ISCTE IUL
Instituto Universitário de Lisboa

IUL School of Technology and Architecture
Department of Information Science and Technology

A Policy-Based Framework Towards Smooth Adaptive Playback for Dynamic Video Streaming over HTTP

Mickaël Rocha da Cunha

Dissertation submitted as partial fulfilment of the requirements for the degree of
Master in Telecommunications and Computer Engineering

Supervisor:
Prof. José André Rocha Sá Moura, Assistant Professor,
ISCTE – IUL

Co-Supervisor:
Prof. Paulo Jorge Lourenço Nunes, Assistant Professor
ISCTE – IUL

October, 2018

Acknowledgements

Success can never have the same taste without suffering. Growth cannot happen without facing difficulties, without doubts and without perseverance. This is perhaps the biggest lesson that this dissertation taught me. The lesson that there are no “too big” challenges, and that everything is possible with hard work, dedication and belief.

Besides learning to believe in yourself, this dissertation and, more largely, this degree showed me once again that someone can become much better if you believe in yourself but more than that if you believe in others. This journey could not have succeeded without the help of many people that I would like to thank here.

First, I would like to thank my colleagues that helped me throughout this journey and this new adventure with a special mention to the ones that stayed closer to me in this path, Leonel Piscalho and Pedro Manso.

Of course, I also would like to thank all my teachers for all the knowledge they passed on to me with an obvious special thanks to my supervisors in this dissertation Professor José Moura and Professor Paulo Nunes for all their support, patience and for all the time they conceded me.

Thanking all my family, especially my cousins that helped me when I got to Lisbon and my cousin Lourenço that helped me in a time of great need without expecting nothing in return.

Finally, I would like to thank from the bottom of my heart my friends, my sister and my parents for never stopped believing in me.

Resumo

O *streaming* de vídeo na Internet é um fenómeno que tem vindo a crescer de forma significativa nos últimos anos e que promete continuar a crescer no futuro. Este facto está associado ao aumento do número de utilizadores na Internet e, sobretudo, à crescente diversificação de dispositivos que se verifica atualmente.

As primeiras soluções utilizadas no *streaming* de vídeo não acomodavam adequadamente o ponto de vista do utilizador na avaliação da qualidade do vídeo, i.e., a Qualidade de Experiência (QoE) do utilizador. Esta debilidade foi suplantada com o protocolo de *streaming* de vídeo adaptativo DASH. A principal funcionalidade deste protocolo é fornecer diferente versões, em termos de qualidade, para o mesmo conteúdo. Desta forma, dependendo do estado da infraestrutura de rede entre o servidor de vídeo e o dispositivo do utilizador, o protocolo DASH seleciona automaticamente a versão do conteúdo mais adequada a essas condições. Tal permite fornecer ao utilizador a melhor qualidade possível para o consumo deste conteúdo.

O principal problema com o protocolo DASH está associado com o ciclo, entre cada cliente e o servidor de vídeo, que controla o débito de cada fluxo de vídeo. De facto, à medida que a rede fica congestionada, o cliente irá começar a requerer ao servidor um fluxo de vídeo com um débito menor. Ainda assim, devido à latência da rede, o protocolo DASH pode não ser capaz por si só de estabilizar o débito do fluxo de vídeo num nível que consiga garantir uma QoE satisfatória para os utilizadores.

A programação de redes é uma área muito popular e ativa na gestão de infraestruturas de redes. Nesta área, o paradigma de *Software Defined Networking* é uma abordagem onde um controlador da rede, com um ponto de vista relativamente abstrato da infraestrutura física da rede, tenta desempenhar uma gestão mais eficiente do encaminhamento de rede.

Neste trabalho estuda-se a junção do protocolo DASH e do paradigma de *Software Defined Networking*, de forma a atingir uma partilha mais adequada dos recursos da rede. O objetivo é implementar uma solução que seja benéfica tanto para a qualidade de experiência dos utilizadores como para a gestão da rede.

Palavras-chave: *Streaming* de vídeo, SDN, DASH, QoE

Abstract

The growth of video streaming in the Internet in the last few years has been highly significant and promises to continue in the future. This fact is related to the growth of Internet users and especially with the diversification of the end-user devices that happens nowadays.

Earlier video streaming solutions didn´t consider adequately the Quality of Experience from the user's perspective. This weakness has been since overcame with the DASH video streaming. The main feature of this protocol is to provide different versions, in terms of quality, of the same content. This way, depending on the status of the network infrastructure between the video server and the user device, the DASH protocol automatically selects the more adequate content version. Thus, it provides to the user the best possible quality for the consumption of that content.

The main issue with the DASH protocol is associated to the loop, between each client and video server, which controls the rate of the video stream. In fact, as the network congestion increases, the client requests to the server a video stream with a lower rate. Nevertheless, due to the network latency, the DASH protocol in a standalone way may not be able to stabilize the video stream rate at a level that can guarantee a satisfactory QoE to the end-users.

Network programming is a very active and popular topic in the field of network infrastructures management. In this area, the Software Defined Networking paradigm is an approach where a network controller, with a relatively abstracted view of the physical network infrastructure, tries to perform a more efficient management of the data path.

The current work studies the combination of the DASH protocol and the Software Defined Networking paradigm in order to achieve a more adequate sharing of the network resources that could benefit both the users' QoE and network management.

Keywords: Video streaming, SDN, DASH, QoE.Table of Contents

Table of Contents

List of Tables

List of Figures

List of Abbreviations

3GPP – 3rd Generation Partnership Project

ABR – Adaptive BitRate

AC3 – Audio Codec 3

ALTO – Application-Layer Traffic Optimization

API – Application Programmable Interface

AVI – Audio Video Interleave

BG – Bitrate Guidance

BO – Buffer Occupancy

BOLA – Buffer Occupancy based Lyapunov Algorithm

BR – Bandwidth Reservation

CABA – Connection Aware Balancing Algorithm

CDP – Cisco Discovery Protocol

CLI – Command Line Interface

CPU – Central Processing Unit

DANE – DASH Assisting Network Element

DASH – Dynamic Adaptive Streaming over HTTP

FESTIVE – Fair, Efficient, Stable, adaptive

GPAC – Group Project on Advanced Content

GUI – Graphical User Interface

HAS – HTTP Adaptive Streaming

HD – High Definition

HDS – HTTP Dynamic Streaming

HLS – HTTP Live Streaming

HTTP – HyperText Transfer Protocol

I2RS – Interface to Routing System

IETF – Internet Engineering Task Force

IIS – Internet Information Services

IP – Internet Protocol

JSON – JavaScript Object Notation

LLDP – Link Layer Discovery Protocol

LTE – Long Term Evolution

MAC – Media Access Control

MD-SAL – Model-Driven Service Abstraction Layer

MKV – MatrosKa Video

ML – Machine Learning

MPD – Media Presentation Description

MPEG – Motion Picture Expert Group

MSS – Microsoft Smooth Streaming

NAT – Network Address Translation

ODL – OpenDayLight

OFM – OpenDayLight Flow Manager

OVS – OpenVSwitch

P2P – Peer-to-Peer

PCE – Path Computation Element

PCEP – Path Computation Element Protocol

PSNR – Peak Signal-to-Noise Ratio

QoE – Quality of Experience

QoS – Quality of Service

RPC – Remote Procedure Call

RTCP – RTP Control Protocol

RTP – Real-time Transport Protocol

RTSP – Real-Time Streaming Protocol

RTT – Round Trip Time

SAD – Sum of Absolute Differences

SDN – Software Defined Networking

SDP – Session Description Protocol

SMA – Simple Moving Average

SSMI – Structural SIMilarity index

TCP – Transport Control Protocol

TV – TeleVision

TVWS – TeleVision White Spaces

UDP – User Datagram Protocol

URI – Uniform Resource Identifier

URL – Uniform Resource Locator

VCP – Video Control Plane

VLAN – Virtual Local Area Network

VLC – VideoLan Client

VM – Virtual Machine

VQF – Video Quality Fairness

XML – eXtensible Markup Language

YANG – Yet Another Next Generation

"Impossible is just a big word thrown around by small men who find it easier to live in the world they've been given than to explore the power they have to change it. Impossible is not a fact. It's an opinion. Impossible is not a declaration. It's a dare. Impossible is potential. Impossible is temporary. Impossible is nothing"

"Do not quit.
Suffer now and live the rest of your life as a champion."

Muhammad Ali

1 – Introduction

1.1. Context and Motivation

Since Internet appeared, the number of users has never stopped growing and this will continue thanks to the development of Internet access particularly in some regions like Africa or the Middle East. This phenomena along with the multiplication of used devices per person will lead to an even higher growth of global traffic that is expected to reach 3.3 ZB per year by 2021 (Cisco, 2017). Among all traffic that exists on the Internet, there is one particular area that is consuming a significant part of world traffic: video streaming. The diversification of devices (especially the ones that we always carry around like smartphones) and the evolution of wireless networks were responsible for the popularity of video streaming and explained that in 2016 it already took 73 percent of all consumer Internet traffic, expecting to reach 82 percent by 2021 (Cisco, 2017).

Besides the fact that the number of users is growing exponentially, or the number of devices to be more accurate, there is another factor that has a tremendous importance when trying to understand today's global video traffic: Quality of Experience (QoE). QoE is a subjective concept that relies on the user's perception of the video quality he or she has access to. It can be defined as the overall performance of a system in the point of view of the user (Joskowicz & Ardao, 2012).

Therefore, QoE is a subjective metric and so can be hard to evaluate in a large scale. During many years, the evaluation of video quality metrics was performed using the Peak Signal-to-Noise Ratio (PSNR). It has since been accepted that this metric didn´t match the perceived quality by human viewers (Joskowicz & Ardao, 2012) which lead to the appearance of a new concept of video quality estimation.

Being a subjective metric, it is hard to evaluate it during the implementation or the testing phase. In the next chapter we will see that several attempts have been done to develop an objective model that could relate to the QoE. Some people may think that we can simply evaluate QoE by measuring bitrate, but it has been proved that the relationship between video bitrate and perceptual quality is not linear (Georgopoulos, Elkhatib, Broadbent, Mu, & Race, 2013). This is due to two main factors. First, the user environment must be considered, i.e., the same video bitrate will not result in the same perception for users with different devices capabilities (Cofano et al., 2017). If you use the same video bitrate for a user on a smart TeleVision (TV) or a user on a smartphone, it will obviously result in different QoE due to the screen size. The second factor is related with the fact that QoE is not only related to the video bitrate but also with other facts as well. There are some factors that simultaneously influence the QoE, as follows: i) video bitrate; ii) rebuffering; and, iii) the number of changes among video versions. The available bitrate for video affects the QoE because if influences the video version that will be required. Therefore, it normally affects the chosen version for the playout. For its turn, the rebuffering frequency affects the QoE because it is highly upsetting having the video stopped during its reproduction and having to wait for it to resume. Finally, too many changes among video versions will also affect the QoE because it creates instability in the reproduction which will be uncomfortable for the user. The rebuffering frequency has been identified as the main factor for the perceptual quality variance (Abuteir, Fladenmuller, Fourmaux, & Universites, 2017). Trying to tune and optimize, at the same time, all those factors, justify why it is so difficult to measure and achieve high QoE.

Besides the video bitrate, we have seen in the previous paragraph that rebuffering and changes among video versions also interfere with QoE. Though, there is a common issue with these three factors. They all are influenced by network conditions and available bandwidth. Users often struggle with incapacity to reach sufficient bandwidth especially when sharing a bottleneck with a high number of users. The bandwidth's division among all users of a given network may lead to an insufficient bandwidth on the user side that could cause rebuffering and low bitrate. Additionally, even if you divide the available bandwidth equally among all users, the differences of devices' resolution between them will result in unfair video quality levels as perceived by end-users (Georgopoulos et al., 2013).

To support a fair video distribution with adequate QoE, several technologies have been created to satisfy the users' expectations with the available networks' capabilities. In addition, although many of them are still in use, there is one that was accepted as one of the main component of video delivery over the Internet by the video streaming industry: Dynamic Adaptive Streaming over HTTP (DASH) (Kua, Armitage, & Branch, 2017).

As it is easily perceived from the acronym DASH, it means an adaptive protocol that operates over the HyperText Transfer Protocol (HTTP). This offers several benefits:

- The protocol was developed over HTTP especially because HTTP could resolve problems regarding Network Address Translation (NAT), firewalls and other middleboxes (Abdullah, 2016).

- HTTP has also become the main standard protocol on the Internet, being used on top of Transport Control Protocol (TCP) to deliver media content (Jimenez, Romero, Rego, Dilendra, & Lloret, 2016). This allows DASH servers to use common Web servers, thus reducing operational costs and allowing the deployment of caches to improve performance and reduce network load (Kua et al., 2017).
- DASH specification, which is detailed in a further chapter, allows clients to request video chunks independently from different servers, maintaining state at client side. This way, clients can switch between servers and avoid problems regarding load balance and fault tolerance (Goldenberg, Qiuy, Xie, Yang, & Zhang, 2004)(Liu et al., 2012).
- Relying on TCP reliability and inter-flow friendliness improves the likelihood that streaming traffic consumes only a fair fraction of the network bandwidth when sharing with other traffic (Kua et al., 2017).

Although DASH was a great improvement in video streaming, some natural characteristics of Adaptive BitRate (ABR) video streaming led to some unwanted consequences. ABR algorithms tend to be unstable and bursty, especially when sharing a bottleneck among a significant number of users. Different video applications may use different adaptation strategies which could lead to an increase network congestion. And, above all, ABR algorithms are normally based upon client’s perspective, having no knowledge of other clients or network perspective. Within this point of view, ABR would tend to act selfishly, aiming to maximize its own QoE regardless of network congestion or other users (Georgopoulos et al., 2013).

Besides being unstable and unfair in the network resources sharing, it has been reported that DASH applications use estimation tools that are inaccurate (Akhshabi, Begen, & Dovrolis, 2011)(Georgopoulos et al., 2013).

These reasons prove that although being very effective for its purpose, DASH can rapidly become insufficient in some conditions (large number of users, unstable connection, large variety of video applications within the same network, etc.). It became necessary to come out with a solution that could allow DASH to be efficient not only on its own. The proposed solution in the current dissertation is to gather all the existent work that has been done with the DASH protocol and to combine it with the Software Defined

Networking (SDN) paradigm. This way, it could improve the QoE for all the users within the same network and also achieve a more efficient management of the network resources.

The SDN paradigm implies the existence of a network controller responsible for all the network logic. Thus, the control plane is decoupled from the data plane since control functionality is removed from the network devices (switches, routers) and moved to the SDN controller. From then on, those devices will act like dummy devices with only low-level functions that forward packets (although normally using flow rules downloaded from the controller) and collect local statistics to send to the controller (Kreutz et al., 2014).

This way, the SDN controller becomes a central point of logic that receives data from all the forwarding devices, thus "building" a complete view of the whole network topology and network flows. This virtualization of the network turns the network much more flexible and easier to control and manage. The controller can set a number of criteria and intents that could permit the network to act, and most of all, react to the network variations in terms of either load or topological change. The main value of the SDN paradigm is that this approach makes the network "programmable" through software applications which is much easier than dealing with rigid hardware devices like switches and routers (Kreutz et al., 2014). The software approach of the network simplifies the communication between the control plane with the management plane and between the control plane and the data plane. This way, SDN simplifies the communication between managers and forwarding devices.

In the context of our scenario, this can be significantly useful since the degree of heterogeneity within the same network can easily oscillate. As we've seen in this section, the number of users, the characteristics of the devices, the different representations of the same video content are dynamic. This way, if the network has an entity with the ability to "see" and "understand" the whole network, it will become much more capable of dealing with the fluctuations of the network and can rise to a higher performance in management both for the users and the network itself.

1.2. Research Question

The research question that serves as a base for this dissertation is "Can the SDN paradigm and the DASH protocol be combined altogether to achieve a better user QoE for video streaming with variable bitrate and a more efficient management of the available network resources?"

1.3. Objectives

The main goal of the current work is to study how the deployment of an SDN controller can enhance the user perceived QoE when combined with the DASH protocol despite adverse networking scenarios. For this, we propose to develop and test a complete solution.

1.4. Investigation Method

To test the efficiency of this proposed solution, several tests were performed in distinct situations, such as follows:

- Neither use of the SDN controller nor DASH protocol;
- Use of the DASH protocol without the SDN controller;
- Use of the SDN controller without the DASH protocol;
- Both use of the SDN controller and the DASH protocol.

1.5. Dissertation Outline

The remainder part of the current dissertation has the following organization. Chapter 2 discusses the related literature. Chapter 3 details the design of the proposed solution. It also discusses the implementation of that solution. Chapter 4 is about the comprehensive evaluation of the proposed solution, including the performed tests and a complete discussion about their results. Finally, Chapter 5 presents some general conclusions about the current dissertation, and it highlights some guidelines about future work.

2 – Literature Review

As it was pointed out in the previous section, video streaming is already the most used application on the Internet. Consequently, several attempts have already been made (and still are) to enhance video streaming for the users, making this area of study one of the most active presently.

In this chapter, some key fundamentals of the video streaming evolution are presented. A special attention will be paid to the DASH protocol, the SDN paradigm and some of their principal concurrent. Then, the most relevant solutions in today's video streaming are presented, showing what concepts they rely on, what solution they use and what results they achieve.

2.1 Video Streaming Evolution

The early years of Internet did not expect that video delivery would gain such importance in the future. Delivering multimedia over the Internet was complicate due to several factors, lack of bandwidth's ahead. Live streaming was particularly complicate and it was thought that User Datagram Protocol (UDP) was more suitable since it did not imply waiting for acknowledgments nor retransmitting lost data. Though, UDP soon revealed problems passing NAT's and firewalls. In addition, it required dedicated servers and infrastructure which meant increase of costs and complexity. On another hand, TCP did not have the same problems but was considered not efficient due to its characteristics since video delivery was considered delay intolerant and high reliability was not considered mandatory. These reasons led to the development of several attempts to improve UDP so that it could deliver video streams properly and coexist with TCP traffic.

Real-time Transfer Protocol (RTP) was one of the first attempt for live streaming over UDP. Its original specification did not include questions regarding data delivery reliability or congestion control, although the last one has since been developed. To do so, the main perspective was, and still is, trying to match the video bitrate with the available network bandwidth. Besides those efforts, the reliability was still an issue and though it wasn´t considered essential, it could still reveal to be problematic especially when losing packets that included I-frames which could cause disruption in the video. Other protocols were developed to perform video streaming over UDP like Real-Time Streaming Protocol (RTSP), Session Description protocol (SDP) or RTP Control Protocol (RTCP). Though, they all found the same issues that RTP did.

Another approach that could be used to delivery video is the usage of Peer-to-Peer (P2P) networks. In this approach there are no centralized servers and each user is both client and server at the same time. P2P networks can be divided in two categories: tree-based and mesh-based. On one hand, the P2P tree-based type is organized in a hierarchical way where content is pushed from the base to the top. This approach is simple but highly sensitive since it could be severely affected when a peer fails. On the other hand, the mesh-based approach, clients randomly connect with each other, which elevates the complexity for managing that network and probably diminishing the network scalability.

Since the solutions based on UDP and P2P did not rise to the expectations that video streaming demanded, searchers decided to develop new solutions that could take advantage of TCP reliability and overcome its flaws. To do so, a playout buffer was added to compensate TCP fluctuations. In the first approaches, the content was encoded with a single bitrate and clients needed to download the entire file, although they could start the playout before the download was completed. The main problem with this approach was that the same quality video was transmitted to clients with different hardware capabilities and connection bandwidth.

To overcome this issue, the DASH protocol was developed so that heterogenous client devices could adapt them to their best possible playback. This protocol is detailed in the next sub-chapter.

2.2 DASH

The DASH protocol is the result of a call for proposal issued by the Motion Picture Expert Group (MPEG) in 2009 and the collaboration between several standard groups such as the 3rd Generation Partnership Project (3GPP). This proposal was the response for the market demand of a protocol that could bring more flexibility in video streaming also due to its exponential growth (Sodagar, Vetro, & Sodagar, 2011).

DASH protocol is designed to provide an adaptive streaming. For that, each video is encoded in several versions with different bitrates. Then, each version is divided into several chunks. All chunks of a same bitrate gathered together are part of the same representation. Though, that does not mean that the whole video could not be played with chunks from different representations. In fact, chunks from the same time interval are aligned at the time line so the client can switch between representations without the need to stop the playback.

The initialization of the protocol implies the existence of a Media Presentation Description (MPD) file where the different existing versions of the video are specified (see Figure 1).

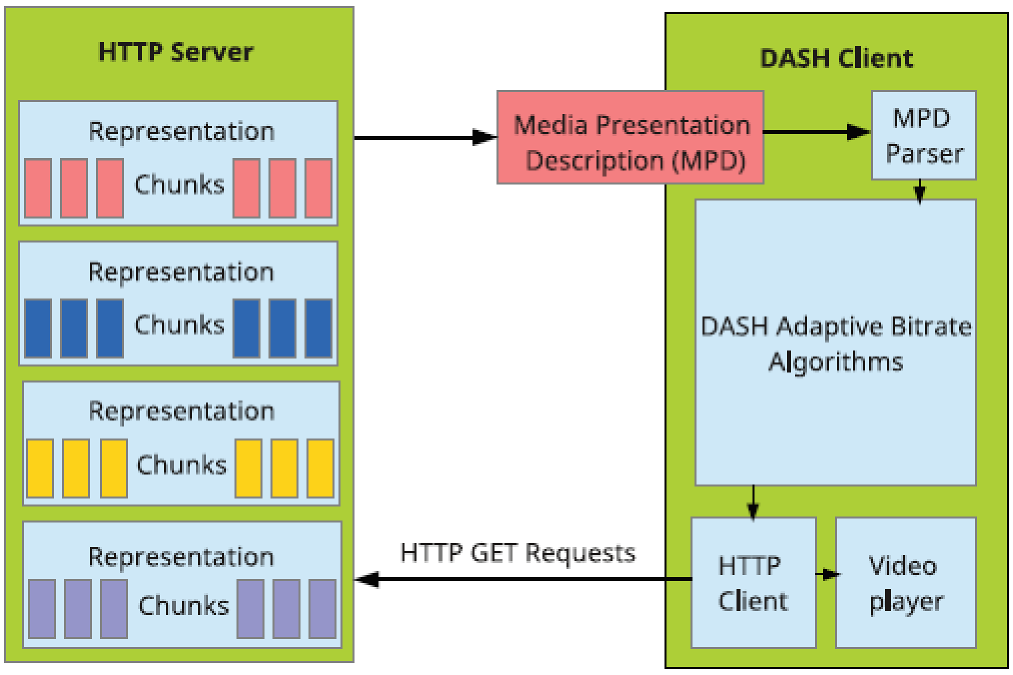


Figure 1: DASH Client-Server Architecture (Kua et al., 2017)

The MPD specification includes video's versions, metadata, codecs and a list of servers and respective Uniform Resource Locator (URL) which they can retrieve the video chunks from. Note that different versions may have a meaning that goes beyond the bitrate aspect. For example, you could have the same video with different versions in different languages.

After receiving the MPD from the server, the client decides which representation he or she wants to see and starts to request the according chunks through normal HTTP GET Requests (see Figure 2). DASH does not actually control the video transmission rate. The video transmission rate is regulated by the underlying TCP algorithm which is determined by the congestion feedback from the client-server network path (Kua et al., 2017).

After the download of the first chunks, the client keeps on periodically downloading new chunks in a chronological order throughout the session and normally keeps some chunks in the buffer to facilitate the playback and avoid rebuffering. During the session the client can switch among versions to meet his will or needs. At this point, it is important for the client playout algorithm to monitor the network bandwidth to adjust the video download with the available resources of both network and client device.

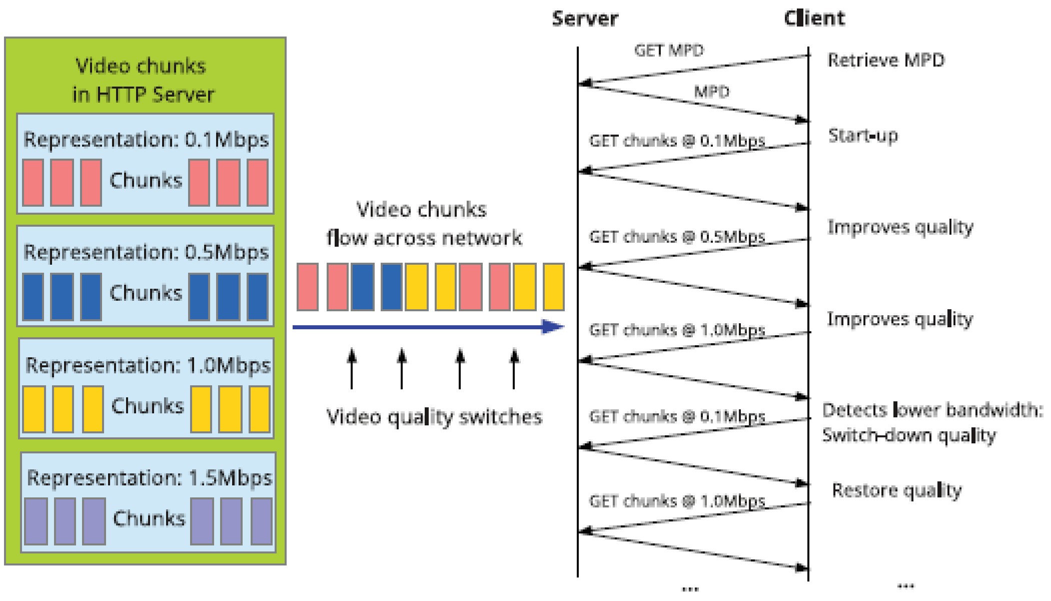


Figure 2: Timeline Illustration of DASH Adaptive Behavior (Kua et al., 2017)

This design provides multiple benefits:

- the client device can adapt its playback according to its available resources;

- the client can switch among different versions without the need to download the complete video;

- the client can easily “jump” at the time line;

- chunks can be retrieved from different servers bringing more flexibility.

Though, DASH is only responsible for describing the different existing versions and indicating what version does the client want. It is not actually responsible for delivering the media and has no power whatsoever in reconfiguring the network. This explains why DASH has, despite all its benefits, a limited power when acting alone. On its own, DASH can never perfectly match the network throughput. It can only select, from a limited set, a representation that goes below the network throughput, thus providing a smoother playback. Nevertheless, the DASH control loop between the client and server can easily become cumbersome if it suffers competition from some “predator” traffic such as the case of UDP. In fact, UDP flows can tend to consume all the available network bandwidth, starving other more-well behaved traffic, such is the case of TCP traffic.

2.3 Alternatives to DASH

As seen in the previous sub-chapter, DASH is a very efficient way to provide adaptive delivery through an HTTP structure. Though, DASH is not the only HTTP Adaptive Streaming (HAS) protocol on the market. In this sub-chapter several other proprietary popular HAS protocols are described.

Apple developed its own adaptive protocol called HTTP Live Streaming (HLS). Such as DASH, HLS also has a file like the MPD here called manifest where information about the different existing versions of the content, length of the chunks, etc. are presented. HLS also divides the content into several segments, even though the segments are normally longer here.

Microsoft's adaptive protocol is called Microsoft Smooth Streaming (MSS). The use of MSS also requires the use of a manifest file and is normally associated to the use of Microsoft Internet Information Services (IIS). Although, since MSS is based on Silverlight, it is necessary to install a separate Silverlight plugin for compatibility purpose.

Another example is Adobe's HTTP Dynamic Streaming (HDS) protocol. Since HDS was developed after HLS and MSS, its specification has similarities with both. Like HLS, in HDS the client periodically downloads data that allows it to know the URL's of the available chunks (Yuval Fisher, 2014). In HDS, the segments are encoded as fragmented MP4 files, i.e., they contain both the audio and the video, similarly to what happens in MSS. Though, Adobe's Flash rapidly changing access roadmap brings some instability in the deployment (Yuval Fisher, 2014).

In this solution we choose DASH because of its acceptance in the market and its main offered advantages (mentioned in sub-chapter 2.2) that can be easily set up. Although, to be completely effective, DASH needs help from an external mechanism to manage the network so that the rate control mechanisms implemented within the DASH system can offer the best possible QoE video playout at each client. One possible "partner" to do so is the SDN paradigm. In the following sub-chapter, SDN is described and it discussed how SDN can help DASH in enhancing the video streaming playout at the client devices.

2.4 SDN

Despite its growth over the last decades, Internet has suffered for a long time from a lack of flexibility. Networks remain rigid and, during many years, they were very hard to manage. This happened because most of the network's middleboxes had a very weak capacity to react to the network activity and the interaction between that equipment and the user/network manager was not efficient. Thus, devices configuration needed to be done individually often in a low-level language making it substantially harder for the network managers.

