, as shown in Figure 4. Table 0 is used for regular IP, ARP,

LLDP, BLDP, etc., table 1 for the SBPs. In particular, Table 0 contains: i) two rules that forwards the traffic with the Ethertypes 0x8847 (MPLS) and 0x8848 (Multicast MPLS) to Table 1; ii) the set of rules that “bridge” the physical interfaces with Virtual Interfaces and vice versa; iii) two rules that forward the LLDP and BLDP traffic to the controller. Table 1 contains the set of rules that forward the packets of the SBPs according to the associated IP VLL or PW service. The coexistence in Table 0 is assured through different levels of priority. The IP VLL service needs both the rules associated to unicast and multicast MPLS Ethertype (more below), while the PW service only needs a rule matching the unicast MPLS Ethertype.

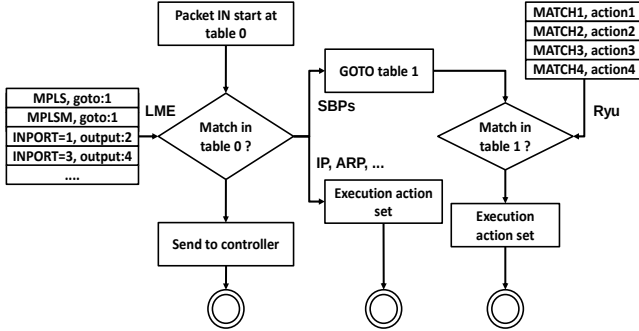


Figure 4. Packet processing in the OFCS flow tables

## 2) Ingress classification and tunneling mechanisms

As regards the ingress classification, the same mechanisms described for the VLAN approach are reused: the classification can be either based on the physical input port or on the incoming VLAN.

Let us analyze the tunneling mechanisms in the OSHI MPLS approach. Figure 5 shows the tunneling realized by OSHI-PE for the IP VLL service in the left half. C stands for Customer, the ingress direction is from customer to core, egress refers to the opposite direction. This solution borrows the IPoMPLS approach, in which an MPLS label is pushed within an existing frame. In this case an input Ethernet frame carrying either an IP or an ARP packet, keeps its original Ethernet header, shown as C-ETH in Figure 5. As we have already stated for the VLAN implementation of the IP VLL, this solution has the problem of exposing the customer source and destination MAC addresses in the core. Moreover, note that the MPLS Ethertype (0x8847) overwrites the existing Ethertype of the customer packets. This does not allow the distinction between IP and ARP packets at the egress node. A solution would be to setup two different bidirectional SBPs: one for the IP and one for the ARP packets. In order to save label space and simplify the operation we preferred to carry IP packets with the MPLS Ethertype and to (ab)use multicast MPLS Ethertype (0x8848) to carry the ARP packets. With this approach, the same MPLS label can be reused for the two SBPs transporting IP and ARP packets between the same end-points.

The “Ethernet over MPLS” (EoMPLS) encapsulation [23] represents the most efficient approach to implement the PW service. As shown in Figure 5, EoMPLS encapsulates the customer packet including its original Ethernet header in an MPLS packet to be carried in a newly generated Ethernet header. Unfortunately, we require a solution that can be

implemented using an Open Source switch and we would like to have a solution that can be fully controlled by OpenFlow. The OpenFlow protocol and most OpenFlow capable switches (including Open vSwitch that we are using for our prototype) do not natively support EoMPLS encapsulation and decapsulation. A similar issue has been identified in [32], in which the authors propose to push an Ethernet header using a so called “input Packet Processing” (iPProc) function before handing the packet to a logical OpenFlow capable switch that - in turn - will push the MPLS label. Obviously this requires a switch with an “input Packet Processing” function capable of pushing an Ethernet header into an existing Ethernet packet. Note that this process is not fully controlled with OpenFlow protocol, as OpenFlow does not support the pushing of an Ethernet header. We cannot directly follow this approach, as Open vSwitch is not capable of pushing Ethernet headers. The right half of Figure 5 shows the approach that we have followed, relying on GRE encapsulation. P stands for Provider and it indicates the headers added/removed by the PE. A packet in the PE is processed in four steps (shown as i1 to i4 in the ingress direction from the CE towards the core and as e1 to e4 in the egress direction from the core toward a customer). The GRE encapsulation introduces an additional overhead (20 bytes for P-IP and 4 bytes for GRE headers) to the standard EoMPLS, but it allowed us to rely on Open Source off-the-shelf components.

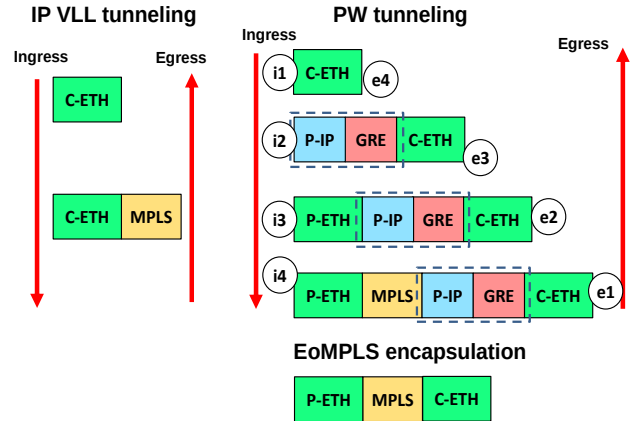


Figure 5. IP VLL and PW tunneling operations at the Provider Edges. The EoMPLS encapsulation format is shown as a reference.

The implementation of the proposed approach required a careful design, whose result is shown in Figure 6. A new entity called ACcess Encapsulator (ACE) is introduced in order to deal with the GRE tunnel at the edges of the pseudo wire tunnel. The detailed design is further analyzed in subsection IV.E.

With this approach it is possible to rewrite the outer source and destination MAC addresses in the core OSHI network, so that they can match the actual addresses of the source and destination interfaces on the OSHI IP/SDN routers. This allows the support of legacy, Ethernet switched networks among the OSHI IP/SDN routers, which can be an important requirement for a smooth migration from existing networks.

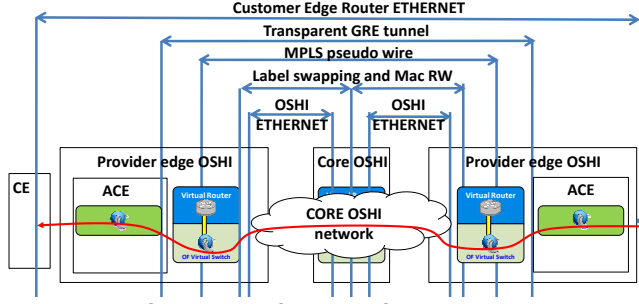


Figure 6. PW implementation in the OSHI node prototype

Both the IP VLL and PW services are realized with SBPs that switches MPLS labels between two end-points (in both directions). We used the Ryu [34] controller and adapted the VLLPusher python script to perform the setup of the SBPs. The script uses the Topology REST API of the Ryu controller in order to retrieve the shortest path that interconnects the VLL end-points. It allocates the MPLS labels and then uses the Ofctl REST API to set the rules for packet forwarding and MPLS label switching. In the setup of a PW service the MAC rewriting actions are added, using the addresses of the OSHI nodes as the outer MAC addresses.

