SDN CONTROL SOLUTIONS FOR AN INFRASTRUCTURE SUPPORTING MULTIMEDIA STREAMING FLOWS

Maria-Daniela TACHE (UNGUREANU)[1], Ovidiu PĂSCUȚOIU[2]

The 5G technology offers today a powerful support for a large range of high-level services, including media-oriented ones, for video, audio and voice flows. Within 5G management and control architecture, the Software-Defined Networking (SDN) technology has been proposed both in studies and standards, as a main component, while cooperating with Network Function Virtualization (NFV). For instance, in 5G slicing, an SDN-based control can play a major role, both at slice virtual tenant level and also at a lower infrastructure level, by dynamically managing network virtual or physical connectivity resources. Several types of SDN controllers are nowadays available. Which one to choose? This paper studies several solutions for SDN controllers, where they have the tasks to configure the SDN data plane transport media flows through SDN switches. The goal of this study is to prepare a SDN controlled infrastructure capable of supporting a study on Quality of Services (QoS) optimizations methods for multimedia streaming. Experiments have been defined and performed in this work, using some of the most common SDN controllers and an evaluation of their capabilities has been attempted. This study can provide real-life information to designers aiming to develop SDN controlled systems and particularly QoS optimized 5G slices for multimedia streaming.

Keywords: Software defined networking, 5G networks, connectivity, SDN controller, data rate, latency, Quality of Service (QoS), applications, QoS optimization, QoS provisioning.

1. Introduction

The 5G technology can support a large range of applications and services and among them multimedia streaming is a major area of applications in digital society. These forms of multimedia content have become integral to people's routines, transforming the way they consume information and entertainment.

In the case of real-time managed high-level services (i.e., those needing some guarantees, which are specified in a Service Level Agreement (SLA) contract the multimedia-rich applications should be served with a controlled level

[1] PhD, National University of Science and Technology POLYTECHNICA of Bucharest, Romania, e-mail: danielatache26@yahoo.com, ORCID ID: 0009-0003-4588-8824

[2] PhD, "Henri Coandă" Air Force Academy, Brașov, Romania, e-mail: ovidiu.pascutoiu@afahc.ro, ORCID: 0009-0009-6918-0218

of Quality of Services (QoS). At network level, the QoS requirements for real-time applications, can refer to the degree of bandwidth assurance for particular streams, limited latency, jitter, and packet/bit error rates. Reliability is also needed to prevent disruptions in communication sessions. The QoS assurance at network level is the basis for offering the end users a good Quality of Experience (QoE), observed at application levels. In 5G one can construct dedicated slices for media distribution meeting the QoS requirements. Therefore, QoS capable management and control are necessary, based on provisioning or, in advanced solutions including dynamic features. SDN cooperating with NFV can be powerful tools in the management and control 5G architectural planes.

The SDN controllers must have capabilities to dynamically configure, supervise and optimize real or virtual network connectivity resources. They can command the network elements on-the-fly, dynamically allocating resources, and prioritizing traffic in response (by installing/modify appropriate flow tables in network nodes) aiming to answer the demands of multimedia services.

Many SDN controllers are offered in the market. Therefore, a need emerges to assess and compare their capabilities versus various criteria such as complexity, cost, openness, real-time response, scalability for large networks, etc.

This paper considers two relevant examples of SDN controllers, which potentially could be used within the context of 5G multimedia streaming control. Our specific objective is to identify the strengths and weaknesses of Open Network Operating System (ONOS) [1] and OpenDaylight (ODL) [2] controllers by conducting an experiment in Mininet [3] framework. By comparing the results, the paper evaluates their capabilities to control an SDN-based infrastructure capable of offering QoS assurance. The output information of this study could help the network operators, researchers, and industry stakeholders in selection of solutions.

2. Related work

This section provides a short overview on state of the art in assessing and contrasting the ONOS and ODL SDN controllers, identifying their respective strengths and weaknesses for various network scenarios, also we are adding context by presenting the state of the art on architectures related to SDN networks.

- Performance Evaluation of ODL and ONOS Controllers

The work of Alex Rodriguez et al. [4] studies the performance aspects of ODL and ONOS controllers, including latency and throughput measured in the Data plane and also evaluated the scalability properties. The study evaluates also how these controllers handle different network sizes and loads. The results achieved showed that ONOS controller has better results in terms of jitter and latency.

- Wireless networks – SDN controlled: A Comparative Analysis

The work [5] evaluates the effectiveness of a few generic SDN controllers in terms of flow setup time, and resource utilization, offering insights into their suitability for managing wireless SDN networks.

- Industrial SDN Applications: A Suitability Study

The work [6] investigates the applicability of ODL and ONOS controllers in industrial SDN applications. It analyzes their performance in controlling and managing industrial networks and also evaluates their compatibility with industrial use cases. As a conclusion, it has been shown that ONOS and ODL had better results regarding the scalability and security versus other controllers like Ryu, Iris or SDN [6].

- Network Function Virtualization (NFV): An Integration Perspective

The work [7] is focused on comparing and evaluating the performance of multiple SDN controllers such as Ryu, ONOS, ODL and identifying the strengths and weaknesses in terms of latency, interface vendor support and traffic loads. The study answers two of the important questions: (a) how fast can a controller respond to PACKET_IN messages And (b) how many PACKET_IN messages can a controller handle per second? The ODL was found to be a better choice in terms of interfaces vendor support, while ONOS presented the best results under various traffic loads.

