Tunnelless SDN overlay architecture for flow based QoS management

Ignacio Brugnoli, Martín Fernández Bon, Diego Mazzuco,
Gabriel Gómez Sena, Claudina Rattaro and Santiago Bentancur
Facultad de Ingeniería, *Universidad de la República*, Montevideo, Uruguay
{brugnolialem, m.fernandez.bon, diego.mazzuco.ortiz}@gmail.com
{ggomez, crattaro, sbentancur}@fing.edu.uy

***Abstract*—Routing policies determined by Internet Service Providers (ISPs) can create sub-optimal communication paths in terms of Quality of Service (QoS). An Overlay Network (ON) architecture enables the definition of custom routing policies between Points of Presence, enabling QoS metrics improvement for a particular application. This work proposes a forwarding strategy to implement a tunnelless overlay architecture, enabling different traffic flows to follow different paths in order to provide different QoS metrics for each one. Besides, the solution does not affect the packets MTU and can be deployed independently of the underlying ISPs. This article introduces a system architecture built over the software-defined networking paradigm, taking advantage of the centralized view of the network resources and the ability to face challenges by deploying applications at the controller level. The main components of the forwarding strategy have been designed, implemented and tested over both an emulated and a real network to demonstrate its feasibility.**

***Index Terms*—Software Defined Network, Forwarding strategy, ONOS, OpenDayLight, Quality of Service**

I. Introduction

The widely extended use and exponential growth of cloud services in our everyday lives, particularly those focused on on-demand content delivery, have shifted the attention and efforts of many researchers all over the world towards the study of alternative solutions to provide certain levels of Quality of Service (QoS) on the Internet. This has become more than evident with the intense use of network resources during the pandemic due to the Covid-19.

Data flows over the Internet are routed according to policies defined by the Border Gateway Protocol (BGP). However, BGP paths may have sub-optimal characteristics in terms of QoS [1] [2] [3] [4]. For instance, it has been shown that for some origin destination pairs and for some time intervals, alternative paths can be found that will significantly improve (around 20%) the end-to-end delay compared to the standard IP path [5] [6].

Once the best path for a particular traffic flow is found, a forwarding strategy is required to enable the packets belonging to that flow to follow the desired path. Classic solutions for performing traffic engineering to overcome this problem, like ATM or IP/MPLS, are not easily feasible on inter-domain scenarios because they become complex and expensive in terms of management, especially due to the intervention of multiple ISPs [7]. A solution applicable to a multi-domain environment is the use of tunnels, but this technique affects the MTU size and thus it may produce packet fragmentation, which may also affect the traffic QoS. Moreover, this represents an expensive and complex solution in terms of tunnel management. On the other hand, Overlay Networks (ON) have been deployed over the years to address the previous problems as well as for privacy, or to be able to apply particular strategies for content distribution. By using an ON, custom routing policies can be applied in order to forward the traffic through the paths with better QoS. Considering the growth of Software Defined Networking (SDN) [8] deployments, new solutions for the stated problems can be developed.

Based on the centralized view of the network provided by the SDN paradigm together with the ON paradigm opportunities, our proposal offers a non tunnel solution for flow based QoS management over an ON. The solution is indeed independent of the collaboration of the underlying ISPs and does not alter the MTU with additional headers. The main contributions of this work are the proposal, design, implementation and validation of a traffic forwarding strategy based on the SDN paradigm. Some preliminary results were published in our previous articles [9]–[11]. In this paper we present a more general proposal including the implementation of a complete application, which has been developed and thoroughly tested for the Open Network Operating System (ONOS) controller [12], that provides a complete solution to the stated problem [13]. As a further contribution of this article, we perform a complete validation of the proposal (improving our preliminary tests presented in [11]) including more diversity of commercial switches and tests with the OpenDayLight (ODL) SDN controller.

The rest of this paper is organized as follows. The following Section discusses related work. Section III focuses on the designed ON architecture, and in Section IV the proposed forwarding strategy is analysed in depth. In Section V the main implementation aspects are addressed as well as the principal challenges faced. In Section VI we validate our proposal in emulated and real scenarios. Finally, Section VII presents the main conclusions of this work.

This work has been supported by the Agencia Nacional de Investigación e Innovación (ANII), Uruguay, Project FMV_1_2017_1_135526, "Routing and metrology in Overlay Networks using the Software Defined Network paradigm".

II. Related work

Performing traffic engineering by applying custom routing policies on an ON has been proposed by various authors. Three of the most relevant works in this area are [2] [1] and [14]. The authors of [2] present an ON which uses IP-in-IP encapsulation. The overlay proposed in [1] is based on overlay servers which add a custom overlay header. Similarly, the approach proposed in [14] also uses IP-in-IP encapsulation between dedicated overlay servers. Naturally, any kind of tunneling technology to deploy an ON can provide the ability to perform traffic engineering, however, our approach consists on a tunnelless ON architecture avoiding any extra headers or encapsulation technique. Moreover, the solution enables a fine grain flow based QoS management. Our proposal is feasible by making use of the SDN technology, which enables a centralized control of the network traffic.

It is also relevant to mention that commercial products within the category SD-WAN, WAN Optimization, hybrid WAN [15], have been deployed to address some of the problems exposed in this article. As there is no industry standard definition for SD-WAN, most solutions involve the use of proprietary devices installed in each point of presence and connected to a controller [16]. The main difference of our proposal with SD-WAN is that our packet forwarding method can be implemented using standard features provided by OpenFlow. Also, whereas most SD-WAN solutions are based on some kind of tunnelling technology (IPSec, MPLS, IP-in-IP, GRE), our solution does not require tunnels.

