Alistair Drew

10567887

Module Leader: Lingfen Sun

Comp3004 REPORT

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

Contents

[Abstract 2](#_Toc73620899)

[Introduction and background 2](#_Toc73620900)

[SDN 2](#_Toc73620901)

[Dash 2](#_Toc73620902)

[Video Streaming Quality 3](#_Toc73620903)

[Steps Taken 3](#_Toc73620904)

[Report Structure 3](#_Toc73620905)

[Testbed and development 4](#_Toc73620906)

[Setup – general structure of the testbed and it’s configuration 4](#_Toc73620907)

[Ubuntu Desktop VM (new 18.4) setup 4](#_Toc73620908)

[Ubuntu Live server VM (ODL) setup 6](#_Toc73620909)

[Code developed. 7](#_Toc73620910)

[Experiments, Results and Analysis 8](#_Toc73620911)

[Discussions, network management via REST API 10](#_Toc73620912)

[Problems encountered and going forward 10](#_Toc73620913)

[Network Management via REST API 11](#_Toc73620914)

[Conclusions 13](#_Toc73620915)

[References 14](#_Toc73620916)

[Appendix 15](#_Toc73620917)

# 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

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

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.

## 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:

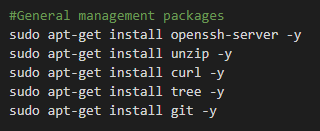
1. Start up the new VM in VMware Workstation.
2. Select the needed ISO image for the operating system of the new VM. In this case either the desktop or live server image.
3. Create the login details for the new VM.
4. Set the name of the new VM to either ODL or new 18.4.
5. Setting the disk capacity 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.

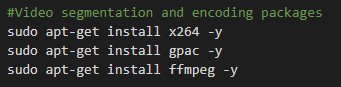
## Ubuntu Desktop VM (new 18.4) setup

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.

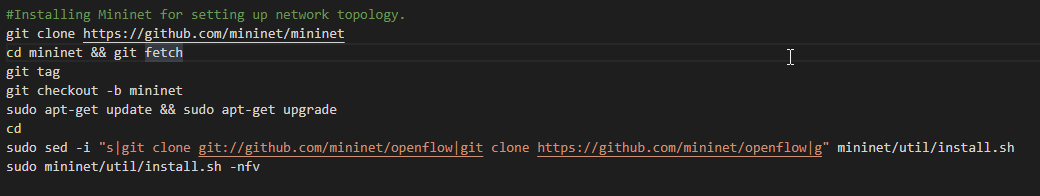
Packages for testbed setup are split into three sections. The first include general management packages, which allow the manipulation of files, either via downloading or for general management of the VM. The second is the Apache2 package that is going to act as the DASH client. Finally, the packages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



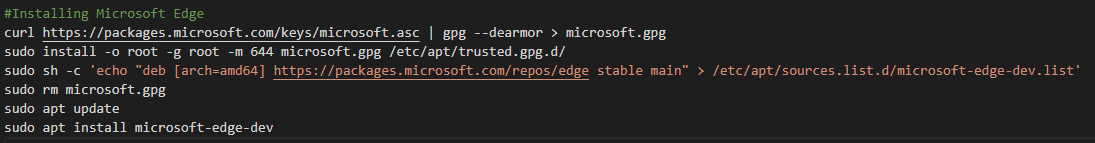




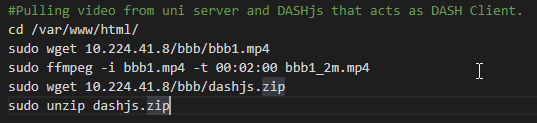
Mininet is installed for adaptively creating the network, it creates the data layer (controller, switch, and hosts) based on a python file.



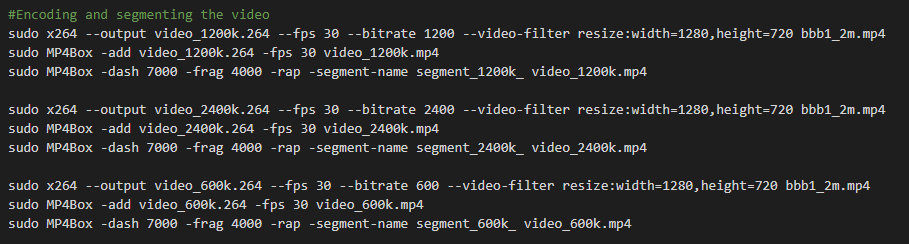
Microsoft Edge is installed as it is a browser that runs on less CPU resources. This is important later in the discussion section.



