Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

DISTRIBUCIÓN MULTICAST DE CONTENIDO MULTIMEDIA EN UN ESCENARIO DE RED VIRTUALIZADO APLICANDO LAS TECNOLOGÍAS DASH Y SDN

TRABAJO FIN DE MÁSTER

Luis Enrique Reyes Zabala

2019

Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

Máster Universitario en
Ingeniería de Redes y Servicios Telemáticos

TRABAJO FIN DE MÁSTER

DISTRIBUCIÓN MULTICAST DE CONTENIDO MULTIMEDIA EN UN ESCENARIO DE RED VIRTUALIZADO APLICANDO LAS TECNOLOGÍAS DASH Y SDN

Autor
Luis Enrique Reyes Zabala

Director
Luis Bellido Triana

Departamento de Ingeniería de Sistemas Telemáticos

2019

Resumen

En los últimos años, hemos presenciado como el sector de las telecomunicaciones ha avanzado a pasos agigantados, lo que ha propiciado el nacimiento de nuevos servicios en la red y la mejora de los servicios existentes. Todo este escenario ha conllevado a la convergencia de servicios multimedia, como video conferencia, video bajo demanda, transmisiones en vivo, entre otros, y del surgimiento de empresas que a día de hoy generan gran parte del tráfico de internet. Por lo tanto, ha surgido la necesidad de desarrollar nuevas tecnologías que permitan garantizar altos niveles de satisfacción en los usuarios de dichos servicios.

Para atender esta problemática, se han desarrollado diferentes técnicas y protocolos. Sin embargo, el Streaming Dinámico Adaptativo sobre HTTP (DASH por sus siglas en inglés), que permite que los reproductores multimedia adaptar la calidad del video de acuerdo a las condiciones de la red, es hoy en día la tecnología más popular para realizar streaming multimedia en internet, al punto de estar estandarizada.

Por su parte, el surgimiento de las Redes Definidas por Software ofrece cambiar las limitaciones de las infraestructuras de red actuales, mediante su principal característica: la separación del plano de control del plano de datos y alojarlo en una unidad lógica de control centralizada. Esto produce que la red sea programable y más flexible que las redes tradicionales.

En este Trabajo Fin de Máster se desea demostrar que es posible aprovechar las bondades que ofrecen tanto SDN como DASH en la transmisión de video en multicast sobre redes móviles. Para ello se utilizó una amplia gama de herramientas, entre las que destacan: VNX para construir topologías de red virtualizadas sobre las que se desarrollaron diferentes pruebas, el controlador de red ONOS junto con sus aplicaciones, y el software UFTP para realizar el envío de ficheros. Adicionalmente, se explica detalladamente el estado del arte de estas tecnologías.

Palabras claves: multicast, SDN, DASH, VoD, video streaming.

Abstract

In the recent years, we have witnessed how the telecommunications industry has enormously grown, which led to originate new services over the web, and the improvement of existing ones. This scenario has produced the convergence of multimedia services, such as video conference, video on demand, live streaming, among others, and the birth of great enterprises that now generate a huge portion of the overall traffic on the web. Therefore, it has been necessary to develop new technologies that guarantee high levels of satisfaction among users.

To solve these new challenges, different techniques and protocols have been developed. However, Dynamic Adaptive Streaming over HTTP, also known as DASH, has turned out to be the most popular technology to stream multimedia content, to the extent of being standardized, because it allows multimedia players to adapt the quality of the video according to the state of the network, which translate in less freezing throughout playback.

On the other hand, the rise of Software Defined Networks, offers to change all limitations related to traditional networks, through it most valuable proposal: the separation of the control plane from the data plane. The control plane is housed in a centralized control logic unit, which entails the network to be programmable and more flexible than the traditional ones.

The main aim of this Project is to demonstrate that it is possible to take advantage of the benefits provided by DASH and SDN in multicast video broadcasting over mobile networks. In order to achieve so, a wide range of tools have been used. Among these tools are: VNX to design and create virtualized network topologies to perform different tests over, ONOS network controller and its applications and UFTP to send files over the network. Additionally, an state-of-the-art of these technologies is given in details.

Key words: multicast, SDN, DASH, VoD, video streaming.

Índice general

Índice de figuras

Índice de tablas

Siglas

3GPP	Third Generation Partnership Project
ACK	Acknowledgement
ACL	Asynchronous Layered Coding
AL-FEC	Application Layer Forward Error Correction
API	Application Programming Interface
ATSC	Advanced Television Systems Committee
CDN	Content Delivery Network
DASH	Dynamic Adaptative Streaming Over HTTP
DP	Data Plane
eMBMS	Evolved Multimedia Broadcast Multicast Service
ETSI	European Telecommunications Standars Institute
FD	Forwarding Device
FEC	Forwarding Error Correction
SDP	Session Description Protocol
IGMP	Internet Group Management Protocol
FLUTE	File Delivery Over Unidirectional Transport
GPL	General Public Licence
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
IPTV	Internet Protocol Television
ISP	Internet Service Provider
LCT	Layered Coding Transport
LXC	Linux Containers
MAC	Media Access Control
MFWD	Multcast Forwarding
MP	Management Plane
MPD	Multimedia Presentation Description
MPEG	Moving Picture Experts Group
MRT	Multicast Routing Tables
NAT	Network Address Translator
NI	Northbound Interfac
ODL	Open DayLight
OF	OpenFlow
ONF	Open Network Foundation
PC	Control Plane
QoE	Quality of Experience

QoS	Quality of Service
RAM	Random Access Memory
REST	Representational State Transfer
ROUTE	Real Time Object Delivery Over Unidirectional Transport
SDM	Software Defined Multicast
SDN	Software Defined Networks
SI	Southbound Interface
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VNX	Virtual Networks over linuX

1 Introducción

1.1 Contexto

Los protocolos de red de control distribuido y transporte que se ejecutan dentro de los routers y switches son la tecnología clave para que la información, en forma de paquetes digitales, pueda viajar alrededor del mundo, pero, a pesar de su amplia adopción, las redes IP son complejas y difíciles de gestionar. Para expresar las políticas de red de alto nivel, los operadores de red necesitan configurar cada dispositivo usando comandos propios de algún fabricante. Adicionalmente, las redes deben soportar fallas (que son de carácter dinámico), y adaptarse a los cambios de carga.

Para hacerlo aún más complicado, las redes tradicionales están verticalmente integradas, lo que significa que el plano de control y el plano de datos están incluidos dentro del mismo dispositivo de red, lo que reduce flexibilidad y obstaculiza la innovación y la evolución de la infraestructura de red.

En los últimos años hemos sido testigos de grandes avances en el área de las telecomunicaciones, lo que ha propiciado la convergencia de servicios multimedia, como video conferencia, video bajo demanda, transmisiones en vivo, entre otros, y del crecimiento exponencial de empresas de streaming como YouTube o Netflix, al punto que hoy en día, entre estas dos compañías se genera aproximadamente la mitad del tráfico total de internet.

Sin embargo, a pesar de las mejoras, y debido al incremento constante del tráfico proveniente de aplicaciones multimedia, siguen existiendo problemas que dificultan la entrega del video y que afectan negativamente la experiencia percibida por los usuarios finales.

Organismos de estandarización han desarrollado distintas tecnologías buscando mejorar los procesos de encapsulación y transporte de contenido multimedia, pero no ha sido hasta hace unos pocos años que MPEG-DASH se ha consolidado como el estándar que busca proveer servicios de streaming de video ininterrumpido a usuarios con condiciones de redes dinámicas, mediante el uso de algoritmos de adaptación de bitrate en la capa aplicación de las redes.

Por su parte, las Redes Definidas por Software (SDN) son un paradigma de red emergente que ofrece cambiar las limitaciones de las infraestructuras de las redes actuales. Primeramente, rompe la integración vertical al separar el control lógico de la red (plano de control) de los routers y switches que reenvían el tráfico (plano de datos). En segundo lugar, con la separación del plano de control del plano de datos, los switches se convierten en simples dispositivos de reenvío y el control es implementado por una unidad centralizada, simplificando así la evolución y configuración de la red.

Las tecnologías SDN y DASH tienen el potencial de mejorar considerablemente la transmisión de video bajo demanda y en vivo sobre la red, y por lo tanto se puede aumentar la calidad de experiencia percibida por los usuarios finales que suscriben estos servicios. Sin embargo, la intersección de estas dos áreas es un campo de estudio muy novedoso sobre el cual se están desarrollando un gran número de investigaciones, y por lo tanto no se cuenta con tanta información en comparación con técnicas tradicionales.

Para el desarrollo de este Trabajo Fin de Máster se recreó un escenario de red donde se realiza la transmisión de video en vivo y bajo demanda, tomando ventaja de la flexibilidad y los beneficios que ofrecen los controladores SDN, y aprovechando las bondades de la codificación del video de acuerdo al estándar MPEG-DASH. Es importante resaltar que durante la fase de investigación y desarrollo se tomó como ejemplo y guía las investigaciones [13] y [14]

1.2 Objetivos

- Distribuir contenido multimedia en una topología de red virtualizada, aprovechando las bondades ofrecidas por DASH, SDN y multicast.

  Para alcanzar este objetivo, fue necesario abordar las siguientes tareas:

  - Analizar en detalle el Streaming Dinámico Adaptativo sobre HTTP, la técnica más popular para la transmisión de video en redes de comunicación.
  - Describir el estado del arte de SDN, abarcando sus aspectos más importantes como la arquitectura, terminología básica y los controladores más populares de la actualidad.
  - Diseñar y construir una topología de red que represente a un escenario real simplificado, sobre el que se pueda transmitir y recibir ficheros de video.
  - Codificar y segmentar un video de acuerdo al estándar MPEG-DASH.
  - Realizar el envío en multicast de ficheros de video codificados de acuerdo al estándar MPEG-DASH entre un servidor web y un cliente.
  - Diseñar un mecanismo de recuperación de segmentos de video en unicast, el cual actuará cuando se pierdan ficheros transmitidos en multicast.

1.3 Estructura del documento

En esta memoria se documenta toda la información relacionada con los fundamentos teóricos, los procedimientos experimentales y las conclusiones obtenidas durante el desarrollo de este Trabajo Fin de Máster.

En el capítulo 2 se realiza una reseña sobre el streaming multimedia, se explica cómo la mayoría de las primeras investigaciones se concentraban en proveer streaming en tiempo real sobre UDP porque se consideraba que TCP no era apto para estos fines, y como luego se adopta el uso de HTTP sobre TCP para la transmisión de video, gracias a las amplias ventajas que ofrece este enfoque.

El siguiente punto de este capítulo es una explicación detallada de los aspectos básicos de DASH, como lo son sus características, arquitectura y funcionamiento. Posteriormente, se hace una definición del envío de datos en multicast, y como este es aplicado a DASH, resaltando sus beneficios para la transmisión de contenido multimedia.

El capítulo 3 abarca todos los fundamentos conceptuales relacionados con las Redes Definidas por Software. Específicamente, se cubren los aspectos básicos de las SDN como lo son las características principales y la terminología básica relacionada con este paradigma de las redes. También se da una explicación detallada de la arquitectura SDN, incluyendo cada una de las capas que la conforman y las funciones que cumplen cada una de ellas.

En este capítulo también se habla sobre el protocolo OpenFlow, y se ahonda en sus características, fundamentos básicos y ventajas principales. Además se hace una descripción del estado del arte de los controladores SDN de software libre más populares en la actualidad, haciendo mayor énfasis en ONOS. Finalmente se realiza una comparativa entre las características y posibilidades de multicast que ofrece cada uno de los controladores descritos para proceder a seleccionar el que se utilizará durante el desarrollo de este proyecto.

El capítulo 4 reseña todo lo relacionado al diseño, construcción y pruebas de funcionalidad del escenario de red virtual sobre el cual se realizaron los distintos experimentos de este proyecto. En primer lugar se nombran y describen las herramientas de software que se emplearon para el diseño y la construcción del escenario. Seguidamente, se realiza una descripción de cada uno de los componentes de la topología de la red y sus respectivas configuraciones, para finalmente, explicar las pruebas que se realizaron para validar el correcto funcionamiento del escenario.

En el capítulo 5 se habla detalladamente sobre todos los procedimientos que se realizaron durante el desarrollo de este proyecto. En primer lugar se explica la metodología seguida para la transcodificación del video, su segmentación y la generación del fichero MPD. Seguidamente, se especifica todo lo relacionado a los escenarios de prueba, tanto en el caso de video bajo demanda, como en el de streaming

en vivo. Se ahonda en las ideas sobre las que se trabajaron, los pasos que se siguieron, los problemas que se presentaron y las soluciones que se les dieron a dichos problemas.

Finalmente, en el capítulo 6, se enumeran todas las conclusiones que se desprenden de la realización de este Trabajo Fin de Máster, y se proponen algunas líneas futuras de investigación.

2 Streaming Multimedia

Originalmente, Internet no fue diseñado para la entrega sostenida de aplicaciones modernas, como el streaming multimedia de alta calidad que demandan grandes cantidades de recursos. La diferencia fundamental entre el tráfico de datos tradicionales y el tráfico de datos multimedia son las restricciones en tiempo real del segundo. La mayoría de las primeras investigaciones que se realizaron en la transmisión de paquetes de vídeo se concentraban en proveer streaming en tiempo real sobre UDP con técnicas que soportaban reserva de recursos y calidad de servicio (QoS), y configurar sistemas finales para soportar la transmisión de video. Sin embargo, estas técnicas presentaban problemas atravesando NATs y firewalls, y por lo tanto, requerían servidores y arquitectura de red dedicados, lo que incrementaba considerablemente los costos de despliegue y añadía muchas complejidades [8].

Por su parte TCP es un protocolo que garantiza la entrega de paquetes, a expensas de retrasos variables, que son consecuencia de la espera de los ACK por parte del emisor y la retransmisión de paquetes perdidos. Considerando que el video es intolerable a los retrasos y que por lo general no necesita alta confiabilidad para ser aceptado, se asumió que TCP no era apto para la entrega de contenido multimedia.

2.1 Streaming de video sobre HTTP

Al inicio de la década del 2000, los investigadores aceptaron que TCP ofrece beneficios para la transmisión de video, y por lo tanto se introdujo un buffer de reproducción en la capa de aplicación para compensar las fluctuaciones de bitrate de TCP. Utilizar HTTP sobre TCP resulta ser muy conveniente ya que presenta los siguientes beneficios:

- Los clientes usan el protocolo estándar HTTP, lo que provee mayor alcance ya que este tipo de tráfico puede atravesar tanto NATs como firewalls.
- Permite el despliegue de caches para mejorar el desempeño de la red y reducir su carga.
- Un cliente solicita cada trozo de video de manera independiente, y mantiene el estado de la sesión de reproducción, de modo que los servidores no necesitan rastrear el estado de dicha sesión. El hecho que se mantenga la sesión de reproducción en el cliente significa que este puede recuperar trozos de video desde múltiples servidores con balanceo de carga y tolerancia a fallos entre servidores HTTP.

Estas ventajas permiten que los proveedores de servicios aprovechen la infraestructura HTTP existente, como CDNs y cache proxy.

2.2 Streaming Dinámico Adaptativo sobre HTTP

DASH se ha convertido en la tecnología más popular para el streaming de video sobre internet. La principal ventaja de DASH es que permite que los reproductores multimedia adapten la calidad del video de acuerdo a las condiciones de la red, mientras que el uso de HTTP permite que los proveedores de contenido desplieguen soluciones altamente escalables utilizando CDNs y caches proxy [7].

2.2.1 Arquitectura DASH

La figura 1 muestra la arquitectura DASH, la cual estructura el contenido multimedia de una manera jerárquica. En el nivel superior se presenta un objeto multimedia entero (video) el cual es denominado Presentación [8]. Una descripción de la presentación multimedia (MPD) describe el contenido de una pieza de video dentro de una duración específica, la cual es denominada Período. En un Período, existen múltiples versiones del contenido, las cuales son denominadas Representaciones. La parte baja de la jerarquía está conformada por los segmentos de video. Existen también diversas estructuras utilizadas para seleccionar entre varios segmentos, de modo que se puedan alcanzar los requerimientos de reproducción.

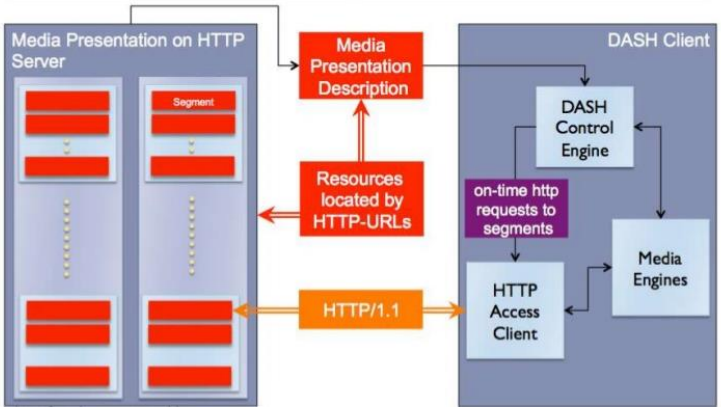

Figura 1. Arquitectura DASH.

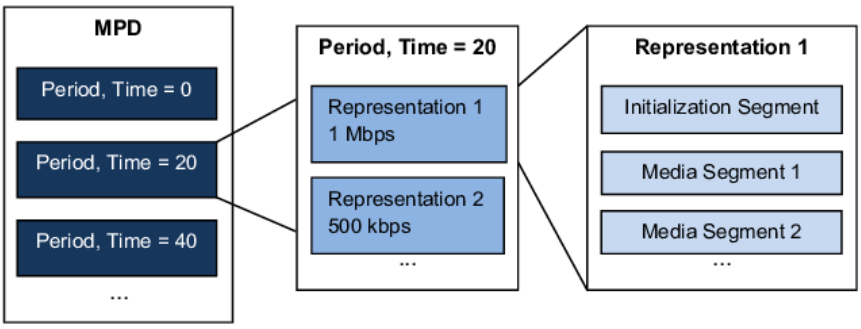

Figura 2. Jerarquización del contenido multimedia en DASH.

Como se muestra en la figura 3, el cliente rápidamente sube su solicitud de bitrate de video durante el inicio para así rellenar su buffer. Cuando el cliente detecta una

reducción en la capacidad del ancho de banda, mediante el uso de señales de feedback de los trozos de video recuperados previamente, solicita una calidad de video más baja. Cuando las capacidades de la red vuelven a aumentar, el reproductor vuelve a solicitar una mejor calidad. Este procedimiento produce como resultado que el cliente reproduzca el video sin problemas, sin tener que sobre provisionar la red o mantener un buffer excesivamente grande.

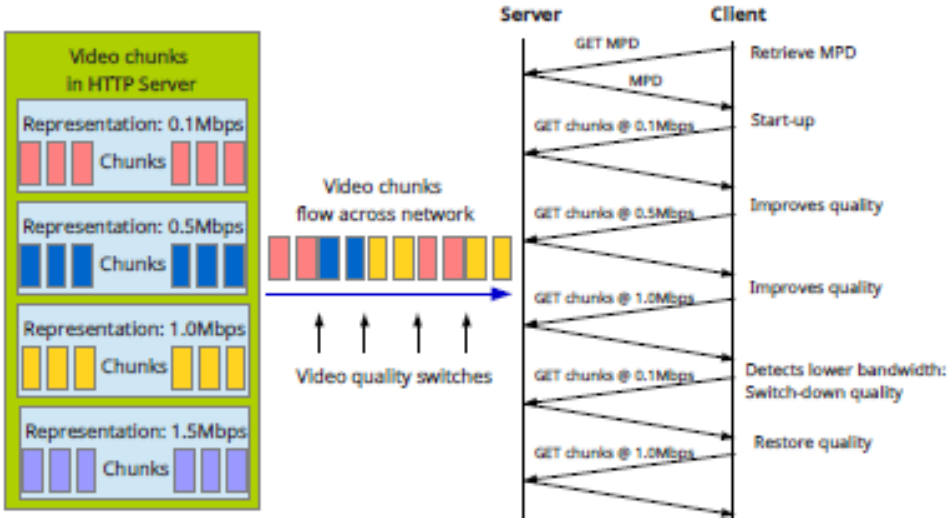

Figura 3. Ilustración del comportamiento adaptativo de DASH

2.3 Aplicando DASH en transmisiones Multicast

Las comunicaciones de red pueden ser clasificadas en tres grupos principales de acuerdo a la cardinalidad de los receptores. El primer grupo son comunicaciones uno a uno, conocidas como unicast, e involucran un emisor y un receptor. El segundo grupo, denominado broadcast, es exactamente opuesto al primero, donde un emisor transmite un mensaje a todos los nodos de la red. En el tercer grupo, el cual se denomina multicast, un emisor transmite un mensaje a un grupo de receptores que se han afiliado previamente a un grupo multicast, tal como se muestra en la figura 4.