Additionally, networks were vertically integrated, i.e., the control plane and the data plane were bundled together, reducing the network flexibility to deploy innovative solutions in a fast and inexpensive way.

The problem is that this type of network represents most of the current Internet Protocol (IP) networks. Therefore, to change the Internet architecture, it would be necessary to replace all the IP equipment (Kreutz et al., 2014). Obviously, this is not feasible so the problem has to be solved in a different way.

One solution that has come up to fix this issue is the uprising of the SDN paradigm. The principal characteristic of this approach is that it breaks the vertical integration by separating the control plane and the data plane (see Figure 3). Thus, the logic is relocated in a controller which is separated from the forwarding devices. Thus, the network devices stop to have any logic and become simple forwarding devices which behavior will be determined by the logic decided by the controller. Additionally, the forwarding devices will collect and send statistics to the controller so it can build a global view of the network.

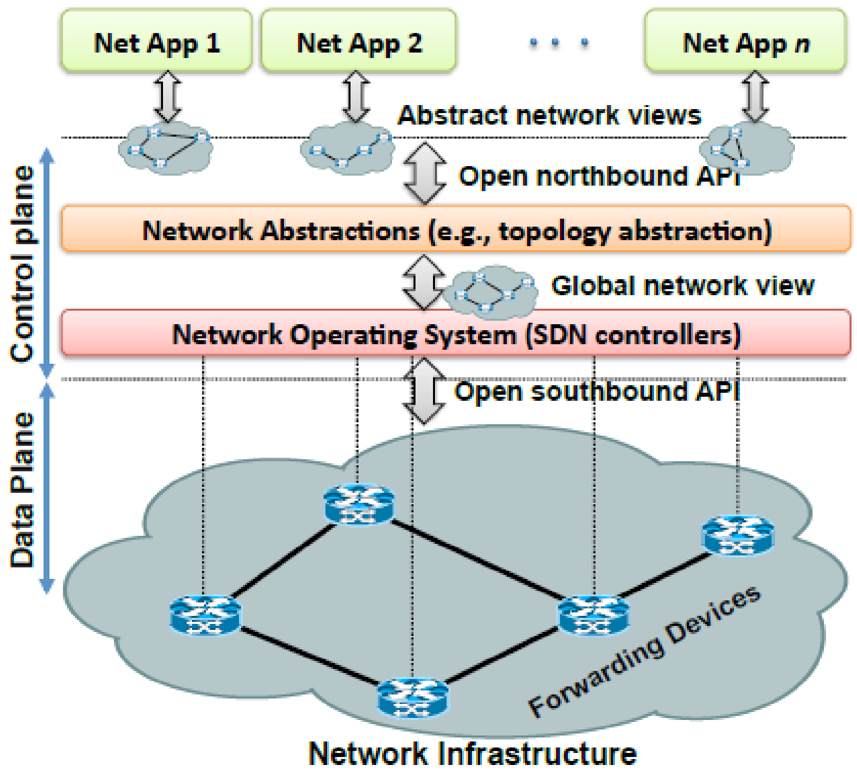


Figure 3: SDN Architecture and its Fundamental Abstractions (Kreutz et al., 2014)

Besides the increase of flexibility, the benefit of this paradigm is that it does not require the addition of much more physical equipment. The implementation of the controller is logic and relies on network virtualization. Another advantage of this approach is that the interaction between the controller and the network managers is performed at a high-level language which is friendlier to them. In this way, the network becomes "programmable" and much easier to manage.

The existence of a controller responsible for all the logic in the network also leads to a centralization of the logic. This way, it is easier to act on the whole network, the controller can inclusively act on its own and the controller can develop a global view of the network which makes it easier for it to take efficient decisions.

Though, that does not mean that the SDN paradigm mandatorily requires a unique controller responsible for all the logic, since it also becomes a single point of failure and it can suffer from scale limitations. There have been some implementations where the separation of the control and data planes were done by the implementation of several controllers. This can be more appropriated in case of very large networks. Though this approach is better in terms of reliability since it adds more points of failure, it loses the purpose of centralization and thus the controller does not have a complete view of the network anymore. So, the control of the entire network becomes more difficult to deploy, because we need to find valid ways to orchestrate the distinct controllers to obtain a global and efficient network control.

After the establishment of an SDN paradigm, it is possible to describe the system upon three planes (see Figure 4): 1) the control plane where the controller is located and where all logic is handled; 2) the data plane, where the forwarding devices are located and which communicates with the controller through the southbound interface; and 3) the management plane where the network applications and programs are located and which communicates with the controller through the northbound interface.

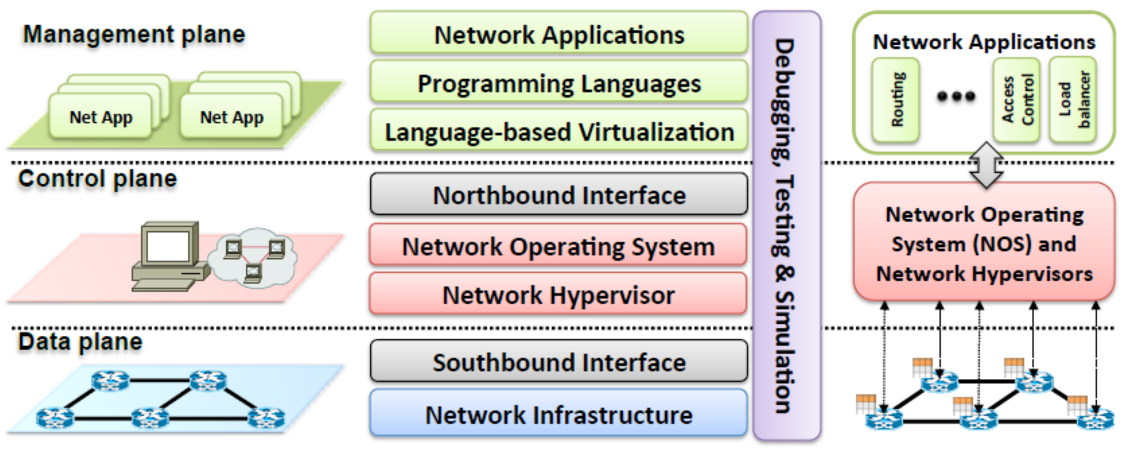


Figure 4: Software-Defined Networks in Planes, Layers, and System Design Architecture (Kreutz et al., 2014)

In this approach, typically the managers set policies and objectives that the network should respect to reach a desired behavior. Then, based on this information, the controller will define flow tables so that the forwarding devices can know what action they shall do when a packet arrives to the receiving port of each one of them.

With this approach, it is possible to design more flexible networks that have the capacity to act on their own, and so to react to significant network fluctuations. This way, the networks can rearrange paths, flows and adapt forwarding of the traffic to the network load so that the resources are used in the best possible way. Most of all, it is possible to build independent networks, which do not require the constant surveillance of managers. All these aspects, of course, apply too to the video streaming scope.

2.5 OpenDayLight Controller

There are currently many examples of existing SDN controllers, both commercial and open source. In the current research an open source project was chosen: OpenDayLight (Boron & Project, 2016). OpenDayLight (ODL) provides similar functionalities as other SDN controllers using many different protocols such as OpenFlow, Path Computation Element Protocol (PCEP), NetConf, etc. One specificity of ODL is that its installation is progressive, i.e., after setting up the ODL interface you'll need to add functionalities called Karaf features that will fit your needs and increase network capabilities. Additionally ODL has a framework called Model-Driven Service Abstraction Layer (MD-SAL) that allows developers to create their own Karaf features in the form of services and protocol drivers and connects them to one another (Boron & Project, 2016).

Thus, ODL is currently considered as one of the most popular SDN controllers. As an open source project, one of ODL's cornerstone is to ensure that it is as "neutral" as possible, i.e., that it is has the largest possible compatibility with devices from different vendors. In fact, and despite of being an open source project, a significant part of ODL contributors are coming from large Telecommunications players such as Cisco, Ericsson, ZTE, Intel, Juniper and many others (“Members - OpenDaylight,” 2018).

This way, ODL has become extremely versatile since it could be used with a large variety of southbound protocols and also be managed with a large variety of northbound applications explaining its growing popularity.

As mentioned above, MD-SAL is a way for developers to interact with the controller through data and interface models. The use of the MD-SAL is based on RESTCONF, which is an HTTP-based protocol that provides REST-like Application Programable Interfaces (API) to manipulate Yet Another Next Generation (YANG) modeled data and invoke YANG modeled Remote Procedure Calls (RPC) using eXtensible Markup Language (XML) or JavaScript Object Notation (JSON) as payload format (“Controller - OpenDaylight Documentation,” 2018).

Through RESTCONF, we have access to two datastores that contain data in the controller: 1) Config, that contains data inserted via controller, and 2) Operational, that contains other data. This way it is possible to use a Web browser or application to send HTTP requests to the controller to perform the desired operation. In the request it is fundamental to correctly set the Uniform Resource Identifier (URI), the content-type and the authentication information so that the request could be successful.

Like any other HTTP request, MD-SAL supports several different operations such as:

- **OPTIONS** – returns the XML description of the resources with the required request and response media types in Web Application Description Language (WADL);
- **GET** – returns a data node from the desired datastore (Config or Operational);
- **PUT** – updates or creates data in the datastore and returns the state about success;
- **POST** – creates the data if it does not exist;
- **DELETE** – removes the data node in the datastore and returns the state about success.

To get a better understanding of those operations we will see an example of a GET and PUT request in detail. Though before detailing those operations it is important to get to know the base classes that RESTCONF works with:

- **InstanceIdentifier** – represents the path in the data tree;
- **ConsumerSession** – used for invoking RPC's;
- **DataBrokerService** – offers manipulation with transactions and reading data from the datastores;
- **SchemaContext** – holds information about YANG modules;
- **MountService** – returns MountInstance based on the InstanceIdentifier pointing to a mount point;
- **MountInstance** – contains the SchemaContext behind the mount point;
- **DataSchemaNode** – provides information about the schema node;
- **SimpleNode** – possesses the same name as the schema node, and contains the value representing the data node value;
- **CompositeNode** – can contain CompositeNodes and SimpleNodes.

Figure 5 shows each step in a GET operation with URI restconf/config/M:N where M is the module name, and N is the node name.

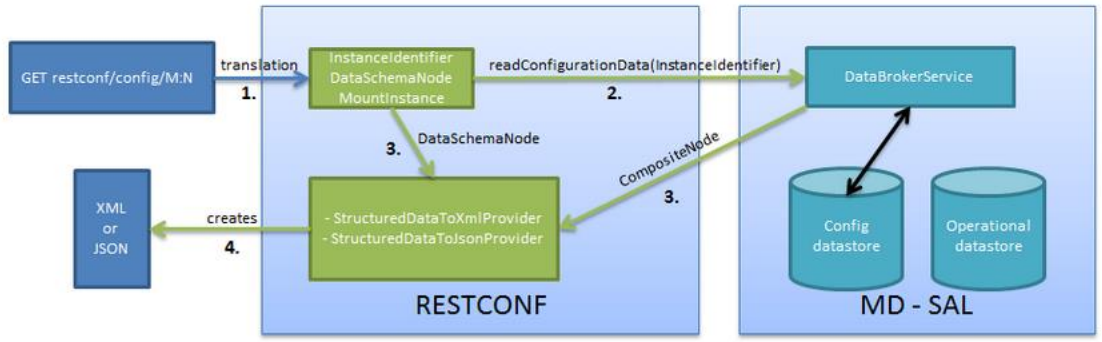


Figure 5: GET Request Timeline in the Config Datastore ("Controller - OpenDaylight Documentation," 2018)

In the first step, the introduced URI is translated to the correspondent path in the data tree that points to the data node. During this translation the information about the schema node that conforms to the pointed data node is obtained. After that, RESTCONF asks for the value of the data node based on the path obtained earlier. Then, DataBrokerService returns CompositeNode as data. One of those two providers will transform the data into an XML or JSON document. Finally, this document is then returned to the client as a response for his request.

For its turn, Figure 6 shows each step in a PUT operation still in the config datastore. The data could either be sent in XML or JSON.

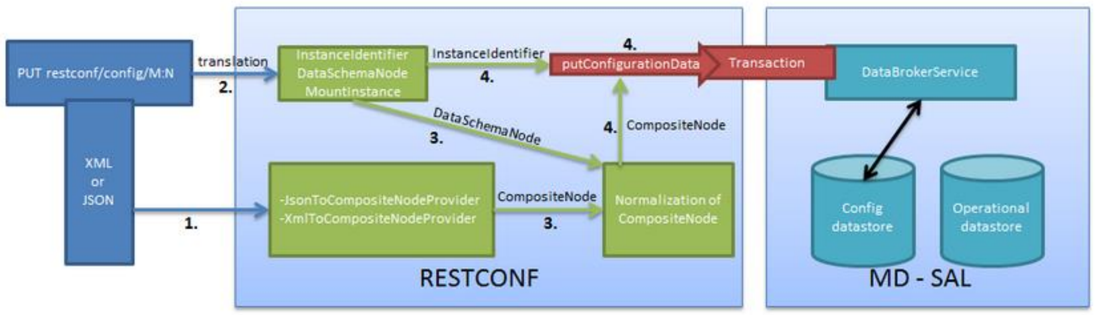


Figure 6: PUT Request Timeline in the Config Datastore ("Controller - OpenDaylight Documentation," 2018)

First, the data is sent to the selected provider for it to transform the data into CompositeNode. In this particular step we can see why it is so important to correctly define the Content-Type in the HTTP header request. It is based on this information that that the correct provider will be chosen. The introduced URI is then translated to the correspondent path in the data tree that points to the data node. During this translation, the information about the schema node that conforms to the pointed data node is obtained. CompositeNode can be normalized by adding additional information from DataSchemaNode. After that, RESTCONF begins the transaction and puts CompositeNode with the path into it. Then based on the result of the transaction, a status code is returned to the client as a response for his original request.

In this project, ODL was used with OpenFlow protocol. In the next sub-chapter, a description of the protocol is done.

2.6 OpenFlow Protocol

OpenFlow's creation is tighten with the concept of SDN. OpenFlow is an open source protocol that could be used along switches from different vendors, no matter what their interface or scripting language, enabling managers to act in a more efficient way on more complex and heterogenous networks using a single protocol. This way, OpenFlow can be considered as an SDN enabler.

OpenFlow acts on a forwarding plane using flow-based tables. In the past, networking has either been based on switching (within a given network) or routing (between different networks). With OpenFlow, networking can not be exactly considered as neither of those two as it could be based on both layers, or even based on other layers. The key idea in OpenFlow is that what is important to manage within a network are flows. The concept of a flow is that a particular service delivers a series of packets that share a common set of header fields which are specified in the match part of an OpenFlow rule. That part of a rule aids each switch to identify, from all the received messages, the ones that satisfy it. After this positive identification, all the messages with “common attributes” are processed exactly in the same way by the action part of the same OpenFlow rule. In this way, it is created the concept of a flow, which is the set of the received messages that share a common set of cross-layer header fields. Consequently, rather than focusing on individual packets, it is important to see packets as parts of larger sets. Individual packets do not make sense isolated, they only do when bundled with packets from the same flows as they build a particular service.

This way, instead of defining switching or routing rules, OpenFlow defines flow rules that can contain fields for elements such as source & destination Media Access Control (MAC) addresses or source & destination IP and could then be considered as a cross layer protocol.

After setting those flow rules, OpenFlow will build forwarding tables that will serve as behavior models that forwarding devices will follow based on conditions stated in the forwarding tables for the incoming packets. Forwarding devices will then take a determined action on what to do about those packets accordingly to the receive information. Some of OpenFlow’s advantage are both the variety of filters that can be applied regarding the incoming packets and the actions that can be taken regarding those same packets. Additionally, if one packet does not match any of the filter defined in the forwarding tables, the device has the possibility to send the packet to the controller that will then decide what to do with it.

In the next sub-chapter, a more detailed overview of the 1.3 OpenFlow specifications is presented.

2.6.1 OpenFlow Switches

To get in a more detailed description, we can describe an OpenFlow switch as one or more flow tables, as well as one group table. One advantage of group tables is that they may define rules that can be applied simultaneously to several flows. This way, it diminishes the number of rules within the switch and improves its performance. These switches can be controlled through an OpenFlow channel. The flow table(s) and the group table will inspect packets and forward them accordingly with the existing configuration based on the flow rules. The OpenFlow channel will allow communication between the OpenFlow switches and the network controller. Thanks to OpenFlow, the controller will be able to create, modify and delete flow rules that will define the switches behavior in the network. The operations regarding the flow rules could both be done “automatically” by the controller in response of network changes in either load, traffic type, or network topology. This way of operating the SDN system is designated as the reactive mode. Alternatively, the SDN system can predict and anticipate a specific network status, downloading a set of flow rules to the switches, before the occurrence of that particular network status. This alternative way of operating is designated as the proactive mode.

One particularity of OpenFlow is that it supports several port types, three to be more accurate: physical ports, logical ports and reserved ports. In fact, the number of OpenFlow ports could be different from the hardware original number of ports since it is possible to disable some original network interfaces for OpenFlow or add supplementary ports to the OpenFlow switch. In general, physical ports correspond to the actual existing ports on the original switch. Although, in some cases, like when an OpenFlow switch is virtualized over the switch hardware, they may correspond to a virtual slice of the corresponding hardware interface of the switch (Heller, 2009). In an opposite way, logical ports do not correspond directly to existing ports on the hardware. The main advantage behind the use of logical ports is that they may be mapped to various physical ports. This way, when a packet is received on a logical port it is sent to the controller; both the logical port and its correspondent physical port(s) will be reported to the controller. Finally, reserved ports specify generic forwarding actions such as sending to the controller, forwarding or forwarding using non-OpenFlow methods.

2.6.2 Flow Tables

After describing the components of OpenFlow switches and OpenFlow ports we will get into a more detailed description of flow tables. As mentioned in sub-chapter 3.2, flow tables serve as configuration "matrices" that will decide how packets are handled in the network and, consequently, define the network behavior. Figure 7 shows the main components of a flow entry belonging to a flow table of an SDN-based switch.

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

Figure 7: Main Components of a Flow Entry in a Flow Table. OpenFlow 1.3 (Heller, 2009)

A flow table corresponds of a series of flow entries and each flow entry has the following components:

- **Match fields** – responsible for matching packets, i.e., determine which packets will be handled by which entries. The matching is based on ingress ports and packets headers (e.g., MAC source addresses, IP destination addresses, etc.) and can optionally be based on metadata specified by a previous table;

- **Priority** – this entry is used when packets match with several entries to determine which entry has priority on this packet processing;
- **Counters** – counts the number of matched packets with the current rule;
- **Instructions** – actions to take with the matched packets;
- **Timeouts** – defines the maximum amount of time or idle time before the flow is expired by the switch;
- **Cookie** – this entry is used to indicate past activity of the entry and can be used by the controller to filter flow statistics, flow modification or flow detection.

The flowchart in Figure 8 can help understanding packet matching in OpenFlow and the chronology events in the switch when they are faced with incoming packets.

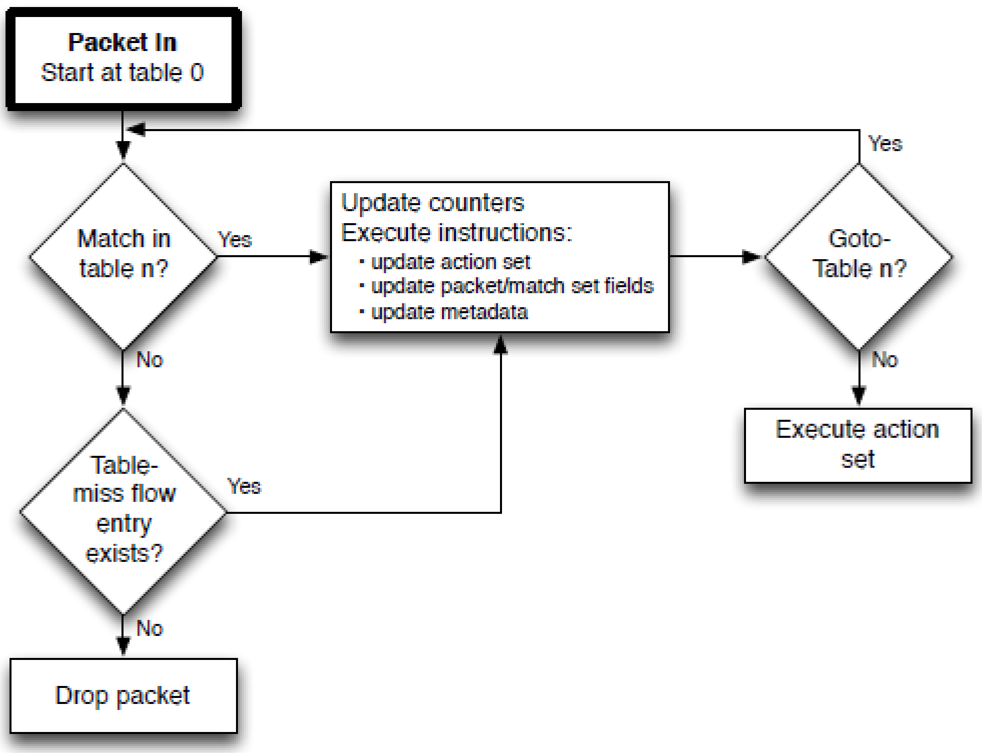


Figure 8: Flowchart Detailing Packet Flow Through an OpenFlow Switch. OpenFlow 1.3 (Heller, 2009)

There are some other important aspects worth mentioning about flow tables. The first one is that there are two types of OpenFlow switches: OpenFlow-only switches that only support OpenFlow operations and OpenFlow-hybrid switches that support “normal” Ethernet switching operations (L2 Ethernet switching, Virtual Local Area Network (VLAN) isolation, L3 routing, etc.) besides following the OpenFlow pipeline (only available since version 1.1 of OpenFlow and later versions). The second aspect concerns precisely this OpenFlow pipeline (see Figure 9). In an OpenFlow switch there are a series of flow tables, each one containing a series of flow entries (although a switch can have just a single flow table).

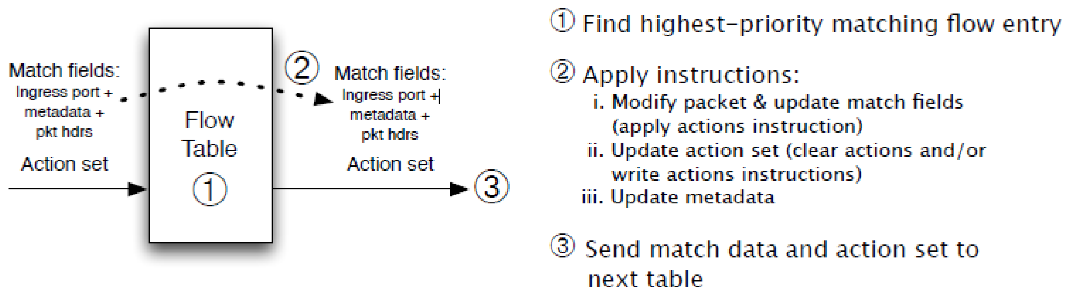

Match fields:
Ingress port +
metadata +
pkt hdrs
Action set
Flow Table
①
②
Match fields:
Ingress port +
metadata +
pkt hdrs
Action set
③
① Find highest-priority matching flow entry
② Apply instructions:
i. Modify packet & update match fields (apply actions instruction)
ii. Update action set (clear actions and/or write actions instructions)
iii. Update metadata
③ Send match data and action set to next table

Figure 9: Packet Flow Through the Processing Pipeline. OpenFlow 1.3 (Heller, 2009)

As referred previously, when a flow table receives a packet it will "search" for matching entries to determine what actions to take on this packet. Though, those flow tables in an OpenFlow switch are all numbered, starting at 0 and the OpenFlow pipeline specifies that after been handled by a certain flow table a packet can only be handled by a flow table with a higher number (see Figure 10).

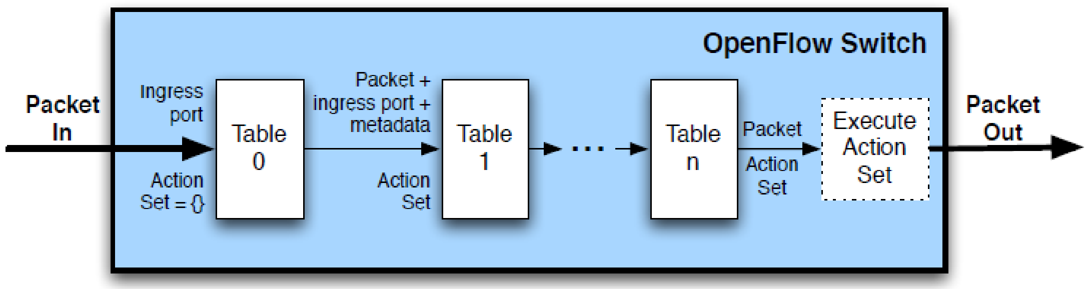


Figure 10: Tables Pipeline. OpenFlow 1.3(Heller, 2009)

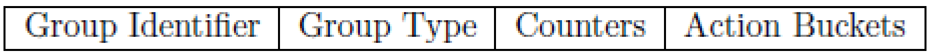
Thus, a packet always begins its "path" within the OpenFlow pipeline in the flow table 0. There the packet is matched with the flow entries of the flow table. Then according to the outcome of the matching, the packet can proceed to another flow table though it can never proceed to a flow table with a lower number. Whenever the packet is not directed to another flow table, the pipeline stops and the packet is processed with its associated action set and usually forwarded to the exterior of the forwarding device.

In cases where packets do not match any of the entries of a flow table, a table miss happens. A table-miss entry in the flow table can specify how to process unmatched packets: options include dropping them, passing them to another table or sending them to the controller.

2.6.3 Group Tables

Besides the possibility of forwarding accordingly to flow tables, OpenFlow enables additional forwarding methods based on group tables. Like flow tables, group tables consist of group entries that flow entries can point to enable those additional forwarding methods. This enables to get a higher scalable SDN system.

In Figure 11, we can see the components existing in each group entry in the group tables which have the following purposes:



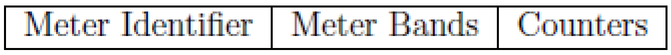
Group Identifier	Group Type	Counters	Action Buckets

Figure 11: Main Components of a Group Entry in the Group Table. OpenFlow 1.3 (Heller, 2009)

- **Group identifier** – unique 32-bit unsigned integer for each group;
- **Group type** – existing to determine the characteristics of the group. Existing group types are:
    - ***all*** (used to execute all buckets in a group in cases like multicast or broadcast),
    - ***select*** (unlike all, this type is used to execute only one bucket in the group),
    - ***indirect*** (use to execute the one defined bucket in the group, i.e., when groups only support a single bucket) and
    - ***fast failover*** (used to execute the first live bucket. In cases where no buckets are live, the packets are dropped.);
- **Counters** – counts the number of matched packets;
- **Action buckets** – contains the list of existing buckets where each action bucket contains a set of actions to execute and associated parameters.

2.6.4 Meter Tables

In addition to flow tables and group tables, another way to select packets for forwarding methods are the one based on meter tables. A meter table consists of several meter entries defined to correspond to some characteristics present in flows (see Figure 12). Actions based on meter tables are one way to implement Quality of Service (QoS) mechanisms such as rate-limiting and can be combined with port-queues to implement more complex QoS frameworks.



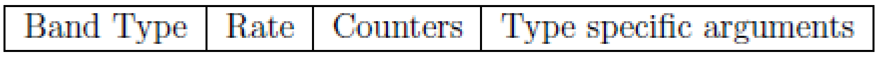
Meter Identifier	Meter Bands	Counters

Figure 12: Main Components of a Meter Entry in the Meter Table. OpenFlow 1.3 (Heller, 2009)

In a meter table, each meter entry has the following components:

- **Meter identifier** – unique 32-bit unsigned integer for each meter;
- **Meter bands** – contains the list of existing meter bands where each meter band specifies the rate of the band and how the packets are processed;
- **Counters** – counts the number of matched packets.

In every meter entry, there could be specified more than one meter band (see Figure 13). The meter band specifies the rate at which band is applied and how the processing of the packets should be done. Though each packet can only be processed by one meter band at a time, the selection on what band is applied depends on the current measured rate. The rate applied is the first configurated one below the measured rate.



Band Type	Rate	Counters	Type specific arguments

Figure 13: Main Components of a Meter Band in a Meter Entry. OpenFlow 1.3 (Heller, 2009)

A meter band is characterized by its rate and has the following components:

- **Band type** – defines the way that packets are processed;
- **Rate** – definition for the lowest band applicable;
- **Counters** – counts the number of matched packets;
- **Type specific arguments** – may be used for some additional arguments.

