PERFORMANCE EVALUATION EXPERIMENTS FOR VIDEO AND VoIP TRAFFIC WITH RYU CONTROLLER AND MININET FRAMEWORK – PART I

Pantelimon-Teodor TIVIG[1], Eugen BORCOCI[2]

Software Defined Networking (SDN) architecture and technology, among other advantages, are expected to be able to manage high volumes of traffic in the future networks. In many popular today services of (especially those multi-media - related) the Quality of Service(QoS) the Quality of Experience (QoE) perceived by the final user is of high significance. This paper considers a SDN controlled network – using a RYU controller and develops dedicated experiments to evaluate the traffic performance tests at the node level connecting final users. The evaluation criteria have been delayed, round trip time and bandwidth. The objective is to determine performance results while considering VoIP and data services. The experiments have used the Iperf tool.

Keywords: Software Defined networking (SDN), Quality of Service (QoS), Quality of Experience (QoE), RYU controller, Mininet, Wireshark

1. Introduction

Today the *Internet* transfers not only text and data, but also high volume of media-related traffic. The need of flexibility and programmability is strong in the current complex frameworks. The SDN technology can offer an answer for controlling high bandwidth demand requested mainly by media-related services (with unicast or multicast flows) while offering potential flexibility for adaptation of the resource allocation to traffic variation. The traditional distributed management and control plane do not natively offer an abstracted logical view of the entire network. Consequently, manual individual configuration of network nodes is necessary, which limits the scalability to keep up with various network condition and services demands. The SDN frameworks logically centralizes the network control, by moving the control software from network nodes into controllers. Network Functions Virtualization (NFV) technology is complementary to SDN and in several recent architectures SDN and NFV allow,

[1] PhD. Student, Doctoral School of Electronics, Telecommunications and Information Technology, University POLITEHNICA of Bucharest, Romania, e-mail: pantytivig@yahoo.com
[2] Prof., Dept. of Electronics and Telecommunications, University POLITEHNICA of Bucharest, Romania, e-mail: eugen.borcoci@elcom.pub.ro

while working in cooperation, the development of new functions that can meet and sustain the revolutionary demands of today's business [1].

In practice a mix of multimedia devices and services is seen, e.g., video devices used for home entertainment due to widely spread adoption of Ultra-High-Definition (UHD), or 4K, video streaming, healthcare monitoring. Voice, and video conferences are omnipresent in every day of life of any student and any worker for collaboration and working. The Cisco annual report shows the effect of the bit rate for 4K video at about 15 to 18 Mbps, that is more than double than HD video bit rate and nine times more than Standard-Definition (SD) video bit rate. These reports estimate that nearly 79 percent of the world's mobile data traffic will be video by 2022 [2]. Because video streaming is seen to be responsible for generating the main percentage of Internet traffic, it is important to know the traffic statistics (e.g., the bit rate observed by the clients) so that the network and video service providers can better adapt to the nowadays demands, and the Quality of Experience could be maximized for all end users [3,4].

This paper is focused on data streaming and VoIP traffic in an SDN controlled network through OpenFlow protocol. The study is focused essentially on data plane, while supposing the the SDN controller proactively installed paths based on flow tables injected into SDN switches. The data traffic load could increase and lead to the network segments overload; this can produce negative effects on the QoS at the user side and consequently on QoE. In an evolved system the SDN controller could be informed about such events and consequently reallocate resources in the network. To this aim, traffic statistics are necessary to collect network congestion information and then upload it to the controller. The SDN controller will decide how to solve the bandwidth needs of different competing voice and data clients flows.

The remainder of the paper is divided as follows; some relevant literature works on the above problems and the domain is presented in section 2. Section 3 summarizes the characteristics of SDN controllers and the Iperf traffic generator. Section 4 explains the methodology of this work. Section 5 presents and discusses the experimental results, and the final section 6 concludes the paper.