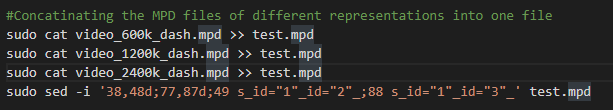
This section of the script pulls DASHjs and the video from university server to act as the DASH Client. It clips the video to 2 minuets for testing purposes.



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



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



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.

Text

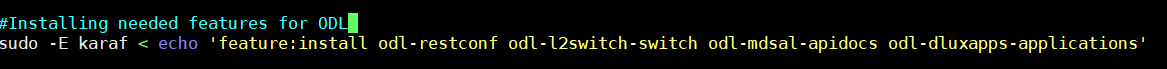
Description automatically generated

Then ODL is Installed from the packages.

Text

Description automatically generated

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



## 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.

Text

Description automatically generated

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.

Text

Description automatically generated

Text

Description automatically generated

# 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.

Graphical user interface, text, application, chat or text message

Description automatically generated

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.

Diagram

Description automatically generated

Figure 3 - Network Topology show by ODL GUI

Table

Description automatically generated

Figure 4 - Nodes shown by ODL GUI

Table

Description automatically generated

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

Text

Description automatically generated with medium confidence

Figure 6 - API's in ODL GUI

Text

Description automatically generated

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