III. System architecture

The proposed system architecture is based on our previous work [10] and it relies on the SDN approach. Two main components are involved in the system: the *Central Group*, integrated by all the instances of the SDN controller, and the *Distributed Group*, integrated by all the Points of Presence (PoPs) on the ON (see Fig. 1).

A. Central group

The *Central Group* consists of a SDN controller, logically seen as a single instance with at least one public IP address, but probably composed by multiple components for security and high availability concerns. It includes four applications:

- Traffic Engineering Application (TEApp): It is responsible for deciding the forwarding policies between *PoPs*, based on the QoS metrics of the available paths.
- Monitoring Application (MonApp): It is responsible for deciding the paths which are to be measured and the moment at which each measurement is to be executed. This application provides the information required for the decisions of *TEApp*.
- Routing Application (RouteApp): It is responsible for managing the data flows and allowing the configuration of forwarding rules in the involved SDN switches of the ON. The forwarding policies are determined by *TEApp* routing decisions and are deployed using the functionalities provided by the SDN controller.

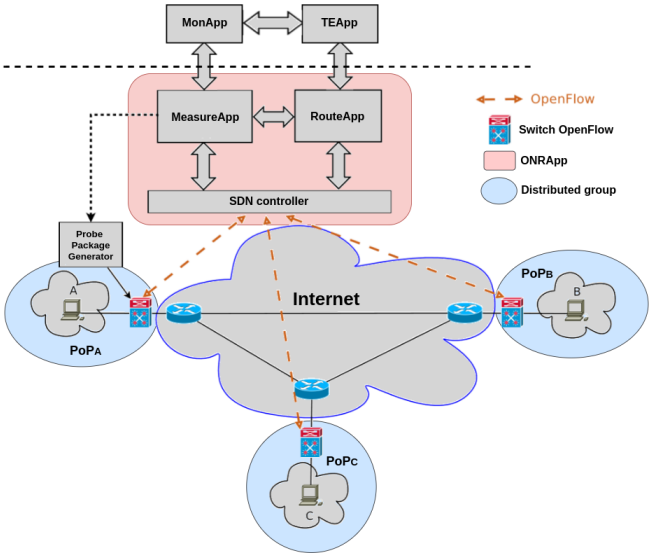


Fig. 1: General system architecture [10].

- Measurement Application (MeasureApp): It is responsible for managing the QoS measurement system, allowing to obtain metrics for a selected path on the ON. This application selects a path on the ON using *RouteApp* services and also manages custom devices called Probe Packet Generators (*PPGs*) to perform the measures .

This paper focuses mainly on the ON implementation and the *RouteApp* details. The main ideas behind *MonApp* and *TEApp* are presented in [6] and [5].

B. Distributed group

The *Distributed Group* includes all the *PoPs* belonging to the ON whose traffic is desired to be controlled. Any *PoP* has the simplified architecture shown in Fig. 2, and includes the following components:

- An OpenFlow Switch to modify and forward the traffic according to the rules provided by the *Central Group*.
- A Probe Package Generator device (*PPG*) capable of generating specific traffic patterns so the desired QoS metrics (delay, throughput, jitter) can be obtained for a specific path.
- The client's LAN which includes the devices that generate the traffic to be controlled.
- The ISP router, for simplicity it is supposed to be the unique Internet access gateway of the *PoP*.

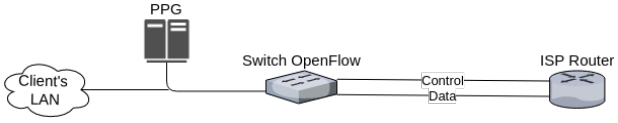


Fig. 2: Internal architecture of a *PoP*.

As shown in Fig. 2, the OpenFlow Switch has two links associated with the ISP router with two public IP addresses. Whereas one of the IP addresses is used for establishing the

control channel with the SDN controller, the other one is needed to implement the forwarding strategy. The in-depth explanation of why a second IP address must be assigned to the OpenFlow Switch is addressed later in Section IV. It is important to notice that both the *PPG* and the client's LAN hosts, need to be "behind" the OpenFlow Switch with respect to the ISP router, so that all outgoing traffic can be managed by the OpenFlow Switch, and particularly by the specific custom forwarding policies. In summary, for each of the *PoPs* the proposed system requires a set of three public IP addresses, one is for the control channel of the OpenFlow Switch, another one used by the proposed forwarding strategy, and the last one to be assigned to the *PPG*.

The proposed system architecture allows to choose custom paths on the ON for each specific traffic flow, so that its QoS metrics meet the desired requirements. Using the forwarding strategy described in the next Section, as the selected traffic traverses the OpenFlow Switches, it will be forced to follow the desired path on the ON.

IV. Forwarding Strategy

As explained earlier, the *RouteApp* application is responsible for managing the paths on the ON, enabling the creation and deletion of custom paths, by handling the OpenFlow tables of the involved switches. The traffic will be forwarded without adding any extra headers, not affecting the MTU and moreover, with complete independence of the ISPs.

According to the ON presented in Fig. 3 and based on the SDN paradigm, the main idea is that if the traffic from host h_1 at *LAN1* to host h_3 at *LAN3* is to be forwarded through PoP_2 (s_2) (green path), the packet destination IP address will be changed at s_1 to the s_2 switch IP address. When the packet arrives at s_2, the packet will be bounced and the destination IP address will be changed to the s_3 switch IP address. Finally at s_3 switch, the packet destination IP address will be changed again to the original destination IP address of h_3.