2. Related work

In the work [5] the authors propose to control IP multicasts into overlay networks through OpenFlow protocol, instead of Internet Group Management Protocol (IGMP) . In [6] rapid recovery in case of path failures is considered into IP multicast-based forwarding system. The study [7] uses a congested link to assess a performance study for traffic shaping that cope with multiple video links while using SDN-type of control. The study [8] shapes the individual traffic flow for each link to a fixed value, which could not be realistic duo to the fact the

number of connections can expand, resulting in the usable bandwidth can be less than the demanded bandwidth leading to low performance. The architecture Server and Network Assistant DASH (SAND) [9] is proposed to improve the QoE for final users in conjunction with video content distribution. This architecture proposes a control plane for video delivery that returns networks-based measurement of the metrics that obtains QoE.

3. SDN Controllers and Traffic generators

SDN Controllers are considered the main elements of the Network Operating System) (NOS). The controller manages the flow of control to and from the various network equipment through Southbound Interfaces and business applications through Northbound Interfaces [10]. It is placed between two parts of the network, at one part is network equipment and another part is the applications. Therefore, the SDN controller is recognized as control plane implmentation in SDN architecture The communication between the business applications and the network must pass through the controller. In SDN area various controllers are available: Ryu, Floodlight, POX, NOX, OpenDaylight, and many others.

This work has used RYU controller given its availability and well-described API's. It is a Python-based software platform. OpenDayLight is an open-source platform that offers an industry-supported framework for the SDN. It is available for free. POX is a Python-based platform and has become widely used compared to NOX. NOX is based on C++ programing language and is an open-source platform SDN control applications. The Floodlight is based on the Java programming language to create applications and it has the REST API's interface for device interfacing [11].

Iperf is an open-source network traffic generator used to measure bandwidth, delay and jitter from host to host. It is licensed under the BSD license. It has support for various parameters related to timing, buffers and protocols (TCP, UDP, SCTP with IPv4 and IPv6) [12]. The tests can be conducted for a set predefined amount of time or using a set amount of data. Iperf uses a client/server model. The server can be initiated even as a daemon and manage connections from multiple clients on different ports [13].

4. Methodology

In this study for emulating the network topology it will be used Mininet framework [14] and OpenVswitch [15]. This platform is very utilized in scientific fields to emulate SDN architectures.

Python programing language [16] is used to define the topology in an automated manner instead of providing commands into Mininet. Ryu SDN controller [17] is entirely written in Python. To evaluate performance statistics for

voice and data services a custom network tree topology was designed (Figure 1) with depth=3; every leaf switch has three hosts.

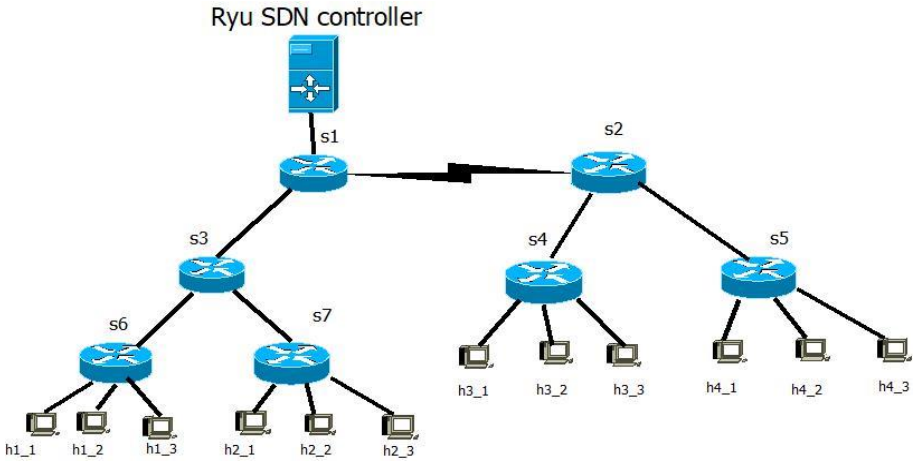

Fig. 1. Diagram for network topology

The test network infrastructure has 7 switches and 12 hosts emulated using Linux Namespaces [18] connected to Ryu. The code of the Python script used for network topology is provided below in Fig. 2