- Carrier-Grade Networks: Security Subsystems

The work [8] studies the integration of a security subsystem with ONOS. It examines its performance ability to reduce deployment runtime among peer applications. The results showed that developing specific policies in an SDN-application offers a 5 to 20% performance overhead.

- Topology related approaches

In [9] Phemius et. al. proposed DISCO, an open and extensible distributed SDN control plane which scope was addressing the resiliency, scalability and extensibility regarding large scale multi-domain networks. A DISCO controller is responsible of a network domain and exchange information with other neighbor domains for end-to-end flow management purposes. Beside the processes involved in the management of the network and the approach of the architects on this solution, the topology can be abstracted as multiple simple architectures involving a controller, switches and hosts named domains which are linked between each other with the scope of manage large distributed networks.

Mamushiane et. al. make use in [7] of an environment consisting on an controller, switches and MACs (hosts). It can be considered a classical topology which for the testing purposes was modified by increasing the number on emulated switches to determine the latency on network or varying the MACs number to

observe the effect on controller performance.

The above studies on ODL and ONOS controllers can help in selection of a suitable SDN controller based on specific network requirements and use cases. Our study is focused on controllers' capabilities to install appropriate paths in the SDN data plane to transport multimedia streaming flows.

3. Comparative analysis of ODL and ONOS for multimedia streaming

It is important to evaluate if a particular SDN controller can fulfill different sets of QoS requirements and if it has enough necessary flexibility and features for some specific use case. The collaboration between SDN controllers and QoS-capable network nodes is crucial for delivering a high-quality and predictable service experience and particularly in 5G deployments. One should evaluate how well a given controller can assure QoS support for functions like traffic classification, traffic policing, traffic shaping, queue management, packet scheduling with different policies and priorities, congestion control, and more. The SDN controller the execution units (nodes or VNFs) must cooperate to translate those the controller decisions into tangible actions. If the execution units lack QoS capabilities, the effectiveness of the entire QoS framework would be compromised.

The open-source Open Network Operating System (ONOS) and the OpenDaylight (ODL) are among the most popular controllers. It is useful to compare their capabilities in order to make a selection between them.

This paper aims to contribute to this decision-making process by conducting a comparative analysis of ONOS and ODL controllers while preparing a SDN infrastructure in the data plane suitable for 5G multimedia streaming. We will evaluate their performance, scalability, extensibility, and adaptability to varying multimedia traffic scenarios.

3.1. Open Network Operating System (ONOS) [10]: it is recognized for its scalability and adaptability in managing diverse network environments. One ONOS's strength lies in its ability to efficiently allocate network resources, prioritize traffic, and dynamically adjust to network conditions in real-time. Therefore, it could be useful in some 5G slicing contexts, where scaling of resources is needed during active traffic sessions.

Studies [11] have shown ONOS's effectiveness in orchestrating network resources, to reduce network latency and ensure high throughput for multimedia applications. As shown in [12] ONOS appears to be significantly more stable than other SDN controllers, and it is much faster in reacting to topology event updates.

The controller's robustness in handling multimedia traffic in a 5G context [13] has led to its adoption in various research projects and experimental

deployments, and it evaluates how ONOS deals with different failure scenarios in the control and data planes. The extensibility of ONOS through its application framework enables the development of customized QoS management solutions tailored to specific multimedia streaming requirements.

The ONOS architecture comprises three functional layers arranged from top to bottom: Application, Core and the Providers + Protocols layer.

The App layer encompasses various applications seamlessly integrated within ONOS. This layer exposes interfaces to external applications and controller administrators through RESTful APIs, GUI, and CLI. It leverages the interfaces provided by the Core layer to execute the logic of network applications.

ONOS allows a distributed core architecture (the control plane is distributed across multiple nodes to enhance scalability). Its primary responsibilities include gathering information about the underlying network's status, collaborating with the App layer to execute network application logic. It can adapt to various Providers, that are modules within ONOS that implement the specific drivers or adaptors required to support different southbound protocols, for example RESTful, BGP, OSPF, ISIS. In the horizontal direction, it employs a cluster mechanism to facilitate communication and synchronization among multiple ONOS instances.

The Protocols layer serves as the implementation hub for a range of southbound protocols as mentioned above. It interfaces with the Core layer through Providers, facilitating the management and control of network devices lower in the stack. To be more specific, the provider and protocols layer manages the interaction with network devices.

The vertical protocol used between the ONOS controller and the network nodes of the data plane is typically the OpenFlow protocol.

3.2. OpenDaylight (ODL) Controller [14]: it offers a flexible and modular platform for network management and control. Its role has been investigated in enhancing QoS for multimedia streaming in 5G networks. ODL's architecture allows for the integration of various plugins and modules, making it suitable for diverse multimedia traffic optimization scenarios.

Through the ODL controller, researchers have implemented dynamic QoS policies, traffic shaping mechanisms, and traffic prioritization schemes to ensure seamless multimedia streaming experiences over 5G networks. Its adaptability and the availability of well-defined APIs have enabled easy integration with other network components, further enhancing its suitability for 5G multimedia QoS management.

ODL adopts a structured architectural approach, featuring well-defined integration points and accessible APIs. These facets enable both end users and networking vendors to actively engage with its robust SDN capabilities. Note that

while ODL can be deployed in a distributed manner, it does not inherently have a distributed core architecture in the same way as ONOS. Its modularity allows for the distribution of specific components.