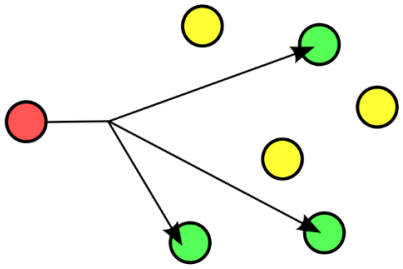

Figura 4. Representación de transmisión de datos en multicast.

Multicast es apto para aplicaciones que sacan provecho de la agrupación lógica de hosts, sobre todo si se requiere transmitir grandes cantidades de datos a múltiples receptores al mismo tiempo. Para este tipo de casos, multicast presenta grandes ventajas en comparación a unicast en términos de ahorro de recursos de red y carga. Los casos de usos más comunes son: video conferencias (donde todos los participantes pertenecen al mismo grupo) y streaming multimedia o IPTV (donde cada grupo multicast representa un canal de TV) [20].

El 3GPP, una iniciativa que reúne a las principales organizaciones del sector de las telecomunicaciones, define como distribuir contenido multimedia sobre redes LTE utilizando DASH. Es importante resaltar que DASH fue originalmente definido para conexiones unicast bidireccionales. Sin embargo, y con el propósito de gestionar de manera más eficiente los recursos de la red, el 3GPP propone que para la entrega de segmentos multimedia a dispositivos móviles se combine DASH con multicast usando eMBMS. En este caso, se crea una sola representación del contenido multimedia, la cual es enviada a todos los dispositivos a través de un canal eMBMS. Por lo tanto, no se utiliza la adaptabilidad de DASH, ya que el receptor no puede escoger entre diferentes representaciones [9].

Considerando que las transmisiones sobre canales eMBMS son propensas a errores, el 3GPP propone el uso de AL-FEC y técnicas de recuperación unicast. Los segmentos multimedia que se transmiten están protegidos con AL-FEC y el cliente utiliza AL-FEC para recuperar los segmentos que se vean afectados por errores en la transmisión. Aquellos segmentos que no puedan ser recuperados mediante esta técnica son recuperados mediante HTTP. Pero como solo existe una representación del contenido, el cliente DASH solicita la misma representación directamente a un servidor web [9].

2.3.1 Protocolos utilizados típicamente en multicast

Existe una amplia gama de protocolos utilizados con diferentes propósitos para las comunicaciones en multicast, siendo alguno de ellos:

RTP

Este protocolo está definido como un estándar IETF que permite conectividad en tiempo real para el intercambio de datos que necesitan prioridad. Su función principal es mover datos entre dos extremos tan eficientemente como sea posible, de acuerdo a las condiciones de la red [34].

RTP presenta una serie de características, entre las que se encuentran:

- Baja latencia.

- Paquetes enumerados secuencialmente y con marcas de tiempo para su reensamblado en caso de que lleguen desordenados.
- No se limita a comunicaciones audiovisuales. Puede ser utilizado para cualquier tipo de transferencia activa de datos.

Sin embargo, RTP no provee QoS ni gestiona la asignación o reserva de recursos necesarios. RTP es utilizado típicamente para la entrega de contenido multimedia (audio y video) en multidifusión.

FLUTE

FLUTE es un protocolo para la entrega unidireccional de ficheros sobre internet. ACL es un protocolo que define el transporte de objetos binarios arbitrarios y que es sumamente apto para la distribución unilateral y altamente escalable en multicast. Sin embargo, para aplicaciones de entrega de ficheros, el transporte de objetos no es suficiente, ya que el sistema final necesita saber que representa cada objeto [43].

FLUTE es un mecanismo de señalización y mapeo de propiedades de ficheros a conceptos de ACL, de forma tal que permite a los receptores asignar esos parámetros a los objetos recibidos. En pocas palabras, ACL provee los mecanismos básicos de transporte que utiliza FLUTE, y por lo tanto, este hereda los requisitos de ACL.

Los ficheros que van a ser transportados mediante el protocolo FLUTE deben cumplir con las siguientes características [33]:

- Identificador del fichero, expresado como una URI. Este identificador puede ser globalmente único. El identificador puede también proveer la ubicación del archivo.
- Nombre del fichero
- Tipo de fichero, expresado como un tipo multimedia MIME. Si se suministra un valor explicito para el tipo de MIME distinto de la extensión del fichero, entonces el valor explícito suministrado debe ser usado como el tipo MIME.
- Tamaño del fichero, expresado en bytes. Si el fichero esta codificado, entonces el tamaño será el observado antes de la codificación de su contenido.
- Propiedades de seguridad del fichero como firmas digitales.

Sin embargo, existen algunos inconvenientes, siendo el principal la poca sensibilidad de algunas redes con los protocolos de control de congestión que suelen ser usados con este protocolo.

ROUTE

Varios organismos de estandarización, incluyendo el Instituto Europeo de Normas de Telecomunicaciones (ETSI), Proyecto Asociación de Tercera Generación (3GPP) y el

Comité de Sistemas de Televisión Avanzada (ATSC) han trabajado para habilitar la entrega de contenido DASH vía broadcast. Aunque previamente se utilizaba el protocolo FLUTE para estos fines, este no fue diseñado para la entrega de contenido en tiempo real que se requiere en las transmisiones difusivas. Para este propósito, ATSC está utilizando el protocolo ROUTE, el cual supera los distintos obstáculos que enfrentaba FLUTE en la entrega de contenido en tiempo real. [25]

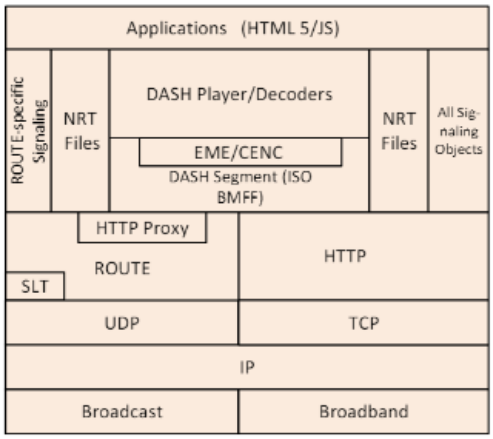

Figura 5. Pila de protocolos de un receptor ATSC

En la figura 5 se muestran las secciones más relevantes de la pila de protocolos para un receptor ATSC 3.0 donde la entrega de contenido DASH a través de ROUTE puede habilitar la entrega de distintos servicios, como solo broadcast, híbrido de broadcast y broadband y solo broadband.

3 Redes Definidas por Software

Las redes de computadores pueden ser divididas en tres planos de funcionalidad: de datos, control y gestión. El plano de datos corresponde a los dispositivos de red que reenvían los datos. El plano de control representa los protocolos usados para poblar las tablas de reenvío de los elementos del plano de datos, mientras que el plano de gestión incluye los servicios de software usados para monitorear y configurar remotamente la funcionalidad de control [1].

Las redes definidas por software son un paradigma de red emergente que ofrece cambiar las limitaciones de las infraestructuras de red actuales. Este paradigma presenta cuatro características fundamentales:

- El plano de control y el plano de datos están desacoplados. Las funcionalidades de control son eliminadas de los dispositivos de red, los cuales se convierten en simples elementos de reenvío de paquetes [30].
- Las decisiones de reenvío están basadas en flujos en lugar de destinos. Un flujo es ampliamente definido por un conjunto de valores de campo de paquetes que actúan como un criterio de cotejo y por un conjunto de acciones. En el contexto SDN, un flujo es una secuencia de paquetes entre un origen y un destino. Todos los paquetes de un flujo reciben las mismas políticas de servicio en los dispositivos de reenvío. La abstracción de flujo permite unificar el comportamiento de diferentes tipos de dispositivos de red, incluyendo routers, switches, firewalls, entre otros.
- La lógica de control es trasladada a una entidad externa, denominada controlador SDN o NOS (sistema operativo de la red, por sus siglas en ingles), que es una plataforma de software que provee recursos esenciales y abstracciones para facilitar la programación de los dispositivos de reenvío basados en una vista de red abstracta y lógicamente centralizada.
- La red es programable a través de aplicaciones de software que se ejecutan sobre el controlador que interactúa con los dispositivos del plano de datos. Esta es la característica más importante de las SDN, y se considera como su principal propuesta de valor.

Es importante resaltar los beneficios que se obtienen de la centralización del control lógico. En primer lugar, modificar políticas de la red es mucho más sencillo y menos propenso a errores a través de lenguajes de alto nivel y componentes de software. Segundo, un programa de control puede reaccionar rápidamente ante cambios en el estado de la red, para mantener las políticas intactas. Por último, la centralización lógica en un controlador que posea conocimiento global del estado de la red simplifica el desarrollo de funciones, servicios y aplicaciones de red más complejas y sofisticadas.

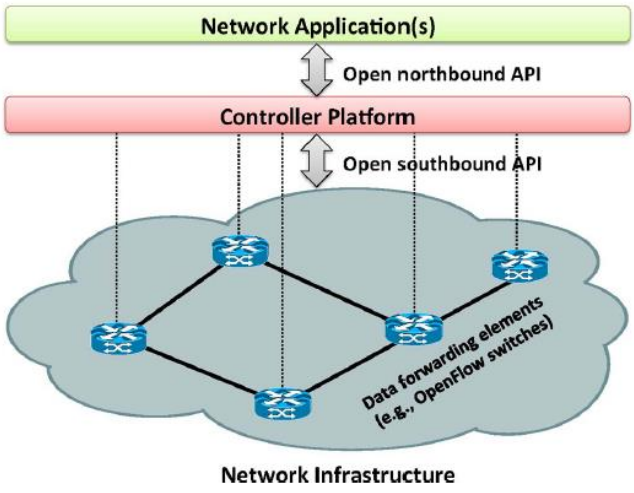

Figura 6. Vista simplificada de la arquitectura SDN

3.1 Terminología básica

Dispositivos de reenvío (FD)

Son dispositivos del plano de datos basados en hardware o software que ejecutan un conjunto de operaciones básicas y que tienen un conjunto de instrucciones bien definidas que se utilizan para decidir las acciones a realizar cuando se reciben paquetes. Estas instrucciones están definidas por interfaces southbound, por ejemplo OpenFlow, y son instaladas en los dispositivos por el controlador SDN implementando los protocolos southboud.

Plano de datos (DP)

Está constituido por los dispositivos de reenvío que están interconectados a través de canales radio o cables. En otras palabras, el plano de datos está representado por la infraestructura de la red.

Interface southbound (SI)

El conjunto de instrucciones de los dispositivos de reenvío está definido por la API southbound, la cual es parte de la interface southbound. Adicionalmente, SI también define los protocolos de comunicación entre los dispositivos de reenvío y los elementos del plano de control.

Plano de control (CP)

Toda la lógica de control reside en las aplicaciones y controladores, por lo tanto, el plano de control puede ser considerado como el "cerebro" de la red. Los dispositivos de reenvío son programados por elementos del plano de control a través de buenas representaciones de SI.

Interface northbound (NI)

El controlador puede ofrecer una API para desarrolladores de aplicaciones. Esta API representa la interface northbound la cual abstrae conjuntos de instrucciones de bajo nivel usados por SI para programar los dispositivos de reenvío.

Plano de gestión (MP)

El plano de control es el conjunto de aplicaciones que toman ventaja de las funciones ofrecidas por NI para implementar control de red y lógica de operación. Esto incluye aplicaciones como routing, firewalls, balanceadores de carga, monitorización, entre otros. Esencialmente, una aplicación de gestión define las políticas, las cuales son traducidas a instrucciones southbound específicas que programan el comportamiento de los dispositivos de reenvío.

3.2 Arquitectura SDN

Tal y como se muestra en la figura 7, la arquitectura SDN puede ser representada como la composición de diferentes capas, cada una de ellas tiene sus funciones específicas. Mientras algunas capas están siempre presentes en un despliegue SDN como la API southbound, el controlador, la API northbound y las aplicaciones de red, otras como el hipervisor están únicamente presentes en despliegues específicos.

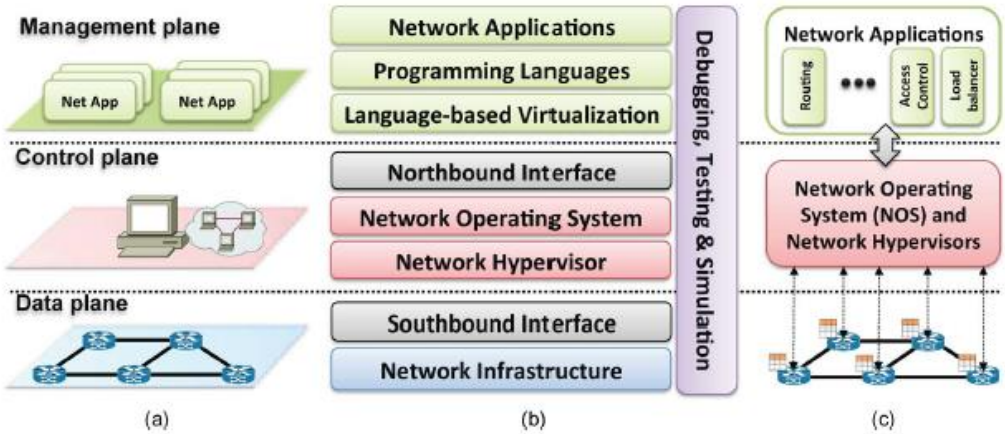

Figura 7. SDN en (a) planos, (b) niveles y (c) arquitectura de diseño del sistema

Capa #1: Infraestructura

Una infraestructura SDN es similar a una red tradicional (compuesta por un conjunto de dispositivos de red como routers, switches, aplicaciones middlebox, entre otros). La principal diferencia reside en el hecho que ahora esos dispositivos físicos tradicionales son simplemente elementos de reenvío sin control embebido o software que tome decisiones autónomas. La inteligencia de la red se traslada de los dispositivos del plano de datos a un sistema de control lógicamente centralizado. Más importante aún, estas nuevas redes están construidas, conceptualmente, sobre interfaces abiertas y estándar, como por ejemplo OpenFlow. Esto es un enfoque crucial para asegurar la compatibilidad

e interoperabilidad de configuración y comunicación entre diferentes dispositivos de los planos de control y de datos.

Capa #2 Interfaces southbound

Las interfaces southbound, también conocidas como API southbound son los puentes entre elementos de control y elementos de reenvío, por lo tanto, son un instrumento crucial para separar las funcionalidades del plano de control y del plano de datos.

Capa #3 Hipervisores de red

La virtualización hoy en día es una tecnología consolidada en la computación moderna. Los principaes requerimientos de una red pueden ser capturados en 2 dimensiones: topología de red y espacio de direcciones. Cada carga de trabajo (workload) requiere diferentes topologías de red, las que pueden ir desde servicios de capa2 o 3, hasta topologías que soporten servicios de L4-L7

Capa #4 Controlador de la red

Los sistemas operativos de red tradicionales proveen abstracción para poder acceder a los dispositivos de bajo nivel, gestionar el acceso concurrente a los recursos subyacentes (disco duro, adaptador de red, CPU, memoria, entre otros) y proveer mecanismos de protección. Estas funcionalidades son clave para incrementar la productividad.

En contraste, hasta ahora las redes han sido gestionadas y configuradas usando conjuntos de instrucciones de bajo nivel específicas para cada dispositivo y sistemas operativos cerrados y que pertenecen a algún propietario, como Cisco, Juniper, etc.

SDN promete facilitar la gestión de la red y alivianar la carga de solventar los problemas de red a través del control lógico centralizado ofrecido por el controlador. Al igual que los sistemas operativos de red, el valor crucial del controlador es proveer abstracción, servicios esenciales y APIs comunes para desarrolladores. Funcionalidades genéricas como estado e información de la topología de la red, descubrimiento de dispositivos y distribución de la configuración de la red pueden ser ofrecidas como servicios del controlador.

El controlador es un elemento sumamente crítico en la arquitectura SDN ya que es el que soporta la lógica de control para generar la configuración de la red basándose en políticas definidas por el operador de la red

Controlador centralizado vs controlador distribuido: un controlador centralizado es una entidad que gestiona todos los dispositivos de reenvío de la red. Naturalmente, un controlador de este tipo puede presentar muchas limitaciones de escalado.

Controladores centralizados, como Floodlight, han sido diseñados para ser sistemas altamente concurrentes para alcanzar el rendimiento deseado.

Contrario a los controladores centralizados, los controladores distribuidos, como ONOS, son escalables pueden abarcar y satisfacer todos los requerimientos de, potencialmente, cualquier entorno. Un controlador distribuido puede ser un cluster centralizado de nodos o un conjunto de elementos físicos distribuidos. Otra propiedad de los controladores distribuidos es la tolerancia a fallos. Cuando un nodo cae, otro se hace responsable tanto de sus funciones como de sus dispositivos

Funciones principales: las funciones de servicio base de la red son consideradas como las funcionalidades esenciales que todo controlador debe proveer. Estos servicios son usados por servicios de otros niveles del sistema operativo y aplicaciones de usuarios. De una forma similar, funciones como la topología, estadísticas, notificaciones, gestión de dispositivos, junto con OSPF y mecanismos de seguridad, son funcionalidades elementales de control de red que las aplicaciones de red pueden usar para construir su lógica.

Southbound: en el nivel más bajo de la plataforma de control, las APIs southbound son vista como una capa de drivers de dispositivos. Estas proveen una interface común para las capas superiores mientras permiten que la plataforma de control puedan usar diferentes APIs southbound y protocolos plug-in para gestionar tanto dispositivos físicos como virtuales, nuevos o existentes. Esto es esencial, por ejemplo, para permitir múltiples protocolos y conectores de gestión de dispositivos. Por lo tanto, en el plano de datos, pueden coexistir una mezcla de dispositivos físicos, virtuales y una variedad de interfaces de dispositivos. Actualmente, la mayoría de los controladores solamente soportan OpenFlow como API southbound.

East and Westbound: APIs eastbound y westbound, son casos especiales de interfaces requeridos para controladores distribuidos. Actualmente, cada controlador implementa sus propias APIs eastbound y westbound. Las funciones de estas interfaces incluyen importar y exportar datos entre controladores, algoritmos para modelos de consistencia de datos y capacidades de monitoreo/notificación.

Similar a las interfaces southbound y northbound, las APIs eastbound y westbound son componentes esenciales de controladores distribuidos para identificar y proveer compatibilidad e interoperabilidad entre diferentes controladores.

Capa #5 Interfaces northbound

Las interfaces northbound y southbound son dos abstracciones elementales del ecosistema de las SDN. Las interfaces southbound ya tienen una propuesta ampliamente aceptada (OpenFlow), pero una interfaz northbound común es aún un asunto abierto.

Los controladores ofrecen una vasta variedad de interfaces northbound, como ad hoc APIs, RESTful APIs, interfaces de programación a múltiples niveles, entre otros. Se espera que una API northbound común emerja a medida que SDN evoluciona.

Capa #6 Virtualización basada en lenguaje

Dos características esenciales de las soluciones de virtualización son la capacidad de expresar modularidad y permitir diferentes niveles de abstracción mientras se garantiza propiedades deseadas, como protección

Capa #8 Aplicaciones de red

Implementan la lógica de control que será traducida en los comandos que serán instalados en el plano de dato, dictando el comportamiento de los dispositivos de reenvío. SDN pueden ser desplegados en entornos de redes tradicionales, desde redes domésticas y empresariales hasta centros de dato y puntos de interconexión de internet. Esta variedad de entornos, ha conllevado a una amplia variedad de aplicaciones de red. Aplicaciones de red existentes ejecutan funcionalidades tradicionales como encaminamiento, balanceo de carga, y refuerzo de políticas de seguridad, pero también explorar enfoques novedosos, como la reducción del consumo energético.

