Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

DISEÑO DE ESCENARIOS VIRTUALES DE DISTRIBUCIÓN DE CONTENIDO MULTIMEDIA CON SOPORTE DE REDES DEFINIDAS POR SOFTWARE

TRABAJO FIN DE MÁSTER

José Andrés Marzo Icaza

2016

Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

Máster Universitario en
Ingeniería de Redes y Servicios Telemáticos

TRABAJO FIN DE MÁSTER

DISEÑO DE ESCENARIOS VIRTUALES DE DISTRIBUCIÓN DE CONTENIDO MULTIMEDIA CON SOPORTE DE REDES DEFINIDAS POR SOFTWARE

Autor
José Andrés Marzo Icaza

Director
Luis Bellido Triana

Departamento de Ingeniería de Sistemas Telemáticos

2016

Resumen

Entre las aplicaciones usadas en internet, el video es uno de los servicios con mayor crecimiento en los últimos años. El crecimiento de servicios de streaming bajo demanda es una prueba del crecimiento de este sector. Para brindar la mejor calidad a los usuarios, es necesario el estudio y desarrollo de mejores protocolos enfocados a brindar una mejor calidad de experiencia (QoE) al usuario. De la misma manera, los recursos de red disponibles no son ilimitados, por lo tanto los protocolos de streaming deben ser diseñados de tal manera que se optimicen los recursos de red.

El streaming dinámico adaptativo sobre HTTP (DASH) ha sido estandarizado luego de juntar varias iniciativas, y es ampliamente usado por los proveedores de contenido multimedia. Es un sistema en que se le da toda la inteligencia al cliente y se utiliza un algoritmo de adaptación para solicitar la calidad de video dependiendo de ciertos parámetros, muchos de los cuales dependerán de la red utilizada y del cuello de botella que puedan encontrarse. Por lo tanto, la elección de la tasa de bits que se debe solicitar al servidor, tendrá un impacto en la calidad percibida por el usuario.

Las redes definidas por software (SDN) son un nuevo paradigma que se diferencia de las redes tradicionales en que se separa el plano de control del plano de datos de los equipos de red como switches y routers. Se tiene un equipo centralizado llamado controlador y además es posible el desarrollo de aplicaciones de red que puedan mejorar y optimizar las redes y servicios existentes.

Se propone en este trabajo de fin de máster, el uso de las tecnologías de las redes definidas por software para estudiar el comportamiento de la distribución de contenido multimedia utilizando principalmente DASH. Además, se utilizarán las aplicaciones disponibles con las SDN para usarlas en conjunto a los clientes de streaming y al algoritmo de adaptación.

Se utilizará la herramienta VNX (Virtual Network over Linux), para que en un esquema virtualizado, se puedan montar escenarios que nos permitan estudiar el compartimiento de la distribución de contenido multimedia, el streaming usando DASH, y el uso de redes definidas por software. Se espera al final de este trabajo evaluar el impacto que se tiene en las métricas de QoE cuando existe congestión en la red, ya sea por tráfico de otras aplicaciones o ya sea por clientes distintos de streaming multimedia.

Palabras Clave: Streaming, video, DASH, SDN, OpenFlow, QoE

Abstract

Among the applications we used on the Internet, video streaming is one of the fastest growing services in recent years. The growth of demand streaming services is testament to the growth of this sector. To provide the best quality to users, the study and development of improved protocols aimed at providing a better quality of experience (QoE) to the user is required. Similarly, the available network resources are limited; therefore streaming protocols should be designed so that network resources are optimized.

Dynamic Adaptive Streaming over HTTP (DASH) was standardized after put together several initiatives, and is widely used by multimedia content providers. It is a system that gives the intelligence to the client and it uses an adaptive algorithm to request the video quality according on certain parameters, many of which depend on the network used and the bottleneck that can be found. Therefore, the choice of the segment bitrate requested to the server, will have an impact on the quality perceived by the user.

Software defined Networks (SDN) are a new paradigm that differs from traditional networks in that the control plane and data plane of the network equipment such as switches and routers are separated. It has a centralized element called the controller and it is possible the development of network applications that can improve and optimize existing networks and services.

It is proposed in this project the use of technologies of software defined networks to study the behavior of content distribution multimedia using DASH. In addition, applications available will be used with SDN for use in conjunction to streaming clients and the adaptive algorithm.

The VNX (Virtual Network over Linux) tool will be used in a virtualized scheme so we can mount scenarios to allow us to study the content distribution multimedia, streaming media using DASH, and the use of software defined networks. It is expected at the end of this project to assess the impact that we have on QoE metrics when there is congestion on the network, either by traffic of other applications or either by another clients streaming at the network.

Key Words: Streaming, video, DASH, SDN, OpenFlow, QoE

Índice general

Resumen i

Abstract iii

Índice general v

Índice de figuras ix

Siglas xi

1 Introducción 1

1.1 Contexto 1

1.2 Objetivos 2

1.3 Estructura del documento 3

2 Streaming Multimedia 5

2.1 Streaming de video bajo demanda 6

2.1.1 Streaming usando UDP 6

2.1.1.1 Streaming HTTP 7

2.1.2 Streaming Adaptivo sobre HTTP 9

2.2 Redes de Distribución de Contenidos 13

2.3 Streaming de video usando Multicast 15

2.3.1 Casos de uso Broadcast/Multicast 15

2.3.2 FLUTE/DASH 16

2.3.3 Handover entre Streaming Broadcast/Multicast y Unicast 17

2.4 Parámetros a considerar para el streaming Multimedia 18

2.4.1 Calidad de Servicio en la red 19

2.4.2 Estrategias para mejorar la calidad de experiencia 19

3 Redes definidas por Software 24

3.1 Características de las Redes Definidas por Software 24

3.1.1 Definición y Arquitectura de las SDN 24

3.1.2 OpenFlow 27
3.1.3 Controladores 32
3.1.4 Aplicaciones 34
3.2 Switches basados en Software 36
3.3 Controlador Floodlight 38
3.3.1 Aplicaciones Floodlight 38
3.3.2 Obtener estadísticas de la red usando el controlador 40
4 Diseño de escenarios para streaming de video usando SDN 42
4.1 Escenarios virtuales 42
4.1.1 Hipervisores 42
4.1.2 VNX 43
4.2 Descripción elementos principales 44
4.2.1 Configuración de los hosts 45
4.2.2 Escenario de pruebas utilizando un solo cliente 52
4.2.3 Escenario de pruebas utilizando varios clientes 53
4.2.4 Escenario de pruebas usando diferentes caminos de red 54
4.3 Conectividad del escenario 55
4.3.1 Pruebas de conectividad 55
4.3.2 Calculo de estadísticas 59
4.4 Algoritmo de adaptación de acuerdo a estadísticas obtenidas usando SDN 61
5 Experimentación y análisis de resultados 64
5.1 Configuración de experimentos 64
5.1.1 Configuración experimentos usando un solo cliente 64
5.1.2 Configuración de experimentos usando varios clientes 65
5.1.3 Configuración de experimentos variando los caminos de red 66
5.2 Resultados 66
5.2.1 Un solo cliente sin congestión 66
5.2.2 Un solo cliente con Congestión variable 68
5.2.3 Un solo cliente con congestión total 71
5.2.4 Simulación Multicast/ Unicast con congestión variable 73

5.2.5 Resumen de los experimentos utilizando un cliente DASH 76
5.2.6 Resultado pruebas usando varios clientes DASH 77
5.2.7 Pruebas usando múltiples caminos de red 81
6 Conclusiones 84
Bibliografía 87

Índice de figuras

Figura 1. Uso de RTP y RTSP para el streaming UDP 7
Figura 2. Buffer del cliente en Streaming HTTP 8
Figura 3. Estándar DASH 9
Figura 4. Distribución de contenido Multimedia 10
Figura 5. Cliente y servidor DASH 11
Figura 6. Estructura del archivo MPD 12
Figura 7. Uso de CDNs en Netflix 14
Figura 8. Transmisión de contenido Multimedia usando red LTE en unicast y multicast 16
Figura 9. Arquitectura FLUTE/DASH 17
Figura 10. Esquema para experimentación usando FLUTE y DASH 18
Figura 11. Inequidad entre varios servicios de streaming compitiendo por recursos de la red 22
Figura 12. Comparación redes tradicionales y SDN 25
Figura 13. Arquitectura SDN 26
Figura 14. Ejemplo de una tabla de flujo OpenFlow 27
Figura 15. Flujo de mensajes OpenFlow 30
Figura 16. Proceso pipeling de un paquete 31
Figura 17. Componentes del controlador SDN 32
Figura 18. Componentes Open vSwitch 37
Figura 19. Esquema del controlador Floodlight 38
Figura 20. Tipos de virtualización 43
Figura 21. Despliegue de escenario en VNX 44
Figura 22. Escenario principal para pruebas de Streaming multimedia 45
Figura 23 . Reproductor DASH 46
Figura 24. Configuración del cliente en VNX 46
Figura 25. Esquema archivo MPD utilizado 47
Figura 26. Configuración del servidor en el escenario de VNX 48
Figura 27. Configuración de hosts para realizar pruebas de congestión 48
Figura 28. Configuración controlador en escenario VNX 49
Figura 29. Logs del controlador 50
Figura 30. Escenario de pruebas utilizando tres clientes 53
Figura 31. Escenario de pruebas usando diferentes caminos de red 54
Figura 32. Captura ICMP entre cliente y servidor 56

Figura 33. Tablas de flujo de cada switch 57
Figura 34. Captura de peticiones HTTP realizadas desde el cliente 58
Figura 35. Mensajes de Log durante la reproducción 58
Figura 36. Recolección de estadísticas del switch OpenvSwitch1 59
Figura 37. Algoritmo de adaptación de DASH tomando estadísticas de la red 63
Figura 38. Sin congestión: Calidad de video por segmento de video 66
Figura 39. Sin congestión: Distribución de calidades de video solicitados 67
Figura 40. Sin congestión: Nivel del buffer 67
Figura 41. Congestión variable: Calidad de video por segmento de video 68
Figura 42. Congestión variable: Distribución de calidades de video solicitadas 69
Figura 43. Congestión variable: Trafico video y de otros usuarios 69
Figura 44. Congestión variable: Nivel del buffer 70
Figura 45. Congestión variable: Estadísticas de cada puerto del switch 70
Figura 46. Congestión total: Calidad de video por segmento de video 71
Figura 47. Congestión total: Distribución de calidades de video solicitadas 72
Figura 48. Congestión total: Trafico video y de otros usuarios 72
Figura 49. Congestión variable: Nivel del buffer 73
Figura 50. Multicast/ Unicast: Calidad de video por segmento de video 74
Figura 51. Multicast/ Unicast: Número de segmentos solicitados 74
Figura 52. Multicast/ Unicast: Nivel del buffer 75
Figura 53. Calidad de video de acuerdo al ancho de banda recibido para tres clientes DASH 78
Figura 54. Número de segmentos recibidos para los tres clientes DASH 78
Figura 55. Trafico usado por los tres clientes DASH 79
Figura 56. Nivel del buffer para los tres clientes DASH 80
Figura 57. Calidad de video solicitada por dos diferentes caminos de red 81
Figura 58. Trafico de video y de otros usuario recibido 82
Figura 59. Estadísticas de red en switch Net0 83
Figura 60. Nivel del buffer al usar escenario de varios caminos de red 83

Siglas

HTTP	Hypertext Transfer Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
DASH	Dynamic Adaptive Streaming over HTTP
RTP	Real Time Protocol
RTSP	Real Time Streaming Protocol
SDN	Software defined Networks
QoS	Quality of service
QoE	Quality of Experience
ISP	Internet Service Provider
LTE	Long Term Evolution
e-MBMS	evolved – Multimedia Broadcast and Multicast system
EPC	evolved Packet Core
AL-FEC	Application Layered Forward Error Correction
FLUTE	File Delivery over Unidirectional Transport
FDT	File Delivery Table
VNX	Virtual Networks over Linux
REST	Representational state transfer
API	Application Programming Interface
CDN	Content Delivery Network
AVC	Advanced Video Coding
MPEG	Moving Picture Experts Group

1 Introducción

1.1 Contexto

La mejora de las redes de telecomunicaciones, ofreciendo mejores prestaciones como mayores tasas de datos, ha permitido el desarrollo, despliegue y mejora de nuevos servicios para dar un valor añadido al uso de estas redes. El consumo de video en internet no es una idea nueva, pero en los últimos años ha obtenido un mayor desarrollo y consumo gracias a servicios brindados por Youtube o Netflix por poner unos ejemplos. La mejora de los servicios de streaming en internet así como la mejora en las redes, son las que han logrado que estos servicios tengan éxito.

A pesar de las mejora en las redes, todavía se presenta congestión en el tráfico de estos servicios debido al aumento de usuarios. El tráfico de video es uno de los principales causantes de cuellos de botella en el internet. Por este motivo, muchos esfuerzos se han enfocado en optimizar el uso de los recursos disponibles usando los protocolos de streaming.

Los usuarios consumen los contenidos de dos maneras, ya sea contenido en vivo, como pueden ser transmisiones de eventos deportivos, o contenido bajo demanda, como es el caso de las películas o series de televisión. Es necesario entender cuáles son las limitantes de cada medio, y entender lo que el usuario espera al recibir estos servicios. Por lo tanto, no solo es necesario crear nuevos métodos de distribución de contenidos multimedia, sino que también es necesario entender como el usuario percibe esos servicios, y encontrar las métricas necesarias para mejorar los servicios existentes en función de la satisfacción del usuario final.