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import Controller,
RemoteController, OVSController
from mininet.node import CPULimitedHost, Host,
Node
from mininet.node import OVSKernelSwitch,
UserSwitch
from mininet.node import IVSSwitch
class CustomTopo( Topo ):
   "Minimal topology with a single switch and two
hosts"
   #Creating switches and links to connect them
   def build( self ):
      for a1 in range(1):
         a1 = self.addSwitch('s1')
      for a2 in range(1):
         a2 = self.addSwitch('s2')
         self.addLink(a1, a2)
      for a3 in range(1):
         a3 = self.addSwitch('s3')
         self.addLink(a3, a1)
      for a4 in range(1):
         a4 = self.addSwitch('s4')
         self.addLink(a1, a4)
      for a5 in range(1):
         a5 = self.addSwitch('s5')
      for a6 in range(1):
         a6 = self.addSwitch('s6')
         self.addLink(a5, a6)
       for a7 in range(1):
         a7 = self.addSwitch('s7')
         self.addLink(a5, a7)
       #Host defining and connecting to each switch
      for h1_ in range(0,3):
         h1 = self.addHost('h1_%s' %(h1_+1))
         self.addLink(a3, h1)
      for h2_ in range(0,3):
         h2 = self.addHost('h2_%s' %(h2_+1))
         self.addLink(a4, h2)
      for h3_ in range(0,3):
         h3 = self.addHost('h3_%s' %(h3_+1))
         self.addLink(a6, h3)
      for h4_ in range(0,3):
         h4 = self.addHost('h4_%s' %(h4_+1))
         self.addLink(a7, h4)

topos = {
   'minimal': CustomTopo
}
```

Fig. 2. Python Script for generating the proposed topology

In the next subsection a VoIP traffic test there will be performed using IPERF UDP test tool. TCP tests are utilized to evaluate the bandwidth between

the links of hosts. Instead, UDP tests are used to evaluate the Latency, Jitter, Packetloss.

5. Traffic tests using Iperf

Jitter, latency and packet loss cannot be measured using TCP test. TCP test is usually utilized for measure the bandwidth. For all traffic tests, we will use the topology from Fig. 2 and L3 switch application [19].

I. TCP tests

A. Preparing the topology

In this test, the research measures the bandwidth link from h1_1 to h1_2, which figure out how much traffic needs to be uploaded. For the test purpose the experiment sent the traffic from the h1_1 to be received at h1_2.

a. Create the topology

sudo mn --custom topo3.py --topo minimal --mac --switch ovs --controller remote

b. Run the RYU controller l3 switch application

ryu-manager ryu.app.simple_switch_13 ryu.app.ofctl_rest

c. In Mininet shell, do ping h1_1 to h1_2

mininet> h1_1 ping -c 6 h1_2

```
mininet> h1_1 ping -c 6 h4_3
PING 10.0.0.15 (10.0.0.15) 56(84) bytes of data.
64 bytes from 10.0.0.15: icmp_seq=1 ttl=64 time=0.810 ms
64 bytes from 10.0.0.15: icmp_seq=2 ttl=64 time=0.437 ms
64 bytes from 10.0.0.15: icmp_seq=3 ttl=64 time=0.277 ms
64 bytes from 10.0.0.15: icmp_seq=4 ttl=64 time=0.225 ms
64 bytes from 10.0.0.15: icmp_seq=5 ttl=64 time=0.277 ms
64 bytes from 10.0.0.15: icmp_seq=6 ttl=64 time=0.425 ms

--- 10.0.0.15 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5098ms
rtt min/avg/max/mdev = 0.225/0.408/0.810/0.196 ms
mininet>
```

Fig. 3. Reachability test between h1_1 and h1_2

d. Check the flows and statistics

sudo ovs-ofctl -O OpenFlow13 dump-flows s1

