COMP3004 REPORT

An investigation into how network conditions/impairments affect end-to-end video streaming quality.

Module Leader: Lingfen Sun

Alistair Drew
10567887

CONTENTS

Abstract 2
Introduction and background 2
SDN 2
Dash 2
Video Streaming Quality 3
Steps Taken 3
Report Structure 3
Testbed and development 4
Setup – general structure of the testbed and it’s configuration 4
Ubuntu Desktop VM (new 18.4) setup 4
Ubuntu Live server VM (ODL) setup 6
Code developed. 7
Experiments, Results and Analysis 8
Discussions, network management via REST API 10
Problems encountered and going forward 10
Network Management via REST API 11
Conclusions 13
References 14
Appendix 15

ABSTRACT

Extremely popular HTTP adaptive video streaming services over the internet such as Over the Top solutions (OTT) like YouTube, Netflix and Twitch must maintain the Quality of Experience of their end-to-end video streaming to remain popular. Due to this Video Service Providers (VSP's) need to investigate how network conditions, the bottom level of the operation effect the Quality of Experience for the user. This report is an investigation into the effect network conditions have on the end-to-end video streaming quality. Software Defined Networks (SDN) used allow for increased flexibility of the network by making it easy to manage and adapt how traffic is handled. Mininet acts as a Network Function Virtualization (NFV) that allows abstraction of network functions from dedicated hardware to standard hardware allowing for flexible network topology creation.

INTRODUCTION AND BACKGROUND

SDN

Proprietary hardware was king in the networking world until the arrival of software-defined networking (SDN), an approach which can both simplify and allow granular control of networks adaptively to adjust for network conditions. This facilitates network configuration and management (Bonfim, Dias and Fernandes, 2019). This makes it especially well-suited for testing the effects network (QoS) parameters have on application (QoS) parameters and in turn the users Quality of Experience (QoE) as shown in Figure 1 (Mok, Chan and Chang, 2011).

SDN is done through 'programming' the network via a centralised controller located in the control plane, the SDN controller makes the decisions on how network traffic is controlled. The data layer which contains the network nodes like switches and hosts are responsible for forwarding traffic according to a set of rules created by the SDN controller. The SDN controller has direct control of the data layer through the southbound protocol OpenFlow. It also interacts with the application layer via the northbound protocol usually via a REST API like ODL.

The purpose of using SDN is to allow easier network management and greater flexibility of traffic flows while also abstracting the implementation of the control plane to non-proprietary software. This makes development and testing of environments like the testbed much more accessible.

DASH

Adaptive Bitrate Streaming (ABR) is part of HTTP streaming protocols like dash, it monitors the state of the viewers network and adapts the quality of the stream accordingly (DASH reference). DASH comprises of media files divided into modular chunks between 2 to 10 seconds called segments. Each segment is encoded into multiple versions of bit-rate quality levels and depending on the frame rate, bit rate and resolution the quality level can change. The details of the different quality levels are then stored in an XML manifest called a Media Presentation Description file (MPD). Within an MPD file the segments are arranged in adaption sets that contain a set of representations such as bitrate or resolutions. These representations can hold interchangeable versions of media content such as different resolutions or bitrates. By having multiple representation sets in a single MPD file it enables the client to adapt the media stream to the current requirements for network bandwidth by interchanging between these representation sets of differing qualities.

VIDEO STREAMING QUALITY

Adaptive Bitrate Streaming (ABR) it can be measured in two ways. The objective metrics are starting bitrate, ending bitrate, initial delay, average buffer frequency and average buffer length. This is a mix of metrics taken from the DASHjs interface and application Quality of Service (QoS) metrics. Objective metrics are recorded automatically via computer programs and subjectively metrics are a matter of user opinion recorded using subjective mean opinion scores (MOS). The subjective MOS score is the degree of delight or annoyance of the user evaluating the video, ranging from excellent to very annoying.

Score	Quality	Impairment
5	Excellent	Imperceptible
4	Good	Perceptible but not annoying
3	Fair	Slightly annoying
2	Poor	Annoying
1	Bad	Very annoying

STEPS TAKEN

The investigation can be divided into a series of steps to be taken sequentially:

1. The first of these steps is to research into the background of ABR and how network conditions effect the QoE of end-to-end video streaming.
2. The second step is to use the research to choose suitable technologies to build the test bed.
3. The third step is to create the test bed itself.
4. The fourth step is to design experiments that measure the correct metrics (packet loss, delay, bandwidth, initial delay, average buffering length, buffering frequency and subjective MOS).
5. The fifth step is to carry out the experiments, then analyse and critically evaluate the results.
6. The sixth step is to summarize the key points and conclude the report.

REPORT STRUCTURE

The structure of the report is split into five parts, section one is the introduction to content of the report and the background information as well as pertinent literature. Section two details and justifies the design of the test bed and its development. Section three explains and justifies the experiments carried out on the test bed as well as analyses the results from the experiments. Section four is about critical evaluation of the report so far as well as the network management via REST API. Finally, section five is the conclusion that is an executive summary of the report.

TESTBED AND DEVELOPMENT

SETUP – GENERAL STRUCTURE OF THE TESTBED AND IT'S CONFIGURATION

The testbed was built on an SDN architecture, this means that it can be split into different layers. Referring to Figure 2 in the appendix you can see that the testbed can be split into three clear layers. The control layer which uses ODL as its controller, the data layer is created adaptively by Mininet (NFV) in this case the remote controller, an OpenvSwitch and the 4 hosts. Finally, the application layer, consisting of the DASH server which is Apache2 and the DASH client which is DASHjs.

The testbed was set up on two Virtual Machines, one built with a 18.04.5 Ubuntu desktop image, the other on the 18.04.5 Ubuntu live server image. Notably ODL was installed on the live server image and accessed as a remote controller from the Ubuntu desktop image. The steps as follows are how the VM's were set up:

1. Start up the new VM in VMware Workstation.
2. Select the needed ISO image for the operating system of the new VM. In this case either the desktop or live server image.
3. Create the login details for the new VM.
4. Set the name of the new VM to either ODL or new 18.4.
5. Setting the disk capacity of the new VM
6. Customising the hardware by setting it to a bridged connection
7. Finishing the settings and let it setup.

The setup of the testbed was automated using a two script files, below the different sections of the script are explained.

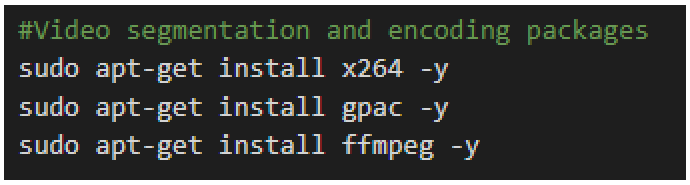
UBUNTU DESKTOP VM (NEW 18.4) SETUP

The script is set up so that packages are installed first, it then goes onto installing necessary parts of the testbed in order, then goes onto doing any 'action that needs to happen. This ensures that the script has all the packages and resources it needs to carry out the 'actions' smoothly.

Packages for testbed setup are split into three sections. The first include general management packages, which allow the manipulation of files, either via downloading or for general management of the VM. The second is the Apache2 package that is going to act as the DASH client. Finally, the packages that are used for processing the video are grouped together.

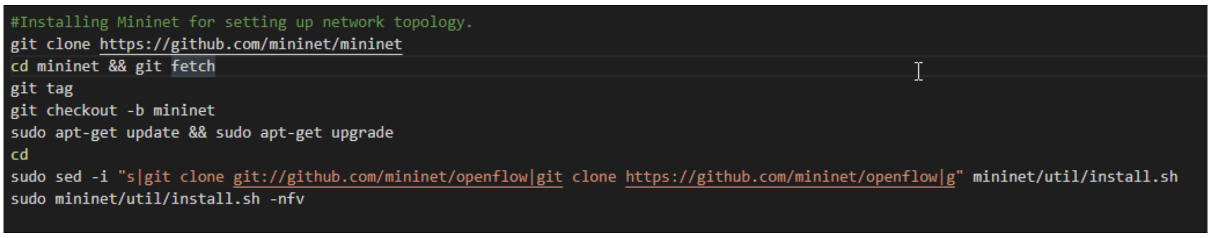
```
#General management packages
sudo apt-get install openssh-server -y
sudo apt-get install unzip -y
sudo apt-get install curl -y
sudo apt-get install tree -y
sudo apt-get install git -y
```

```
#Packages for webserver hosting to act as DASH Server.
sudo apt-get install apache2 -y
```



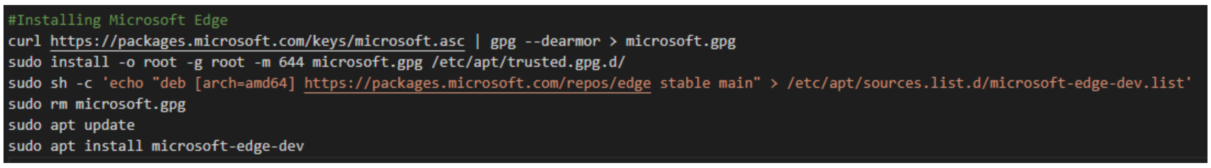
```
#Video segmentation and encoding packages
sudo apt-get install x264 -y
sudo apt-get install gpac -y
sudo apt-get install ffmpeg -y
```

Mininet is installed for adaptively creating the network, it creates the data layer (controller, switch, and hosts) based on a python file.



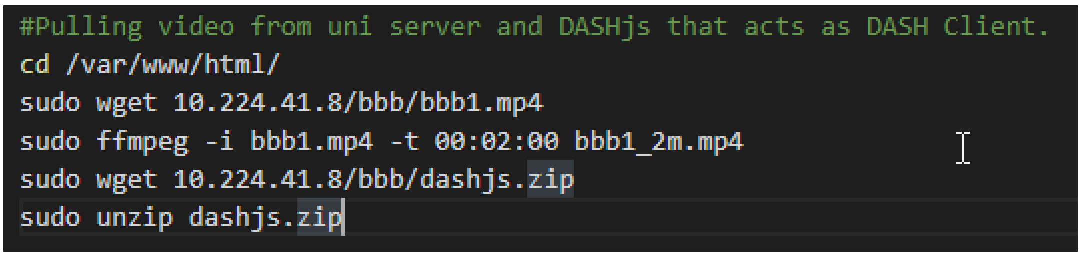
```
#Installing Mininet for setting up network topology.
git clone https://github.com/mininet/mininet
cd mininet && git fetch
git tag
git checkout -b mininet
sudo apt-get update && sudo apt-get upgrade
cd
sudo sed -i "s|git clone git://github.com/mininet/openflow|git clone https://github.com/mininet/openflow|g" mininet/util/install.sh
sudo mininet/util/install.sh -nfv
```

Microsoft Edge is installed as it is a browser that runs on less CPU resources. This is important later in the discussion section.