The southbound interface in ODL ensures compatibility with a wide array of networking technologies and hardware provided by various vendors. This allows ODL to harness the capabilities of diverse vendors' equipment effectively.

On the other hand, the northbound interface of ODL exposes APIs designed to cater to end users' needs, as well as fostering compatibility with other cloud technologies. This facet facilitates seamless integration and interaction with ODL's capabilities, enhancing its versatility and usability in a variety of networking environments.

3.3. Ostinato [15] is an open-source network traffic generator often used for network testing and research purposes. Its key functional components are:

1. User Interface (GUI): - makes Ostinato a user-friendly application that help the user in the process of packet generation and traffic modeling. The packets can be designed with specific attributes so the testing process can be as customized as the scenario imposes. Those customizations can be related to IP addresses, protocols, headers, content and more.
2. Packet Crafting Engine: - reside at the core of Ostinato and is the entity that facilitate the packet crafting, structuring and customization process.
3. Protocol Support: - enables users to emulate different types of network communications like data transfers, VoIP calls or media streaming. The range of supported protocols is rich and consist of well-known protocols like TCP, UDP, ICMP, IP and also other more specific, related to different applications used in testing scenarios.

Ostinato offers benefits for:

- Mininet Integration: - allows users to generate traffic while interacting with SDN controllers and different network topologies. In the Mininet environment, Ostinato can be used as a separate host in the topology.
- performance and scalability: Because of the multi-thread architecture and the distributed mode, Ostinato is known for high performance and scalability.
- conducting large-scale testing scenarios to evaluate performance of SDN controllers.

Ostinato is a tool which can support various Use Cases:

- Network performance testing: - jitter, packet loss, latency are some of the most common tests that Ostinato can conduct. Other testing scenarios can assume stress-testing for network devices and capabilities measuring under different traffic conditions.

- Vulnerability assessment: - different patterns or abnormalities can be induced into the traffic to evaluate the behaviors of security systems and to evaluate the network resilience. By identifying possible defense breaches of the network, Ostinato can help in assessment experiments.
- Security professionals utilize Ostinato to evaluate the resilience of network based on protocol analysis by diving into the packet composition. The packets can be captured on the route and modified to study the behavior of the protocols used to send that packets or what issues may appear by modifying the structure of captured packets.
- SDN controller evaluation: -by varying the amount of traffic, the timing that traffic occurs or simulating different network scenarios, Ostinato can evaluate the performance and response of SDN controllers, the goal being to observe the way different controllers react to network conditions changes.

Documentation and Support:

As it can be seen from the official page https://ostinato.org/docs/ , Ostinato have plenty of documentation sources like FAQ's, forums, blogs that aims to help and guide users regarding Ostinato. It is accessible for both beginners (installation and configuration tutorials), and also to advanced users that may want to personalize the Ostinato experience.

As a bottom line, we can say that Ostinato is a very robust packet generation network tool whose capability is to test the network using different approaches and techniques. For those reasons we can consider that is probably a must have asset in network testing and analysis.

We have chosen Ostinato for our experiments because it can be seamlessly integrated with Mininet, a popular network emulator. This integration allows us to create realistic network topologies and inject customized traffic patterns into Mininet networks.

3.4 Mininet [16]: is an open-source network emulator that facilitates the creation and customization of network topologies that use the SDN principles:

1. Realistic network simulation:
    - It can create virtual networks with a high degree of similarity with the physical real-world networks, making from Mininet a very valuable solution for testing SDN solutions for networks, including the 5G models.
2. Cost-efficient development:
    - Because it doesn't need expensive physical hardware, Mininet eliminates the cost problem, allowing developers to test a lot of scenarios with the benefit of not being forced to buy any physical device.

Use Cases for Mininet:

1. SDN Controller Testing:
   - By allowing developers to modify controller behavior and performance, Mininet facilitates the evaluations of SDN controllers in 5G context. Different network topologies or traffic flow scenarios can be emulated for the purpose of SDN testing.
2. Prototyping and Development:
   - Mininet supports development, testing or improvement of the SDN. Those approaches can be related to applications, protocols or even services that networks run so we can assume is a valuable tool for 5G evaluations.
3. QoS and Traffic Engineering:
   - Developers can enable QoS features and deploy traffic engineering strategies on SDN networks, Mininet can inform the designers whether the tested environment has an optimal resource utilization and the network is efficiently managed.
4. Network Security Testing:
   - Intrusion-detection scenarios can be applied on SDN networks emulated with Mininet for the purpose of evaluating the security effectiveness of different topologies.

Mininet offers several benefits [16]:

- Rapid prototyping: Mininet is easily accessible; it is simple to create or modify networks, change behavior of components, establish communication paths and test how all things can work better in different scenarios the time for launching innovative applications or services is considerably reduced.
- Repeatable testing: conducting repeated testing scenarios on the same topology guarantee the stability of a topology and also lower the chance of an unidentified issue. Mininet offers the possibility to launch controlled and repeatable experiments to fulfill this scope.
- Resource-efficiency: Mininet is a cost-effective SDN testing solution because is based on virtualized network resources, meaning that developers are not restricted by equipment costs to conduct their testing scenarios.

4. Experiment implementation

4.1 Objectives

The objective of this study is to use the ONOS and ODL SDN controllers in order to create a SDN infrastructure able to support a future evaluation of the capabilities on QoS assurance in static or dynamic contexts. This experiment is

still preliminary and focused on creating the framework to evaluate some QoS mechanism and determine parameters in topologies controlled by ONOS or ODL.

