Towards an SDN Network Control Application for Differentiated Traffic Routing

Davide Adami*, Gianni Antichi†,
Rosario G. Garroppo‡, Stefano Giordano‡, Andrew W. Moore†
*CNIT Research Unit, Pisa
†Computer Lab, University of Cambridge
‡Department of Information Engineering, University of Pisa

***Abstract*—In the last years, Software Defined Networking has emerged as a promising paradigm to foster network innovation and address the issues coming from the ossification of the TCP/IP architecture. The clean separation between control and data plane, the definition of northbound and southbound interfaces are key features of the Software Defined Networking paradigm. Moreover, a centralised control plane allows network operators to deploy advanced control and management strategies. Effective traffic engineering and resources management policies allow to achieve a better utilisation of network resources and improve end-to-end service performance. This paper deals with the architectural design and experimental validation of a control application that enables differentiated routing for traffic flows belonging to different service classes. The new control application makes routing decisions leveraging on OpenFlow network statistics, i.e., taking advantage of real-time network status information. Moreover, a Deep Packet Inspection module has been developed and integrated in the control application to detect VoIP traffic with Session Initiation Protocol signalling, enforcing this way policies for a differentiated treatment of VoIP traffic. Finally, a functional validation is performed in emulated environment.**

***Index Terms*—SDN, VoIP, Differentiated Routing, QoS, SIP**

I. Introduction

Building, running and maintaining enterprise networks is getting more complicated and difficult. Part of the problem is the proliferation of real-time applications (*i.e.*, voice, video, gaming) which demand more and more bandwidth and low-latency connections. Nowadays, there are two main approaches used to enable such applications in enterprise networks: resource over-provisioning or implementation of resource management schemes. While the former oversizes the network enough to meet the expected peak demand with a substantial margin of safety, the latter tries to manage the available bandwidth through traffic management and classification techniques. The problems related to these two approaches are, respectively, the waste of resources and the high complexity/costs that derive from the use of high–speed and reliable middle-boxes. In this scenario, Software Defined Networking (SDN) with OpenFlow (OF) represents a third approach: it provides both framework and tools to enable symbiotic linkage between network paths and user applications. SDN is an emerging network architecture where network control is decoupled from data plane and is directly programmable. Northbound and southbound interfaces are defined between controller/applications and controller/network devices. OF [1] is the standard communication protocol that allows the controller to access the forwarding plane of a network device. Such a migration of control, formerly tightly bound in individual network devices, into accessible computing devices is not novel: separating inter-domain routing from individual routers using logically centralised control system was proposed by Caesar et al. [2] to make routers more manageable and flexible. In addition, SDN enables the underlying infrastructure to be abstracted for applications and network services, which can treat the network as a logical or virtual entity. Following this new shiny paradigm, the network service provider can better recognise the best path in the physical network infrastructure for a real-time application and provide therefore Quality of Service (QoS). The aim of this work is to take advantage of the SDN paradigm to enforce differentiated routing depending on QoS requirements. The proposed SDN control application includes a Deep Packet Inspection (DPI) module able to detect Session Initiation Protocol (SIP) signalling messages. The SIP messages parser implemented in the DPI module allows to know the parameters of the VoIP traffic flows (*i.e.*, source and destination IP address pairs, and source and destination RTP port pairs) during the setup phase of the call. Such information make it possible to recognise and manage the VoiP traffic flows directly on the data plane, and to apply a differentiated routing strategy with respect to the common best-effort traffic. It is worth noting that the DPI manages only SIP signalling messages, to reduce the scalability problems that can appear when DPI is applied to all traffic. The solution is then validated through an extensive functional tests campaign. The paper presents the related work in section II, while the background on SIP protocol is summarised in section III. Section IV describes the developed architecture, while section V presents experimental results. Section VI concludes the paper.