3.3 OpenFlow

OpenFlow es considerado como uno de los primeros estándares de las redes definidas por software (SDN). El concepto original de OpenFlow nació en 2008, en la Universidad de Stanford (Estados Unidos), y definió el protocolo de comunicación en ambientes SDN que le permite a un controlador SDN interactuar directamente con el plano de datos de los dispositivos de red como lo son switches y routers, tanto físicos como virtualizados de modo que permite adaptarse de una manera mucho más óptima a los cambios de requerimientos del negocio [2]. Desde entonces, y hasta la actualidad OpenFlow ha sido gestionado por la Open Networing Foundation (ONF). [2]

Tal como se muestra en la imagen 8, el protocolo OpenFlow puede ser dividido en cuatro componentes, los cuales son [28]:

- Capa de mensajes: es el núcleo del protocolo, y define la estructura y semántica para todos los mensajes. Esta capa tiene la habilidad de construir copiar, comparar, imprimir y manipular mensajes.
- Máquina de estado: define el comportamiento de bajo nivel del protocolo, y es utilizada para describir acciones como negociación, capacidad de descubrimiento, control de flujos, entregas, entre otros.
- Interface del sistema: define como el protocolo interactúa con el mundo exterior. Este componente identifica las interfaces necesarias y opcionales junto con su uso previsto.

- Configuración: prácticamente todos los aspectos del protocolo tienen configuraciones o valores iniciales.
- Modelo de datos: cada switch OpenFlow mantiene un modelo de datos relacionales que contiene los atributos para cada abstracción OpenFlow. Estos atributos pueden describir las capacidades de una abstracción, su estado de configuración o algún conjunto de estadísticas.

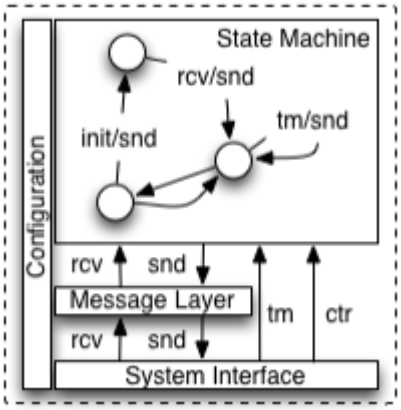

Figura 8. Componentes del protocolo OpenFlow.

Para trabajar en un ambiente OpenFlow, cualquier dispositivo que se quiera comunicar con un controlador SDN debe soportar este protocolo. En la imagen 9 se muestra una representación esquemática de los dispositivos habilitados para OpenFlow. A través de esta interfaz, el controlador agrega los cambios a las tablas de flujos de los switches y routers, permitiendo que los administradores de red puedan particionar el tráfico, controlar los flujos para desempeños óptimos y probar nuevas configuraciones y aplicaciones. [2]

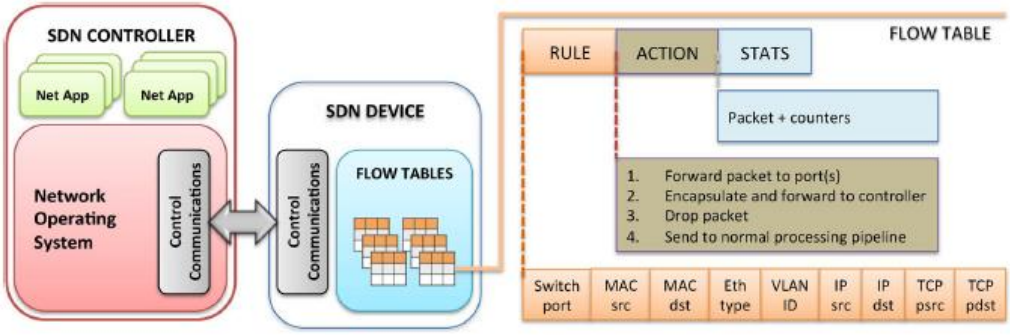

Figura 9. Dispositivos SDN habilitados para OpenFlow.

El uso de OpenFlow presenta una seria de beneficios, entre los que destacan:

- Programabilidad
  - Habilita innovación y diferenciación
  - Acelera la introducción de nuevos servicios y características
- Inteligencia centralizada
  - Aprovisionamiento simplificado

- Desempeño optimizado
- Políticas de gestión granulares

- Abstracción
  - Desacople del plano de control y el plano de datos

3.3.1 Switch OpenFlow

Un switch OpenFlow se puede dividir en dos secciones: agente y plano de datos (figura 10). El agente habla el protocolo OpenFlow con uno o más controladores de red, por lo tanto, traduce los comandos emanados por el controlador en instrucciones de bajo nivel que son enviados al plano de datos mediante el protocolo interno requerido. Por su parte, el plano de datos ejecuta todas las acciones concernientes a la manipulación y reenvío de paquetes. Sin embargo, dependiendo de su configuración, puede enviar los paquetes al agente para que estos sean tratados con mayor detalle.

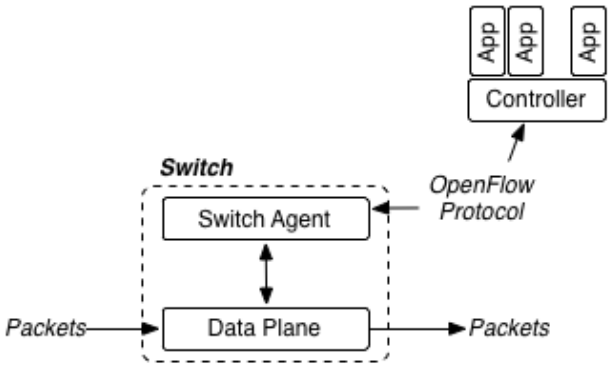

Figura 10. Anatomía de un switch OpenFlow.

El plano de datos de un switch OpenFlow está conformado por puertos, tablas de flujos, flujos, clasificadores y acciones.

- Puertos: los paquetes entran y salen del switch a través de los puertos. De acuerdo a la versión del protocolo, se soportan diferentes tipos de puertos, propiedades y configuraciones.
- Clasificadores: compara un paquete con los registros de la tabla de flujos.
- Acciones: gobiernan la forma en como el paquete es procesado una vez haya coincidido con un registro de la tabla de flujos. Entre las acciones básicas que se pueden ejecutar sobre los flujos se encuentran:
  - Reenviar los flujos a través de un puerto determinado.
  - Reenvío del flujo de paquetes al controlador. Esto solo se ejecuta cuando ese flujo no está en la tabla de flujos, por lo tanto el controlador decide si agregarlo o no
  - Descarte de flujo.
- Tabla de flujos: es el bloque de construcción básico de la arquitectura del switch. Este componente se explica en mayor detalle en la próxima sección

- Flujo: se define como un conjunto de paquetes que comparten las mismas características.

Tabla de flujos

Cada tabla de flujos contiene entradas que están compuestas por seis campos, los cuales son [29]:

- Campos de coincidencias (match fields): son utilizados para seleccionar los paquetes que coincidan con los valores de los campos. Estos campos pueden ser obligatorio u opcionales, entre los que se encuentran:
    - Puerto de ingreso: identifica el puerto del switch por donde llegó un paquete. Puede ser un puerto físico o virtual
    - Origen y destino:
    - Número de protocolo IPv4 o IPv6: es un valor que indica la próxima cabecera en el paquete.
    - Origen y destino IPv4 o IPv6: cada entrada puede ser una dirección exacta, un valor enmascarado, un valor de máscara de subred, o un comodín.
    - Puerto TCP de origen y destino: valor exacto o comodín.
    - Puerto UDP de origen y destino: valor exacto o comodín.
    - Opcionales: puerto físico, metadata, tipo Ethernet, VLAN ID, Etiqueta de flujo IPv6, puerto SCTP de origen y destino, prioridad de usario VLAN, ARP opcode, entre otros.
- Prioridad: prioridad relativa de las entradas de la tabla de flujos
- Contadores: se actualizan por paquetes coincidentes. Las especificaciones de OpenFlow definen una variedad de contadores que incluyen el número de bytes y paquetes recibidos por puerto y por tabla de flujos, el número de paquetes pérdidos, la duración de flujo, entre otros.
- Instrucciones: acciones que se ejecutan cuando ocurre una coincidencia.
- Timeouts: tiempo máximo de espera antes de que el flujo expire
- Cookie: datos opacos seleccionados por el controlador para filtrar estadísticas de un flujo, modificar o eliminar un flujo.

3.3.2 Ciclo de vida de un paquete

Cada paquete experimenta el mismo comportamiento cuando atraviesa un switch OpenFlow (figura 11). Cuando el paquete llega, se crea una llave que contiene información del paquete (valores de algunos campos del protocolo) y algunos metadatos recolectados como puerto de llegada, tiempo de llegada, entre otros. Esta clave es utilizada para seleccionar un flujo (de la tabla de flujos) y aplicar sus acciones asociadas.

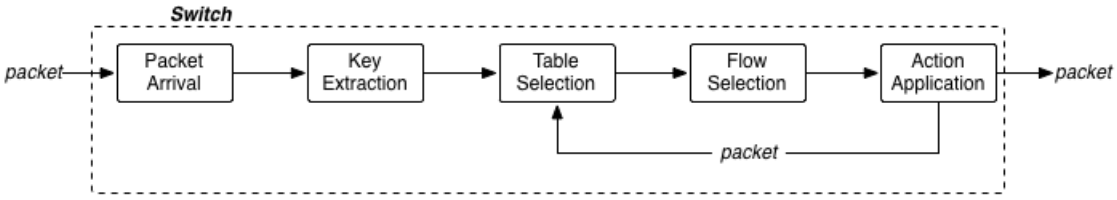

Figura 11. Representación gráfica del ciclo de vida de un paquete.

3.3.3 Mensajes OpenFlow

El protocolo OpenFlow describe el intercambio de mensajes que toma lugar entre un controlador y un switch OpenFlow, y a través de estos permite que el primero realice acciones para agregar, modificar y eliminar entradas en la tabla de flujos. OpenFlow presenta tres tipos de mensajes, los cuales son [29]:

- Controlador a switch: estos mensajes son iniciados por el controlador, y en algunos casos, require una respuesta por parte del switch. Estos mensajes permiten al controlador gestionar el estado lógico del switch, incluyendo su configuración y detalles de las tablas de flujos. Entre los mensajes que conforman esta clase se encuentran:
    - Características: solicita las capacidades de un switch. El swtch responde especificando sus características.
    - Configuración: establece y consulta parámetros de configuración.
    - Modificar estado: agrgar, eliminar y modifcar entradas de las tablas de flujo y establecer propiedades a los puertos del switch.
    - Leer estado: recoge información del switch, como configuración actual, estadísticas y capacidades.
    - Packet-out: direcciona un paquete a un puerto específico del switch.
    - Esta clase también incluye mensajes packet-out, los cuales son utilizados cuando un switch envía un paquete al controlador y este decide no eliminarlo sino que lo direcciona a un puerto específico del switch.
    - Barrera: los mensajes de solicitudes y respuesta de barreras son utilizados por el controlador para asegurar que los mensajes de dependencias se han cumplido o para recibir notificaciones para operaciones completadas.
    - Solicitud de rol: establece o consulta el rol del canal OpenFlow.
    - Configuración asíncrona: establece un filtro a los mensajes asíncronos o consulta dicho filtro.
- Asíncronos: estos mensajes son enviados sin que sean solicitados por el controlador e incluye mensajes de estados. Esta clase está conformada por los siguientes mensajes:
    - Packet-in: transferencia de paquetes al controlador.

    - Flujo eliminado: informar al controlador sobre la eliminación de una entrada en la tabla de flujos.
    - Estado del puerto: informar al controlador sobre un cambio de puertos.
    - Error: notificar al controlador de errores.
- Simétricos: estos mensajes son enviados sin ser solicitados por el controlador o el switch, y son sumamente útiles. Los mensajes que conforman esta clase son:
    - Hello: intercambio entre el switch y el controlador al inicio de la conexión.
    - Echo: los mensajes echo de solicitud y respuesta, pueden ser enviados tanto desde el switch y como desde el controlador.
    - Experimentador: para funciones adicionales.

3.4 Enfoques de sistemas de VoD basado en SDN y multicast

Considerando que la intersección de las tecnologías DASH y SDN es un área sumamente novedosa, no existe un gran número de investigaciones que exploren sus diversos aspectos y beneficios. Uno de los enfoques más importantes que se pueden encontrar hasta la fecha es Multicast definido por Software (SDM) [26], en el cual se diseña un servicio de uno a muchos. Este servicio puede ser utilizado entre las CDN y los ISPs para habilitar multicast de streams de video en vivo OTT en las redes del ISP. Por lo tanto, el ISP ofrece una API de control a la CDN, mediante la cual se informa al ISP sobre los streams de video que van a ser entregados mediante multicast. Este stream es reenviado dentro de la red del ISP utilizando switches SDN.

El controlador SDM, un elemento primordial en esta solución, es responsable de la gestión y mantenimiento de grupo multicast, y determina el árbol multicast correspondiente a la capa de red que debe ser establecido dentro de la red del ISP. Por lo tanto, el árbol con la ruta más corta es computado, y es el que se usará para reenviar los ficheros de video. Una vez computado, el controlador SDM traduce la topología del árbol en el flujo SDN correspondiente y lo reenvía al controlador SDN. Estas entradas son las responsables de reenviar el tráfico a los grupos multicast en el plano de datos de la red. Si se añade un nuevo cliente a un grupo multicast existente, el controlador SDM actualiza el árbol correspondiente y mediante una notificación al controlador SDN, dispara los cambios necesarios en el registro del flujo [19].

Este enfoque, el cual ha sido implementado como un prototipo y ha sido ampliamente evaluado, ha mostrado flexibilidad, eficiencia ante tráfico considerablemente alto, muy buena escalabilidad y una habilidad importante para balancear la carga dentro de las redes de los ISPs.

3.5 Controladores SDN

El controlador SDN es el "cerebro" de una red definida por software. En concreto, es la aplicación que actúa como punto de control estratégico en una SDN, gestiona el flujo de control hacia los routers/switches a través de la southbound API y gestiona las aplicaciones y lógica del negocio superiores a través de la northbound API para desplegar redes inteligentes. [10]

En la actualidad existen diversos controladores SDN, con características diferentes. Entre los más populares se encuentran:

3.5.1 ONOS

ONOS es el primer sistema operativo de código libre que apunta específicamente a los proveedores de servicios y redes críticas. El propósito de este sistema operativo es proveer alta disponibilidad, escalabilidad y desempeño acorde a los requerimientos de la red. Adicionalmente ONOS cuenta con abstracciones y APIs northbound para muy útiles para facilitar el desarrollo de aplicaciones, y abstracciones e interfaces southbound para permitir el control de dispositivos Legacy y OpenFlow [3]

Arquitectura

La arquitectura de ONOS fue diseñada basada en los siguientes principios: [4]

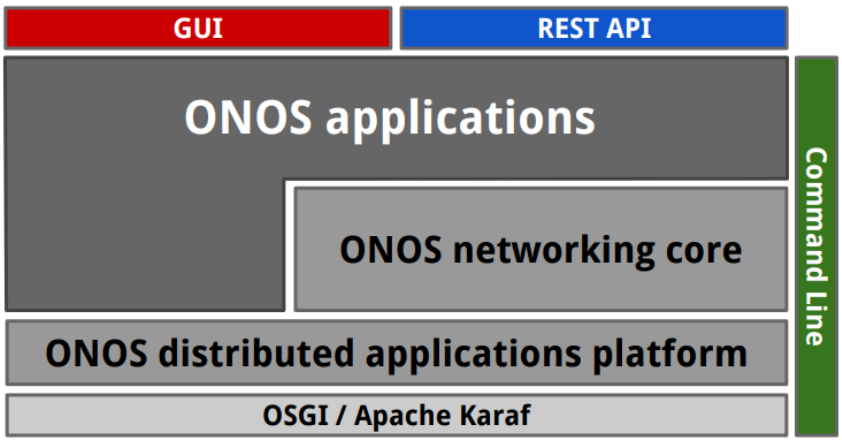

Figura 12. Arquitectura del controlador SDN ONOS [5].

Núcleo distribuido

Provee escalabilidad, alta disponibilidad y desempeño. Para garantizar escalabilidad, ONOS se despliega como un servicio en un clúster de servidores, y el mismo software se ejecuta en cada servidor. Es precisamente esta característica la forma en la que se brinda agilidad estilo web al plano de control SDN y a las redes de proveedores de servicios. Los operadores de red pueden añadir servidores incrementalmente, sin interrupción alguna. Aplicaciones y dispositivos de red no tienes que saber si están trabajando en un solo servidor, o en un conjunto de servidores.

En lo que a la disponibilidad se refiere, esta se logra gracias al despliegue simétrico, que es un principio de diseño sumamente importante y es el responsable de proporcionar recuperaciones rápidas en caso de fallos. Mediante intercambio de mensajes que siguen el modelo "publish/subscribe" las instancias pueden informar y ser informadas, de manera muy rápida, sobre el estados de las demás instancias. El núcleo distribuido también provee servicios de elección de líder.

APIs Northbound:

Incluye dos abstracciones muy poderosas. La primera de ellas, es el Intent Framework, el cual permite a una aplicación solicitar un determinado servicio de la red sin tener que saber los detalles de cómo se ejecutara dicho servicio. Esto permite que los operadores de red y los desarrolladores de aplicaciones puedan programar la red. Este framework toma todas las solicitudes de todas las aplicaciones, resuelve cuales no pueden ser acomodadas, resuelve los conflictos entre aplicaciones, aplica las políticas que correspondan y entrega los servicios solicitados a la aplicación. Por su parte, la vista global de la red, que es la segunda abstracción, provee una vista de la red (host, switches, enlaces, etc) a la aplicación, que, a través de APIs, pueda observar esta vista como un grafo de red, permitiendo así su programación.

APIs Southbound

Permite el funcionamiento de protocolos "enchufables" que controlan tant dispositivos OpenFlow como Legacy. Las abstracciones Southbound de ONOS representan cada elemento de la red como un objeto en una forma genérica. A través de esta abstracción, el núcleo distribuido puede mantener el estado de los elementos de la red sin tener que saber sus especificaciones. La abstracción de los elementos de la red también permite la adición de nuevos protocolos y dispositivos.

Modularidad

Se refiere a como el software ha sido estructurado en componentes, y como esos componentes se relacionan entre sí [3].

Últimas versiones

Tabla 1. Versiones del controlador ONOS [6].

Nombre	Versión	Fecha
Nightgale	1.13.1	Mayo, 2018
Magpie	1.12.0	Diciembre, 2017
Loon	1.11.2	Marzo, 2018
Kingfisher	1.10.4	Agosto, 2017
Junco	1.9.2	Junio, 2017

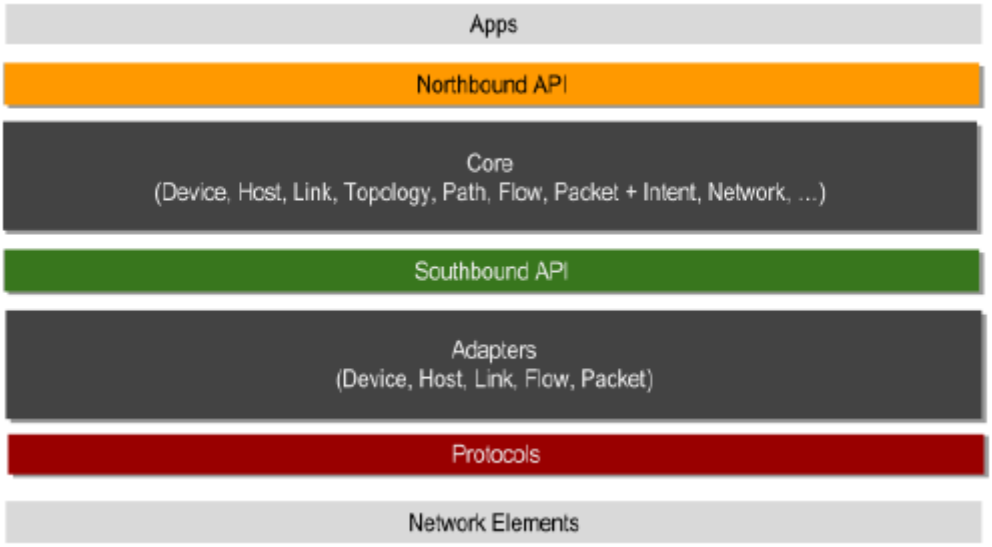

Figura 13. Capas del controlador SDN ONOS [5].

3.5.2 Open Daylight

Es una plataforma modular abierta utilizada para personalizar y automatizar redes de cualquier tamaño y escala. El proyecto OpenDaylight surgió del movimiento SDN, enfocado en la programabilidad de las redes, y presenta las siguientes características [11]:

- Orientado a modelo: el núcleo de la plataforma ODL es la capa de abstracción de servicio orientado a modelo (MD-SAL). En esta plataforma, los dispositivos y las aplicaciones de red son representados como objetos o modelos, cuyas interacciones son procesadas en el SAL.
- Modular y multiprotocolo: la plataforma ODL está diseñada para permitir tanto a usuarios como a proveedores de soluciones máxima flexibilidad en construir un controlador acorde a sus necesidades. El diseño modular de ODL permite que cualquiera, en el ecosistema ODL, pueda aprovechar los servicios creados por otros. Adicionalmente, ODL soporta una amplia variedad de protocolos de comunicación.
- Seguridad, escalabilidad, estabilidad y desempeño: la comunidad ODL provee mejoras continuas en todos sus proyectos en las áreas de seguridad, escalabilidad, estabilidad y desempeño.