The 5G system architecture is very complex; it has been investigated in many studies and also defined in 3GPP, ITU-T, ETSI, 5GPPP standards, e.g., [17][18]. In the management and control subsystem, as well in the user plane, support technology like SDN, NFV and cloud/edge computing are proposed – based essentially on virtualization. The SDN subsystems are mainly used for connectivity control both between physical and virtual network functions (VNF). The SDN and NFV are cooperating. Moreover, a 5G sliced systems architecture can be split in two macro-layers [19]. The upper macro-layer is specific to each slice tenant. Here, a *tenant SDN controller* dynamically configures and chains VNFs to realize network services in the tenant domain. It only controls the SW applications of the VNFs for configuration and chaining purposes, but not their underlying Network Function Virtualization Infrastructure (NFVI) resources, which belong to the lower layer.

Another *infrastructure SDN controller* manages and controls is the NFVI (lower macro-layer) network resources (placed in a NFVI-Points of Presence or a WAN) to set up the connectivity for communicating the tenant VNFs in the infrastructure domain. It manages and controls the connectivity among the virtualization containers that host the tenant VNFs' software applications.

The scope of this study is limited. It cannot cover the overall aspects of a 5G system architecture; it is focused on SDN control subsystem and investigates several variants of realization of such a control.

The network entities involved are open V-switches and SDN controllers, hosts and traffic generators. The communication between controller and switches is based on the OpenFlow protocol. The popular User Datagram Protocol (UDP) protocol is transporting the traffic. For real-time flows re-synchronization at receiver side, the Real Time Protocol (RTP) is used.

A simplified picture of the subsystem SDN architecture is presented in Fig. 1. This subsystem is oriented to the lower layer of the general 5G architecture, i.e. on user/data plane belonging to the NFVI. To simplify the picture, the real traffic paths through the switches are not fully presented in the picture. The assembly comprised two virtual switches, OVS1 and OVS2 controlled by an SDN controller. The switches aggregate traffic from eight hosts (h1-h8). The traffic was generated using *iperf* (this can be also done by using Ostinato traffic generator) and transported via UDP and RTP combinations; the traffic circulates between specific hosts pairs - for instance, from host 'x' to host 'y' via switches —allowing for a granular analysis of network performance.

In reference to the state of the art, the proposed architecture topology is a standard SDN topology consisting on a SDN controller, switches and hosts. Other studies [7][9] on SDN used similar topologies with the focus on topology impact

over the network performance and how those performances could be improved by scaling either the network dimension or by distributing it to separate networks. Our approach on the study is to test how the network responds on different scenarios related to the protocols of the data passing through the network.

In a 5G sliced context, this infrastructure could model a part of a 5G core network where the data paths might belong to the same or different slices.

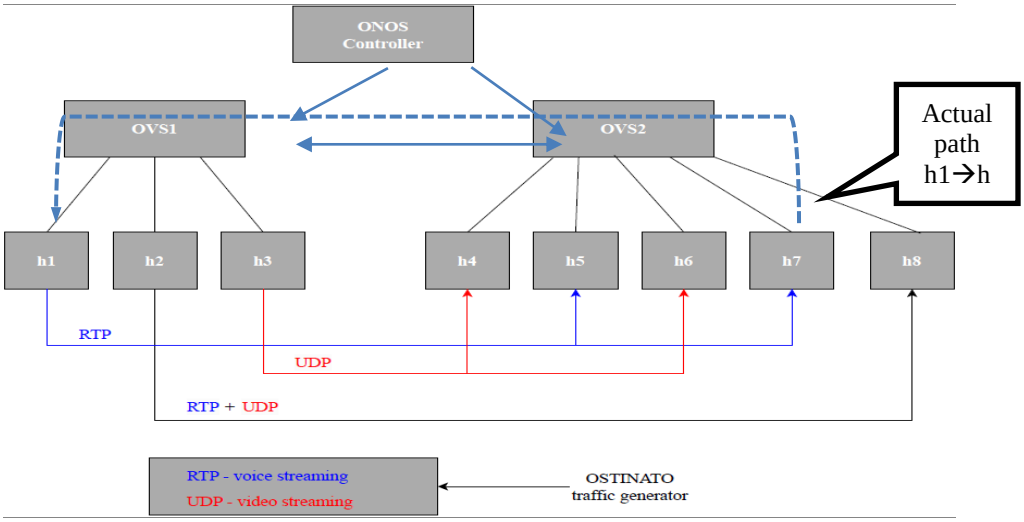

Fig. 1: Proposed system architecture

Network performance, especially latency and throughput, was recorded for the combination of RTP (Real-time Transport Protocol) and UDP (User Datagram Protocol) traffic [20]. Error rates were also monitored to evaluate the controllers' efficiency in managing network flows under varying load conditions.

4.2 Implementation details

The simulation model was built using Mininet version 2.3, a virtual network emulator running on a Linux platform, specifically using a virtual private server (VPS) environment provided by Hetzner [21]. This setup ensured that computational resources - channeled and allocated according to the experiment requirements - were sufficient to prevent any restrictions caused by hardware limitations. Each VPS was equipped with defined computational resources to reflect real-world deployment scenarios as closely as possible. In terms of specific resources, each VPS was configured with a certain amount of RAM, storage capacity and processing power. For example, each VPS could be allocated 4GB of RAM, 2 CPU cores and 50GB of SSD storage, but these values varied depending on the needs of the experiment. A total of 5 virtual machines were used to create a robust network environment and allow multiple instances of Mininet and SDN controllers such as ONOS and ODL to run simultaneously. This approach ensured

the support for a detailed and accurate simulation of network performance on quality of service (QoS) in SDN for multimedia streaming.