Cuando un usuario está observando un video por streaming en internet, muchas veces puede querer disfrutar el contenido lo más rápido posible, sin interrupciones así sea a una baja calidad. En cambio, pueden existir otros usuarios que deseen observar el contenido con la más alta calidad y puedan soportar tener algunas interrupciones en la reproducción con tal de observar los detalles que solo un video de alta calidad puede ofrecer. Por lo tanto, obtener estas métricas de calidad se vuelve muchas veces subjetivas, pero de todas maneras necesitan ser analizadas.

Otro punto que se debe analizar es si debe considerar que el usuario sea el que elija la calidad deseada, o hacer esta tarea de forma automática. Los métodos de streaming actuales permiten que se haga la adaptación de manera automática de acuerdo a las características de la red. Por lo tanto, las redes ocupan un pilar fundamental para el consumo de contenidos multimedia.

La calidad que va a percibir el usuario dependerá de la red de acceso usada. Varios proveedores implementan diversas soluciones para brindar una mejor calidad de servicio a sus usuarios. Los proveedores están conscientes de la importancia del video en internet, y saben que gran cantidad del tráfico que pase por sus redes dependerá de cuanto consuman contenido multimedia los usuarios. Soluciones como las redes de distribución de contenidos, aunque de propiedad de empresas externas a los proveedores, han ayudado a distribuir eficientemente el tráfico multimedia en las redes de los proveedores de internet. De la misma manera han ayudado al éxito de empresas como las anteriormente nombradas Youtube o Netflix, así como otros competidores como Hulu, Amazon, etc.

Los proveedores usan equipos de comunicaciones que realizan funciones específicas. Todas las funcionalidades están por lo general en una caja cerrada, sean switches, routers, etc. Un nuevo paradigma llamado *Software Defined Networking (SDN)* pretende eliminar esta limitante. El uso de las SDN permitirá tener redes más abiertas e interoperables. Como usar las ventajas ofrecidas por este nuevo paradigma, debe ser prioridad para la comunidad investigadora. Como se ha ido mencionando, las ventajas principales serán tener una mejor calidad percibida por los usuarios y uso más eficiente de los recursos de red.

Para desarrollar este trabajo de fin de master se consideran las tecnologías actuales de streaming de video, y además se usarán las tecnologías de redes definidas por software como alternativa a las redes tradicionales IP. De la misma manera se aprovecharán los nuevos avances en tecnologías de virtualización para poder estudiar cómo funciona la distribución de contenido y poder realizar escenarios de pruebas.

1.2 Objetivos

Los objetivos que se esperan cumplir al finalizar este trabajo de fin de máster son los siguientes:

1. Analizar las técnicas usadas para la distribución de contenido multimedia, especialmente del streaming de video, analizando cómo ha sido su evolución hasta llegar a los estándares que tenemos actualmente.
2. Analizar los parámetros que se deben considerar para brindar la mejor calidad a los usuarios. Se recolectará y analizará la información actualmente disponible en la literatura.
3. Descripción y análisis de las redes definidas por software, tomando especial atención a las aplicaciones que nos puedan ayudar a optimizar los recursos de red cuando se realiza distribución de contenido multimedia.
4. Implementación de escenarios para realizar experimentos que nos permitan analizar cómo funciona el streaming multimedia.

5. Utilizar las aplicaciones de las redes definidas por software para optimizar la entrega de contenido multimedia. Por medio de los experimentos diseñados se debe evaluar su correcto funcionamiento.

1.3 Estructura del documento

La memoria de este trabajo de fin de máster se enfocará primero en describir la parte teórica del problema a estudiar y de las herramientas a utilizar. El capítulo 2 realizará una descripción de los protocolos usados para el streaming multimedia. Se analizarán en menor detalle los protocolos usando UDP, para luego analizar con más detalle los protocolos de streaming basados en HTTP, los cuales serán la base para el desarrollo de este trabajo, especialmente del uso de algoritmos de adaptación usado en aplicaciones de streaming estandarizadas. Se analizarán las técnicas usadas como las redes de distribución de contenidos dando un ejemplo de como un popular servicio de streaming usa sus servicios.

De la misma manera que se analiza el caso de la entrega de contenido usando unicast, es decir cuando se envía el contenido desde un único emisor a un único receptor; también se lo analizará cuando se utiliza multicast, es decir cuando se envía el contenido a varios destinatarios. Dado que no es posible utilizar los protocolos de streaming basados en HTTP cuando se realiza multicast, en ese capítulo se analizan los protocolos usados para ese caso en particular.

Por último, se analizan los parámetros a considerar para mejorar la calidad percibida por el usuario. Se describen los parámetros de calidad de servicio en la red a considerar, así como una medida más idónea para analizar la calidad que percibe el usuario como es la calidad de experiencia (QoE).

En el capítulo 3 se describen los aspectos de las redes definidas por software y de la virtualización que serán usados en este trabajo. Se describe la arquitectura de las SDN y en que se diferencian de las redes IP tradicionales. Para comprender los aspectos principales de las SDN, se describe en este trabajo el protocolo OpenFlow, los controladores usados y su importancia y ventajas, y por último se describen las aplicaciones como el valor añadido que tienen las redes definidas por software.

Debido a que en este trabajo se utilizarán herramientas de virtualización para el desarrollo de los escenarios y experimentos, se describen las herramientas de virtualización más extendidas. Se analizan y enumeran los switches basados en software y que pueden ser usadas en sustitución de los switches basados en hardware, especialmente en ambientes de experimentación de redes y protocolos. En este trabajo se utilizará VNX (Virtual Networks over Linux) y se describe su funcionamiento en ese capítulo.

En el capítulo 4 se describen los escenarios realizados utilizando VNX. Se analizan varios escenarios, como puede ser un solo cliente, varios clientes, con congestión o sin congestión, unicast o multicast/unicast. Se describe cada elemento que interviene así como su configuración. En ese capítulo de describirá el diseño de un algoritmo de adaptación que funciona como un proxy a las peticiones del reproductor de video, y que además tiene acceso a las estadísticas generadas por el controlador SDN de las características de la red de acceso, en especial del cuello de botella.

Finalmente, en el capítulo 5 se muestran los resultados obtenidos luego de realizar cada uno de los experimentos preestablecidos. Las métricas que se describen en el capítulo 2 serán usadas para evaluar el algoritmo de adaptación usado para el streaming de video. Se debe poder concluir como afectan diversos parámetros como la congestión de la red, o que varios usuarios estén compartiendo recursos para visualizar algún contenido multimedia.

2 Streaming Multimedia

La importancia del video en internet debe ser considerada al analizar la cantidad de aplicaciones de streaming multimedia como video conferencias, IPTV o video en demanda. Estimaciones de Cisco indican que el consumo de tráfico de video IP será del 80% del total IP para el 2018 [1]. Por lo tanto, es necesario tener recursos de red y sobre todo aplicaciones para clientes que puedan proveer de alta Calidad de Experiencia (QoE) al usuario. Aplicaciones actuales como Youtube o Silverlight envían los paquetes de video sobre HTTP y utilizan un algoritmo de adaptación sobre el mismo.

Se analiza en este capítulo los principales métodos de streaming, analizando en detalle el algoritmo que está siendo adoptado como estándar actualmente, como es el caso del Streaming de Video Adaptativo sobre HTTP.

Cualquier aplicación de red multimedia usa video y audio. En el caso del video se tiene la particularidad de tener a característica de tener una alta tasa de bits. Esta tasa de bits variará dependiendo la calidad del video, teniendo tasas de bits bajas desde 100 kbps a tasas de bits más altas cercanas a 5000 kbps en el caso de video HD a 1080p. Las aplicaciones de video consumen el mayor ancho de banda de la red, por lo tanto es necesario considerar los requerimientos de alta tasa de bits. Este video puede ir comprimido, teniendo que elegir entre la calidad de video con la tasa de bits recibida. Para lograr la compresión se utilizan dos tipos de redundancia: la redundancia espacial y la redundancia temporal. La primera es la redundancia dada a una imagen en particular, tomando en cuenta que un video es un conjunto de imágenes. La segunda, representa las repeticiones observadas en las sucesiones respectivas de las imágenes. De esta manera se logra comprimir un video a prácticamente cualquier tasa de bits deseada. Esta particularidad, es de gran ayuda para tener varias copias del mismo video pero de diferentes calidades, cada uno a una tasa de bits diferente. Esta característica en especial permite al usuario seleccionar la calidad de video dependiendo de la red de acceso usada.

Las aplicaciones multimedia pueden agruparse en tres grandes categorías, cada una de las cuales requiere un análisis diferenciado:

1. Streaming de audio y video en demanda
2. Voz y Video sobre IP
3. Streaming de audio y video en vivo.

Voz y Video sobre IP se refiere a las conversaciones en tiempo real utilizando el internet. Esta aplicación puede verse como el sustituto de las clásicas redes de voz

conmutadas por circuitos. Compañías de Internet como Skype y varios proveedores de telecomunicaciones se encargan de brindar este servicio. Los atributos importantes en este caso son la sincronización y la tolerancia a la perdida de datos. Voz y Video sobre IP es muy sensible al retardo. La sincronización es importante para la voz al ser estas aplicaciones dedicadas a conversaciones sensibles al retardo.

En cambio, el streaming en vivo multimedia vendría a reemplazar a los sistemas tradicionales de broadcast de radio y televisión. Para utilizar este esquema se puede usar tecnologías de IP Multicast (basados en la capa de red) o usar esquemas de Multicast basados en la capa de aplicación como P2P o CDNs.

El streaming de multimedia bajo demanda combina tanto el video y el audio. Ambos son parecidos, pero el streaming de audio utiliza tasas de bits más bajas. En este caso se tiene contenido multimedia que ha sido previamente almacenado en algún o algunos servidores. Por lo tanto se tiene una comunicación cliente-servidor para poder obtener el contenido deseado. Este es el esquema usado por aplicaciones populares de internet como Youtube o Netflix. Para propósitos de este documento, este es el esquema que se analizará.

2.1 Streaming de video bajo demanda

El streaming se refiere a la técnica que nos permite empezar la reproducción del video unos segundos después de recibir el video del servidor, de esta manera reproducirá cierta parte del video y al mismo tiempo realizará las solicitudes al servidor de las partes que siguen del video. La reproducción del video es interactiva, es decir puede pausar, adelantar, retroceder o escoger un instante de tiempo preciso para la reproducción. Es necesario que se evite congelamientos de la imagen, es decir que la reproducción debe ser continua. Los tipos de streaming que pueden ser utilizados son: UDP Streaming, HTTP Streaming y Streaming adaptativo basado en HTTP. Los sistemas actuales usan los dos últimos mencionados.

2.1.1 Streaming usando UDP

Usando streaming basado en UDP, el servidor transmite el video a una tasa que coincide con la tasa de consumo de video en el cliente al registrar los trozos de video sobre UDP a una tasa estable. El servidor puede poner los paquetes en la red a la tasa de consumo del video sin las restricciones que da el protocolo de control de congestión de TCP. Antes de pasar los trozos de video a UDP el servidor encapsula los trozos en paquetes de video como RTP (*Real Time Transport Protocol*).

Es posible agregar características de control utilizando el protocolo RTSP (*Real Time Streaming Protocol*). Este protocolo ofrece ventajas como capacidades multi-servidor, posibilidad de añadir nuevos parámetros y utiliza mecanismos de seguridad web. En

la Figura 1 se realiza una descripción de su funcionamiento y como se usa en conjunto RTP y RTSP para el streaming. Es necesario utilizar RTSP para realizar una conexión con un servidor web y para enviar comandos de control. Para la reproducción se utiliza el protocolo RTP.

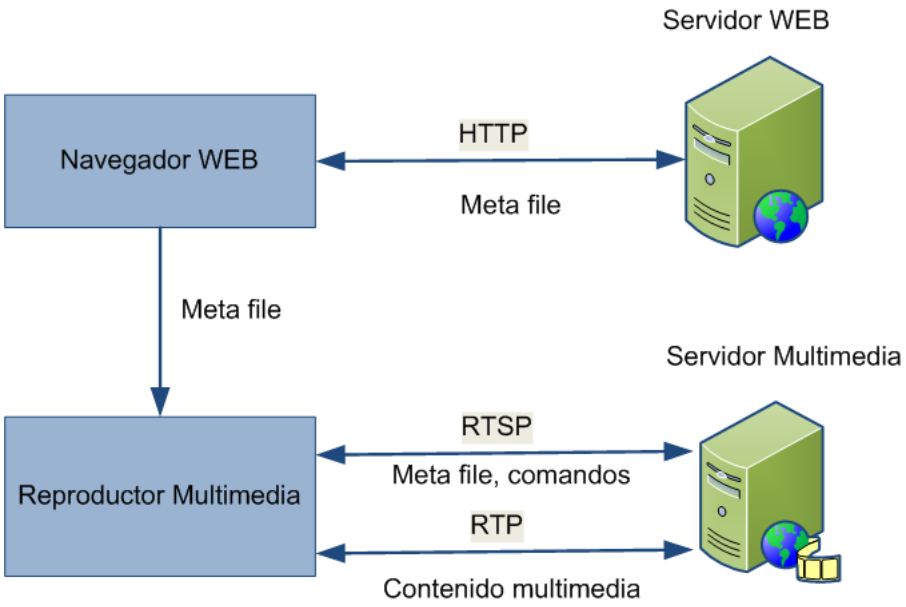


Figura 1. Uso de RTP y RTSP para el streaming UDP

Entre los inconvenientes encontrados se tiene:

- No es posible el envío constante de datos al ser imprevisible el canal disponible.
- Se requiere un servidor RTSP para hacer un seguimiento al estado del cliente.
- Muchos firewalls bloquean el paso de UDP.