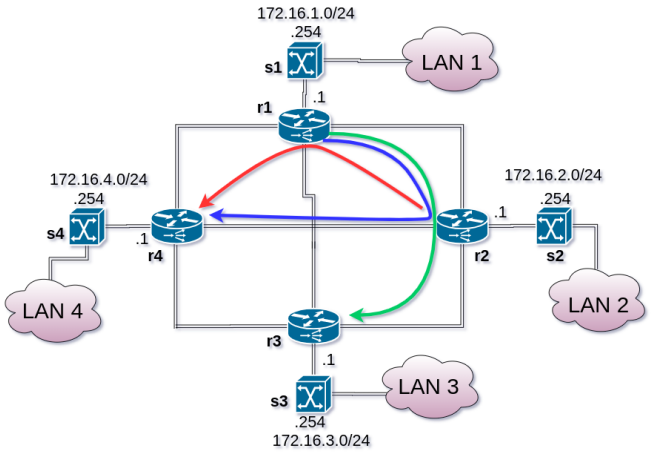


Fig. 3: Four $PoPs$: PoP_1 with (r_1, s_1, *LAN1*), PoP_2 with (r_2, s_2, *LAN2*), PoP_3 with (r_3, s_3, *LAN3*) and PoP_4 with (r_4, s_4, *LAN4*). Three different Paths on the ON (green, red and blue).

Since the ISP router r_2 is highly likely to have "Reverse Path Filtering" enabled, the source IP address of the bounced packet must be also changed to a valid IP address assigned to PoP_2. Thus, the bounce process at PoP_2 will change both the source and destination IP addresses of the packet. The source IP address for bounced packets will be the address assigned to the bounce switch (s_2 in the example).

It is well known that transport (layer 4 or L4 for short) flows are identified by the 5-tuple: {Source IP address, Destination IP address, L4 Source Port, L4 Destination Port, L4 Protocol}, so changing the source or destination IP address on the way would break this identification. Then, based on the above ideas and challenges, the proposed forwarding method uses the following definitions:

- A *Flow* will be identified by:

$$Flow = \{Src\_IP, Dst\_IP, Src\_Port, Dst\_Port, L4\_protocol\}.$$

- A *Link* will be defined by an ordered *PoPs* list: $L_{i,j} = [PoP_i, PoP_j] \quad / \; i, j \in \mathbb{N}$. On a full mesh topology between *PoPs*, it will have a total of $n * (n-1)$ directional Links.
- A *Path* will be the set of *PoPs* the flow will follow from source to destination.
- An Overlay Network Routing Policy (ONRP) composed by a *Path* and the relevant fields to completely identify a flow from source to destination, will be defined by:

$$ONRP = \{ONRP\_Id, Priority, Src\_Subnet, Dst\_Subnet, Src\_Port, Dst\_Port, L4\_Protocol, Path, ONAT\_Id\}. \quad (1)$$

With the *ONRP* definition, complex forwarding policies can be implemented. Matching rules with sub-nets as source or destination, allows generalized forwarding policies to be applied to all the flows originated at or destined to a sub-net. This proposal will also allows the Src_Port and Dst_Port to be wild-carded, so that the *ONRP* can match a set of flows, providing great flexibility. The inclusion of a priority level enables general policies to be overwritten with more specific flow rules, thus enabling complex and fine grained forwarding strategies. Each *ONRP* has a straight correspondence with a set of consistent flow entries that have to be added to all the OpenFlow switches in the path, thus allowing the selected packets to follow the specified route.

Each *ONRP* contains a path involving m *PoPs*, and thus traversing $m-1$ links, where $m \in \mathbb{N}$. To enable the proper identification of traffic from different *ONRPs*, an identifier number called *Local ONRP_Id* must be provided for each *ONRP* on each Link. Thanks to the centralized SDN approach, the management of those identifiers can be easily performed. This work proposes to carry this *Local ONRP_Id* into the L4 source port field, in order to avoid the need of additional headers, but it should be clear that a method to retrieve the original source port value must be provided. Using L4 ports

to store the *Local ONRP_Id* implies a theoretical limit of 2^{16} *ONRPs* per link, if only one IP address is assigned to each OpenFlow switch. If more than one public IP address is assigned to every OpenFlow switch on the ON, this limit can be increased and thus the scalability of the solution.

TABLE I: Example of four *ONRPs*.

ONRP_Id	1	2	3	4
Src_subnet	172.16.1.8/32	172.16.1.0/24	172.16.2.0/24	172.16.1.0/24
Dst_subnet	172.16.4.10/32	172.16.3.0/28	172.16.4.4/32	172.16.3.0/24
L4_protocol	UDP	*	TCP	TCP
Src port	15194	*	*	*
Dst port	15193	*	80	*
Path	PoP_1,PoP_2, PoP_4	PoP_1,PoP_2, PoP_3	PoP_2,PoP_1, PoP_4	PoP_1,PoP_4, PoP_3
ONAT Id	3	4	5	6

As the presented method needs to change the IP addresses and L4 ports headers, a proper way to retrieve the original values must be provided in order to have a successful end-to-end communication. To address this point, the *ONRP* includes a link to an additional structure called *Overlay Network Address Translation Table (ONAT Table)*. This additional table, will contain *ONAT* entries with the original IP addresses and L4 ports of the flows matching the *ONRP*. The identifier of these entries, called *ONAT_Id*, is proposed to be carried on the way using the packet destination port header.