The ODL (version Phosphorus) and ONOS (version 2.5) controllers were installed on Linux-based Virtual Private Servers (VPS). This installation was carried out using standard package management commands like *sudo apt-get*, ensuring that both controllers operated on identical software versions to maintain consistency in experimental conditions. The ODL controller utilizes RESTful HTTP requests to modify network configurations, such as setting bandwidth limits and prioritizing certain types of traffic. This allows for flexible and dynamic traffic management. The ONOS controller implements similar bandwidth policies using the REST API, offering an equivalent but distinct approach in managing network traffic.

The ODL and ONOS controllers have been implemented and tested as described in the setup_sdn_controllers () function.

Virtual Network Functions (VNFs) have been implemented for traffic generation and network simulation. This incorporated the use of Mininet, as delineated in the *setup_mininet_topology()* function, to construct a virtual network topology. Virtual switches and hosts are defined, thereby crafting a network that facilitated the emulation of traffic and network conditions. The network links between hosts and switches have an initial specified bandwidth of 10 Mbps and a delay of 0.005 milliseconds.

We used iperf and rtpsend (given the fact that our example consists in a small topology) to initiate specific traffic flows between the simulated network entities, employing for the transport the UDP and RTP protocols, but we strongly recommend using Ostinato for more accurate results in bigger topologies.

The rate of traffic generation specified in units relevant to the context was packets per second. The size of each packet in the network traffic is 1470 bytes.

Parameters such as bandwidth limits, traffic prioritization, and error rates, have been controlled manually, using scripts developed by the authors. For monitoring network performance and capturing relevant data, the *monitor_traffic()* function was utilized. This allowed for the capture of traffic from virtual network interfaces and its storage in *pcap* files for subsequent analysis. Using this function, we were able to record key network performance parameters, such as latency and throughput, under various network load conditions.

5. System setup and configuration

Starting with the schematic representation of the network we want to test, we created a python script whose role is to add the network elements we need to the Mininet. Below is the representation of our network. The script describes the

steps to create the different entities in the network, h=host, s=switch, c=controller, the static way in which they are connected to each other and the IP addressing for each element as well as the communication policies between the hosts.

The script was named of the form toposcript.py and for each type of controller it was run in Mininet using a command of the form

sudo mn --custom ~/<script location>/toposcript.py --topo mytopo --controller=remote,ip=[IP] where IP is the IP of the machine on which the controller was installed.

Fig. 2 – Results of ICMP test after topology implementation

The ONOS controller was downloaded in the /opt folder using the curl utility, using the command

curl -XGET -O https://repo1.maven.org/maven2/org/onosproject/onos-releases/2.0.0/onos-2.0.0.tar.gz

Since the kit is in an archive, we had to extract the files from the archive using the command

tar -xvf onos-2.0.0.tar.gz -C /opt/onos

Fig. 3 – graphical representation of a test topology in ONOS interface

ONOS will discover and display the hosts only when they exchange data with each other, which is why I ran the ***pingall*** command in Mininet, the hosts thus initiating communication with each other, and ONOS thus becomes aware of their existence.

We proceed with the actual installation of the ODL controller that we will download from the official website as in *Fig. 4*. We unzip the karaf-0.8.4.zip

archive, then check the existence of the folder created after unzipping and start the controller. The actual starting of ODL is done by accessing the*/bin/karaf* executable from the unzipped folder.

```
mininet@mininet-vm:~$ curl -XGET -O https://nexus.opendaylight.org/content/repos
itories/opendaylight.release/org/opendaylight/integration/karaf/0.8.4/karaf-0.8.
4.zip
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  351M  100  351M    0     0  12.9M      0  0:00:27  0:00:27 --:--:-- 9807k
```

Fig. 4 - Download process of OpenDaylight

We notice that everything works as it should after creating the topology, and to open OpenDaylight I accessed the address of the virtual machine in the browser, in the form: *http://IPVM:8181/index.html#/topology*
Here, we have hosts, switches, and flows, along with the links connecting hosts to switches and switches to each other, as defined within the switch configuration and illustrated in Fig. 5.