2.1.1.1 Streaming HTTP

Los clientes pueden tolerar un retardo desde que solicitan el video hasta que empieza la reproducción del mismo. Además, puede tener una reserva en el buffer de la aplicación. Durante la sesión de streaming el contenido de video se transmite en dos fases. La primera es una fase de buffering y la segunda es la fase en estado estable [2]. En la fase de buffering la tasa de reproducción del video solicitado está limitada por el ancho de banda disponible. El reproductor de video comienza a reproducir cuando tiene suficientes datos disponibles en el buffer. En la fase de estado estable, la tasa promedio de descarga es ligeramente superior que la tasa de bits del video.

En el streaming HTTP se tiene la ventaja con respecto al streaming UDP en que se puede pasar cualquier tipo de firewall o NAT al pasar todo usando el protocolo HTTP de la capa de aplicación. El contenido multimedia es almacenado como si fuera un

archivo adicional con su respectivo URL. Si el usuario desea ver un video, realiza una petición GET para la URL respectiva. El servidor envía el contenido dentro de un mensaje de respuesta HTTP tan pronto como el control de congestión y el control de flujo de TCP lo permitan. El cliente recibe el video y lo almacena en el buffer hasta que pase cierto umbral y comienza a reproducirlo.

El cliente puede intentar descargar el video a una tasa más alta que la tasa que está usando. Por lo tanto, estaría realizando una captura previa, o un *prefetching* de las tramas de video y almacenarlas en un buffer. De esta manera los mecanismos para evitar la congestión de TCP intentarán usar todo el ancho de banda disponible entre el cliente y el servidor. En la capa de transporte TCP, una ventana de congestión es usada para controlar los flujos de datos insertados en la red. El tamaño de la ventana de congestión es ajustada dinámicamente de acuerdo al nivel de congestión en la red y al espacio de almacenamiento en el buffer del cliente TCP [3]. Este buffer no hay que confundirlo con el buffer del cliente, el cual depende del reproductor usado. En TCP, los datos son colocados en el buffer de envío TCP antes de ser transmitidos. Si este buffer está lleno, el servidor dejará de enviar más datos del video al socket. La aplicación del cliente lee los bytes del buffer receptor TCP a través del socket del cliente y coloca los bytes en el buffer del cliente de la aplicación .Un buffer en la aplicación del cliente indirectamente impone un límite en la tasa a la que puede enviarse el video desde el servidor al cliente al utilizar streaming sobre HTTP.

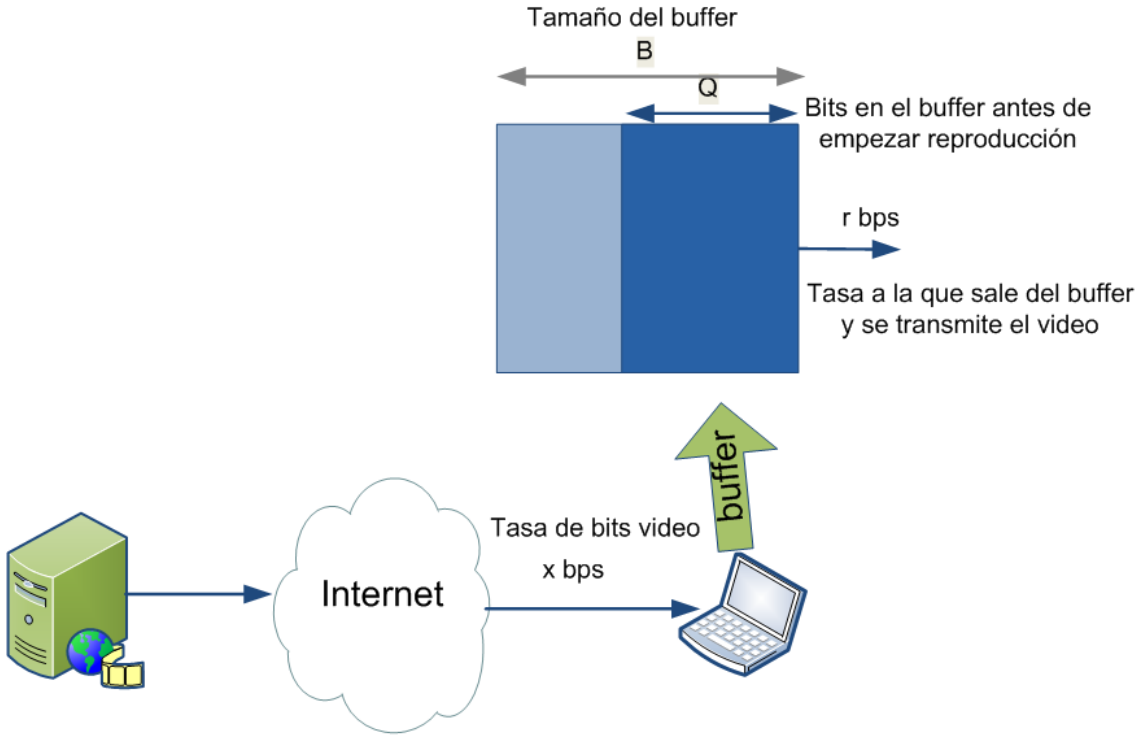


Figura 2. Buffer del cliente en Streaming HTTP

En la Figura 2 puede observarse un análisis del buffer del lado del cliente. Se muestra el tamaño del buffer de la aplicación del cliente **B (bits)** y el número de bits que deben ser puestos en el buffer antes de empezar la reproducción **Q (bits)**. Se transmite el video a una tasa **r (bps)**. El servidor envía bits a una tasa constante de **x (bps)**. Entonces de determina el tiempo en que empieza la reproducción $\mathbf{t_p}$ y el tiempo en que el buffer del cliente se llena $\mathbf{t_f}$.

$t_p = {}^{Q}/_{x}$ (Tiempo que empieza la reproducción)

Si x < r → $\mathbf{t_f} = \infty$ (El buffer del cliente nunca se llena)

Para x > r, el tiempo para que el buffer tenga Q bits es ${}^{Q}/_{x}$ segundos. El tiempo para añadir los restantes (B-Q) segundos es de ${}^{B-Q}/_{x-r}$ segundos. Por lo tanto, el tiempo para que el buffer del cliente se llene viene dado por la siguiente ecuación:

$$t_f = \frac{Q}{x} + \frac{B-Q}{x-r}$$

2.1.2 Streaming Adaptivo sobre HTTP

En el caso del streaming sobre HTTP revisado anteriormente se tiene la posibilidad de recibir una sola calidad de video, sin importar las características de la red. Se han analizado varios modelos y para este trabajo se profundizará en el uso del Streaming dinámico adaptativo basado en HTTP, comúnmente conocido como DASH y el cual esta estandarizado por el DASH Industry Forum (DASH IF) y aceptado por varios fabricantes (ver Figura 3). MPEG-DASH (ISO/IEC 23009-1:2012) fue publicado como un ISO en Marzo del 2012 y una segunda edición en Mayo del 2014 como ISO/IEC 23009-1:2014 [4]. DASH tiene varias ventajas como transmisión confiable sobre TCP, reúso de servidores existentes HTTP e infraestructura de cache y compatibilidad con NATs y firewalls.

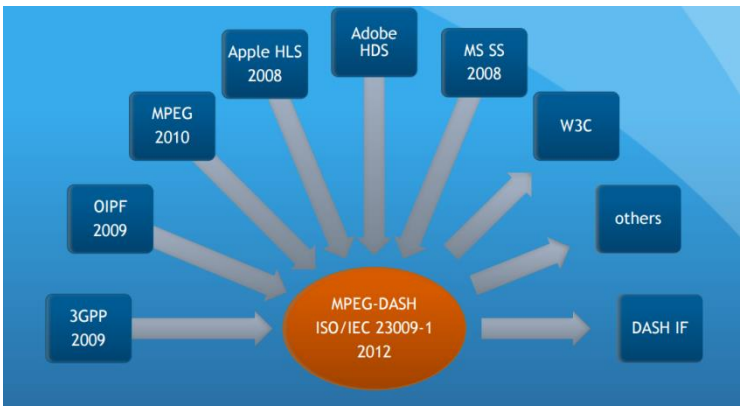


Figura 3. Estándar DASH

DASH especifica dos elementos principales: el formato en que el video es descargado y la descripción del video a ser descargado, que viene a estar representado por un archivo de manifiesto llamado MPD (*Media Description Protocol*) que provee una URL por cada versión del video y su tasa de bits. El estándar de DASH específica que se codifican los archivos de video a diferentes tasas de bits para un mismo contenido, y es almacenado en un servidor web. En el servidor cada secuencia de video es separada en varias representaciones de diferente calidad, diferenciándose en la tasa de bits o la resolución. Cada una de estas representaciones contiene uno o varios segmentos. Estos segmentos pueden ser decodificados independientemente. Cuando la cantidad disponible de ancho de banda es suficientemente alta, el cliente selecciona segmentos de una tasa de bits alta y seleccionará una tasa de bits baja cuando el ancho de banda disponible sea bajo. El cliente selecciona diferentes segmentos, una a la vez con peticiones de mensajes HTTP GET. La Figura 4 muestra una arquitectura de la distribución del contenido multimedia basado en el streaming HTTP [5]. El video es almacenado en varios servidores de origen los cuales contienen las diferentes calidades de video codificadas.

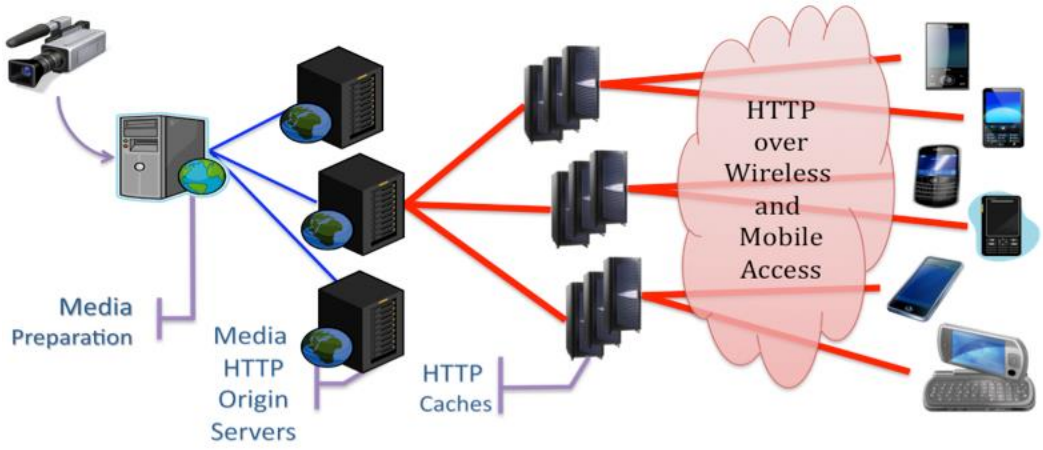


Figura 4. Distribución de contenido Multimedia

Cuando el cliente descarga segmentos, al mismo tiempo está realizando un algoritmo de adaptación que determina la política de selección de la mejor representación de acuerdo a ciertos parámetros. Estos parámetros pueden variar dependiendo el cliente, y pueden ser el ancho de banda medido o el nivel del buffer, y en algunos casos, ambos parámetros. Cuando se recibe un segmento, se calcula el algoritmo para conocer la calidad que se va a pedir en el siguiente segmento. Debido a que un cambio brusco de calidad puede afectar la experiencia del usuario, existen varias versiones de calidad de video intermedias. De esta manera se logra realizar las transiciones de calidad gradualmente. Si el algoritmo de adaptación lo determina, será posible ir subiendo la calidad del video.

En la Figura 5 se describe el sistema. El archivo MPD es un documento XML que describe el video que está disponible en el servidor para que el cliente pueda hacer una selección de la representación de video que mejor se ajuste a las características de su equipo y red. Información como el ID de la representación, la URL y la tasa de bits se almacenan en el archivo MPD. Cuando el cliente se conecta a un servidor DASH para solicitar un contenido, obtiene el archivo MPD después de establecer una conexión TCP entre el servidor y el mismo. La complejidad es dada a los clientes, ya que son ellos los que deben ejecutar el algoritmo de adaptación para escoger la mejor representación. Los clientes adaptan el video cambiando las representaciones conforme hay un cambio en las características de la red como el ancho de banda disponible o el retardo y envía una petición al servidor para obtener una nueva representación.