The complete specification of the proposed method should be clearer with the following examples. Suppose that someone wants to implement a routing policy over an ON, with particular rules for the three colored paths shown in Fig. 3. An example of the required *ONRPs* making use of wild-card fields can be shown in Table I. In this example, when the host with IP address 172.16.1.8 at PoP_1 sends a UDP message matching *ONRP #1* in Table I, the flow entries at all OpenFlow switches in the path will change the packet fields as shown in Table II. This means that the involved IP addresses at the intermediate links are the ones assigned to the OpenFlow switches (all ending in .254 in the example), the source ports will store the *Local ONRP_Ids*, and the destination ports will store the *ONAT_Ids*, thus enabling the retrieval of the original packet fields at the destination OpenFlow switch (Table III). It should be clear that the mechanism is transparent to the end hosts because all the relevant packet headers are preserved at source and destination hosts.

A more complex example using *ONRP #2* will show the strength of the proposed method. Suppose two different hosts at PoP_1 (172.16.1.8 and 172.16.1.10) want to communicate with the same host 172.16.3.3 at PoP_3. As *ONRP #2* makes use of wild-cards, both flows will match it. Table IV shows both flows following the same path on the ON, thus using

TABLE II: Flow matching *ONRP #1* at every Link.

	Src Host	L1,2	L2,4	Dst Host
Src IP	172.16.1.8	172.16.1.254	172.16.2.254	172.16.1.8
Dst IP	172.16.4.10	172.16.2.254	172.16.4.254	172.16.4.10
Protocol	UDP	UDP	UDP	UDP
Src port	15194	2	1	15194
Dst port	15193	53	53	15193

TABLE III: *ONAT* Table #3, entry #53 for the example flow.

ONAT_Id	Src IP	Dst IP	Proto	Src port	Dst port
53	172.16.1.8	172.16.4.10	UDP	15194	15193

the same *Local ONRP_Ids*. The original packet fields are retrieved at the destination OpenFlow switch, looking at the corresponding *ONAT* entry whose identifier is carried at the destination port header, as shown in Table V.

In summary, the proposed forwarding strategy can be seen as a kind of NAPT (Network Address/Port Translation) solution, but involving both source and destination addresses translation. The solution allows also both global and fine grained routing policies, and thanks to the SDN approach, the implementation can be performed with applications and modules centrally deployed at the *Central Group*.

V. Implementation

The initial implementation approach, which modifies only the source and destination IP addresses on the way was implemented, as a proof of concept, over a Mininet [17] simulated network and using the POX [18] controller (see details in [9] and [10]). This first implementation has allowed us to face some challenges to be addressed, in order to be able to design a more general and scalable solution, the one that is presented in this article.

The SDN controller is undoubtedly one of the fundamental pieces within the SDN architecture, since it is responsible of the control of the entire network. There are many works that focused on the evaluation of the commercially available controllers [19] [20] [21] considering parameters like throughput, latency and scalability. We have chosen two of the best evaluated controllers to implement our new proposal: Open Network Operating System (ONOS) [12] and OpenDayLight (ODL) [22]. In both cases, the communication between the controller and the switches at each *PoP* is performed through the OpenFlow Protocol.

A. ONOS implementation

Considering its features, market adoption, documentation and performance reports [23], ONOS has been chosen to perform a complete implementation of our proposal. An application called *ONRApp* (Overlay Network Routing Application) has been developed with the goal of performing the automatic management of the ON topology, the route creation and management processes, and all the components of the measurement system [13]. This last feature is not described in this work.

The *ONRApp* application creates an abstraction layer between the SDN application plane and the complex process of implementing the forwarding policies at the OpenFlow switches and also other required network functionalities. The new implemented services are exposed by *ONRApp* through a REST API so that external entities such as *TEApp* and *MonApp* can automatize and perform all the required functions. Proper functions to manage the *PoPs* (register, de-register, list, connect), to manage the *ONRPs* (create, modify, delete,

TABLE IV: Two flows following the same Path both matching *ONRP #2*.

	Src Host		L1,2		L2,3		Dst Host	
	Flow 1	Flow 2	Flow 1	Flow 2	Flow 1	Flow 2	Flow 1	Flow 2
Src IP	172.16.1.8	172.16.1.10	172.16.1.254	172.16.1.254	172.16.2.254	172.16.2.254	172.16.1.8	172.16.1.10
Dst IP	172.16.3.3	172.16.3.3	172.16.2.254	172.16.2.254	172.16.3.254	172.16.3.254	172.16.3.3	172.16.3.3
Protocol	TCP	TCP	TCP	TCP	TCP	TCP	TCP	TCP
Src port	43000	49000	2	2	1	1	43000	49000
Dst port	8080	22	1	2	1	2	8080	22

TABLE V: *ONAT* entries for two flows matching *ONRP #2*

ONAT_Id	Src_IP	Dst_IP	Proto	Src_port	Dst_port
1	172.16.1.8	172.16.3.3	UDP	43000	8080
2	172.16.1.10	172.16.3.3	UDP	49000	22

list) and to manage the measurements (init, stop, get, trigger, stop), are provided by the implemented REST API and are available through POST and GET methods using JSON format for sending and receiving parameters.

Any OpenFlow Switch have multiple "Flow Tables", each of them populated with a set of "Flow Entries" [24] and we have made use of this feature. The *ONRApp* application configures the required flow entries associated with a certain *ONRP* by exchanging specific OpenFlow messages with the switches. As stated earlier, the *ONRP* specification allows the use of wild-carded fields, and therefore some traffic may match multiple flow entries. To solve this, a priority must be established for the flow entries in order to ensure the desired matching order, otherwise the order will be undefined [24]. For example, as shown in Table I, packets matching the *ONRP #4* will also match the *ONRP #2*, and therefore the priority will define the final matching order. The detailed algorithm for priority calculation can be found in [25].