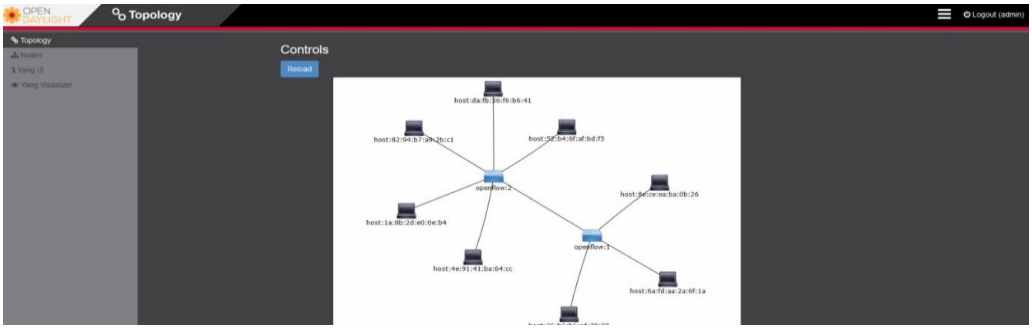

Fig. 5 – graphical representation of the topology in ODL interface

As a final test, we created the script below that helped us verify the traffic between several hosts, and the output can be seen in *Fig. 6*.

```
# Configuring UDP traffic from H3 to H4 and H6 with Video QoS (DSCP 0xb8)

h3.cmd('iperf -c 10.0.2.4 -u -p 5002 -t 40 -S 0xb8 > /home/h3_udp_output.txt 2>&1 &')
# SET DSCP FOR VIDEO
h4.cmd('iperf -s -u -p 5002 > /home/ h4_udp_output.txt 2>&1 &')
h6.cmd('iperf -c 10.0.2.4 -u -p 5002 -t 40 -S 0xb8 > /home/ h6_udp_output.txt 2>&1 &')

# Configuring RTP traffic from H1 to H5 and H7 with Voice QoS (DSCP 0x28)

h1.cmd('iperf -c 10.0.2.5 -u -p 5001 -t 30 -S 0x28 > /home/h1_rtp_output.txt 2>&1 &')
# Set DSCP for Voice
h5.cmd('iperf -s -u -p 5001 > /home /h5_rtp_output.txt 2>&1 &')
h7.cmd('iperf -s -u -p 5001 > /home/ h7_rtp_output.txt 2>&1 &')
```

```
# Configuring mixed RTP and UDP traffic from H2 to H8 with mixed QoS

h2.cmd('iperf -c 10.0.2.3 -u -p 5001 -t 50 -S 0x28 >/home/h2_mixed_output.txt 2>&1 &')
 # Set DSCP for Voice and Video
h8.cmd('iperf -c 10.0.2.5 -u -p 5002 -t 50 -S 0x28 >/home/h8_mixed_output.txt 2>&1 &')
# Set DSCP for Voice and Video

# Applying QoS for RTP (Voice) traffic - Set DSCP 0x28 on H1 and H2
h1.cmd('tc qdisc add dev h1-eth0 root handle 1: htb default 12')
h1.cmd('tc class add dev h1-eth0 parent 1: classid 1:1 htb rate 1mbit ceil 1mbit')
h1.cmd('tc filter add dev h1-eth0 protocol ip parent 1:0 prio 1 u32 match ip dscp 0x28 0xff00
flowid 1:1')
# Applying QoS for UDP (Video) traffic - Set DSCP 0xb8 on H3 and H4
```

Traffic Control (QoS):

- tc commands are used to configure traffic shaping and quality of service (QoS) for different types of traffic based on the Differentiated Services Code Point (DSCP).
    - **Voice traffic (RTP):** DSCP 0x28 is used for voice traffic on h1 and h2.
    - **Video traffic (UDP):** DSCP 0xb8 is used for video traffic on h3 and h4.

Traffic Generation (iperf):

- **iperf** is used to generate UDP traffic, specifying the -S option to set the DSCP values.
    - Voice traffic is sent from h1 to h5 and h7.
    - Video traffic is sent from h3 to h4 and h6.
    - Mixed traffic is generated from h2 to h8.

DSCP 0x28 corresponds to a specific value in the DSCP field, which is used for classifying and prioritizing traffic. It is part of the **Differentiated Services** (DiffServ) model for QoS.
DSCP 0x28 is often used for **RTP (Real-Time Protocol)** traffic, which includes **Voice over IP (VoIP)** calls. In our case, the DSCP value indicates that the traffic is high-priority and should be treated with low latency, minimal jitter, and low packet loss, which is essential for real-time communications.
DSCP 0xb8 is often used for **UDP-based video traffic**, where the goal is to ensure a smooth streaming experience with relatively low latency and minimal interruptions, but it is not as high-priority as real-time **voice traffic (DSCP 0x28)**.

```
------------------------------------------------------------
Client connecting to 10.0.2.5, UDP port 5001
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.0.2.1 port 58722 connected with 10.0.2.5 port 5001
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Diagram
[  3]  0.0- 30 sec   1.23 MBytes   10 Mbits/sec   0.005ms    0/1470 (0%)
[  3] Sent 1470  datagrams
```

Fig. 6 - Example of traffic values recorded at the host level

6. Experimental results

Using the previous described procedure, we tested the network capabilities for the entire proposed system using the paths and protocols described in it. The tables 1 - 4 below are summarizing our test results and capabilities of the tested SDN controllers (ONOS, ODL). Based on the jitter and loss, parameters comparative graphs in Fig. 7 and 8 were designed, to provide a better image on the final results.

Table 1

Traffic results for proposed architecture using ONOS Controller

ONOS				
Traffic for 1470 sent datagrams	Bandwidth Mb/s	Injected Traffic size Mb	Jitter	Loss
H1 to H5 voice	10	10	0.005 ms	0/1470 (0%)
H1 to H7 voice	10	30	0.010 ms	30/1470 (2.04%)
H3 to H4 video	20	20	0.015 ms	0/1470 (0%)
H3 to H6 video	20	40	0.018 ms	5/1470 (0.34%)
H2 to H8 mixt	30	30	0.009 ms	0/1470 (0%)
H2 to H8 mixt	30	90	0.20 ms	25/1470 (1.70%)

Key Distinction:

- **Bandwidth (Mb/s):** The maximum rate at which a link can carry traffic (e.g., 10 Mb/s means the link can transmit 10 megabits per second).
- **Injected Traffic (MB):** The total amount of data sent over the link during the test. For example, "30 MB of traffic" refers to **30 megabytes total**, but it does not specify the rate at which this traffic is sent.
  For example, H1 to H7 voice - Here, "30 MB" is the total amount of data transmitted **over the test duration**, not a traffic rate. If the test duration is long enough (in this case 30 seconds, the average traffic rate will still fit within the link's 10 Mb/s capacity, resulting in low packet loss – 2.04%. If the test duration is shorter (e.g., 10 seconds), the traffic rate would increase proportionally, potentially exceeding the bandwidth and causing greater packet loss. The test duration for traffic generation is set explicitly in the **iperf commands** using the -t option. The -t flag specifies the duration of the test in **seconds**.