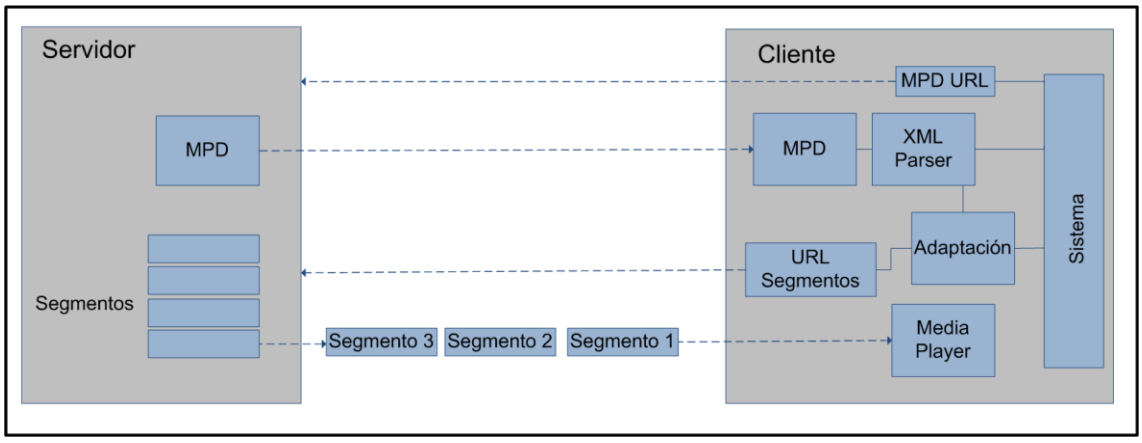


Figura 5. Cliente y servidor DASH

El cliente debe realizar el siguiente proceso:

1. Se obtiene el archivo MPD por HTTP.
2. Se parsea el contenido del archivo, es decir, se analiza sintácticamente; y de esta manera se obtiene la información necesaria para la reproducción.
3. Con la información de los tipos de reproducción, se seleccionan las representaciones adecuadas de acuerdo al algoritmo de adaptación y se solicitan los segmentos respectivos con peticiones HTTP.
4. Se reciben los segmentos respectivos y se comenzara a reproducir en el reproductor multimedia y al mismo tiempo se monitorea la red y sigue funcionando el algoritmo de adaptación para seguir solicitando los segmentos.

El contenido del archivo MPD se puede representar por un modelo de datos jerárquico tal como está representado en la Figura 6 [6]. Se representan uno o más periodos. Cada periodo tiene un tiempo de inicio y duración. De la misma manera cada periodo tiene uno o varios *Adaptation Sets* los cuales tienen información del contenido

multimedia, que puede ser el video, su audio asociado, subtítulos, etc. Cada Adaptation Set tiene varias representaciones, cada una con una codificación diferente, es decir de diferentes calidades. En el caso que sea el audio, tendrá información de tasa de bits que corresponderán si es estéreo de baja calidad o sonido envolvente de baja calidad. Cada representación deberá especificar la tasa de bits usada, la resolución, el número de canales, el códec usado, entre otras características. Cada representación se divide en múltiples segmentos, de forma secuencial de acuerdo a la duración del video, siempre empezando por un segmento de inicialización. Cada uno de estos segmentos tiene una URL, la cual será solicitada por el cliente al servidor utilizando una petición HTTP.

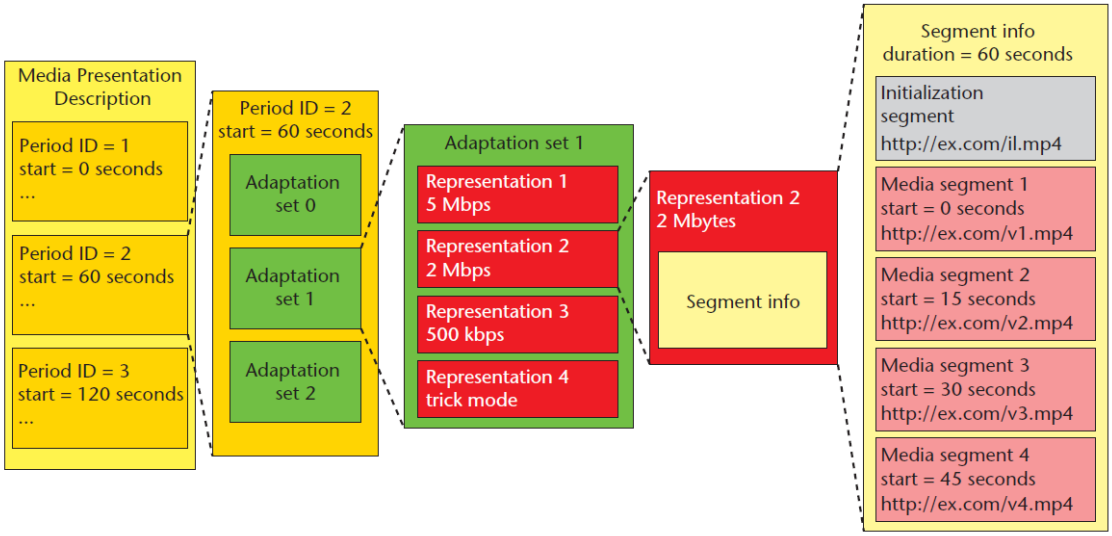


Figura 6. Estructura del archivo MPD

Tradicionalmente DASH asume el uso de MPEG AVC, también llamado H.264 para la entrega de contenido. En este caso, se guardan en el servidor representaciones de diferentes calidades y tasas de bits. Es decir si se tiene un video en 480p, puede haber otro en 1080p con la misma información del archivo de video anterior, solo que con mayor calidad. Esto significa que los proveedores de contenido deben aumentar el espacio disponible en disco solo para el mismo contenido. Una alternativa es el uso de SVC (*scalable video coding*) [7], el cual provee secuencias de video con varias calidades y en un solo segmento. Esto se logra usando varias capas en diferentes representaciones. El contenido de video es codificado en una capa base (base layer) y en una o más capas de mejora (enhancement layers) usando SVC. La capa base puede ser decodificada independientemente pero las capas de mejora son dependientes de la capa anterior. La capa base tiene la menor calidad y cada capa de mejora representa un aumento en la calidad del video. Como desventaja con respecto a AVC se tiene que esta codificación tiene un aumento en la sobrecarga por la codificación multi-capa y en la sobrecarga de

señalización. Otro formato que está empezando a usarse es H.265/HEVC que debe aumentar su uso en los próximos años [8].

2.2 Redes de Distribución de Contenidos

Con la creciente demanda de contenido multimedia, realizar el streaming a varios usuarios y en diferentes partes del mundo es un desafío. Aunque inicialmente se puede pensar que tener todo este contenido en un centro de datos es la mejor solución, la realidad es que un usuario dependiendo su localización geográfica puede estar pasando por varios ISP hasta tener el contenido deseado. Esto tiene consecuencias negativas con la calidad percibida por el usuario debido a que en ciertos caminos pueden produciré cuellos de botella resultando en retardos y por lo tanto en congelamientos constantes del contenido visualizado por el usuario.

La solución que han optado varios proveedores de contenidos como Netflix, Amazon o Hulu es utilizar redes de distribución de contenidos (CDNs). Una CDN tiene servidores distribuidos geográficamente manteniendo copias del contenido multimedia en cada uno de ellos, por lo que el usuario será siempre redirigido al servidor que más le convenga.

Los flujos de comunicaciones se dividen en dos flujos en las CDNs, uno entre el cliente y el servidor delegado (*surrogate server*), y otro entre el servidor delegado y el servidor original. Gracias a este esquema los usuarios pueden observar una mayor calidad de servicio y los proveedores de contenidos pueden ofrecer servicios más confiables y procesar más peticiones de sus usuarios. Del mismo modo existe u tercer involucrado que se ve beneficiado del uso de las CDNs, que son los ISPs, los cuales se benefician al tener servidores CDNs en sus redes al ver el tráfico en su core reducido.

Se utiliza DNS para interceptar y re direccionar las peticiones. Los proveedores de contenido como Netflix contratan uno o varios CDNs distribuidos geográficamente para enviar el contenido multimedia a sus suscriptores. En el caso de un proveedor de contenido XYZ por ejemplo se tendría una URL asignada para cada video como http://video.XYZ.com/6YDUB32 por poner un ejemplo. El usuario visita la página web de XYZ y selecciona el video http://video.XYZ.com/6YDUB32, luego envía una consulta DNS a video.XYZ.com. Después, el servidor DNS local del usuario (LDNS) hace un relay de la consulta DNS a un servidor DNS autoritativo para XYZ, el cual observa la cadena "video" en el hostname. El servidor DNS autoritativo en lugar de devolver una dirección IP devuelve al LDNS un hostname en el dominio del proveedor CDN. Desde este momento las consultas DNS van directamente a la infraestructura del CDN. En otra consulta DNS que realice el LDNS, el servidor DNS del CDN retornará la dirección IP de un servidor de contenido de la infraestructura de la CDN.

El LDNS reenviará la dirección IP al host y el usuario establecerá una sesión TCP con el servidor correspondiente.

En la Figura 7 se puede ver un esquema de la plataforma de video Netflix y como hace uso de las CDN [9]. Netflix hace uso de la infraestructura de un tercero, en este caso la nube de Amazon para tener sus servidores y el contenido multimedia, además de algunos proveedores de CDN. Cuando selecciona un video el usuario obtiene un archivo de manifiesto de los servidores de la nube de Amazon con información como la lista de CDNs disponibles y los varios tipos de calidad de video como fueron descritos anteriormente en DASH. Para escoger la CDN se escogerá el primero en base a un ranking definido por Netflix. Luego de seleccionar el CDN, el cliente y el servidor interactuarán tal como fue descrito en el capítulo 2.1.2.

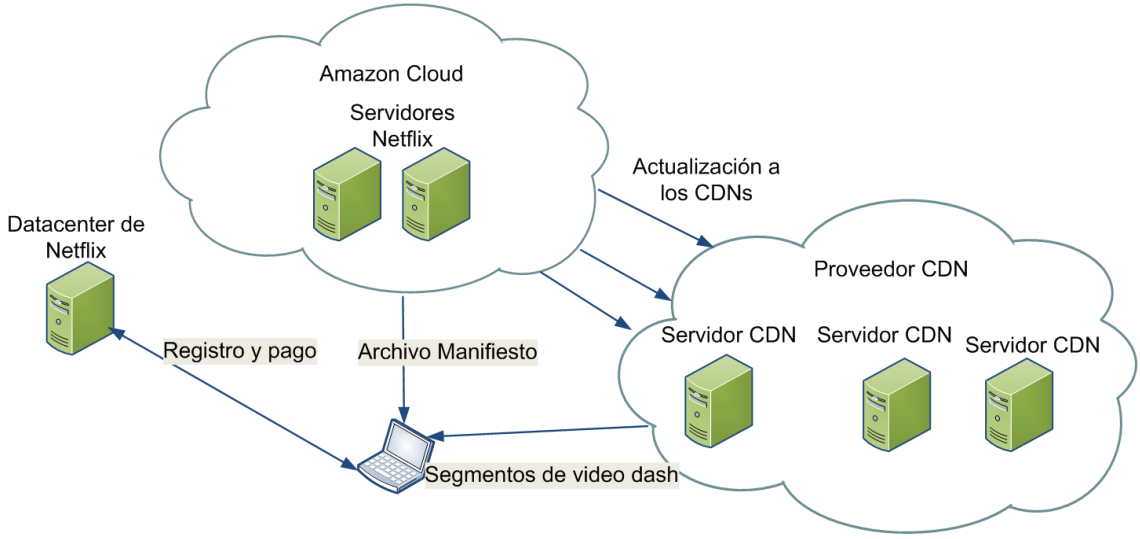


Figura 7. Uso de CDNs en Netflix

Para terminar está descripción de las CDNs se analizan las dos alternativas usadas con respecto a la colocación de los servidores [10]. La primera es una filosofía "enter deep" que es impulsada por Akamai, uno de los principales proveedores de CDNs. Su enfoque es colocar clusters de servidores en las redes de acceso de los ISP. De esta manera se tienen los servidores más cerca de los usuarios y se mejore en el retardo que percibe el usuario.

El segundo enfoque coloca los servidores lo más cerca de los puntos de presencia PoP de los principales ISP ("bring ISPs to the home"). En este caso se seleccionan pocos lugares que están conectados por redes de alta velocidad. En este caso no se usa la red de acceso de los ISP teniendo un menor mantenimiento y menos sobrecarga pero con el aumento del retardo. Este enfoque es usado por Limelight y otros CDN pequeños.

Es posible combinar varios esquemas como el uso de P2P y la Cloud Computing para la implementación de las CDNs. Amazon CloudFront es un ejemplo de este caso.

Se tiene la ventaja que se puede usar el esquema "pago por uso" y ahorrar en inversión de infraestructura.

2.3 Streaming de video usando Multicast

El streaming adaptativo sobre HTTP se ha convertido en la principal implementación realizada por varios fabricantes como Apple, Microsoft o Adobe. A pesar de lo anterior expuesto, este tipo de streaming no soporta transmisión de datos usando multicast, como si lo hacen protocolos como RTP. Aunque existen estudios desarrollados para el uso de multicast basado en RTP [11], para efectos de este documento se realizará un análisis en multicast basado en archivos, que vendría ser el caso de DASH.