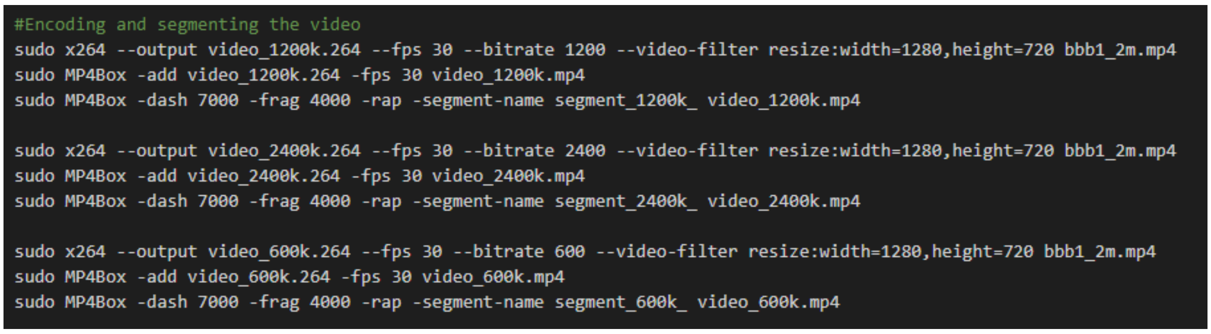
```
#Installing Microsoft Edge
curl https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor > microsoft.gpg
sudo install -o root -g root -m 644 microsoft.gpg /etc/apt/trusted.gpg.d/
sudo sh -c 'echo "deb [arch=amd64] https://packages.microsoft.com/repos/edge stable main" > /etc/apt/sources.list.d/microsoft-edge-dev.list'
sudo rm microsoft.gpg
sudo apt update
sudo apt install microsoft-edge-dev
```

This section of the script pulls DASHjs and the video from university server to act as the DASH Client. It clips the video to 2 minuets for testing purposes.



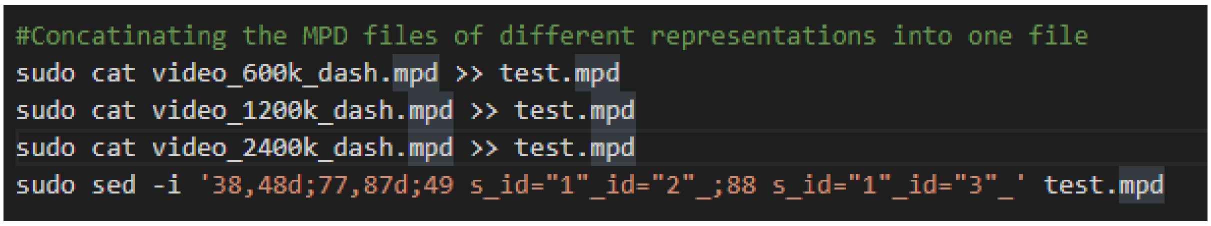
```
#Pulling video from uni server and DASHjs that acts as DASH Client.
cd /var/www/html/
sudo wget 10.224.41.8/bbb/bbb1.mp4
sudo ffmpeg -i bbb1.mp4 -t 00:02:00 bbb1_2m.mp4
sudo wget 10.224.41.8/bbb/dashjs.zip
sudo unzip dashjs.zip
```

Moving onto the ‘action’ section of the script, it then segments and encodes the clipped video into different adaption sets in multiple MPD files.



```
#Encoding and segmenting the video
sudo x264 --output video_1200k.264 --fps 30 --bitrate 1200 --video-filter resize:width=1280,height=720 bbb1_2m.mp4
sudo MP4Box -add video_1200k.264 -fps 30 video_1200k.mp4
sudo MP4Box -dash 7000 -frag 4000 -rap -segment-name segment_1200k_ video_1200k.mp4

sudo x264 --output video_2400k.264 --fps 30 --bitrate 2400 --video-filter resize:width=1280,height=720 bbb1_2m.mp4
sudo MP4Box -add video_2400k.264 -fps 30 video_2400k.mp4
sudo MP4Box -dash 7000 -frag 4000 -rap -segment-name segment_2400k_ video_2400k.mp4

sudo x264 --output video_600k.264 --fps 30 --bitrate 600 --video-filter resize:width=1280,height=720 bbb1_2m.mp4
sudo MP4Box -add video_600k.264 -fps 30 video_600k.mp4
sudo MP4Box -dash 7000 -frag 4000 -rap -segment-name segment_600k_ video_600k.mp4
```

Now the multiple MPD files are concatenated into one MPD file to access via DASHjs. This enables adaptive bitrate streaming (ABR)

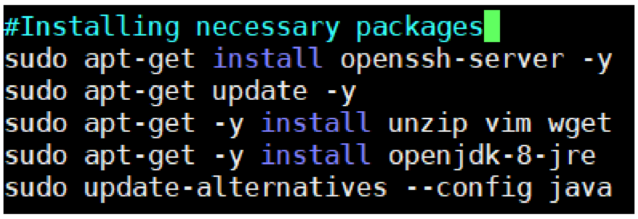


```
#Concatinating the MPD files of different representations into one file
sudo cat video_600k_dash.mpd >> test.mpd
sudo cat video_1200k_dash.mpd >> test.mpd
sudo cat video_2400k_dash.mpd >> test.mpd
sudo sed -i '38,48d;77,87d;49 s_id="1"_id="2"_;88 s_id="1"_id="3"_' test.mpd
```

The network topology for Mininet is created via a python file called 'network.py'. This is further explained in the code developed section.

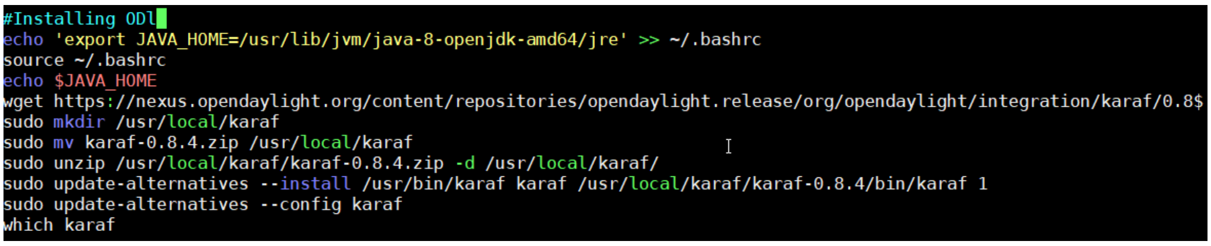
UBUNTU LIVE SERVER VM (ODL) SETUP

Like the previous script the packages are installed first for so ODL can be installed correctly.



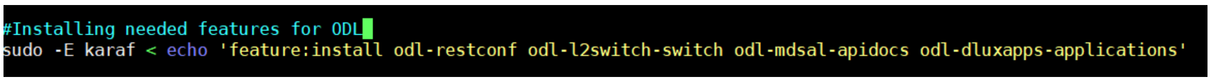
```
#Installing necessary packages
sudo apt-get install openssh-server -y
sudo apt-get update -y
sudo apt-get -y install unzip vim wget
sudo apt-get -y install openjdk-8-jre
sudo update-alternatives --config java
```

Then ODL is Installed from the packages.



```
#Installing ODL
echo 'export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre' >> ~/.bashrc
source ~/.bashrc
echo $JAVA_HOME
wget https://nexus.opendaylight.org/content/repositories/opendaylight.release/org/opendaylight/integration/karaf/0.8$
sudo mkdir /usr/local/karaf
sudo mv karaf-0.8.4.zip /usr/local/karaf
sudo unzip /usr/local/karaf/karaf-0.8.4.zip -d /usr/local/karaf/
sudo update-alternatives --install /usr/bin/karaf karaf /usr/local/karaf/karaf-0.8.4/bin/karaf 1
sudo update-alternatives --config karaf
which karaf
```

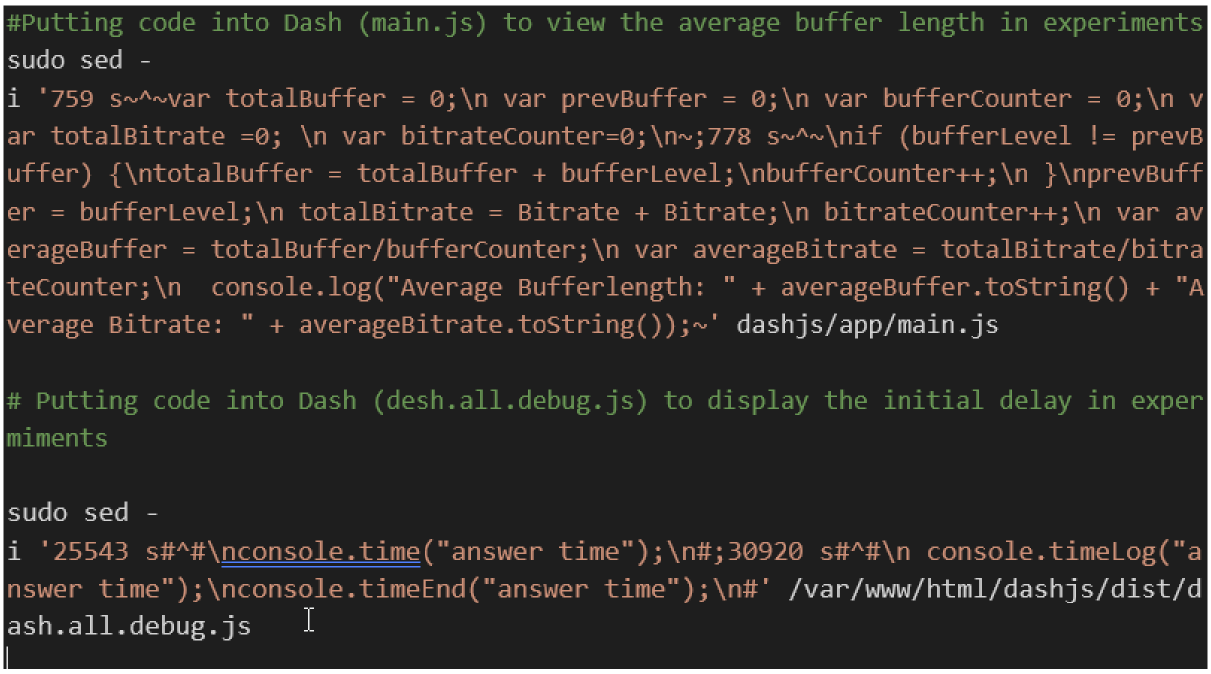
Finally running ODL and installing the necessary features needed. This completely sets up ODL Oxygen to act as the remote controller.



```
#Installing needed features for ODL
sudo -E karaf < echo 'feature:install odl-restconf odl-l2switch-switch odl-mdsal-apidocs odl-dluxapps-applications'
```