Table 2

Traffic results for proposed architecture using ODL Controller

ODL				
Traffic for 1470 sent datagrams	Bandwidth Mb/s	Injected Traffic size Mb	Jitter	Loss

H1 to H5 voice	10	10	0.005 ms	0/1470 (0%)
H1 to H7 voice	10	30	0.010 ms	150/1470 (14.65%)
H3 to H4 video	20	20	0.015 ms	0/1470 (0%)
H3 to H6 video	20	40	0.018 ms	10/1470 (0.68%)
H2 to H8 mixt	30	30	0.009 ms	0/1470 (0%)
H2 to H8 mixt	30	90	0.20 ms	50/1470 (3.40%)

Table 3

Comparative table of the jitter regarding ONOS and ODL controllers

Jitter Graph	H1-H5 voice	H1-H7 voice	H3-H4 video	H3-H6 video	H2-H8 mixt	H2-H8 mixt
ONOS	0.005	0.01	0.015	0.018	0.009	0.2
ODL	0.005	0.01	0.015	0.018	0.009	0.2

Table 4

Comparative table of the loss regarding ONOS and ODL controllers

Loss Graph	H1-H5 voice	H1-H7 voice	H3-H4 video	H3-H6 video	H2-H8 mixt	H2-H8 mixt
ONOS	0	0.0204	0	0.0034	0	0.017
ODL	0	0.1465	0	0.0068	0	0.034

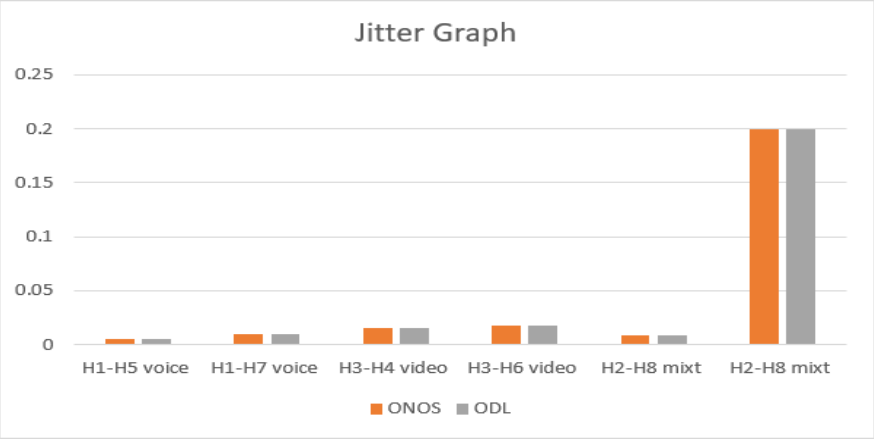

Fig. 7 – ONOS and ODL graph based on collected jitter values

In literature, others tried to do performance tests of controllers. In the case of [7], the tests revealed also that ONOS tend to have better performance compared to ODL based on some tests that supposed a variation on emulated topology (switches and hosts). To conclude the results, our tests related not to the topology but to the data passing through the network, confirms once again that ONOS controller has better performances than ODL controller. The advantage of using SDN-type control inside 5G architecture is definitely a strong approach inside the 5G architecture (see [17], [18], [19] above and many others). However, further investigation of different variants of controller solutions is valuable.

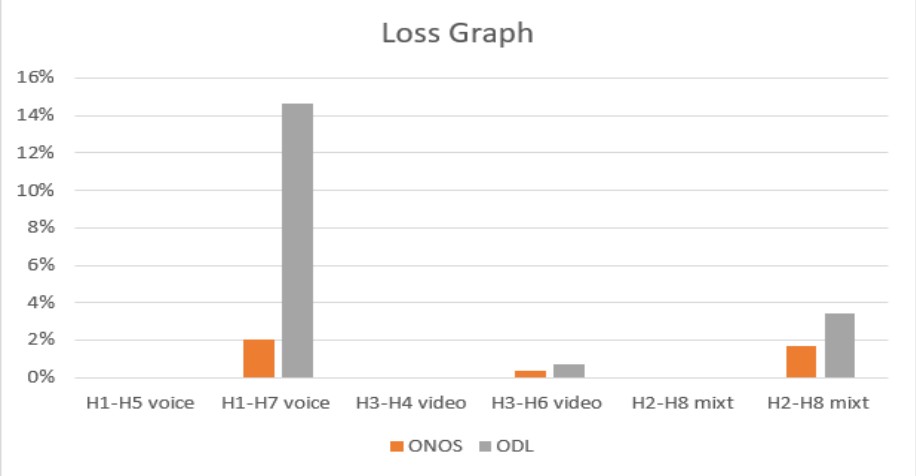

Fig. 8 – ONOS and ODL graph based on collected loss values

7. Conclusion

This study had the purpose of selecting and installation/interconnection of the complex system components (controllers, generators, Mininet) and creating the environment with sets of virtual machines for different functions. This assembly can model a subsystem in a SDN overall architecture.