Al momento del desarrollo de esta memoria, ODL se encontraba en su octavo release, denominado Oxygen, el cual cuenta con un plug-in P4 para la plataforma, lo que aumenta su valor para segmentos específicos del mercado, como lo son los grandes proveedores de cloud. Adicionalmente, esta versión del controlador presenta otras características, entre las que se encuentran [40]:

- Contenedores: el proyecto del motor de orquestación de contenedores incluye un plug-in para Kubernetes y extensiones Northbound para entornos mixtos de MV-contenedores.
- Armonización: para soportar este enfoque hacia la armonización, los desarrolladores de ODL han aumentado la eficiencia de los releases a través de mejoras en la arquitectura y en los procesos. Uno de los primeros pasos fue la implementación de Karaf 4 en Nitrogen, la versión previa a Oxygen. Con Oxygen, se empieza una transición hacia un modelo de distribución gestionada, el cual desacopla proyectos que no tienen impacto sobre el núcleo del proyecto, permitiendo que estos evolucionen a su propio ritmo.

El controlador ODL está escrito en Java y se basa tecnologías como OSGI (framework de backend), Karaf y YANG(lenguaje de modelado de datos). Este controlador provee una serie subsistemas orientados a modelo como la base para aplicaciones Java. Entre estos subsistemas se encuentran [41]:

- Configuración de subsistemas: es un framework de configuración, inyección de dependencias y activación que permite commitear configuraciones e inyecciones de dependencias, y permite reconexión en tiempo de ejecución.
- MD-SAL: funcionalidad de mensajería y almacenamiento para datos y notificaciones modeladas por desarrolladores de aplicaciones. Esta usa YANG como modelador para la interfaz y definición de datos, y provee tiempo de ejecución para servicios centrados en mensajes y datos.
- MD-SAL Clustering: habilita soporte para la funcionalidad principal de MD-SAL y provee acceso transparente a datos modelados con YANG.

El controlador ODL también permite el acceso a aplicaciones y datos externos utilizando:

- NETCONF: protocol RPC basado en XML que proporciona habilidades al cliente para invocar RPCs modelados con YANG, recibir notificaciones y leer, modificar y manipular data manipulada con YANG.
- RESTCONF: protocolo basado en HTTP que provee APIs para manipular data modelada con YANG e invocar RPCs modelados en YANG utilizando los formatos JSON o XML.

La figura 14 muestra la estructura del controlador de ODL. En la interfaz Southbound ODL soporta múltiples protocolos, comenzando con un plug-in de OpenFlow 1.0. Los módulos se conectan dinámicamente en la capa de abstracción de servicios (SAL), la cual resuelve como entregar el servicio solicitado indiferentemente del protocolo utilizado

entre el controlador y los dispositivos de red. Esto ofrece protección a las aplicaciones, a medida que los protocolos evolucionan en el tiempo.

Para que el controlador pueda controlar los dispositivos bajo su dominio necesita saber las capacidades, alcanzabilidad y otras características de los dispositivos. Toda esta información es almacenada y gestionada por el Gestor de Topología. Otros componentes como el gestor ARP, el rastreador de hosts, gestor de dispositivos y gestor de switches ayudan a nutrir la base de datos de la topología para el Gestor de Topología.

Por otra parte, el controlador expone una API Northbound abierta que es utilizada por aplicaciones. ODL soporta el framework OSGi y REST bidireccional. La lógica de negocio reside en las aplicaciones, que utilizan en controlador para recolectar inteligencia de la red, ejecutar algoritmos que hacen análisis y para orquestar reglas en la red.

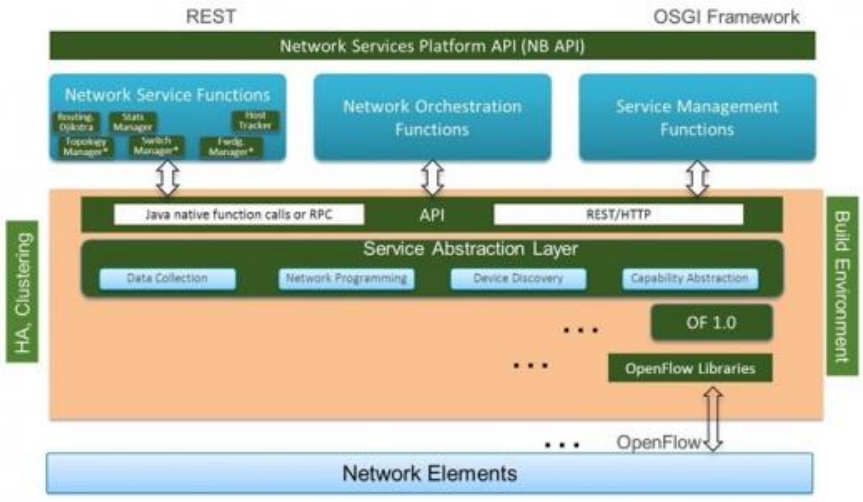

Figura 14. Estructura del controlador SDN ODL.

3.5.2.1.1 RYU

Es un framework SDN, que provee componentes software con APIs bien definidas para facilitar a desarrolladores la creación de nuevas aplicaciones de gestión y control de red. RYU soporta diversos protocolos para la gestión de redes, como OpenFlow en sus versiones 1.0, 1.2, 1.3, 1.4, 1.5, y su código está disponible bajo licencia Apache 2.0 [12].

En Ryu existen distintas definiciones básicas. Las aplicaciones son entidades que implementan diversas funcionalidades. Estas aplicaciones se envían distintos tipos de mensajes entre ellas, los cuales son denominados eventos. Cada aplicación tiene una cola FIFO para recibir eventos, por lo tanto estos son procesados de acuerdo al orden de llegada. [32]

Ryu está conformado por los siguientes elementos [31]:

Ejecutables

- Ryu-manager: el ejecutable principal.

Componentes básicos

- Ryu.base.app_manager: el gestor central de las aplicaciones Ryu. Entre sus funciones principales están cargar las aplicaciones Ryu, proveer contexto a las aplicaciones y enrutar mensajes entre dichas aplicaciones.

Controlador OpenFlow

- Ryu.controller.controller: es el componente principal del controlador OpenFlow. Este se encarga de gestionar conexiones entre switches, y generar y enrutar eventos a las entidades correspondientes.
- Ryu.controller.dpset: gestiona los switches.
- Ryu.controller.ofp_event: define los eventos OpenFlow.
- Ryu.controller.ofp_handler: gestión OpenFlow básica, incluyendo negociación.

Codificador y decodificador del protocolo OpenFlow

Formado por elementos que incluyen las definiciones, codificadores y decodificadores de las implementaciones de las distintas versiones del protocolo OpenFlow soportado por Ryu.

3.5.3 FloodLight

Es un controlador SDN diseñado por Big Switch Networks, que trabaja con el protocolo OpenFlow para orquestar flujos de tráfico en un entorno SDN. Este controlador está escrito en Java e incluye REST APIs para permitir a los desarrolladores adaptar software y desarrollar nuevas aplicaciones.

Floodlight es un controlador OpenFlow y una colección de aplicaciones construidas sobre el controlador. El controlador ejecuta un conjunto de funcionalidades básicas para controlar una red OpenFlow, mientras que las aplicaciones ejecutan diversas funciones para solventar las diferentes necesidades de los usuarios sobre la red. La figura 15 muestra la arquitectura de Floodlight, y se puede apreciar la relación entre el controlador, las aplicaciones construidas como módulos Java y las aplicaciones construidas sobre la REST API [37].

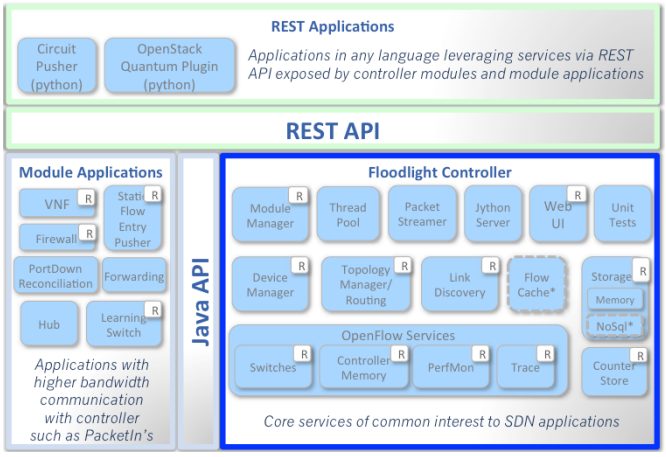

Figura 15. Arquitectura del controlador Floodlight

Cuando se ejecuta Floodlight, se inician tanto el controlador, como las aplicaciones Java que se hayan cargado en el fichero de propiedades. La API REST es expuesta por todos los módulos que se están ejecutando a través del puerto REST. Cualquier aplicación REST, indiferentemente del lenguaje en el que esté escrita, puede recuperar información e invocar servicios mediante el envío de comandos HTTP al puerto REST del controlador.

Para el momento del desarrollo de esta memoria, Floodlight contaba con seis versiones, siendo la más reciente la versión 1.2. Las versiones antiguas de este controlador son la 0.85, 0.90, 0.91, 1.0 y 1.1

3.6 Selección del controlador SDN

En los últimos años se han desarrollado diversos controladores open source. Estos controladores fueron diseñados para demostrar el potencial de SDN y no para ser base de un producto comercial, por lo tanto, no presentan características como escalabilidad, disponibilidad y alto desempeño [3]. Para seleccionar el controlador a utilizar para el desarrollo de este Trabajo Fin de Máster se tomó en cuenta tanto las características principales de cada controlador SDN así como las posibilidades que estos ofrecen para la transmisión de paquetes en multicast.

3.6.1 Caracteristicas

La tabla 2 contiene, de manera resumida, las características principales de los controladores descritos anteriormente. Estas características, entre las que destacan el lenguaje en el que se desarrolló el controlador, la consistencia, entre otras, serán de suma importancia al momento de seleccionar el controlador a utilizar.

Tabla 2. Comparativa de los controladores SDN más populares [1].

Nombre	Arquitectura	API northbound	Consistencia	Tolerancia a fallos	Escrito en
Floodlight	Centralizada	RESTfull API	No	No	Java
ONOS	Distribuido	RESTful API	Débil, Fuerte	Si	Java
Ryu	Centralizado	Ad-hoc API	No	No	Python
ODL	Distribuido	REST, RESTCONF	Débil	No	Java

3.6.2 Posibilidades de multicast que ofrecen los controladores SDN

ONOS

ONOS cuenta con una amplia gama de aplicaciones northbound que implementan distintas funcionalidades y protocolos de red. Entre estas se encuentra la aplicación Multicast Forwarding.

Multicast Forwarding

Este módulo presenta una arquitectura que está compuesta por cuatro funcionalidades primarias, las cuales son:

- **Tabla de rutas multicast (MRT):** es, esencialmente, el repositorio de las rutas multicast mantenidas por ONOS. Las rutas multicast están divididas en dos categorías, las cuales son:
    - Multicast desde cualquier origen (ASM)
    - Multicast desde un origen específico (SSM).

  MRT mantiene un conjunto de grupos McastRouteGroup correspondiente para cada grupo IPv4 e IPv6 para el cual MFWD mantiene estado. El McastRouteGroup tiene dos propósitos: mantener el estado de reenvío para ASM y servir de contenedor padre para todas las rutas McastRouteSources que comparten la misma dirección de grupo multicast [17].

  Los ConnectPoints son construcciones de ONOS que consisten de un nombre y número de puerto de switch. Para el propósito de multicast, los ConnectPoints se utilizan para determinar por donde entra la data multicast a la red SDN, así como también el conjunto de ConnectPoints por donde estos salen de la red.

- **Gestor de los Intents multicast:** este elemento es responsable de registrar, formular y cambiar en el IntentService de ONOS intents basado en el estado específico de McastRoute. Otra de sus funciones es registrar un

SinglePointToMultiPointIntent para cada ruta multicast que es utilizada para renviar estado multicast. También es responsable de liberar y limpiar intents después de que la aplicación MFWD es desactivada.

- **Multicast Forwarding:** es el encargado de manejar la data multicast en vivo. Para los paquetes multicast entrantes que no coincidan con una entrada de reenvío multicast, el modulo creará una entrada de reenvío con un ConnectPoint de ingreso pero sin uno de egreso. El estado estará listo cuando y si los receptores indican interés. Cuando data multicast es recibida, y si se tiene una regla que coincida, y si esa entrada tiene uno o más ConnectPoints de salida, la entrada es completada con el ConnectPoint de ingreso y es entregado al MulticastIntentManager, y por lo tanto, crea un SinglePointToMultiPointIntent que es usado posteriormente para instalar el flujo correspondiente en los switches relevante. Si un estado de reenvío multicast no existe, el McastForwarding creará el estado multicast.
- **MFWD CLI:** permite que operadores y aplicaciones externas examinen y modifiquen el estado mfwd.

Esta aplicación permite realizar tres acciones básicas, las cuales son:

- Añadir: mediante el commando *mcast-join* se instala un flujo <Fuente, grupo multicast>. Se debe especificar la dirección IP de la fuente (servidor), la IP del grupo multicst, el puerto de ingreso para cada miembro del grupo multicast, y el puerto de egreso del servidor
- Eliminar: permite eliminar un flujo multicast mediante el comando *mcast-delete*, pasándole los mismos parámetros mencionados en el comando *mcast-join*.
- Mostrar: con el comando *mcast-show* se pueden mostrar las rutas multicast instaladas en formato json.

OpenDaylight

El controlador OpenDaylight soporta multicast IPv4, ya que entre todos los protocolos que emplea se encuentra IGMP. Para realizar multicast se deben realizar una serie de actividades, entre las que se encuentran:

- Reenviar todas las solicitudes IGMP al controlador, de modo que este pueda saber cuáles son los host que desean unirse al grupo multicast. Para esto se debe crear un conjunto de reglas en el controlador.

- Seguidamente, todas las direcciones en el rango multicast deben ser reenviadas al controlador. De esta manera, el controlador podrá saber cuál es la fuente del tráfico multicast.
- Finalmente, se debe crear un grupo OpenFlow por cada grupo multicast conocido por el controlador, y añadir los puertos por donde se recibe el tráfico IGMP.

ODL ofrece diferentes acciones que se pueden ejecutar para gestionar los grupos OpenFlow en los switches. Estas acciones son agregar, modificar y eliminar grupos. Adicionalmente, es posible chequear el estado de un grupo bien sea directamente en el switch o en el controlador a través de RESTCONF y buscar estadística de los grupos a en el controlador mediante el uso de RESTCONF [42].

Figura 16. Estructura del comando para la creación de grupo multicast en ONOS.

RYU

RYU cuenta con librerías que implementan las distintas versiones del protocolo IGMP. Una instancia de librería ryu.lib.packet.igmp.igmp_v3 tiene, por lo menos, los siguientes atributos [36]:

- Msgtype: un tipo de mensaje para v2, o una combinación de versiones y un tipo de mensaje para v1.
- Maxresp: máximo tiempo de respuesta en décimas de segundos.
- Csum: un valor de checksum. Cero significa que se calcula automáticamente cuando se codifica.
- Address: la dirección IP de un grupo.
- S_flag: cuando se setea a 1, los routers eliminan el timer del proceso
- Qrv: variable de robustez para una consulta
- Qqic: un intervalo de tiempo para una consulta, en segundos.
- Num: el número de servidores multicast
- Srcs: una lista de direcciones IPv4 de los servidores multicast.

Esta librería cuenta con dos métodos, cada uno con funciones específicas:

Parser(buf)

Este método se usa cuando se va a decodificar un paquete. Decodifica la cabecera de un protocolo con compensación cero en un bytearray y devulve los siguientes objetos:

- Un objeto para describer la cabecera decoficada.
- Una subclase pacjet_base.PacketBase apropiada para el resto del paquete. Ninguno cuando se considera que el resto del paquete carga bruta.
- El resto del paquete.

Serializable(payload, prev)

Este método solamente se utiliza cuando se quiere codificar un paquete, y su funcionamiento consiste en codificar la cabecera del protocolo y devolver un bytearray que contiene dicha cabecera. Payload es el resto del paquete, el cual seguirá inmediatamente a la cabecera, mientras que prev es una subclase packet_base.PacketBase para la cabecera del protocolo externo.

Floodlight

Floodlight maneja multicast por defecto desde el plano de control, sin embargo, su rendimiento no es óptimo para el streaming de video. Para este fin, es necesario utilizar un módulo Java que cree árboles de reenvío para enviar los streams al destino deseado.

Si bien no existe un módulo Java que sea recomendado en la documentación de este controlador, si existen muchas aplicaciones creadas por desarrolladores que forman parte de la comunidad Floodlight. En [38] y [39] se presentan dos soluciones que implementa multicast IPv4, aunque la documentación de ambas soluciones es prácticamente inexistente, lo cual es contraproducente para la realización de este proyecto.

Considerando las características de los controladores SDN resumidas en la Tabla 2, y las aplicaciones, herramientas y funcionalidades que cada uno de estos ofrece para manipular y controlar tráfico multicast, se decantó por utilizar el controlador ONOS para el desarrollo de este trabajo, ya que es un controlador distribuido, código libre y que presenta tolerancia a fallos. Adicionalmente, ONOS cuenta con una amplia gama de aplicaciones northbound, que, combinadas con la RESTFul API, facilita su uso para distintos casos. A pesar de que la versión 1.13.1 fue lanzada a mediados del año 2018, se decidió utilizar la versión 1.12, que presentaba mayor estabilidad al momento del desarrollo de este proyecto.

4 Diseño y construcción del escenario

En esta sección se describen las herramientas utilizadas durante el diseño y la construcción del escenario de red sobre el cual se realizó este proyecto, los componentes más importantes de la topología diseñada y las pruebas realizadas para validar su correcto funcionamiento.

4.1 Herramientas utilizadas

Las herramientas que se utilizaron para el diseño y la construcción del escenario de red sobre el cual se desarrolló este proyecto son las siguientes:

4.1.1 Redes Virtuales sobre Linux (VNX)

Es una herramienta de virtualización de propósito general, de código abierto, desarrollada por el departamento de Ingeniería Telemática de la Universidad Politécnica de Madrid (DIT-UPM), diseñada para construir bancos de pruebas de redes virtuales de manera automática. Esta herramienta permite la definición y despliegue automático de escenarios de red compuestos de diferentes tipos de máquinas virtuales (Linux, Windows, Dynamips routers, etc) interconectadas de acuerdo a la topología diseñada por el usuario, con la posibilidad de conectar estos escenarios a redes externas [21].

VNX está compuesta por dos partes principales:

- Un lenguaje XML que permite la descripción de escenarios de red virtuales
- El programa VNX, el cual se encarga de analizar la descripción del escenario y construirlo sobre una máquina Linux.

VNX presenta funcionalidades muy útiles, entre las que destacan:

- Integración de nuevas plataformas de virtualización para permitir que las máquinas virtuales corran sistemas operativos distintos de Linux.
- Usa *libvirt* para interactuar con las capacidades de virtualización del anfitrión, lo que permite el uso de un variado número de plataformas de virtualización (KVM, XEN, etc.)
- Integra plataformas de virtualización de routers como *Dynamips* y *Olive* para permitir emulación límitada de routers CISCO y Juniper.
- Integra soporte a contenedores Linux (LXC).
- Gestión individual de máquinas virtuales.
- Autoconfiguración y capacidades de ejecución de comandos para varios sistemas operativos.
- Integración de Openvswitch con soporte para configuración de VLAS, conexiones inter switches y configuración de parámetros SDN.

4.1.2 Servidor Web Apache

El servidor HTTP Apache es un desarrollo de software colaborativo que apunta a crear una implementación robusta, funcional, de grado comercial y completamente disponible de un código fuente de un servidor HTTP. Este proyecto es parte de la Apache Software Foundation [15].

4.1.3 Iperf

Es una herramienta diseñada para realizar mediciones activas del ancho de banda máximo alcanzable en redes IP. Soporta el ajuste de diversos parámetros relacionados con timing, buffers y protocolos. En cada test realizado se reporta el ancho de banda alcanzado, pérdida de paquetes y otras estadísticas [16].