II. Related Work

To the best of our knowledge a limited amount of effort has been done till now to enable QoS in OF-enabled networks. Egilmez et al. [3] propose a novel OF controller for multimedia delivery with end-to-end QoS support. The idea is to group incoming traffic as data and multimedia flows, where the multimedia flows are dynamically placed on QoS guaranteed routes and the data flows remain on their traditional shortest-path. Unfortunately they differentiated

data flows from multimedia flows using static fields of the packet (*i.e.*, TOS, source IP, port numbers). They do not try to recognise if a new flow is a real multimedia flow or not. Wallner et al. [4] propose another approach to QoS using Floodlight controller. Although they clearly show how to guarantee QoS using the DSCP IP field, it is not clear how to recognise an actual multimedia stream. Jeong et al. [5] propose a QoS-aware network operating system (QNOX), providing QoS-aware virtual network embedding, end-to-end network QoS assessment, and collaborations among control elements in other domain network. The authors try to find the best path according to some user-defined constraints such as packet-loss, end-to-end delay, etc. Finally, Ishimori et al. [6] propose a framework to enhance QoS management procedures in OF networks. Unfortunately such an idea would require an OF switch able to handle the new proposed primitives, while our solution is compatible with the existing ones.

III. The SIP protocol

SIP is an application level protocol that allows to establish, modify and tear down multimedia sessions (*e.g.*, voice calls or video conferencing). The SIP protocol uses URI (Uniform Resource Identifier) schemes to identify users, both location-based services and SIP servers to locate users, enabling this way personal mobility. The protocol is based on the request/response paradigm and is text-based.

To set up a session, the INVITE message is sent by the User Agent Client (UAC) towards the SIP server, which is responsible to locate the next SIP server to forward the message or the User Agent Server (UAS) used by the receiver. When the INVITE message arrives to the UAS, if the receiver accepts to participate in the session, it sends a response message (usually 200 OK) towards the SIP server, which will forward it towards the UAC through one or more intermediate SIP servers.

Both the INVITE and the 200 OK messages contain in their "body" field information on the media and related parameters (*i.e.*, media type, codec, codec configuration parameters, transport protocol used for the media, etc.). The format used to communicate the media parameters is defined in the set of rules for describing multimedia sessions, known as Session description Protocol (SDP, [7]). The SDP allows to describe a series of basic features for the proper exchange of information during a session, such as:

- the IP address of the host that will receive the traffic of a particular media involved in the session;
- the transport protocol (TCP or UDP) and its port number;
- the announced media type (video, audio, etc.), the type of encoding used for the media (*e.g.*, H.261 or MPEG for video, G.729 or G.711 for audio) and configuration parameters (*e.g.*, the sample period for the audio, the activation or not of the Voice Activity Detection algorithms, etc.)

SDP can also carry information as the name and purpose of the session, the temporal characteristics of the session, the starting and closure time, or the transmission capacity required.

SDP description consists of a series of text lines each one with a structure *type* = *value*. In particular, type is always specified by a single character, conversely value is a string with a format that depends on the type. The example shown in figure 1 describes a session where the host generating the INVITE message communicates that it wants to receive the media at the IP address 10.0.0.3 (the *c* type), and it is able to receive audio on the port 3000 with the RTP protocol and with encoding 0 and 8[1] (the *m* type). This example shows that the media session can be easily identified analysing the SDP information carried out by the SIP messages, the IP destination address, transport protocol and destination port of the traffic flow associated to one direction.

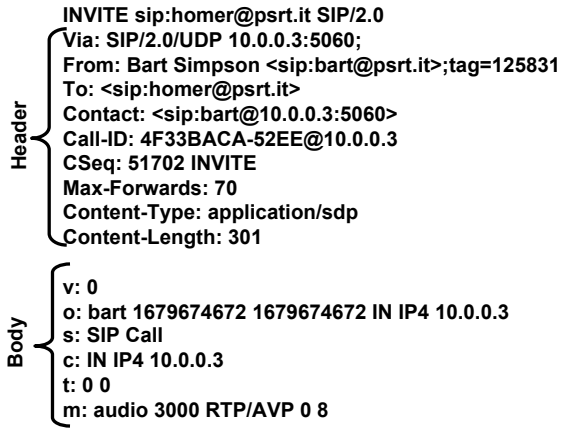

Fig. 1. An example of INVITE message.