2.6.5 OpenFlow Channel

One fundamental concept behind OpenFlow is the constant communication between the switches and the controller. This communication is made upon an OpenFlow channel for each switch and could be done through three message types:

- **Controller-to-switch messages** – messages initiated by the controller that may or may not require response. Those messages can either be to query the switch about current state, configuration, etc. or to set orders to the switches;
- **Asynchronous messages** – messages sent from the switches to the controller without its solicitation. They are mainly for acknowledge the controller about events happening in the switches such as packet arrival, switch state change or errors;
- **Symmetric messages** – messages sent in either way without the solicitation of any of the parts. They are manly used to acknowledge each other upon a new connection or that the connection is still alive.

So OpenFlow can be synthetized as a protocol that enables communication between the controller and the switches.

2.7 SDN Applications

In this sub-chapter we describe several examples of applications that can be installed on top of the ODL controller to get an easier interaction and help managing the controller and, therefore, the network resources.

2.7.1 OpenDayLight Flow Manager (OFM)

OFM is an application used to quickly visualize OpenFlow data such as current topologies, existing flows, hosts and statistics. It has an easy installation and could then be accessed through a web browser.

After initialization, OFM gathers all OpenFlow inventory and renders it. In Figure 14, we can see the output of the *Topology Tab* where we can see the current existing switches, links and hosts.

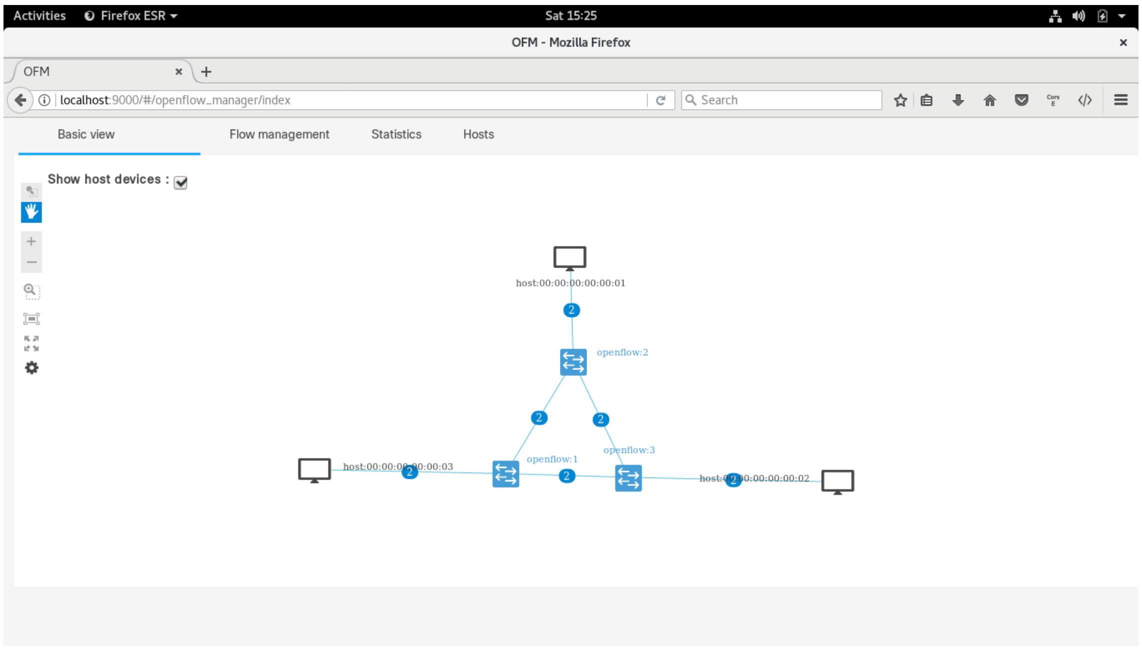
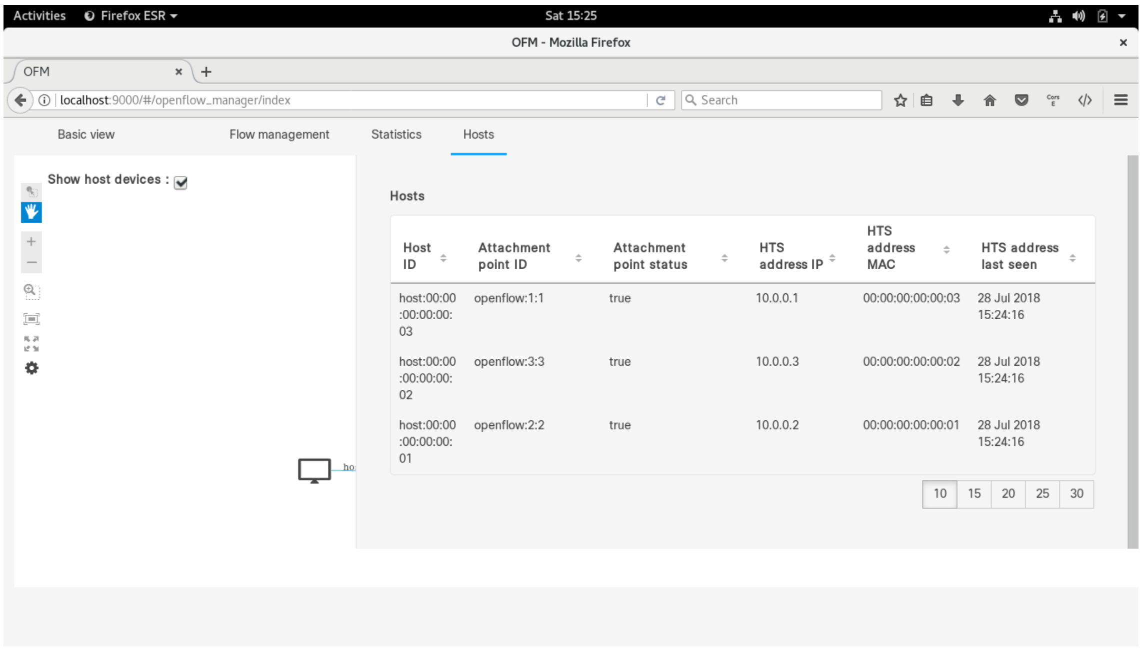


Figure 14: Topology in OFM

In addition to the topology, hosts can be seen in their own Hosts Tab (see Figure 15) where all details are presented including hosts ID, switch which they are connected to, IP address, and MAC address.



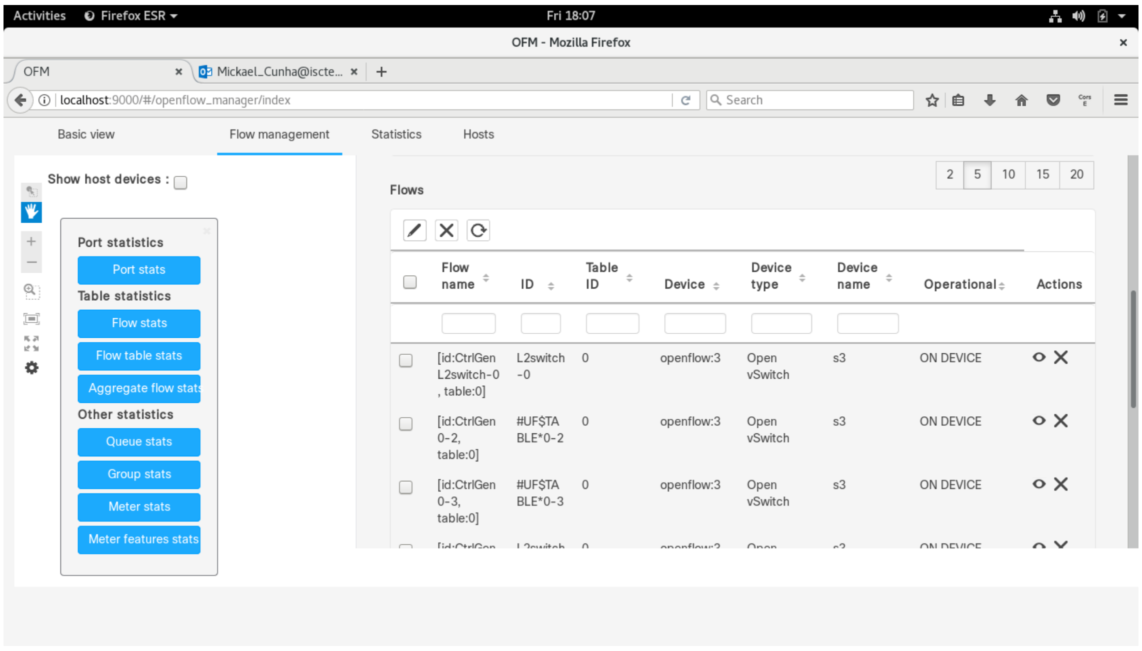
Hosts

Host ID	Attachment point ID	Attachment point status	HTS address IP	HTS address MAC	HTS address last seen
host:00:00:00:00:00:03	openflow:1:1	true	10.0.0.1	00:00:00:00:00:03	28 Jul 2018 15:24:16
host:00:00:00:00:00:02	openflow:3:3	true	10.0.0.3	00:00:00:00:00:02	28 Jul 2018 15:24:16
host:00:00:00:00:00:01	openflow:2:2	true	10.0.0.2	00:00:00:00:00:01	28 Jul 2018 15:24:16

10 15 20 25 30

Figure 15: Hosts Tab in OFM

The *Flow Management tab* includes not only the possibility to observe the details about every flow but also provides the ability to write new flows that can be added using any normal flow characteristics (see Figure 16).



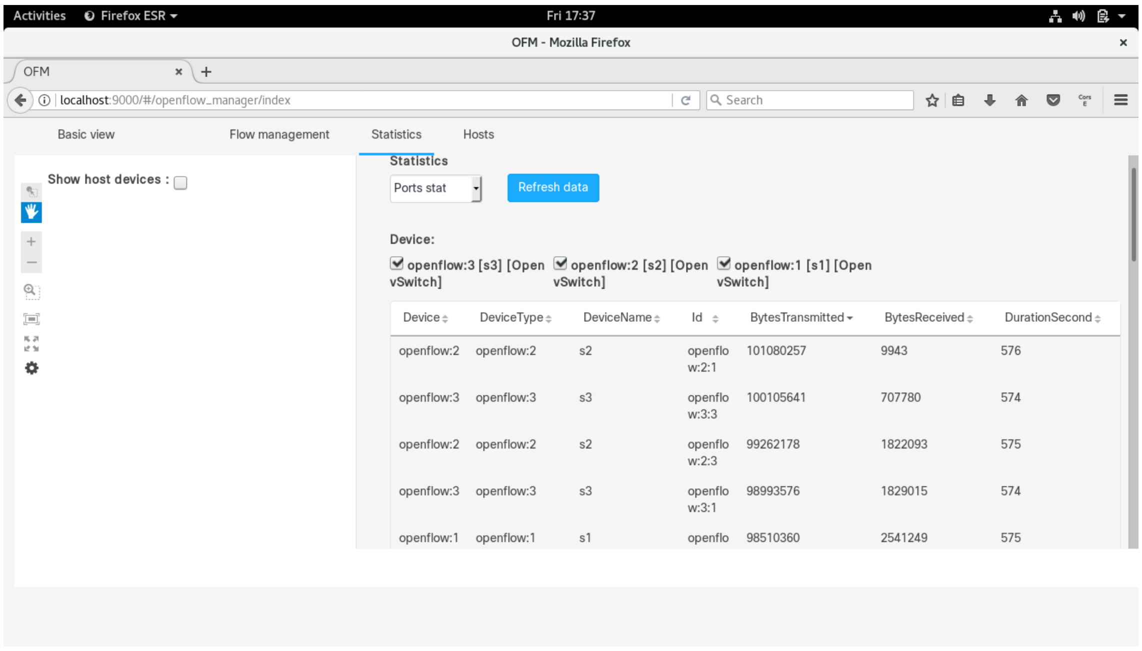
2 5 10 15 20

Flows

Flow name	ID	Table ID	Device	Device type	Device name	Operational	Actions
[id:CtrlGen L2switch-0, table:0]	L2switch-0	0	openflow:3	Open vSwitch	s3	ON DEVICE	
[id:CtrlGen 0-2, table:0]	#UF$TABLE*0-2	0	openflow:3	Open vSwitch	s3	ON DEVICE	
[id:CtrlGen 0-3, table:0]	#UF$TABLE*0-3	0	openflow:3	Open vSwitch	s3	ON DEVICE	
[id:CtrlGen	L2switch	0	openflow:3	Open	s3	ON DEVICE	

Figure 16: Flows Tab in OFM

Another great aspect about OFM is the possibility to check statistics about the network activity. Furthermore, the statistics can be analyzed through a large amount of options. This way, it is possible to gather several statistics such as ports activity, flows, flow tables, queues, groups or meters (see Figure 17).



Statistics

Device:

☑ openflow:3 [s3] [Open vSwitch] ☑ openflow:2 [s2] [Open vSwitch] ☑ openflow:1 [s1] [Open vSwitch]

Device	DeviceType	DeviceName	Id	BytesTransmitted	BytesReceived	DurationSecond
openflow:2	openflow:2	s2	openflow:2:1	101080257	9943	576
openflow:3	openflow:3	s3	openflow:3:3	100105641	707780	574
openflow:2	openflow:2	s2	openflow:2:3	99262178	1822093	575
openflow:3	openflow:3	s3	openflow:3:1	98993576	1829015	574
openflow:1	openflow:1	s1	openflo	98510360	2541249	575

Figure 17: Statistics Tab in OFM

2.7.2 SFlow-RT

Another application that can run on top of ODL is SFlow-RT. It is an application used to gather statistics at real time where we can add several sub-applications to analyze the data through different perspectives.

The retrieving of the data is performed upon sFlow agents embedded in network devices, hosts and applications and are then converted into actionable metrics. Besides having several pre-configured sub-applications, it is also possible to develop custom sub-applications that will use the data fetched by the sFlow agents to visualize the desire feature.

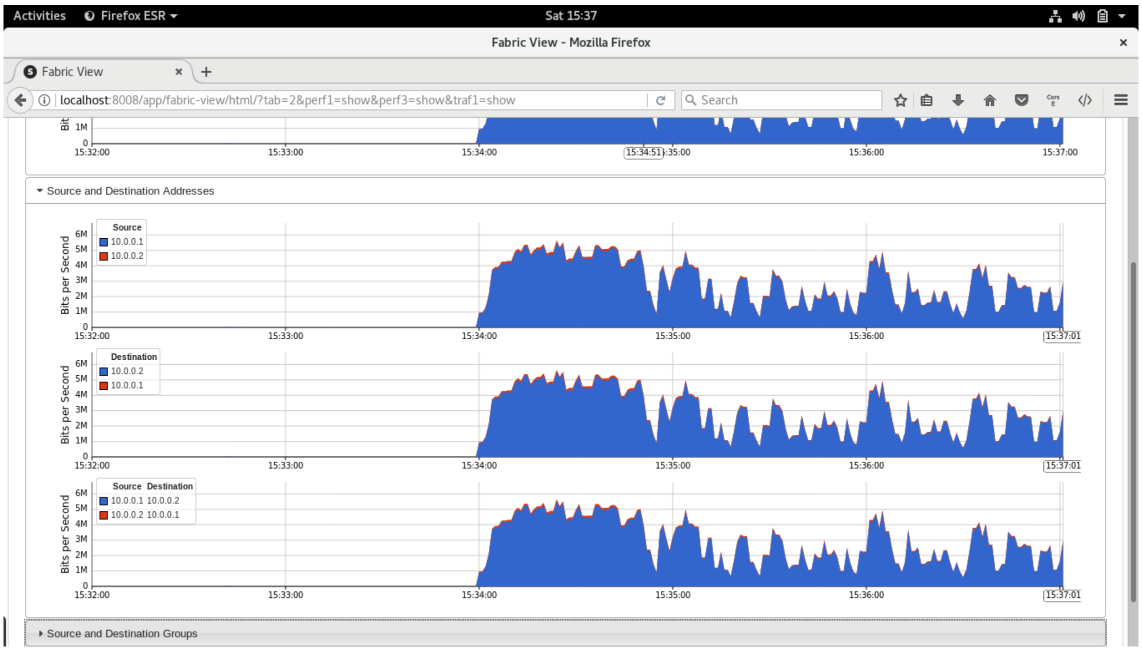
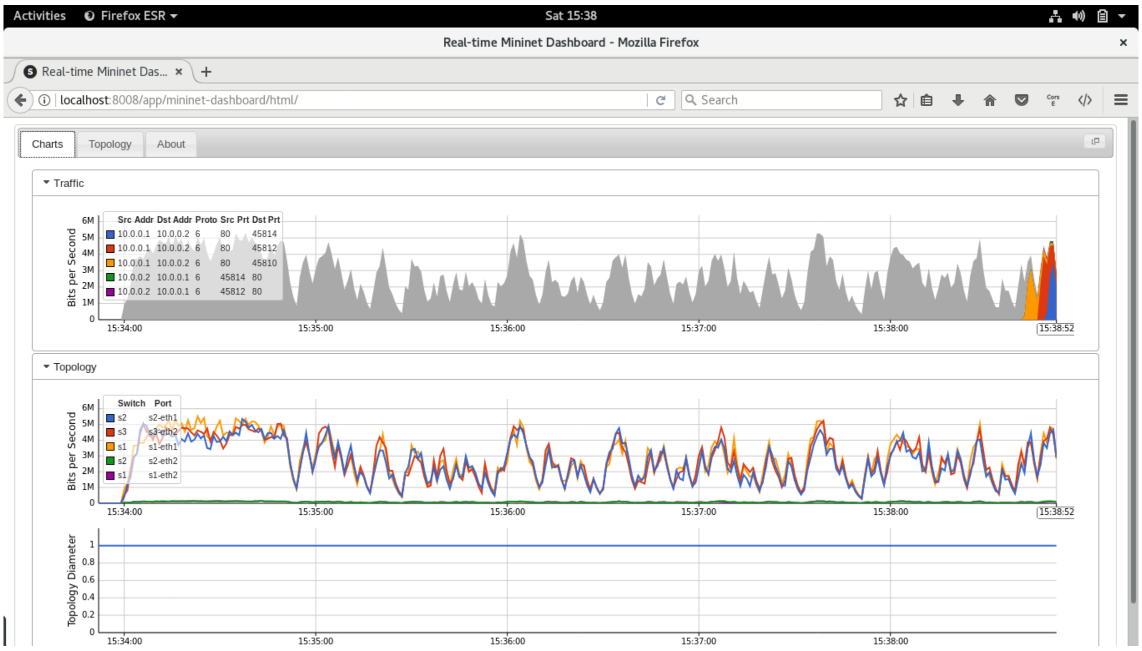


Figure 18: Fabric View Sub-Application

Pre-configured sub-applications include options to check data upon several aspects such as ports, IP address, protocol types or flows. For example, if we compare sFlow-RT with OFM, OFM only showed statistics upon a counter that had to be refreshed to check evolution, where sFlow-RT displays the statistics at real time and through charts, which are easier to observe and interpret (see Figure 18 and Figure 19).



	Src Addr	Dst Addr	Proto	Src Prt	Dst Prt
🟦	10.0.0.1	10.0.0.2	6	80	45814
🟥	10.0.0.1	10.0.0.2	6	80	45812
🟧	10.0.0.1	10.0.0.2	6	80	45810
🟩	10.0.0.2	10.0.0.1	6	45814	80
🟪	10.0.0.2	10.0.0.1	6	45812	80

	Switch	Port
🟦	s2	s2-eth1
🟥	s3	s3-eth2
🟧	s1	s1-eth1
🟩	s2	s2-eth2
🟪	s1	s1-eth2

Figure 19: Real Time Mininet Dashboard Sub-Application

2.8 Alternatives to SDN

Besides traditional SDN solutions there are some alternatives “hybrid” solutions that look beyond OpenFlow. In this sub-chapter some of these solutions are presented.

The first of these solutions relies on the implementation of a Path Computation Element (PCE) which is an entity that can compute a network path based on a network graph. PCE are especially used in needs of more complex paths across several domains (Dhody, Architect, & Technologies, 2015).

Another example is the Application-Layer Traffic Optimization (ALTO). ALTO is a protocol that can provide abstract network information to applications. The applications could then use the obtained information to use the network resources in a more efficient way (Dhody et al., 2015).

Interface to Routing System (I2RS) is an interface developed by the Internet Engineering Task Force (IETF). It was built to cover some functions and features that were not covered by the SDN controller such as: control of routers, control of routing protocols and management of the “routing system”. These tasks had to be done manually through the Command Line Interface (CLI). That is the reason why I2RS was developed, to provide an approach that could make the use of the YANG Models and the RESTCONF applications simpler (Dhody et al., 2015). Besides that, I2RS is near real time which can turn the communication with the controller much faster without the need to get into specific semantics of a routing protocol (Dhody et al., 2015). In addition, since the routing protocols are aware of what is being installed on the routing tables, the information injected by I2RS can still be “seen” by other local routing processes (White Russ, 2017).

2.9 Related Work

In this sub-chapter, the main existing proposals that are related to this dissertation are discussed. Table 1 summaries those proposals highlighting whether or not they use SDN and/or DASH and stating the main qualities and flaws of each proposal.

Table 1: List of Related Research

Related Proposal's Title	Streaming	SDN	DASH	+	-
SDN Based User-Centric Framework for Heterogeneous Wireless Networks (Lu, Lei, Wen, Wang, & Chen, 2016)	Can be applied	Yes	No	Has the capacity to get closer to the user expectations.	Doesn´t apply directly to streaming nor have adaptive mechanisms. Oriented to wireless networks.
On optimizing scalable video delivery over media aware mobile clouds (Politis, Tselios, Lykourgiotis, & Kotsopoulos, 2017)	Yes	Yes	No	Controller is used along with QoE-monitoring entities to build a global vision of the network.	Does not use DASH and so can not use the benefits of existing HTTP implementations.
Variable-Threshold Buffer Based Adaptation for DASH Mobile Video Streaming (Abuteir et al., 2017)	Yes	No	Yes	Considers the instability of network bandwidth.	Doesn´t have mechanisms to modify and thus maximize network resources.
Improving Fairness, Efficiency, and Stability in HTTP-Based Adaptive Video Streaming With FESTIVE (Estive, Jiang, Sekar, & Zhang, 2014)	Yes	No	Yes	Notions of Fairness, Efficiency and Stability among several players.	Doesn´t have mechanisms to modify and thus maximize network resources.
Dynamic Adaptive Video Streaming on Heterogeneous TVWS and Wi-Fi Networks (Bedogni et al., 2017)	Yes	No	Yes	Takes advantage of unused frequencies of the electromagnetic spectrum.	Doesn´t have mechanisms to modify and thus maximize network resources.
QoE optimization through in-network quality adaptation for HTTP Adaptive Streaming (Bouten et al., 2012)	Yes	No	HAS	Takes advantage of proxies' existing caches. The algorithm can be oriented to different objectives.	Deployment lacks an entity that could have a complete view of the network, thus improving the efficiency of the solution.
SDN for Segment based Flow Routing of DASH (Cetinkaya, Karayer, Sayit, & Hellge, 2015)	Yes	Yes	Yes	Has a simple architecture where download paths can be different for each segment.	Results do not show a very high improvement.
SDN-based Throughput Allocation in Wireless Networks for Heterogeneous Adaptive Video Streaming Applications (Taha, Garcia, Jimenez, & Lloret, 2017)	Yes	Yes	Yes	Uses TCP throughput, brings more fairness and stability to the network.	Experimental setups only use one network type.
Design and Experimental Evaluation of Network-assisted Strategies for HTTP Adaptive Streaming (Cofano et al., 2017)	Yes	Yes	Yes	Introduce a Video Control Plane more oriented to specific video needs. Uses different approaches and different algorithms to evaluate the results.	Experimental setups only use one network type.
An SDN Architecture for Privacy-Friendly Network-Assisted DASH (Kleinrouweler, Cabrero, & Cesar, 2017)	Yes	Yes	Yes	Implements a Service Manager that has two different adaptation mechanisms that bring more stability and a higher bitrate. Has a privacy mechanism.	Applying architecture demands a very high computing power to the Service Manager and thus would be

					very hard to implement in a larger network and it can easily deplete the battery energy of mobile devices.
Towards Network-wide QoE Fairness Using OpenFlow-assisted Adaptive Video Streaming (Georgopoulos et al., 2013)	Yes	Yes	Yes	Considers the heterogeneity of devices, besides bringing more QoE and fairness.	Uses a utility function only designed to fit the test source video.
A Scalable User Fairness Model for Adaptive Video Streaming Over SDN-Assisted Future Networks (Mu et al., 2016)	Yes	Yes	HAS	Develops a model based upon three metrics: video quality, switching impacts and cost efficiency. that evaluate not only, the factors that affect QoE on the user side, but also the effects on the network operation costs.	Although their research is based on HAS and SDN, it doesn´t evidence the relevance of each technology in the solution. Furthermore, authors referred that the notion of fairness can be more explored in future work.
A Policy-Based Framework Towards Smooth Adaptive Playback for Dynamic Video Streaming over HTTP	Yes	Yes	Yes	Uses different scenarios to ascertain the importance of each tool e to verify that their importance can be maximized when used simultaneously.	Traffic discrimination can be more explored. Results can be evaluated by more objective QoE metrics.

First of all, it is important to refer that the SDN paradigm has many applications besides the video streaming scope. Though, in pretty much all its applications, SDN has one common purpose, this purpose is to make the network more flexible. With such heavy network loads as the ones occurring nowadays, it is important to implement mechanisms capable of reacting to instantaneous traffic variations and managing the network flows in an efficient way. This aspect could be very important is some environments such as datacenters or in Telecommunications networks.

For instance, (Zorzi, Zanella, Testolin, De Filippo De Grazia, & Zorzi, 2016) developed a new paradigm where concepts of deep learning and Machine Learning (ML) are added to the SDN paradigm to build autonomous networks. This way, in addition to having the capability to understand and react to the network activity, the network can develop the ability to make decisions on its own based on previous experiences. Thus, it reduces the necessary interaction between the controller and the human managers. In a lightly similar way (Lu et al., 2016) built a framework that is oriented to the user’s perspective where the use of SDN allows the network to have a closer fit to the user’s necessities. This, obviously can be used in the video streaming scope but also with other traffic applications. Though, this framework is oriented to wireless networks. Besides that, as referred above, this solution was not directly developed for the video streaming scope. Thus, it does not use the DASH protocol or no other adaptive protocol, having no specific adaptive mechanism that could be used in the video streaming scope.

In the field of enhancing video streaming though the SDN paradigm, (Politis et al., 2017) came out with a solution where the controller is used along with QoE-monitoring entities. That combination is then used to build a global vision of the "ecosystem". The paper also appeals to the benefits of network virtualization in the management of network resources. Though this solution is not based on DASH, but on RTP, thus it does not take advantage of existing HTTP structures. To evaluate their results, authors drove a series of experiments with a test-bed that included the SDN network and an emulated Long-Term Evolution (LTE) Enhanced Packet Core part. Their results have shown that the percentage of packet loss dropped significantly: 40% for the base layer and 70% for the enhancement layer. Jitter was reduced by 16% for the base layer and 2% for the enhancement layer and a small improvement has also been noticed for the throughput: respectively 3% and 5%. In terms of PSNR, the solution has shown improvements of 1.78-2.34 dB depending on which video was used.

SDN can be used for other applications besides video streaming. However, there are also many solutions that do not use SDN for enhancing the QoE in video streaming. Many of them focus on improving the performance of the adaption algorithms. Though, this perspective brings shorts advantages in crowded networks as the streams continue to have a bitrate too low. This happens because with this approach there is no mechanism capable of optimizing the existing network resources (Kleinrouweler et al., 2017).

Regarding the adaptation algorithm, there are two main strategies. One based on the Buffer Occupancy (BO) while the other is based upon the throughput estimation.

(Abuteir et al., 2017) developed an example of a solution based on BO. In this solution the buffer has a variable threshold that is determined based on the variation in chunk download time. Then based on the threshold value and the BO, the algorithm selects the appropriate bitrate for the next chunk. This way, playout can response in a smoother way to the instability of network bandwidth. They evaluated their results relying on the widely used (Yin, Jindal, Sekar, & Sinopoli, 2015) QoE utility function which uses different parameters (startup delay, average bitrate, instability and rebuffering duration) to evaluate QoE and reach an improvement between 15% and 55% compared to benchmarks algorithms. Though, similarly to what happens with DASH, this solution has no power on the network. Its only power is to have a more efficient selection on the correct video bitrate to choose.

For example of a solution based upon throughput estimation, we can mention the one develop by (Estive et al., 2014). In this case, a Fair, Efficient, Stable, adaptIVE (FESTIVE) algorithm is developed to improve the QoE. As others HTTP video streaming players, FESTIVE selects the video bitrate for each chunk that are requested independently. The selected bitrate and the scheduling of the requests are based upon the observed throughput for each chunk. Presented results showed that FESTIVE outperformed existing solutions in terms of fairness by 10%, stability by 50% and efficiency by 10% and that it imposed a minimal overhead in a fully functional player.