### 3) Requirements on protocol and tools versions

The MPLS solution needs at least OpenFlow v1.1, which makes possible to handle MPLS. Both the SDN controller and the SDN Capable Switch need to support at least OF v1.1 (most controller and switches jumped from OF v1.0 to v1.3). Considering our tools, an Open vSwitch version compliant with OF v1.3 has been released in summer 2014, making it possible to start the implementation of the MPLS based approach. As for the controller, we switched from Floodlight to Ryu because as of June 2014, only the latter included the support for OF 1.3.

### 4) The Virtual Switch Service (VSS), also known as VPLS

The PW service can be used as a building block for creating more complex services, like for example the Virtual Switch Service (VSS). While a PW service instance bridges two layer 2 end-points, using the VSS service a set of end-points can be transparently bridged into a virtual layer2 switch (see Figure 7). The ports of a VSS instance correspond to an arbitrary set of ports of the Provider Edge nodes. This service is called Virtual Private LAN Service (VPLS) in RFC 4761 [33]. A VSS provides the same VPLS service described in the RFC but its implementation is based on SDN and does not exploit other control plane functionalities, therefore we renamed it.

The VSS needs to relay upon the PW service, because using the IP VLL service the OSHI network does not provide a transparent forwarding of layer 2 packets. The VSS service is built using PWs that connect the end-points to one or more “bridging points” in the OSHI network in which a virtual layer 2 switch instance is reserved to bridge the packets coming from the PW.

A VSS instance is deployed in three steps: i) bridging point selection; ii) bridging point deployment; iii) bridging point interconnection. In the first step, a python script called VSSSelector retrieves the topology from the controller and then

chooses the OSHI nodes that will host the virtual switches. In the second step according to the output of VSSSelector the virtual switches are deployed as additional instances of Open vSwitch in the selected OSHI nodes (Virtual Bridging Points – VBP in Figure 9). The final step is the deployment of the PWs that will interconnect the CEs to the VBPs and the VBPs among each other. We implemented two versions of the bridging point selection (first step above): i) un-optimized; ii) optimized. In the un-optimized version a single node is randomly selected in the topology and used to deploy the virtual bridge. For the optimized version, we observed that the selection of an optimal internal topology to implement a VSS exactly corresponds to the Steiner tree problem [35]. We implemented the heuristic defined in [36] to find an approximate solution of the minimal Steiner tree. Then, using the tree topology obtained from the heuristic, a set of virtual switches is deployed only in the branching points of the tree. These branching points are connected each other and with end-points with direct Pseudo Wires. In this way the packets enters the Virtual Switches only in the branching points.

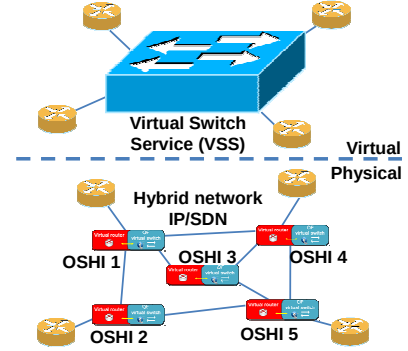


Figure 7. Virtual Switch Service (VSS) in the OSHI network

### E. OSHI detailed node architecture

In order to support the PW and VSS services, the architecture of an OSHI node needs to be more complex with respect to the high level architecture shown in Figure 2. Figure 8 provides a representation of the proposed solution for the PE nodes. As discussed above, the difficult part is the support of encapsulation and de-encapsulation in the OSHI PE nodes, for which we resorted to use GRE tunnels (see the right side of Figure 5). The different encapsulation steps in the ingress (i1-i4) and egress direction (e1-e4) are represented using the same numbering of Figure 5. The OF Capable Switch only handles the push/pop of MPLS labels, while the ACE handles the GRE encapsulation. The ACE is implemented with a separate instance of Open vSwitch, in particular we have an ACE instance running in a separate Linux network namespace [30] for each customer. For each PW, the ACE has two ports: a “local” port facing toward the CE locally connected to the PE node and a “remote” one facing towards the remote side of the PW. The remote port is a GRE port provided by OVS, therefore the ACE receives the customer layer 2 packets on the local ports and sends GRE tunneled packets on the remote port (and vice-versa). The interconnection of OFCS ports and ACE ports (the endpoints of the yellow pipes in Figure 8) are realized using the concept of Virtual Ethernet Pair [30] offered by the Linux Kernel.

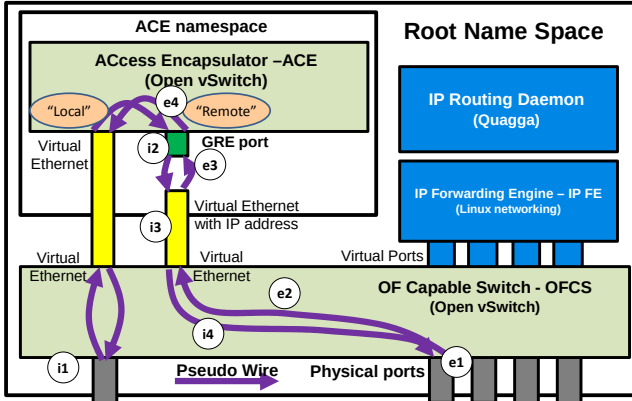


Figure 8. OSHI-PE architectural details

Differently from the Virtual Port (shown on the right side of Figure 8), the Virtual Ethernet are always associated in pairs. In our case, for each PW two Virtual Ethernet pairs are needed, one pair is used to connect the CE port of OFCS with the local port of ACE, another pair to connect the remote port of the ACE with the physical ports towards the remote side of the PW. Three virtual Ethernet endpoints are used as plain switch ports (two belong to the OFCS, one to the ACE), the last one - on the ACE - is configured with an IP address and it is used as the endpoint of the GRE tunnel (Virtual Tunnel Endpoint, i.e. VTEP). These IP addresses are not globally visible, but they have a local scope within the network namespaces associated to the customer within all the OSHI nodes. This approach greatly simplifies the management of the services, as the same addresses for the GRE VTEP can be reused for different customers. As a further simplification, static ARP entries are added on the Virtual Ethernet for each remote tunnel end (remote VTEP). For each customer, a simple centralized database of IP and MAC addresses (used for GRE tunnels) is needed.

Proper OpenFlow rules needs to be setup in the OF Capable Switch to ensure the transit of packets. On the access port (i1) these rules are provided by the LME at the time of the ACE creation, while in the i4 and e2 cases they are pushed by the OpenFlow Controller during the PW establishment.