IV. Controller Architecture

This section describes the architecture of the proposed control application (see figure 2), which has been developed leveraging on POX, a platform for the rapid development and prototyping of network control software using Python. The modules communicate through a publish/subscribe event manager system that comes from the core of the POX [8] controller. Further details on the modules are provided in the following.

A. `Topology Discovery` module

The `openflow.discovery` provided within the Betta branch of the POX repository and `host_tracker` are the main components needed to keep updated information about hosts, switches and links in the network. The first one is the key element to perform the network discovery. It is in charge of sending Link Layer Discovery Protocol (LLDP) [9] packets to all the connected switches through packet_out messages. These messages instruct the switches to send LLDP packets out to all of their ports. Once a switch receives the packet_out

[1]the association between these numbers and the codecs can be found in http://www.iana.org/assignments/rtp-parameters

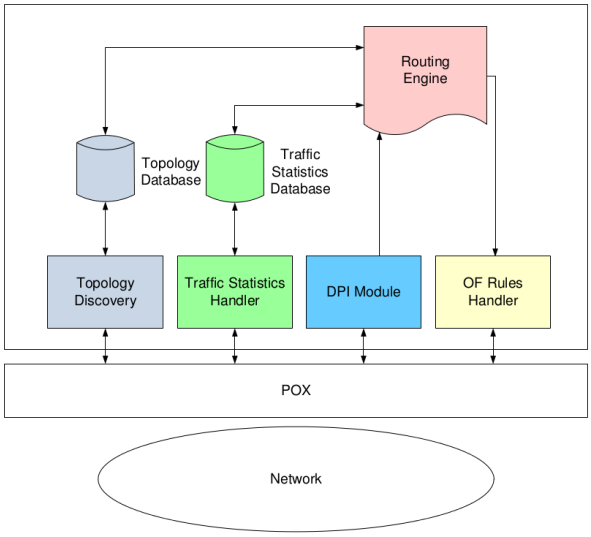

Fig. 2. The Architecture of the Developed Control Application.

message, it sends the LLDP packets out over all its ports to other connected devices. If the neighbour device is an OF switch, it will perform a flow lookup. Since the switch does not have a flow entry for this LLDP message, it will send this packet to the controller via a packet_in message. When the controller receives the packet_in, it analyses the packet and creates a connection in its discovery table for the two switches. All the remaining switches in the network will similarly send a packet_in to the controller, which creates a complete network topology. LLDP messages are periodically exchanged and events are raised at the controller when links go up/down or new links are added/removed. Information on switches and links are maintained in the Network database. The host_tracker keeps track of the hosts in the network (*i.e.*, where they are and how they are configured - at least their MAC/IP addresses). When a change occurs, the component raises a specific event. In short, host_tracker works by examining packet_in messages, and learning MAC and IP addresses. The controller periodically ARP-ping hosts to see if they are still alive. The data collected are stored in a soft-state manner inside the Hosts database, containing one-to-one mappings between each host (textiti.e., IP and MAC addresses) and the switch is connected to (textiti.e., datapath ID and port number). The Topology database is obtained joining the Network database with the Hosts database.