A good aspect about this research is that it brings notions of Fairness, Efficiency and Stability among several players. On the other hand, since it does not use SDN, it does not have mechanisms to modify and adapt the network to changing conditions. Thus, it will not take full advantage of network resources nor have any mechanism that could protect or enhance the video stream.

Improving the ABR algorithm is a common strategy for many researchers. But there are other perspectives that also do not use the SDN paradigm. (Bedogni et al., 2017) presented a solution that takes advantage of some unused frequencies of the electromagnetic spectrum. The algorithm developed in (Bedogni et al., 2017) is built upon the idea that a significant part of today's connections is coming from indoor, where Wi-Fi already experiences congestion and coverage holes. To relieve this pressure and enhance the communication range, the solution uses the TV White Spaces (TVWS) networks thanks to the sub-GHz frequencies and favorable propagation techniques. The Connection Aware Balancing Algorithm (CABA) presented is evaluated based on the DASH protocol. Although, authors refer that their proposal is general and compatible with others dynamic adaptive video protocols. Their results showed that TVWS brings a more reliable connection although at a slower speed and provided a larger communication range thanks to a better obstacle penetration. Another conclusion was that CABA could improve QoE since it could distribute the segment downloads through the available radio technologies even though it does not have mechanisms to act within its own network that could enhance or protect the video stream.

(Bouten et al., 2012) is based on HAS and aims to a global management of the network. In their research, a rate adaptation algorithm is used by proxy servers, within a multimedia delivery network, to determine the maximum quality each client is allowed to download. This is done considering the available outbound bandwidth at the proxy while optimizing a particular objective. Besides taking advantage of proxies' existing caches, one of the advantages of this research is that the algorithm can be oriented to different objectives. The algorithm can be used to improve the overall QoE of all users, to improve the QoE of one user based on its subscription or to minimize the number of times that a client has access to a rate lower than its subscription. Though, this deployment lacks an entity that could have a complete view of the network, thus improving the efficiency of the solution.

So yet, some solutions have been showed in this chapter regarding different ways to improve the QoE in video streaming. Next, several proposals are discussed, which are more recent proposals that are nearer to the one presented in the current dissertation. The common point among all of these proposals is that they all combine the DASH protocol with the SDN paradigm. Though, as it will be detailed in the next paragraphs, despite being based on a similar principle, there are several differences between them.

(Cetinkaya et al., 2015)'s solution uses OpenFlow to implement the communication between the controller and the switches. Then, the controller can determine all possible paths between the client and the server and it is responsible to decide which path will be used for the client to retrieve the selected video chunks. The path is selected according to the compromise between the capacity to send the segment at that moment and the minimization of the path length. This is done thanks through the send of OFPC_PORT_STATISTICS OpenFlow messages that give information about the load on the links to the controller and through a quite simple architecture. Obtained results showed that the proposed solution provided up to 180 Kbps increase in bitrate and a reduction of outage duration and startup delay even though the improvement is not highly substantial.

In (Taha et al., 2017), authors focus on providing an algorithm that could bring more fairness to all users while avoiding oscillation of the video quality and guaranteeing a stable buffer length. Their algorithm is based on the TCP throughput. The controller collects data about the clients' requests and uses this information to calculate the bandwidth allocated to each client. It also estimates the required throughput for each client thanks to an SDN-traffic monitoring module. Then, after obtaining the necessary information about all users, the controller sets a threshold to determine the maximum bitrate that each client has access to, so that it could guarantee a fair and smooth playback for every one of them. For the evaluation of the QoE of the end-users, authors use Jain's algorithm. Their results showed an improvement of 39% of fairness among users for two competing clients, 43% for three competing clients and 46% for four. Though, their research showed interesting results, they have been obtained using a very specific network type.

Although it is also based on the SDN paradigm, (Cofano et al., 2017)'s solution gives a great focus on the ABR algorithm similarly to some others solutions that have been mentioned earlier in this chapter. Authors used the SDN paradigm to came out with a new concept called Video Control Plane (VCP). VCP has some similarities with the normal Control Plane but is more oriented to video streaming and aim to enforce Video Quality Fairness (VQF). To reach an optimal solution, they have compared two possible approaches in an SDN network: one where they allocate network bandwidth slices to video flows and another where they guide the video players in the bitrate selection. In the ABR part, they compared several algorithms to determine the impact at the client-side. In this case, they compared three algorithms: Conventional, PANDA and Elastic. Conventional is a rate-based algorithm that uses bandwidth estimation. PANDA is also a rate-based algorithm but is a probe-and-adapt algorithm that increments the bitrate to probe the available bandwidth. Elastic is a level-based algorithm that controls the playout buffer length by varying the video bitrate. Their research also only uses one network type in its evaluation experiments.

In (Cofano et al., 2017) three different approaches were considered: The Bandwidth Reservation (BR), the Bitrate Guidance (BG), and a hybrid that combines the two previous ones. In the BR the controller reserves dedicated bandwidth slices to video flows while in the BG the controller computes the optimal video bitrate according to the video quality management policy. The estimated values are then sent to the clients that are responsible for downloading the segments according to the selected bitrate.

To evaluate the results, several QoE-related metrics were used such as Structural SIMilarity index (SSMI) (Zhou Wang & Alan C. Bovik, Fellow, 2004) (which is an objective method that correlates well the quality of an image with the human perception), the switching frequency and the download rate. The results have shown that all the approaches were able to provide a fairer QoE across all video sessions compared to a baseline where no VCP was used.

In (Kleinrouweler et al., 2017), authors have designed and implemented a so-called DASH Assisting Network Element (DANE). In addition to the improvement of the QoE by increasing the bitrate and reducing the numbers of changes among versions and the fair sharing of the available bandwidth. This system also implements a safety mechanism that fully supports encrypted video streams and, thus, is privacy friendly.

The system has a simple four-level structure where top level is the service management plane where the interactions between human managers and the controller happen. The second level is filled by the controller which is responsible for collecting data and transmitting instructions to the forwarding devices. The third level contains the programmable forwarding devices, which unique function is to forward traffic according to the instructions received from the controller. Finally, in the last level are the end-users that use DASH to adapt their stream. Their DASH player uses Buffer Occupancy based Lyapunov Algorithm (BOLA) as ABR and has two mechanisms that improve the streaming performance: target bitrate signaling and dynamic traffic control. The combination of those two mechanisms revealed both an increase of bitrate and stability.

Although, this solution is very interesting, authors referred themselves that their architecture demands a very high computing power to the Service Manager and thus would be very hard to implement in a larger network and it can easily deplete the battery energy of mobile devices.

In (Georgopoulos et al., 2013), the authors propose a fairness framework that also uses OpenFlow to implement the controller. The controller builds a module that inspects the network and receives information from the MPD parser. Upon these data, the controller then generates flow tables and a DASH plugin that ensure that the decisions taken by the module are properly spread and respected across the network. Besides QoE and fairness, their research focus on the heterogeneity of devices and concerns on providing mechanisms that could evaluate the needs of every device in order to take full advantage of the available resources. To do so, SSIM is used as the video quality metric, to describe the relationship between bitrate and QoE so that each user can reach his maximum QoE. To evaluate their results, the authors set up a testbed that recreated a home environment with a regular home router and three device types: a tablet, a smartphone and a High Definition (HD) television. The results are based on a utility function that maps the bitrate of a video at a particular resolution and the QoE perceived by the user. The results showed great improvements regarding stability and QoE fairness since a higher mean QoE is obtained to all users with a very low QoE variance. The main problem with this proposal is that it is based on a utility function that was only designed to fit the test source video characteristics.

(Mu et al., 2016) developed a user-level resource allocation model which maximizes the fairness between user experience on adaptive media streams. Their solution aims to provide the highest possible QoE for each user, by enabling a fair share of the resources while considering the costs regarding the network operation. To do so, their model is based upon three metrics: i) video quality; ii) switching impacts; and iii) cost efficiency. Those three metrics are very relevant since they evaluate not only, the factors that affect QoE on the user side, but also the effects of their mechanisms on the network operation. This leads to a fairer share of the global resources and guarantee that a single video session is not interfering with other video sessions or other applications. Experimental results demonstrate the performance and feasibility of their design for video distribution over future networks. Although their research is based on HAS and SDN, it doesn´t evidence the relevance of each technology in the solution. Furthermore, authors referred that the notion of fairness can be more explored in future work.

Finally, a brief mention to some attempts that have been done to develop objective models that could described and evaluate QoE. Those attempts were realized because it is often hard to evaluate such a subjective and complex concept as QoE is, although it is relevant because they could provide a quick and effective way to evaluate results in an experiment. This way, researchers could use a metric that correlates with the human perception without the need to use a large panel of people.

For example, (Joskowicz & Ardao, 2012) developed a model that could evaluate video quality of any codec, bitrate or display format. For that, it uses an objective estimation of the spatial-temporal activity based on the average Sum of Absolute Differences (SAD). The obtained results have shown that this model fits very well to the perceived video quality, in any combination of codec, bitrate or display format.

For their turn, (Bocchi, De Cicco, & Rossi, 2016) built their own model where authors evaluate the impact of the network QoS on the user QoE. The main conclusion of their work is that the network throughput is lowered by packet losses and Round-Trip Time (RTT) thus increasing the rebuffering frequency. The rebuffering frequency has been identified as the main factor for QoE decreasing. This aspect proves that the temporal structure, instead of spatial artifacts, is also an important factor affecting the QoE.

As seen in this sub-chapter, several solutions have already been presented to enhance QoE through a large variety of strategies. Some of them are also based on the simultaneous use of DASH and SDN. Though, it seems to lack in the literature a solution that evaluates how well each technology can enhance QoE at its contribution. One aspect that is focused in the current work is the proof of that fact. In the current work, several scenarios were designed to identify the individual relevance that each tool can have on the enhancement of the QoE for the users and on the achievement of a more efficient management of the network available resources. In addition, we also aim to study again how the two technologies, i.e., DASH and SDN, operating simultaneously can protect the video streaming with variable rate from concurrent starving-resources traffic types, such as the case of UDP traffic.

3 – System Architecture and System Implementation

This chapter presents a description of the system design, aiming to provide a comprehensive understanding of the tools that were used to implement the system and how they were deployed.

3.1 System Design

As mentioned in the Literature Review, a network with an SDN controller can be divided upon three planes: Management plane, Control plane and Data plane. Figure 20 represents the system design upon the three planes.

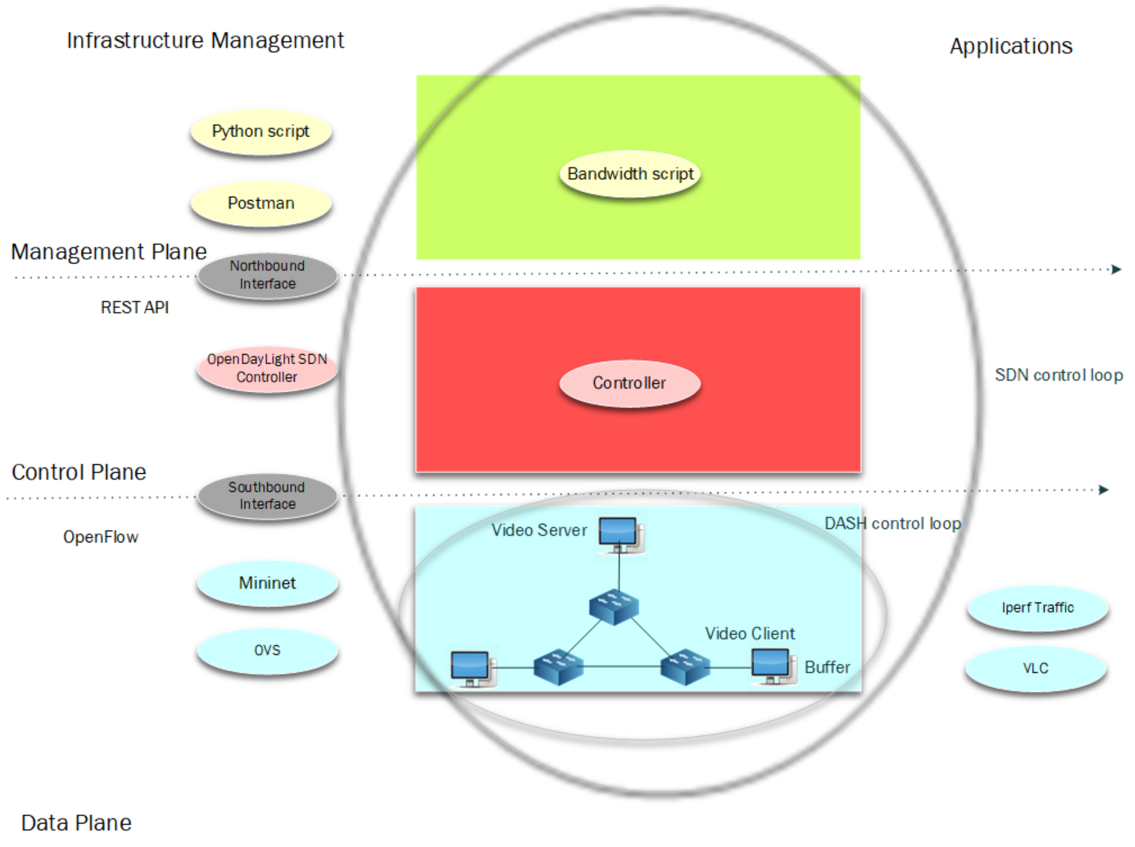


Figure 20: System Design

The management plane is where the managers act and define mechanisms that will be activated by the controller so that the network behavior respects some particular rules. In this design, the behavior of the network is conditioned by the playout of the video. The system needs to guarantee that the video stream has enough resources to be able to play the content even when there is some ‘interference’ from other sources of traffic. To guarantee that, the available network bandwidth is constantly monitored thanks to a custom python script called Bandwidth.py (see Figure 20 and Figure 21). Whenever this script detects that a congestion on the link is imminent, it will activate the QoS mechanisms that aim to protect the video playout. The QoS mechanisms are based on

OpenVSwitch (OVS) queue policies (a more detailed description of this mechanism is done in sub-chapter 3.3). To guarantee that the traffic is correctly filtered, it is necessary to map the queues with the correspondent flows. To do so, Postman ("Postman Introduction," 2018) is used to send HTTP requests that will be sent to the controller through the Northbound interface thanks to the REST API.

The control plane is where all the logic is contained thanks to the controller. In this design the chosen controller was ODL. To emulate the network, the Mininet emulator was adopted and a custom designed python script (that will be described in the next sub-chapter) was developed to build the network topology. The communication between the control plane and the data plane is done through the Southbound interface using the OpenFlow protocol.

Finally, in the data plane the OVS switches will forward the traffic accordingly to the policies sent from the controller. In this case, we can look at the controller as a "messenger" that will deliver the policies set by the management plane to the data plane.

One evidence that comes out of Figure 20 is the existence of two control loops: the DASH control loop and the SDN control loop. The DASH player has its own control loop that allows it to evaluate the available bandwidth at the client side. Based on this evaluation, it can then adjust the version it chooses accordingly to the available resources. Though, this control loop only actuates on the player side and have no power on the network. The existence of the SDN control loop is used to correct that. Since it has a complete view of the network, the SDN control loop has the power to actuate on the whole network. In addition, and since it actuates on the whole network, it will provide more resources to the DASH client. Thus, it also influences the DASH control loop and ultimately leads to a higher end-user QoE.

3.2 QoS Scripts

In the implementation of this system, besides the use of already existing tools, it was necessary to design some custom tools so that the system could have the desired behavior. Those tools were developed through python scripts as they are easy to modify and are suited in situations where large quantities of tests are needed. Using python scripts allows you to quickly change parameters in the design and thus optimizing the system over and over until the system behaves as wanted.

For example, python scripts were used throughout all this research to emulate network topologies thanks to Mininet. In this example, the use of scripts permits to quickly change parameters of the emulated network such as topology, hosts, switches, etc. The script used in the final design of the system is called triangulo.py and builds the topology that will be described in Chapter 4.

Besides triangulo.py, another python script was developed called bandwidth.py. This script is used to monitor the network and to detect congestion on the links. The usage of this script is interesting to understand the role and the importance of the controller in the network. Thus, to monitor the bandwidth on a link, the script periodically queries the controller about the amount of data that has been transferred on this time interval in a form of a GET request through the REST API. The script does this query every five seconds and calculates the correspondent bandwidth. If the bandwidth goes above a certain limit, it will activate the traffic shaping policies and the switches will start to forward traffic accordingly to the policies received from the controller. This way, it is possible to see how central the role of the controller is in the control of the network.

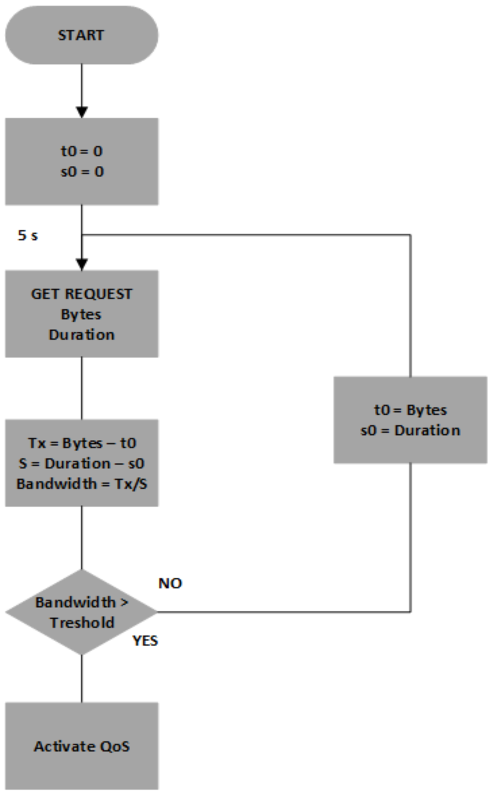


Figure 21: Bandwidth 's Flow Chart

Every time the monitor script detects that a congestion is imminent on the link, i.e., that the used bandwidth goes above the set level, it activates the QoS mechanism. As referred in the previous sub-chapter, the QoS mechanism is based on OVS queues. Thus, the script sets a QoS policy for the link and *n* queues intended for controlling *n* different types of traffic. In the current work, three types of traffic are considered: video traffic for the first client, video traffic for the second client and iperf traffic. Therefore, three queues are created, each one with an assigned bitrate.

In the next sub-chapter a description of OpenVSwitch, which is a software-based switch, is presented. This aims to help understanding how this interaction between the controller and the switch is performed.

3.3 OpenVSwitch

OVS is a multilayer software switch created in connection with the concept of SDN. The goal of this software is to create a production quality platform that could provide management interfaces and take the forwarding functions into a programmatic and logic plane (O. V. Switch, 2014) (see Figure 22).

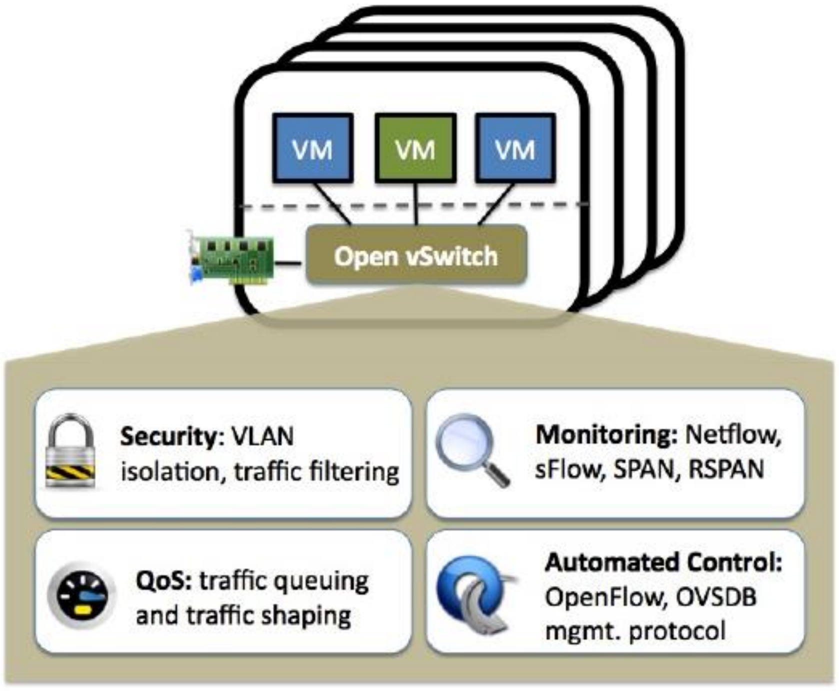

VM
VM
VM
Open vSwitch
Security: VLAN isolation, traffic filtering
Monitoring: Netflow, sFlow, SPAN, RSPAN
QoS: traffic queuing and traffic shaping
Automated Control: OpenFlow, OVSDB mgmt. protocol

Figure 22: OVS Capabilities (O. V. Switch, 2014)

One feature which is available in OVS and which is used in this solution is QoS. In OVS, QoS is available through two mechanisms: 1) traffic policing and traffic shaping. Traffic policing handles ingress traffic, i.e., traffic that enters in the switch. This mechanism is based on a simple form of QoS where packets are dropped when they exceed a stipulated rate. Traffic shaping handles egress traffic, i.e., traffic that comes out of the switch. In this mechanism, queues are configured with a given rate and the switch will queue the egress packets checking that the traffic that is transmitted through a given interface does not exceed the stipulated rate. This mechanism is more elaborated than policing and so more accurate.

In the current project, the used mechanism was traffic shaping applied to a specific switch output interface, since it allows a more flexible and powerful control method. In this case, the first step is setting a QoS policy which is the maximum rate that traffic can be transmitted through this interface and then, setting a variable number of queues where the minimum and maximum rate are specified. This way, it is possible to filter traffic regarding different criteria and make sure that one specified type of traffic stays within a certain rate. Note, obviously, that no queue can get a rate higher than the maximum rate configured initially within the associated QoS policy.

In this case, QoS mechanisms in OVS can be used to schedule the traffic according to its QoS and guarantee that undesired traffic does not exceed a certain rate, thus not overconsuming the network resources.

Though, OVS is only responsible for setting the QoS policy (maximum rate) and setting the queues. In order to achieve the desired behaviour in the network, i.e., in order for the switch to act as wanted, it is necessary to interact to the controller (e.g., through Postman) so it could then transmit the required policies to the switch according to the network activity. Thus, OVS provides the tools to “prepare” the switches but all the “intelligence” and related decisions are managed by the controller.

3.4 Mininet

One fundamental tool in the current project was Mininet. Mininet is a network emulator that is very popular in the “networking” community as it offers the possibility to develop network studies and experiments without the necessity of actual physical hardware. Network elements present in Mininet have similar behaviors as real hardware elements offering the possibility to use them as if they were actual physical hardware and have similar responses to similar events. For example, it is possible to run any program on a Mininet host as if it was running on a real scenario. In the case of our study, if we run a streaming service between two hosts (one acting as a server and the other acting as a client) the responses in terms of delay, packet drops and bandwidth would be the same as if a similar system was built with physical hardware (“Introduction to Mininet,” 2018).

Besides that, Mininet’s horizon of possibilities and versatility is huge as it is possible to design custom topologies in seconds that include pretty much any network device and change those topologies easily. In addition to provide a very large set of network devices, Mininet also offers a large set of parameters for those devices especially regarding performance. For instance, when adding a host, it is possible to set the share of Central Processing Unit (CPU) resources that this host will have access to. When adding a link, it is possible to set several parameters such as bandwidth, delay or the maximum queue size.

Besides the possibility to run custom python scripts, Mininet allows you to run custom files through the CLI. This can be very useful to combine Mininet with other applications (e.g., sFlow-RT which is described in sub-chapter 2.7.2). Moreover, Mininet is pretty user-friendly as it provides the capacity to design custom topologies based on a few python lines. Furthermore, it also provides a Graphical User Interface (GUI) called miniedit where it is possible to design the desired topology by “dragging” the devices into the desktop. After setting the desired topology, miniedit will automatically transpose the topology into python code so the user can save this topology and run it on Mininet later.

One of Mininet’s strongest quality and useful feature (especially in the current study) comes from the fact of using SDN. Based on the use of OpenFlow and related tools, it is possible to program switches in Mininet so that their actions regarding incoming packets would be as desired. OpenFlow makes emulators like Mininet much more useful, since network system designs, including custom packet forwarding with OpenFlow, can easily be transferred to hardware OpenFlow switches for line-rate operation (“Introduction to Mininet,” 2018).

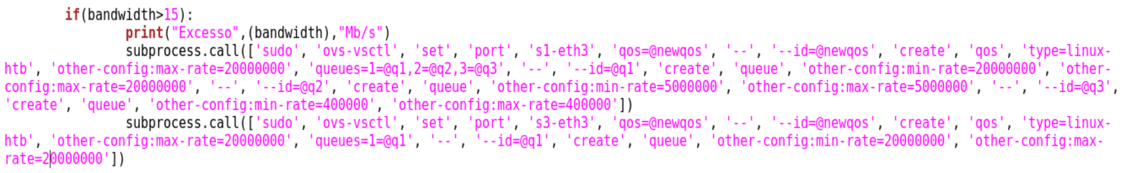
If not specified, Mininet will always set a default controller to use along the topology. Though it is possible to combine the Mininet topology with any SDN controller. For instance, in this case ODL was used as a remote controller.

As we’ve seen the Mininet’s python API really facilitates its use. Furthermore, it makes possible to use standard python libraries in the Mininet execution. For example, in this study, it was necessary to run a server-client approach between hosts within Mininet. This was easily set up by running the SimpleHTTP server class in one of the hosts (simply by opening a terminal from the host). Then, it could be easily accessed by another host also by opening a terminal. The host could access the server through the desired application (HTTP GET, web browser, video player, etc.). This exemplifies how easy it becomes to set diverse testing scenarios thanks to Mininet and thanks to its compliance with other existing tools, such as existing libraries, SDN controllers, etc.

3.5 QoS Strategies

The desired goal of the current implementation was to design a system based on the two technologies (SDN and DASH) that could protect a video stream inside a network even when facing a high competition between applications for network resources. The flexibility of the SDN controller brings an almost infinite set of possible strategies to be combined with the DASH protocol and achieve that purpose. The strategy chosen in this implementation was based on the traffic shaping mechanism available through OVS and presented in sub-chapter 3.3. All network traffic was analyzed and filtered by application to guarantee that, in cases where there is a congestion in the link, the video stream would be protected over other applications traffic to guarantee a certain quality of playout to the client. This mechanism is only possible thanks to the controller since it is responsible for overviewing the traffic in the network and for setting and transmitting the forwarding rules that will be then used by the OVS switches. Besides that, the controller is also responsible (through the bandwidth.py script) for monitoring the links to check if a bottleneck is present or not and to check if it is necessary to activate the QoS mechanisms that will act to protect the video.