As discussed above, an instance of ACE in the PE node is used to handle all the PWs of a single customer and runs in a private network namespace. In addition we had to configure a private folders tree for each ACE instance, as it is needed to guarantee proper interworking of difference instances of OVS in the same PE node.

Coming to the implementation of the VSS, the internal design of an OSHI node that hosts a VSS Bridging Point (VBP) is shown in Figure 9. The design is quite similar to the one analyzed before for the PW encapsulation. A VBP is implemented with an OVS instance that does not have local ports, but only remote ones. A VPB instance represents a bridging point for a single VSS instance and it cannot be shared among VSS instances.

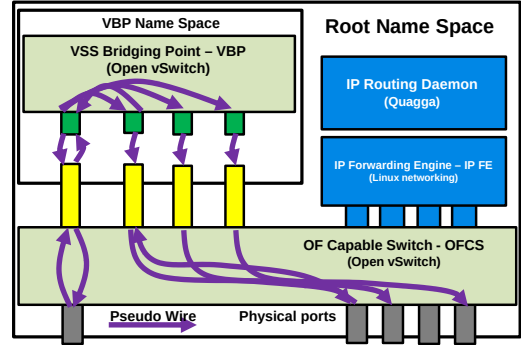


Figure 9. An OSHI node that hosts a bridging point for a VSS

### 1) Considerations on alternative design choices

Considering that a single instance of Open vSwitch can support several independent switches, a simpler design would consist in implementing the ACEs shown in Figure 8 as separate switches within the same Open vSwitch instance that runs the OFCS. For N customers, this solution would use one OVS instance instead of N and only the root network namespace instead of N additional namespaces, reducing the memory requirements versus the number of customers. The drawback of this solution is that handling the GRE tunnels of all customers in the same network namespace requires the management of disjoint IP numbering spaces for the tunnel endpoints of different customers. There is also a second reason why we preferred to implement the more complex design with the separate namespaces shown in Figure 8. In this case, the ACE can be turned in a "Virtual Router" by including an instance of a routing daemon (Quagga) in its network namespace. Such virtual router is the basic component of Layer 3 VPN services that could complement the Layer 2 PW and VSS services realized so far. With the choice of the more complex design we tradeoff scalability with simplification of the service management and easier development of new services.

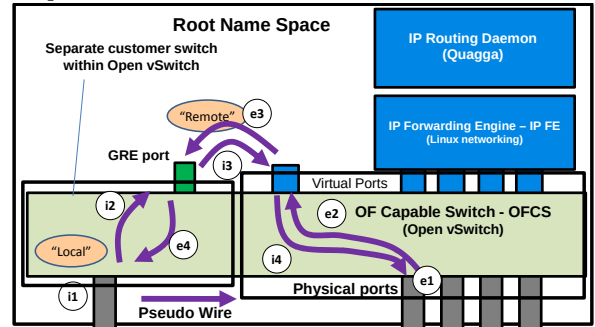


Figure 10. PW implementation design without ACE

A second consideration is that the handling of GRE tunneling has been recently introduced in Linux kernels. The use of the GRE module provided by Open vSwitch is not required anymore (it was required when we started the design of our solution). This can lead to a simpler design for tunneling that does not require the ACE, as shown in Figure 10. Anyway, this solution has the same drawbacks discussed above in terms of management of IP addresses for the tunnel endpoints, because there are not separate network namespaces for the customers, and cannot be easily extended to support Layer 3 services.

#### F. OSHI: gap analysis, ongoing and future work

Let us consider the above-described solution for PW encapsulation, based on GRE tunneling performed by the ACE. It has been designed as a replacement of the more efficient Ethernet over MPLS (EoMPLS) encapsulation described in [22], which cannot be realized by the current version of Open vSwitch. The GRE tunneling introduces a transport and a processing overhead. The transport overhead is 20 (IP header) + 16 (GRE header) bytes for each packet. The processing overhead depends on the implementation architecture. Our solution (shown in Figure 8) was not meant to be highly efficient but only to demonstrate the feasibility of the approach with a working component. We do not plan to improve the efficiency of the solution, rather we believe that native Ethernet over MPLS (EoMPLS) encapsulation should be provided by open source switches and we are considering to extend the Open vSwitch and other open source switches to support EoMPLS.

Assuming that a switch supports EoMPLS, a second important gap to be filled is the lack of support for such tunneling operations in the OpenFlow protocol. Note that the lack of encapsulation support in OpenFlow does not only concern EoMPLS, but also other tunneling solutions like GRE, VXLAN. The only tunneling solution currently supported by OpenFlow is the PBB (Provider Backbone Bridges, also known as “mac-in-mac”), but this solution is not supported by Open vSwitch. For GRE and VXLAN, using OpenFlow it is possible to control packets already tunneled (and specific matches have been introduced in OF 1.4 for VXLAN), but it is not possible to control the encapsulation (i.e. pushing the GRE, VXLAN headers) and de-encapsulation (i.e. popping the header) operations. Currently, external tools are needed to manage the GRE or VXLAN tunnel end-points (e.g. using the switch CLIs - Command Line Interfaces or switch specific protocols, like ovsdb-conf for Open vSwitch), with added complexity in the development, debug and operations. Extending OpenFlow protocol with the capability to configure the tunneling end-points would be a great simplification in the management of SDN based services.

The OSHI solution is an open starting point to design and implement additional “core” functionality and user oriented services. As for the core functionality we are considering traffic engineering mechanisms and implemented a flow assignment heuristic for optimal mapping of PWs with required capacity on the core OSHI links. As for additional services, we are considering Layer 3 VPNs based on the PW service. Following the same approach used for the VSS service, the idea is to deploy virtual router instances within the OSHI nodes that can exchange routing information with routers in the CE nodes. Finally, we are working on an Open Source implementation of Segment Routing [20] on top of OSHI. These ongoing efforts are reported on the OSHI web page [6], with links to documentation and source code.

#### V. MANTOO: MANAGEMENT TOOLS FOR SDN/NFV EXPERIMENTS ON MININET AND DISTRIBUTED SDN TESTBEDS

Mantoo is a set of Open Source tools meant to support SDN experiments over emulators and distributed testbeds. Mantoo is able to drive and help the experimenters in the

different phases that compose an experiment: design, deployment, control and measurement, as described in the next subsections. Mantoo includes: a web based GUI called Topology3D (Topology and Services Design, Deploy and Direct, Figure 11), a set of scripts to configure and control emulators or distributed testbeds; a set of scripts for performance measurements. The overall Mantoo workflow is represented in Figure 12. Using the Topology3D, the user can design its experiment in terms of physical topology and services, start the deployment of the topology and run the experiments exploiting the provided measurement tools. The design of Mantoo and of its components is modular and it can be easily extended to support scenarios that go beyond the use cases of our interest.