B. `Traffic Statistics Handler` module

The `Traffic Statistics Handler` (TSH) module gathers ports and flows counters (statistics) from each OF switch. This information is stored in the Traffic Statistics Database. In particular, TSH periodically sends to each switch an OF message asking for Received Bytes (Rx) and Transmitted Bytes (Tx) counters at each port and keeps track of the time when responses are received. We indicate with T_1 and T_2 the time two consecutive responses are received and with LTL the Link Traffic Load in bps for each switch interface. TSH estimates the input and output LTL as follows:

$$LTL_{IN} = \frac{(Rx(T_2) - Rx(T_1)) * 8}{T_2 - T_1} \quad (1)$$

$$LTL_{OUT} = \frac{(Tx(T_2) - Tx(T_1)) * 8}{T_2 - T_1} \quad (2)$$

where LTL_{IN} and LTL_{OUT} are computed with the counters from one of the two interfaces connected to the same link, chosen randomly. This allows to reduce the amount of traffic related to the collection of network statistics, since only the traffic statistics acquired on one of the two link end-points are necessary. Basically, the TSH module allows to associate different costs to each link direction, thus enabling asymmetric routing. However, to avoid unstable behaviours due to sudden spikes of traffic, the LTL is filtered by means of a first-order low-pass filter. The output of the filter, denoted as Smoothed LTL (SLTL), is used by the `Routing Engine` (RE) module to establish the costs of the links. The configuration parameters of the TSH module (*e.g.*, the requests time period and the filter coefficients) should be chosen by the network administrator taking into account traffic dynamics and expected network responsiveness. The analysis of the control application with different configuration parameters is out of the scope of this paper for the sake of brevity.

C. `Routing Engine` module

The `Routing Engine` (RE) module receives information about the topology from TD, and SLTL values for each link direction from TSH. If no SLTL information is received, RE calculates the shortest path tree from each source node to all possible destinations applying the Dijkstra algorithm to a graph with default costs assigned to the edges representing the network links. On the other hand, if SLTL information is provided, RE builds two graphs: the first one for the privileged traffic class (e.g., VoIP traffic) and the other one for all the remaining traffic (e.g., best effort traffic). These graphs differ for the costs assigned to the network links, which depend on both the traffic class (i.e., VoIP, best effort) and the link utilisation (LU), calculated as the ratio between the SLTL and the link capacity. It is worth highlighting that research concerning the definition of the cost function shape or other QoS metrics is out of scope. Table I reports the cost values we use in Section V for VoIP and best effort traffic. In the case of VoIP traffic, the cost associated to each link increases proportionally with the link utilisation. On the other hand, in the case of best effort traffic, a high cost value is assigned to link with low link utilisation and such a cost decreases when the link utilisation increases. If the link utilisation is higher than a given threshold (i.e., we set it to 85%), a high link cost is associated for both types of traffic to avoid congestion. Once the weighted directed graphs (one for each traffic class) are calculated, the Dijkstra algorithm is run in order to find the shortest paths for each origin-destination nodes pair.

TABLE I
INTERFACE UTILISATION, TRAFFIC CLASSES AND LINK METRICS

Utilization	VoIP	Best Effort
$< 30\%$	10	80
$< 60\%$	50	30
$< 85\%$	80	10
$> 85\%$	150	150