2.3.1 Casos de uso Broadcast/Multicast

Transmitir usando unicast es eficiente cuando los servicios requieren un enlace bidireccional como por ejemplo comunicaciones de video y voz en tiempo real o para streaming de videos en demanda. El broadcast es más eficiente para cuando hay que dar servicios a muchos usuarios localizados en la misma área y que consumen el mismo contenido multimedia al mismo tiempo. De la misma manera es eficiente para llevar el contenido a muchos dispositivos al mismo tiempo sin intervención del usuario, como puede ser caching de contendido, publicidad o actualizaciones de software. Multicast nos permite conocer que usuarios han activado determinados servicios. En el caso de LTE, usando eMBMS (evolved Multimedia Broadcast/Multicast Service), multicast es solamente usado para el transporte de datos dentro de la red [12].

Con el desarrollo de eMBMS en LTE se tienen mejoras en la capa de servicios gracias a las características de la red LTE. Mejoras en la codificación de video para altas resoluciones y técnicas de *Forward Error Correction (FEC)* que es utilizada en canales unidireccionales para la mejora en la transmisión. Es posible entregar contenido multimedia de mayor calidad con calidad de servicio y habilitar entrega de contenido a los usuarios usando *user equipment (UE)* caching.

La arquitectura de la capa de servicios en eMBMS involucra a entidades como dispositivos de usuarios finales, servidores de aplicaciones, servidores de video streaming, servidores de contenido de software y CDNs. Para el eMBMS, el dispositivo de usuario final no se conecta directamente a los servidores de video o de contenido software en el caso de las conexiones unicast. Puede establecer una conexión sin pasar por el core de la red móvil (EPC). Usando conexiones broadcast y multicast se utiliza una entidad llamada MBMS-GW. Se realiza el envío del contenido desde el servidor (CDN por ejemplo) pasando por el core de la red por medio de un Broadcast/Multicast Service Center (BM-SC) y llega a la red de acceso por medio del MBMS-GW que enviará contenido al dispositivo del usuario (ver Figura 8). En la

siguiente sección se analizará cómo se entrega el contenido al usuario usando el protocolo FLUTE en conjunto a DASH.

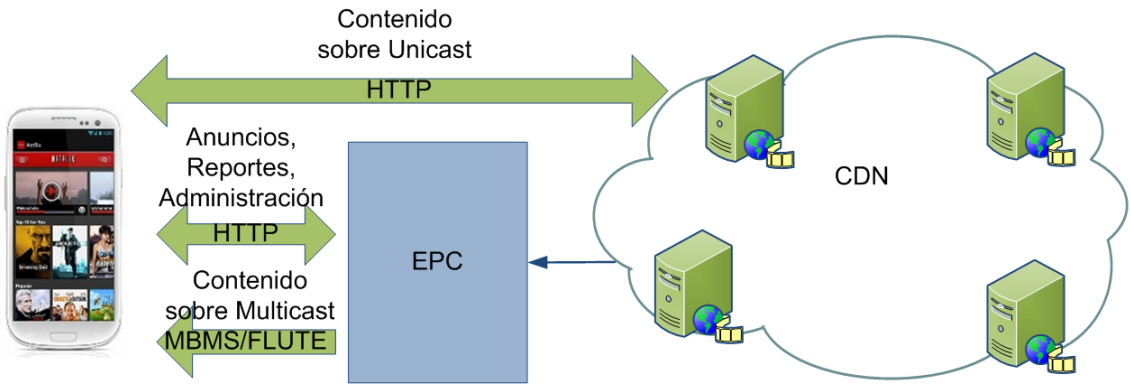


Figura 8. Transmisión de contenido Multimedia usando red LTE en unicast y multicast

2.3.2 FLUTE/DASH

Las redes Multicast no garantizan la entrega exitosa de los paquetes a diferencia de protocolos usando unicast como HTTP. Para este caso se usa FLUTE (*File Delivery over Unidirectional Transport*), que está definido en el RFC 6726 [13] para la entrega de archivos sobre una red que esté usando multicast. Para asegurar confiabilidad en la entrega, FLUTE realiza los siguientes mecanismos:

- AL-FEC (*Application Layered Forward Error Correction*) para añadir redundancia y corregir errores.
- Retransmisiones para recibir paquetes perdidos previamente.
- Sesiones de reparación de archivos para solicitar los paquetes que no han sido recibidos.

FLUTE usa un archivo de metadatos en XML llamado FDT (*File Delivery Table*) para describir las propiedades de los objetos transmitidos durante la sesión. Cada vez que exista actualizaciones se las envían al cliente como instancias FDT para que esté al tanto de los objetos que se han creado dinámicamente. FLUTE se puede realizar para transmisiones tanto en vivo como en demanda.

A pesar del uso de técnicas de FEC, es posible que no todos los segmentos de video sean recuperados con el uso de FLUTE sobre eMBMS. Si la tasa de errores es alta durante la transmisión de los segmentos de video es posible que no se tengan los suficientes paquetes para para recuperar los segmentos. La solución en este caso es usar un hibrido entre FLUTE y DASH. En este caso, los segmentos de video que se han perdido pueden ser recuperados usando HTTP unicast.

El 3GPP define como utilizar DASH en conjunto con los servicios de streaming en una red LTE usando el eMBMS, reemplazando la transmisión unicast usada en DASH

por una multicast que es la que necesitaría. Se utiliza el protocolo UDP al usar esta arquitectura. En la Figura 9 [14] se puede observar los componentes de este enfoque en conjunto de FLUTE y DASH utilizado para la entrega de contenido. El video es enviado usando FLUTE usando la red eMBMS, es decir usando multicast. El Cliente FLUTE decodifica los segmentos recibidos, corrige los errores de ser necesario y los pasa al Cache HTTP. Utilizando DASH, se puede obtener los segmentos de video del Cache HTTP si es que fueron recibidos correctamente en la sesión multicast. En el caso que no se encuentra en el cache, se lo solicita al servidor utilizando una petición HTTP y luego se la almacena en el HTTP Cache. En esta arquitectura, el uso de FLUTE y de multicast es transparente para DASH.

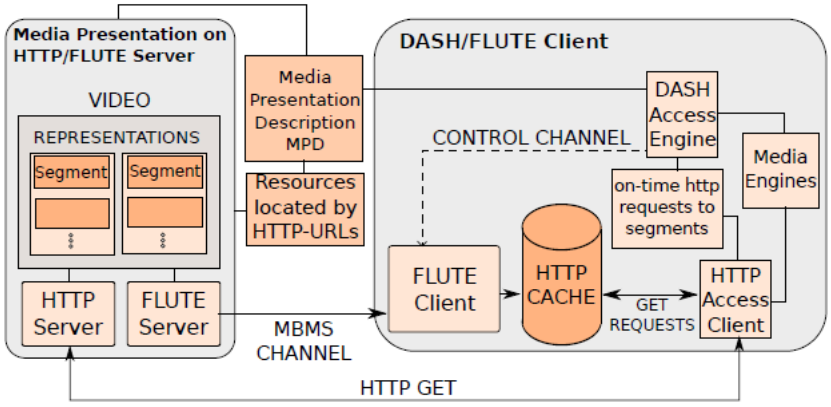


Figura 9. Arquitectura FLUTE/DASH

2.3.3 Handover entre Streaming Broadcast/Multicast y Unicast

A diferencia del caso de transmisión de contenido utilizando DASH donde observamos que es posible que el cliente pueda recibir distintas calidades de video dependiendo el algoritmo de adaptación, usando streaming multicast un usuario recibe una sola representación que depende del ancho de banda del canal multicast y de la redundancia que agrega AL-FEC. Si se desea tener una transmisión de video continua, es necesario considerar el handover entre las sesiones multicast y unicast.

Es necesario acceder al archivo de manifiesto MPD en cada caso, es decir tener diferentes MPD para cada sesión. El archivo de Multicast tendría una sola representación y el archivo MPD para unicast tendría las representaciones que considere necesarias. El reproductor multimedia debe manejar la información de los dos archivos, cargarlos al inicio, actualizarlos.

Se utiliza en este documento una propuesta [14] de usar un solo archivo MPD que contenga todas las representaciones para el caso unicast y marcar la representación para el streaming multicast. Al usar streaming unicast, el reproductor elegirá la representación dependiendo el algoritmo de adaptación que seleccionará la calidad

dependiendo el ancho de banda disponible. Al realizar el cambio a multicast, se escogerá solo la representación marcada.

En el archivo XML se especifica en la representación elegida para multicast, un esquema llamado *accessTech* y se lo marca como multicast para especificar que esa representación es la que debe ser entregada sobre por el eMBMS.

El prototipo usado en [14] se describe en la Figura 10. Se tiene un servidor de contenidos que contiene el contenido multimedia, un servidor FLUTE y un servidor web Apache para responder a las peticiones unicast. Se modela la red LTE controlando el ancho de banda, la latencia y las perdidas. Por ultimo en el cliente se incluye un cliente FLUTE, un reproductor DASH, un servidor proxy y un cache HTTP.

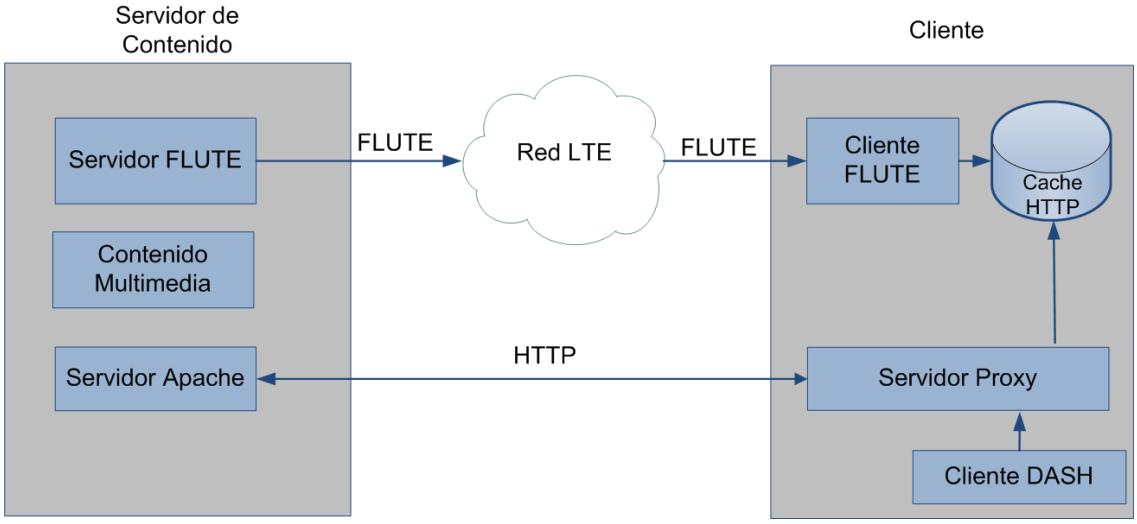


Figura 10. Esquema para experimentación usando FLUTE y DASH

2.4 Parámetros a considerar para el streaming Multimedia

La calidad de experiencia (QoE) se refiere al grado de placer o disgusto que se tiene con un servicio en particular en base al cumplimiento de sus expectativas. Para el caso de este documento, el servicio en particular es el streaming de video. Se diferencia de la calidad de servicio (QoS) en que está ultima está relatada a parámetros como perdida de paquetes o el throughput en la red o aplicaciones. En cambio la QoE es una medida subjetiva que dependerá de la perspectiva del usuario.

Al ser la QoE una medida subjetiva, se utiliza el *Mean Opinion Score* (MOS) de los usuarios. Esta medida da el grado de satisfacción que tiene el usuario con el servicio. Es posible obtener una relación entre las métricas de QoS y de QoE ya que por lo general suelen estar relacionados.

2.4.1 Calidad de Servicio en la red

La QoS en la red es el rendimiento que se tiene en el camino seguido entre el cliente y el servidor, donde se incluyen parámetros como la tasa de paquetes perdidos, el *round-trip time* RTT y el ancho de banda de la red. Estos parámetros están relacionados entre sí y el que afecte a uno puede afectar a todos. Estos parámetros durante la descarga de contenido pueden tener fluctuaciones, lo cual es común en una red tipo best-effort como internet. Esto puede agravarse si se está transmitiendo en un medio inalámbrico, donde las condiciones del canal están variando en el tiempo gracias al desvanecimiento (fading), interferencia o ruido. A nivel de la aplicación se verá afectada la transmisión debido a la disminución del throughput y el retardo introducido que ocasionará que el buffer del cliente se llene lentamente o que incluso se agote.

El algoritmo de adaptación de DASH decide cuales de los segmentos son descargados (además de la disponibilidad que viene dado en el archivo MPD) por las condiciones de la red, en este caso por el ancho de banda estimado (throughput) en el enlace entre el cliente y el servidor.

2.4.2 Estrategias para mejorar la calidad de experiencia

En el streaming de video basado en HTTP se tiene que los principales factores de QoE son el retardo inicial y la parada del video. Al agregar el algoritmo de adaptación como en el caso de DASH, se introduce un nuevo parámetro que viene dado por el cambio de la calidad de video. Los factores de influencia pueden ser clasificados en la influencia que percibe el usuario y la influencia técnica [15].