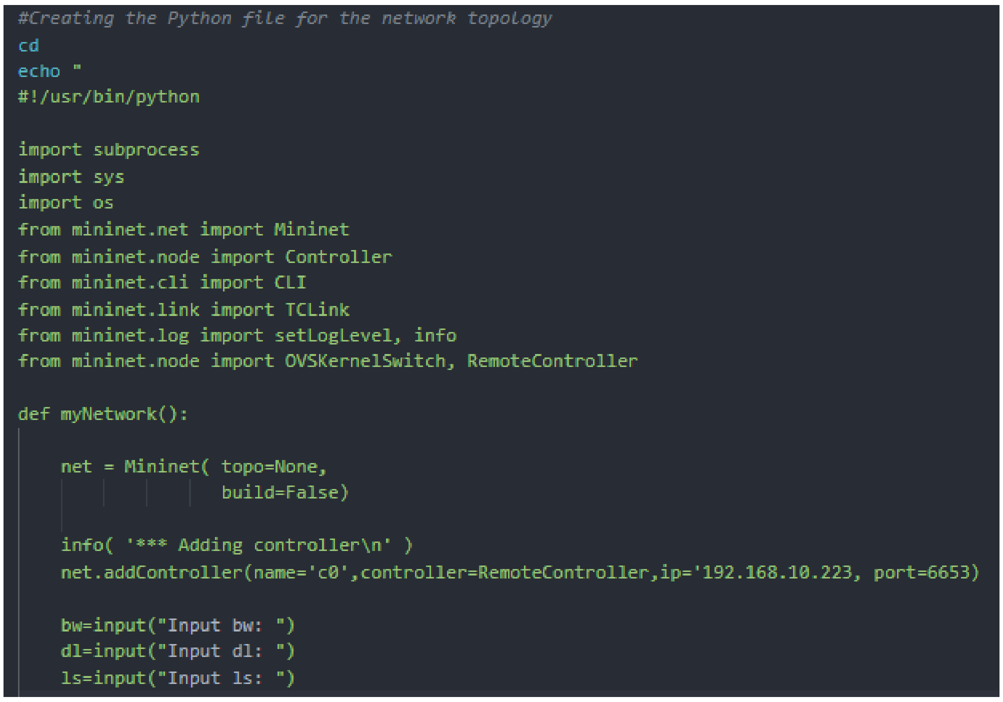
CODE DEVELOPED.

Additional code developed for taking more accurate metrics and display them in the console in inspect element. The code edits DASHjs itself so that calculations for QoS application metrics (initial delay, average buffer event and average buffer time) are outputted to the console in the inspect element in Microsoft Edge.

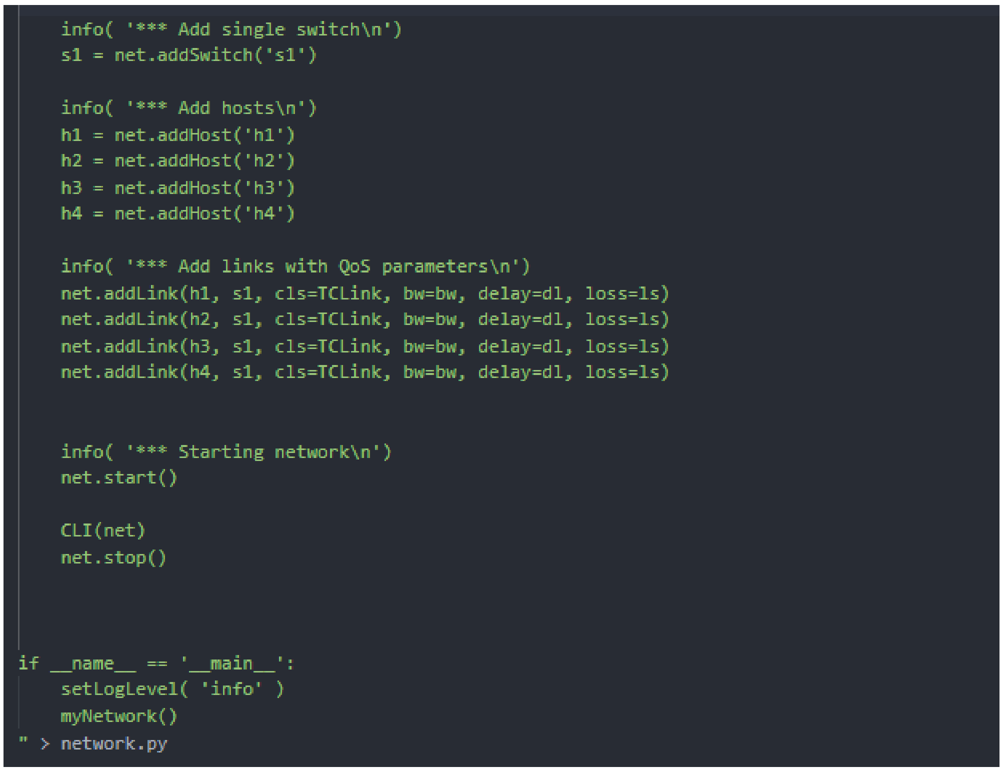


```
#Putting code into Dash (main.js) to view the average buffer length in experiments
sudo sed -
i '759 s~^~var totalBuffer = 0;\n var prevBuffer = 0;\n var bufferCounter = 0;\n v
ar totalBitrate =0; \n var bitrateCounter=0;\n~;778 s~^~\nif (bufferLevel != prevB
uffer) {\ntotalBuffer = totalBuffer + bufferLevel;\nbufferCounter++;\n }\nprevBuff
er = bufferLevel;\n totalBitrate = Bitrate + Bitrate;\n bitrateCounter++;\n var av
erageBuffer = totalBuffer/bufferCounter;\n var averageBitrate = totalBitrate/bitra
teCounter;\n  console.log("Average Bufferlength: " + averageBuffer.toString() + "A
verage Bitrate: " + averageBitrate.toString());~' dashjs/app/main.js

# Putting code into Dash (desh.all.debug.js) to display the initial delay in exper
miments

sudo sed -
i '25543 s#^#\nconsole.time("answer time");\n#;30920 s#^#\n console.timeLog("a
nswer time");\nconsole.timeEnd("answer time");\n#' /var/www/html/dashjs/dist/d
ash.all.debug.js
```

This python code is integrated into the setup script for the desktop VM. It works with Mininet to set up the network data layer, and allows the user to adaptively decide how much the QoS network metrics (bandwidth, delay, and packet loss) are impacting the created network whenever its ran. This allows for easier testing without changing the source code which could compromise reliability of the testbed.



```
#Creating the Python file for the network topology
cd
echo "
#!/usr/bin/python

import subprocess
import sys
import os
from mininet.net import Mininet
from mininet.node import Controller
from mininet.cli import CLI
from mininet.link import TCLink
from mininet.log import setLogLevel, info
from mininet.node import OVSKernelSwitch, RemoteController

def myNetwork():

    net = Mininet( topo=None,
                   build=False)

    info( '*** Adding controller\n' )
    net.addController(name='c0',controller=RemoteController,ip='192.168.10.223, port=6653)

    bw=input("Input bw: ")
    dl=input("Input dl: ")
    ls=input("Input ls: ")
```



```
    info( '*** Add single switch\n')
    s1 = net.addSwitch('s1')

    info( '*** Add hosts\n')
    h1 = net.addHost('h1')
    h2 = net.addHost('h2')
    h3 = net.addHost('h3')
    h4 = net.addHost('h4')

    info( '*** Add links with QoS parameters\n')
    net.addLink(h1, s1, cls=TCLink, bw=bw, delay=dl, loss=ls)
    net.addLink(h2, s1, cls=TCLink, bw=bw, delay=dl, loss=ls)
    net.addLink(h3, s1, cls=TCLink, bw=bw, delay=dl, loss=ls)
    net.addLink(h4, s1, cls=TCLink, bw=bw, delay=dl, loss=ls)


    info( '*** Starting network\n')
    net.start()

    CLI(net)
    net.stop()



if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()
" > network.py
```

EXPERIMENTS, RESULTS AND ANALYSIS

The experiments were designed to test and outline the effects that network Quality of Service (QoS) parameters effect application QoS parameters and in turn user Quality of Experience (QoE). This is because the investigation hinges on how each of these sets of parameters impact each other to ultimately effect end-to-end video streaming quality. Figure 1 visually outlines how each level of QoS parameters impact each other.

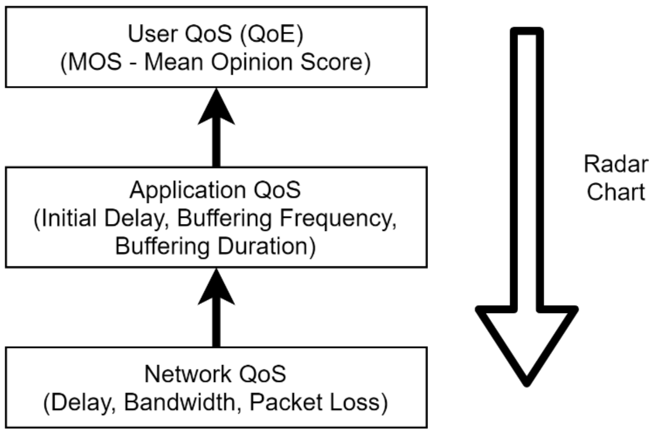

User QoS (QoE)
(MOS - Mean Opinion Score)
Application QoS
(Initial Delay, Buffering Frequency, Buffering Duration)
Network QoS
(Delay, Bandwidth, Packet Loss)
Radar Chart

Figure 1 - Different levels of Quality-of-Service Metrics (Mok, Chan and Chang, 2011)

The experiments consisted of testing for 6 measurements split between objective and subjective measurements, they are Starting Bitrate, Ending Bitrate, average initial delay, average buffer frequency, average buffer length and subjective MOS. The starting and ending bitrates shows how the network adapts to different network conditions. Tested average initial delay, tested average buffer frequency, and tested average buffer length are application QoS metrics that outline how the network QoS metrics have impacted the application QoS metrics and in turn user QoE. To ensure accuracy of results each experiment has been repeated four times and averaged. The subjective MOS was a measurement directly based on the user QoE, and by taking it alongside the objective metrics you can view the correlation on how certain trends of objective data effect the user QoE.

The network conditions tested include three levels of packet loss across three levels of bandwidth. Figures 11 and 12 show the full list of results, Figure 2 outlines selected data from the experiments that show the correlation between the network QoS impairments and how they affect end-to-end video streaming quality.

Experiment/Network Conditions	Starting Bitrate	Ending Bitrate	Tested Average Initial Delay (s)	Tested Average Buffer frequency	Tested Average Buffer Length (s)	Subjective MOS
Experiment 1.1 --> 1Mbps - 0ms - 0%	600	600	0.201	0	14.775	2
Experiment 1.3 --> 1Mbps - 0ms - 8%	600	600	4.6	11	3.95	1
Experiment 3.1 --> 10Mbps - 0ms - 0%	2400	2400	0.236	0	25.125	4
Experiment 3.3 --> 10Mbps - 0ms - 8%	600	600	0.171	9.75	4.525	1

Figure 2 - Results table culminating the core results of my experiments.

As indicated by the small bandwidth tests experiments 1.1 and 1.3, the video streaming has limited throughput, this is shown by the shorter buffer length in the lower bandwidth tests. The smaller buffer length leads the video to micro buffer and as a result makes the video look jumpy in places as the video struggles to buffer video smoothly. As a result, the adaptive bitrate streaming (ABR) adapts to a lower bitrate to compensate, naturally this causes the quality of the video to suffer and as a result the subjective MOS score to be low. This means that a small bandwidth does not have the resources to support high user QoE.