As mentioned, the implementation makes intensive use of having multiples flow tables at the OpenFlow switches. The "Flow Table 0" is used to match the traffic handled by the forwarding architecture, that means, all TCP and UDP traffic between our *PoPs*, and also the ARP traffic, as explained later. After matching "Flow Table 0" the traffic will be forwarded to "Flow Table 1 or 2" for further analysis. The "Flow Table 1" is used to match the traffic originated at the *PoP*, that means, the origin switch of a path. The "Flow Table 2" is used to match the traffic at any intermediate switch and also at the final switch of a path. The "Flow Table 3" and onward are used to implement the *ONAT* Tables. It is worth to mention that the design of the forwarding strategy can support any traffic using ports as L4 identifiers with the aditional condition that they can be overwritten by OpenFlow actions. At the time of this writing, the ONOS controller and the OpenFlow protocol have little support for other protocols beyond TCP and UDP.

Following the example of *ONRP #1* shown in Tables I, II and III, the Flow Table 0 at switch s_1 will contain the flow entries for matching the traffic between PoP_1 and PoP_4, as shown in Table VI (only the UDP matching rule is shown, but another one for TCP traffic is also needed). After matching the specified rule in Flow Table 0, the packets will go to the Flow Table 1 (shown in Table VII), where the specific flow packets will be forwarded to the corresponding *ONAT #3* in the example. At the Flow Table 3 an entry will match the packets and the corresponding action will be to change the required fields in order that the packets follows the desired route through PoP_2, as shown in Table VIII. At the intermediate

TABLE VI: Switch $s1$, Flow Table 0 entry for *ONRP #1*.

	Field	Value	Comment
Match	Ethertype	0x0800	IPv4
	Src IP	172.16.1.0/24	PoP_1 network
	Dst IP	172.16.4.0/24	PoP_4 network
	Protocol	0x11	UDP
	Action	**Value**	**Comment**
Action	Go-to Table	1	ONRPs matching

TABLE VII: Switch $s1$, Flow Table 1 entry for *ONRP #1*.

	Field	Value	Comment
Match	Ethertype	0x0800	IPv4
	Src IP	172.16.1.8/32	PoP_1 host
	Dst IP	172.16.4.10/32	PoP_4 host
	Protocol	0x11	UDP
	Src Port	15194	
	Dst Port	15193	
	Action	**Value**	**Comment**
Action	Write-Metadata	1	Global ONRP_Id
	Go-to Table	3 (example)	Appropiate ONAT

TABLE VIII: Switch $s1$, Flow Table 3 entry for *ONRP #1*.

	Field	Value	Comment
Match	Metadata	1	ONRP#1
	Ethertype	0x0800	IPv4
	Src IP	172.16.1.8/32	Original Src Address
	Dst IP	172.16.4.10/32	Original Dst Address
	Protocol	0x11	UDP
	Src Port	15194	Original Src Port
	Dst Port	15193	Original Dst Port
	Action	**Value**	**Comment**
Action	Set Ethertype	0x0800	IPv4
	Set Src MAC	OF Switch MAC	
	Set Dst MAC	ISP Router MAC	
	Set Src IP	172.16.1.253/32	Switch s_1 Address
	Set Dst IP	172.16.2.253/32	Switch s_2 Address
	Set Src Port	2	ONRP_Id for link 1-2
	Set Dst Port	53	ONAT_Id 53
	Send to port	ALL	All interfaces

OpenFlow switches, similar flow entries should be installed so as the arriving traffic belonging to the *ONRP #1* follows the selected path on the ON. Finally at the end point switch, using a flow entry similar to the one specified in Table VIII, the packets will be rewritten with its original values stored at *ONAT #53* in the example.

All the needed address rewriting for proper traffic forwarding (IPs, ports, and MACs addresses), are performed by using standard OpenFlow actions like SET_DL_SRC, SET_DL_DST, SET_NW_SRC, SET_NW_DST, SET_TP_SRC, SET_TP_DST.

During the system's implementation, some optimisations have arisen being the most relevant the ones focused on reducing the cost of the matching processing at the controller level (a), and at the OpenFlow switch pipeline (b).

In the first case (a), it is clear that when a packet arrives at an OpenFlow switch and is sent to the controller as an PACKET_IN, the matching of the packet is needed to be done twice in order to decide the *ONRP* to which it belonged: once at the switch and again at the controller level. In order to avoid the cost of this duplication, the *ONRP_id* is configured in the PACKET_IN's COOKIE field. By doing this, at the controller level, the application *ONRApp* becomes able to recover this information and use it directly without repeating the matching process.

In the second case (b), when a packet arrives at an OpenFlow switch, is matched and the associated *ONRP* is obtained, if the packet needs to proceed further to the flow tables associated with the *ONATs*, it would be required to repeat the *ONRP* matching. In order to avoid this, once the *ONRP* is obtained, the *ONRP_id* is configured in the METADATA field, as shown in Table VIII. This enables the *ONAT's* flow tables to match only against this identifier, making this flow entries simpler. Both optimisations imply a great improvement of the processing cost and enable the system to be much more scalable.

Regarding other relevant *ONRApp* features, every OpenFlow Switch at the ON must have a dedicated IP address to implement the forwarding strategy. Therefore, each OpenFlow switch needs to handle the required ARP protocol messages to be able to dialog with the corresponding ISP router. To solve this requirement a specific module has been implemented to handle the ARP requests and replies by implementing a global ARP table for the whole ON [25].

The complete solution has been thoroughly tested including functional and performance tests, as reported in [25] (ONOS version 2.1.0). Many other challenges have been faced during the design and implementation but it is important to mention that the ONOS controller has been a good choice, because it provides a great framework with enough documentation and community support.