```
tivig@ubuntu:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s7
 cookie=0x0, duration=10.856s, table=0, n_packets=5, n_bytes=490, priority=1,ip,nw_src=10.0.0.4,nw_dst=10.0.0.15 actions=output:"s7-eth4"
 cookie=0x0, duration=10.853s, table=0, n_packets=5, n_bytes=490, priority=1,ip,nw_src=10.0.0.15,nw_dst=10.0.0.4 actions=output:"s7-eth1"
 cookie=0x0, duration=15.890s, table=0, n_packets=190, n_bytes=23216, priority=0 actions=CONTROLLER:65535
```

Fig. 4. Flow statistics

The study of the network using the ping utility is by analyzing the ICMP sequence packets sent into the time amount and the number of bytes. The Fig. 3 presents the reachability test between to hosts h1_1 pings h4_3 having the 10.0.0.15 IP address and send the 64 bytes of data with the time 0.810 ms. Into the

Fig. 4 ovs-ofctl the handshake has been fulfilled, the table miss entry has been transferred using l3 forwarding rules. ICMP echo reply has arrived successfully. Fig. 5 verifies the port statistics. Verifying the flow table in s7, Fig. 4 the table miss entries of the flow table s1 priority 1 are observed:

(i) source ip address: "nw_src=10.0.0.4" host h1_1 → Acitons (actions)=output: "s7_eth4"

(ii) source ip address: "nw_src=10.0.0.15" host h4_3 → Acitons (actions)=output: "s7_eth1"

e. Check the switch statistics

sudo ovs-ofctl -O OpenFlow13 dump-ports s7

```
tivig@ubuntu:~$ sudo ovs-ofctl -O OpenFlow13 dump-ports s7
OFPST_PORT reply (OF1.3) (xid=0x2): 5 ports
  port LOCAL: rx pkts=0, bytes=0, drop=240, errs=0, frame=0, over=0, crc=0
           tx pkts=0, bytes=0, drop=0, errs=0, coll=0
           duration=70.959s
  port  "s7-eth1": rx pkts=227, bytes=27583, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=56, bytes=6111, drop=0, errs=0, coll=0
           duration=70.988s
  port  "s7-eth4": rx pkts=20, bytes=1688, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=265, bytes=32210, drop=0, errs=0, coll=0
           duration=70.996s
  port  "s7-eth2": rx pkts=12, bytes=1016, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=259, bytes=31686, drop=0, errs=0, coll=0
           duration=70.993s
  port  "s7-eth3": rx pkts=11, bytes=926, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=259, bytes=31686, drop=0, errs=0, coll=0
           duration=70.988s
```

Fig. 5. Ports statistics

At this moment we have all set to run our traffic tests.

B. Traffic test from h1_1 to h4_3

The objective: Generate TCP traffic from h1_1 to h4_3 (Measure bandwidth from h1_1 to h4_3).

h1_1 transmitter

h4_3 receiver

a. Start the Iperf server in h4_3

h4_3 iperf –s &

b. Start the IPERF client in h1_1 and connecting to h4_3

h1_1 iperf -c h4_3

c. Analyze the results by flows

sudo ovs-ofctl -O OpenFlow13 dump-flows s1