High bandwidth test such as experiment 3.1 shows an increased buffer length, bitrates and subjective MOS score. This indicates that with a higher bandwidth the video buffer length increases meaning it can run the video more smoothly and the ABR does not have to adjust the bitrate to compensate. This leads to a high subjective MOS score and a better user QoE.

High Packet loss applied to both bandwidths in experiments 1.3 and 3.3 severely impairs the levels of all the measured metrics, this is because it leads to a large amount of the packets having to be retransmitted. It immediately causes the ABR to adapt the bitrate to the lowest setting due to the extreme trouble it has buffering the video with such low buffer lengths. The biggest issue is the buffer events which were previously not happening become common, these events ruin the user QoE as it becomes extremely annoying to wait through multiple buffering events.

In summery high bandwidth is needed to maintain a high level of end-to-end video streaming quality however if high levels of packet loss are introduced it can impair the operation of the network and ruin user QoE. Additionally, by itself low bandwidth is enough to cause low quality end-to-end streaming, however with added packet loss it makes a bad situation worse and leads to extremely bad quality video steaming. It is worth noting that in additional experiments low levels of packet loss did not significantly impair video streaming quality when there was a high bandwidth.

DISCUSSIONS, NETWORK MANAGEMENT VIA REST API

PROBLEMS ENCOUNTERED AND GOING FORWARD

So far, the investigation has been quite extensive, and of course there have been several challenges have been encountered. The 3 biggest areas of obstacles being, my initial skill level, issues with setting up the testbed and performance issues with Mininet.

I had initially fully set up the testbed on a VM made with the 16.4 Ubuntu VM, the only issue being that when running Mininet on it the browser ran extremely slow and was lagging to the point of interfering with recording experiment results. Fixes such as installing and using a browser that was lighter on CPU resources helped but changes such as shifting to a VM made with the 18.04.5 Desktop image fixed the issue as it drastically improved performance.

Multiple times across the coursework I ran into issues when setting up the testbed where different packages such as ODL worked fine on the 16.4 Ubuntu VM but when installed on the new 18.04.5 Desktop VM it stopped working due to being incompatible. As a result, fixes like having to use RYU as the controller were used, however these also came with issues such as the RYU REST API GUI being insufficient compared to ODL. Consequently, work arounds like figuring out how to install a newer version of ODL took a long time due to my initially low skill level.

Going forward if I were to create the test environment again, I would avoid trying to implement VM's and instead use software such as Docker that would allow for rapid creation of similar and containerized environments. Having some experience with Docker before this it was much easier to use because it mitigated the incompatibility issues I otherwise encountered in this investigation.

NETWORK MANAGEMENT VIA REST API

This is the ODL REST API GUI that shows details about the network from the controller information such as the network topography, the nodes and their related flow tables can be accessed from the ODL GUI.

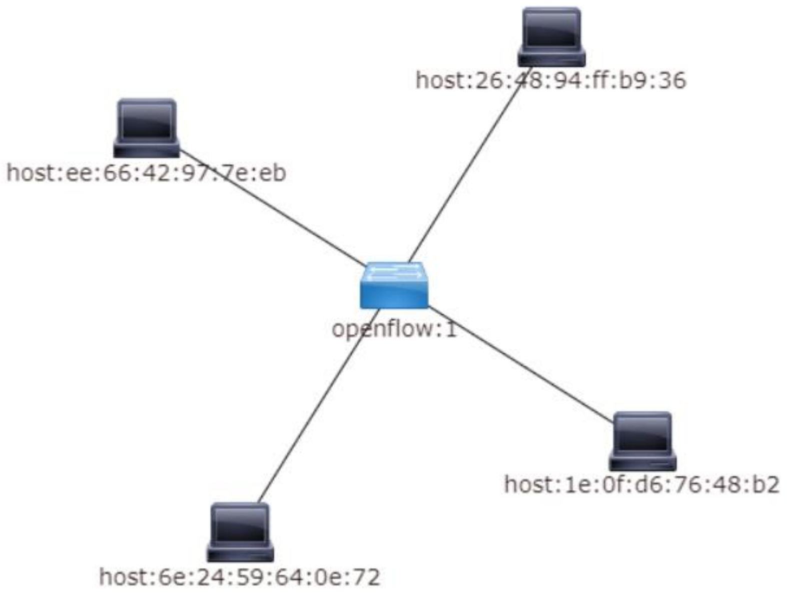
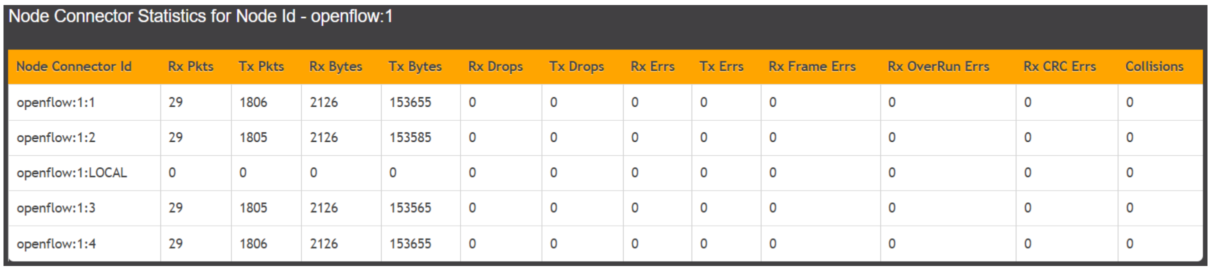
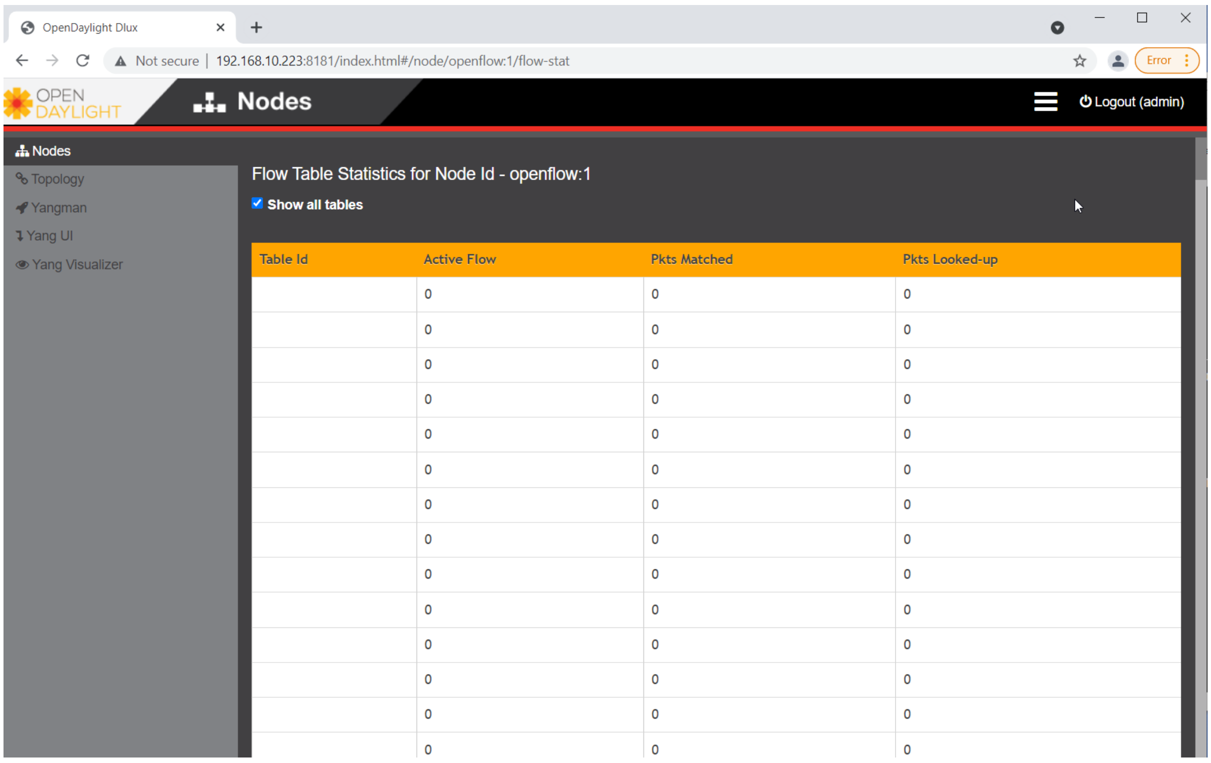


Figure 3 - Network Topology show by ODL GUI


Node Connector Statistics for Node Id - openflow:1

Node Connector Id	Rx Pkts	Tx Pkts	Rx Bytes	Tx Bytes	Rx Drops	Tx Drops	Rx Errs	Tx Errs	Rx Frame Errs	Rx OverRun Errs	Rx CRC Errs	Collisions
openflow:1:1	29	1806	2126	153655	0	0	0	0	0	0	0	0
openflow:1:2	29	1805	2126	153585	0	0	0	0	0	0	0	0
openflow:1:LOCAL	0	0	0	0	0	0	0	0	0	0	0	0
openflow:1:3	29	1805	2126	153565	0	0	0	0	0	0	0	0
openflow:1:4	29	1806	2126	153655	0	0	0	0	0	0	0	0

Figure 4 - Nodes shown by ODL GUI



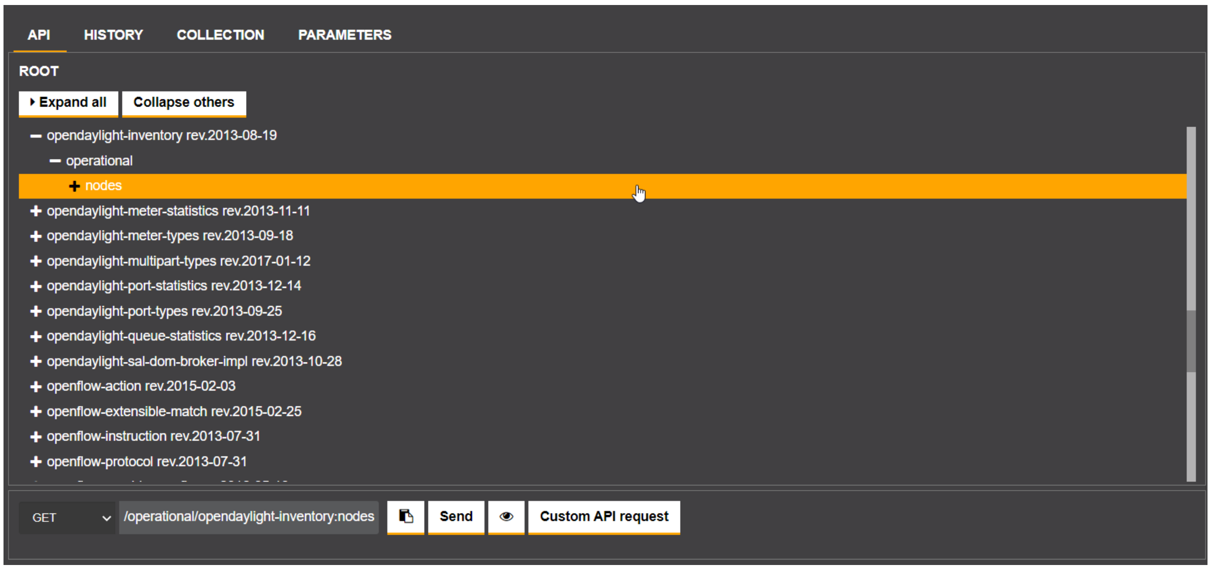
Flow Table Statistics for Node Id - openflow:1