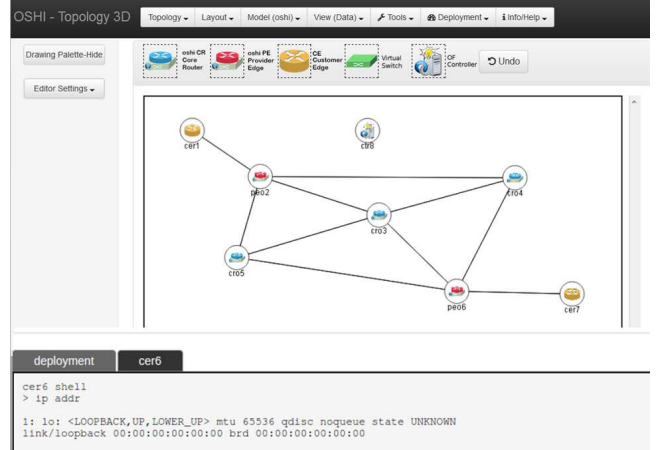


Figure 11. The Topology3D (Topology and Services Design, Deploy & Direct) web Graphical User Interface

#### A. Design Phase

The Topology3D offers a web GUI to design a network topology and to configure the services for an experiment (see Figure 11). It consists in a JavaScript client and a Python back-end. A link to a public instance of the Topology 3D can be accessed from [6]. The Topology3D is meant to be an extensible framework that can support different *models* of topology and services. A model corresponds to a technological domain to be emulated and is characterized by the set of allowed node types (e.g. routers, switches, end-hosts), link types, service relationships and related constraints. As shown in Figure 12 the Topology3D takes in input a textual description of the model. The model description is used to configure the topology designer page, to enforce the constraints when the user is building the topology and/or during the validation of the topology. So far, we have provided two models: 1) the OSHI topology domain, including two types of OSHI nodes (Core Routers, Provider Edge routers), the Customer Edge routers which are also used as traffic source/sinks and the SDN controllers; 2) a generic layer 2 network with OpenFlow capable switches, end-nodes and SDN controllers. Each model is decomposed in a set of *views*. A view is a perspective of a model, which focuses on some aspects hiding the unnecessary details. For example, the OSHI model is decomposed in 5 views: data plane, control plane and 3 views for the 3 services (IP VLLs, Pseudo Wires and Virtual Switches). In the data plane view, users can design the physical topology in terms of nodes (OSHI CR and

PE, Controllers, and CEs) and links; in the control plane view, users define the association of OSHI nodes with controllers; in the service views users can select the end points of the services.

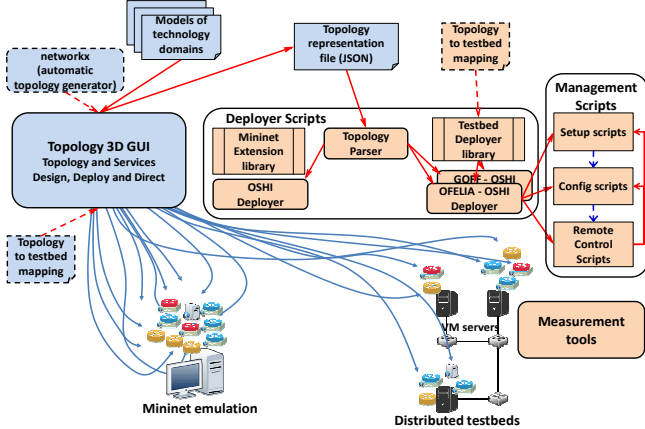


Figure 12. Mantoo enabled emulation workflow

Topology3D integrates Networkx [24], a pre-existing Python package for the creation/manipulation of complex networks, making it also possible to randomly generate a data plane topology with certain given characteristics.

The Topology3D exports the representation of the views (topology and services) in a JSON format, which becomes the input for the deployment phase, as described hereafter.

### B. Deployment phase

The deployment phase translates the designed topology into the set of commands that instantiate and configure the nodes and the services for a given experiment. This phase can target different execution environments for the experiments, by means of a specific “Deployer”. So far, we targeted one emulator (Mininet) and two distributed SDN testbeds (the OFELIA testbed [8] and the GÉANT OpenFlow Facility - GOFF [37]). We are now planning another Deployer that will allow the deployment and the control of experiments over the Amazon Web Services.

Technically, the deployment phase is performed by a set of python scripts (Topology Deployer) that parse the JSON file with the representation of the views and produce further scripts (mostly shell scripts). The proper execution of these scripts deploys the experiment either over Mininet or over a distributed SDN testbed. The Testbed Deployer (see Figure 13) and the Mininet Extensions (see Figure 14) are Python libraries that are used by the actual Deployers. The Mininet Extensions library is tailored for the Mininet emulator, while the Testbed Deployer currently supports the OFELIA and the GOFF testbeds but it can be easily extended to support other testbeds.

#### 1) Mininet Extensions

By default, Mininet only provides the emulation of hosts and switches. We extended Mininet introducing an extended host, capable of running as a router and managed to run the Quagga and OSPFD daemons on it. Then we have added Open vSwitch to it, as needed to realize the OSHI node. Another enhancement to the default Mininet setup depends on our requirement to reach the emulated nodes via SSH from an

external, “non-emulated” process. For this purpose, we introduce a fictitious node in the root namespace of the hosting machine that is connected to the emulated network and works as relay between the emulated world of Mininet and the “real” world of the hosting machine. The details on the specific Mininet deployment architecture can be found in [25]. The Mininet Extensions library is able to automate all the aspects of an experiment. This includes the automatic configuration of IP addresses and of dynamic routing (OSPF daemons) in all nodes, therefore relieving the experimenter from a significant configuration effort. As for the software design, the library extends Mininet providing new objects and API that seamlessly integrate with existing Mininet objects.

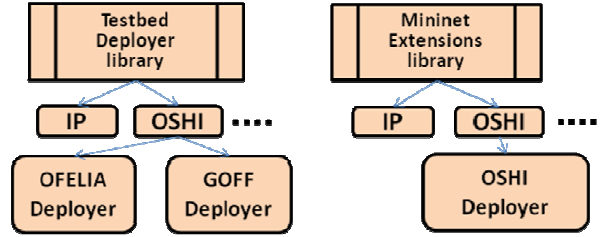


Figure 13. Testbed Deployer

Figure 14. Mininet Extensions

#### 2) Deployment over distributed SDN testbeds

We implemented and tested a Deployer for the OFELIA testbed and one for the GOFF testbed. These two testbeds share a similar architecture as they are based on the OCF (OFELIA Control Framework) [8]. The two testbeds manage differently the out-of-band connectivity. Specifically, in the OFELIA testbed there is a management network with private IP addresses, while in the GOFF testbed all the virtual machines use a public IP address. The OFELIA testbed slice we used is hosted in the CREATE-NET island, composed by 8 OpenFlow capable switches and 3 Virtualization Servers for the experimental Virtual Machines (VMs). The GOFF testbed offers five sites, each one hosting two servers, which respectively run the OF equipment (based on OVS) and Xen [38], for hosting VMs. The GOFF testbed supports all the services (IP VLLs, PW and VSS) that we have described, while due to older Linux kernels it does not support namespaces. For this reason, PW (and VSS) services cannot be deployed on the OFELIA testbed facility.