```
tivig@ubuntu:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s7
 cookie=0x0, duration=76.978s, table=0, n_packets=501047, n_bytes=21968748526, priority=1,ip,nw_src=10.0.0.4,nw_dst=10.0.0.15 actions=output:"s7-eth4"
 cookie=0x0, duration=76.973s, table=0, n_packets=391371, n_bytes=25830870, priority=1,ip,nw_src=10.0.0.15,nw_dst=10.0.0.4 actions=output:"s7-eth1"
 cookie=0x0, duration=2462.548s, table=0, n_packets=343, n_bytes=37826, priority=0 actions=CONTROLLER:65535
```

Fig. 6. Flow statistics

We notice from Fig. 6 , traffic in the two directions h1_1 to h4_3 traffic is high. Represents the data traffic. h4_3 to h1_1 traffic is to a smaller extent. This is TCP Acknowledge traffic.

d. Evaluate the results by ports (see Fig. 7)

sudo ovs-ofctl -O OpenFlow13 dump-ports s7

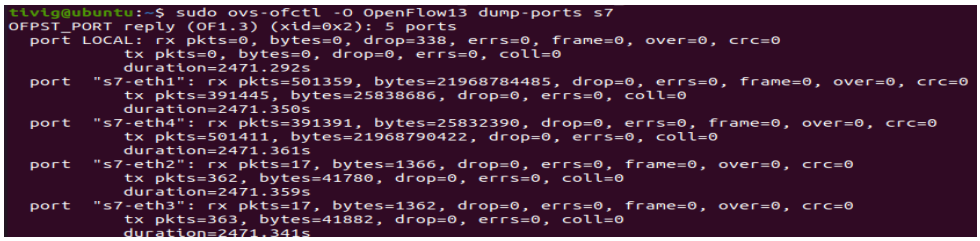

Fig. 7. Ports statistics

h1_1-----port1, port4---h4_3

Forward traffic:

- h1 transimits. port1 receives.
- port4 transmits, h4_3 receives.

Acknowledge:

- h4_3 transmits, port4 receives.
- port1 transmits. h1_1 receives.

C. Bidirectional traffic test h1_1 to h4_3 (sequentially)

a. Start IPERF server in h4_3

iperf -s &

b. Start the IPERF client in h1_1 and connecting to h4_3

h1_1 iperf -c h4_3 –r

Once the h1_1 to h4_3 (with ip address 10.0.0.15) traffic test is completed it will start the h4_3 to h1_1 traffic test, the test results can be seen below into Fig. 8.

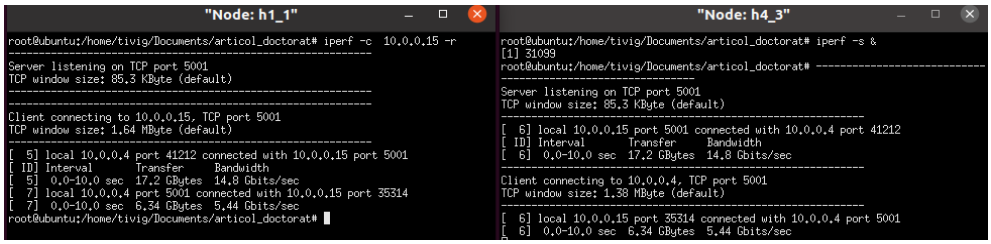

Fig. 8. Bidirectional test from h1_1 to h4_3

D. Parallel traffic test h1_1 to h4_3

a. Start IPERF server in h4_3
iperf -s &

b. Start the IPERF client in h1_1 and connecting to h4_3
iperf -c h4_3 –d

The test is executed from both sides h1_1 to h4_3 as well h4_3 to h1_1, and data results are presented into Fig. 9.

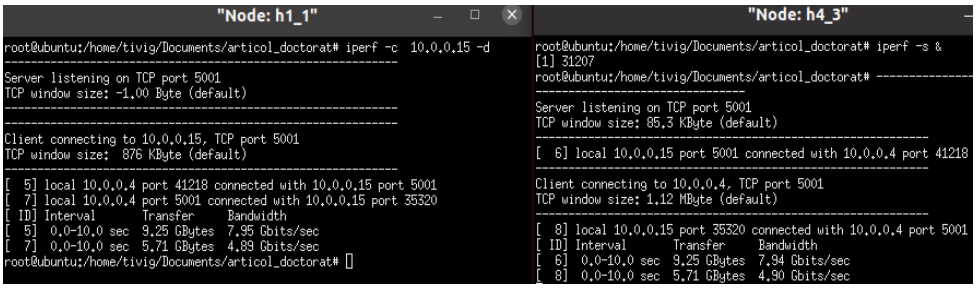

Fig. 9. Parallel traffic test h1_1 to h4_3

E. Traffic test from h1_1 to h4_3 with multiple sessions

a. Start IPERF server in h4_3
iperf -s &

b. Start the IPERF client in h1_1 and connecting to h4_3, (see Fig. 10).
iperf -c h4_3 -P 5

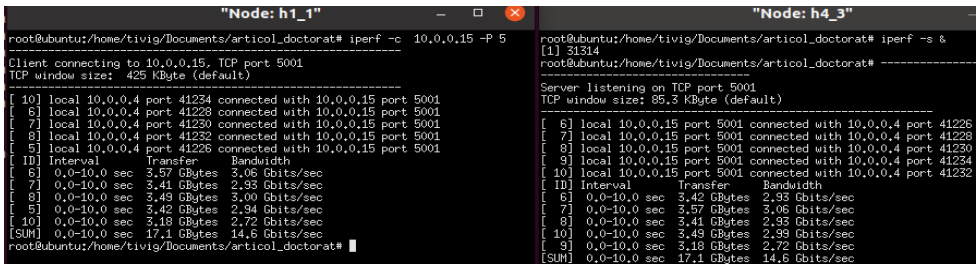

Fig. 10. Traffic test results with multiple sessions

II. UDP tests with IPERF

The UDP test allow to test different packet size variation (64 bytes) and number of packets (10 Packets/s) that can send. Allow a granular control of the bandwidth of UDP traffic. The UDP test allow to measure the Packet loss, latency and bandwidth. One of the downsides of IPERF is that it does not provide increase granularity in UDP test.

a. Start IPERF UDP server in h4_3 (see Fig. 11)