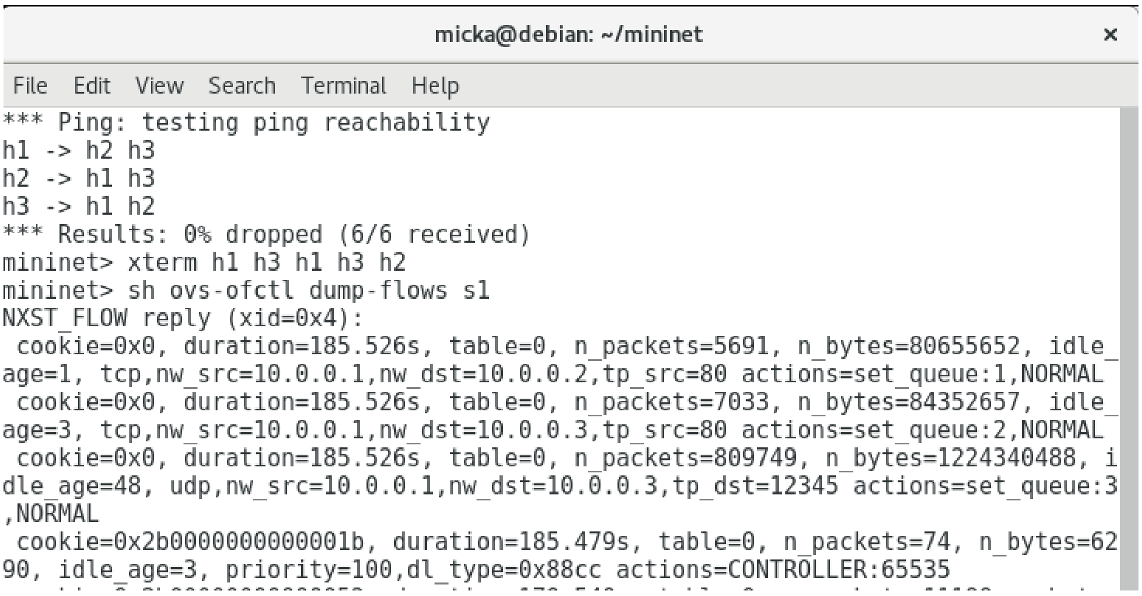
Every time the bandwidth.py script detects that the used bandwidth goes above the limit, it will activate the queues mechanisms. In Figure 23, it is possible to see the commands that create the queues (‘create’ ‘queues=1=@q1, 2=@q2, 3=@q3’) along with their respective rates (e.g., for q1, ‘other-config:min-rate=20000000’’other-config;max rate=2000000’)



```
        if(bandwidth>15):
                print("Excesso",(bandwidth),"Mb/s")
                subprocess.call(['sudo', 'ovs-vsctl', 'set', 'port', 's1-eth3', 'qos=@newqos', '--', '--id=@newqos', 'create', 'qos', 'type=linux-
htb', 'other-config:max-rate=20000000', 'queues=1=@q1,2=@q2,3=@q3', '--', '--id=@q1', 'create', 'queue', 'other-config:min-rate=20000000', 'other-
config:max-rate=20000000', '--', '--id=@q2', 'create', 'queue', 'other-config:min-rate=5000000', 'other-config:max-rate=5000000', '--', '--id=@q3',
'create', 'queue', 'other-config:min-rate=400000', 'other-config:max-rate=400000'])
                subprocess.call(['sudo', 'ovs-vsctl', 'set', 'port', 's3-eth3', 'qos=@newqos', '--', '--id=@newqos', 'create', 'qos', 'type=linux-
htb', 'other-config:max-rate=20000000', 'queues=1=@q1', '--', '--id=@q1', 'create', 'queue', 'other-config:min-rate=20000000', 'other-config:max-
rate=20000000'])
```

Figure 23: Queues Mechanism in bandwidth.py Script

Figure 24 shows the QoS-based rules that the controller has transmitted to the switch. The traffic is filtered through users and through ports (in other words, application). After filtering the traffic, the switch will set the traffic into queues that will have access to a specific rate. It is possible to see that depending on what port the packet is sent and depending on the IP address, the switch will take different actions. For example, if h3 is accessing the video server (nw_dst=10.0.0.3 & tp_src=80), the switch will set the traffic into queue 2 (actions=set_queue:2).



```
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet> xterm h1 h3 h1 h3 h2
mininet> sh ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=185.526s, table=0, n_packets=5691, n_bytes=80655652, idle_
age=1, tcp,nw_src=10.0.0.1,nw_dst=10.0.0.2,tp_src=80 actions=set_queue:1,NORMAL
 cookie=0x0, duration=185.526s, table=0, n_packets=7033, n_bytes=84352657, idle_
age=3, tcp,nw_src=10.0.0.1,nw_dst=10.0.0.3,tp_src=80 actions=set_queue:2,NORMAL
 cookie=0x0, duration=185.526s, table=0, n_packets=809749, n_bytes=1224340488, i
dle_age=48, udp,nw_src=10.0.0.1,nw_dst=10.0.0.3,tp_dst=12345 actions=set_queue:3
,NORMAL
 cookie=0x2b0000000000001b, duration=185.479s, table=0, n_packets=74, n_bytes=62
90, idle_age=3, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
```

Figure 24: QoS-Based Rules on OVS Switch

The different queues are intended to fit different traffic needs. In Table 2 and Table 3, a summary of the created queues for S1 and S3 are presented.

Table 2: Configured Queues for S1

Queue	Purpose	Rate
Q1	Video for h2	20 Mbits/s
Q2	Video for h3	5 Mbits/s
Q3	Iperf	400 kbits/s

Table 3: Configured Queues for S3

Queue	Purpose	Rate
Q1	Video for h2	20 Mbits/s

4 – System Evaluation

In this chapter, a description of the System Evaluation is done. First a description of the strategies and the tools that were used to set the testing environment is done. Then, a more detailed description of the defined scenarios is performed. Finally, the obtained results are presented and discussed on whether or not they meet the goals of the current dissertation.

4.1 Testing Environment

The whole testing environment was developed on a Debian Virtual Machine (VM) hosted on VirtualBox.

The testing environment includes a network topology built with Mininet through the triangulo.py script associated with an SDN remote controller implemented with ODL. The topology is built in triangle where three switches are all connected with each other and where each switch is connected to one host, leading to a total of three switches and three hosts (see Figure 25). In order to get a better control of the network and to have a more accurate study of the system, a limitation has been set on the links. This limitation has been set through OVS's QoS mechanism where a certain rate is stipulated, forcing the egress traffic on the switches not to reach higher rates than the stipulated QoS policy. In this case, the links are limited to 20 Mbits/s.

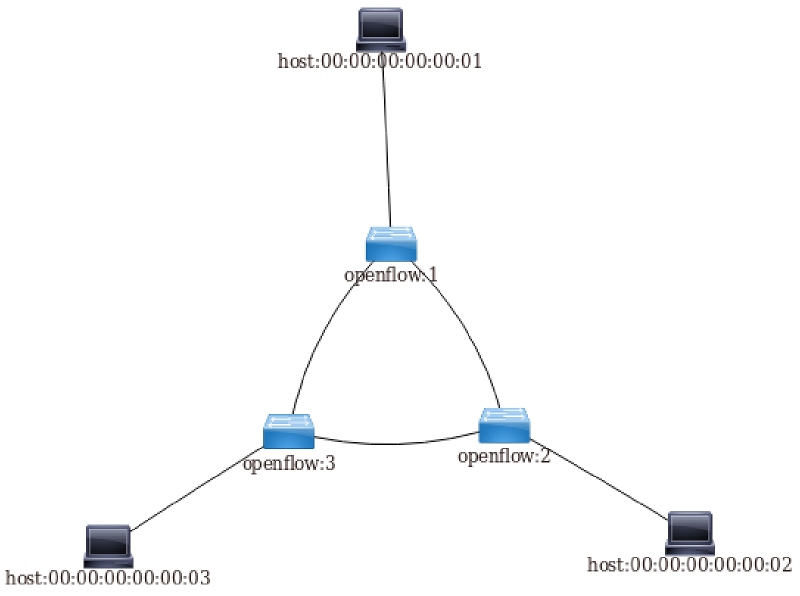


Figure 25: Topology Built With triangulo.py Script

As said in the previous paragraph, the topology is built in triangle. The topology was built like that so that all switches could communicate directly with each other, though that brings some issues. Within a network, if there is more than one L2 path between the same two nodes, that creates a switching loop. This will lead to instability in the forwarding tables since switches are “flooded” with repeated equal MAC addresses from different ports. Basically, what happens is that switches receive frames from different sources and will start to broadcast them, this phenomenon will repeat itself several times in a short period of time leading to a situation where switches will start to broadcast each other broadcast. This way, all available bandwidth will be used and the switch processors won´t be able to handle the load.

The implementation of the remote controller associated to this topology permits to resolve the switching loop problem and, therefore, to guarantee a complete connectivity between every device in the topology. The mechanism that prevents forwarding loops and ensures communication between all switches is the Link Layer Discovery Protocol (LLDP) (Ochoa Aday, Cervelló Pastor, & Fernández Fernández, 2015). LLDP is an L2 open-standard protocol that, as it is stated in its name, enables the discovery of the link between two hosts within a network. The protocol is quite similar to Cisco Discovery Protocol (CDP) (Ochoa Aday et al., 2015), although it could be applied to any brand device since it is vendor-neutral in opposite to CDP. What LLDP allows is for devices to introduce themselves to the network, i.e., giving the network information about themselves regarding their identity, capacity and “neighbors”.

To get a better understanding of LLDP’s functioning, let’s look at the example of our topology. The example corresponds to OpenFlow v 1.3.

As mentioned before, the topology includes three switches connected in triangle and each switch communicates to the controller. Note that the topology in Figure 26 does not include any interface 1 since that in every switch the interface 1 is connected to the corresponding host, this way the figure is more accurate to the existing topology. After setting up, the controller orders each switch to flood LLDP packets through all its egress ports. For example, S1 will forward LLDP packets through port 2 and port 3. S3 will receive switch 1 LLDP packets through its port 2 and forward them to the controller. When the controller will receive S1 information through S3 it will conclude that those two switches are connected through that interface.

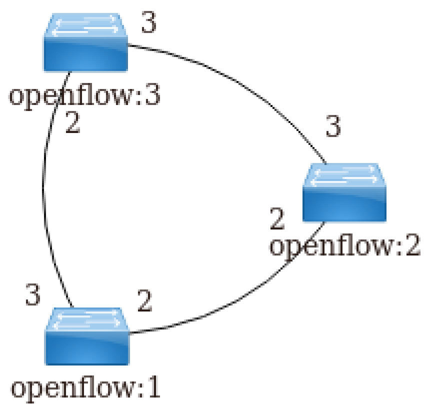


Figure 26: Network Topology

This process is constant as it is necessary to keep refreshing the devices information to guarantee that the topology remains equal as its initial state so that the controller maintains an accurate control of the network. The controller repeatedly orders the switches to flood packets to keep updated information about the network topology. Otherwise, if the controller does not receive any more information about one particular device or one particular link, it will assume that it is because it no longer exists.

Next a description of each step of the LLDP is made, from the handshake between one switch and the controller until another switch forwards to the controller an LLDP packet received from the first switch.

The handshake is the process where two entities present them to each other (in this case a switch and the controller). The handshake begins with S1 sending a HELLO packet to the controller, letting it know that it exists in the network. After receiving the HELLO packet, the controller sends a request querying the switch’s features which are then replied by the switch (see Figure 27).

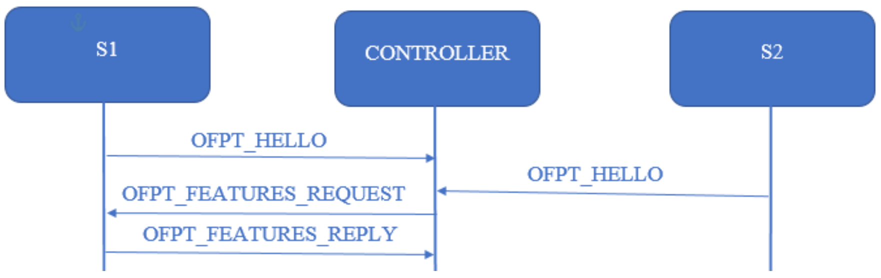


Figure 27: LLDP's Handshake: Step 1

In the second step, the controller sends a BARRIER_REQUEST to the switch. Those types of messages are used by the controller to set a synchronization point, i.e., to ensure that all previous state messages were processed before the BARRIER_RESPONSE is sent back to the controller (see Figure 28).

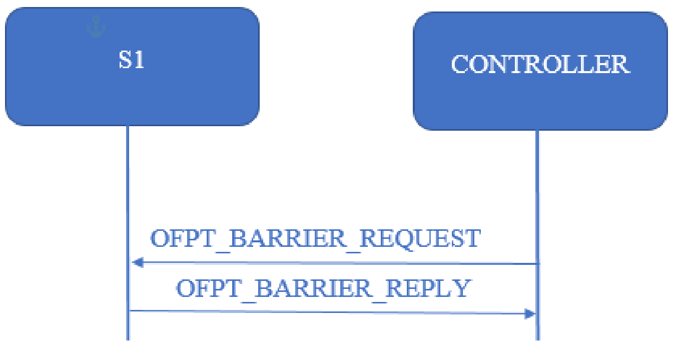


Figure 28: LLDP's Handshake: Step 2

The MULTIPART messages are messages used by the controller to maintain state from the datapath. In this step, the MULTIPART message is an OFMP_DESC. Those type of messages are used to get information regarding the switch's manufacturer, hardware revision, software revision, serial number and a description field (see Figure 29).

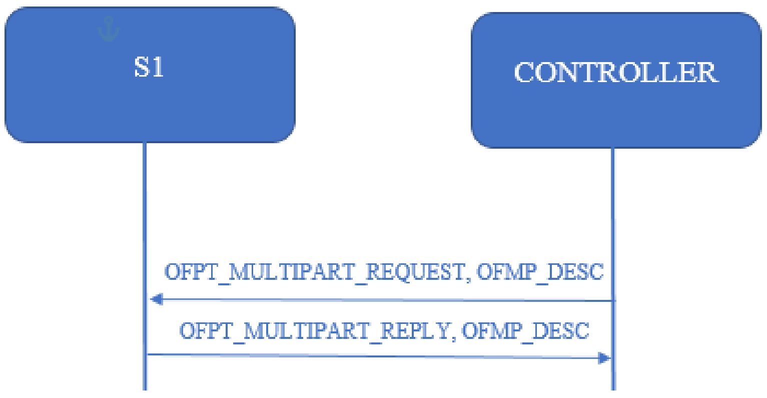

S1
CONTROLLER
OFPT_MULTIPART_REQUEST, OFMP_DESC
OFPT_MULTIPART_REPLY, OFMP_DESC

Figure 29: LLDP's Handshake: Step 3

The PORT_DESC are used by the controller to get information regarding all switch's ports that support OpenFlow (see Figure 30).

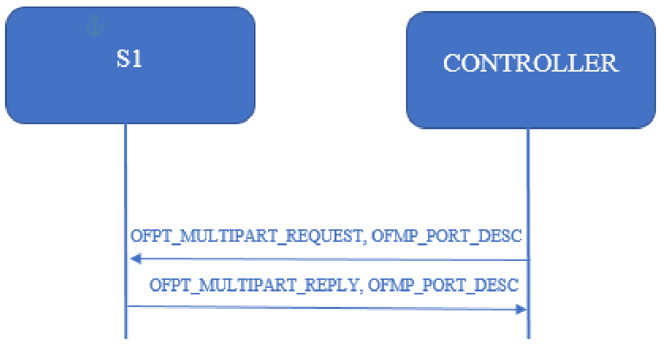

S1
CONTROLLER
OFPT_MULTIPART_REQUEST, OFMP_PORT_DESC
OFPT_MULTIPART_REPLY, OFMP_PORT_DESC

Figure 30: LLDP's Handshake: Step 4

The OFPMP_METER_FEATURES statistics request messages provide the set of features of the metering subsystem (see Figure 31).

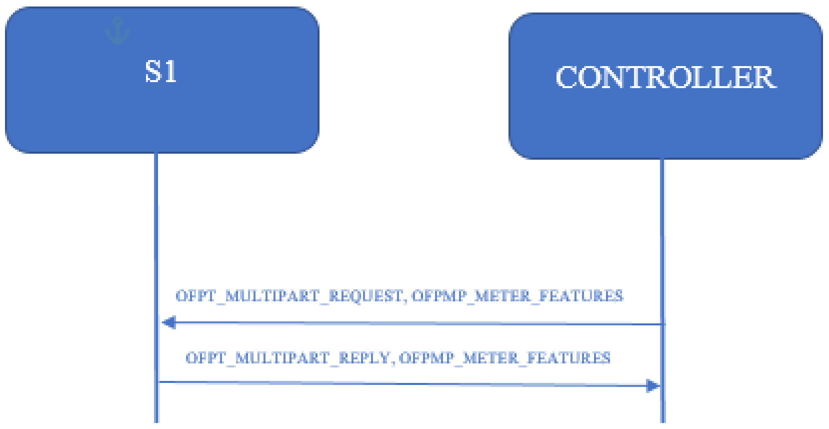

S1
CONTROLLER
OFPT_MULTIPART_REQUEST, OFPMP_METER_FEATURES
OFPT_MULTIPART_REPLY, OFPMP_METER_FEATURES

Figure 31: LLDP's Handshake: Step 5

In step 6, the controller retrieves information regarding the capabilities of groups on a switch (see Figure 32).

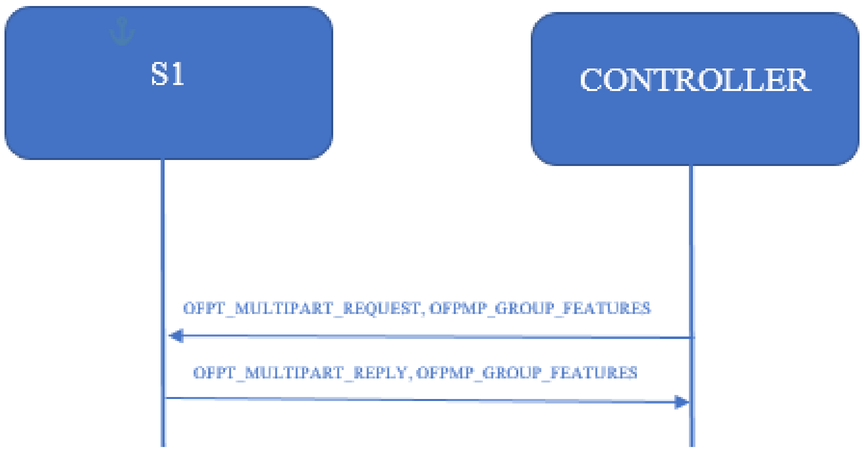

S1
CONTROLLER
OFPT_MULTIPART_REQUEST, OFPMP_GROUP_FEATURES
OFPT_MULTIPART_REPLY, OFPMP_GROUP_FEATURES

Figure 32: LLDP's Handshake: Step 6

The ROLE messages are messages that the controller sends to the switch to set or update its role whether at master or slave. The ROLE value is used in networks containing several controllers. Each network can only have one master controller (see Figure 33).

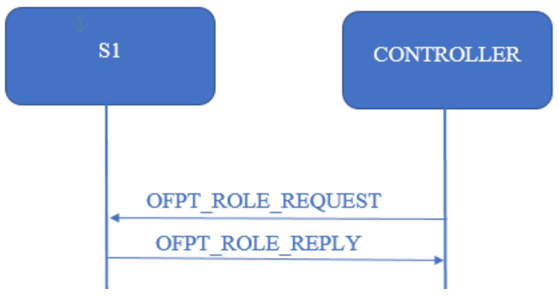


Figure 33: LLDP's Handshake: Step 7

The OFPMP_TABLE messages are a MULTIPART message type used to get information about the switch’s tables (see Figure 34).

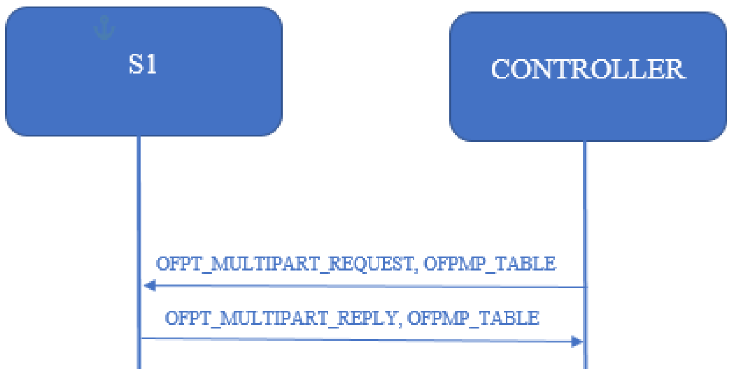

S1
CONTROLLER
OFPT_MULTIPART_REQUEST, OFPMP_TABLE
OFPT_MULTIPART_REPLY, OFPMP_TABLE

Figure 34: LLDP's Handshake: Step 8

The OFPMP_GROUP_DESC multipart request message provides a way to list the set of groups on a switch, along with their corresponding bucket actions (see Figure 35).

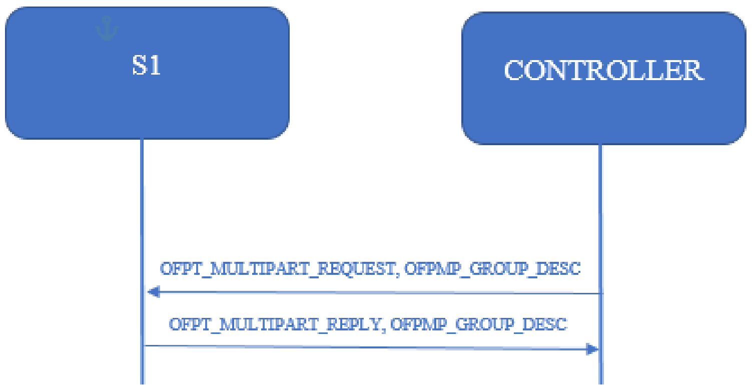

S1
CONTROLLER
OFPT_MULTIPART_REQUEST, OFPMP_GROUP_DESC
OFPT_MULTIPART_REPLY, OFPMP_GROUP_DESC

Figure 35: LLDP's Handshake: Step 9

The types of messages in Figure 36 are used to get statistics for one or more groups.

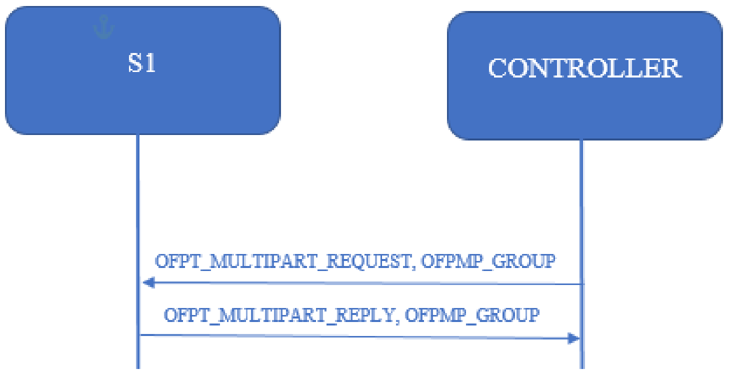

S1
CONTROLLER
OFPT_MULTIPART_REQUEST, OFPMP_GROUP
OFPT_MULTIPART_REPLY, OFPMP_GROUP

Figure 36: LLDP's Handshake: Step 10

The type of messages in Figure 37 are used to get information about individual flow entries.

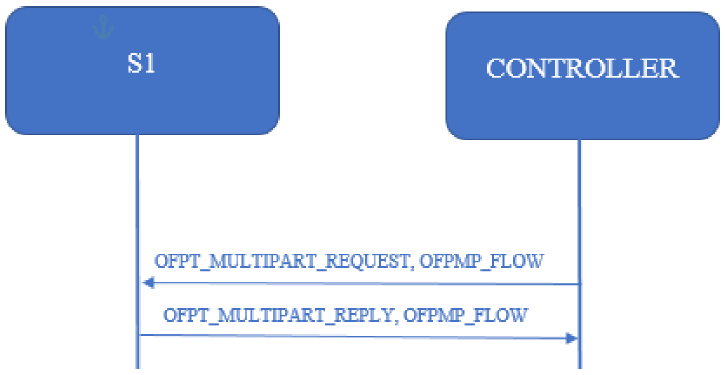

S1
CONTROLLER
OFPT_MULTIPART_REQUEST, OFPMP_FLOW
OFPT_MULTIPART_REPLY, OFPMP_FLOW

Figure 37: LLDP's Handshake: Step 11

The type of messages in Figure 38 are used to get information about port statistics.

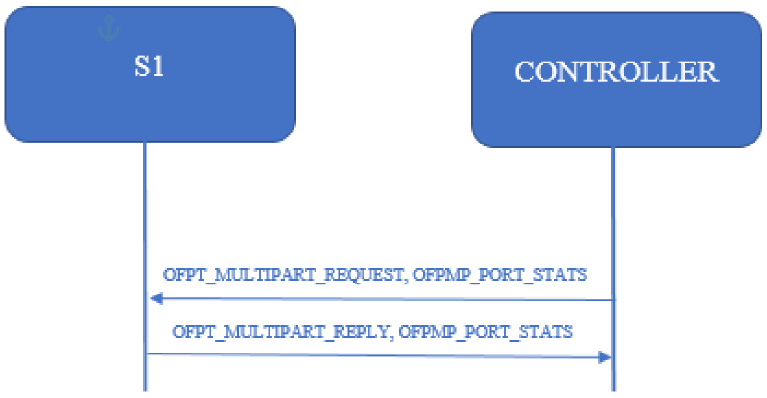

S1
CONTROLLER
OFPT_MULTIPART_REQUEST, OFPMP_PORT_STATS
OFPT_MULTIPART_REPLY, OFPMP_PORT_STATS

Figure 38: LLDP's Handshake: Step 12

The OFPMP_QUEUE multipart request message provides queue statistics for one or more ports and one or more queues. With this step, the handshake between the switch and the controller is over and the controller has all information it needs regarding S1 (see Figure 39).

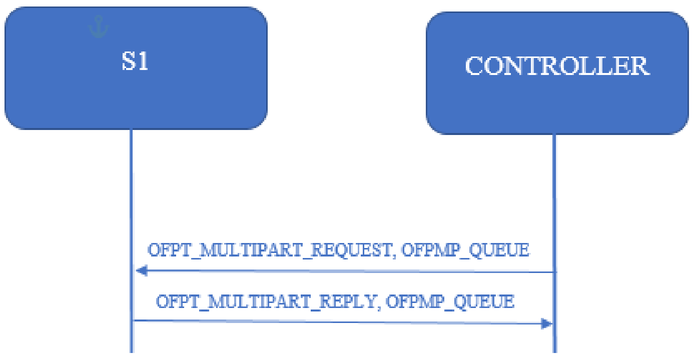

S1
CONTROLLER
OFPT_MULTIPART_REQUEST, OFPMP_QUEUE
OFPT_MULTIPART_REPLY, OFPMP_QUEUE

Figure 39: LLDP's Handshake: Step 13

After gathering all necessary information about the switches, the controller needs to discover the links that unify them. To do so, the controller sends a PACKET_OUT to all known switches in the network ordering them to send LLDP packets through all their egress ports. Thus, the switches will start to send LLDP packets that will reach the correspondent switch on the other side of the link. Every time that a switch receives an LLDP packet coming from another switch, it will execute an action rule, initially downloaded to that switch from the SDN controller. That action rule is about to forward the received LLDP packet to the controller via a PACKET_IN. When the controller receives S1's LLDP packet encapsulated in the PACKET_IN sent from S2, it will conclude that those two switches are connected to each other on whatever ports are indicated in the packet. This process will repeat itself for all links periodically, allowing the controller to keep track of the network's links (see Figure 40).

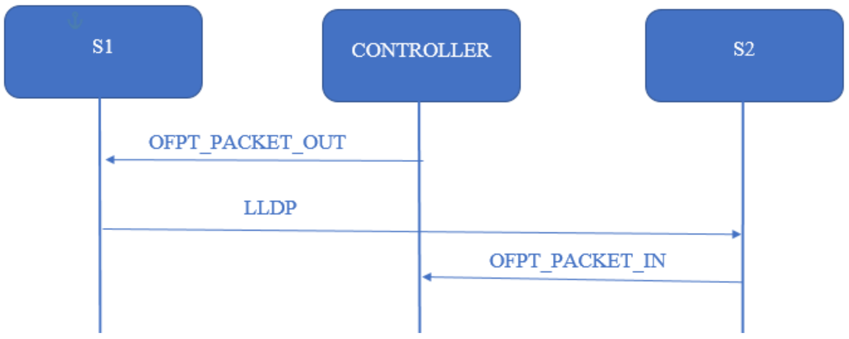


Figure 40: LLDP Packet Forwarding

After setting the topology and guarantee that every device could reach any other device in the topology, different roles were attributed to each host. In the scenario defined in the current dissertation, Host 1 was defined as the video server where the content would be stored and the other two hosts were defined as clients that would access the content hosted on h1. Besides the video server, an iperf server has been set on h3 so that h1 could inject traffic to it, thus creating competition on the link between S1 and S3 since the video is delivered through this link (see Figure 41). Iperf is a tool used to measure the maximum achievable bandwidth on IP networks. It supports several protocols included UDP and can be used to create UDP streams of specified bandwidth, such as it is in the current dissertation ("iPerf - The TCP, UDP and SCTP network bandwidth measurement tool," 2018).