Show all tables

Table Id	Active Flow	Pkts Matched	Pkts Looked-up
	0	0	0
	0	0	0
	0	0	0
	0	0	0
	0	0	0
	0	0	0
	0	0	0
	0	0	0
	0	0	0
	0	0	0
	0	0	0
	0	0	0
	0	0	0
	0	0	0

Figure 5 - Nodes - extended flow table

Due to SDN architecture, the network is managed from a centralized controller. The ODL GUI allows the user to access information in the controller about the northbound and southbound protocols, allowing them to see data from the switch and hosts in the data layer. The user can manipulate the controller from the GUI and as such manage the network using the centralized controller. An example being if the network had four hosts and two of them were idle the user could access the ODL GUI to disable the two hosts through the controller, allowing for spare bandwidth not being used by the idle hosts to be used by the hosts that are active. Improving



```
— opendaylight-inventory rev.2013-08-19
    — operational
        + nodes
+ opendaylight-meter-statistics rev.2013-11-11
+ opendaylight-meter-types rev.2013-09-18
+ opendaylight-multipart-types rev.2017-01-12
+ opendaylight-port-statistics rev.2013-12-14
+ opendaylight-port-types rev.2013-09-25
+ opendaylight-queue-statistics rev.2013-12-16
+ opendaylight-sal-dom-broker-impl rev.2013-10-28
+ openflow-action rev.2015-02-03
+ openflow-extensible-match rev.2015-02-25
+ openflow-instruction rev.2013-07-31
+ openflow-protocol rev.2013-07-31
```

/operational/opendaylight-inventory:nodes

Figure 6 - API's in ODL GUI

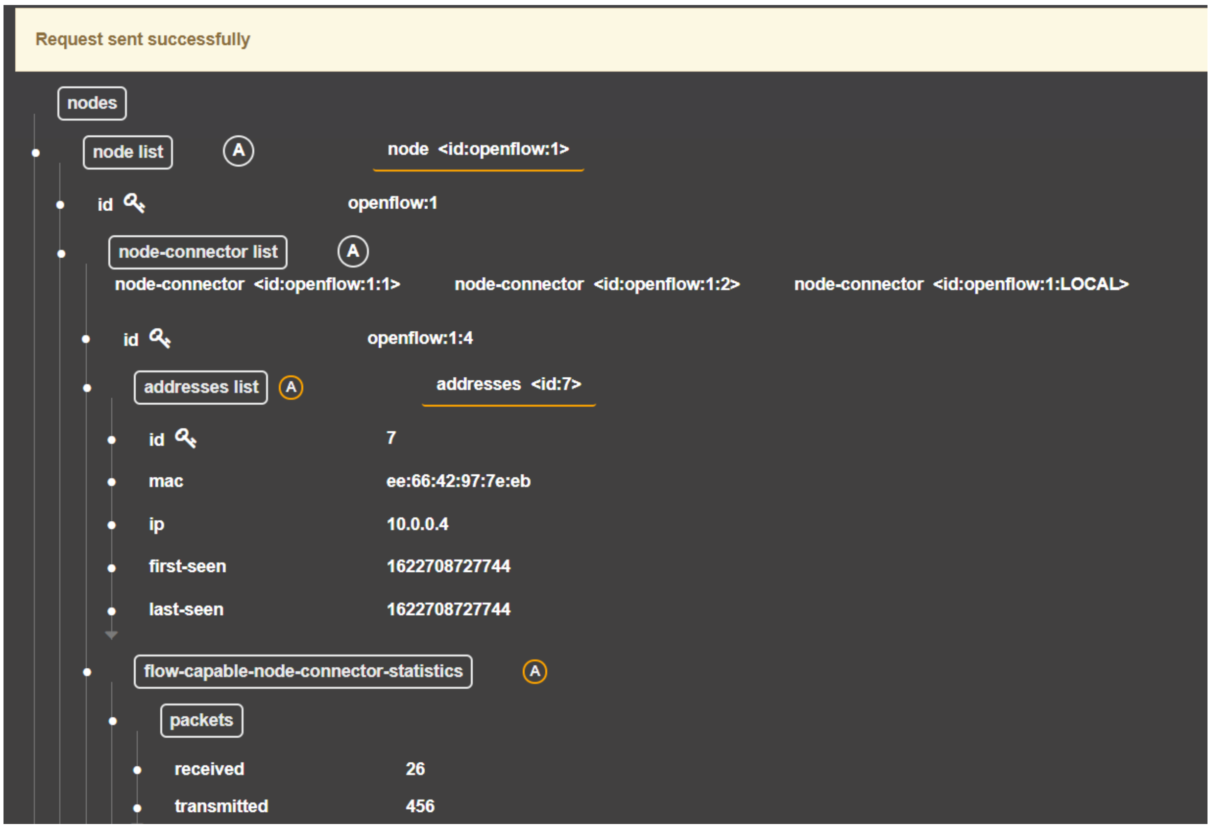

```
Request sent successfully

nodes
  node list   (A)        node <id:openflow:1>
    id                   openflow:1
    node-connector list  (A)
      node-connector <id:openflow:1:1>   node-connector <id:openflow:1:2>   node-connector <id:openflow:1:LOCAL>
      id                 openflow:1:4
      addresses list (A)   addresses <id:7>
        id               7
        mac              ee:66:42:97:7e:eb
        ip               10.0.0.4
        first-seen       1622708727744
        last-seen        1622708727744
      flow-capable-node-connector-statistics   (A)
        packets
          received       26
          transmitted    456
```

Figure 7 - Data retrieved from API in ODL GUI

CONCLUSIONS

Overall, this investigation went well, it was thorough in ensuring that everything was as fair and reliable as possible so that the results I got out of the testbed could be trusted. The large amount of testing for the experiments although overengineered provided a reliable database of information for analysis and helped explaining the correlations between how different network condition impairments impact end-to-end video streaming quality. It made life a lot easier when it came to comprehensibly writing out and understanding the analysis of the results.

The investigation was enjoyable but difficult due to lack of experience in networking. Given all that was learned in the process, going forward Docker would be used instead of VM's. This is simply to avoid a lot of the compatibility issues that come with using VM's as setting up the testbed took the most time.

REFERENCES

1. Barakabitze, A., Barman, N., Ahmad, A., Zadtootaghaj, S., Sun, L., Martini, M. and Atzori, L., 2020. QoE Management of Multimedia Streaming Services in Future Networks: A Tutorial and Survey. *IEEE COMMUNICATIONS SURVEYS & TUTORIALS*, 22(1).
2. Bonfim, M., Dias, K. and Fernandes, S., 2019. Integrated NFV/SDN Architectures. *ACM Computing Surveys*, 51(6), pp.1-39.
3. Kreutz, D., M. V Ramos, F., Verı´ssimo, P., Rothenberg, C., Azodolmolky, S. and Uhlig, S., 2015. https://ieeexplore.ieee.org/abstract/document/6994333. *Proceedings of the IEEE*, 103(1).
4. Mok, R., Chan, E. and Chang, R., 2011. Measuring the Quality of Experience of HTTP Video Streaming. In: *12th IFIP/IEEE International Symposium on Integrated Network Management*. [online] Dublin, Ireland: IEEE. Available at: <https://www.researchgate.net/publication/221293512_Measuring_the_Quality_of_Experience_of_HTTP_Video_Streaming> [Accessed 18 May 2021].
5. Gohar, A. and Lee, S., 2020. Multipath Dynamic Adaptive Streaming over HTTP Using Scalable Video Coding in Software Defined Networking. *Applied Sciences*, 10(21), p.7691.
6. Chan, A., Zeng, K., Mohapatra, P., Lee, S. and Banerjee, S., 2010. Metrics for Evaluating Video Streaming Quality in Lossy IEEE 802.11 Wireless Networks. In: *2010 Proceedings IEEE INFOCOM*. [online] San Diego, CA, USA: IEEE. Available at: <https://ieeexplore.ieee.org/abstract/document/5461979> [Accessed 20 May 2021].

APPENDIX

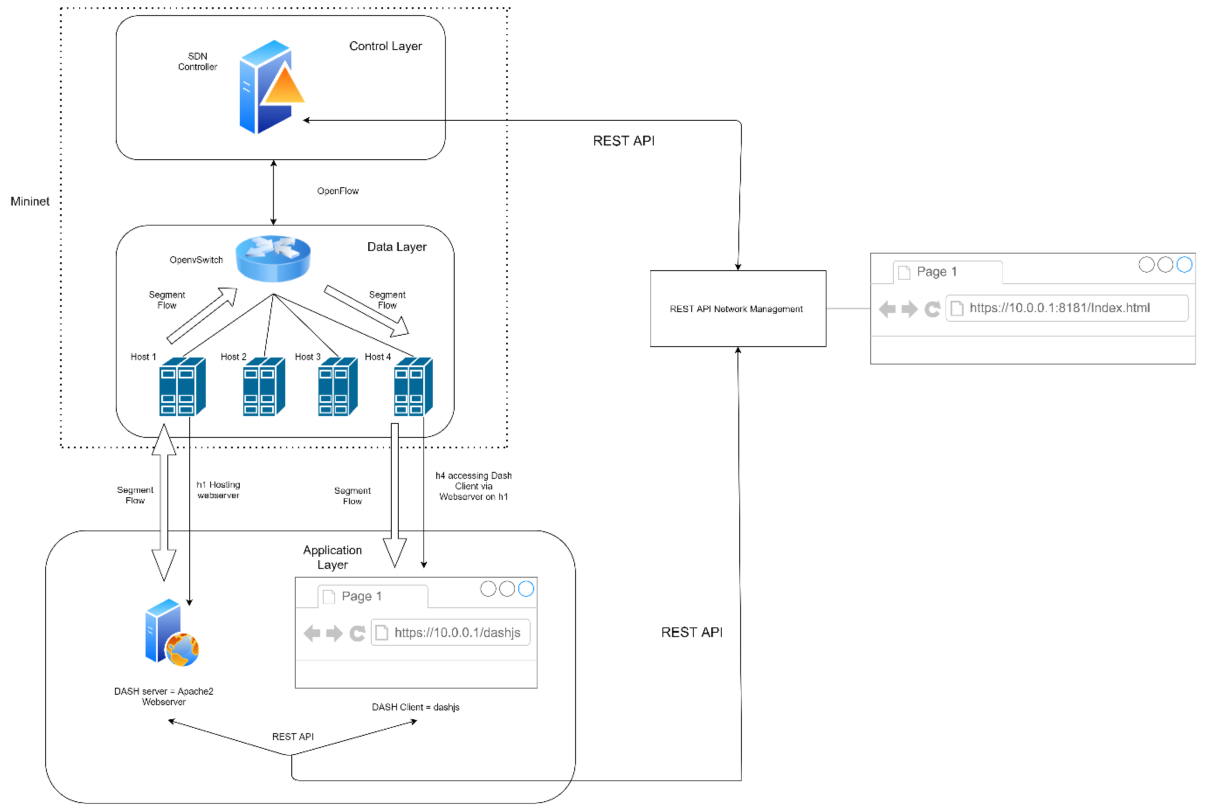
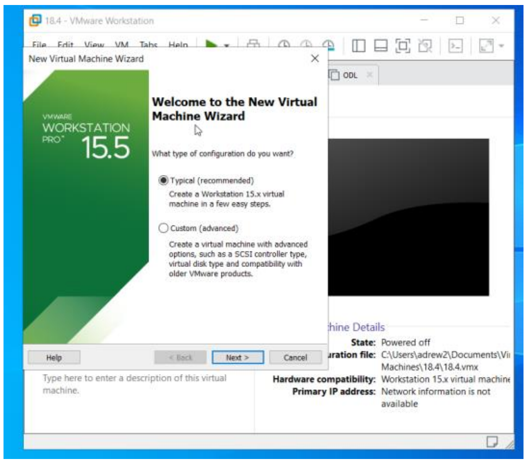