4.1.4 Aplicaciones del controlador ONOS

OpenFlow Base

Provee el conjunto básico de dispositivos y flujos de paquetes que se basan en el protocolo OpenFlow para interactuar con dispositivos de red.

OpenFlow

Aplicación de ONOS que provee el conjunto básico de proveedores OpenFlow junto con proveedores de ubicación de hosts ARP/NDP y proveedores de enlaces LLDP.

Forwarding

Aplicación perteneciente a la categoría de gobierno de tráfico, responsable de proveer tráfico entre hosts utilizando programación de flujos salto a salto mediante la intercepción de paquetes para los que no hay un flujo objetivo que concuerde en el plano de datos. Para que la aplicación *Multicast Forwarding* pueda funcionar correctamente, es necesario que esta aplicación también sea implementada.

4.1.5 Karaf

Karaf es un contenedor polimórfico, ligero y poderoso que pertenece al proyecto Apache. Puede ser utilizado como un contenedor stand alone y soporta un amplio rango de tecnologías, como cloud, imágenes dockers, etc. Karaf puede ser escalado desde un contenedor muy ligero hasta un servicio empresarial complejo. Entre sus características se encuentran [18]:

- Despliegue: permite el drag and drop de ficheros en el directorio de despliegue. Karaf automáticamente detectará el tipo de archivo e intentará desplegarlo.
- Consola completa: provee una consola muy similar a la de UNIX desde donde se puede gestionar por completo el contenedor.
- Configuración dinámica: provee un conjunto de comandos para gestionar su propia configuración. Todos los ficheros de configuración están centralizados en

la carpeta etc. Cualquier cambio en el archivo de configuración es detectado y recargado.

- Instancias: múltiples instancias de Karaf pueden ser gestionadas directamente desde una instancia principal.
- Remoto: contiene un servidor SSHd que le permite al usuario acceder a la consola de manera remota. La capa de gestión también se puede acceder de esta manera.

4.1.6 GPAC

Es una implementación de los sistemas MPEG-4 que provee las siguientes características [22]:

- Empaquetado de contenido multimedia: GPAC cuenta con codificadores y multiplexores, herramientas para la publicación y distribución de ficheros MP4, 3GPP o 3GPP2.
- Reproducción de contenido multimedia: soporta una amplia variedad de protocolos y estándares para la reproducción de contenido multimedia.
- Streaming de contenido multimedia.

4.1.7 FFMPEG

Es un framework multimedia, capaz de decodificar, codificar, transcodificar, multiplexar, desmultiplexar, filtrar y reproducir una amplia variedad de formatos multimedia. FFMPEG contiene las librerías libavcodec, libavutil, libavformat, libavfilter, libavdevice, libswscale y libswresample. [23]

4.1.8 Dash.js

Es una iniciativa del foro industrial DASH para establecer un framework de calidad de producción para la construcción de reproductores de video y audio que reproduzcan contenido MPEG-DASH usando librerías JavaScript del lado del cliente y aprovechando las ventajas que ofrecen las librerías de la API Media Source Extensions definidas por W3C. Todo el código de este proyecto está publicado bajo licencia BSD-3. [24]

4.1.9 UFTP

UFTP es un programa que se encarga de transferir ficheros en multicast encriptado, y fue diseñado para enviar archivos simultáneamente a múltiples receptores de manera segura, eficiente y confiable. El funcionamiento de este software se puede resumir en tres fases, las cuales son [44]:

- Fase de Anuncio/Registro: configura la sesión de transferencia multicast de ficheros y negocia todos los parámetros de encriptado. El servidor envía un anuncio sobre una IP multicast pública que los clientes están escuchando. Los clientes habilitados envían un registro para responder a dicho anuncio.

Seguidamente el servidor enviara mensajes de confirmación si el encriptado está deshabilitado, o las claves de encriptado si esta es habilitada.

- Fase de transferencia de ficheros: comienza con la fase de información del fichero a ser enviado, donde el servidor envía un mensaje describiendo dicho fichero. Esta descripción incluye el nombre del fichero, el tamaño y la forma en cómo se va a dividir en bloques. Una vez finalizada esta fase, comienza la fase de transferencia de datos, donde cada bloque de datos es enviado por el servidor. Considerando que UDP no garantiza que los bloques de datos lleguen en orden, cada uno de estos está enumerado para que el cliente pueda reemzamblarlos. Cuando el servidor envía todos los bloques, envía un mensaje al cliente indicando esto.
- Fase de culminación/confirmación: cierra la sesión entre cliente y servidor. Comienza con un mensaje enviado por el servidor donde se indica el final de la sesión. El cliente responde a este mensaje y el servidor finalmente lo confirma.

4.1.10 CORS

Es un mecanismo que utiliza encabezados HTTP adicionales para permitir que un cliente, en un domino distinto, pueda acceder a determinados recursos de un servidor [45]. Para el desarrollo de este proyecto fue necesario utilizar un plug-in de CORS para Google Chrome ya que, por razones de seguridad, los exploradores restringen las peticiones HTTP de origen cruzado iniciadas dentro de un script.

4.2 Descripción de los componentes

En la imagen 17 se puede observar el escenario construido en VNX. Todos los elementos son máquinas virtuales y cada una de ellas cumple con una función específica. Para ello, fue necesario crear un root filesystem, a través de un script [47], que incluye todos los servicios y software necesario para simular un escenario del mundo real.

Los componentes de este escenario virtual son:

4.2.1 Controlador de la red

Tal como se señaló en secciones anteriores, el controlador SDN seleccionado fue la versión 1.12 de ONOS, el cual se ejecuta en una máquina virtual de tipo LXC con arquitectura x86_64, y cuyo nombre en el escenario virtual es “ONOS”. Esta máquina virtual cuenta con un sistema operativo Linux Ubuntu 18.04 y, en a través de su interfaz de red, está conectada a la red “netmgt”, que es una red de gestión.

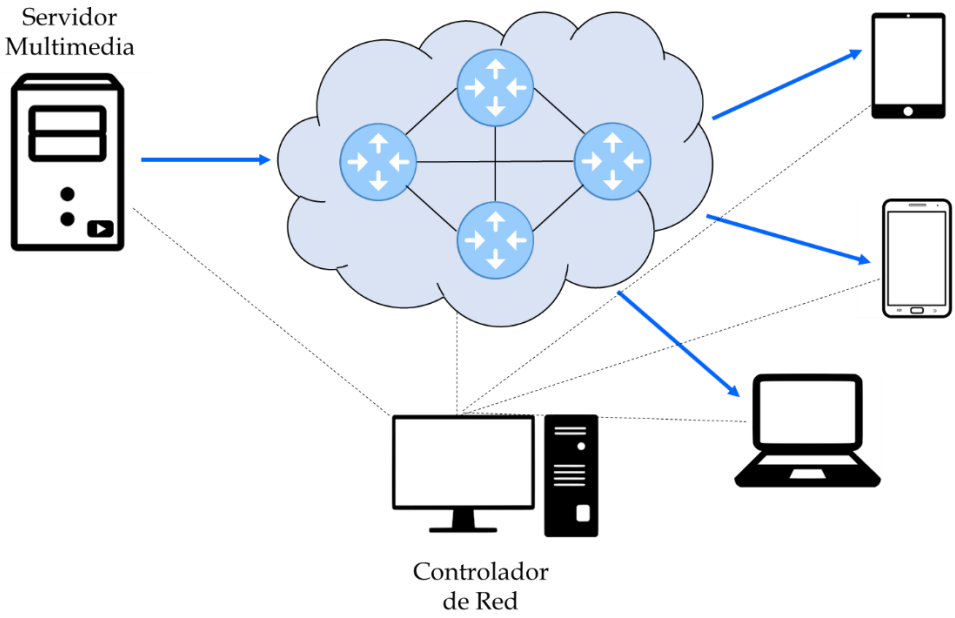

Figura 17. Escenario de red

Como se muestra en la figura 18, para la configuración del controlador se utilizaron distintos ficheros. En primer lugar, con el archivo "network-cfg_ipv4.json" se configura la aplicación *reactive routing*, donde se crean las distintas subredes del escenario, y se asignan las direcciones IPv4 y se configuran el número de puertos de cada switch. Por su parte, con el fichero "start-ONOS.sh" se importan las aplicaciones de ONOS necesarias para que el controlador ejecute las funciones deseadas, y finalmente se inicia el servicio con el uso del comando *service onos start*.

```
<!-- ONOS -->
<vm name="ONOS" type="lxc" arch="x86_64">
 <filesystem type="cow">/usr/share/vnx/filesystems/vnx_rootfs_lxc_ubuntu64-18.04-v025-onos-1.13.1-TFM</filesystem>
 <if id="2" net="netmgt">
  <ipv4>10.100.200.1/24</ipv4>
 </if>
 <!-- Copy configuration files -->
 <filetree seq="on_boot" root="/root/conf/">conf/ONOS/start-ONOS.sh</filetree>
 <filetree seq="on_boot" root="/opt/onos/">conf/ONOS/options</filetree>
 <filetree seq="on_boot" root="/opt/onos/config/network-cfg.json">conf/ONOS/network-cfg_ipv4.json</filetree>
 <!-- Configure and Start ONOS (IPv4) -->
 <exec seq="config-ONOS" type="verbatim">
    echo 'alias onos-client=/opt/onos/karaf/bin/client' >> /root/.bashrc
    /root/conf/start-ONOS.sh
 </exec>
</vm>
```

Figura 18. Creación de la máquina virtual con ONOS 1.12

Una vez que se construye el escenario y se inicia el servicio ONOS, se debe ingresar a la consola del controlador y crear la ruta multicast tal como se muestra en la figura 19.

```
onos> mcast-join 192.168.0.10 230.4.4.1 of:000000000102/3 of:000000000101/3 of:0
00000000100/3
Added the mcast route: McastRoute{source=192.168.0.10, group=230.4.4.1, origin=S
TATIC}
```

Figura 19. Creación del grupo multicast en ONOS.

De la información contenida en la imagen 19 es necesario explicar cada uno de los parámetros necesarios para la creación de la ruta multicast. Primeramente hay que indicar la dirección IPv4 de la fuente, que en este caso es la 192.168.0.10 correspondiente al "Servidor Web". De la fuente, también es necesario indicar el switch y el puerto al cual se conecta el servidor, el cual es el puerto número tres del "switch2" (of:000000000102/3).

El siguiente parámetro a especificar es la dirección IPv4 multicast a utilizar. En este caso se consideró la IP 230.4.4.1 la cuál es la dirección multicast por defecto utilizada por la herramienta UFTP.

El último parámetro que se debe definir son los sumideros, es decir, los host que se van a subscribir al grupo multicast para recibir la información que va a ser transmitida. Para ello es necesario indicar únicamente el switch y el puerto al que se conecta cada host. Para los efectos de este proyecto, y debido a limitaciones técnicas, solamente se añadieron dos hosts al grupo multicast, los cuales son las máquinas virtuales denominadas "Proxy-A", la cual se conecta a través del puerto tres del "switch1" (of:000000000101/3), y "Proxy-B", que se conecta al puerto tres del "switch0"(of:000000000100/3).

Finalmente, se comprueba que se instala el intent "*SinglePointToMultiPoint*" en el controlador (Figura 20).

```
onos> intents
Id: 0x2
State: WITHDRAWN
Key: 0x2
Intent type: SinglePointToMultiPointIntent
Application Id: org.onosproject.mfwd
Common ingress selector: ETH_TYPE:ipv4, IPV4_SRC:192.168.0.10/32, IPV4_DST:230.4
.4.1/32
Treatment: [NOACTION]
Ingress connect points and individual selectors
 -> Connect Point: of:000000000102/3   Selector: Inherited
Egress connect points and individual selectors
 -> Connect Point: of:000000000101/3   Selector: Inherited
 -> Connect Point: of:000000000101/4   Selector: Inherited
```

Figura 20. Instalación del intent SinglePointToMultiPoint en ONOS.

4.2.2 Servidor Web

Tal como su nombre lo indica, este componente de la topología diseñada cumple con la función de un servidor web común. En él, se almacenan los ficheros de video codificados a una determinada tasa de bit para ser entregados a un cliente. En sí, este elemento es una máquina virtual LXC de arquitectura x86_64 que cuenta principalmente con un sistema operativo Ubuntu 18.04 y el software del servidor web Apache. Esta MV tiene una interfaz de red que se conecta al puerto número tres del "switch2. Al construirse esta MV, se copia el software necesario para su funcionamiento, los segmentos de video codificados, y un script de configuración. Dicho script compila, instala y configura tanto OpenSSL como UFTP, crea los directorios necesarios para el

almacenamiento de los ficheros de video, con sus respectivos permisos, e inicia el servicio Apache. Este script se ejecuta a través de la secuencia "config-Apache".

```
<!-- Servidor Apache -->
<vm name="Servidor-Web" type="lxc" arch="x86_64">
 <filesystem type="cow">/usr/share/vnx/filesystems/vnx_rootfs_lxc_ubuntu64-18.04-v025-onos-1.13.1-TFM</filesystem>
 <if id="1" net="switch2">
  <ipv4>192.168.0.10/24</ipv4>
 </if>
 <!--if id="2" net="bridge-B">
  <ipv4>10.100.11.20/24</ipv4>
 </if-->
 <route type="ipv4" gw="192.168.0.1">default</route>
  <filetree seq="on_boot" root="/home/">conf/Apache/start-Apache.sh</filetree>
  <filetree seq="on_boot" root="/home/">conf/Apache/start_sending.sh</filetree>
  <filetree seq="on_boot" root="/home/">conf/Apache/star-multi-streaming.sh</filetree>
  <filetree seq="on_boot" root="/home/">conf/installOpenSSL.sh</filetree>
  <exec seq="config-Apache" type="verbatim">/home/start-Apache.sh</exec>
   <filetree seq="on_boot" root="/var/www/html/">/media/sf_TFM/Video_prueba/Segments/</filetree>
   <filetree seq="on_boot" root="/usr/local/src/">uftp-4.9.8</filetree>
   <filetree seq="on_boot" root="/home/">conf/openssl-1.0.2o.conf</filetree>
</vm>
```

Figura 21. Creación de la máquina virtual "Servidor Web".

4.2.3 Proxy

En la composición del escenario, las máquinas "Proxy-A" y "Proxy-B" son elementos importantes para alcanzar los objetivos planteados durante el desarrollo de este proyecto. Su función principal es recibir las peticiones del "Host" y entregarle el contenido solicitado, si lo tuviese en caché. Para ello, a estas máquinas virtuales LXC de arquitectura x86_64 se les crea la secuencia "config-Squid" donde se ejecuta un script que crea los directorios necesarios con los permisos correspondientes, compila, instala y configura el software que se utiliza en este proyecto y finalmente inicia todos los servicios indispensables para llevar a cabo sus funciones. El código empleado para la construcción de estos elementos se puede observar en las figura 22(a) y 22(b).

```
<!-- Proxy A-->
<vm name="Proxy-A" type="lxc" arch="x86_64">
 <filesystem type="cow">/usr/share/vnx/filesystems/vnx_rootfs_lxc_ubuntu64-18.04-v025-onos-1.13.1-TFM</filesystem>
 <if id="1" net="switch1">
  <ipv4>192.168.1.10/24</ipv4>
 </if>
 <if id="2" net="bridge-A">
  <ipv4>192.168.10.10/24</ipv4>
 </if>
 <route type="ipv4" gw="192.168.1.1">default</route>
  <!-- Copiar archivos de conf -->
  <filetree seq="on_boot" root="/home/">conf/Squid/start-Squid.sh</filetree>
  <filetree seq="on_boot" root="/home/">conf/Squid/peticion.sh</filetree>
  <filetree seq="on_boot" root="/home/">conf/installOpenSSL.sh</filetree>
  <filetree seq="on_boot" root="/usr/local/src/">uftp-4.9.8</filetree>
  <filetree seq="on_boot" root="/home/">conf/openssl-1.0.2o.conf</filetree>
  <exec seq="config-Squid" type="verbatim">/home/start-Squid.sh</exec>
</vm>
```

Figura 22(a). Creación de la máquina virtual "Proxy-A".

```
<!-- Proxy B -->
<vm name="Proxy-B" type="lxc" arch="x86_64">
 <filesystem type="cow">/usr/share/vnx/filesystems/vnx_rootfs_lxc_ubuntu64-18.04-v025-onos-1.13.1-TFM</filesystem>
 <if id="1" net="switch0">
  <ipv4>10.100.100.10/24</ipv4>
 </if>
 <if id="2" net="bridge-B">
  <ipv4>10.100.11.10/24</ipv4>
 </if>
 <route type="ipv4" gw="10.100.100.1">default</route>
 <!-- Copiar archivos de conf -->
 <filetree seq="on_boot" root="/home/">conf/Cliente/start-Cliente.sh</filetree>
 <filetree seq="on_boot" root="/home/">conf/Squid/peticion.sh</filetree>
 <filetree seq="on_boot" root="/home/">conf/installOpenSSL.sh</filetree>
 <filetree seq="on_boot" root="/usr/local/src/">uftp-4.9.8</filetree>
 <filetree seq="on_boot" root="/home/">conf/openssl-1.0.2o.conf</filetree>
 <exec seq="config-Cliente" type="verbatim">/home/start-Cliente.sh</exec>
</vm>
```

Figura 22(b). Creación de la máquina virtual "Proxy-B".

Cada una de estas máquinas virtuales cuenta con dos interfaces de red. El "Proxy-A" se conecta "switch1" a través de la interfaz "eth1", mientras que por "eth2" se conecta al puente "bridge-A". Por su parte, el "Proxy-B" se conecta al "switch0" y a la red de gestión "netmgt" a través de las interfaces "eth1" y "eth2", respectivamente.

4.2.4 Host

El host es un ordenador portátil que cuenta con las siguientes características:

- RAM: 8Gb
- Procesador: 4 núcleos
- Memoria de video: 128Mb
- HDD: 500Gb

Adicionalmente, en el escenario virtual se le crean tres interfaces de red al host, tal como se muestra en la figura 23, para que pueda tener comunicación con los demás elementos de red. Específicamente, la primera y la segunda interfaz del host se conectan a los puentes "bridge-A" y "bridge-B", respectivamente, mientras que la tercera se conecta a la red de gestión, para poder acceder a la consola de administración web del controlador. Es importante destacar que para que exista comunicación entre el host y las máquinas virtuales de tipo lxc, debe haber un puente virtual de por medio. Adicionalmente, con este despliegue se obtiene la ventaja de simular que cada "Proxy" y el host son un solo usuario final. Es en el host que se puede reproducir el video, a través del navegador web Google Chrome, para realizar las pruebas requeridas.

```
<hostif net="bridge-B">
  <ipv4>10.100.11.3/24</ipv4>
</hostif>
<hostif net="bridge-A">
  <ipv4>192.168.10.30/24</ipv4>
</hostif>
<hostif net="netmgt">
  <ipv4>10.100.200.30/24</ipv4>
</hostif>
```

Figura 23. Creación de la interfaz de red en el Host.

4.2.5 Switches virtuales

El escenario cuenta con tres switches virtuales de tipo openvswitch conectados en malla. Cada uno de los switches cuenta con el protocolo OpenFlow versión 1.3 y todos están conectados al controlador SDN a través del puerto TCP 6633. Adicionalmente, se debe especificar el nombre del switch, el modo de fallo, la dirección MAC y los parámetros de conexión. En la figura 24 se puede observar el código empleado para la creación de los switches virtuales y el valor que se le asigna a cada uno de los parámetros mencionados.

```
<!--SDN switches -->
<net name="switch0" mode="openvswitch" controller="tcp:10.100.200.1:6633" of_version="OpenFlow13"
   fail_mode="secure" hwaddr="00:00:00:00:01:00">
   <connection name='s0s1' net='switch1'/>
</net>
<net name="switch1" mode="openvswitch" controller="tcp:10.100.200.1:6633" of_version="OpenFlow13"
   fail_mode="secure" hwaddr="00:00:00:00:01:01">
   <connection name='s1s2' net='switch2'/>
</net>
<net name="switch2" mode="openvswitch" controller="tcp:10.100.200.1:6633" of_version="OpenFlow13"
   fail_mode="secure" hwaddr="00:00:00:00:01:02">
   <connection name='s2s0' net='switch0'/>
</net>
```

Figura 24. Creación de los switches virtuales OpenFlow

En la tabla 3 se resume toda la información relevante de los elementos de la red.

Tabla 3. Información de los elementos de la red.