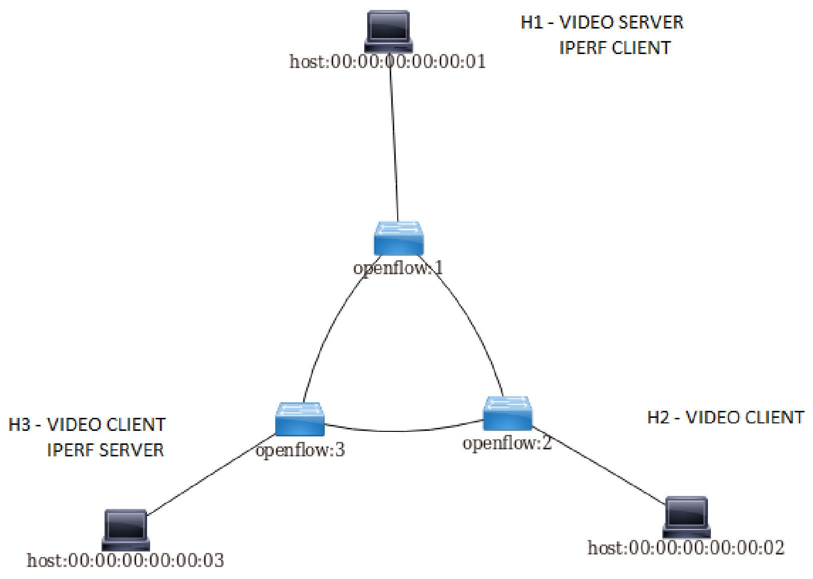


Figure 41: Role of Each Host in the Topology

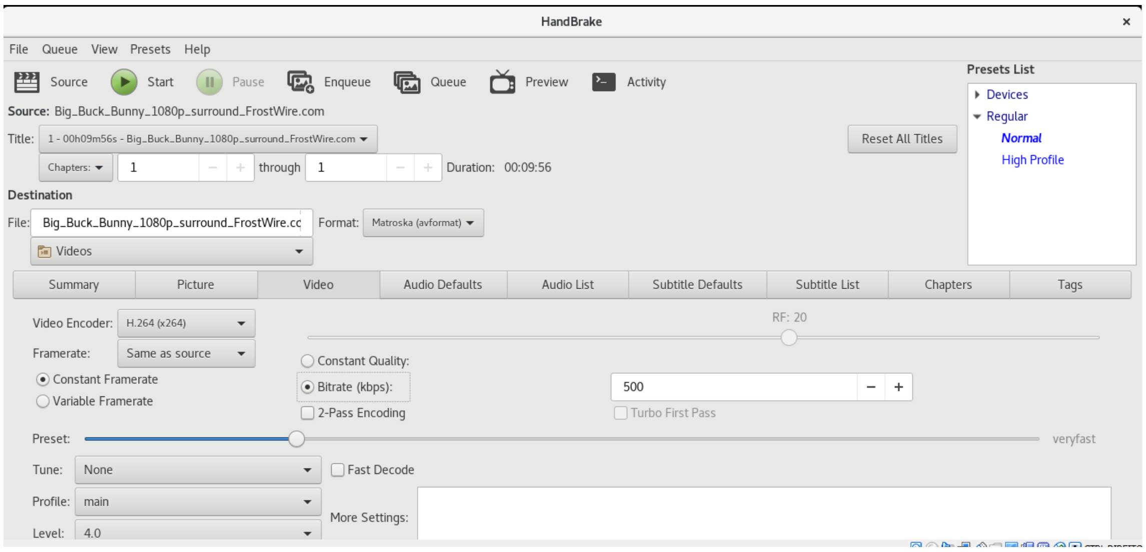
The video server was implemented with the python HTTP simpleServer which is included by default in Mininet. To activate the server, we simply have to open a terminal in h1, go to the desire path through the CLI and then activate the server in the port 80 thanks to the command “python3 -m http.server 80&”. Then, the clients could access the server simply by opening a terminal in Mininet and accessing it through the desired application. In this case, the VideoLan Client (VLC) player (“VideoLAN - Documentation,” 2018) was used by each client to reproduce the video.

4.2 Design of the DASH Content

The video chosen to design this experiment was “Big Buck Bunny” (see Figure 42) which is a short-animated picture developed under free software and largely used by the scientific and academic community in research papers (Georgopoulos et al., 2013).

Figure 42: Big Buck Bunny Movie

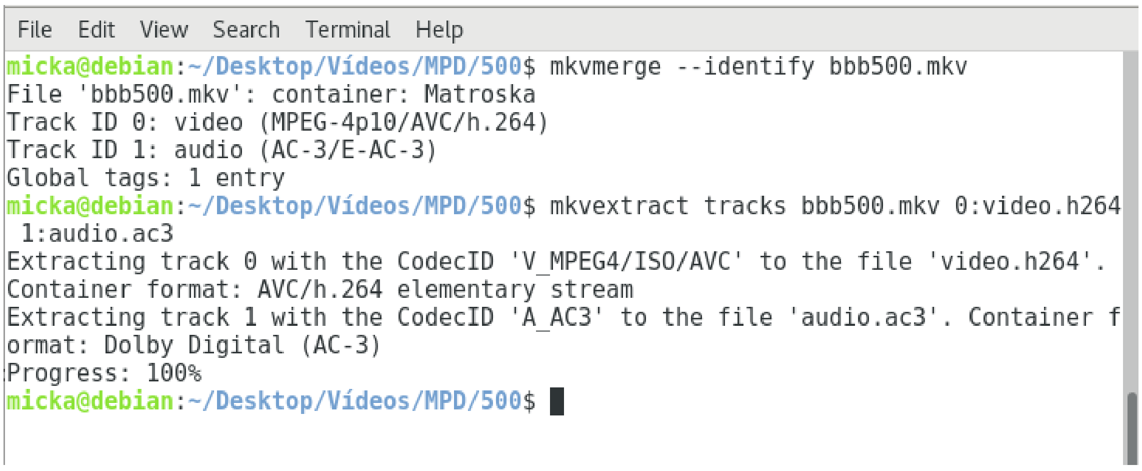
In order to fulfill the DASH requirements, the video needed to be transposed into an MPD file. To do so, several tools were used to transform the original Audio Video Interleave (AVI) file into the MPD file. The first of these tools was the *Handbrake* software that was used to convert the AVI file into an MatrosKa Video (MKV) one as this would make the extraction of the video and the audio tracks easier. Besides the conversion into an MKV file, *Handbrake* was also used to encode the original 1080p file into a series of files coded with different bitrates (see Figure 43). This was done so that the MPD file could present the same content under different video qualities. All video shared the same coding parameters excepting the bitrate. Files were encoded using the following bitrates: 500 kbps, 1000 kbps, 2000 kbps and 4000 kbps.



Source: Big_Buck_Bunny_1080p_surround_FrostWire.com
Title: 1 - 00h09m56s - Big_Buck_Bunny_1080p_surround_FrostWire.com
Chapters: 1 through 1 Duration: 00:09:56
Destination
File: Big_Buck_Bunny_1080p_surround_FrostWire.cc Format: Matroska (avformat)
Videos
Video Encoder: H.264 (x264)
Framerate: Same as source
(●) Constant Framerate
( ) Variable Framerate
RF: 20
( ) Constant Quality:
(●) Bitrate (kbps): 500
[ ] 2-Pass Encoding
[ ] Turbo First Pass
Preset: veryfast
Tune: None
[ ] Fast Decode
Profile: main
More Settings:
Level: 4.0

Figure 43: Video Conversion and Coding of the 500 kbps Version

The extraction of the video and audio tracks was done using a software called *MKVToolNix* and originated audio files in the Dolby Audio Codec 3 (AC3) format and video files in the H.264 format (see Figure 44).



```
micka@debian:~/Desktop/Vídeos/MPD/500$ mkvmerge --identify bbb500.mkv
File 'bbb500.mkv': container: Matroska
Track ID 0: video (MPEG-4p10/AVC/h.264)
Track ID 1: audio (AC-3/E-AC-3)
Global tags: 1 entry
micka@debian:~/Desktop/Vídeos/MPD/500$ mkvextract tracks bbb500.mkv 0:video.h264
 1:audio.ac3
Extracting track 0 with the CodecID 'V_MPEG4/ISO/AVC' to the file 'video.h264'.
Container format: AVC/h.264 elementary stream
Extracting track 1 with the CodecID 'A_AC3' to the file 'audio.ac3'. Container f
ormat: Dolby Digital (AC-3)
Progress: 100%
micka@debian:~/Desktop/Vídeos/MPD/500$
```

Figure 44: Extraction of the Tracks Through MKVToolNix

After gathering different versions of the same content in the required formats, we had all necessary “ingredients” to build the MPD file. This process was done using *MP4Box* which is a multimedia packager developed under the Group Project on Advanced Content (GPAC). This tool was firstly used to join the video and audio files into an MP4 file and then to segment each version to build the manifest. The segmentation was done so that the DASH protocol could easily retrieve chunks from different versions of the same content.

To reproduce the content, the chosen media player was VLC. VLC is currently an extremely popular player thanks mostly to its acceptance regarding codecs and file formats. Though many people do not know VLC’s power regarding other applications including streaming. VLC offers a large set of parameters for streaming including several network protocols such as HTTP, RTP, RTSP, UDP, etc. in addition to support several network streaming formats like Apple’s HLS, or MPEG-DASH. Though reproducing of DASH content have been a possibility since a few years thanks to the VLC DASH plugin, VLC’s compatibility with DASH was just officially released since the Vetinari version (3.0.0) in 2018.

4.3 Description of the Scenarios

In the current evaluation, two types of traffic were used: video streaming and iperf to simulate common traffic. In the initial state, no QoS mechanisms are implemented, thus providing the applications the freedom to use the desired bandwidth. Though, if the controller detects that a link is getting congested, it will activate the QoS mechanisms. Then, the video stream will be privileged, having access to a rate similar to the rate available on the link (20 Mb/s) when the iperf traffic will be limited to a lower rate (400 kb/s).

As mentioned before, the flexibility brought by SDN can provide a very high number of possible designs and implementations. For example, in this case we imagine a scenario where the two video clients would have access to a different rate. In this case, h2 would be a “premium” client that could have access to the highest possible rate when h3 would be a “regular” client that would access to a rate that could guarantee him a sufficient rate to a satisfactory playout (5 Mb/s), though it would not have access to the maximum available rate like h2. This discrimination could, for example, be used in a commercial way to distinguish services or to distinguish devices types so that devices could have access to a sufficient rate that could fit their needs. This, once again, proves how flexible can SDN be and how it could really facilitate the work of network managers.

Once the QoS mechanism is activated, traffic will be mapped into queues that will guarantee that one specific traffic would not consume more bandwidth what it is supposed to.

To ascertain the relevance of the DASH protocol and the SDN paradigm, several scenarios were designed in order to compare results between scenarios and prove that the user's QoE could be enhanced if both technologies were used simultaneously. The evaluation environment included four scenarios:

1 – Streaming using neither DASH nor SDN

2 – Streaming using only DASH

3 – Streaming using only SDN

4 – Streaming using both DASH and SDN

Table 4: Summary of the Scenarios

Scenario	DASH	SDN
1	✗	✗
2	✓	✗
3	✗	✓
4	✓	✓

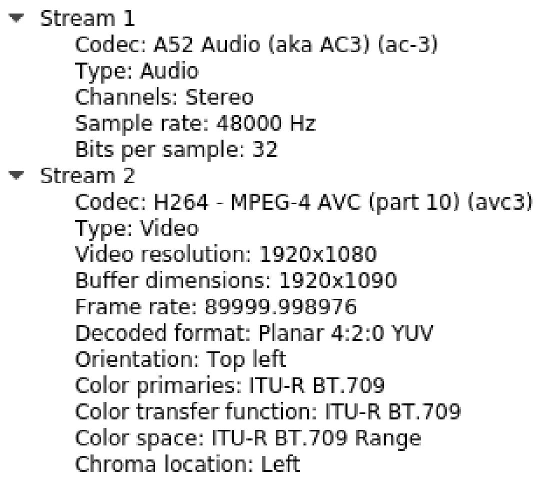
Note that although some of the scenarios are presented as using neither SDN nor DASH (or using both), this requires a further explanation. In every scenario, SDN is used to break free from the switching loop and guarantee the limitation of the links. Though in scenarios presented as not using SDN, there were no implementations of mechanism that could enhance the video streaming. Thus, the controller acts as a "dummy" controller just used to guarantee the initial topology and to bring a testing environment that could be more similar for all scenarios. If all scenarios present a similar initial state, the results could better be compared with each other leading to a fairer analysis and study.

For its turn, the DASH protocol is also used in the scenarios presented as DASH free in a way that an MPD file is used. Though, the MPD file used in these scenarios only has one representation, thus the adaptive mechanisms available in the DASH protocol are not used.

The first intention for the tests was to reproduce the video through both video clients while injecting iperf traffic to create a competition on the principal link (S1-S3). Unfortunately, processing limitations of the VM made that intention impossible. The VM was not able to handle two video sessions running on VLC while running Wireshark. It was then decided to choose another testing strategy that, even though smaller in terms of participants and traffic, was just as relevant for observing and understanding the behavior of the system.

After several runouts, it was observed that the video delivery was always done through the same path. This fact was taking advantage of in the design of the solution as well as in the design of the testing scenarios.

All scenarios were tested upon similar conditions: a video stream was reproduced by h2 with a one second buffer size (see more info in Figure 45), then h1 would inject UDP iperf traffic to h3 bringing competition on the link between S1 and S3 (where the video stream to h2 also goes by (see Figure 46)). As UDP iperf tends to use all available bandwidth within a link, the different scenarios were made to observe how the video stream reacts to the interference of the iperf traffic and how the proposed technologies could mitigate this interference.



```
▼ Stream 1
    Codec: A52 Audio (aka AC3) (ac-3)
    Type: Audio
    Channels: Stereo
    Sample rate: 48000 Hz
    Bits per sample: 32
▼ Stream 2
    Codec: H264 - MPEG-4 AVC (part 10) (avc3)
    Type: Video
    Video resolution: 1920x1080
    Buffer dimensions: 1920x1090
    Frame rate: 89999.998976
    Decoded format: Planar 4:2:0 YUV
    Orientation: Top left
    Color primaries: ITU-R BT.709
    Color transfer function: ITU-R BT.709
    Color space: ITU-R BT.709 Range
    Chroma location: Left
```

Figure 45: Video Stream Metadata

The links have a 20 Mbits/s capacity, this way in order to consume the highest bandwidth possible, the iperf request were done with a 25 Mbits/s parameter. Thus, if no mechanisms were activated to control the traffic, it would consume all 20 Mbits/s available bandwidth.

Table 5: Initial Capacities

	Rate
Links capacity	20 Mbits/s
Iperf Traffic (request)	25 Mbits/s (reaches ≈ 20 Mbits/s)
Video traffic	Depends on available bandwidth

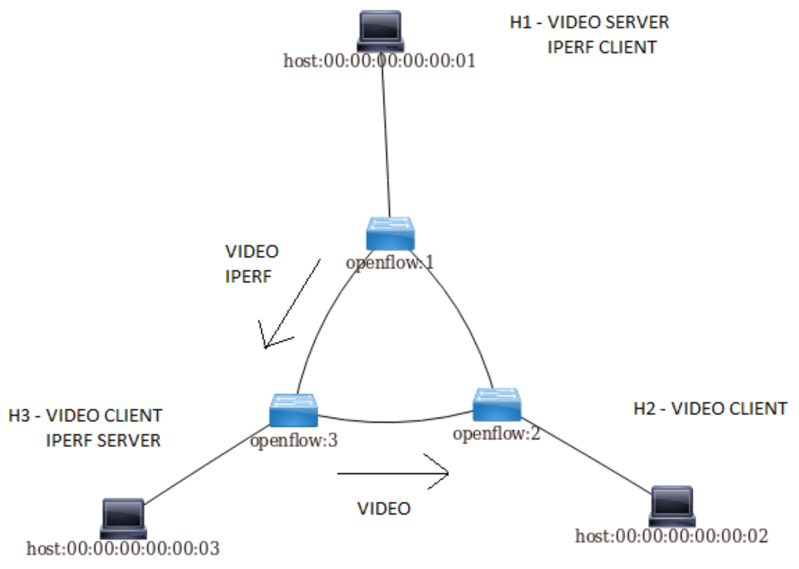


Figure 46: Network Flows

The results were obtained thanks to Wireshark where the reached bandwidth is presented for each traffic throughout time.

4.4 Results

4.4.1 Scenario 1: No DASH and no SDN

This scenario has the simplest implementation as no mechanism of adaption is used. Of course, this scenario expects a bad reaction from the video stream since it has no mechanisms to protects itself to the interference of the iperf traffic. Figure 47 presents both bandwidth for video and iperf for the first scenario.

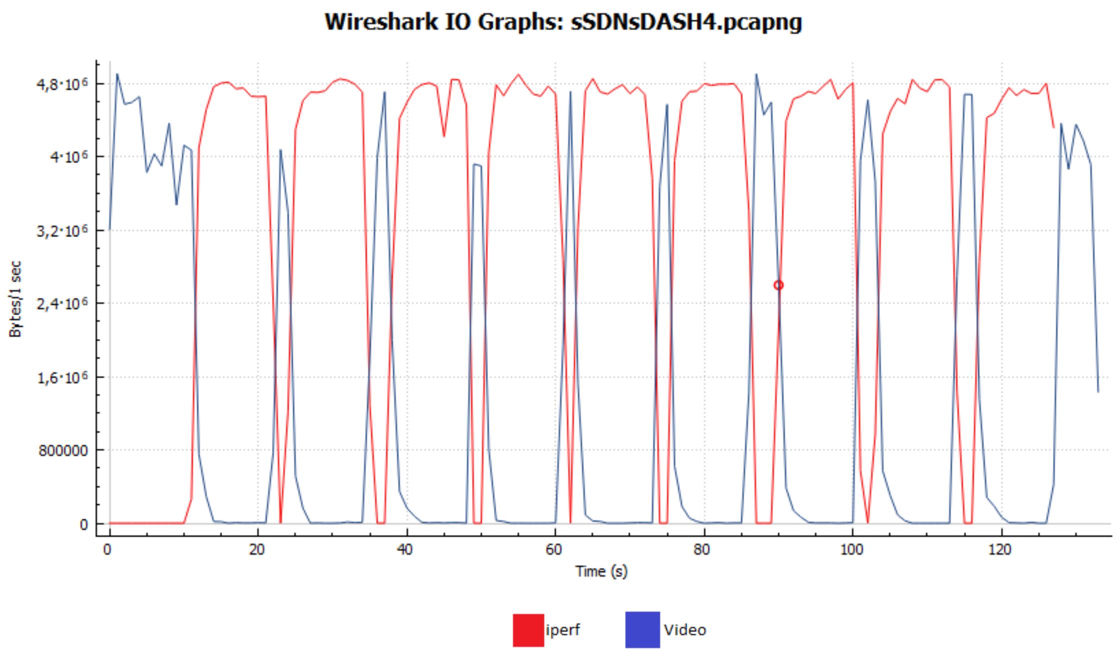


Figure 47: Results for Scenario 1

As expected, the results to the video stream are pretty unsatisfactory. It is possible to see that before the iperf traffic arrives, the video stream shows no problem reaching a high bandwidth that could allow to reproduce the content properly. Though, after the iperf traffic is injected, it starts to consume all available bandwidth and the video stream drops, not being able to retrieve anymore chunks until the iperf traffic stops. A clear evidence of this fact is the way how the video traffic peaks between the intervals where iperf is down (corresponding to the time before another iperf request is done).

Figure 48 represents the Simple Moving Average (SMA) results for scenario 1. This graph was obtained by modification of the previous graph (see Figure 47) by changing the SMA period. It was added a filter of 10 interval SMA. A similar graph is present in the results of each scenario.

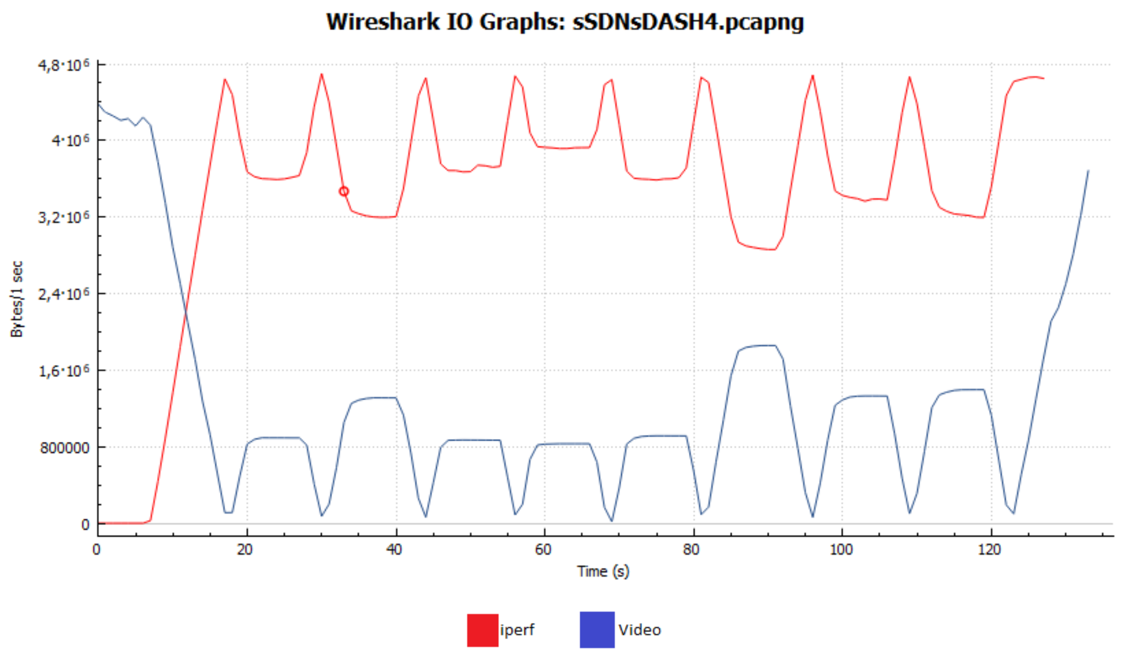


Figure 48: Simple Moving Average Results for Scenario 1

4.4.2 Scenario 2: DASH without SDN

Though this scenario already includes adaptive mechanisms existing in DASH, it is to expect that the results (see Figure 49) will not be that far from the results from scenario 1 because of the "aggressivity" of iperf. DASH brings adaptive mechanisms that allows to adapt the playout and choose a representation that will meet the available bandwidth. Though, these mechanisms will not have any effect if almost all available bandwidth is consumed by another application as it is expected to happen with iperf. In other words, if the video stream will not have access to a minimum bandwidth, it will not be able to reproduce the content even in the lowest version.

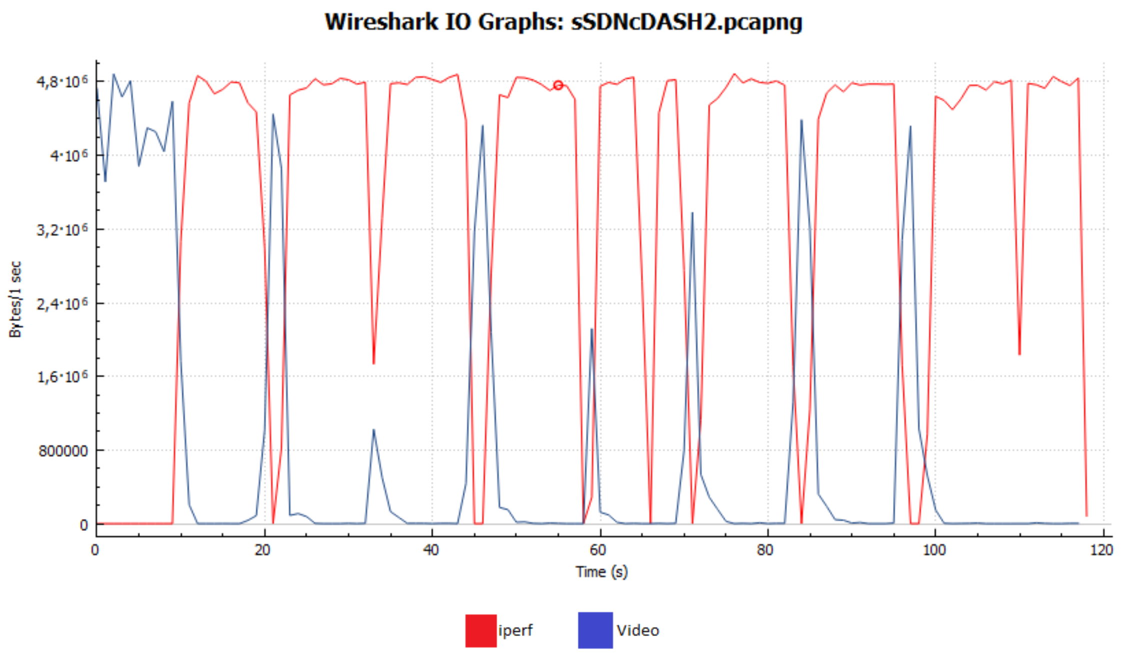


Figure 49: Results for Scenario 2

As expected, results are not much better than in the first scenario. Like in the previous case, it is possible to see that the video stream cannot go on when the iperf appears. Even though it is possible to see some attempts to lower the version, those are not well succeeded as traffic drops out again when iperf returns. Figure 50 represents the SMA results for scenario 2.

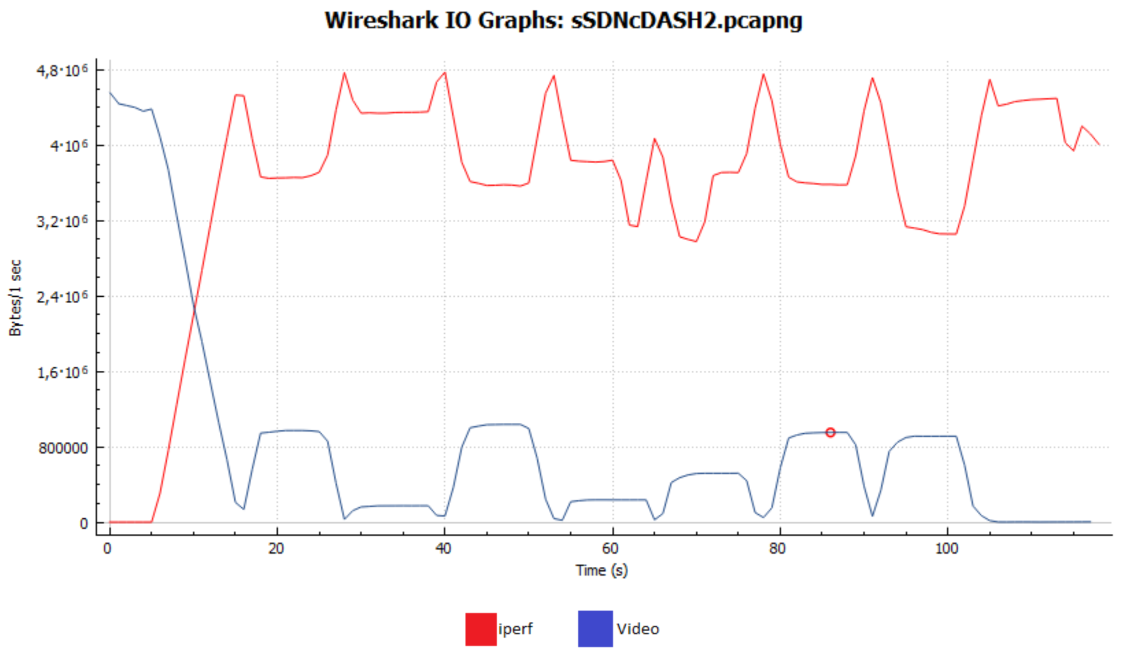


Figure 50: Simple Moving Average Results for Scenario 2

Figure 51 represents the downloaded chunks for each representation.

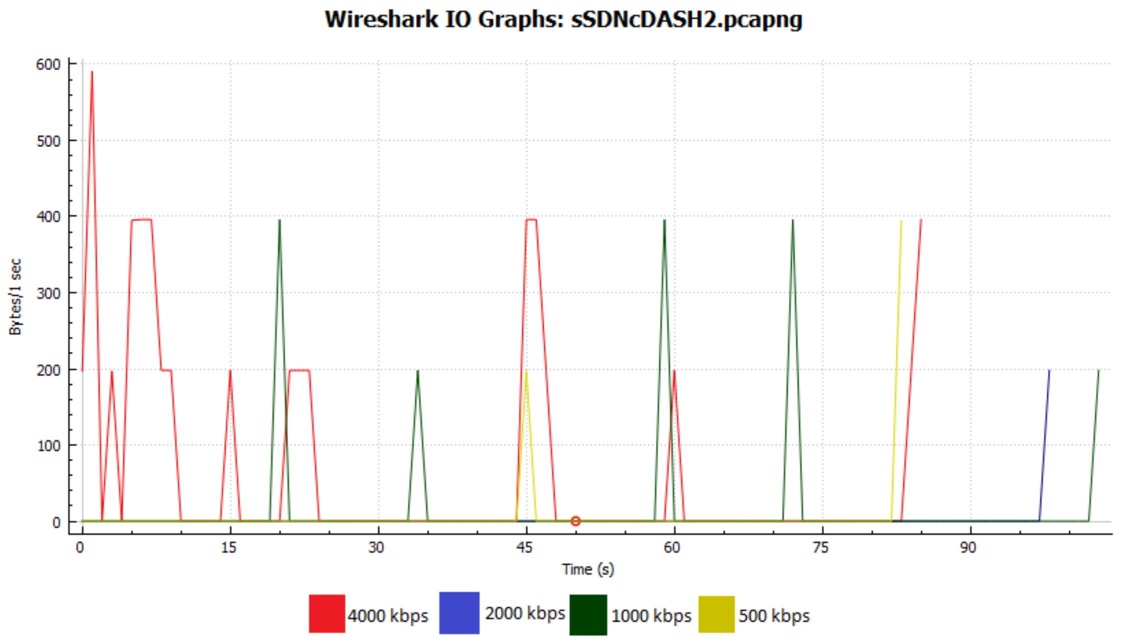


Figure 51: Downloaded Chunks for Scenario 2

This figure shows well the attempts from the player to adapt its reproduction to the network conditions. Though, those attempts are not well succeeded due to the iperf traffic that consumes all bandwidth. The attempts happen in the gaps between the iperf requests. It is possible to see that, in those gaps, the DASH player tries several times to retrieve a version with a lower bitrate to proceed with the playout. Though, as soon as the iperf traffic returns, the bandwidth for the video becomes insufficient again and the playout stops again.