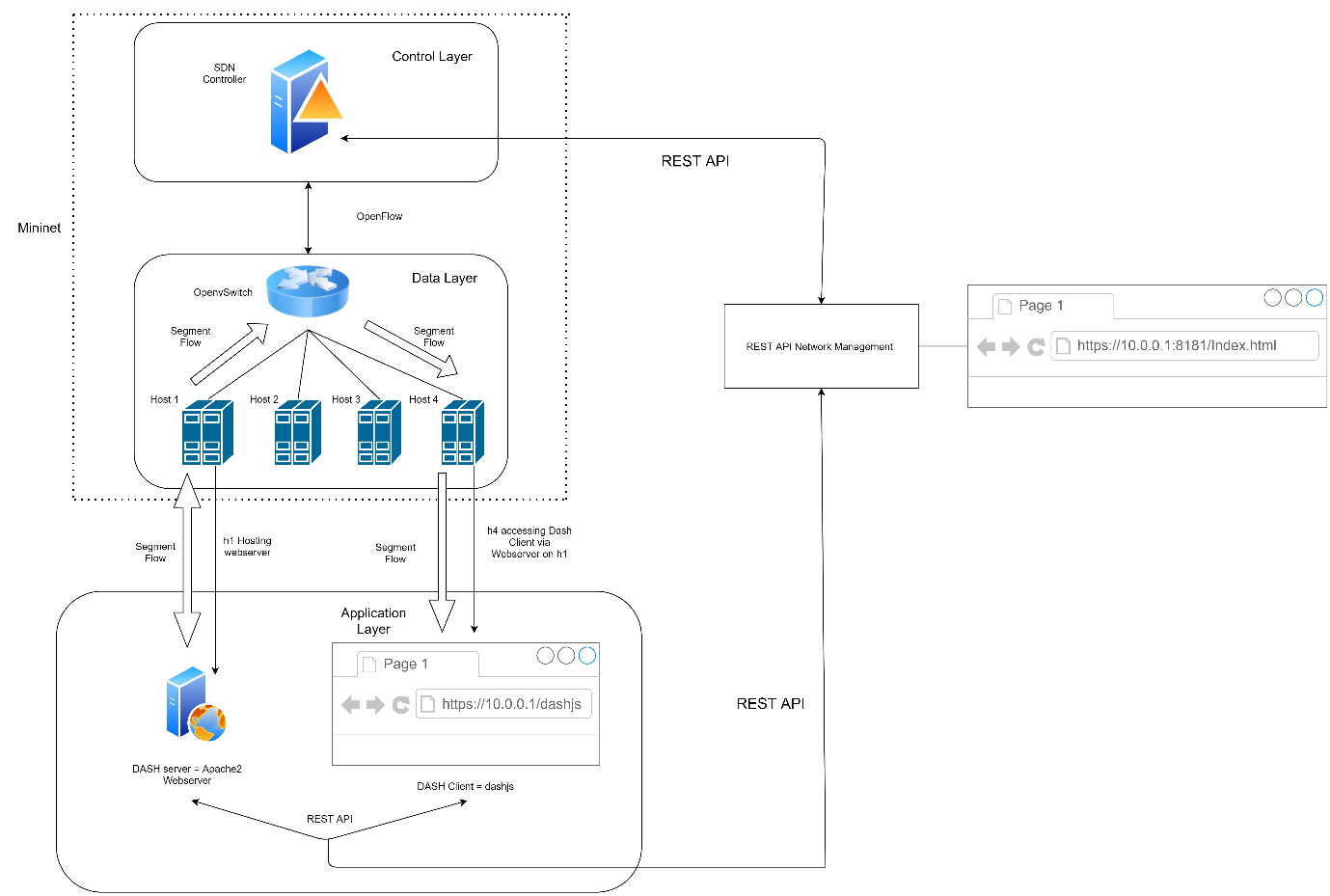


Figure - Overall Testbed Architecture

Graphical user interface, application

Description automatically generated

Figure 9 - Starting up new VM.

Graphical user interface, text, application

Description automatically generated

Figure 10 - Setting the ISO image for new VM

Graphical user interface, application

Description automatically generated

Figure 11 - Giving VM login details

Graphical user interface, application

Description automatically generated

Figure 12 - Setting name of the new VM

Graphical user interface, text, application

Description automatically generated

Figure 13 - Setting the storage capacity of new VM

Graphical user interface, text, application, email

Description automatically generatedGraphical user interface, application

Description automatically generated

Figure 14 - Setting Up Bridged connection and finishing

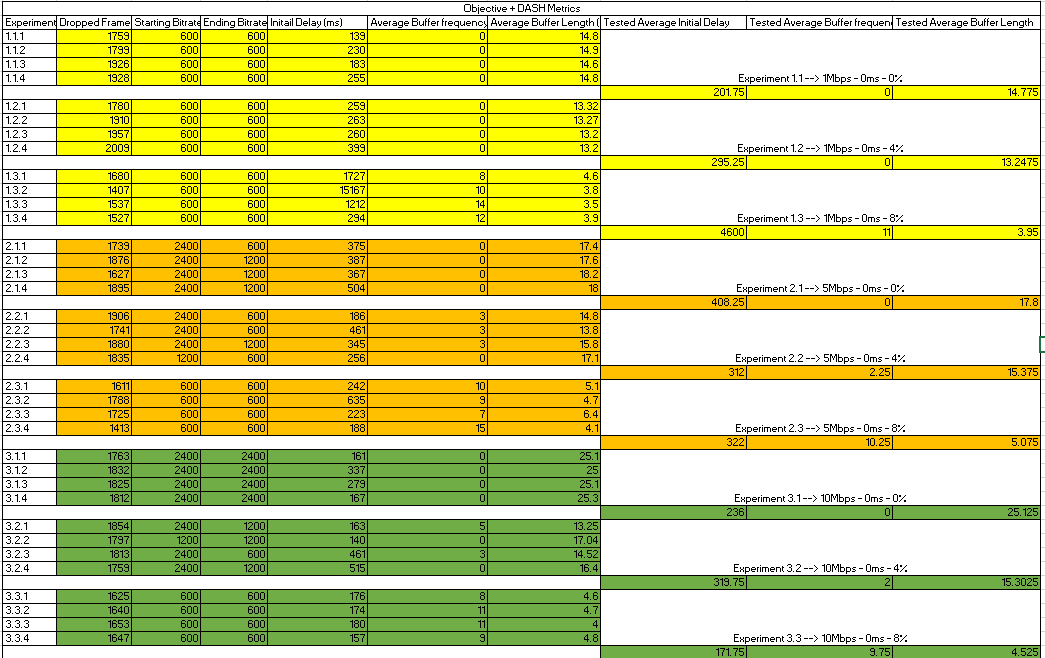


Figure - Full table of objective tests

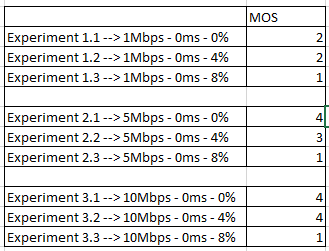


Figure - Full table of subjective tests

Table

Description automatically generated

Figure 17 - How Buffering events are shown in console.

Table

Description automatically generated

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

Graphical user interface

Description automatically generated

Figure 19 - Network Topology ODL GUI

Calendar

Description automatically generated

Figure 20 - Nodes ODL GUI



Figure 21 - Data from the flow table in ODL GUI