B. OpenDayLight implementation

We have not developed a complete application over this controller so far, but by enabling some ODL features we have managed to configure the OpenFlow switches with the desired static forwarding rules in order to validate the proposal (ODL version 0.4.4-Beryllium-SR4). More precisely the rules have been configured using the REST API provided by "odl-OpenFlowplugin-all" and "odl-restconf" features [26] [22]. We have also added the "odl-l2switch-all" feature in order to support the classic L2 switch forwarding.

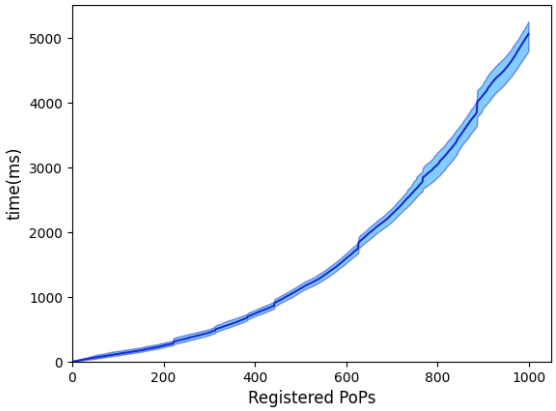


Fig. 4: Performance: Creation PoPs Test.

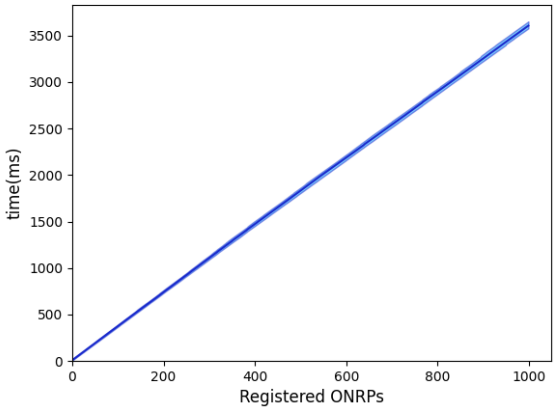


Fig. 5: Performance: Creation ONRPs Test.

The REST API runs over HTTP and accepts requests in both JSON or XML format, and we have used the former for the current implementation. Particularly, this API has been found very adequate to statically manage the switches flow entries.

VI. Evaluation

A. Simulated environment

In order to evaluate the proposed forwarding strategy a Mininet environment [17] has been deployed for both ONOS and ODL controllers. With both controllers we have performed the functional validation of the forwarding strategy and the ON implementation. Furthermore for the ONOS application developed, we have been able to evaluate the scalability and performance of the solution. To show the scalability of the ONRApp application, two tests are performed that measure the time it takes the application from the moment the request arrives until it is sent to ONOS to install the corresponding flow inputs. The first test measures the time it takes to register a certain amount of *PoPs* and the second to create *ONRPs*. Both test are repeated 100 times and we have plotted the 10th and 90th percentiles and the median. These tests can

be found and can be replicated in the Gitlab repository [27]. The topology for the tests, which is represented in Fig. 3, is implemented on Mininet with a direct link between the ONOS controller and each of the OpenFlow switches. In the first test (Fig. 4) an exponential graph is obtained because each time a *PoP* is registered, all the links to the previous *PoPs* need also to be registered, giving a total $n(n-1)/2$ links for each n *PoPs*. Secondly, the other test (Fig. 5) gives a linear value since the consumption per *ONRP* is constant and does not depend on how many *ONRPs* have been registered before. Lastly, the results of both tests give an average of 5 ms per *PoP* when adding add 1000 *PoPs*, and 3.5 ms per *ONRP*. Considering that a new *PoP* registration is a transient event when a new point is added to the network, and that *ONRP* times are constant in the order of milliseconds, it seems that these values are good enough to allow the use of the application. For a real deployment, the internal ONOS delay to communicate with the OpenFlow switches must be considered, but this delay is intrinsic to the network topology and hardware deployed. Tests have been performed in a PC with the following characteristics: Dell Inspiron 5570, Intel Core i7-8550U CPU @1.80GHz, 8GB DDR4 ram @2400MHz and a ST1000LM035-1RK1 hard disk.

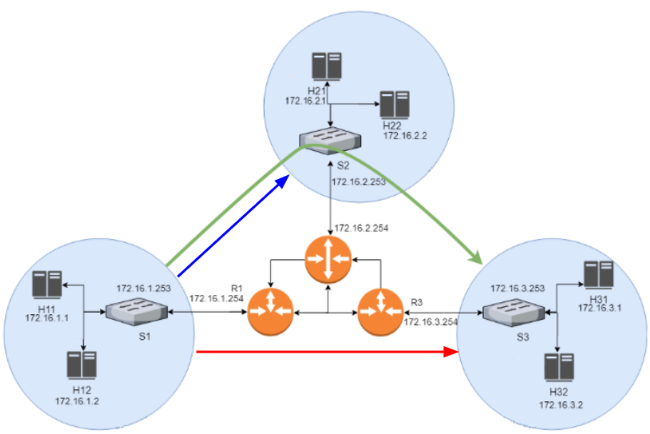
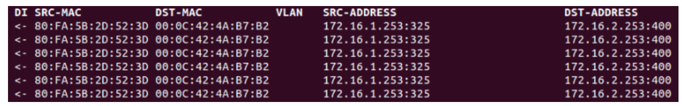


Fig. 6: Implemented Real Overlay topology.