iperf -u -s &

b. Run the IPERF client in h1_1 and connecting to h4_3 and generate bandwidth of 15Mbsp (see Fig. 11)

iperf -u -c h4_3 -b 15m

Frequently we need to execute traffic tests between the hosts to test our network topology. A part of the test may include:

- VoIP traffic test,
- Video streaming mp4 video test using VLC media player [21],
- Various/parallel streams/sessions,
- Variable traffic rate (20Pkts/s for first 60s, then 200Pkts/s for the rest of the test),
- TCP test between the any two hosts,
- UDP test.

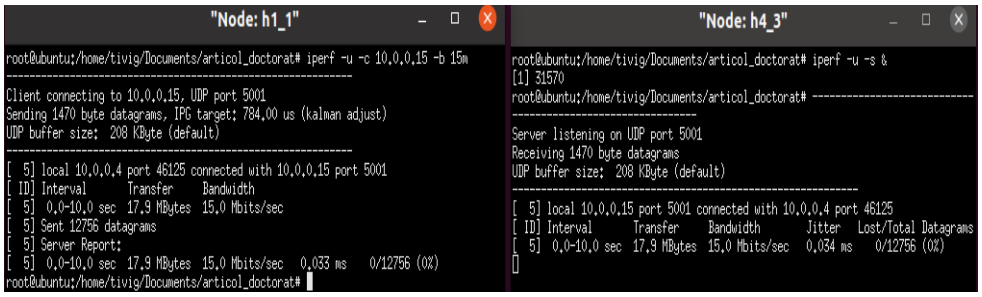

Fig. 11. UDP traffic test

III. VoIP Tests

The VoIP requires a reliable connection to carry a significant quantity of UDP packets among of the presence VoIP points with minimal packet loss, delay. If there is encountered a decrease in packet loss, delay or available bandwidth can perturb the quality of the VoIP calls.

Usually, the bandwidth necessary for traffic voice over IP is 64 kbps. To have a concluded test between two VoIP end point the test must send enough ICMP ping packets into bidirectional flow of UDP packets having a determinant interval and payload to be sent on long time frames. We will set the ToS bit, which represents the quality of service into the IP field. The starting of the punctual testing implies turning on the server placed on the first open flow switch from the topology, switch s1, as can we see from Fig. 2. The proposed test is bidirectional test. As soon as the server will receive the command to start the test, it will generate the 64kbp of traffic to the client. The first objective is to test a first VoIP call of 64Kbps. The second objective is to test parallel VoIP calls.

A. Single 64Kbps voice call test

Begin the 64Kbps VoIP traffic test between h1_ 1to h4_ 3 for 60 seconds and backwards.

a. Run the IPERF UDP server in h4_3, as can bee seen from Fig. 12

From node h4_3 we issue the command (see right side of the Fig. 12):
iperf --server --udp --len 300 --tos 184 -fk --interval 5 &

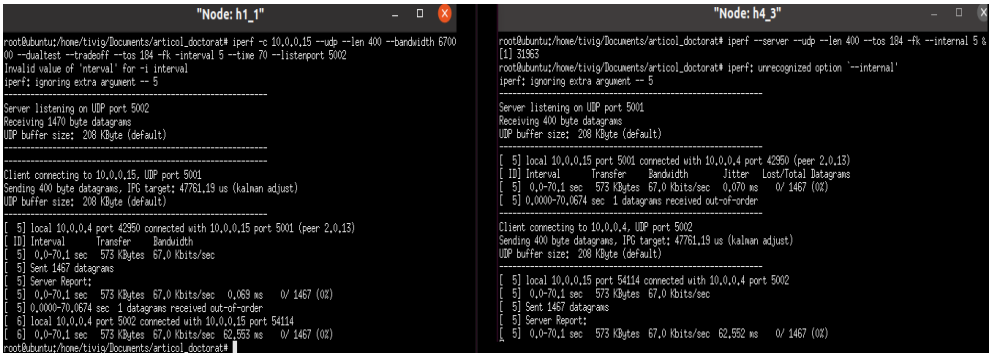

Fig. 12. Result for a single 64Kbps voice call test

The options are explained below:
- h4_3 source host,
- server run iperf in server mode,
- udp protocol,
- len the length of buffers to read or write,
- tos type-of-service for outgoing packets,
- fk the format to print the bandwidth numbers in. Here the 'k' = Kbits/sec is used,
- interval sets the interval time in seconds between periodic bandwidth, jitter, and loss reports.

b. Run the IPERF UDP client in h1_1(see the left side of the Fig. 12)