Elemento de red	Dirección IPv4	OpenFlow	Puertos
ONOS	10.100.200.1	N/A	N/A
Switch0	10.100.100.1	000000000100	2
Switch1	192.168.1.1	000000000101	4
Switch2	192.168.0.1	000000000102	3
Servidor-Web (Eth1)	10.100.11.20	N/A	N/A
Proxy-A – Eth1	192.168.1.10	N/A	N/A
Proxy-A – Eth2	192.168.10.10	N/A	N/A
Proxy-B – Eth1	10.100.100.10	N/A	N/A
Proxy-B – Eth2	10.100.11.10	N/A	N/A
Host – bridge-B	10.100.11.3	N/A	N/A
Host – bridge-A	192.168.10.30	N/A	N/A
Host – netmgt	10.100.200.30	N/A	N/A

4.3 Pruebas de conectividad

Antes de realizar las pruebas de streaming, se procedió a validar la construcción de los elementos que conforman la topología y la comunicación entre ellos, tanto en unicast como en multicast.

En primer lugar se validó el correcto funcionamiento del servicio ONOS, y que todas las aplicaciones necesarias estuvieran instaladas y en estado activo (Figura 25).

```
onos> apps -a -s
*   9 org.onosproject.optical-model        1.12.0   Optical Network Model
*  15 org.onosproject.route-service        1.12.0   Route Service Server
*  29 org.onosproject.intentsynchronizer   1.12.0   Intent Synchronizer
*  30 org.onosproject.sdnip                1.12.0   SDN-IP
*  31 org.onosproject.hostprovider         1.12.0   Host Location Provider
*  32 org.onosproject.lldpprovider         1.12.0   LLDP Link Provider
*  33 org.onosproject.openflow-base        1.12.0   OpenFlow Base Provider
*  34 org.onosproject.openflow             1.12.0   OpenFlow Provider Suite
*  42 org.onosproject.drivers              1.12.0   Default Drivers
*  48 org.onosproject.mfwd                 1.12.0   Multicast Forwarding
*  90 org.onosproject.proxyarp             1.12.0   Proxy ARP/NDP
* 118 org.onosproject.reactive-routing     1.12.0   SDN-IP Reactive Routing
* 135 org.onosproject.fwd                  1.12.0   Reactive Forwarding
onos>
```

Figura 25. Aplicaciones instaladas y activadas en ONOS.

En segundo lugar se validó que los elementos de la red se pudieran comunicar entre sí. Para ello se realizaron pings unicast entre cada componente de la topología y se observó que en cada caso se obtuvo una respuesta satisfactoria. Luego, se validó el funcionamiento de la aplicación MFWD de ONOS, mediante el uso de la herramienta iperf. Esta prueba consistió en un "ping multicast" enviado por el "Servidor Web" y recibido tanto por el "Proxy-A" como por el "Proxy-B". Al igual que en la primera prueba la respuesta fue satisfactoria y se comprobó que el envío y recepción de paquetes en multicast se realiza correctamente. Los detalles de esta prueba se pueden observar en las imágenes 26, 27 y 28.

```
Client connecting to 230.4.4.1, UDP port 5001
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
Setting multicast TTL to 32
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 192.168.0.10 port 33481 connected with 230.4.4.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec   131 KBytes  1.07 Mbits/sec
[  3]  1.0- 2.0 sec   126 KBytes  1.03 Mbits/sec
[  3]  2.0- 3.0 sec   129 KBytes  1.06 Mbits/sec
[  3]  3.0- 4.0 sec   128 KBytes  1.05 Mbits/sec
[  3]  4.0- 5.0 sec   128 KBytes  1.05 Mbits/sec
[  3]  5.0- 6.0 sec   129 KBytes  1.06 Mbits/sec
[  3]  6.0- 7.0 sec   128 KBytes  1.05 Mbits/sec
[  3]  7.0- 8.0 sec   128 KBytes  1.05 Mbits/sec
```

Figura 26. Envío de "ping multicast" desde el "Servidor Web", usando iperf.

```
Server listening on UDP port 5001
Binding to local address 230.4.4.1
Joining multicast group  230.4.4.1
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 230.4.4.1 port 5001 connected with 192.168.0.10 port 57010
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0- 1.0 sec   129 KBytes  1.06 Mbits/sec   0.917 ms    0/   90 (0%)
[  3]  1.0- 2.0 sec   128 KBytes  1.05 Mbits/sec   0.815 ms    0/   89 (0%)
[  3]  2.0- 3.0 sec   128 KBytes  1.05 Mbits/sec   1.257 ms    0/   89 (0%)
[  3]  3.0- 4.0 sec   129 KBytes  1.06 Mbits/sec   0.526 ms    0/   90 (0%)
[  3]  4.0- 5.0 sec   128 KBytes  1.05 Mbits/sec   0.749 ms    0/   89 (0%)
[  3]  5.0- 6.0 sec   128 KBytes  1.05 Mbits/sec   0.964 ms    0/   89 (0%)
[  3]  6.0- 7.0 sec   128 KBytes  1.05 Mbits/sec   0.870 ms    0/   89 (0%)
[  3]  7.0- 8.0 sec   128 KBytes  1.05 Mbits/sec   0.364 ms    0/   89 (0%)
```

Figura 27. Recepción del "ping multicast" en el "Proxy-A", usando iperf.

```
Server listening on UDP port 5001
Binding to local address 230.4.4.1
Joining multicast group  230.4.4.1
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 230.4.4.1 port 5001 connected with 192.168.0.10 port 33481
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0- 1.0 sec   129 KBytes  1.06 Mbits/sec   0.575 ms    0/   90 (0%)
[  3]  1.0- 2.0 sec   126 KBytes  1.03 Mbits/sec   0.463 ms    0/   88 (0%)
[  3]  2.0- 3.0 sec   129 KBytes  1.06 Mbits/sec   0.430 ms    0/   90 (0%)
[  3]  3.0- 4.0 sec   128 KBytes  1.05 Mbits/sec   0.421 ms    0/   89 (0%)
[  3]  4.0- 5.0 sec   128 KBytes  1.05 Mbits/sec   0.508 ms    0/   89 (0%)
[  3]  5.0- 6.0 sec   129 KBytes  1.06 Mbits/sec   0.253 ms    0/   90 (0%)
[  3]  6.0- 7.0 sec   128 KBytes  1.05 Mbits/sec   0.438 ms    0/   89 (0%)
[  3]  7.0- 8.0 sec   128 KBytes  1.05 Mbits/sec   0.778 ms    0/   89 (0%)
```

Figura 28. Recepción del "ping multicast" en el "Proxy-B", usando iperf.

5 Experimentación

Para realizar todas las pruebas requeridas durante el desarrollo de este Trabajo Fin de Máster se utilizó el video "Big Buck Bunny", el cual es un corto animado desarrollado por el Instituto Blender mediante el uso de herramientas de software libre. [27]

5.1 Transcodificación del vídeo y generación de segmentos

De acuerdo a la propuesta del 3GPP sobre la combinación de DASH y multicast para enviar segmentos multimedia a dispositivos móviles sobre redes LTE, la cual se describe en la sección 2.3, se generó una única representación del video. Para la transcodificación del video se utilizó la herramienta *ffmpeg* descrita en el capítulo 4. Las características de dicha representación se pueden encontrar en la tabla 4.

Tabla 4. Características de la representación del vídeo.

Representación	Bitrate	Escala
1	2 Mbps	1280:720

Por su parte, para generar los segmentos de video y para crear el fichero .MPD se utilizó la herramienta *MP4Box*, también descrita en el capítulo 4. Cada segmento se creó con una duración de 2 segundos. En la figura 29 se puede observar la estructura del archivo .MPD generado.

```
<?xml version="1.0"?>
<!-- MPD file Generated with GPAC version 0.5.1-DEV-rev5619  on 2018-08-15T19:18:26Z-->
<MPD xmlns="urn:mpeg:dash:schema:mpd:2011" minBufferTime="PT1.500000S" type="static" mediaPresentat
 <ProgramInformation moreInformationURL="http://gpac.sourceforge.net">
  <Title>videoBBB.mpd generated by GPAC</Title>
 </ProgramInformation>

 <Period duration="PT0H9M56.46S">
  <AdaptationSet segmentAlignment="true" bitstreamSwitching="true" maxWidth="1280" maxHeight="720" :
   <SegmentList>
    <Initialization sourceURL="videoBBB_init.mp4"/>
   </SegmentList>
   <Representation id="1" mimeType="video/mp4" codecs="avc3.42c01e" width="1280" height="720" frame
  </AdaptationSet>
 </Period>
</MPD>
```

Figura 29. Estructura del archivo .mpd

5.2 Streaming de video en multicast

La idea general sobre la que se trabajó durante la fase de experimentación de este proyecto, y que se puede observar en la figura 30, es que el "Servidor Web" pueda realizar el envío de ficheros de video simultáneamente a varios clientes. Para ello se ideó que cada cliente contase con un proxy, a manera de caché, y que mediante la ejecución de un script, el "Servidor Web" realice el envío de los segmentos de video a cada uno de los proxies que se hayan unido al grupo multicast. Cada cliente entonces realizará peticiones unicast a su proxy para solicitar los ficheros de video que correspondan. Si algún fichero de video se pierde durante la transmisión en multicast, entonces se acciona un mecanismo de recuperación unicast de segmentos multimedia.

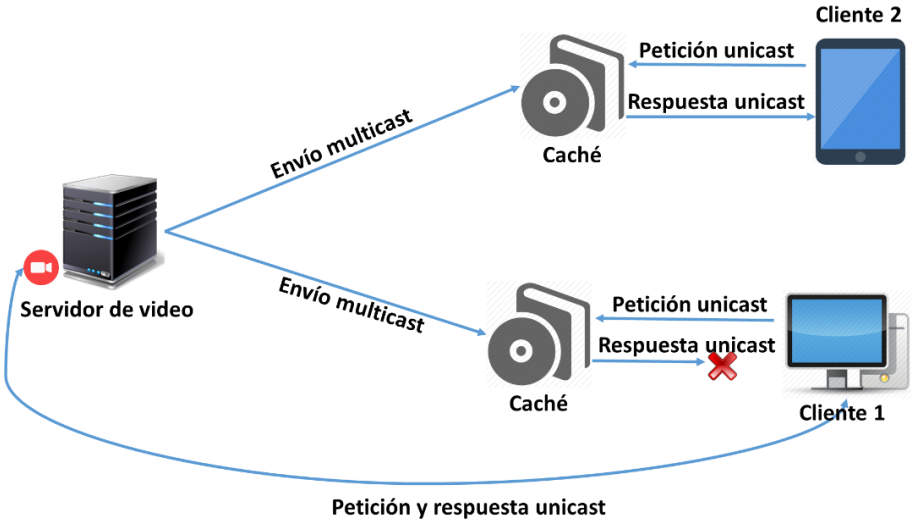

Figura 30. Representación esquemática del funcionamiento del escenario de red

5.2.1 Envío de ficheros en multicast

Para realizar el envío multicast de los ficheros se probaron diversos métodos y herramientas. En primer lugar, se intentó utilizar el software del proyecto ATSC_ROUTE, el cual es una implementación del protocolo ROUTE, que está compuesto por dos módulos: un emisor y un receptor, y cuyo funcionamiento consiste en el envío de datos sobre una red de área local (LAN), permitiendo que el receptor pueda sintonizar cualquiera de los canales que estén siendo transmitidos [25]. En la imagen 31 se muestra la configuración básica de ambos módulos. El código fuente de este proyecto se descargó desde GitHub y se intentó compilar, sin embargo y a pesar de haber cumplido con todos los requisitos indicados en la documentación, no pudo compilarse, por lo tanto se descartó esta opción. Algunos de los errores que se encontraron, están documentados en la sección de anexos. Cabe destacar que la última actualización de este proyecto fue en noviembre de 2016 y actualmente no está siendo mantenido.

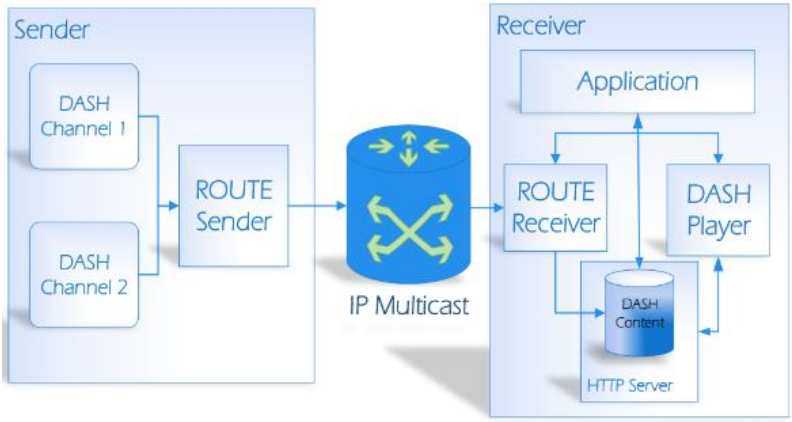

Figura 31. Configuración básica de emisores y receptores ATSC-ROUTE

Una vez agotados todos los recursos con el ATSC_ROUTE, y no haber obtenido un resultado satisfactorio, se optó por utilizar el código del proyecto MAD_FLUTE [43]. Este proyecto fue desarrollado en la Universidad de Tampere, Finlandia, y consiste en la implementación del protocolo FLUTE, el cual se describe en detalle en el capítulo 2. Al igual que en el caso anterior, también se encontraron una serie de problemas con este software. Todos los inconvenientes que se presentaron están relacionados con el hecho de que este código tiene más de diez años sin ser actualizados (la última actualización que se observa en su web data del mes de marzo del año 2007), y el sistema operativo sobre el que se está trabajando es un Ubuntu 18.04, el cual, para el momento del desarrollo de este trabajo, era la última versión publicada por Linux. En consecuencia muchas de las librerías que este software necesita ya no se encuentran disponibles.

Por último se utilizó el software UFTP, el cual también se describe en el capítulo 4. Este software se descargó, se compiló y se instaló sin inconveniente alguno. Para ello fue necesario instalar y configurar previamente el software OpenSSL. Todas estas acciones se realizaron a través del script installOpenSSL.sh el cual se puede encontrar en la sección de anexos. En la imagen 32 se muestra una captura de pantalla donde se puede observar el envío de un fichero utilizando esta herramienta.

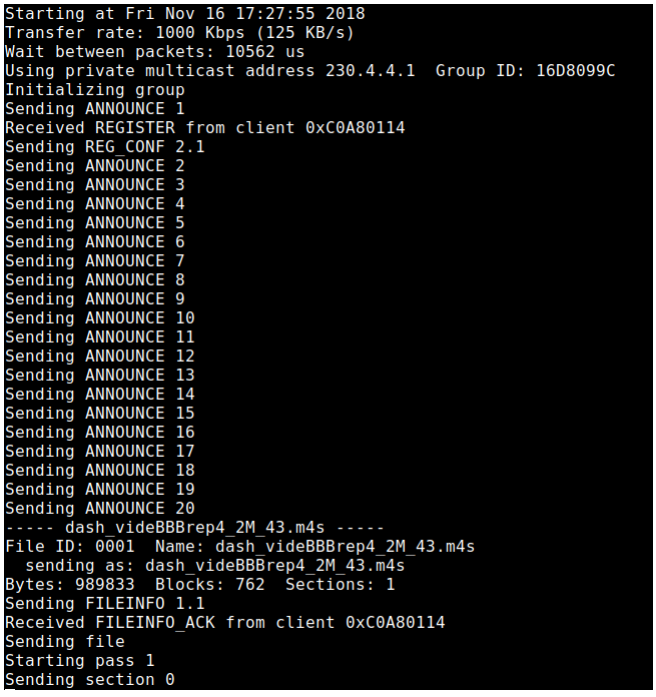

Figura 32. Envío de ficheros utilizando la herramienta UFTP

Para realizar el envío de ficheros desde el "Servidor Web" es necesario ejecutar el comando correspondiente con las opciones pertinentes, siendo su forma básica la siguiente:

uftp –M dirección_IP_multicast –P dirección_IP_multicast fichero_a_enviar

Por su parte, para recibir los ficheros es necesario que cada cliente interesado esté previamente unido al grupo multicast, para luego ejecutar el comando con las opciones correspondientes. En general este comando tiene la siguiente forma:

uftpd –D directorio_donde_se_almacenaran_los_ficheros

Si se ejecuta el envío desde el servidor y no hay ningún cliente escuchando, el envío no se realiza. Esto significa que es necesario que existan clientes "escuchando" a la IP multicast antes de realizar el envío. Todos los scripts diseñados para estos efectos se pueden encontrar en la sección de anexos.

5.2.2 Mecanismo de recuperación unicast de segmentos de video perdidos

Tal como se comentó en la sección 2.3, la transmisión de segmentos multimedia a través de un canal eMBMS es propensa a sufrir errores que impactan directamente la calidad de experiencia percibida por el usuario. Considerando esta posibilidad, se ideó un mecanismo de "recuperación" de segmentos de video en unicast. Este mecanismo solamente se activa cuando el cliente solicita a su proxy un fragmento de video que este no posee. En este caso, si el proxy devuelve a su cliente un error, este pide dicho segmento de video directamente al "Servidor Web".

Una vez que fueron estudiadas las bases teóricas descritas en [9], se procedió a analizar en profundidad la lógica del código JavaScript del cliente DASH. A groso modo, lo primero que hace este código es leer el fichero .MPD que se generó previamente, el cual se explica en la sección 5.1. Para poder leer dicho archivo, se añade al fichero *sources.json* del cliente DASH la URL donde se encuentra el fichero .MPD. Seguidamente, se crea una solicitud HTTP que se compone de la dirección IP de la máquina a la cual se le va a hacer la solicitud (en este caso, el "Servidor Web") y el nombre del fichero que se va a solicitar junto con su ubicación. Luego, se obtienen las métricas DASH y se inicia la reproducción del segmento de video recuperado. Si un segmento de video no se consigue, el cliente vuelve a solicitarlo. En total, se realiza un máximo de cinco intentos para recuperar un determinado fichero. Si realizado el último intento no se ha podido recuperar el segmento deseado, el cliente detiene la reproducción del video y muestra un mensaje de error, como el que se muestra en la imagen 33.

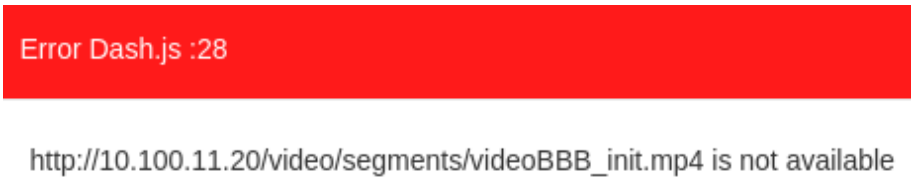

Figura 33. Ejemplo de error mostrado por el cliente DASH cuando no consigue un fichero

Habiendo comprendido el funcionamiento del cliente DASH, se ideó la siguiente solución para recuperar el fichero no encontrado: cuando el cliente solicita a su proxy un

determinado segmento de video y este le responde con un error *404 not found*, el cliente modifica la petición original para ejecutarla directamente contra el "Servidor Web". En otras palabras, se modifica la IP a donde se va a lanzar la petición HTTP pero se mantiene el segmento de video a solicitar. El fragmento de código modificado del cliente DASH se muestra en la imagen 34.

```
const onload = function () {
    if (httpRequest.response.status >= 200 && httpRequest.response.status <= 299) {
        handleLoaded(true);

        if (config.success) {
            config.success(httpRequest.response.response, httpRequest.response.statusText, httpRequest.response.responseURL);
        }

        if (config.complete) {
            config.complete(request, httpRequest.response.statusText);
        }
    }else if(httpRequest.response.status == 404){

                var new_serviceLocation = "http://192.168.0.10/video/segments/";
                var original_url = httpRequest.url;
                var fields = original_url.split("/");
                var video_segment = fields[5];
                var new_url = new_serviceLocation.concat(video_segment);

                httpRequest.url = new_url
                httpRequest.request.serviceLocation = new_serviceLocation;
                httpRequest.request.url = new_url;

                const modifiedUrl = requestModifier.modifyRequestURL(request.url);
                const verb = request.checkExistenceOnly ? HTTPRequest.HEAD : HTTPRequest.GET;
```

Figura 34. Código modificado en el cliente DASH para recuperar los segmentos de video desde el "Servidor Web".