Related to the testing results we showed that both controllers are sensitive on variations of the packet size in relation with the allocated bandwidth and while the jitter results remained the same, the loss parameter changed during the different tests.

The contribution of this study consists in testing the connectivity capabilities of a SDN assembly (controller, switches, hosts) for several types of controllers and proposing an approach that separates the data flows, based on the protocols used at transport level to send the data.

The proposed topology could support (in a future) more complex evaluations related to QoS, such as bandwidth assurance, QoS policies, priorities, overloads of the data paths, changing dynamically of the flows requirements which would need interventions of the controllers lead us to configuration of all the SDN controlled infrastructure (by ONOS or ODL) and testing the functionalities concerning the traffic transport (via UDP or UDP + RTP) in appropriate way between different hosts.

Our goal is to continue studying these SDN controllers in order to test in depth the way of implementing QoS policies and algorithms that can be applied to them in order to not only have functionality tests but also performance tests

R E F E R E N C E S

[1]. https://wiki.opennetworking.org/, Accessed November 4, 2024
[2]. https://www.opendaylight.org/. Accessed November 4, 2024
[3]. http://mininet.org/. Accessed November 4, 2024

[4]. A. Rodriguez, J. Quiñones, Y. Iano and M. A. Q. Barra, "A Comparative Evaluation of ODL and ONOS Controllers in Software-Defined Network Environments," 2022 IEEE XXIX International Conference on Electronics, Electrical Engineering and Computing (INTERCON), Lima, Peru, 2022, pp. 1-4, doi: 10.1109/INTERCON55795.2022.9870107.

[5]. M. Bano, S. S. A. Gilani and A. Qayyum, "A Comparative Analysis of Hybrid Routing Schemes for SDN Based Wireless Mesh Networks," 2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Exeter, UK, 2018, pp. 1189-1194, doi: 10.1109/HPCC/SmartCity/DSS.2018.00200.

[6]. Urrea C, Benítez D. Software-Defined Networking Solutions, Architecture and Controllers for the Industrial Internet of Things: A Review. Sensors. 2021; 21(19):6585. https://doi.org/10.3390/s21196585

[7]. L. Mamushiane, A. Lysko and S. Dlamini, "A comparative evaluation of the performance of popular SDN controllers," 2018 Wireless Days (WD), Dubai, United Arab Emirates, 2018, pp. 54-59, doi: 10.1109/WD.2018.8361694.

[8]. Yoon, Changhoon et al. “A Security-Mode for Carrier-Grade SDN Controllers.” Proceedings of the 33rd Annual Computer Security Applications Conference (2017)

[9]. Phemius, Kevin, Mathieu Bouet, and Jérémie Leguay. "Disco: Distributed multi-domain sdn controllers." 2014 IEEE network operations and management symposium (NOMS). IEEE, 2014.

[10]. https://wiki.onosproject.org/display/ONOS/Overview+of+ONOS+architecture Accessed November 4, 2024

[11]. Goswami, B., Hu, S., Feng, Y. (2022). Software-Defined Networking for Real-Time Network Systems. In: Tian, YC., Levy, D.C. (eds) Handbook of Real-Time Computing. Springer, Singapore. https://doi.org/10.1007/978-981-287-251-7_69

[12]. https://opennetworking.org/wp-content/uploads/2019/09/ONOSvsODL-report-4.pdf. Accessed November 4, 2024

[13]. Lucas V. Ruchel, Rogério C. Turchetti, and Edson T. de Camargo. 2022. Evaluation of the robustness of SDN controllers ONOS and ODL. Comput. Netw. 219, C (Dec 2022). https://www.sciencedirect.com/science/article/abs/pii/S1389128622004376

[14]. https://wiki.opendaylight.org/pages/viewpage.action?pageId=336424. Accessed November 4, 2024

[15]. B. R. Patil, M. Moharir, P. K. Mohanty, G. Shobha and S. Sajeev, "Ostinato - A Powerful Traffic Generator," 2017 2nd International Conference on Computational Systems and Information Technology for Sustainable Solution (CSITSS), Bengaluru, India, 2017, pp. 1-5, doi: 10.1109/CSITSS.2017.8447596.

[16]. https://opennetworking.org/mininet/. Accessed November 4, 2024

[17] 3GPP TS 23.501 V17.2.0 (2021-09) "Technical Specification Group Services and System Aspects;System architecture for the 5G System (5GS)", Stage 2, (Release 17), https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3144

[18] 5GPPP Architecture Working Group, "View on 5G Architecture", Version 4.0, August 2021, https://5GPPP.eu/wp-content/uploads/2021/08/Architecture-WPv4.0 forPublicConsultation.pdf , https://5g-ppp.eu/white-papers/

[19] ETSI GR NFV-EVE 012 V3.1.1 (2017-12), Release 3; NFV Evolution and Ecosystem; Report on Network Slicing Support with ETSI NFV Architecture Framework

[20]. Li, Y., Niyato, D., Wang, P., Kim, D. I., & Han, Z. (2020). A Survey on Network Slicing for 5G and Beyond: Fundamentals, Architectures, and Applications. IEEE Communications Surveys & Tutorials, 22(1), 674-707. doi:10.1109/comst.2019.2940746

[21]. https://www.hetzner.com/. Accessed November 4, 2024