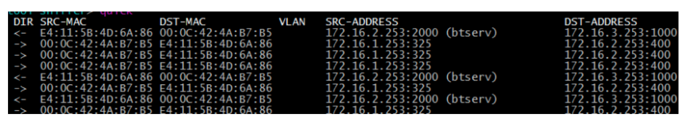
B. Real environment

To validate the ON architecture and forwarding algorithm in a real scenario, a testbed with commercial switches has been deployed with some tests previously reported in [11]. Fig. 6 depicts the main components of our real scenario described below.

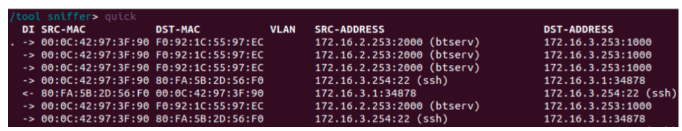
1) The Internet network is implemented by a full mesh of three Mikrotik RouterBOARD 433AH ($R1$, $R2$ and $R3$) which represents the gateway router of the three PoPs.
2) We have made two main deployments with different hardware to implement the $S1$, $S2$ and $S3$ switches.
   - For the first one we use a Pica8 switch [28] (model P3297, PicOS version 2.6.4) supporting OpenFlow 1.3. Three bridges are created, so as to provide three required switches.
   - For the second deployment three TP-LINK TL-WDR4300 (Hardware version: 1.6, FLASH 8 MB, RAM 128 MB) are used. The stock firmware was replaced with OpenWRT [29] version 19.07.2 and the package OpenVSwitch [30] version 2.11.1 was installed in order to support OpenFlow 1.3.
3) The two controllers ODL and ONOS runs on dedicated PCs for simplicity. We use both ONOS and ODL when implementing the OpenFlow switches with Pica8 and only ODL for the TP-LINK switches.
4) At each PoP internal network we placed a linux laptop representing a client hosts ($H11$, $H21$, $H31$).



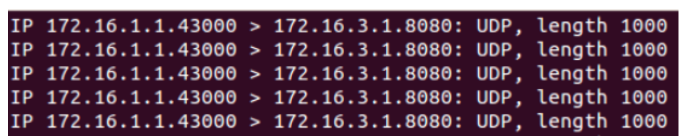
(a) Traffic capture in R1. Traffic modified by S1.



(b) Traffic capture in R2. Traffic from R2 to S2 and vice-verse.



(c) Traffic capture in R3. Traffic from R3 to S3, before S3 applies the corresponding actions.



(d) Traffic capture in H31.

Fig. 7: Traffic captures at each link

The first deployment with the Pica8 switch work as expected and is reported in [11]. For the second one, with TP-LINK switches, we use only the ODL controller with static flow entries. To test the system, a UDP flow from $H11$ to $H31$ port 8080 is forwarded through PoP_2, changing the IP addresses and transport ports on the way as shown in Table IX. The packet captures at $R1$ (Fig. 7a), $R2$ (Fig. 7b) and $R3$ (Fig. 7c) show the modified packets in transit. Finally Fig. 7d shows the packets arriving at final node $H31$.

TABLE IX: Validation: UDP flow from H11:43000 to H31:8080.

	H11 Host	S1 to S2 link	S2 to S3 link	H31 Host
Src IP	172.16.1.1	172.16.1.253	172.16.2.253	172.16.1.1
Dst IP	172.16.3.1	172.16.2.253	172.16.3.253	172.16.3.1
Protocol	UDP	UDP	UDP	UDP
Src port	43000	325	2000	43000
Dst port	8080	400	1000	8080

VII. Conclusions

The proposed forwarding strategy provides a way to ensure that the selected flows will follow specific paths on the overlay network. Moreover, this can be done without any collaboration of the ISP equipment and also without introducing new extra headers to the packets. We demonstrate that our proposal can be properly implemented over a SDN architecture considering the benefits of a software implementation at a centralized point of the network. More precisely, we have implemented and deployed the solution in a real environment, integrating commercial OpenFlow switches (Pica8 and TP-Links) and considering two of the most popular SDN controllers: ONOS and ODL. Functional and performance tests have been done, obtaining great results. A Java application called *ONRApp* has been developed with the goal of performing the automatic management of the ON topology, the route creation and management processes. It is available in [13].

Some topics are kept open for future work. First of all, we are working on a deployment over a more realistic Internet-scale scenario. We also want to analyse further improvements for an IPv6 scenario, where it is feasible to assign a dedicated range of IPv6 addresses for each ON. It may be interesting to analyse the security implications of the proposed solution, though it relies on the intrinsic security of the components: OpenFlow protocol and switches, controller and the application itself. It would be also interesting to extend *ONRApp* to ODL controller.

References

[1] B. D. Vleeschauwer, F. D. Turck, B. Dhoedt, P. Demeester, M. Wijnants, and W. Lamotte, "End-to-end QoE Optimization Through Overlay Network Deployment," in *Information Networking, 2008. ICOIN 2008.*, Jan 2008.

[2] D. G. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient Overlay Networks," in *18th ACM SOSP*, 2001.

[3] H. Zhang, L. Tang, and J. Li, "Impact of overlay routing on end-to-end delay," in *Proceedings of 15th International Conference on Computer Communications and Networks*, 2006, pp. 435–440.