In mininet CLI we issue the below set of traffic parameters:
h1_1 iperf -c 10.1.1.12 --udp --len 300 --bandwidth 67000 --dualtest --tradeoff --tos 184 -fk --interval 5 --time 60 --listenport 5002

B. Multiple Parallel calls VoIP calls test

Run the IPERF UDP server in h4
mininet>h4 iperf --server --udp --len 300 --tos 184 -fk --interval 5 --parallel 4
mininet>iperf -c 10.1.1.4 --udp --len 300 --bandwidth 67000 - --tradeoff --tos 184 -fk --interval 5 --time 60 --listenport 5002 --parallel 4

The options are explained below:
- tradeoff means that the server will connect back to the client,

- interval means the interval time in seconds between periodic bandwidth, jitter, and loss reports,
- listenport specifies the port that the server will connect back to the client on,
- parallel number of simultaneous connections to make to the server.

The graphical results from multiple parallel VoIP calls tests can be see into Fig. 13.

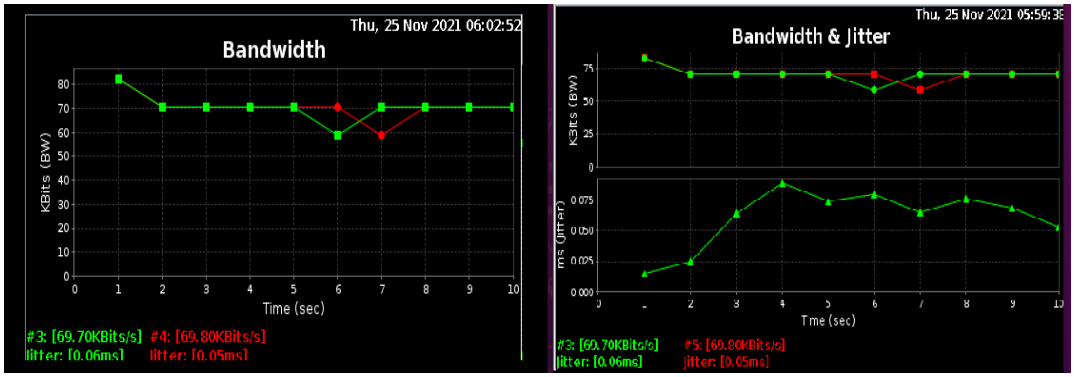

Fig. 13. Results from multiple parallel VoIP calls test

The left side of the figure is the result measured at the client side (h1_1) whereas the right side of the figure are the results taken at the other end of the connection (h4_3).

6. Conclusions

This paper exposes couple SDN experiments using RYU SDN controller, IPERF, Python programing language for instantiating the network topology and Mininet framework in which was tested the different QoS parameters for voice and data traffic with different number of flows requirements for voice and data. The work reports traffic statistics of current testbed, number of packets, number of calls and average bitrate for the voice and data.

For developing this topology, the Mininet framework was chosen: complex framework, well-known platform into the research industry to develop and test SDN multiple topologies.

A future development and testing work will be done using more complex topologies and involving this time real case scenarios for multicasting video streaming into a testing medium with heavily loaded networks to discover pre-roll delays that could appear in such traffic patterns in case of video on demand. Following this direction, after finishing the proposed experiments and testing the Mininet framework capabilities at scale, the final scope will be to propose new ways to improve video streaming transmission.

REFERENCES

[1]. *T. Hu, Z. Guo, P. Yi, T. Baker and J. Lan,* Multi-controller Based Software-Defined Networking: A Survey IEEE, 2018.
[2]. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual- internet-report/white-paper-c11-741490.html.