The Management Scripts automate and facilitate the setup, configuration and the deployment of an experiment. They relieve the experimenter from tedious and error prone activities. As shown in Figure 12, the OFELIA/GOFF Deployer Scripts automatically produce the configuration files that are given in input to the Management Scripts for emulating a given topology, composed of access and core OSHI nodes (OSHI-PE and OSHI-CR) and end points (CEs and SDN controllers). This includes the automatic configuration of IP addresses and of dynamic routing daemons (OSPF) on all nodes, saving a significant time for the node configuration. Each node (CR, PE or CE) is mapped into a different VM running in a Virtualization Server of a given testbed (OFELIA or GOFF). Two mechanisms can be used to map an emulated node on a VM: 1) a resource file (called “topology-to-testbed”) with a list of IP addresses of available VMs can be given to the Deployer, which

automatically chooses the VMs for the emulated nodes; 2) it is possible to manually assign the target VM (identified by its IP address) for an emulated node, either editing a mapping file or graphically using the Topology3D GUI.

A management host coordinates the overall process, usually also executing the Deployer scripts. The management host and the VMs communicate over a management network. The configuration files generated by the Deployers scripts are uploaded on a repository reachable by the VMs (e.g. a webserver running on the management host). During the deployment process these files are downloaded by each VM belonging to the experiment.

The Management Scripts are logically decomposed in Remote Control Scripts, Setup Scripts and Config Scripts:

- The Remote Control Scripts, based on Distributed Shell (DSH), are used by the management host for distributing and executing remote scripts and commands. They enable root login without password, avoid initial ssh paring and configure the DSH in the management VM. Once DSH has been properly configured with the IP of the VMs belonging to the experiment, it can run commands on a single machine, on a subset, or on all the deployed VMs. It is also possible to execute parallel commands speeding up the deployment.
- The Setup Scripts are used to turn a generic VM provided by the testbed into an emulated node (CR, PE, CE or controller). Their main role is to install and configure the software modules needed.
- The Config Scripts are needed to configure a specific experiment and in particular its topology, setting up the link (tunnels) among the VMs.

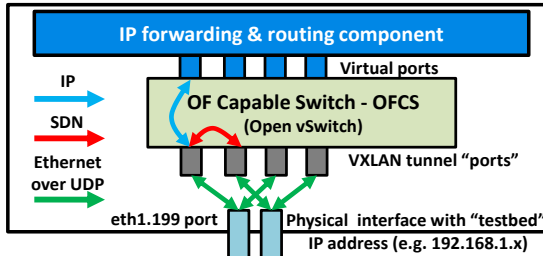


Figure 15. Implementing VXLAN tunnels using Open vSwitch (OVS)

In order to replicate an experimental topology emulating the network links among CRs, PEs and CEs an overlay of Ethernet over UDP tunnels is created among the VMs. In our first design the tunnels were built using the user space OpenVPN tool (with no encryption). The performance was poor, as performing encapsulation in user space is very CPU intensive. Therefore we considered a second design approach based on VXLAN tunnels [29] provided by Open vSwitch. OVS implements VXLAN tunnels in kernel space [28], dramatically improving performance with respect to OpenVPN. The design of the VXLAN tunneling solution for OSHI over a distributed testbed is reported in Figure 15. We only use VXLAN as a point-to-point tunneling mechanism (the VXLAN VNI identifies a single link between two nodes) and we do not need underlying IP multicast support, as in the full VXLAN model. The OF Capable OVS is also able to perform encapsulation and de-capsulation of VXLAN tunnels. Each tunnel corresponds to a port in the switch

### C. Control phase (running the experiments)

In the Mininet based experiments it is possible to open consoles on the emulated nodes using the web GUI of the Topology3D. The consoles show the output generated by the ssh processes connected to the nodes (deployed in the Mininet emulator). The generated output is conveyed to the terminal shell running in the experimenter browser, leveraging the WebSocket API, where each terminal has a separate WebSocket channel. The same functionality for the experiments over the distributed testbeds is currently under development.

### D. Measurement Phase

In order to automate as much as possible the process of running the experiments and collecting the performance data over distributed testbeds we have developed an object oriented multithreaded Python library called Measurement Tools. The library offers an intuitive API that allows the experimenter to “program” his/her tests. Using the library we can remotely (through SSH) run the traffic generators (iperf) and gather load information (CPU utilization) on all nodes (VMs). As for the load monitoring, taking CPU measurements from within the VMs (e.g. using the *top* tool) does not provide reliable measurements. The correct information about the resource usage of each single VM can be gathered from the virtualization environment, for example in a Xen based system we relied on the *xentop* tool, which must be run as root in the hosting Xen server. Therefore, for the OFELIA environment we have developed a python module that collects CPU load information for each VM of our interest in the Xen server using *xentop* and it formats it in a JSON text file. The Measurement Tools retrieve the JSON file from the python module with a simple message exchange on a TCP socket. Instead, in the GOFF environment the measurement data are provided through a Zabbix interface [39]. In order to retrieve the needed information we have realized a python module able to gather the data from the Zabbix API.

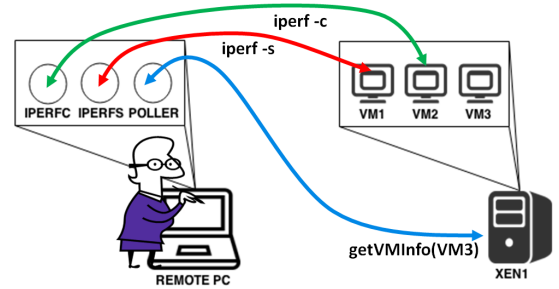


Figure 16. Measurement Tools in action

The Measurement Tools provide a general framework that can be easily adapted to different needs. Currently we have developed tools able to generate UDP traffic and to gather CPU load information from the virtualization environment. An experimenter can easily extend this framework to run his/her tests and collect the measures of interest.

## VI. PERFORMANCE EVALUATION ASPECTS

In this section we analyze some performance aspects of the OSHI solution and experimental platform, describing the results of some experiments. The first three experiments

(sections VI.A, VI.B, VI.C) relied on our first prototype using the VLAN tag based mechanism for the SBPs described in section IV.C and were performed over an Ofelia testbed. We used the iperf tool as traffic source/sink in the CE routers and generate UDP packet flows from 500 to 2500 packet/s. In all the experiments UDP datagram size was 1000 bytes. We preliminarily assessed that packet size has negligible impact on processing load. Using UDP packets ranging from 100 bytes to 1400 bytes, performance has been only influenced by the packet rate. We evaluate the CPU load in the PE routers with our xentop based Measurement Tools. We execute periodic polling and gather the CPU load of the monitored VMs. In each run we collect 20 CPU load samples with polling interval in the order of two seconds: the first 10 samples are discarded and the last 10 are averaged to get a single CPU load value. Then we evaluate the mean and the 95% confidence intervals (reported in the figures) over 20 such runs.