D. `Deep Packet Inspection` module

The `Deep Packet Inspection` (DPI) module is responsible for traffic classification and therefore for the assignment of each traffic flow to a pre-defined class. In this section we will focus on the architectural principles behind the design of the `DPI` module developed in our prototype for the detection of VoIP and best effort traffic. We point out that the aforementioned principles can be also applied to other kinds of traffic, just changing the DPI policy. According to the standard, when an OF-enabled switch receives a frame that matches no entry in its flow table, it sends to the controller only 128 bytes of the original frame, thus allowing the analysis of the fields in the packet header up to the transport layer. We point out that the controller must receive all the signalling messages exchanged by the proxy servers (*i.e.*, INVITE, 200 OK) in order to have a real-time understanding of the calls status. When the controller receives the frame, the DPI engine checks if the packet contains a SIP message. In our prototype, we considered the case where VoIP sessions are set-up through the exchange of SIP signalling messages carried out within UDP packets with destination port 5060, and content-Type field set to `application/sdp`. Other specific detection rules should be developed to account for VoIP sessions using as control plane SIP over TCP or SIPS, but are out of the scope of this work. It is relevant to observe that the SIP detection strategy is important to reduce the scalability problems that can appear when DPI is applied to all traffic. Indeed, the detection permits to apply the parser rules only to the SIP messages. If an INVITE message is detected, the module analyses the SDP information in the body of the message. At the end of this procedure, the module knows the media channels of the VoIP session (i.e., the source and destination IP addresses, and source and destination UDP ports of the RTP traffic flow) in one direction. The information on the RTP traffic flow in the opposite direction is acquired intercepting the SIP message response. For each new SIP session, the DPI gives to the RE the parameters of the RTP channels. Then, using the graph associated to VoIP traffic, the RE calculates the routes for the new RTP traffic flows and transfers the obtained forwarding rules to the OF Rules Handler. Such a module is in charge of setting up the right rules directly to the OF-enabled switches. The chosen approach permits the controller to intercept the BYE message, used to tear down the SIP session, and as a consequence to remove the forwarding rules installed for the associated RTP traffic flows. If the controller receives a non-SIP message, the RE calculates the shortest path and the corresponding forwarding rules using the graph associated to best effort traffic, and transfers the rules to the OF Rules Handler.

V. EXPERIMENTAL VALIDATION

This section evaluates the effectiveness of our control application that enables the enforcement of differentiated routing policies. More specifically, the new SDN controller is implemented in the Mininet emulation environment [10] and the tests aim to analyse the behaviour of the control application in a typical enterprise network scenario where best effort and VoIP traffic flows are generated. We remark that the test campaign is focused on demonstrating the correct operation of the proposed application. Performances can be highly variable depending on the adopted SDN policy (*i.e.*, re-active vs pro-active) and the type of traffic (*i.e.*, percentage of VoIP traffic).

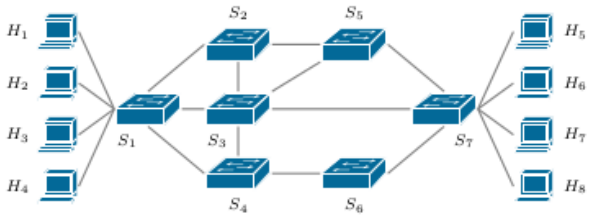

Fig. 3. The network scenario.

The emulated network (see Figure 3) consists of 8 Hosts (*e.g.*, Mininet Virtual Machines) and 7 OF-enabled switches (*e.g.*, Mininet Open vSwitches). Each link of the network is assumed to have a bandwidth of 1 Mbps. To evaluate the ability of our SDN controller to make routing decisions according to link load conditions and traffic classes, we activate and deactivate VoIP sessions and best effort (BE) traffic flows. Figure 4 shows the traffic measured at the output interfaces of the switch S_7. Three BE traffic flows start at different times and last for the whole duration of the experiment. Each flow is characterised by a specific source-destination nodes pair in order to easily highlight the traffic data associated to the flow during the test. Furthermore, each BE flow generates a different traffic load. On the contrary, only one source-destination nodes pair injects VoIP traffic in the network, but during the experiment three different VoIP sessions are set-up and torn down. The traffic associated to the VoIP sessions has been set to 100 Kbps. During the set up phase of the VoIP sessions, the controller calculates the shortest path for the VoIP traffic taking into account the metrics associated to the links for such traffic class. Then, the forwarding rules are set in the switches for data traffic.

Figures 5 and 6 show the values of the cost observed during the emulation for BE and VoIP traffic, respectively. The figures highlight that our control application is able to update links metrics according to the load conditions of the network. Hence, the controller can establish the forwarding paths of new flows taking into account the network status and the traffic classes. Details on the paths selected by the controller, when the new flows start, are reported in the following.