Los factores de influencia percibida son las siguientes:

- Tiempos de espera
    - Retardo inicial
    - Frecuencia de parada
    - Duración de parada
- Adaptación del video
    - Frecuencia de cambio
    - Amplitud
    - Tiempo en cada capa
- Calidad de video
    - Resolución espacial
    - Escalabilidad temporal
    - Calidad de la imagen
- Factores de contexto
    - Dispositivo

    - Contenido
    - Uso

Mientras tanto, los factores técnicos encontrados pueden clasificarse en:

- Del lado del servidor
    - Codificación del video
    - Tamaño de los segmentos
- Del lado del cliente
    - Reproductor de video
    - Adaptación
    - Buffer
- Algoritmo de adaptación
    - Monitoreo de QoS
    - Estimación de tasa de bits
    - Nivel del buffer
- Interacción entre entidades
    - Múltiples clientes DASH
    - DASH y TCP
    - DASH y otras aplicaciones

El retardo de inicio es el tiempo que se demora en reproducir el video luego de iniciarlo. Es dependiente del buffer y la calidad de la red por la que están pasando los segmentos de video. La cantidad del buffer que debe llenarse inicialmente necesita ser equilibrado entre la longitud de su retardo y el riesgo de paradas previos. Que sea mayor la longitud del nivel del buffer para que empiece la reproducción repercutirá en menores o nulos números de paradas futuro. Aunque el tiempo de espera hasta que se complete cierto nivel del buffer es esperado por el cliente, paradas del video involucra interrupción del contenido visualizado. Efectos del retardo inicial son perjudiciales solo cuando el usuario está haciendo lo que podría llamarse una especie de "zapping". Pero si realmente tiene la intención de ver el video, entonces puede tolerar ese retardo inicial.

Como se ha detallado anteriormente, la parada involucra la detención de la reproducción debido a que no hay contenido en el buffer. El problema se da si el throughput de la aplicación de video es más bajo que la tasa de bits del video. La reproducción se detendrá hasta que el buffer tenga una cierta cantidad de datos de video. En [16] se hace una relación entre el MOS y el número de paradas y encuentran que los usuarios pueden tolerar a lo mucho un evento por reproducción si es de pocos segundos. Mientras el streaming sin adaptación es limitado por las

fluctuaciones de la red, el uso de un algoritmo de adaptación puede mitigar los efectos de las paradas.

La adaptación es otro factor a tomar en cuenta para medir la QoE. Se realiza el cambio de calidad de video al enviar segmentos son una tasa de bits que corresponda con las características de la red. Como se mencionó anteriormente, el algoritmo de adaptación puede disminuir las paradas si es usado eficientemente. Es importante tener en cuenta la frecuencia de cambio en la calidad de video. Aunque es verdad que realizar cambios es mejor que tener siempre una baja calidad; una frecuencia alta puede resultar molestosa para los usuarios. Una disminución escalonada de la calidad de video es calificada mejor que una sola disminución de calidad. Lo que sí está claro, es que el algoritmo de adaptación debe primero intentar mantener la calidad más alta posible.

La calidad percibida puede ser afectada por las diferentes dimensiones en la adaptación como pueden ser la calidad de la imagen, resolución espacial y temporal. La adaptación temporal se refiere a los cambios entre las tasas de las tramas. Por ejemplo cambios de 15 fps a 30 fps no son detectadas por la mitad de los usuarios. Solo una reducción de la calidad de la imagen, es decir un aumento de los parámetros de cuantización y una disminución a 10 fms tendría efectos negativos para los usuarios. En lo que se refiere a la adaptación espacial (pixeles por línea), los cambios no deben exceder de la mitad del tamaño original [17].

Del lado del servidor es importante la preparación del contenido que será luego solicitado por el cliente. Luego de seleccionar la codificación utilizada (AVC, SVC, HEVC), el contenido a optimizar son los segmentos de video. Los segmentos deben ser lo suficientemente cortos para permitir la adaptación debido a los cambios en las condiciones de la red. Pero de la misma manera deben ser lo suficientemente largos para permitir una alta eficiencia en la codificación y para que la sobrecarga se mantenga baja. Por lo tanto se tiene este problema a considerar en el diseño y preparación del servidor de contenidos.

Del lado del cliente es donde se deben tomar más acciones al ser donde está casi toda la inteligencia. El cliente debe decidir cuales segmentos debe descargar, por lo tanto el algoritmo de adaptación debe escoger la correcta representación para maximizar la QoE. En [18] se realiza un análisis de las estrategias usadas realizando mediciones del tiempo de llegada de los segmentos, retardo de inicio del video, retardo cliente y servidor, el tiempo para empezar a descargar el siguiente segmento y la manera en que se tratan los segmentos perdidos. Cada estrategia usada refleja un diferente comportamiento. Por lo tanto la estrategia dependerá de las condiciones de la red y las prioridades de QoE deseadas por el usuario. Un análisis desarrollado en [19]

selecciona las representaciones en el caso de MPEG-AVC de acuerdo al ancho de banda actual, nivel actual del buffer y la tasa de bits promedio de cada segmento.

La interacción entre otras entidades en la red también debe tomarse en cuenta para la evaluación de la QoE en el streaming de video. Se debe considerar la interacción entre: diferentes clientes, otras aplicaciones en instancias del mismo cliente e instancias del cliente en otras aplicaciones [15].

Cuando se tienen varios clientes utilizando streaming de video basado en HTTP, un problema dentro de la red es la estabilidad ocasionada por la frecuencia de los eventos. Mientras más clientes existan en la red compartiendo el mismo cuello de botella por el ancho de banda, más frecuente será el cambio de calidad de video en los mismos. A cual cliente será dada la mejor calidad inicialmente dependerá del orden de llegada al sistema. La competición de varios clientes por el uso de la red es otro desafío a considerar. Es posible que un solo cliente utilice la mayoría del ancho de banda disponible. Mientras mayor es el número de clientes, la inequidad aumenta. La equidad en TCP (TCP fairness) dice que si se tiene K conexiones TCP compartiendo el mismo cuello de botella de ancho de banda R entonces cada cliente debe recibir una tasa promedio de R/K. Desafortunadamente esto no se da cuando el RTT es diferente en todas las sesiones TCP. A pesar de ese comportamiento en TCP, el problema entre las múltiples instancias de streaming se da por la competición que tienen entre ellas, especialmente al usar el algoritmo de adaptación. Se observa un ejemplo [20] en la Figura 11. Por último, la utilización de la red también produce un impacto. Puede existir una baja utilización de los recursos de la red debido a la oscilación de las calidades de video entre las múltiples instancias. En resumen, la estabilidad, equidad y utilización del ancho de banda resultan en factores determinantes en la QoE desde la perspectiva del usuario.

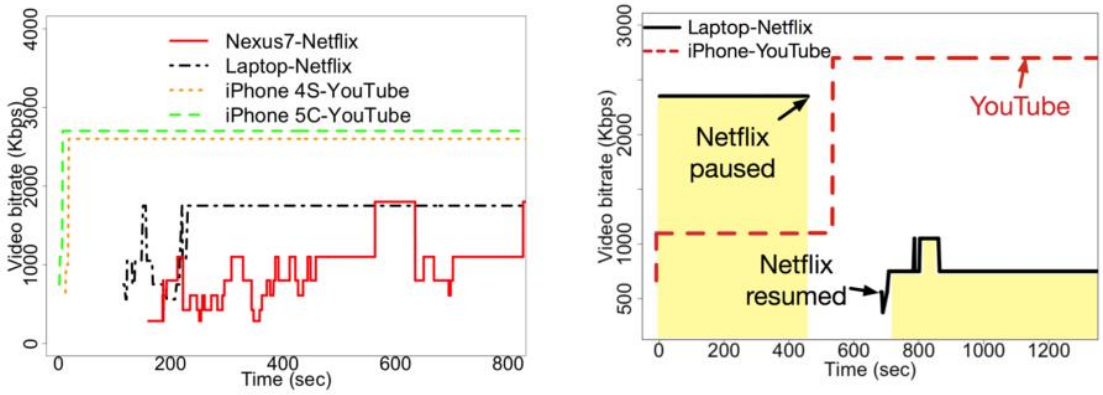


Figura 11. Inequidad entre varios servicios de streaming compitiendo por recursos de la red

El impacto que tienen otras aplicaciones en los clientes utilizando streaming de video adaptativo basado en HTTP involucran tráfico de navegación web, juegos en línea o video conferencia, pero falta investigación para saber cómo el patrón de estas aplicaciones podría afectar a los clientes de streaming adaptativo [15]. Por el contrario, otro factor a considerar es el impacto que tienen los clientes de streaming con otras aplicaciones. Los clientes están constantemente pidiendo segmentos de video sobre HTTP, por lo general archivos pequeños, lo que causa que TCP sobreestime el ancho de banda resultando en un efecto bufferbloat [21], que se refiere a la existencia de una gran cantidad de buffers en el sistema. Tener varios buffers en el sistema disminuye la QoE Por lo tanto es difícil pasar por un cuello de botella para transmisiones que no sean para streaming de video.

3 Redes definidas por Software

Las redes definidas por software (SDN) son un nuevo paradigma de comunicaciones que separa el plano de datos del plano de control de la red. Desplegar SDNs tiene como objetivo reducir costos usando virtualización, automatización y simplificación.

Se describe en este capítulo los principios de desarrollo de las SDN, sus principales protocolos y un detalle de las aplicaciones que nos serán de ayuda para la implementación de este trabajo de fin de master.

3.1 Características de las Redes Definidas por Software

3.1.1 Definición y Arquitectura de las SDN

Las redes tradicionales usan algoritmos que están implementados en hardware dedicado. Por lo general están implementados en ASICs (Application Specific Integrated Circuits) para realizar funciones específicas como puede ser el reenvió de paquetes. Problemas como la falta de escalabilidad, seguridad, confiabilidad son aspectos a considerar para el futuro de las redes. Además, las redes deben ser capaces de adaptarse a los cambios sin tener que realizar siempre cambios totales en hardware y software. Las SDN es una de los últimos paradigmas que han ganado adeptos en el despliegue de nuevas redes de datos.

Varias startup pequeñas se han dedicado al desarrollo de las SDN como Big Switch, Pica8, Cyan y NoviFlow, ofreciendo un mercado más competitivo y abierto, siendo este uno los principales objetivos del desarrollo de las SDN [22]. Otro ejemplo de desarrollo es la implementación de equipos *Whitebox Switches* donde el hardware y el software son vendidos de forma separada [23].

SDN es un paradigma que separa el plano de control del plano de datos en una red de datos. Este esquema cambia la inteligencia de la red a un nuevo elemento llamado controlador. Usando los protocolos de red tradicionales donde el control está en cada uno de los equipos de red y se quiera aplicar un esquema de enrutamiento, se tendría que usar alguno existente quedando poco margen para la experimentación. Usando un esquema de SDN, operadores de red pueden implementar nuevas estrategias de enrutamiento tomando los requerimientos de las aplicaciones de red. El operador necesita implementar estas estrategias en el controlador. El controlador envía comandos que contienen lo que viene a ser conocido como reglas de flujo para los

elementos de red, como puede ser un switch. Se tiene la ventaja que el controlador puede tener una vista de toda la topología de red al comunicarse con los switches. Esta información puede ser el tráfico que pasa en cada enlace, paquetes perdidos o paquetes descartados. Se muestra en la Figura 12 un ejemplo con una comparación de las redes tradiciones con este nuevo paradigma que son las SDN.

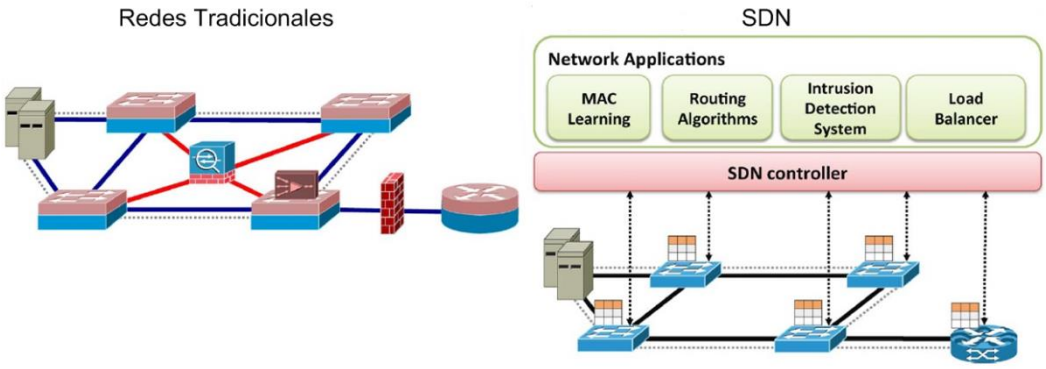


Figura 12. Comparación redes tradicionales y SDN

Una de las mayores ventajas de implementar las SDN es la simplificación. Los dispositivos de networking se han convertido en elementos muy complejos al tener que dotarles con cada vez más inteligencia. Tratar de agregar nuevas características a estos dispositivos ya establecidos se convierte en una implementación complicada por los cambios que serían necesarios en esos equipos. Como se explicó anteriormente, la posibilidad de tener el plano de control separado de los equipos físicos permite realizar los cambios de una manera más ágil y dinámica. De la misma manera es posible simplificar el mantenimiento y administración de los equipos de red al cambiar protocolos como SNMP y usar sistemas basados en políticas [22].

Otro de los beneficios de usar SDNs son los beneficios en la investigación e innovación. En las últimas décadas los fabricantes de equipos de networking han controlado todos los aspectos de los equipos como hardware, firmware y software. Este software por lo general es cerrado. Innovaciones realizadas por universidades e investigadores en general se pueden realizar al tener una mayor libertad en el despliegue de nuevos sistemas y cambios más agiles en las redes. Con las redes tradicionales, al tener muchos sistemas cerrados, la innovación es más compleja.