The other experiments (sections VI.D, VI.E) relied on the second prototype using the MPLS tag based mechanism presented in section IV.D and were performed over the GÉANT OpenFlow testbed.

#### A. Best Effort IP performance in OSHI

We consider the design in Figure 2 and compare the forwarding performance of IP Best Effort packets in OSHI (where each packet crosses the Open vSwitch two times, marked as “OSHI IP” in Figure 17) with plain IP forwarding (the Open vSwitch is removed and the OSHI node interfaces are directly connected to IP forwarding engine, marked as “ROUTER IP”). In the next section, we refer to this scenario as “VLAN”, as no tunneling mechanism is used. This experiment is not automatically deployed using the Topology3D and Deployer, and we setup a limited topology with two CE nodes and two OSHI nodes. In the experiment results (see [25] for details) we can appreciate a CPU load penalty for OSHI IP forwarding with respect to ROUTER IP forwarding ranging from 11% to 19% at different rates. The theoretical CPU saturation rate for plain ROUTER IP forwarding is in the order of 14000 p/s. OSHI IP forwarding reduces the theoretical CPU saturation rate to something in the order of 12500 p/s (corresponding to 11% performance penalty).

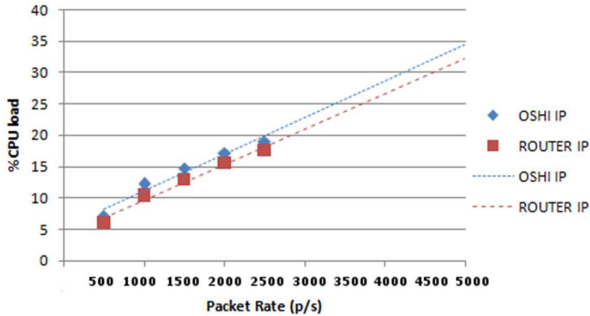


Figure 17. Best Effort IP forwarding performance.

#### B. Performance comparison of tunneling mechanisms

In this experiment we evaluate the processing overhead introduced by the tunneling mechanisms (OpenVPN and VXLAN) used to deploy the overlay experimental topologies

over distributed SDN testbeds. We deployed the topology represented in Figure 19 over the physical testbed topology in Figure 18 (3 OF switches and 3 virtualization servers).

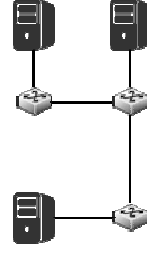


Figure 18. Physical network in the CREATE-NET Ofelia island

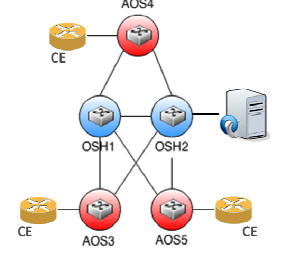


Figure 19. Overlay network for the experiment on Ofelia

Figure 20 compares the CPU load for OSHI IP forwarding in the OpenVPN, VXLAN and plain VLAN (no tunneling) scenarios. It can be appreciated that VXLAN tunneling adds a reasonably low processing overhead, while OpenVPN tunneling would dramatically reduce the forwarding capability of an OSHI node in the testbeds. The theoretical CPU saturation rate for OpenVPN tunneling is in the order of 3500 p/s, which is 4 times lower than in the plain VLAN case. The theoretical CPU saturation rate for VXLAN tunneling is only ~8% lower than the plain VLAN case, showing that VXLAN is an efficient mechanism to deploy overlay topologies.

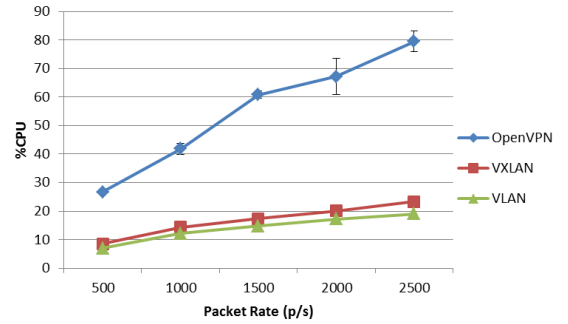


Figure 20. CPU Load for different tunneling mechanisms.

#### C. Performance comparison of different forwarding approaches over the distributed SDN testbed

In this experiment we evaluate the processing load of different forwarding approaches over the distributed SDN testbeds considering the topology shown in Figure 19. For the OSHI solution, we consider IP forwarding (OSHI IP) and SBP forwarding (OSHI VLL). Then we consider plain IP forwarding as a reference (ROUTER IP).

We executed the performance tests of OSHI IP, OSHI VLL and ROUTER IP using the VXLAN tunneling solution. Figure 21 provides reports the experiment results. As shown in Figure 15, in this case the plain IP forwarding (ROUTER IP) has to go through Open vSwitch which handles the VXLAN tunneling in any case, therefore as expected it has no advantage with respect to OSHI IP. The OSHI VLL solution is the least CPU intensive and its theoretical CPU saturation rate is in the order of 13000 p/s. The CPU performance penalty of OSHI IP forwarding w.r.t. OSHI VLL is less than 10%, and its theoretical CPU saturation rate is in the order of 12000 p/s.

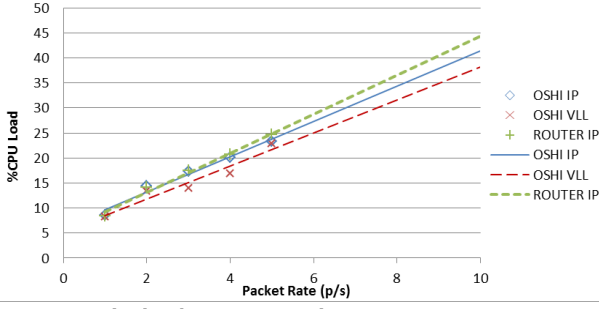


Figure 21. CPU load with VXLAN tunneling.

#### D. Performance evaluation of encapsulation for PW service

In this experiment we evaluate the performance penalty introduced by the encapsulation mechanism implemented for the PW service (section IV.E). We have performed this experiment over the GOFF testbed (physical topology is represented in Figure 22) using the overlay topology shown in Figure 23. As usual, the iperf tool is used as traffic source/sink in the CE routers and generates UDP packet flows. We evaluate the CPU load in the OSHI-PE5, with a periodic polling approach. Each sample is provided by Zabbix every minute, and it is the average calculated in this period with 1-second-interval samples. For each load level (packet rate) we executed a single run of 7 minutes and collect 7 CPU load values, the first 2 are discarded and the last 5 are averaged to get a single CPU mean load value. Then we evaluate the relative standard deviation (RSD) to ascertain the reliability of the results. The RSD is always smaller than 5% in all runs.