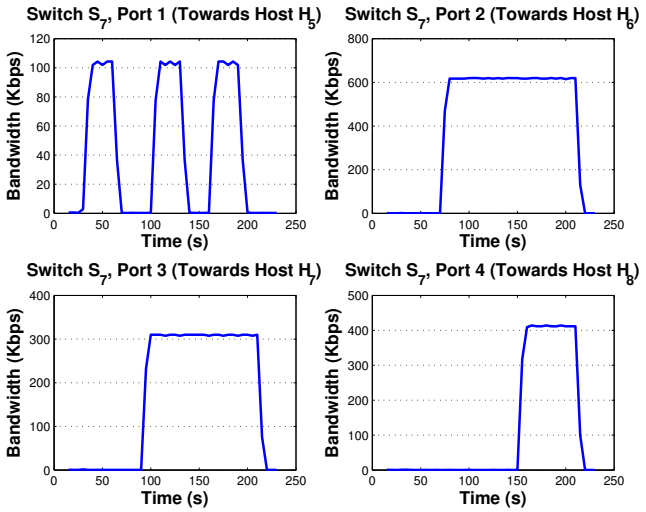

Fig. 4. Traffic load measured at the diverse outports of S_7.

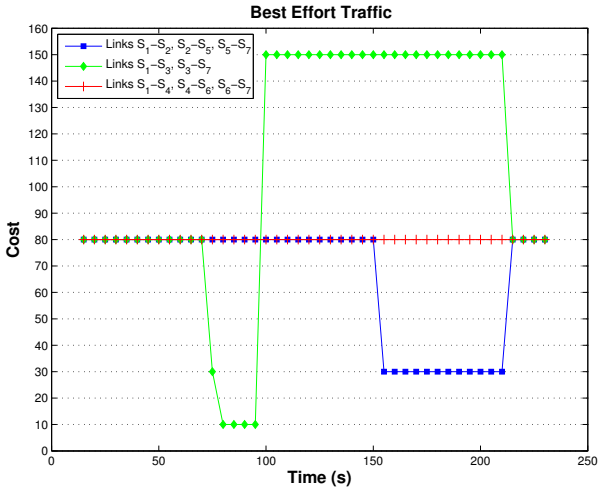

Fig. 5. Observed BE cost values vs. time for each link.

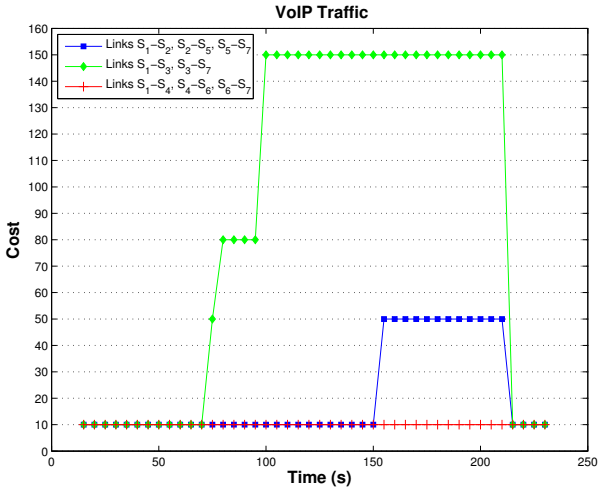

Fig. 6. Observed VoIP cost values vs. time for each link.

At $t = 30$ s, a VoIP session starts between H_1 and H_5. S_1 receives the SIP messages and forwards them to the controller. Since the network is completely unloaded, all links have the same cost, i.e., 80 for BE and 10 for VoIP, as highlighted in Figures 5 and 6. In this case, the forwarding path for VoIP traffic corresponds to the shortest path in terms of number of hops from H_1 to H_5. The path is computed using the Dijkstra algorithm with the network links costs shown in the Figures 5 and 6 at $t = 30$ s. These values are obtained starting from the link utilisation data collected by the OF-enabled switches. Then, the controller installs the forwarding rules in the switches along the path. To verify whether the forwarding rules are correctly installed, Figure 7 reports the utilisation of network links at $t = 50$ s. As expected, the utilisation of links S_1-S_3 and S_3-S_7 is around 10%, whereas all other network links are not utilised.