[4] H. Pucha, Y. Zhang, Z. M. Mao, and Y. C. Hu, "Understanding network delay changes caused by routing events," in *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 73–84. [Online]. Available: https://doi.org/10.1145/1254882.1254891

[5] M. Mouchet, M. Randall, M. Ségneré, I. Amigo, P. Belzarena, O. Brun, B. Prabhu, and S. Vaton, "Scalable Monitoring Heuristics for Improving Network Latency," in *IEEE/IFIP Network Operations and Management Symposium*, Budapest, Hungary, Apr. 2020. [Online]. Available: https://hal.laas.fr/hal-02413636

[6] S. Vaton, O. Brun, M. Mouchet, P. Belzarena, I. Amigo, B. J. Prabhu, and T. Chonavel, "Joint minimization of monitoring cost and delay in overlay networks: Optimal policies with a markovian approach," *Journal of Network and Systems Management*, vol. 27, no. 1, pp. 188–232, Jan 2019. [Online]. Available: https://doi.org/10.1007/s10922-018-9464-1

[7] A. Bahnasse, F. E. Louhab, H. A. Oulahyane, M. Talea, and A. Bakali, "Novel sdn architecture for smart mpls traffic engineering-diffserv aware management," *Future Generation Computer Systems*, vol. 87, pp. 115 – 126, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X17323725

[8] The Open Networking Foundation White papers, "Software-Defined Networking: The New Norm for Networks," The Open Networking Foundation White papers, [Online] https://www.opennetworking.org, Apr. 2012.

[9] P. Belzarena, G. G. Sena, I. Amigo, and S. Vaton, "Sdn-based overlay networks for qos-aware routing," in *Proceedings of the 2016 Workshop on Fostering Latin-American Research in Data Communication Networks*, ser. LANCOMM '16. New York, NY, USA: ACM, 2016, pp. 19–21. [Online]. Available: http://doi.acm.org/10.1145/2940116.2940121

[10] I. Amigo, G. G. Sena, M. Chami, and P. Belzarena, "An sdn-based approach for qos and reliability in overlay networks," in *Network Traffic Measurement and Analysis Conference, TMA 2018, Vienna, Austria, June 26-29, 2018*, 2018, pp. 1–2. [Online]. Available: https://doi.org/10.23919/TMA.2018.8506581

[11] S. Bentancur, M. Fernández, G. Gómez Sena, C. Rattaro, and I. Brugnoli, "Flow-based QoS forwarding strategy: a practical implementation and evaluation," in *The Fourth International workshop on Software Defined Networks and Network Function Virtualization, SDN-NFV2020 (accepted for publication)*, June 30th to July 3rd 2020.

[12] The Open Networking Foundation, "Onos: Github repository." [Online]. Available: https://github.com/opennetworkinglab/onos

[13] I. Brugnoli, M. Fernández, and D. Mazzuco, "Onrapp: Overlay network routing application (git repository)," 2019. [Online]. Available: https://gitlab.fing.edu.uy/tesis/onra/

[14] O. Brun, L. Wang, and E. Gelenbe, "Big Data for Autonomic Intercontinental Overlays," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 3, pp. pp.575 – 583, 2016. [Online]. Available: https://hal.laas.fr/hal-01461990

[15] Z. Yang, Y. Cui, B. Li, Y. Liu, and Y. Xu, "Software-defined wide area network (sd-wan): Architecture, advances and opportunities," in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*, 2019, pp. 1–9.

[16] S. P. Rachuri, A. A. Ansari, D. Tandur, A. A. Kherani, and S. Chouksey, "Network-coded sd-wan in multi-access systems for delay control," in *2019 International Conference on contemporary Computing and Informatics (IC3I)*, 2019, pp. 32–37.

[17] Mininet, "Mininet." [Online]. Available: http://mininet.org/download/

[18] POX, "The pox network software platform." [Online]. Available: https://github.com/noxrepo/pox/

[19] M. Darianian, C. Williamson, and I. Haque, "Experimental evaluation of two openflow controllers," in *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, Oct 2017, pp. 1–6.

[20] L. Mamushiane, A. Lysko, and S. Dlamini, "A comparative evaluation of the performance of popular sdn controllers," in *2018 Wireless Days (WD)*, 2018, pp. 54–59.

[21] S. Secci, A. Diamanti, J. Sánchez, M. Vílchez, M. T. Bah, P. Vizarreta, C. Machuca, S. Scott-Hayward, and D. Smith, "Security and performance comparison of onos and odl controllers," 2019.

[22] OpenDaylight, "Documentation release beryllium." [Online]. Available: https://buildmedia.readthedocs.org/media/pdf/opendaylight/stable-beryllium/opendaylight.pdf

[23] F. Pakzad, "Comparison of software defined networking (sdn) controllers. part 7: Comparison and product rating." [Online]. Available: https://aptira.com/comparison-of-software-defined-networking-sdn-controllers-part-7-comparison-and-product-rating/

[24] The Open Networking Foundation, "Openflow switch specification 1.5.1," 2015. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf

[25] I. Brugnoli, M. Fernández, and D. Mazzuco, "Overlay network routing application (onrapp)," jun 2019. [Online]. Available: https://iie.fing.edu.uy/publicaciones/2019/BFM19

[26] OpenDaylight, "Odl openflowplugin release master." [Online]. Available: https://buildmedia.readthedocs.org/media/pdf/odl-openflowplugin/stable-oxygen/odl-openflowplugin.pdf

[27] I. Brugnoli, M. Fernández, and D. Mazzuco, "Onrapp performance test (git repository)," 2019. [Online]. Available: https://gitlab.fing.edu.uy/diego.mazzuco/TestPerformanceONRApp

[28] Pica8, "Picos support for openflow 1.3." [Online]. Available: https://docs.pica8.com/display/PicOS21116cg/

[29] "Openwrt." [Online]. Available: https://openwrt.org/

[30] Linux foundation collaborative projects, "Open vswitch." [Online]. Available: http://docs.openvswitch.org/en/latest/intro/what-is-ovs/