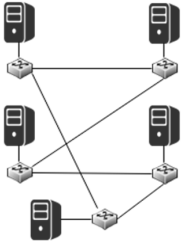


Figure 22. GOFF Physical network

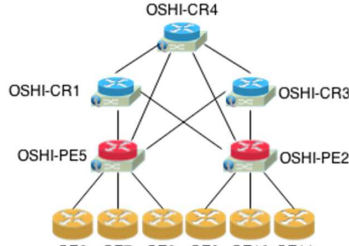


Figure 23. Overlay network for the experiment on GOFF

In the PE nodes, the implementation of the IP VLL service is based on the design shown in Figure 2, while the PW service considers the architecture described in Figure 8. We want to estimate the overhead introduced by the ACE and by the operations of the GRE tunnel. We generate UDP packet flows with a rate ranging from 2000 to 18000 packet/s (datagram size is 1000 byte as usual). The core topology is represented in Figure 23. We have 3 CEs connected to the PEs acting as traffic sources/sinks. This was needed because we can generate at most 6000 packet/s in a single CE, keeping the CPU load of the CE VMs under a safety threshold.

In the experiment results (see Figure 24) we can appreciate a CPU load penalty for OSHI PW forwarding in the order of 15%-21%. Apparently, the CPU load penalty is decreasing in relative terms at higher CPU load. These results show the potential improvements that could be achieved by natively supporting EoMPLS tunneling in the switches instead of using the developed ACE and the GRE encapsulation.

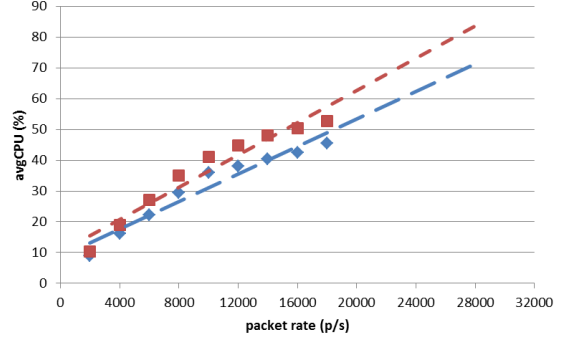


Figure 24. CPU load for different OSHI services.

#### E. Performance analysis of OVS internal mechanisms.

In this section, we shortly report about two experiments that concern the evaluation of OVS internal mechanisms. These experiments do not directly concern OSHI, but they support the choice of OVS as the software based OpenFlow capable switch integrated in OSHI node and show the effectiveness of the proposed Mantoo platform for the setup, deployment and control of the experiments and the collection of performance results. For space reasons, the detailed results have not been included and can be found in [40].

The first experiment investigates the impact of the kernel flow cache implemented in OVS. In the OVS architecture, the first packet of a flow arriving at a node is forwarded to ovs-switchd, running as a Linux user space process. Ovs-switchd processes the packet, forwards it (if a matching rule exists) and finally it installs a “kernel flow”. The kernel flows act like a cache: if another packet, belonging to the same flow, arrives in a short time, it will be forwarded using the kernel flow cache, otherwise it will be forwarded in user space. OVS performance is optimal as long as the packets are forwarded using the kernel flow cache. In our experiment we can force the packets of a set of flows to be handled by the kernel cache or in user space. For the same traffic pattern we measured 40% CPU utilization for kernel cache processing and 94% utilization for user space processing. For the OSHI solution, we gather the design insight that the number of active SBPs should remain within the limit of the kernel flow table. With the hardware used in our experiment (and the default software configuration of OVS), the size of the kernel flow table was 10000 entries. Table 1 allows to evaluate how many VLLs or PWs can be switched in PEs and CRs without recurring to user plane processing (with the exception of the first packet of each flow).

Table 1 Required kernel flow table entries for VLLs and PWs

|                    | PE | CR |
|--------------------|----|----|
| Unidirectional VLL | 2  | 1  |
| Bidirectional VLL  | 4  | 2  |
| Unidirectional PW  | 3  | 1  |
| Bidirectional PW   | 6  | 2  |

For example, an incoming PW in a PE requires 3 entries: one from physical port to ACE, one to cross the ACE and one from the ACE port to the outgoing physical port.

In the second experiment we evaluate how the number of active flows in the flow tables influences the forwarding performance of OVS. In particular we analyze two scenarios:

kernel cache enabled and disabled. The comforting result is that increasing the number of active flows in the tables does not influence the forwarding performance in both cases. This is obviously valid as long as the active flows are less than the size of the tables. The results is a prove of the efficient implementation of flow lookup mechanisms both in the kernel cache and in the user space tables, at least for the traffic patterns that we have used in our experiments.

## VII. CONCLUSIONS

In this paper we presented a novel architecture and implementation of a hybrid IP/SDN (OSHI) node. Developed according to an Open Source model, OSHI enables researchers to investigate novel applications in the emerging hybrid IP/SDN carrier networks scenario. Results of performance tests executed both in single-host emulators (Mininet) and in distributed SDN testbeds demonstrate that OSHI is suitable for large-scale experimentation settings.

The OSHI hybrid node design is based on the combination of an OpenFlow capable switch, Open vSwitch, and of the Quagga routing daemon (using OSPF), running on Linux OS and using plain Linux networking for IP forwarding. OSPF routing is used to establish the basic connectivity among nodes and between nodes and SDN controller (therefore the solution does not need any “out-of-band” connections between the nodes and the SDN controller). The data plane supports the coexistence of best effort IP forwarding and SDN based forwarding over the so-called “SDN Based Paths” - SBPs. The preferred solution for implementing SBPs is using MPLS labels. The traditional distributed MPLS control plane is not needed anymore, as all MPLS circuits (Label Switched Paths, now termed SBP) are established by means of the SDN controller.

As a first service, we implemented layer 2 Pseudo Wires (PW): customer Ethernet frames are encapsulated and carried within SBPs over MPLS. The practical design of this service required the use of GRE tunnels terminated in Provided Edge nodes, because Open vSwitch cannot directly encapsulate Ethernet over MPLS flows: customer Ethernet frames are encapsulated in GRE tunnels and then MPLS is pushed in the GRE packets. On top of the layer 2 PW service we have built a layer 2 Virtual Switch Service (VSS), closely resembling the layer 2 VPLS solution over MPLS. To offer this service, the so called VSS bridging points (see Figure 9) are deployed within OSHI nodes and interconnected with L2 Pseudo Wires to customer ports and among each other. The interconnection topology is optimized using a heuristic approach. We note that our implementation of the VSS closely resemble a Network Function Virtualization (NFV) solution. In fact, we are able to redirect traffic towards arbitrary nodes of the OSHI network using PWs, and we are able to dynamically instantiate a network function (layer 2 switching in our case) to process the redirected traffic. Using the SDN approach, all complex control plane functions that take decisions (e.g. optimal tree evaluation) and enforce that decisions (e.g. creation of PWs) are executed outside the OSHI network nodes.