Como se puede observar en la imagen 34, se trabaja sobre la función *onload* del fichero *HTTPLoader.js* del cliente DASH, y lo que se hace es añadir el bloque de código *else if (httpRequest.response.status == 404)*. Dicho en otras palabras, si el estado de la respuesta que envía el "Proxy" al cliente es igual a 404, se entra en este nuevo trozo de código y se procede a modificar la petición HTTP original. Un ejemplo del formato de la petición original es el siguiente:

http://192.168.10.10/video/segments/dash_videBBBrep4_2M_1.m4s

Para realizar la modificación de la petición HTTP original se crean las variables *new_serviceLocation* donde se almacena la URL que almacena los segmentos de video en el "Servidor Web", *original_url* que como su nombre lo indica almacena la URL original como cadena de texto, *fields* que es un arreglo que en cada posición almacena el contenido de la variable *original_url* separado por el símbolo "/", *video_segment* que es igual a la última posición del arreglo *fields*, es decir, el nombre del fichero de video a solicitar, y por último la variable *new_url* que es la concatenación de las variables *new_serviceLocation* y *video_segment*. En la tabla 5 se puede apreciar, a manera de ejemplo, el contenido de cada una de las variables mencionadas.

Tabla 5. Contenido de las variables del nuevo fragmento de código javascript del cliente DASH.

Variable	Contenido
new_serviceLocation	http://192.168.0.10/video/segments/
original_url	http://192.168.10.10/video/segments/dash_videBBBrep4_2M_1.m4s
fields	http:, , 192.168.10.10, video, segments, dash_videBBBrep4_2M_1.m4s
video_segment	dash_videBBBrep4_2M_1.m4s
new_url	http://192.168.0.10/video/segments/dash_videBBBrep4_2M_1.m4s

5.2.3 Escenario VoD

Como se explicó en la sección anterior, el envío de ficheros se realizó mediante un script donde en el "Servidor Web" se ejecuta el comando:

uftp –M 230.4.4.1 –P 230.4.4.1 –R 4000

Donde las opciones *–M* y *–P* indican que la dirección IP es multicast y privada, respectivamente, y la opción *–R* especifica la tasa de transmisión en Kbps. Por su parte, para poder recibir los segmentos de video en todos aquellos receptores (proxies) que se encuentren unidos al grupo multicast se ejecutó el comando:

uftpd –D /var/www/html/video/segments

En este caso, la opción *–D* indica el directorio en donde se desean almacenar los ficheros recibidos. Para emular un escenario VoD, solo fue necesario un cliente (representado por la máquina virtual donde se desarrolló este proyecto). La reproducción del video se visualizó mediante el explorador Google Chrome. Una vez que se ejecutó el script mencionado en el "Servidor Web", se esperó unos pocos segundos para que el proxy del cliente recibiera una cantidad considerable de ficheros, y así evitar paradas durante la reproducción del video.

Se inició la reproducción del video en el cliente DASH, y gracias a las herramientas de desarrollo que provee Google Chrome, se pudo mirar en detalle todo lo que iba sucediendo. Durante la reproducción se pudo apreciar como en determinado momento el cliente DASH solicita al "Proxy" el segmento de video *dash_videBBBrep4_2M_10.m4s* y el código de la respuesta obtenida es *404 Not Found*. Como se comentó anteriormente, el script de envío de ficheros fue diseñado para que no envíe determinados segmentos de video. En este caso, este segmento no fue enviado intencionalmente para así poner a prueba el mecanismo de recuperación. Seguidamente se puedo observar como el cliente, al haber obtenido el error *404 Not Found*, solicita el segmento de video al servidor web, obteniendo así una respuesta satisfactoria, para que, finalmente, el siguiente segmento

fuera recuperado del "Proxy" y no del "Servidor web". Es decir, el cliente DASH vuelve a su funcionamiento normal.

5.2.4 Escenario Live

Una vez estudiado el caso de video bajo demanda, también se quiso emular un caso de live streaming. La idea fue simular un escenario real de streaming en vivo, o IPTV, en donde varios clientes se unieran a la transmisión en diferentes momentos. Traduciéndolo a nuestra topología de red, cada cliente establecería comunicaciones unicast con el "Servidor Web" y cuando el controlador de la red determine que el número de clientes es muy alto, cambiaría todas las comunicaciones unicast por multicast, de modo que se pueda sacar mayor provecho a los recursos de la red.

Inconvenientes encontrados y soluciones aplicadas

Sin embargo, surgieron algunos inconvenientes que impidieron desarrollar esta idea de la manera que fue originalmente concebida. El primer problema encontrado está relacionado con la actualización o modificación del grupo multicast. Si los clientes se unen al grupo multicast en diferentes momentos, este se tendría que modificar (en el controlador de la red) en "caliente" cada vez que un cliente desee unirse a la transmisión, o separarse del grupo multicast. Sin embargo, de acuerdo a la documentación del sitio oficial de ONOS [46], para el momento del desarrollo de este proyecto el módulo *multicast forwarding* está siendo desarrollado en fases, y por lo tanto, aún presenta una funcionalidad bastante básica. Por lo tanto, no se encontró una manera de hacerlo. Para solucionar esta situación, se resolvió por añadir previamente todos los clientes al grupo multicast, de la siguiente manera:

mcast-join 192.168.0.10 230.4.4.1 of:000000000102/3 of:000000000101/3 of:000000000100/3

Por último, y por las razones argumentadas en el párrafo anterior, no se encontró una manera para que el controlador de la red pueda determinar de manera automática el número de clientes que se encuentran asociados a un grupo multicast para proceder a cambiar las transmisiones unicast por multicast. Para subsanar esta situación, se ideó que el Host, mediante el mecanismo de recuperación de segmentos de video en unicast, recuperase los ficheros desde el "Servidor Web", y que en un momento aleatorio, se iniciase la transmisión en multicast de los segmentos de video. Por lo tanto, el Host ya no consumiría estos ficheros desde el servidor, sino que lo haría desde su Proxy.

En la imagen 35 se puede apreciar la captura de tráfico tomada con la herramienta Wireshark, en la cual se observa que, una vez activada la transmisión en multicast, el host pasa a consumir los segmentos de video desde el "Proxy-B", y no desde el "Servidor

Web". Cabe destacar que en la figura 39 se muestra en detalle el funcionamiento del protocolo utilizado por el software UFTP, el cual es explicado en detalle en el capítulo 4

Source	Destination	Protocol	Length	Info
192.168.0.10	230.4.4.1	UFTP	1366	FILESEG
192.168.0.10	230.4.4.1	UFTP	1366	FILESEG
192.168.0.10	230.4.4.1	UFTP	1366	FILESEG
192.168.0.10	230.4.4.1	UFTP	1366	FILESEG
192.168.0.10	230.4.4.1	UFTP	1366	FILESEG
192.168.0.10	230.4.4.1	UFTP	703	FILESEG
192.168.0.10	230.4.4.1	UFTP	74	DONE
10.100.100.10	192.168.0.10	UFTP	66	COMPLETE
192.168.0.10	230.4.4.1	UFTP	74	DONE
10.100.100.10	192.168.0.10	UFTP	66	COMPLETE
192.168.0.10	230.4.4.1	UFTP	74	DONE
10.100.100.10	192.168.0.10	UFTP	66	COMPLETE
192.168.0.10	230.4.4.1	UFTP	74	DONE
10.100.100.10	192.168.0.10	UFTP	66	COMPLETE
192.168.0.10	230.4.4.1	UFTP	74	DONE
10.100.100.10	192.168.0.10	UFTP	66	COMPLETE
192.168.0.10	230.4.4.1	UFTP	70	DONE_CONF

Figura 35. Captura del tráfico en la interfaz eth1 del "Proxy-B" al iniciar la transmisión multicast.

Ilustración práctica de los beneficios de utilizar multicast en streaming multimedia

Para apreciar los beneficios del uso de multicast en el streaming multimedia, se iniciaron transmisiones en unicast entre el "Servidor Web" y cada uno de los Proxy. Seguidamente, y luego de transcurrido algunos segundos, se sustituyeron las transmisiones en unicast por una única transmisión en multicast.

Aunque ya todos los clientes pertenecían al grupo multicast, como se comentó anteriormente, resultó necesario simular que estos no se unan a la vez. Para ello, se inició la recepción de ficheros, a través de UFTP, en momentos diferentes. En la imagen 36 se puede observar como el "Servidor Web" intenta enviar los segmentos de video a la máquina virtual denominada "Proxy-A", en la cual no se ha iniciado la recepción, por lo tanto, el anuncio entra en timeout.

```
Starting at Mon Nov 26 11:51:05 2018
Transfer rate: 1000 Kbps (125 KB/s)
Wait between packets: 10562 us
Using private multicast address 192.168.1.10  Group ID: 6487C20E
Initializing group
Sending ANNOUNCE 1
Sending ANNOUNCE 2
Sending ANNOUNCE 3
Sending ANNOUNCE 4
Sending ANNOUNCE 5
Sending ANNOUNCE 6
Sending ANNOUNCE 7
Sending ANNOUNCE 8
Sending ANNOUNCE 9
Sending ANNOUNCE 10
Sending ANNOUNCE 11
Sending ANNOUNCE 12
Sending ANNOUNCE 13
Sending ANNOUNCE 14
Sending ANNOUNCE 15
Sending ANNOUNCE 16
Sending ANNOUNCE 17
Sending ANNOUNCE 18
Sending ANNOUNCE 19
Sending ANNOUNCE 20
Announce timed out
```

Figura 36. Timeout durante el envío de ficheros al cliente "Proxy-A".

Por su parte, en la imagen 37 se aprecia la recepción de ficheros en el cliente "Proxy-B", el cual está habilitado para recibir ficheros.

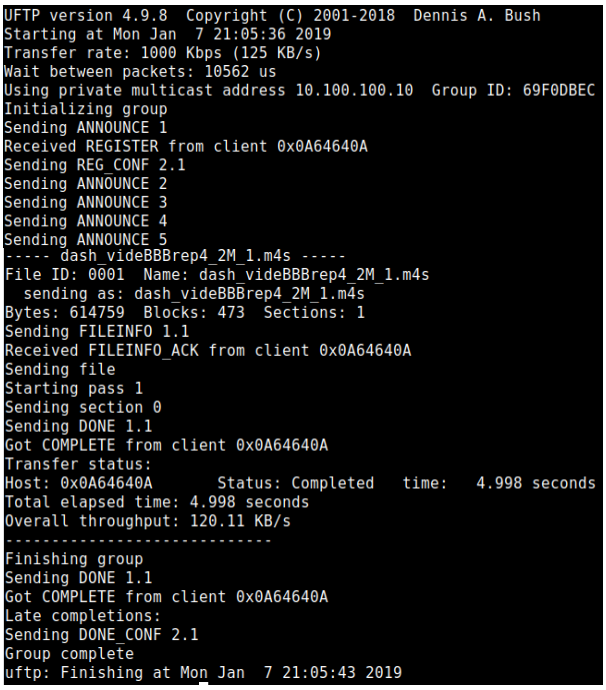

Figura 37. Envío de un segmento de video a la máquina "Proxy-B".

Por su parte, en la imagen 38 se puede observar cómo el "Servidor Web" transmite un segmento de video utilizando la IP 230.4.4.1 y recibe ACK de parte de ambos clientes, es decir, el tipo de envío de ficheros cambia de unicast a multicast.

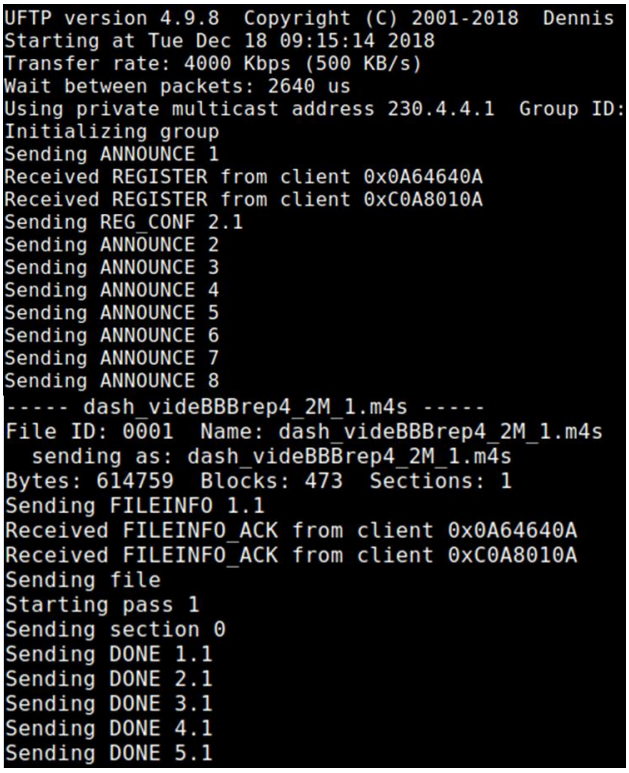

Figura 38. Transmisión de ficheros de video en multicast.

Habiéndose comprobado que los segmentos de contenido multimedia se recibían en ambas máquinas, se procedió a iniciar su reproducción para verificar que, al igual que en el caso del escenario VoD, este se realizaba sin problema alguno y que, cuando fuese necesario, se activase el mecanismo de recuperación de ficheros. Antes de iniciar la reproducción del video se esperó unos pocos segundos (aproximadamente 20) desde el momento que inició la transmisión en multicast, de modo que el nivel del buffer fuera lo suficientemente alto para minimizar el número paradas durante la reproducción.

Para finalizar, mediante el uso de la herramienta Wireshark, se realizaron capturas del tráfico en la red durante múltiples transmisiones simultáneas en unicast, y durante una única transmisión en multicast. En estas capturas se puede observar cómo se aprovechan de mejor manera los recursos de la red al utilizar transmisiones multicast en la distribución de contenido multimedia.

En la figura 39, se observa que cuando el "Servidor Web" mantuvo comunicaciones unicast simultaneas con el "Proxy-A" y el "Proxy-B" el consumo de ancho de banda fue considerablemente alto, siendo el máximo nivel alcanzado 63 Mbps y el promedio fue de 40 Mbps, aproximadamente.

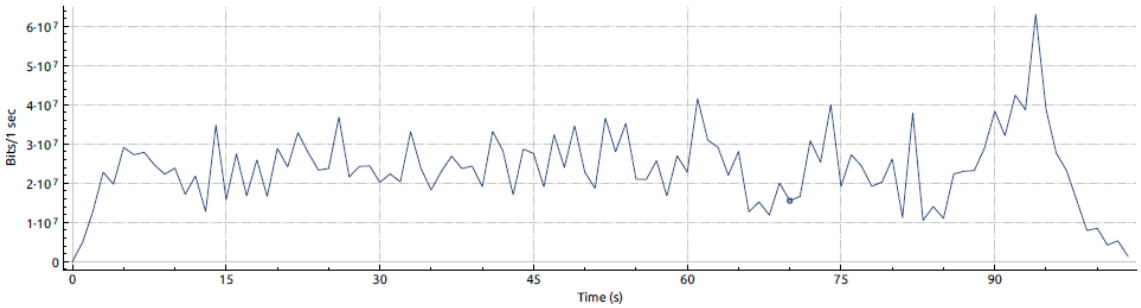

Figura 39. Consumo del ancho de banda con múltipes comunicaciones en unicast simútaneas.

Por su parte, en la imagen 40 podemos observar cómo se reduce considerablemente el ancho de banda al utilizar el envío de ficheros en multicast a todos los clientes interesados. En esta gráfica podemos observar que el mayor pico de consumo fue de aproximadamente 7.48 Mbps, lo que representa un 88% menos en relación al pico observado en la figura 44. Por su parte, el ancho de banda promedio utilizado durante multicast fue de aproximadamente 5Mbps, un valor considerablemente menor a los 40 Mbps observados cuando se establecieron múltiples comunicaciones unicast simultaneas.

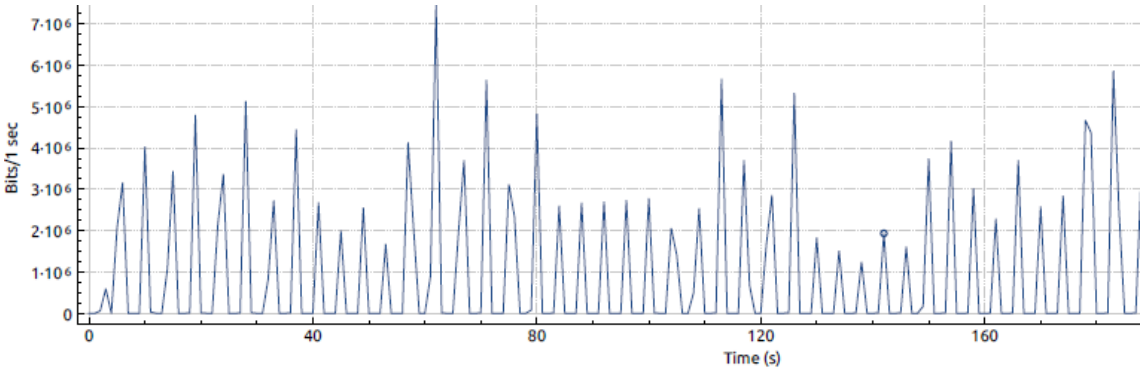

Figura 40. Consumo del ancho de banda durante la transmisión de ficheros en multicast.

6 Conclusiones

Durante la fase de investigación de este Trabajo Fin de Máster, se analizó exhaustivamente la tecnología que actualmente es la más utilizada para la transmisión de contenido multimedia en internet: DASH. Este análisis abarcó distintos aspectos, como su arquitectura, la manera jerárquica en la que estructura el contenido y los beneficios que presenta, siendo esta última la razón principal por la cual está técnica es ampliamente utilizada por gigantes tecnológicos dedicados al streaming de video como YouTube o Netflix. Toda esta información permitió asentar bases teóricas indispensables para desarrollar las siguientes fases de este proyecto y comprender la realidad de este sector.

Se estudió el funcionamiento y las características de los protocolos ROUTE y FLUTE, que son de los más utilizados para la transmisión de video sobre internet, así como los inconvenientes que presentan sus implementaciones de software libre, para así comprender cuál de todos ellos era el más apto para alcanzar los objetivos trazados.

En lo que a SDN respecta, se investigaron los aspectos teóricos indispensables para comprender a plenitud esta tendencia innovadora del mundo de las comunicaciones. Esta investigación abarcó diferentes áreas como lo son su arquitectura, los términos básicos utilizados dentro de este nuevo paradigma de las redes de computadores, y especialmente su propuesta de valor, que a la vez es su principal diferencia con respecto a las redes clásicas, que es la separación del plano de control del plano de datos, y alojarlo en una unidad de control lógico centralizada. Esta separación permite que las redes sean programables y mucho más flexibles.

Se describen las características y los mensajes de OpenFlow, el protocolo mayormente utilizado en SDN, el ciclo de vida de un paquete y la anatomía de un switch OpenFlow. Al igual que en el caso de DASH, todos estos concepto fueron necesarios para el desarrollo de la fase práctica de este proyecto, así como para comprender los retos a los que se enfrentan los profesionales del área de las redes de comunicación

Dentro del área SDN, se indagó sobre el estado del arte de algunos de los controladores más populares del momento: Open Networking Operating System (ONOS), Open Daylight, RYU y Floodlight. De cada uno de ellos se estudiaron sus principales características, arquitectura, últimas versiones, y sobre todo, las posibilidades que ofrecen cada uno de ellos para envíos en multicast.

Basado en el estado del arte de los controladores SDN estudiados, se seleccionó la versión 1.12.0 del controlador ONOS. Se escogió esta versión ya que, para el momento que se comenzó con el desarrollo de este proyecto, era más estable que la última versión disponible (1.13.1). ONOS cuenta con una vasta documentación, una amplia comunidad

de desarrolladores y un módulo robusto que soporta multicast, que si bien sigue en pleno desarrollo, fue lo suficientemente sólido para ejecutar todas las actividades previstas. Resulta necesario destacar que el hecho de contar con conocimientos previos relacionados con ONOS, producto de prácticas de laboratorios impartidas durante este Máster, también fue un factor influyente durante la selección del controlador.

En lo que a la parte experimental del proyecto concierne, se diseñó y construyó, mediante la herramienta VNX, una topología de red virtualizada formada por los elementos descritos en el capítulo 4. Cada uno de los elementos de la red contaba con el software necesario para realizar todas las pruebas pertinentes. Mediante el uso de las herramientas "ping" se comprobó que los elementos de la red se podían comunicar entre sí.