4.4.3 Scenario 3: SDN without DASH

In opposition to the two previous scenarios, this third scenario expects very different results (see Figure 52). Since the SDN controller has activated the QoS-based rules, it is expected that the iperf traffic will be strongly limited, leaving a very high portion of the network resources to the video stream. Therefore, the video stream should have enough resources to get a smooth playback without being interfered by the iperf traffic.

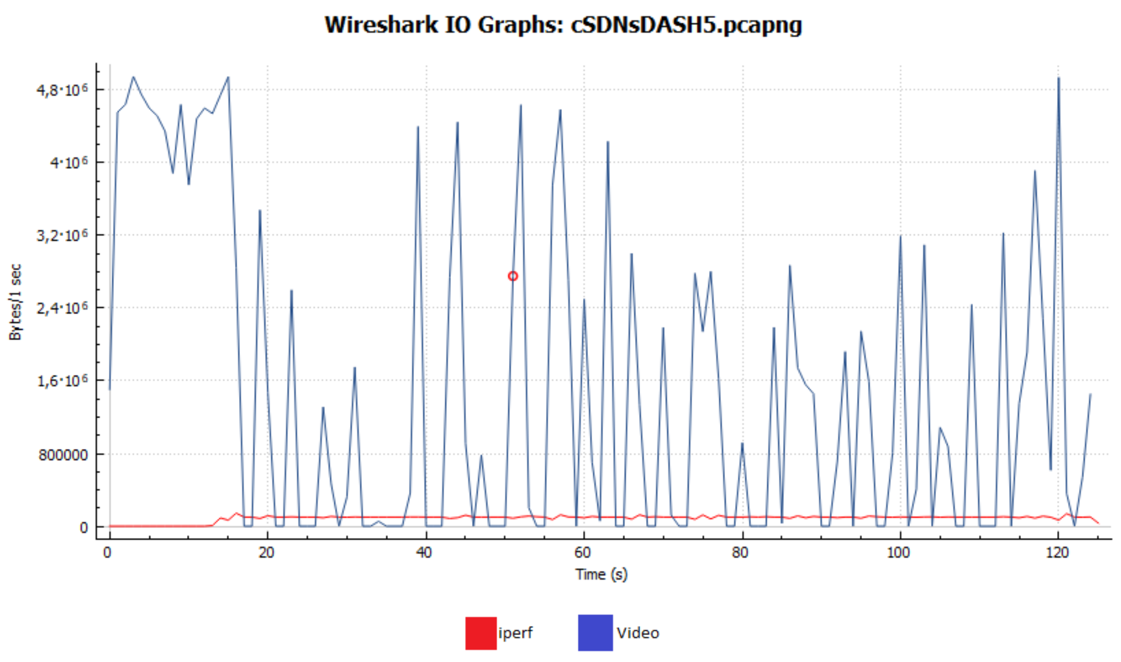


Figure 52: Results for Scenario 3

Indeed, the results are completely different from the first two scenarios. It is possible to see that the iperf traffic is strongly limited, reaching a very low bandwidth. Thus, the video stream is not disturbed by the iperf traffic and the playout goes on as if the iperf traffic did not exist. Figure 53 represents the SMA results for scenario 3.

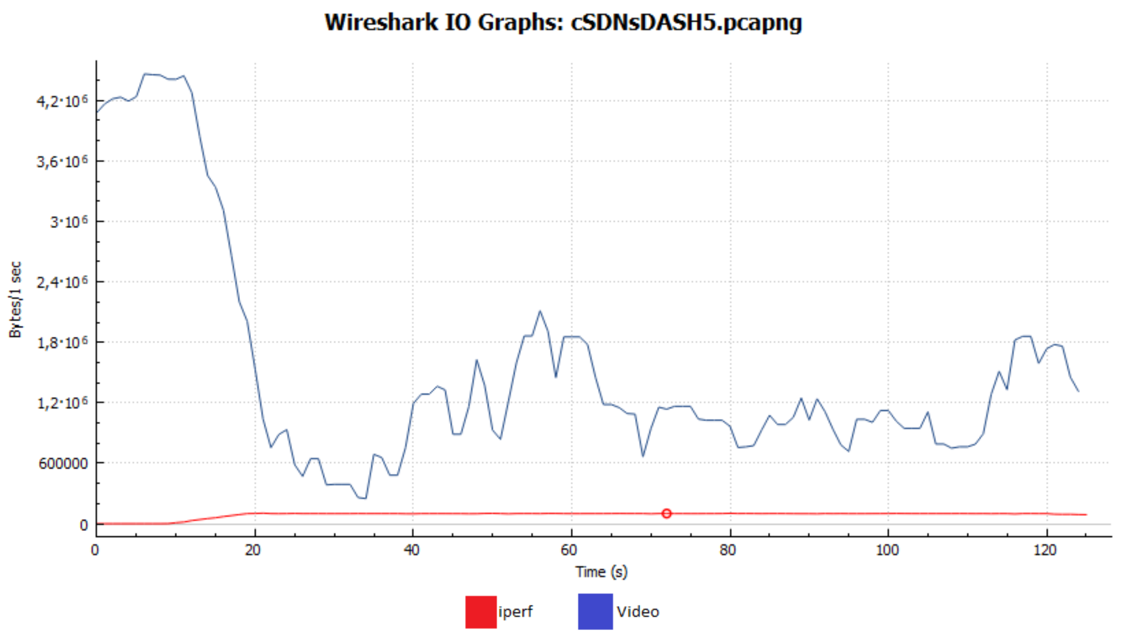


Figure 53: Simple Moving Average Results for Scenario 3

4.4.4 Scenario 4: SDN and DASH

Since in the previous scenario, the video stream already had enough resources to reproduce the content without any problem, it is to expect that in this scenario where both technologies are used, the results (see Figure 54) would be quite similar.

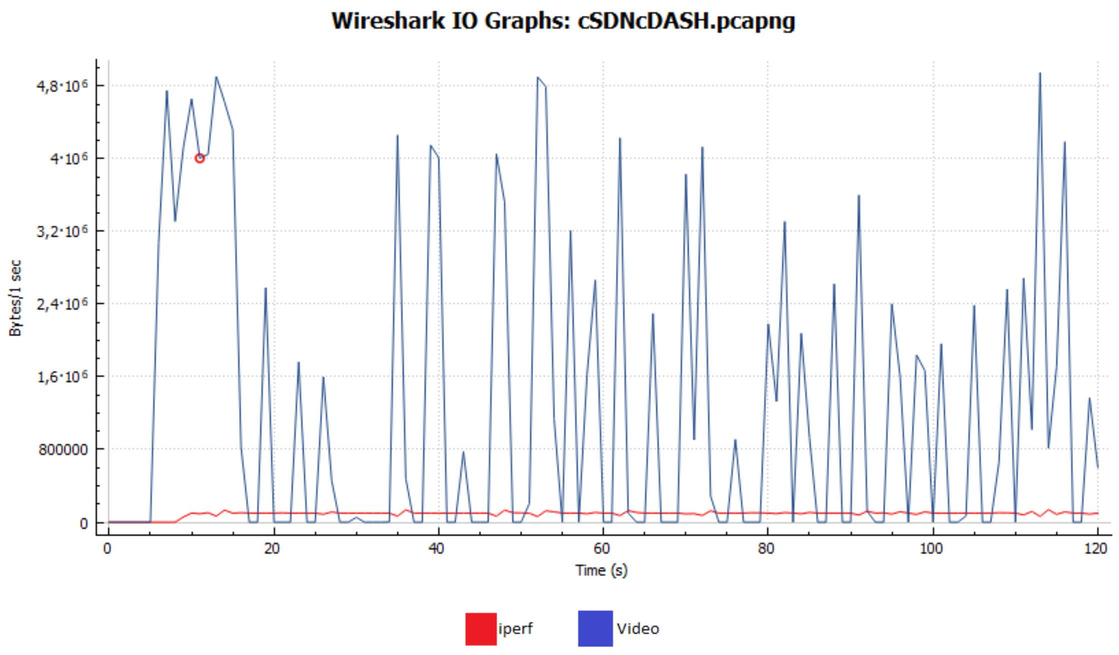


Figure 54: Results for Scenario 4

As predicted, the results in this scenario are pretty similar to the ones obtained in the third scenario. This can be explained by the fact that in this scenario, the player has enough resources to reproduce the highest representation which is the only representation that was available in the third scenario. Since in both scenarios, the players had enough resources to reproduce the highest version of the video, it is normal that the results for both are so similar. Figure 55 represents the SMA results for scenario 4.

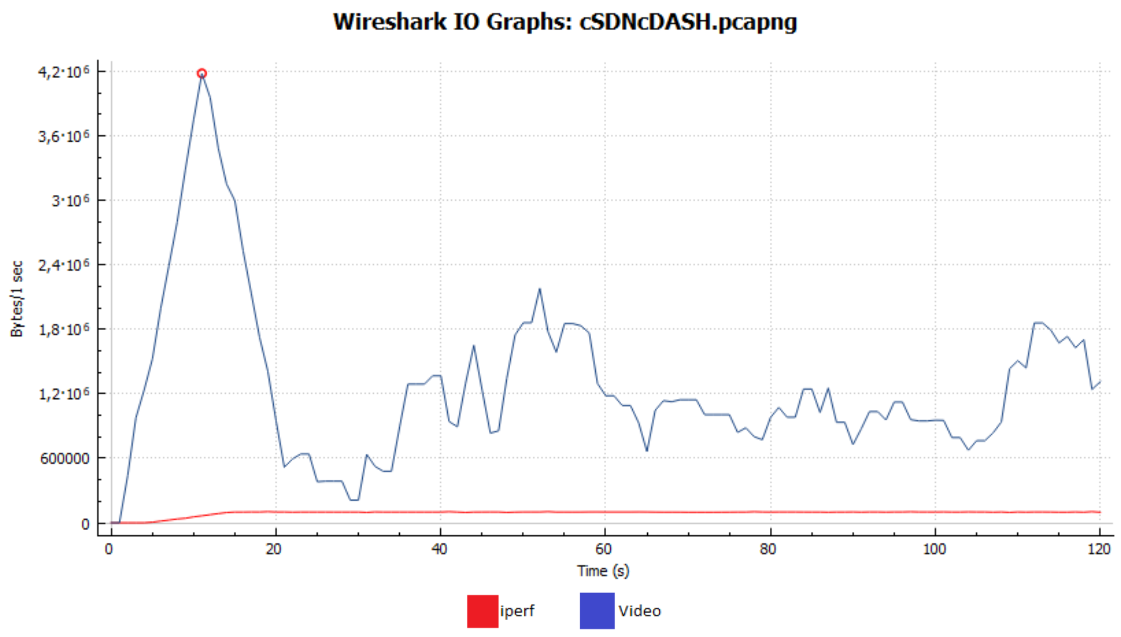


Figure 55: Simple Moving Average Results for Scenario 4

In Figure 56, it is also possible to see that the only version requested was the highest one (4000 kbps).

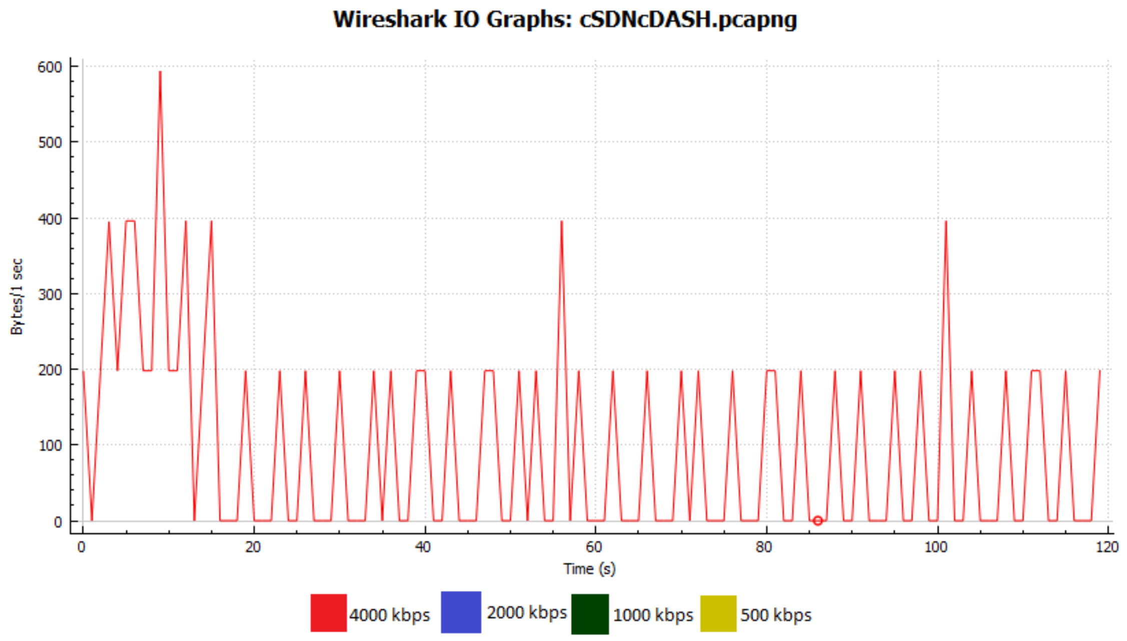


Figure 56: Downloaded Chunks for Scenario 4

This can be explained by the fact that the player did not need to "fight" for the network resources. Thus, despite having the possibility to switch among versions, the player "chooses" not to because it has enough resources to reproduce the highest version. This contrasts with the results from the second scenario where the player did not have enough resources to reproduce the content (even in the lowest version) and where it was possible to see several attempts to adapt the playback.

4.4.5 Additional Tests

As it was pointed out, the first scenarios could not fully evidence the importance of DASH in this solution. In order to correct that, it was decided to run some additional tests that could prove the efficiency of DASH. The problem in the initial scenarios was related to the aggressivity of the iperf traffic that consume either almost all or almost none of the bandwidth. Thus, the video client was practically never able to reproduce the video or the opposite where it could reproduce it without any trouble.

To design efficient tests, it was then necessary to implements testing designs where the iperf traffic was less aggressive so it could consume only part of the resources. This way, the video clients would have access to a varying bandwidth, and so, would have a playback that would vary accordingly to the available bandwidth (in the scenarios where DASH was used obviously).

To lower this aggressivity, two different strategies were adopted. In the first two scenarios, this was done simply by requesting a lower bandwidth in the command line. Though, in the last two scenarios, this was done in a different way. The iperf requests from the command line were done the same way as the original tests but the rate provided to the queue that would handle the iperf traffic were less limited. Since the limitation was less strong, the iperf traffic would have access to more bandwidth, providing less bandwidth to the video client (that would still be able to play the video though). This second strategy has the advantage of pointing out once again how SDN could provide more control of the network. Both strategies were designed in order to provide a 17 Mbits/s bandwidth to the iperf traffic remaining approximately 3 Mbits/s to the video traffic.

4.4.6 Additional Tests: Scenario 1

This scenario is predictable to have results (see Figure 57) not that different from the original one. Although the iperf traffic will not be as aggressive as it was in the original test, the video client still will not have enough resource to reproduce the video. This is also due to the fact that the video only has the highest version since it does not use DASH.

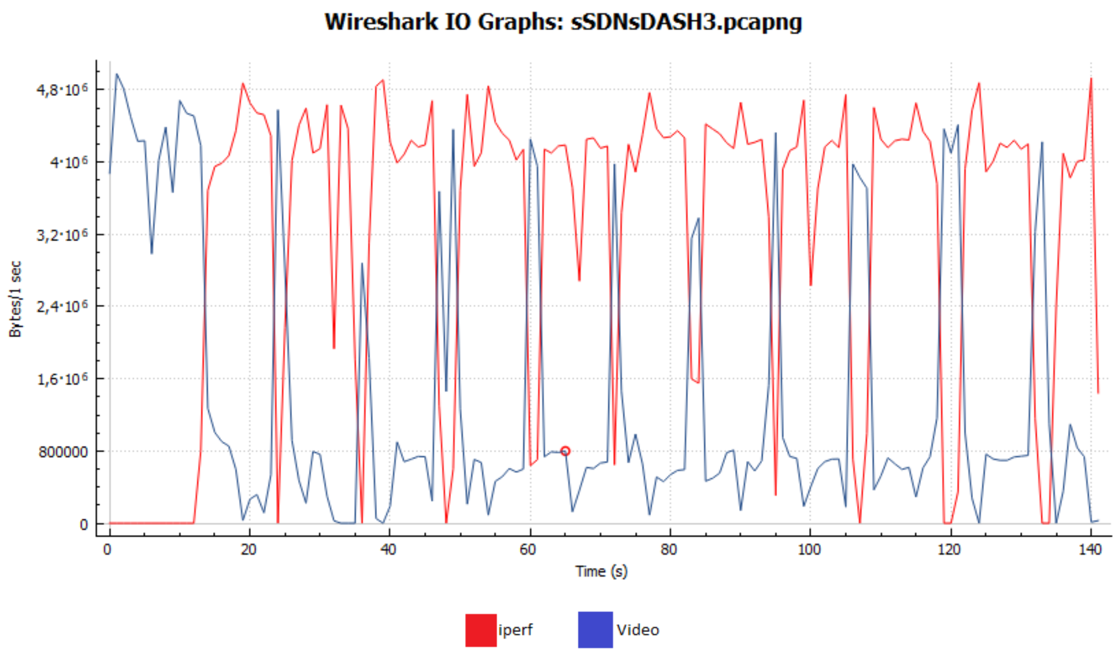


Figure 57: Results for the Additional Scenario 1

As predicted, the results were quite similar to the original scenario. Even though the iperf traffic was less intense, the video client still did not have enough resource to play the video. Though, it is possible to see that the video reached a bandwidth a little more stable than it did in the original scenario. Figure 58 represents the SMA results for additional scenario 1.

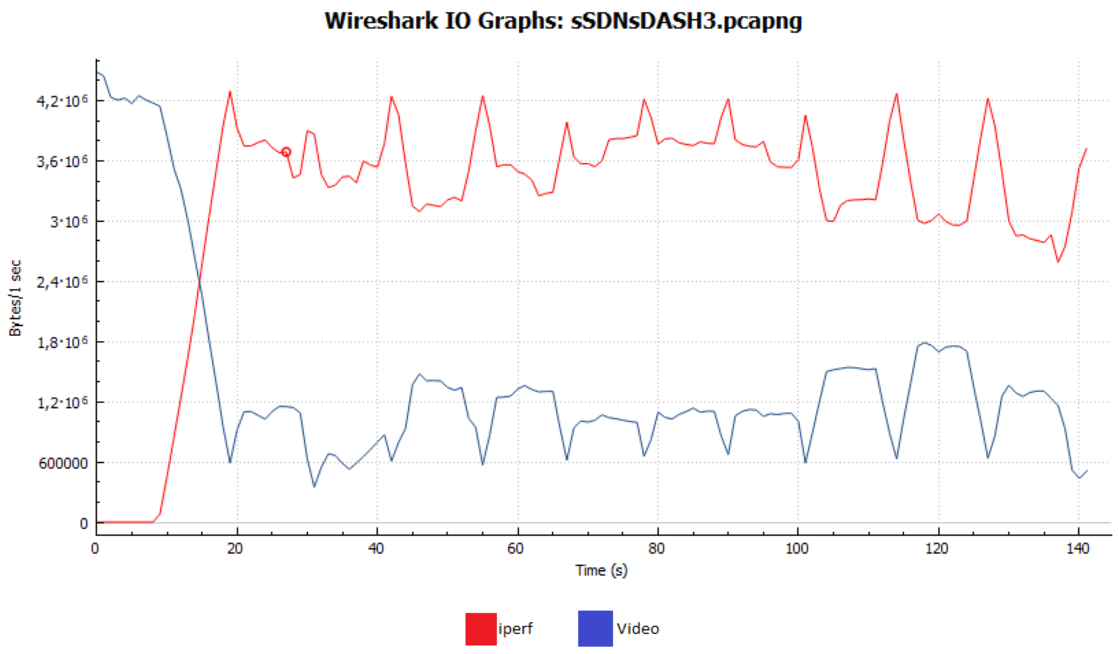


Figure 58: Simple Moving Average Results for the Additional Scenario 1

4.4.7 Additional Tests: Scenario 2

Unlike scenario 1, this scenario is expected to have distinct results (see Figure 59) both from the original scenario and from scenario 1. Since the iperf traffic would still leave some available bandwidth, it is to expect that the client could reproduce the video, even tough, it will not have sufficient resources to reproduce the highest version of the content.

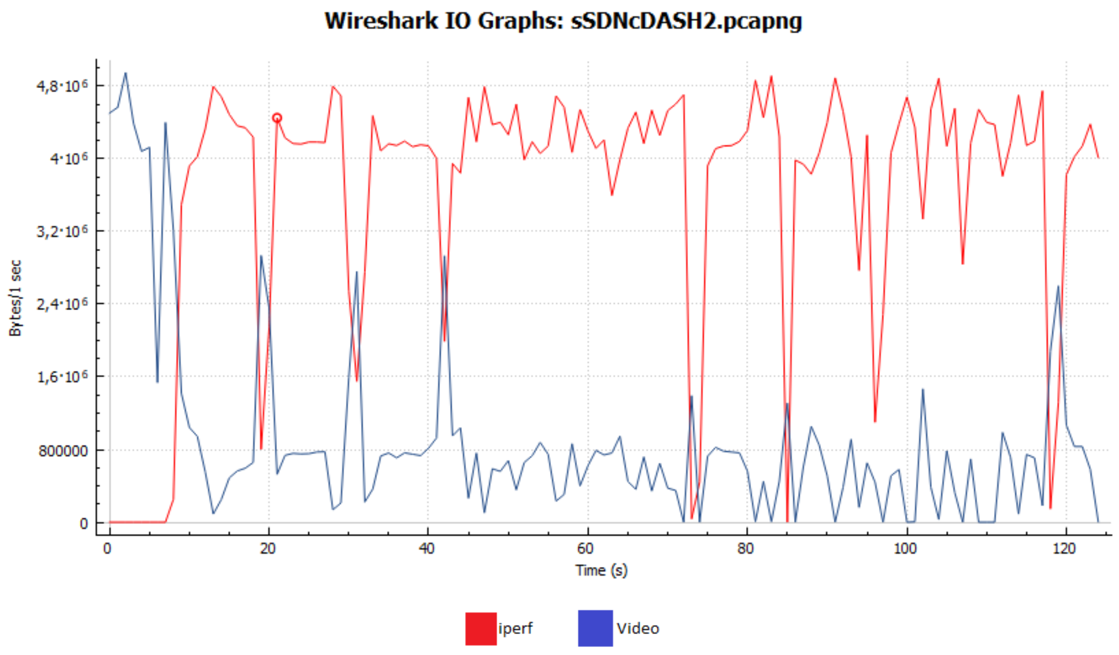


Figure 59: Results for the Additional Scenario 2

The results showed that the player was trying to adapt his playback to the available resources. The player is constantly trying to fetch the highest version possible and, although it does not reach the highest one, it is still able to go on with a lower version and not stop the playout as it happened both in the previous scenario and in the original one.

Figure 60 and Figure 61 are very important to understand two facts.

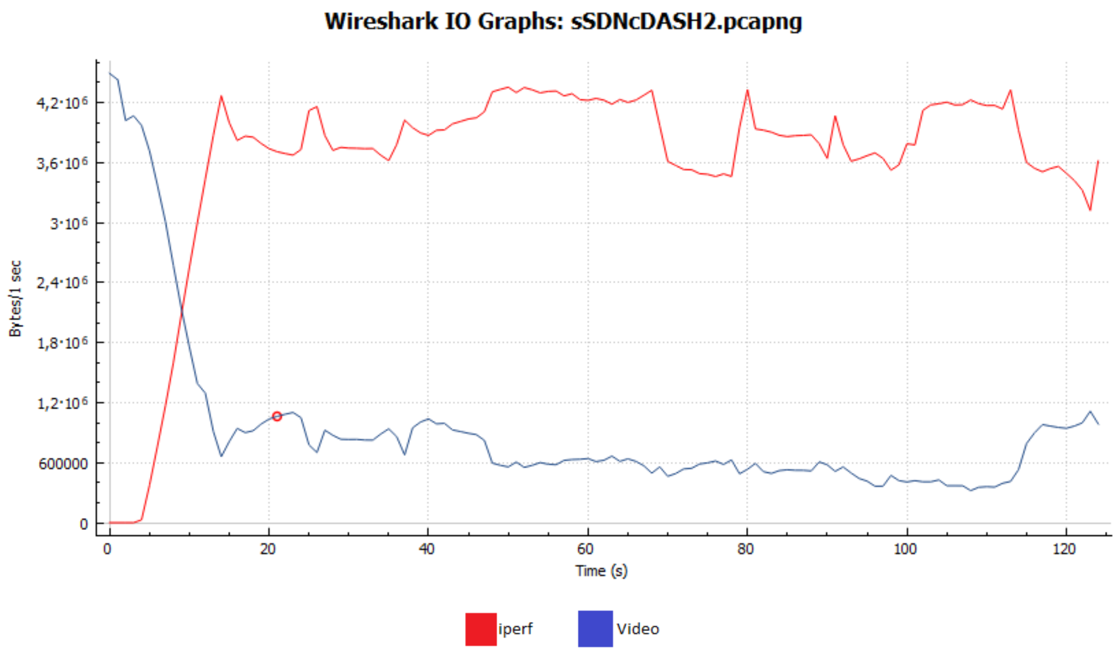


Figure 60: Simple Moving Average Results for Additional Scenario 2

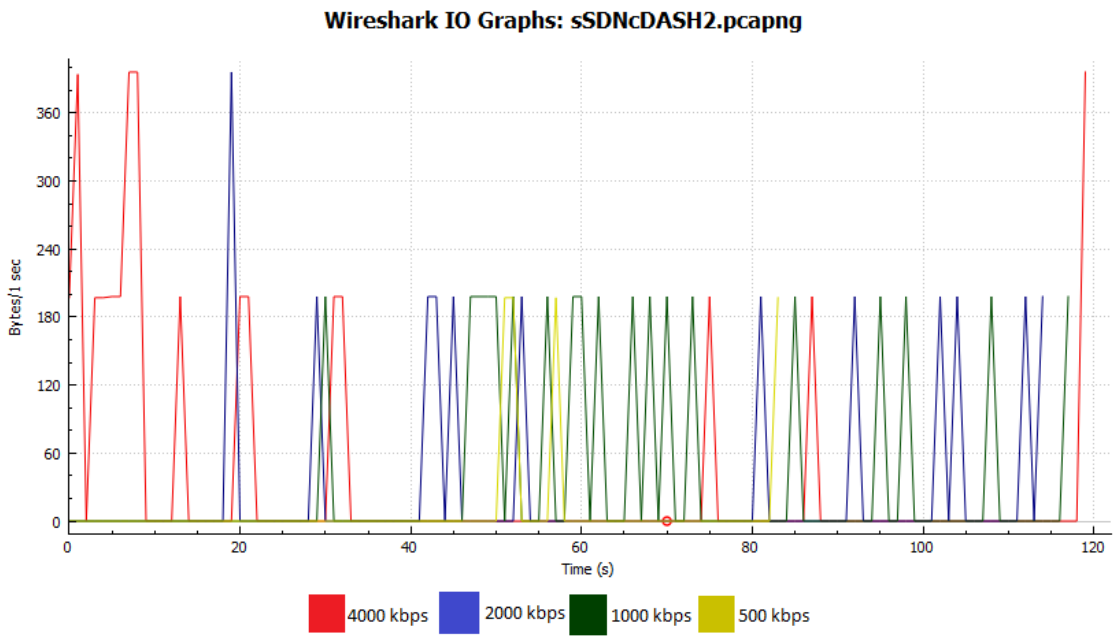


Figure 61: Downloaded Chunks for Additional Scenario 2

First looking at the average results, it is possible to see that they were not really higher than in the previous scenario. Although, that does not mean that it did not achieve better results. More than looking at the numbers and only compare the achieve bandwidth for each testing scenario, it is important to look at the graphs and understand the behavior of the network. As it was pointed out in the early chapters of this dissertation, the QoE is not only affected by the bitrate but also by other factors such as the delay, number of changes among the diverse video versions and most of all: the rebuffering frequency. If we only look at the achieved bandwidth, it is understandable that the previous scenario would achieve results not inferior to this one since the only used representation (4000 kbps) is higher than the ones that dominate this scenario.