Figure 8 - Overall Testbed Architecture


18.4 - VMware Workstation
File Edit View VM Tabs Help
ODL
VMWARE WORKSTATION PRO™ 15.5
Help
< Back
Next >
Cancel

New Virtual Machine Wizard

Welcome to the New Virtual Machine Wizard

What type of configuration do you want?

[✓ Typical (recommended)]
Create a Workstation 15.x virtual machine in a few easy steps.

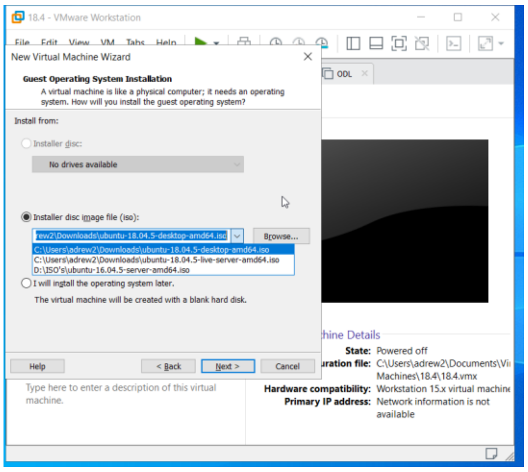
Custom (advanced)
Create a virtual machine with advanced options, such as a SCSI controller type, virtual disk type and compatibility with older VMware products.

Type here to enter a description of this virtual machine.

hine Details

State:	Powered off
uration file:	C:\Users\adrew2\Documents\Vi Machines\18.4\18.4.vmx
Hardware compatibility:	Workstation 15.x virtual machine
Primary IP address:	Network information is not available

Figure 9 - Starting up new VM.



18.4 - VMware Workstation

New Virtual Machine Wizard

Guest Operating System Installation
A virtual machine is like a physical computer; it needs an operating system. How will you install the guest operating system?

Install from:

○ Installer disc:
No drives available

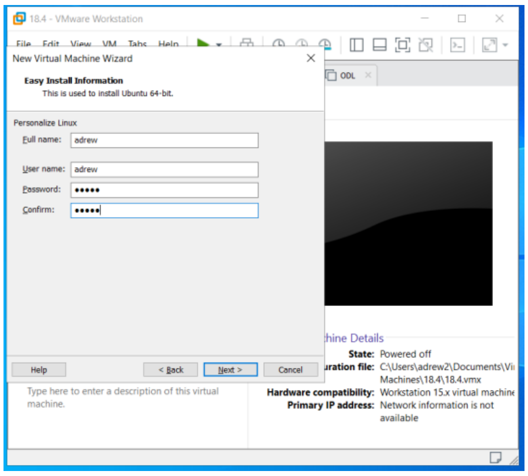
● Installer disc image file (iso):
rew2\Downloads\ubuntu-18.04.5-desktop-amd64.iso
C:\Users\adrew2\Downloads\ubuntu-18.04.5-desktop-amd64.iso
C:\Users\adrew2\Downloads\ubuntu-18.04.5-live-server-amd64.iso
D:\ISO's\ubuntu-16.04.5-server-amd64.iso

○ I will install the operating system later.
The virtual machine will be created with a blank hard disk.

State: Powered off
Hardware compatibility: Workstation 15.x virtual machine
Primary IP address: Network information is not available

File Edit View VM Tabs Help
ODL
Browse...
Help
< Back
Next >
Cancel
Type here to enter a description of this virtual machine.

Figure 10 - Setting the ISO image for new VM



18.4 - VMware Workstation

New Virtual Machine Wizard

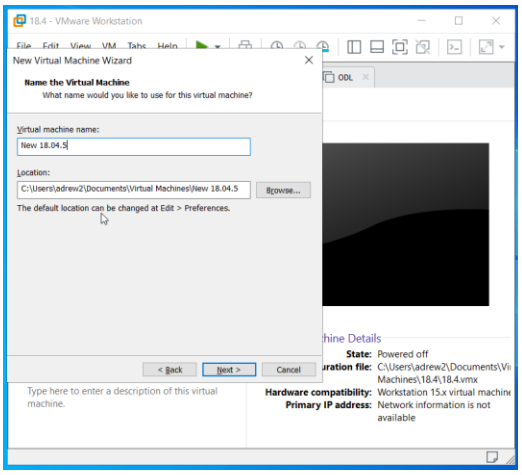
Easy Install Information
This is used to install Ubuntu 64-bit.

Personalize Linux
Full name: adrew
User name: adrew
Password: •••••
Confirm: •••••

State: Powered off
Hardware compatibility: Workstation 15.x virtual machine
Primary IP address: Network information is not available

File Edit View VM Tabs Help
ODL
Help
< Back
Next >
Cancel
Type here to enter a description of this virtual machine.

Figure 11 - Giving VM login details



18.4 - VMware Workstation

New Virtual Machine Wizard

Name the Virtual Machine
What name would you like to use for this virtual machine?

Virtual machine name:
New 18.04.5

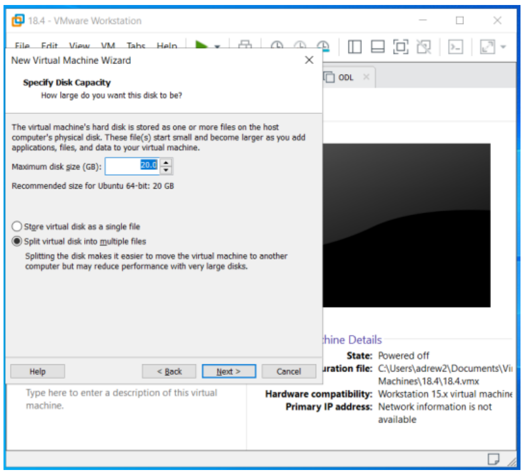
Location:
C:\Users\adrew2\Documents\Virtual Machines\New 18.04.5

The default location can be changed at Edit > Preferences.

State: Powered off
Hardware compatibility: Workstation 15.x virtual machine
Primary IP address: Network information is not available

File Edit View VM Tabs Help
ODL
Browse...
< Back
Next >
Cancel
Type here to enter a description of this virtual machine.

Figure 12 - Setting name of the new VM



New Virtual Machine Wizard

Specify Disk Capacity

How large do you want this disk to be?

The virtual machine's hard disk is stored as one or more files on the host computer's physical disk. These file(s) start small and become larger as you add applications, files, and data to your virtual machine.

Maximum disk size (GB): 20.0

Recommended size for Ubuntu 64-bit: 20 GB

○ Store virtual disk as a single file
[✓ ● Split virtual disk into multiple files]

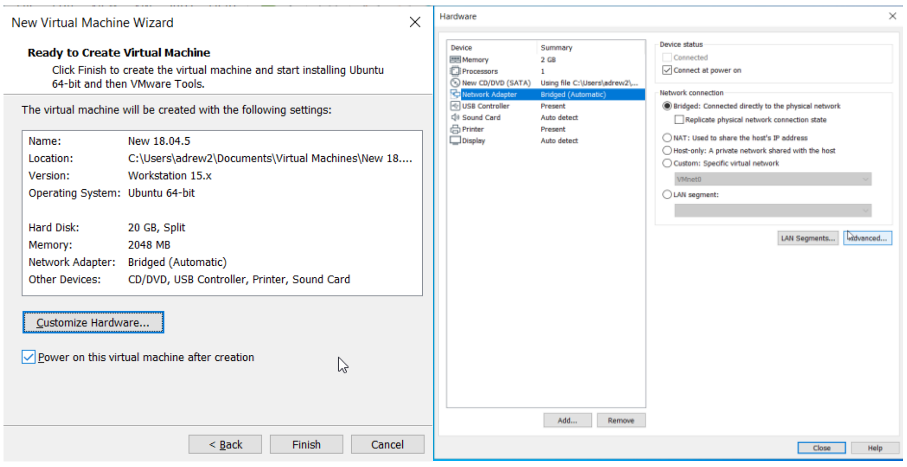
Splitting the disk makes it easier to move the virtual machine to another computer but may reduce performance with very large disks.

Type here to enter a description of this virtual machine.

hine Details

State:	Powered off
uration file:	C:\Users\adrew2\Documents\Vi Machines\18.4\18.4.vmx
Hardware compatibility:	Workstation 15.x virtual machine
Primary IP address:	Network information is not available

Figure 13 - Setting the storage capacity of new VM



New Virtual Machine Wizard

Ready to Create Virtual Machine

Click Finish to create the virtual machine and start installing Ubuntu 64-bit and then VMware Tools.