[3]. *P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race,* Towards network-wide QoE fairness using openflow-assisted adaptive video streaming, in Proceedings of the 2013 ACM SIGCOMM workshop on Future human-centric multimedia networking, ACM, 2013, pp. 15-20.
[4]. *P. Panwaree, J. W. Kim and C. Aswakul,* Packet Delay and Loss Performance of Streaming Video over Emulated and Real OpenFlow Networks, 29th International Technical Conference on Circuit/Systems, Computers and Communications (ITC-CSCC), Phuket, Thailand, July 1-4, 2014, pp. 777- 779.
[5]. *Y. Nakagawa, K. Hyoudou, and T. Shimizu,* A management methodof IP multicast in overlay networks using openflow, in Proc. Hot Topics in Software Defined Networks (HOtSDN). Helsinki: ACM, 2012, pp. 91–96.
[6]. *D. Kotani, K. Suzuki, and H. Shimonishi,* A design and implementation of OpenFlow controller handling IP multicast with fasttree switching, in Proc. IEEE Symp. Applicat. and the Internet (SAINT), Izmir, July 2012, pp. 60–67.
[7]. *M. Jarschel, F. Wamser, T. Hohn, T. Zinner, and P. Tran-Gia,* SDN based application-aware networking on the example of youtube video streaming, in Proc. European Workshop on Software Defined Networks (EWSDN), Berlin, Oct. 2013, pp. 87–92.
[8]. *S. Ramakrishnan et al.,* SDN Based QoE Optimization for HTTP Based Adaptive Video Streaming, IEEE International Symposium on Multimedia (ISM), December 2015.
[9]. ISO/IEC JTC1/SC29/WG11 (MPEG) Report, Technical report, ISO, November 2013.
[10]. *H. Babbar and S. Rani*, "Emerging prospects and trends in software-defined networking," Journal of Computational and Theoretical Nanoscience, Vol.16, pp. 4236–4241, 2019.
[11]. *H. Babbar, Himanshi & Rani, Shalli,* Performance Evaluation of QoS metrics in Software Defined Networking using Ryu Controller, IOP Conference Series: Materials Science and Engineering, 2021.
[12]. https://datatracker.ietf.org/doc/html/rfc4960.
[13]. http://das t.nlanr.net/Projects /Iperf.
[14]. http://mininet.org/.
[15]. https://www.openvswitch.org/.
[16]. https://www.python.org.
[17]. https://ryu-sdn.org/.
[18]. https://man7.org/linux/man-pages/man7/namespaces.7.html.
[19]. *T. P. Tivig, E. Borcoci,* Layer 3 Forwarder Application – Implementation Experiments Based on Ryu SDN Controller, Dubai, International Symposium on Networks, Computers and Communication, Dubai-UAE, October 31- November 2, 2021.
[20]. https://datatracker.ietf.org/doc/html/rfc4337.
[21]. "VLC Media Player" [Online]. Available: https://www.videolan.org/vlc/index.html.