Con el módulo Multicast Forwarding instalado, se creó una ruta o grupo multicast a través de la consola de ONOS. En dicho grupo, se indicó la dirección IP de la fuente (en este caso el "Servidor Web"), la dirección IP multicast a la cual van a suscribirse los clientes, y el identificador del switch y el puerto al que están conectados cada uno de los interesados en unirse al grupo multicast. Habiendo realizado esto, y con el usó de la herramienta iperf, se comprobó que había comunicación multicast en la topología virtualizada.

Mediante las herramientas GPAC y FFMPEG se transcodificó el video de pruebas a un bitrate de 2Mbps, se segmentó y se procedió a generar el fichero de manifiesto MPD, que contiene la información referente a la única representación del video que fue producida. Este fichero MPD es sumamente importante porque es el que el cliente va a leer para saber que segmento de video tiene que solicitar a su proxy.

Se consiguió realizar el envío de ficheros en multicast con el uso de la herramienta UFTP, que en la capa de transporte usa el protocolo UDP. Si bien es cierto que la idea original de este proyecto era utilizar un protocolo pensado para la entrega de contenido multimedia como FLUTE o ROUTE, se presentaron un alto número de inconvenientes con las implementaciones software libre de estos protocolos.

Ante ficheros perdidos durante la transmisión en multicast, y de acuerdo a la propuesta del 3GPP, se ideó un mecanismo de recuperación de segmentos de video en unicast para reducir, o en su defecto evitar, el número de paradas durante la reproducción del video. Este mecanismo se basó en la modificación del código JavaScript del cliente DASH, que permite modificar la petición HTTP ante un error 404.

Con todas las soluciones ideadas, se procedió a simular un escenario de streaming bajo demanda. Se inició el envío de ficheros en multicast desde el "Servidor Web" y el cliente DASH procedió a consumirlos desde su Proxy. Cuando fue necesario, entró en

acción el mecanismo de recuperación de segmentos de video en unicast, el cual funcionó de manera satisfactoria.

Se procedió a simular un escenario de streaming en vivo, sin embargo, se presentó un alto número de inconvenientes que impidieron que este ejercicio se desarrollara de la forma en que se pensó originalmente. Los problemas más limitantes fueron el no poder modificar el grupo multicast en ONOS "en caliente" y que el controlador no determine de manera automática cuando cambiar de transmisiones unicast a multicast. Estas limitantes son inherentes a la etapa de desarrollo en la que se encuentra ONOS.

Al finalizar este trabajo, se pudo constatar que a pesar de las limitaciones inherentes a los recursos utilizados, es posible aprovechar las bondades ofrecidas tanto por SDN como por DASH para distribuir contenido multimedia desde un servidor a diversos clientes, tanto en la modalidad on demand, como en vivo, de manera que se puedan aprovechar de manera óptima los recursos de la red mediante el envío en multicast de los ficheros de video.

6.1 Futuras líneas de investigación

Tomando como base las ideas conceptuales, los procedimientos seguidos y los inconvenientes que se presentaron durante el desarrollo de este Trabajo Fin de Máster, se desprenden varias motivaciones futuras. En primer lugar, se podría estudiar la distribución multicast de contenido multimedia en un entorno SDN, pero implementando un controlador distinto de ONOS, con el que se pueda interactuar a través de una API REST para así poder emular escenarios más complejos.

Seguidamente, y considerando que el mecanismo de recuperación de segmentos multimedia en unicast que se ideó en este Trabajo Fin de Máster consistió en la adaptación del código del cliente DASH a nuestra topología, se propone desarrollar otro mecanismo que permita mayor flexibilidad y adaptabilidad ante la pérdidas de ficheros en unicast.

Una mejora que permitiría obtener mejores resultados sería instalar un escenarío con elementos físicos, especialmente si se considera que el objetivo fundamental de esta investigación era la exploración de la intersección de las tecnologías DASH, SDN y multicast en la distribución de contenido multimedia, y que por esta misma razón se trabajó en una topología de red virtualizada.

7 Bibliografía

[1] D. Kreutz, F. Ramos, P. Veríssimo, C. Rothenberg, S. Azodolmolky, S. Uhlig, "Software-Defined Networks: A Comprehensive Survey", Proceeding of the IEEE, Volume 103, Issue 1. Disponible en: https://ieeexplore.ieee.org/document/6994333.

[2] "What is OpenFlow and how it relates to SDN", https://www.sdxcentral.com/sdn/definitions/what-is-openflow/, [Online], 2017.

[3] "Introducing ONOS – a SDN network operating system for Service Providers", http://onosproject.org/wp-content/uploads/2014/11/Whitepaper-ONOS-final.pdf, [Online], 2014.

[4] "ONOS Project", https://onosproject.org/, [Online].

[5] A. Campanella, A. Dandoush, L. Manassakis, "Open Network Operating System", 2017.

[6] "Release model", https://wiki.onosproject.org/display/ONOS/Release+Model, [Online].

[7] J. Kleinrouweler, S. Cabrero, P. Cesar, "Delivering Stable High-Quality Video: An SDN Architecture with DASH Assisting Network Elements", 2016.

[8] J. Kua, G. Armitage, P. Branch, "A Survey of Rate Adaptation Techniques for Dynamic Adaptive Streaming over HTTP", IEEE Communications Survey & Tutorials, Volume 19, Issue 3, 2017.

[9] C. Lentisco, L. Bellido, E. Pastor, "Reducing Latency for Multimedia Broadcast Services Over Mobile Networks", IEEE Transactions on Multimedia, Volume 19, Issue 1, 2017.

[10] "What are SDN Controllers (or SDN controllers platform", https://www.sdxcentral.com/sdn/definitions/sdn-controllers/, [Online], 2017.

[11] "Open Daylight Platform Overview", https://www.opendaylight.org/what-we-do/odl-platform-overview, [Online].

[12] "RYU SDN framework", https://osrg.github.io/ryu/, [Online].

[13] I. Domínguez Martínez-Casanueva, "Distribución multicast de vídeo en diecto sobre la red móvil LTE", 2015.

[14] J. Marzo Icaza, "Diseño de escenarios virtuales de distribución de contenido multimedia con soporte de redes definidas por software", 2016.

[15] "Apache HTTP Server Project", https://httpd.apache.org/ABOUT_APACHE.html, [Online].

[16] What is iperf/iperf3", https://iperf.fr/, [Online].

[17] "ONOS Multicast Forwarding Architecture", https://wiki.onosproject.org/display/ONOS/ONOS+Multicast+Forwarding+Architecture, [Online], 2015.

[18] "Apache Karaf Container Documentation" http://karaf.apache.org/manual/latest/#_overview, [Online].

[19] C. Koch, S. Hacker, D. Hausheer, "VoDCast: Efficient SDN.based Multicast for Video on Demand", 2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Network (WoWMoM)", 2017.

[20] A. Menabo, "Evaluating SDN and SDN-based Multicast for Network Intensive Services in UNINETT", 2015.

[21] "Virtual Networks Over Linux Main Page", http://web.dit.upm.es/vnxwiki/index.php/Main_Page, [Online].

[22] "GPAC Multimedia open source project", https://gpac.wp.imt.fr/home/gpac-features/, [Online].

[23] "About FFMPEG", https://www.ffmpeg.org/about.html, [Online].

[24] "Dahs Industry Forum projet repository", https://github.com/Dash-Industry-Forum/dash.js/wiki, [Online], 2018.

[25] "ATSC_ROUTE Project repository and documentation", https://github.com/waqarz/ATSC_ROUTE, [Online], 2017.

[26] J. R¨uckert, J. Blendin, R. Hark, and D. Hausheer, "Flexible, Efficient, and Scalable Software-Defined Over-the-Top Multicast for ISP Environments with DynSDM," IEEE Transactions on Network and Service Management (TNSM), Vol. 99, pp. 1–14, 2016.

[27] "Big Buck Bunny", https://peach.blender.org/, [Online].

[28] "OpenFlow overview", http://flowgrammable.org/sdn/openflow/#tab_protocol, [Online], 2018.

[29] W. Stallings, "Software Defined Networks and OpenFlow", The Internet Protocol Journal, Volume 16, No. 1, https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-59/161-sdn.html, [Online], 2013.

[30] S. Islam, N. Muslim, J. Atwood, "A Survey on Multicasting in Software-Defined Networking", IEEE Communications surveys & Tutorials, Volume 20, Issue 1, 2018

[31] "RYU Libraries", https://ryu.readthedocs.io/en/latest/components.html#libraries, [Online].

[32] "RYU application programming model", https://ryu.readthedocs.io/en/latest/ryu_app_api.html,[Online].

[33] T.Paila, M.Luby, R,Lehtonen, V. Roca, R.Walsh, "File Delivery Over Unidirectional Transpor documentation", https://www.ietf.org/rfc/rfc3926.txt,[Online], 2004.

[34] "Introduction to Real-time Transport Protocol", https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Intro_to_RTP, [Online].

[35] "What is IGMP and how does it work?" https://www.dell.com/support/article/es/es/esbsdt1/sln68773/what-is-igmp-and-how-does-it-work-technical-tip-132521?lang=en, [Online].

[36] "IGMP", https://ryu.readthedocs.io/en/latest/library_packet_ref/packet_igmp.html?highlight=multicast, [Online].

[37] "Floodlight – The controller", https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343548/The+Controller, [Online].

[38] "Flodlight Simple Multicast project repository", https://github.com/hexec/floodlight-simple-multicast,[Online], 2017.

[39] "Multicas SDN application using Floodlight project repository", https://github.com/daniel666/multicastSDN, [Online], 2014.

[40] "Open DayLight documentation",https://www.opendaylight.org/what-we-do/current-release/oxygen, [Online].

[41] "Open DayLight Controller: Architectural Framework", https://wiki.opendaylight.org/view/OpenDaylight_Controller:Architectural_Framework, [Online].

[42] "Open DayLight OpenFlow Plugin: End to End groups", https://wiki.opendaylight.org/view/OpenDaylight_OpenFlow_Plugin:End_to_End_Groups,[Online].

[43] "MAD Project's Home Page", http://mad.cs.tut.fi/index.html, [Online], 2007.

[44] "UFTP – Encrypted UDP based FTP with multicast", http://uftp-multicast.sourceforge.net/, [Online]

[45] "Control de acceso HTTP", https://developer.mozilla.org/es/docs/Web/HTTP/Access_control_CORS, [Online].

[46] "ONOS Multicast use case", https://wiki.onosproject.org/display/ONOS/Multicast+Use+Case, [Online], 2015.

[47] D. Fernandez, R. Alvarez, Script para la creación de rootfs personalizados VNX LXC a partir de un rootfs VNX LXC básico, 2018.

8 Anexos

En esta sección se encuentran documentados los errores más importantes que se presentaron durante el desarrollo de este proyecto, así como los scripts diseñados para la automatización de las actividades descritas en esta memoria.

8.1 Problemas encontrados durante la compilación e instalación del ATSC_ROUTE

Durante la compilación del código e instalación del software del proyecto ATSC_ROUTE se presentaron un alto número de inconvenientes, donde la mayoría de ellos no se pudieron resolver. En primer lugar, ambas interfaces web presentan funcionalidad nula. En el caso de la GUI del módulo *Sender* no funciona ni el combo box que permite seleccionar la interfaz de red por donde se va a realizar el envío de datos, ni el botón que permite guardar la configuración, por lo tanto no fue posible realizar ninguna acción por esta vía.

Seguidamente se procedió a modificar los ficheros de configuración correspondientes. Una vez realizados los cambios pertinentes, se intentó compilar el código. Sin embargo, tal y como se muestra en la figura 41, las trazas de la compilación arrojaban errores y por ende el proceso no se completaba de manera satisfactoria.

```
Makefile:13: recipe for target 'flute.o' failed
make[1]: *** [flute.o] Error 1
make[1]: se sale del directorio '/var/www/html/Work/Route_Sender/flutelib'
Makefile:18: recipe for target 'flutelib' failed
make: *** [flutelib] Error 2
root@Servidor-Web:/var/www/html/Work/Route_Sender#
```

Figura 41. Fragmento de traza durante la compilación del módulo "Sender"

Analizando las trazas se pudo observar que las librerías *flutelib*, *flute* y *multisflute*, que conforman este módulo, estaban causando problemas. Durante la compilación de la primera, como se aprecia en la imagen 42 se arrojaron una serie de errores relacionados con un conjunto de variables no declaradas y tipos de datos desconocidos en el archivo *flute.c*, el cual se encuentra en este directorio.

```
flute.c:1232:33: error: 'CURLOPT_ERRORBUFFER' undeclared (first use in this function)
   code = curl_easy_setopt(curl, CURLOPT_ERRORBUFFER, errorBuffer);
                                 ^~~~~~~~~~~~~~~~~~
flute.c:1234:14: error: 'CURLE_OK' undeclared (first use in this function)
   if(code != CURLE_OK) {
              ^~~~~~~~
flute.c:1242:4: warning: implicit declaration of function 'curl_easy_cleanup' [-Wimplicit-function-declaration]
    curl_easy_cleanup(curl);
    ^~~~~~~~~~~~~~~~~
flute.c:1246:33: error: 'CURLOPT_USERAGENT' undeclared (first use in this function)
   code = curl_easy_setopt(curl, CURLOPT_USERAGENT, "mbms-rel6-FLUTE-repair/0.1");
                                 ^~~~~~~~~~~~~~~~~
flute.c:1335:34: error: 'CURLOPT_URL' undeclared (first use in this function)
    code = curl_easy_setopt(curl, CURLOPT_URL, apd->bmFileRepair->sessionDescriptionURI);
                                  ^~~~~~~~~~~
flute.c:1350:34: error: 'CURLOPT_WRITEFUNCTION' undeclared (first use in this function)
    code = curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, write_to_buffer);
                                  ^~~~~~~~~~~~~~~~~~~~~
flute.c:1368:34: error: 'CURLOPT_WRITEDATA' undeclared (first use in this function)
    code = curl_easy_setopt(curl, CURLOPT_WRITEDATA, (void*)&chunk);
                                  ^~~~~~~~~~~~~~~~~
flute.c:1383:11: warning: implicit declaration of function 'curl_easy_perform' [-Wimplicit-function-declaration]
    code = curl_easy_perform(curl);
           ^~~~~~~~~~~~~~~~~
```

Figura 42. Errores durante la compilación de la librería flutelib.

Las otras dos librerías que presentaron problemas durante la compilación, arrojaron exactamente el mismo error (figura 43). Este error está relacionado con la ausencia de dos ficheros específicos: *lflutelib* y *lexpat*

```
root@coit-vm:/home/coit/Desktop/Prueba2-TFM/ATSC_ROUTE-master/Work/Route_Sender/flute# make VERBOSE=1
gcc -o ../bin/flute_sender main.o -L../lib -lflutelib -lalc -lsdp -lexpat -lpthread -lm -lz -lcurl -L/usr/local/ssl/lib -lcrypto
/usr/bin/ld: cannot find -lflutelib
/usr/bin/ld: cannot find -lexpat
collect2: error: ld returned 1 exit status
Makefile:23: recipe for target 'flute' failed
make: *** [flute] Error 1
root@coit-vm:/home/coit/Desktop/Prueba2-TFM/ATSC_ROUTE-master/Work/Route_Sender/flute#
```

Figura 43. Errores durante la compilación de las librerías flute y multisflute.

Igualmente, al compilar el módulo *Receiver* se observaron errores de compilación similares a los arrojados durante la compilación del módulo *Sender*: la librería *multi sesión flute application* no consigue los ficheros *lflutelib*, *lexpat* (figura 44).

```
-----------------------------------------
*** Example multisession FLUTE application ***
-----------------------------------------
mkdir bin; cd multis_flute; make clean; make
mkdir: cannot create directory ‘bin’: File exists
make[1]: Entering directory '/home/coit/Desktop/Prueba2-TFM/ATSC_ROUTE-master/Work/Route_Receiver/multis_flute'
rm -f ../bin/multis_flute *.o *~
make[1]: Leaving directory '/home/coit/Desktop/Prueba2-TFM/ATSC_ROUTE-master/Work/Route_Receiver/multis_flute'
make[1]: Entering directory '/home/coit/Desktop/Prueba2-TFM/ATSC_ROUTE-master/Work/Route_Receiver/multis_flute'
gcc -c -Wall -O3 -g -DLINUX -D_LARGE_FILE_API -D_LARGEFILE64_SOURCE -D_ISOC99_SOURCE -D_XOPEN_SOURCE=500 -I../flutelib -I/usr/local/ss
l/include -o main.o main.c
main.c: In function ‘start_flute_process’:
main.c:296:7: warning: variable ‘retval’ set but not used [-Wunused-but-set-variable]
   int retval = 0;
       ^~~~~~
gcc -o ../bin/multis_flute main.o -L../lib -lflutelib -lalc -lsdp -lexpat -lpthread -lm -lz -lcurl -L/usr/local/ssl/lib/ -lcrypto -L..
/libwebsockets-master/build/lib -lwebsockets
/usr/bin/ld: cannot find -lflutelib
/usr/bin/ld: cannot find -lexpat
collect2: error: ld returned 1 exit status
Makefile:21: recipe for target 'multis_flute' failed
make[1]: *** [multis_flute] Error 1
make[1]: Leaving directory '/home/coit/Desktop/Prueba2-TFM/ATSC_ROUTE-master/Work/Route_Receiver/multis_flute'
Makefile:32: recipe for target 'multis_flute' failed
make: *** [multis_flute] Error 2
root@coit-vm:/home/coit/Desktop/Prueba2-TFM/ATSC_ROUTE-master/Work/Route_Receiver#
```

Figura 44. Errores durante la compilación del módulo “Receiver”.

Es importante resaltar que en el repositorio del proyecto se provee escasa documentación sobre la arquitectura del software, guía de configuración, instalación, compilación, etc. Por lo tanto, los errores mencionados no pudieron ser corregidos.

La compilación del programa si se puede realizar de manera satisfactoria omitiendo las librerías que presentan fallos. Sin embargo, el mismo seguía sin presentar funcionalidad alguna.

8.2 Script installOpenSSL

```
#!/bin/bash
echo "Starting the configuration of OpenSSL"
sudo mv /home/openssl-1.0.2o.conf /etc/ld.so.conf.d/
echo "Reloading the dynamic link"
sudo ldconfig -v
echo "Configuring the binary"
mv /usr/bin/c_rehash /usr/bin/c_rehash.BEKUP
mv /usr/bin/openssl /usr/bin/openssl.BEKUP
echo "Modifying PATH"
sed -e 's|/usr/local/ssl/bin:||g' -i /etc/environment
```

```
sed -e 's|PATH="\(.*\)"|PATH="/usr/local/ssl/bin:\1"|g' -i /etc/environment
source /etc/environment
echo "Checking configuration is succesful"
which openssl
echo "----------------------------------------------------------------------"
echo "Compiling and installing UFTP"
cd /usr/local/src/uftp-4.9.8
sudo make clean
sudo make
sudo make install
```

Este script se ejecuta automáticamente al iniciar el escenario virtual, y las acciones que realiza es instalar y configurar la herramienta OpenSSL para luego compilar e instalar el software UFTP

8.2.1 Scripts de envío en escenario VoD

```
#!/bin/bash
COUNTER=1
PATH=/var/www/html/video/segments/
FOURTH_BASE=dash_videBBBrep4_2M_
MPD=videoBBB.mpd
INIT=videoBBB_init.mp4
/usr/bin/uftp -M 230.4.4.1 -P 230.4.4.1 $PATH$MPD $PATH$INIT
while [  $COUNTER -lt 325 ]; do
    if [ $COUNTER -ne $(($RANDOM%324)) ];
    then
        /usr/bin/uftp -M 230.4.4.1 -P 230.4.4.1 -R 4000 $PATH$FOURTH_BASE$COUNTER.m4s
        /usr/lib/klibc/bin/sleep 1.5
    else
        echo No se envia el fichero $PATH$FOURTH_BASE$COUNTER.m4s

    fi
let COUNTER=COUNTER+1
done
```

Como se puede observar, al inicio del script hay una condición que permite que se realice el envío de ficheros siempre que la variable COUNTER sea menor a 325. Esto se debe a que la duración del video es de aproximadamente de 10 minutos con 40 segundos. Si se considera que cada segmento tiene una duración de 2 segundos obtenemos un total de 324 segmentos. Seguidamente, se agrega otra condición la cual permite que se realice el envío de los ficheros de video cuyo identificador sea diferente a un número aleatorio entre 1 y 324. Esto se hace con el propósito de forzar que no se envíe algún fichero, y así simular la pérdida de segmentos multimedia, garantizando así que en algún momento

de la reproducción del video, entre en acción el mecanismo de recuperación de segmentos.