The virtual machine will be created with the following settings:

Name:	New 18.04.5
Location:	C:\Users\adrew2\Documents\Virtual Machines\New 18....
Version:	Workstation 15.x
Operating System:	Ubuntu 64-bit
Hard Disk:	20 GB, Split
Memory:	2048 MB
Network Adapter:	Bridged (Automatic)
Other Devices:	CD/DVD, USB Controller, Printer, Sound Card

☑ Power on this virtual machine after creation

Hardware

Device	Summary
Memory	2 GB
Processors	1
New CD/DVD (SATA)	Using file C:\Users\adrew2\...
Network Adapter	Bridged (Automatic)
USB Controller	Present
Sound Card	Auto detect
Printer	Present
Display	Auto detect

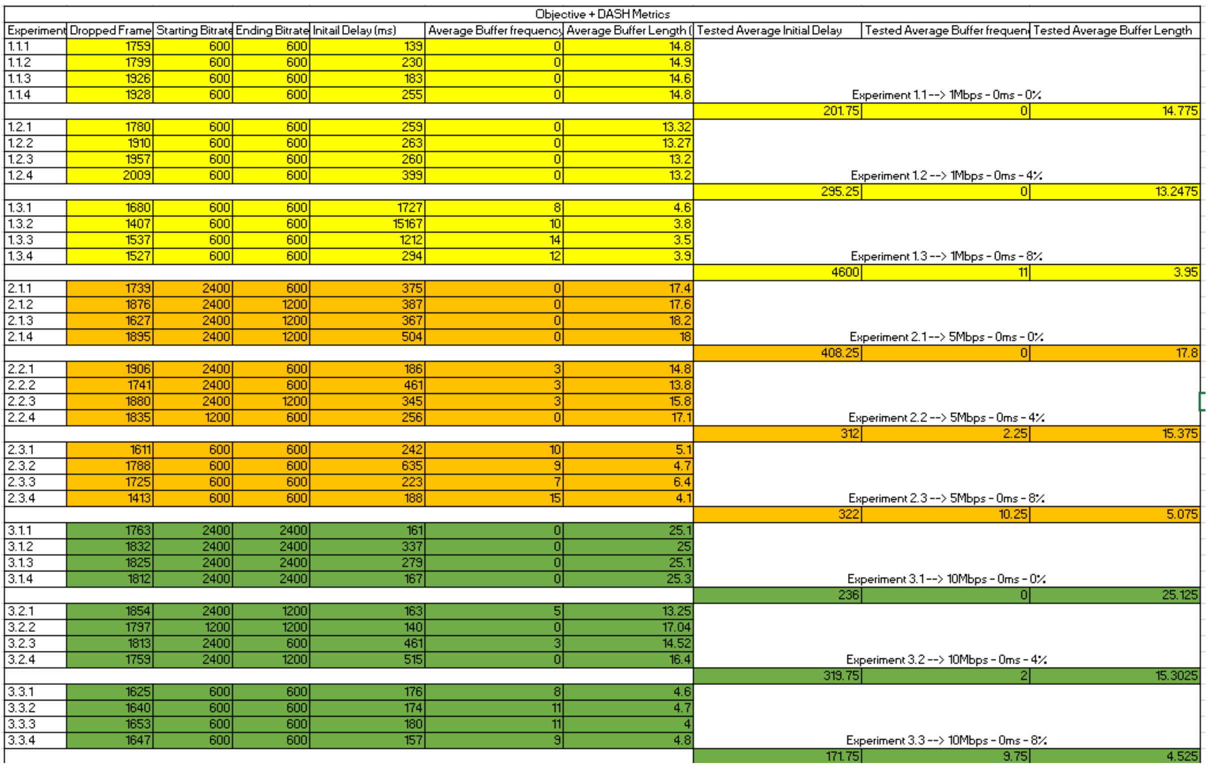
Device status

☐ Connected
☑ Connect at power on

Network connection

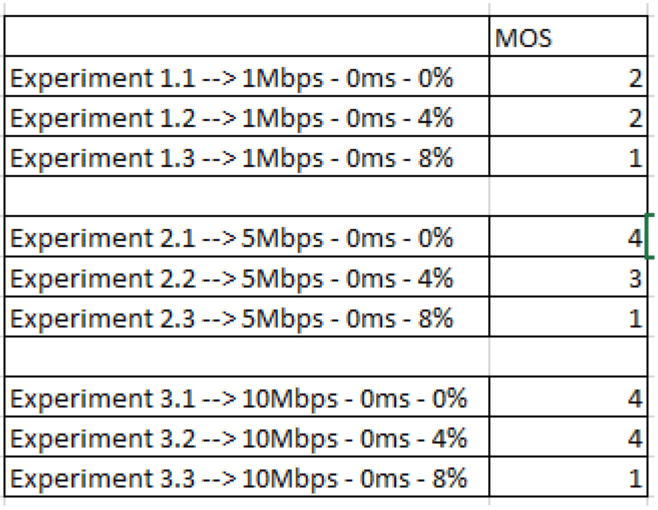
[✓ ● Bridged: Connected directly to the physical network]
☐ Replicate physical network connection state
○ NAT: Used to share the host's IP address
○ Host-only: A private network shared with the host
○ Custom: Specific virtual network
VMnet0
○ LAN segment:

Figure 14 - Setting Up Bridged connection and finishing



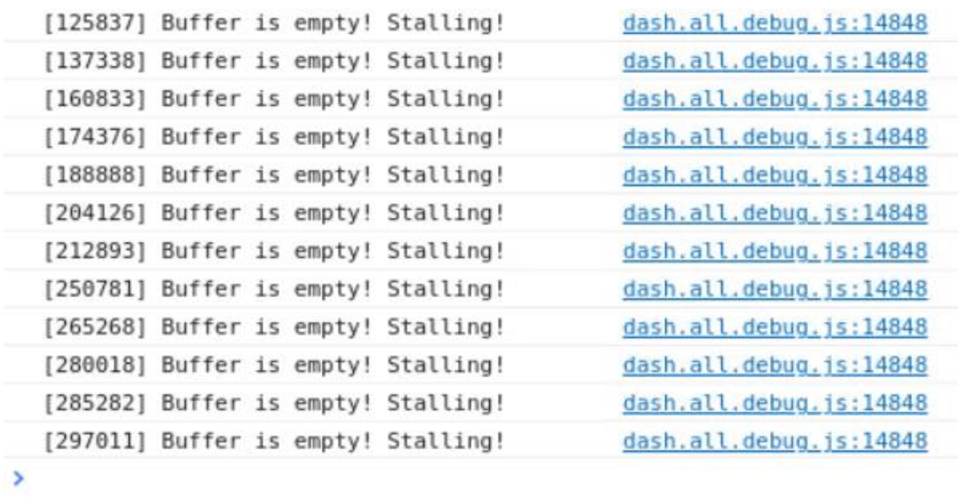
Objective + DASH Metrics									
Experiment	Dropped Frame	Starting Bitrate	Ending Bitrate	Initail Delay (ms)	Average Buffer frequency	Average Buffer Length (	Tested Average Initial Delay	Tested Average Buffer frequen	Tested Average Buffer Length
1.1.1	1759	600	600	139	0	14.8			
1.1.2	1799	600	600	230	0	14.9			
1.1.3	1926	600	600	183	0	14.6			
1.1.4	1928	600	600	255	0	14.8		Experiment 1.1 --> 1Mbps - 0ms - 0%	
							201.75	0	14.775
1.2.1	1780	600	600	259	0	13.32			
1.2.2	1910	600	600	263	0	13.27			
1.2.3	1957	600	600	260	0	13.2			
1.2.4	2009	600	600	399	0	13.2		Experiment 1.2 --> 1Mbps - 0ms - 4%	
							295.25	0	13.2475
1.3.1	1680	600	600	1727	8	4.6			
1.3.2	1407	600	600	15167	10	3.8			
1.3.3	1537	600	600	1212	14	3.5			
1.3.4	1527	600	600	294	12	3.9		Experiment 1.3 --> 1Mbps - 0ms - 8%	
							4600	11	3.95
2.1.1	1739	2400	600	375	0	17.4			
2.1.2	1876	2400	1200	387	0	17.6			
2.1.3	1627	2400	1200	367	0	18.2			
2.1.4	1895	2400	1200	504	0	18		Experiment 2.1 --> 5Mbps - 0ms - 0%	
							408.25	0	17.8
2.2.1	1906	2400	600	186	3	14.8			
2.2.2	1741	2400	600	461	3	13.8			
2.2.3	1880	2400	1200	345	3	15.8			
2.2.4	1835	1200	600	256	0	17.1		Experiment 2.2 --> 5Mbps - 0ms - 4%	
							312	2.25	15.375
2.3.1	1611	600	600	242	10	5.1			
2.3.2	1788	600	600	635	9	4.7			
2.3.3	1725	600	600	223	7	6.4			
2.3.4	1413	600	600	188	15	4.1		Experiment 2.3 --> 5Mbps - 0ms - 8%	
							322	10.25	5.075
3.1.1	1763	2400	2400	161	0	25.1			
3.1.2	1832	2400	2400	337	0	25			
3.1.3	1825	2400	2400	279	0	25.1			
3.1.4	1812	2400	2400	167	0	25.3		Experiment 3.1 --> 10Mbps - 0ms - 0%	
							236	0	25.125
3.2.1	1854	2400	1200	163	5	13.25			
3.2.2	1797	1200	1200	140	0	17.04			
3.2.3	1813	2400	600	461	3	14.52			
3.2.4	1759	2400	1200	515	0	16.4		Experiment 3.2 --> 10Mbps - 0ms - 4%	
							319.75	2	15.3025
3.3.1	1625	600	600	176	8	4.6			
3.3.2	1640	600	600	174	11	4.7			
3.3.3	1653	600	600	180	11	4			
3.3.4	1647	600	600	157	9	4.8		Experiment 3.3 --> 10Mbps - 0ms - 8%	
							171.75	9.75	4.525

Figure 15 - Full table of objective tests



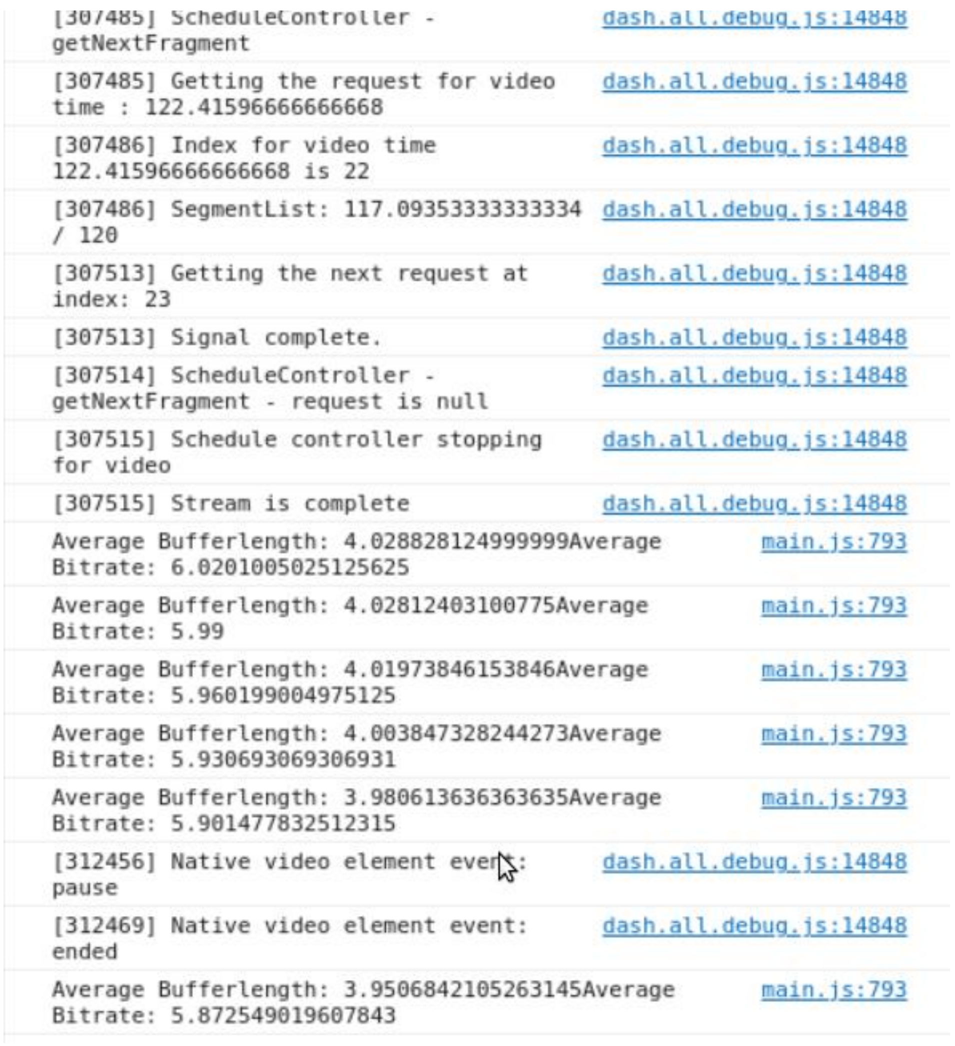
	MOS
Experiment 1.1 --> 1Mbps - 0ms - 0%	2
Experiment 1.2 --> 1Mbps - 0ms - 4%	2
Experiment 1.3 --> 1Mbps - 0ms - 8%	1
Experiment 2.1 --> 5Mbps - 0ms - 0%	4
Experiment 2.2 --> 5Mbps - 0ms - 4%	3
Experiment 2.3 --> 5Mbps - 0ms - 8%	1
Experiment 3.1 --> 10Mbps - 0ms - 0%	4
Experiment 3.2 --> 10Mbps - 0ms - 4%	4
Experiment 3.3 --> 10Mbps - 0ms - 8%	1