En la Figura 13 [24] se muestra la arquitectura de SDN. Es un modelo de tres capas que consiste en:

- **Capa de aplicación:** Incluye aplicaciones de comunicaciones como priorización de VoIP, aplicaciones de seguridad como firewalls, balanceadores de carga, etc.
- **Capa de Control:** Lo que solía ser el plano de control en los switches/routers ahora está centralizado. Esto permite tener una red programable.

- **La capa de infraestructura:** Incluye switches y routers físicos. El tráfico es pasado basado en tablas de flujo. Esta capa es inalterada en SDN con el único detalle que las tablas de flujo son manejadas por el controlador que se encarga de insertarlas o removerlas de los equipos de red

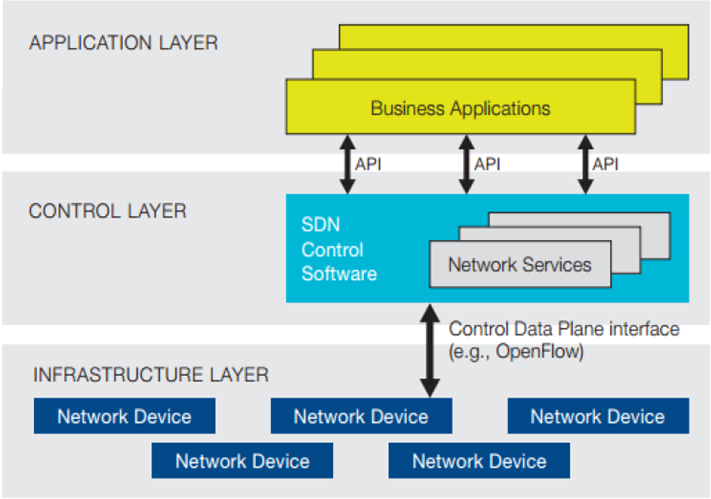


Figura 13. Arquitectura SDN

Las instrucciones realizadas entre los equipos de red de la capa de infraestructura y la capa de control son definidas por la **Interfaz Southbound.** El protocolo usado con mayor aceptación es el protocolo OpenFlow. En cambio, la comunicación realizada entre la capa de control y as aplicaciones es realizada por la **Interfaz NorthBound,** que es utilizada para ofrecer una API a los desarrolladores de aplicaciones.

Las aplicaciones SDN son implementadas en la capa superior del controlador (por la interfaz northbound) y no debe confundirse con las aplicaciones de la capa 7 del modelo OSI. Estas aplicaciones realmente son parte de las capas 2 (enlace de datos) y capa 3 (Red) del modelo OSI. Las aplicaciones SDN se comunican con el controlador para insertar flujos proactivos en los dispositivos y recibir paquetes que han sido reenviados al controlador. El controlador también puede enviar un flujo proactivo para modificar la tabla de flujos en base al tráfico que está circulando por la red.

El otro tipo de flujo es el flujo reactivo. Estos flujos son insertados en respuesta a un paquete que ha sido reenviado desde el equipo de red al controlador. Cuando el equipo de red recibe un paquete, revisa su tabla de flujos para saber qué hacer con el mismo, si está en su tabla decide reenviarlo a un puerto en específico o eliminarlo. En caso que no se encuentre, lo envía al controlador.

Las tablas de flujo son la principal estructura de datos en un dispositivo SDN. Las tablas de flujo permiten a os elementos de red evaluar los paquetes entrantes y tomar

decisiones en base al contenido de los mismos. En SDN a diferencia de una red tradicional, estas acciones son realizadas de una manera más programable.

Las tablas de flujo consisten de entradas de flujo, ordenadas en orden de prioridad y tienen como componentes principales los campos de cabecera y la acción a realizar. Al entrar un paquete se comparan los campos de cabecera, empezando por las entradas de mayor prioridad, hasta que se tenga una coincidencia.

3.1.2 OpenFlow

OpenFlow es uno, o tal vez el más importante de los protocolos usados en SDN. En lo que se refiere a la interfaz Southbound, es prácticamente el único implementado junto a otras implementaciones propietarias. OpenFlow define el comportamiento entre el plano de datos y el plano de control. Se pueden definir dos elementos principales: El switch OpenFlow y el controlador OpenFlow. Este protocolo define los mensajes usados para comunicar estos dos dispositivos.

OpenFlow tiene varias versiones disponibles, empezando desde la 1.0 a la 1.5. Cada nueva versión ha ido añadiendo extensiones y mejoras. La ONF (Open Networking Foundation) [25] es la organización dedicado a la difusión y estandarización con respecto a las SDN y OpenFlow.

MAC src	MAC dst	IP Src	IP Dst	TCP dport	...	Action	Count
*	10:20:.	*	*	*	*	port 1	250
*	*	*	5.6.7.8	*	*	port 2	300
*	*	*	*	25	*	drop	892
*	*	*	192.*	*	*	local	120
*	*	*	*	*	*	controller	11

Figura 14. Ejemplo de una tabla de flujo OpenFlow

En la Figura 14 se puede observar un ejemplo de una tabla de flujo. Se observan diferenciados los campos de cabecera, la acción a realizar y los contadores. Cada entrada contiene un valor especifico, o un wildcard como un * (ANY). Los contadores son mantenidos por tabla, por flujo, por puerto o por colas. Los campos de cabecera dependerán de la versión de OpenFlow usada. Se tiene de acuerdo a la versión 1.3 [26] entre las campos de cabecera protocolos IP, Ethernet, TCP, UDP, MPLS. Siendo los campos en la Tabla 1 los obligatorios en el switch OpenFlow.

Tabla 1. Campos requeridos para tabla de flujo OpenFlow

Campo	*Descripción*
OXM_OF_IN_PORT	Puerto del switch
OXM_OF_ETH_DST	Dirección Ethernet destino
OXM_OF_ETH_SRC	Dirección Ethernet origen
OXM_OF_ETH_TYPE	Tipo Ethernet del payload OpenFlow
OXM_OF_IP_PROTO	IPv4 o IPv6
OXM_OF_IPV4_SRC	Dirección IPv4 origen
OXM_OF_IPV4_DST	Dirección IPv4 destino
OXM_OF_IPV6_SRC	Dirección IPv6 origen
OXM_OF_IPV6_DST	Dirección IPv6 destino
OXM_OF_TCP_SRC	Puerto TCP origen
OXM_OF_TCP_DST	Puerto TCP destino
OXM_OF_UDP_SRC	Puerto UDP origen
OXM_OF_UDP_DST	Puerto UDP destino

El switch o dispositivo de red al recibir un paquete realiza las siguientes acciones [27]:

- **Forward:** Reenvía los paquetes a determinados puertos.
  - **ALL**: Envía paquetes a todas las interfaces
  - **CONTROLLER**: Encapsula el paquete y lo envía al controlador.
  - **Local**: Envía el paquete a la pila local de networking del switch
  - **TABLE**: Realiza una acción en la tabla de flujo para paquetes que vienen desde el controlador.
  - **IN_PORT**: Enviar el paquete por el puerto de entrada.
- **Drop**: Todos los paquetes que cumplan la condición de la entrada de flujo deben ser descartados.

Si se trata de un switch OpenFlow hibrido, es decir que puede comportarse como un switch Ethernet o un router IP, entonces puede soportar lo siguiente:

- **NORMAL**: Procesa el paquete usando métodos tradicionales de capa 2 o capa 3 soportados por el switch.
- **FLOOD**: Realiza una inundación usando spanning tree, sin incluir el puerto de entrada.

En caso de pérdida de conectividad entre el controlador y el switch se utiliza un mecanismo agregado en la versión 1.1. El switch entra en uno de estos modos en caso de pérdida de conectividad con el controlador: Los modos de operación son:

- **Fail secure mode:** El switch continua operando como un switch OpenFlow, pero todos los paquetes que le lleguen serán descartados.

- **Fail standalone mode**: Switch deja de funcionar en modo OpenFlow y comienza a operar como un router o switch tradicional. Al recuperarse la conexión, vuelve a su funcionamiento normal.

La comunicación entre el controlador y switch tiene tres fases, las cuales componen la inicialización, la operación y el monitoreo. Los mensajes son transmitidos usando un canal seguro de comunicaciones TLS (Transport Layer Security). El switch y el controlador se autentican enviándose certificados firmados con su clave privada. Los mensajes OpenFlow empiezan con una cabecera que contiene el número de versión, el tipo de mensaje, la longitud del mensaje, y el ID de la transacción del mensaje. Los mensajes OpenFlow entran en las siguientes categorías:

- **Mensajes simétricos**: Son enviados entre el controlador al switch sin que sean solicitados. Ejemplo de estos mensajes son los HELLO que son intercambiados luego de establecer una conexión segura y son usados para determinar la más alta versión OpenFlow entre el switch y el controlador. Otro mensaje son los ECHO, usados para confirmar se mantenga la conectividad.
- **Mensajes asincrónicos** ***Async:*** Son enviados desde el switch al controlador sin haber sido solicitados por el controlador. Se usan para actualizar al controlador de los eventos en la red y los cambios de estado en los switches. El mensaje PACKET_IN es la forma en que el switch pasa los paquetes al controlador para poder saber cómo manejarlos. El switch puede informar al controlador que una entrada de flujo debe ser removida por un mensaje FLOW_REMOVED. PORT_STATUS para informar cambios de los puertos y mensajes ERROR para notificar problemas.
- **Mensajes controlador-switch.** Son mensajes iniciados por el controlador para administrar o inspeccionar el estado de los switches. Pueden ser divididos en: Configuración switch, comando desde controlador, estadísticas, configuración de colas y barreras. Entre estos mensajes está el mensaje PACKET_OUT. El controlador usa este mensaje para enviar paquetes de datos. FLOW_MOD lo usa el controlador para modificar entradas de flujo existentes. STATS lo usa para obtener estadísticas de los equipos de red.

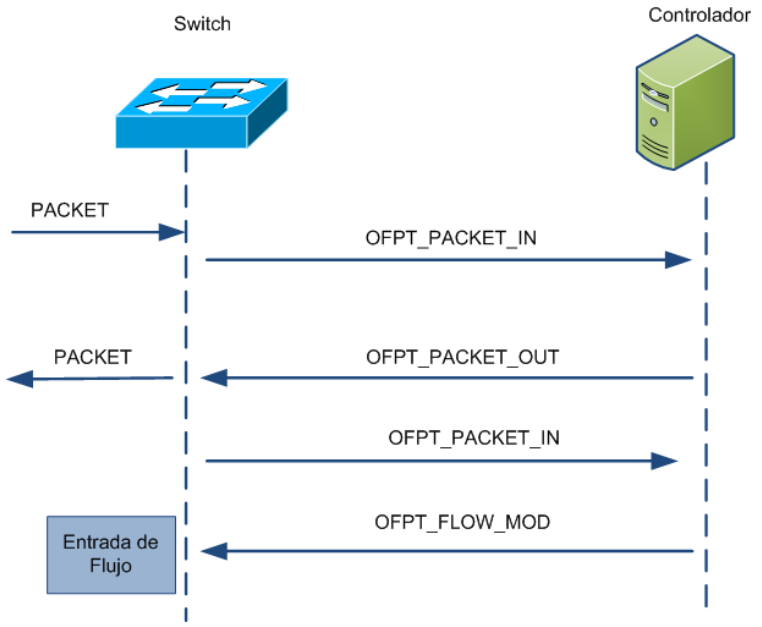


Figura 15. Flujo de mensajes OpenFlow

Se muestra en la Tabla 2 los distintos mensajes OFPT (OpenFlow message types) agrupados en su respectiva categoría [28].

Tabla 2. Tipos de Mensajes OFPT en OpenFlow

Tipo de Mensaje	*Categoría*	*Subcategoría*
HELLO	Simétrico	Inmutable
ECHO_REQUEST	Simétrico	Inmutable
ECHO_REPLY	Simétrico	Inmutable
FEATURES_REQUEST	Controlador-switch	Configuración switch
FEATURES_REPLY	Controlador-switch	Configuración switch
GET_CONFIG_REPLY	Controlador-switch	Configuración switch
PACKET_IN	ASYNC	NA
FLOW_REMOVED	ASYNC	NA
PORT_STATUS	ASYNC	NA
ERROR	ASYNC	NA
PACKET_OUT	Controlador-switch	Comando desde controlador
FLOW_MOD	Controlador-switch	Comando desde controlador
STATS_REQUEST	Controlador-switch	Estadísticas

El controlador llena las entradas del switch con las tablas de flujos. El switch al recibir un paquete debe tomar una decisión basada en su tabla de flujos. En caso de no encontrarla, se produce un *table miss* y se reenvía el paquete al controlador (en la versión 1.0). La versión 1.3 expande su uso al agregar la entrada de flujo *table-miss*. El controlador programará esta entrada de flujo en el switch de la misma manera que una entrada de flujo normal. Todos los campos deben ser wildcard para asegurar que todos los paquetes que no encuentren una coincidencia en la tabla de flujo se ejecuten.

La versión 1.2 agrego la acción *set_field* para poder actualizar los campos de la cabecera en las tablas de flujo.