Though, in the previous scenario, the playout had to stop every time an iperf injection appeared and the video had to stop for longer periods of time. This severely affects the user's QoE. In opposition, when DASH was used, the player was able to adapt the playout to the available resources and switch among versions in order not to stop the reproduction. In Figure 61, it is possible to see a dominance from the middle versions (2000 kbps & 1000 kbps). Thanks to the DASH protocol, the reproduction of the video never stopped, sacrificing a little of its visual quality towards a higher QoE.

4.4.8 Additional Tests: Scenario 3

This scenario is the one where the results (see Figure 62) are expected to be more distinct from its corresponding original scenario. In the original scenario, the iperf was so limited by the controller that the video client could easily reproduce the video even though it only had the highest version. Although, in this scenario, the limitation set of the iperf queue is by far less restrictive than it was in the original scenario. This way, since the iperf would only leave around 3 Mbits/s, it is to expect that the video client would not have enough bandwidth to reproduce the highest version, and so, would not have enough bandwidth to have a smooth playback at all. In sum, the results for this scenario should be similar from the first scenario.

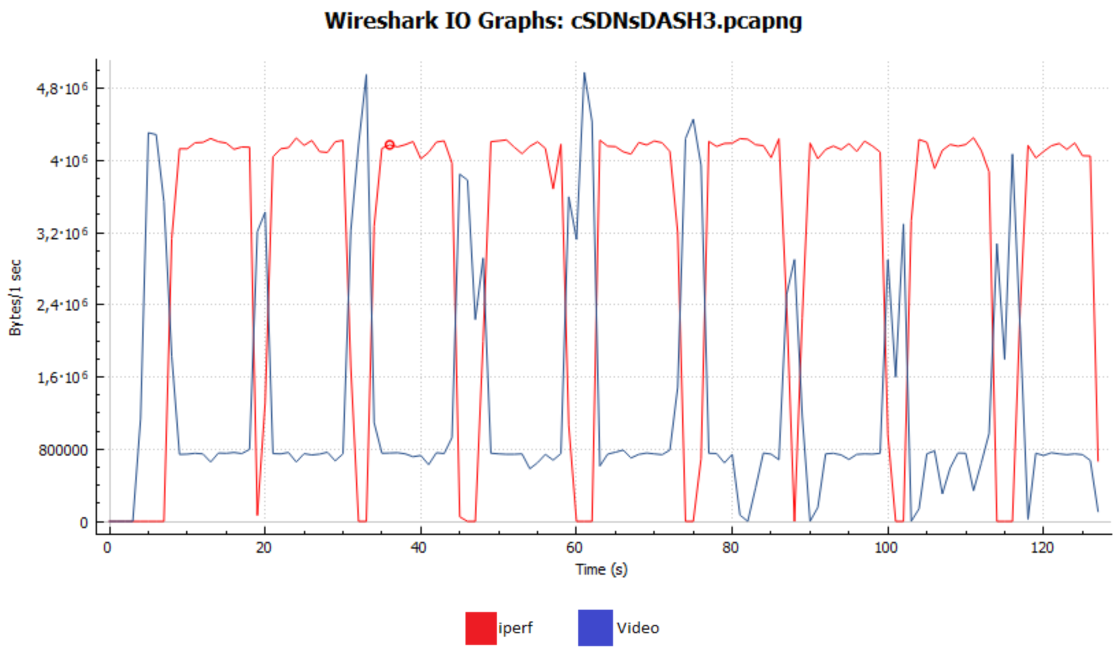


Figure 62: Results for the Additional Scenario 3

Here, it is possible to see that unlike what happened in the original scenario, the reproduction struggles to go on every time the iperf traffic appears. Even though the iperf traffic is less aggressive, it is still too strong to allow the video player to reproduce the highest representation. Figure 63 represents the SMA results for additional scenario 3.

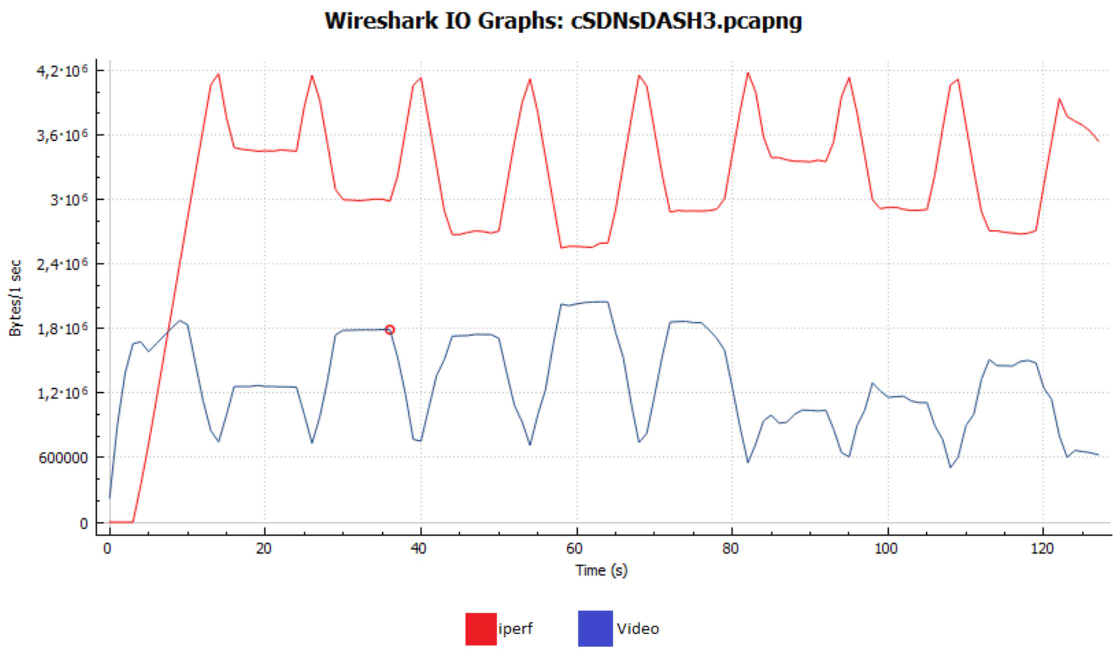


Figure 63: Simple Moving Average Results for the Additional Scenario 3

4.4.9 Additional Tests: Scenario 4

This scenario is the one where the relevance of the DASH protocol should be more evident. This scenario has more control on the network thanks to the SDN controller and, since the video player is expected to get access to a varying bandwidth, it is to expect that player should always react to the available bandwidth in order to reach the highest possible playback. In addition to that, this scenario should achieve very different results (see Figure 64) from the previous one, therefore, proving how DASH could enhance the QoE in equal situations.

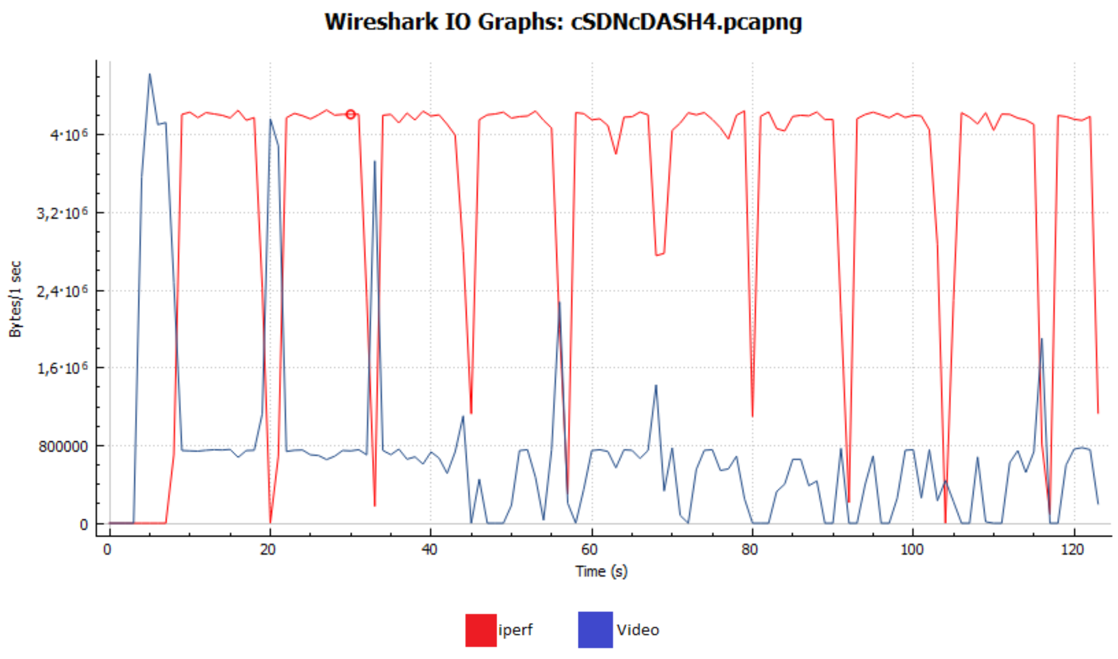


Figure 64: Results for the Additional Results for the Scenario 4

Once again, these results show the enhancement of QoE that DASH can bring thanks to its adaptation techniques. The graph clarifies how DASH adapts and settles for a lower version in order to proceed with its reproduction and adapt to the interference brought by the iperf traffic.

Figure 65 and Figure 66 show how DASH increases the stability of the reproduction through the positive contribution introduced by the SDN part of the proposed solution.

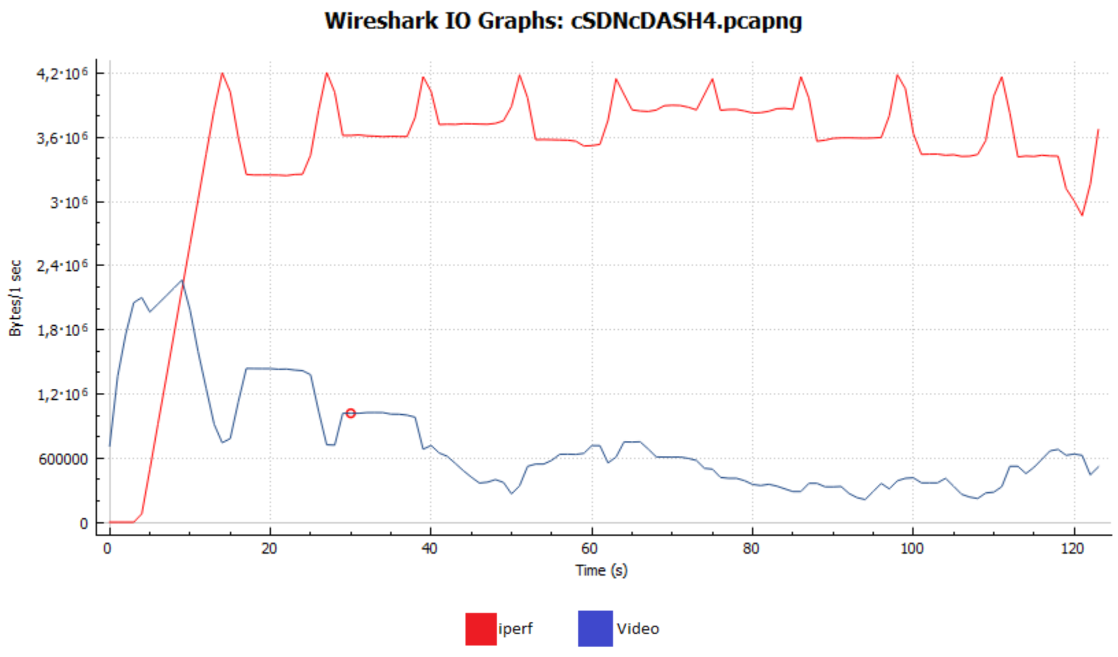


Figure 65: Simple Moving Average Results for the Additional Scenario 4

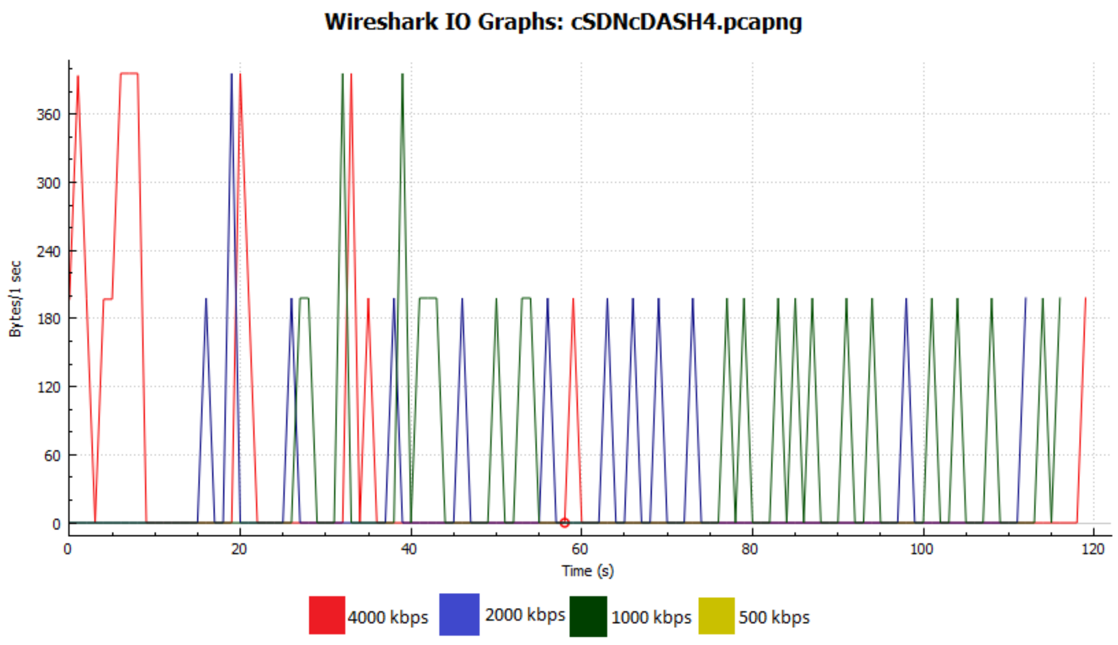


Figure 66: Downloaded Chunks for Additional Scenario 4

It is possible to see that the DASH player is able to perform a good estimation of the available resources and decide to choose a reproduction according to that. From these results we can conclude that faced to a constant interference, the DASH player will adopt a constant behavior that will allow it to keep the best possible reproduction against those conditions. From the moment that interference stops, it will be able to readapt and achieve the best playout again. Furthermore, if we compare these results with the ones from the additional scenario 2, it is possible to see that the SDN controller brings more control over the network and that, even though the rate attributed to the iperf traffic is the same, the player reaches more stability and settles for a certain version through a longer period of time.

Table 6: Summary of the Results

Scenario	Conclusion
1 – Neither SDN nor DASH	The client DASH player did not have enough resources to reproduce the video and the playout had to stop every time the concurrent iperf traffic was on
2 – DASH without SDN	Even though the player tried to fetch a lower version, the network resources remained insufficient to face the competition of the iperf traffic
3 – SDN without DASH	The player had plenty enough resources to play the content
4 – SDN and DASH	The player had plenty enough resources to play the content and it was stabilized reproducing the video version associated to the highest rate during the entire time of our experience
Additional 1	The player did not have enough resources to reproduce the video at its highest rate and the playout had to stop every time the iperf traffic appeared
Additional 2	Facing the competition of the iperf traffic the player was able to switch to lower video rate versions, including a few times to the lowest rate one, and go on with the video reproduction without any interruption
Additional 3	The player did not have enough resources to reproduce the video at its highest rate and the playout had to stop every time the iperf traffic appeared
Additional 4	The results evidence that the SDN part of the proposed solution in cooperation with the DASH control loop can increase the stability of the video reproduction at the client DASH player by diminishing the number of times that the DASH control loop changes among the diverse video versions, when we compare the current results with the ones of Additional 2. In addition, and by its opposition to what we have seen from the results of Additional 2, analyzing the current results, there is no evidence that the client DASH played anytime the lowest video rate (i.e., 500 kbps)

5 – Conclusions and Future Work

In this chapter, some final main conclusions are discussed and some indications about possible future work are anticipated.

5.1 Main Conclusions

One of the most currently accepted technology to provide QoE to the users in IP-based video streaming is the DASH protocol. Though, DASH can be highly sensitive to changes on the network load, including more severe cases where the network becomes congested. Besides the struggle that happen due to a high number of users within a network, nowadays networks are much more heterogeneous facing devices with different capacities, and devices that very quickly can connect to or disconnect from an edge network. Considering these scenarios, the network infrastructures become hard to be managed. In this way, some novel solutions are needed to bring more flexibility and a much faster response to the evolution within a network.

The current research aimed to propose and to study a novel solution that combines the DASH protocol and the SDN paradigm. The idea was to combine the adaptivity of the DASH mechanisms and its structure based on HTTP along with the flexible programming feature that SDN offers to the network management.

Though some researches had already been made combining DASH and SDN, they did not highlight the influence that each one of those tools could have on the enhancement of QoE to the user.

The obtained results in the current research showed that those technologies are efficient tools regarding enhancement of both video streaming and network management.

The influence of the SDN paradigm was clearly seeable in the results where the scenarios using SDN showed more promising results from the scenarios that did not. Additionally, this research has shown that, besides providing a higher control of the network and delivering a high amount of available network resources to a traffic type such as video streaming. SDN manages the network infrastructure via high-level abstracted management policies that enhance already existing low-level control mechanisms, such as the one that adjusts the rate of DASH video streaming. In addition, the DASH rate control, empowered by the positive contribution of SDN, ensures a more stable video rate, despite, the existence of bursts of UDP concurrent traffic.

The influence of DASH alone was harder to identify in the first results. This is mainly due to the characteristics of the scenarios where the traffic injected with iperf was so aggressive that it was almost impossible to observe the benefits that the adaptive mechanisms existing in the DASH protocol can have. To evidence how DASH could enhance QoE on its own, additional tests were run where the concurrent traffic was less aggressive. Those additional tests showed how DASH's adaptation techniques could have positive effects on a video playout especially regarding the rebuffering frequency that has been identifying as the most critical parameter that affects QoE. Besides that, it is also important to refer that the DASH specification permits the use of existing HTTP structures which are a very high portion of today's Internet infrastructures.

Thinking about the research question that served as based for this dissertation: "Can the SDN paradigm and the DASH protocol be combined altogether to achieve a better user QoE for video streaming with variable bitrate and a more efficient management of the available network resources?"

It is possible to answer that, in fact, those two technologies can be used to enhance QoE. Furthermore, those two technologies can be combined and, when they are, it is possible to reach a higher QoE for the users while achieving a more efficient use of the network resources from the network managers.

The evolution of Internet and network management is a journey that has never stopped since its creation and a lot of strategies and tools were used since that time. Obviously, this journey will continue and these two "vehicles" that are DASH and SDN showed that they still have a long road to go on that journey.

5.2 Future Work

As it was repeatedly pointed out throughout the current dissertation, the combination of DASH and SDN brings more flexibility and a high number of possible strategies.

Keeping this thought, there are several possible paths to continue with the current work. One of the major relevant part to explore is related to the discrimination of the traffic. In the current work, the video traffic was separated from the iperf traffic and different rates were assigned to different users. Future work could include a higher discrimination for more traffic types along with a higher discrimination of the users. The users' discrimination could be used to distinguish users accordingly to their devices' capabilities, and then to adapt the streams. Thus, each user could have access to a rate that could satisfy the specific needs of its device.

Another area that can be explored in the future work is related to the evaluation of the end-user’s QoE. In the current work, the system evaluation was based on the behavior of the DASH player. Future work could include an evaluation that could relate to some others experimental proposals that aim to objectively and subjectively evaluate the end-user’s QoE.

References

Abdullah, M. (2016). A Novel CDN Testbed for Fast Deploying HTTP Adaptive Video Streaming. *Proceedings of the 9th EAI International Conference on Mobile Multimedia Communications*, 65–71. https://doi.org/10.4108/eai.18-6-2016.2264163

Abuteir, R. M., Fladenmuller, A., Fourmaux, O., & Universites, S. (2017). Variable-Threshold Buffer Based Adaptation for DASH Mobile Video Streaming, 1–7.

Akhshabi, S., Begen, A. C., & Dovrolis, C. (2011). An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. *Proceedings of the Second Annual ACM Conference on Multimedia Systems - MMSys '11*, 157. https://doi.org/10.1145/1943552.1943574

Bedogni, L., Trotta, A., Di Felice, M., Gao, Y., Zhang, X., Zhang, Q., … Bononi, L. (2017). Dynamic Adaptive Video Streaming on Heterogeneous TVWS and Wi-Fi Networks, 1–14. https://doi.org/10.1109/TNET.2017.2728320

Bocchi, E., De Cicco, L., & Rossi, D. (2016). Measuring the Quality of Experience of Web users. *ACM SIGCOMM Computer Communication Review*, *46*(4), 8–13. https://doi.org/10.1145/3027947.3027949

Boron, R., & Project, O. (2016). OpenDaylight Documentation Documentation.

Bouten, N., Famaey, J., Latre, S., Huysegems, R., Vleeschauwer, B. D., Leekwijck, W. V., & Turck, F. D. (2012). QoE optimization through in-network quality adaptation for HTTP Adaptive Streaming. *Network and Service Management (Cnsm), 2012 8th International Conference and 2012 Workshop on Systems Virtualiztion Management (Svm)*, 336–342.

Cetinkaya, C., Karayer, E., Sayit, M., & Hellge, C. (2015). SDN for segment based flow routing of DASH. *IEEE International Conference on Consumer Electronics - Berlin, ICCE-Berlin*, *2015–Febru*(February), 74–77. https://doi.org/10.1109/ICCE-Berlin.2014.7034284

Cisco. (2017). Cisco Visual Networking Index: Forecast and Methodology, 2015-2020. *Forecast and Methodology*, 22. https://doi.org/1465272001663118

Cofano, G., Cicco, L. De, Zinner, T., Nguyen-Ngoc, A., Tran-Gia, P., & Mascolo, S. (2017). Design and Performance Evaluation of Network-assisted Control Strategies for HTTP Adaptive Streaming. *ACM Transactions on Multimedia Computing, Communications, and Applications*, *13*(3s), 1–24. https://doi.org/10.1145/3092836

Controller - OpenDaylight Documentation. (2018). Retrieved October 21, 2018, from https://docs.opendaylight.org/en/stable-oxygen/developer-guide/controller.html#overview

Dhody, D., Architect, S. S., & Technologies, H. (2015). Various Alternatives to achieve SDN.

Estive, W. F., Jiang, J., Sekar, V., & Zhang, H. (2014). Improving Fairness , Ef fi ciency , and Stability in HTTP-Based Adaptive Video Streaming, *22*(1), 326–340.

Georgopoulos, P., Elkhatib, Y., Broadbent, M., Mu, M., & Race, N. (2013). Towards network-wide QoE fairness using openflow-assisted adaptive video streaming. *Proceedings of the 2013 ACM SIGCOMM Workshop on Future Human-Centric Multimedia Networking - FhMN '13*, (November 2015), 15. https://doi.org/10.1145/2491172.2491181

Goldenberg, D. K., Qiuy, L., Xie, H., Yang, Y. R., & Zhang, Y. (2004). Optimizing cost and performance for multihoming. *ACM SIGCOMM Computer Communication Review*, *34*(4), 79. https://doi.org/10.1145/1030194.1015478

Heller, B. (2009). OpenFlow Switch Specification 1.0.0. *Current*, *0*, 1–36.

https://doi.org/10.1002/2014GB005021

Introduction to Mininet. (2018). Retrieved October 21, 2018, from https://github.com/Mininet/Mininet/wiki/Introduction-to-Mininet

iPerf - The TCP, UDP and SCTP network bandwidth measurement tool. (2018). Retrieved October 31, 2018, from https://iperf.fr/

Jimenez, J. M., Romero, O., Rego, A., Dilendra, A., & Lloret, J. (2016). Study of Multimedia Delivery over Software Defined Networks. *Network Protocols and Algorithms*, *7*(4), 37. https://doi.org/10.5296/npa.v7i4.8794

Joskowicz, J., & Ardao, J. C. L. (2012). A parametric model for perceptual video quality estimation. *2012 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting*, *49*, 49–62. Retrieved from http://www.springerlink.com/index/Y35J3661VP267134.pdf

Kleinrouweler, J. W., Cabrero, S., & Cesar, P. (2017). An SDN Architecture for Privacy-Friendly Network-Assisted DASH. *ACM Transactions on Multimedia Computing, Communications, and Applications*, *13*(3s), 1–22. https://doi.org/10.1145/3092838

Kreutz, D., Ramos, F. M. V., Verissimo, P., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2014). Software-Defined Networking: A Comprehensive Survey, 1–61. https://doi.org/10.1109/JPROC.2014.2371999

Kua, J., Armitage, G., & Branch, P. (2017). A Survey of Rate Adaptation Techniques for Dynamic Adaptive Streaming Over HTTP. *IEEE Communications Surveys & Tutorials*, *19*(3), 1842–1866. https://doi.org/10.1109/COMST.2017.2685630

Liu, X., Dobrian, F., Milner, H., Jiang, J., Sekar, V., Stoica, I., & Zhang, H. (2012). A case for a coordinated internet video control plane. *ACM SIGCOMM Computer Communication Review*, *42*(4), 359. https://doi.org/10.1145/2377677.2377752

Lu, Z., Lei, T., Wen, X., Wang, L., & Chen, X. (2016). SDN based user-centric framework for heterogeneous wireless networks. *Mobile Information Systems*, *2016*. https://doi.org/10.1155/2016/9874969

Members - OpenDaylight. (2018). Retrieved October 21, 2018, from https://www.opendaylight.org/support/members

Mu, M., Broadbent, M., Farshad, A., Hart, N., Hutchison, D., Ni, Q., & Race, N. (2016). A scalable user fairness model for adaptive video streaming over SDN-assisted future networks. *IEEE Journal on Selected Areas in Communications*, *34*(8), 2168–2184. https://doi.org/10.1109/JSAC.2016.2577318

O. V. Switch. (2014). Open vSwitch, *31*(5), 1–58.

Ochoa Aday, L., Cervelló Pastor, C., & Fernández Fernández, A. (2015). Current Trends of Topology Discovery in OpenFlow-based Software Defined Networks. *International Journal of Distributed Sensor Networks*, *5*(2), 1–6. https://doi.org/10.1109/LCN.2015.7366363

Politis, I., Tselios, C., Lykourgiotis, A., & Kotsopoulos, S. (2017). On optimizing scalable video delivery over media aware mobile clouds. *IEEE International Conference on Communications*. https://doi.org/10.1109/ICC.2017.7996601

Postman Introduction. (2018). Retrieved October 31, 2018, from https://www.getpostman.com/docs/v6/

Sodagar, I., Vetro, A., & Sodagar, I. (2011). Industry and Standards The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE MultiMedia*, *18*(4), 62–67. https://doi.org/10.1109/MMUL.2011.71

Taha, M., Garcia, L., Jimenez, J. M., & Lloret, J. (2017). Networks for Heterogeneous Adaptive Video Streaming Applications, 963–968.

VideoLAN - Documentation. (2018). Retrieved October 31, 2018, from https://www.videolan.org/doc/

White Russ. (2017). Interface to the Routing System: In the Background. Retrieved October 21, 2018, from https://blog.ecitele.com/i2rs-in-the-background

Yin, X., Jindal, A., Sekar, V., & Sinopoli, B. (2015). A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. *Proc. ACM SIGCOMM*, 325–338. https://doi.org/10.1145/2785956.2787486

Yuval Fisher, R. N. (2014). An Overview of HTTP Adaptive Streaming Protocols for TV Everywhere Delivery.

Zhou Wang, & Alan C. Bovik, Fellow, I. (2004). Image Quality Assessment: From Error Visibility to Structural Similarity. *Ieee Transactions on Image Processing, 13*(4), 1–14. https://doi.org/10.1109/TIP.2003.819861

Zorzi, M., Zanella, A., Testolin, A., De Filippo De Grazia, M., & Zorzi, M. (2016). COBANETS: A new paradigm for cognitive communications systems. *2016 International Conference on Computing, Networking and Communications, ICNC 2016*. https://doi.org/10.1109/ICCNC.2016.7440625