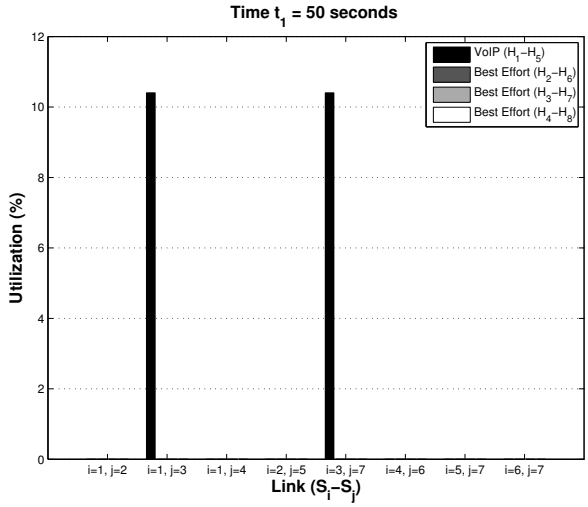

Fig. 7. Links utilisation observed at $t = 50$ s.

At $t = 60$ s, the VoIP session between H_1 and H_5 is closed and the generated VoIP traffic goes to zero.

Then, at $t = 70$ s, a best effort traffic flow (BE1) with rate 600 Kbps starts between H_2 and H_6. Since the network is completely unloaded, again the same cost for each traffic class has been assigned to all links. Hence, also in this case, the controller, using the Dijkstra algorithm, selects the minimum hop count path (*i.e.*, S_1-S_3, S_3-S_7), as the forwarding path to set in the network switches for this flow.

At $t = 90$ s, another best effort traffic flow (BE2) at 300 Kbps is activated between H_3 and H_7. The controller selects again the links S_1-S_3 and S_3-S_7 as forwarding path for BE2.

At $t = 100$ s, a new VoIP session between H_1 and H_5 is started. Since the path S_1-S_3-S_7 is used by BE1 and BE2, the controller executes the Dijkstra algorithm after the cost of each link has been updated according to the values reported in Table I and shown in the Figures 5 and 6 for $t = 90$ s. In this network scenario, the selected forwarding path for the VoIP traffic is S_1-S_2-S_5-S_7. Figure 8 reports the utilisation of network links to show the correct installation of the calculated forwarding

paths In particular, we can observe that the utilisation of the links belonging to the VoIP path is around 10%, whereas the links associated to the BE traffic is around 90% (i.e., the sum of BE1 and BE2).

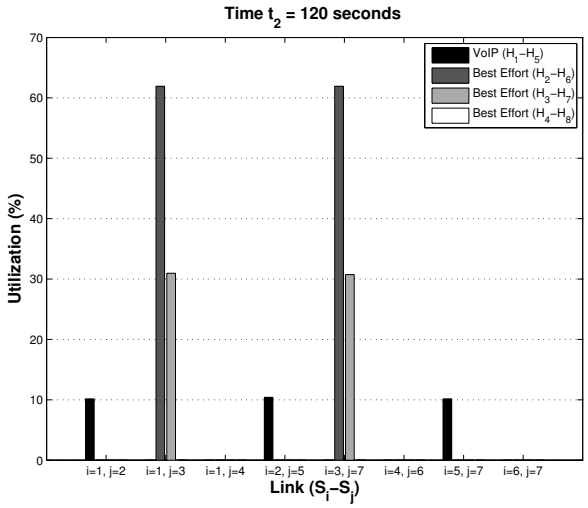

Fig. 8. Links utilisation observed at $t = 120$ s.