Figure 16 - Full table of subjective tests



```
[125837] Buffer is empty! Stalling!          dash.all.debug.js:14848
[137338] Buffer is empty! Stalling!          dash.all.debug.js:14848
[160833] Buffer is empty! Stalling!          dash.all.debug.js:14848
[174376] Buffer is empty! Stalling!          dash.all.debug.js:14848
[188888] Buffer is empty! Stalling!          dash.all.debug.js:14848
[204126] Buffer is empty! Stalling!          dash.all.debug.js:14848
[212893] Buffer is empty! Stalling!          dash.all.debug.js:14848
[250781] Buffer is empty! Stalling!          dash.all.debug.js:14848
[265268] Buffer is empty! Stalling!          dash.all.debug.js:14848
[280018] Buffer is empty! Stalling!          dash.all.debug.js:14848
[285282] Buffer is empty! Stalling!          dash.all.debug.js:14848
[297011] Buffer is empty! Stalling!          dash.all.debug.js:14848
>
```

Figure 17 - How Buffering events are shown in console.



```
[307485] ScheduleController -              dash.all.debug.js:14848
getNextFragment
[307485] Getting the request for video     dash.all.debug.js:14848
time : 122.41596666666668
[307486] Index for video time              dash.all.debug.js:14848
122.41596666666668 is 22
[307486] SegmentList: 117.09353333333334   dash.all.debug.js:14848
/ 120
[307513] Getting the next request at       dash.all.debug.js:14848
index: 23
[307513] Signal complete.                  dash.all.debug.js:14848
[307514] ScheduleController -              dash.all.debug.js:14848
getNextFragment - request is null
[307515] Schedule controller stopping      dash.all.debug.js:14848
for video
[307515] Stream is complete                dash.all.debug.js:14848
Average Bufferlength: 4.028828124999999Average          main.js:793
Bitrate: 6.0201005025125625
Average Bufferlength: 4.02812403100775Average           main.js:793
Bitrate: 5.99
Average Bufferlength: 4.01973846153846Average           main.js:793
Bitrate: 5.960199004975125
Average Bufferlength: 4.003847328244273Average          main.js:793
Bitrate: 5.930693069306931
Average Bufferlength: 3.980613636363635Average          main.js:793
Bitrate: 5.901477832512315
[312456] Native video element event:       dash.all.debug.js:14848
pause
[312469] Native video element event:       dash.all.debug.js:14848
ended
Average Bufferlength: 3.9506842105263145Average         main.js:793
Bitrate: 5.872549019607843
```

Figure 18 - How Average buffer length and initial delay are shown in console.

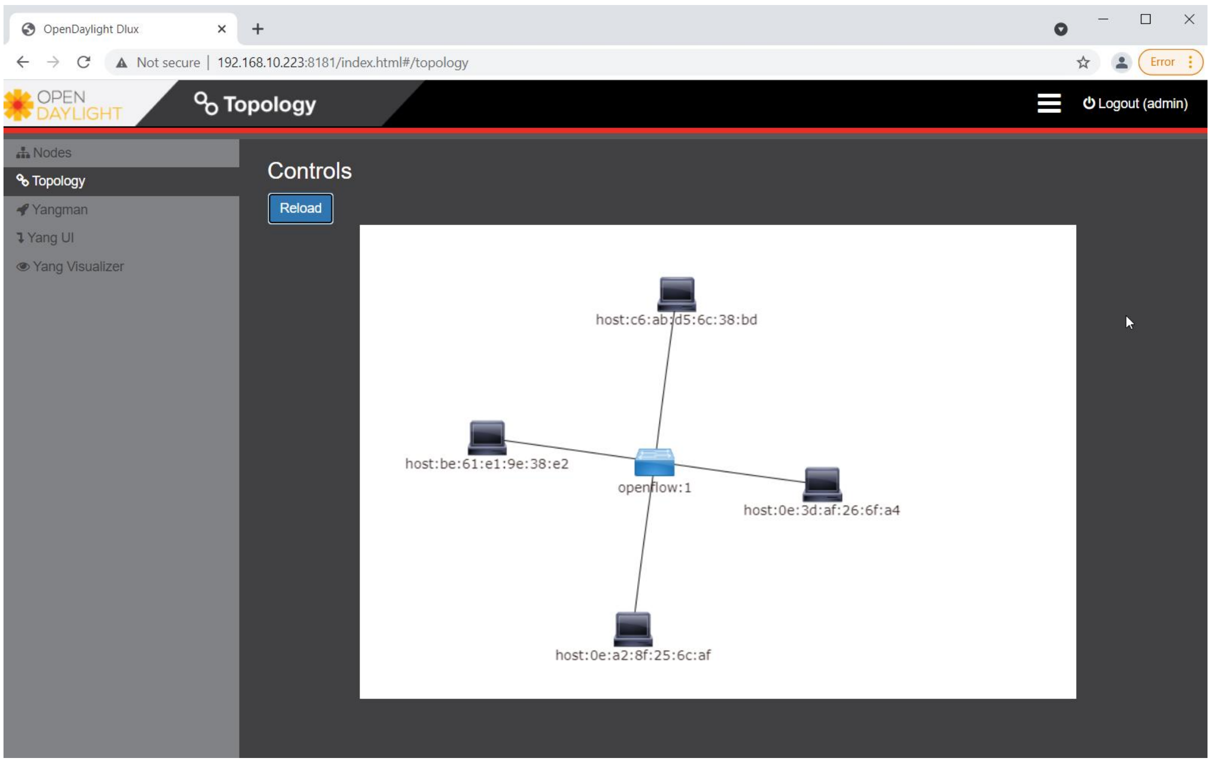
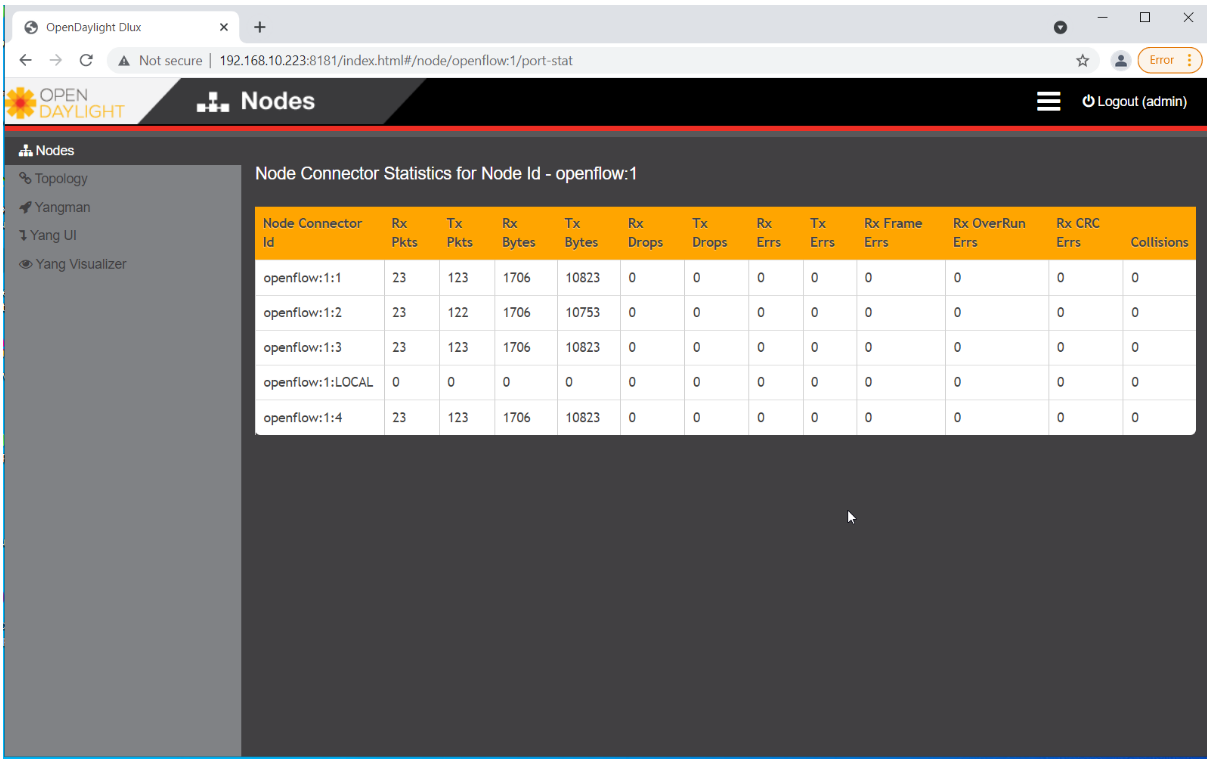


Figure 19 - Network Topology ODL GUI



Node Connector Statistics for Node Id - openflow:1

Node Connector Id	Rx Pkts	Tx Pkts	Rx Bytes	Tx Bytes	Rx Drops	Tx Drops	Rx Errs	Tx Errs	Rx Frame Errs	Rx OverRun Errs	Rx CRC Errs	Collisions
openflow:1:1	23	123	1706	10823	0	0	0	0	0	0	0	0
openflow:1:2	23	122	1706	10753	0	0	0	0	0	0	0	0
openflow:1:3	23	123	1706	10823	0	0	0	0	0	0	0	0
openflow:1:LOCAL	0	0	0	0	0	0	0	0	0	0	0	0
openflow:1:4	23	123	1706	10823	0	0	0	0	0	0	0	0

Figure 20 - Nodes ODL GUI

	18	69	88

Figure 21 - Data from the flow table in ODL GUI