The planning and execution of relatively large scale experiments on SDN technology or services is a complex

task. Mantoo is a suite of supporting tools for experiments with OSHI based services. It includes an extensible web GUI framework for designing and validating a topology, called Topology3D. The topology is automatically deployed over the chosen emulation platform. Execution and Measurement tools simplify running the experiments and collecting performance measurements.

So far, we presented the OSHI architecture mostly as an experimenter tool: it easily configures VMs as hybrid IP/SDN nodes and performs experiments at relatively large scales using Mininet emulator or resources over distributed testbeds. On the other hand, we the recently started working with OSHI architecture on white box switches [4], in particular using the P-3922 10Gbe switch from Pica8. This work goes into the direction of implementing OSHI in devices that can perform switching and routing at line speed over production networks. Details on these white boxes experiment scenarios and results are available at [6].

## VIII. ACKNOWLEDGMENTS

This work was partly funded by the EU in the context of the DREAMER project [31], one of the beneficiary projects of the GÉANT Open Call research initiative.

## IX. REFERENCES

- [1] “Software-Defined Networking: The New Norm for Networks”, ONF White Paper, April 13, 2012
- [2] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, S. Uhlig, “Software-defined networking: A comprehensive survey”, *Proceedings of the IEEE*, 103(1), 2015
- [3] C. E. Rothenberg, R. Chua, J. Bailey, M. Winter, C. Corrêa, S. Lucena, M. Salvador, “When Open Source Meets Network Control Planes”, *IEEE Computer*, vol.47, no.11, Nov. 2014
- [4] R. Sherwood “Tutorial: White Box/Bare Metal Switches”, Open Networking User Group meeting, New York, May 2014 <http://www.bigswitch.com/sites/default/files/presentations/onug-baremetal-2014-final.pdf>
- [5] S. Salsano, P. L. Ventre, L. Prete, G. Siracusano, M. Gerola, E. Salvadori, “Open Source Hybrid IP/SDN networking (and its emulation on Mininet and on distributed SDN testbeds)”, *EWSDN 2014*, 1-3 September 2014, Budapest, Hungary
- [6] OSHI homepage <http://netgroup.uniroma2.it/OSHI>
- [7] S. Vissicchio et al., “Opportunities and Research Challenges of Hybrid Software Defined Networks”, *ACM SIGCOMM Computer Communications Review*, Editorial Zone (April 2014).
- [8] Marc Suñé et al., “Design and implementation of the OFELIA FP7 facility: The European OpenFlow testbed”, *Computer Networks*, Vol. 61, March 2014
- [9] Mark Berman et al., “GENI: A federated testbed for innovative network experiments”, *Computer Networks*, Vol. 61, March 2014
- [10] Internet2 homepage - <http://www.internet2.edu/>
- [11] Internet2 Software Defined Networking Group homepage <http://www.internet2.edu/communities-groups/advanced-networking-groups/software-defined-networking-group/>
- [12] J. Kempf, et al “OpenFlow MPLS and the open source label switched router”. In *proc. of the 23rd ITC*, 2011
- [13] C. Rothenberg et al. “Revisiting routing control platforms with the eyes and muscles of software-defined networking”, *HotSDN'12 Workshop*, Aug. 2012
- [14] Quagga homepage - <http://www.nongnu.org/quagga/>
- [15] J. Stringer, D. Pemberton, Qiang Fu, C. Lorier, R. Nelson, J. Bailey, C.N.A. Correa, C. E. Rothenberg, “Cardigan: SDN distributed routing fabric going live at an Internet exchange”, *IEEE Symposium on Computers and Communication (ISCC)*, 23-26 June 2014
- [16] P. Lin, J. Hart, U. Krishnaswamy, T. Murakami, M. Kobayashi, A. Al-Shabibi, K.-C. Wang, and J. Bi, “Seamless Interworking of SDN and

- IP" ACM SIGCOMM Computer Communication Review, Vol. 43, No. 4, 2013
- [17] ONOS – Open Network Operating System home page, <http://onosproject.org/>
  - [18] S. Jain, et al, "B4: Experience with a Globally-Deployed Software Defined WAN" in SIGCOMM, 2013
  - [19] S. Vissicchio, L. Vanbever, L. Cittadini, G. Xie and O. Bonaventure, "Safe Update of Hybrid SDN Networks", Technical report, <http://hdl.handle.net/2078.1/134360>
  - [20] C. Filsfils, S. Previdi, (Eds.) et al. "Segment Routing Architecture", draft-ietf-spring-segment-routing-01, Feb 2015
  - [21] E. Rosen et al., "MPLS Label Stack Encoding", IETF RFC 3032
  - [22] S. Bryant, P. Pate, "Pseudo Wire Emulation Edge-to-Edge (PWE3) Architecture", IETF RFC 3985, March 2005
  - [23] L. Martini, E. Rosen, N. El-Aawar, G. Heron, "Encapsulation Methods for Transport of Ethernet over MPLS Networks", IETF RFC 4448, April 2006
  - [24] Networkx homepage - <http://networkx.github.io/>
  - [25] P. L. Ventre et al. "OSHI technical report" available at [6]
  - [26] Floodlight's home page - <http://www.projectfloodlight.org>
  - [27] Open vSwitch home page - <http://openvswitch.org/>
  - [28] J Pettit, E. Lopez, "OpenStack: OVS Deep Dive", Nov 13 2013, <http://openvswitch.org/slides/OpenStack-131107.pdf>
  - [29] M. Mahalingam et al. "VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks", draft-mahalingam-dutt-dcops-vxlan-09.txt, April 10, 2014
  - [30] S. Lowe blog post "Introducing Linux Network Namespaces" <http://blog.scottlowe.org/2013/09/04/introducing-linux-network-namespaces/>
  - [31] DREAMER home page - <http://netgroup.uniroma2.it/DREAMER/>
  - [32] J. Medved, A. McLachlan, D. Meyer, "MPLS-TP Pseudowire Configuration using OpenFlow 1.3" draft-medved-pwe3-of-config-01
  - [33] VPLS, RFC 4761 – <https://tools.ietf.org/html/rfc4761>
  - [34] RYU homepage – <http://osrg.github.io/ryu/>
  - [35] Steiner tree – <http://mathworld.wolfram.com/SteinerTree.html>
  - [36] L. Kou, G. Markowsky, L. Berman, "A Fast Algorithm for Steiner Trees", Acta Informatica, 1981, Volume 15, Issue 2, pp 141-145
  - [37] GOFB homepage – <https://openflow.geant.net/#>
  - [38] The Xen Project homepage <http://www.xenproject.org/>
  - [39] Zabbix homepage - <http://www.zabbix.com/>
  - [40] S. Salsano (ed.) et al. "DREAMER final report", Open Call Deliverable OCG-DS1.1 of GÉANT project, March 2015.