At $t = 150$ s, the VoIP session is torn down and a new best effort traffic flow (BE3) at 400 Kbps is started between H_4 and H_8. Since the utilisation of the links S_1-S_3 and S_3-S_7 is more than 90%, the cost assigned to such links is 150. In this scenario, the Dijkstra algorithm gives S_1-S_2-S_5-S_7 as the forwarding path for BE3. Then, the controller correctly installs the forwarding rules for this traffic flow in the switches belonging to the calculated forwarding path.

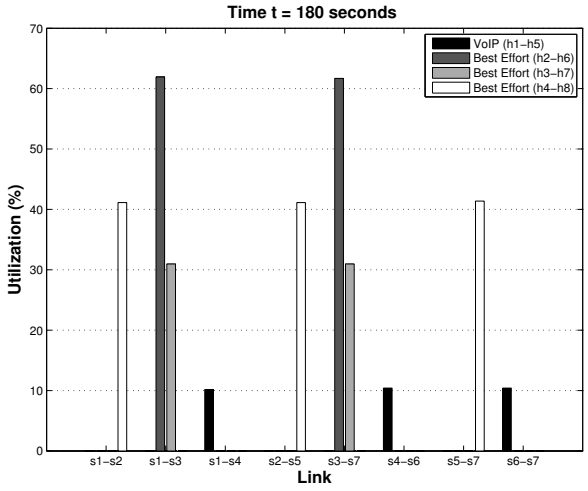

Fig. 9. Links utilisation observed at $t = 180$ s.

At $t = 160$ s, a new VoIP session starts between H_1 and H_5. Since the costs of the network links have been updated according to the cost function for VoIP traffic, and taking into account that there are three active best effort flows, the selected forwarding path for VoIP traffic flow is S_1-S_4-S_6-S_7. Figure 9 shows the utilisation of network links at t = 180 to evaluate if the forwarding rules are correctly installed by the controller. As highlighted in the figure, the utilisation of links chosen for the VoIP traffic is around 10%, whereas the utilisation of the links S_1-S_2, S_2-S_5, and S_5-S_7 is around 40% due to BE3. The utilisation of the links used by BE1 and BE2 remains around 90%.

VI. Conclusions and Future Work

In this paper, we presented the architectural design along with the experimental validation of a control application that enables differentiated routing for traffic flows belonging to different service classes.

The application is built on top of POX and makes routing decisions leveraging on real–time network statistics. A DPI engine has been developed and integrated in the control framework to classify the type of traffic, enforcing this way different policies for differentiated traffic routing. This architecture has been validated in an emulated environment (*i.e.*, Mininet) when both best-effort and privileged traffic are present. We plan to add different classes of services (*i.e.*, gold, silver, bronze, best effort), each one associated to a different link metric. This will also require an improvement of the DPI engine.

Acknowledgements

This work was supported by the EPSRC INTERNET Project EP/H040536/1.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, 2008.

[2] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, "Design and implementation of a routing control platform," in *Symposium on Networked Systems Design & Implementation (NSDI)*. USENIX Association, 2005.

[3] H. Egilmez, S. Dane, K. Bagci, and A. Tekalp, "Openqos: An openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks," in *Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2012.

[4] R. Wallner and R. Cannistra, "An sdn approach: Quality of service using big switchs floodlight open-source controller," in *Asia-Pacific Advanced Network (APAN)*, 2013.

[5] K. Jeong, J. Kim, and Y.-T. Kim, "Qos-aware network operating system for software defined networking with generalized openflows." in *Network Operations and Management Symposium (NOMS)*. IEEE/IFIP, 2012.

[6] A. Ishimori, F. Faria, I. Carvalho, E. Cerqueira, and A. Abelem, "Automatic qos management on openflow software-defined networks," 2012.

[7] *RFC 2327, available at http://www.ietf.org/rfc/rfc2327.txt*.

[8] *POX, available at http://www.noxrepo.org/pox/about-pox/*.

[9] IEEE, "Station and media access control connectivity discovery," *IEEE LAN/MAN Standards Committee, IEEE Std. 802.1ab*, 2009.

[10] *Mininet, available at http://mininet.org*.