Desde la versión 1.1 se agregó la opción de usar múltiples tablas de flujo. Con esta opción es posible aplazar el procesamiento de los paquetes a las siguientes tablas de flujo (usando la instrucción GOTO). Este proceso es llamado *pipeline*. Ahora las entradas de flujo pueden ser encadenadas por una instrucción en una entrada de flujo apuntando a otra tabla de flujos. Se pueden añadir acciones a una *acción set*. *La acción set* es inicializada y modificada por todas las instrucciones que pasen por el pipeline. Al terminar el pipeline, todas las acciones son ejecutadas [29]. Este proceso de *pipeline* es detallada en la Figura 16.

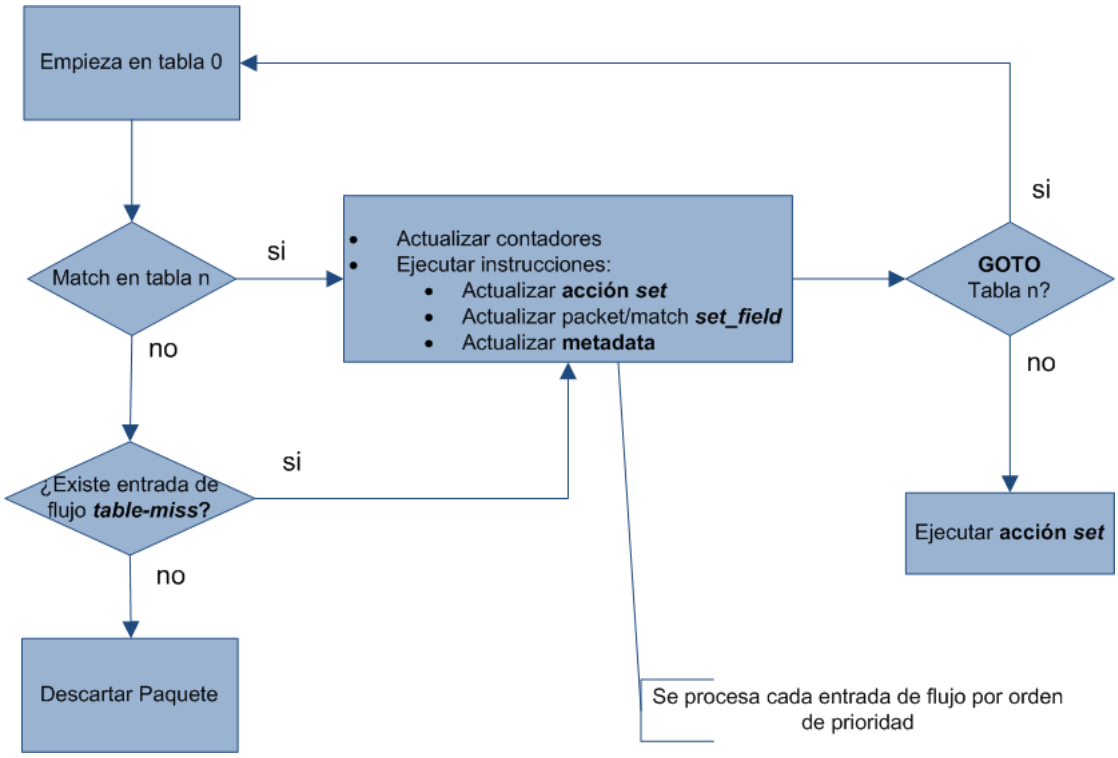


Figura 16. Proceso pipeling de un paquete

Los siguientes términos son importantes conocer para usar el protocolo OpenFlow:

- **Instrucción:** Describe el procesamiento OpenFlow realizado al tener una coincidencia en la entrada de flujo.
- **Acción:** Operación que reenvía un paquete a un puerto o modifica el paquete (Ej: decrementando campo TTL). Pueden ser parte de un set de instrucciones o asociados a una acción bucket con una entrada de grupo.
- **Acción Set:** Conjunto de acciones asociadas a un paquete que son acumuladas mientras el paquete pasa por cada tabla y que son ejecutados cuando la instrucción SET indica al paquete que salga el pipeline.
- **Metadata:** Para pasar información entre tablas.

- **Group:** Lista de acciones bucket.
- **Meter:** Elemento del switch que puede medir y controlar la tasa de paquetes. Desencadena una *meter band* si la tasa de paquetes se excede de un umbral. Si el *meter band* descarta el paquete, es llamado Rate Limiter.

3.1.3 Controladores

El controlador es el componente en las SDN que tiene a inteligencia de todos los elementos de red. El controlador tiene una vista general de la topología de la red, puede controlar cada uno de los elementos de la capa de infraestructura, implementar políticas y además proveer una API northbound para las aplicaciones. Los controladores pueden venir además con su propio conjunto de aplicaciones como funciones de switch (*learning switch*), firewall o balanceadores de carga.

Los componentes principales del controlador están descritos en Figura 17. Los módulos proveen las funciones del controlador y son sus elementos principales. Tiene el southbound API para comunicarse con los elementos de red de la capa de infraestructura. En este caso esta estandarizado el uso del protocolo OpenFlow. Por ultimo tiene la northbound API para comunicarse con las aplicaciones. A diferencia del southbound, la API northbound no está estandarizada (aunque hay esfuerzos por la ONF [30]). Por lo tanto se tienen varias implementaciones (REST API, Python API, JAVA API), y su uso dependerá del controlador usado.

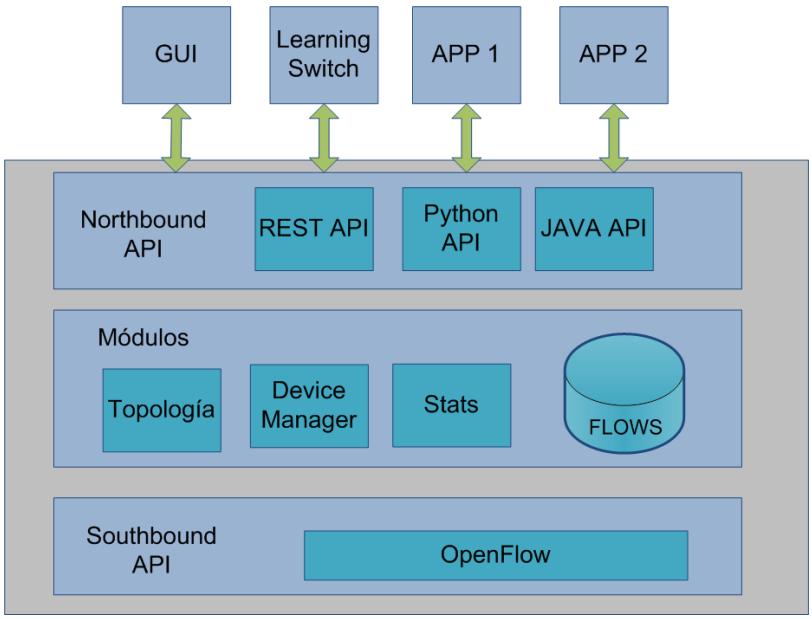


Figura 17. Componentes del controlador SDN

Como se mencionó anteriormente, los módulos describen el funcionamiento del controlador. Las principales funciones son:

- Descubrimiento dispositivos usuario final: Descubrir dispositivos como computadores, impresoras, móviles, etc.
- Descubrimiento dispositivos de red: Descubrimiento de switches, routers, etc.
- Gestión de topología de dispositivos de red: Información de la interconexión de los dispositivos de red.
- Gestión de los flujos: Mantener base de datos de los flujos gestionados por el controlador y realizar coordinación con los dispositivos para asegurar sincronización de las entradas de flujo con la base de datos.

Hay algunas implementaciones disponibles de controladores. Estos pueden ser comerciales u open-source. A continuación se describen los principales:

POX: Fue desarrollado a partir del primer controlador disponible (NOX), el cual fue implementado en C++. POX en cambio se desarrolló usando Python. Permite un rápido desarrollo al tener una curva rápida de aprendizaje. Soporta la versión 1.0 de OpenFlow. No se encuentra documentación actualizada.

OpenDayLight (ODL): Es soportado por el OpenDaylight Foundation. Tiene soporte de las versiones 1.0 y 1.2 de OpenFlow. Tiene integración con OpenStack. Tiene una API REST para interactuar con el gui-web de forma remota. Está basado en java. Para acceder a ODL por el Northbound API, se usa REST y de esa forma usar sus aplicaciones.

RYU: Controlador open-source desarrollado en Python. Soporta versiones de OpenFlow 1.0, 1.1, 1.2, 1.3, 1.4 y 1.5. El código está disponible como licencia Apache 2.0. Tiene soporte con OpenStack.

Floodlight: Es un controlador open-source desarrollado en Java. Su origen fue el controlador Beacon. Soporta las versiones de OpenFlow 1.0 y 1.3 y tiene soporte experimental para la 1.1, 1.2 y 1.4. Posee integración con OpenStack. Utiliza la librería OpenFlowJ-Loxi para implementar OpenFlow, lo que le permite invocar a los métodos independiente de la versión OpenFlow de los switches. Como desventaja, se tiene una alta curva de aprendizaje.

HP Virtual Application Networks (VAN) SDN Controller: Es una solución comercial desarrollada por HP. Compatible con las versiones 1.0 y 1.3 de OpenFlow. Provee una API que permite el uso de aplicaciones de terceros.

Tabla 3. Controladores SDN

Nombre	*Northbound API*	*Licencia*	*Lenguaje*	*Versión OpenFlow*
POX	Ad-hoc API	GPLv3	Python	V 1.0
OpenDaylight	REST, RESTCONF	EPL v1.0	Java	V 1.0 y 1.3
RYU	Ad-hoc API	Apache 2.0	Python	V 1.0 -1.5
Floodlight	REST API	Apache 2.0	Java	V 1.0 y 1.2
HP VAN SDN	REST API	-	Java	V 1.0

3.1.4 Aplicaciones

Las aplicaciones SDN funcionen en una capa superior al controlador, conectándose por medio de la API northbound API, tal como se detalla en la Figura 17. Las aplicaciones SDN se encargan de gestionar las entradas de flujo que son programadas en los dispositivos de red usando la API del controlador para gestionar los flujos (OpenFlow). Todas estas aplicaciones, en una red tradicional son realizadas por los equipos de red sean routers o switches. Entre las funciones a realizar por las aplicaciones se encuentran las siguientes:

- Configurar flujos para encaminar paquetes a través del mejor camino en un enlace.
- Balanceo de carga a través de múltiples caminos.
- Reaccionar a cambios de la topología como fallos de enlace o al agregar o eliminar dispositivos o enlaces.
- Redirigir tráfico para realizar inspección o alguna tarea de seguridad.

Las aplicaciones son manejadas por eventos que vienen del controlador así como entradas externas. Las entradas externas pueden ser sistemas de monitoreo como IDS, NetFlow o incluso peers de BGP. La aplicación SDN funciona como un *listener* para los eventos y el controlador puede invocar un *método* cuando ocurra un evento. Entre los eventos que maneja la aplicación están el descubrimiento de usuarios finales, de dispositivos de red o cuando llega un paquete [28]. En el caso de un evento que llegue un paquete, eso puede ser por una entrada de flujo que dice que envié el paquete al controlador o porque no existe entrada de flujo en el switch.

Se utilizan las API como un medio para obtener información de aplicaciones externas. Una de estas API es la REST API. REST es usada como un método para hacer llamadas entre redes, utiliza HTTP para obtener datos o indicar ejecución de operaciones sobre los datos. Aplicaciones REST se basan en un protocolo cliente/servidor sin estado, un conjunto de operaciones definidas (operaciones de HTTP GET, PUT, POST y DELETE), una sintaxis universal (la URI) y el uso de hipermedios (HTML o XML). La REST API usa un puerto TCP estándar y por lo tanto no tiene problemas al pasar por firewalls.

La aplicación SDN puede responder un evento de varias maneras, dependiendo su programación. Puede ser algo básico como descargar las entradas de flujo para el dispositivo recién descubierto o algo más complejo basándose e información de estado obtenida de otra fuente que no sea el controlador. Existen dos tipos de aplicaciones: Proactiva y Reactivas.

Las aplicaciones reactivas funcionan de acuerdo a llamadas realizadas desde el controlador, cuando un paquete es reenviado al mismo. El controlador está escuchando por medio de su Listener API. Por medio de una Java API se puede registrar listeners y recibir callbacks cuando arriben los paquetes. Los callbacks vienen con metadata que incluye el switch y el puerto en que llego el paquete. Los listeners son escritos en el lenguaje nativo del controlador. Las aplicaciones reactivas trabajan principalmente a bajo nivel y con el protocolo OpenFlow.

Entre las aplicaciones reactivas se encuentran (para el caso de floodlight):

- SwitchListener: Recibe notificaciones cuando se añade un switch o cambio el estado de uno de sus puertos.
- DeviceListener: Cambios en dispositivos de usuario final.
- MessageListener: Obtiene notificaciones cuando un paquete arriba al controlador.

Luego de recibir el evento, la aplicación puede realizar una acción al procesar el paquete como pueden ser:

- Acciones específicas de paquetes: Indicar al switch que envié el paquete a un determinado puerto por ejemplo.
- Acciones específicas de flujos: Programar nuevas entradas de flujo en el switch.

Las aplicaciones proactivas funcionan en un alto nivel a diferencia de las reactivas. Pueden ser implementadas usando APIs nativas o APIs REST. Al no tener que reaccionar a eventos de la red, no necesita un *listener.* En su lugar, las aplicaciones proactivas son estimuladas por fuentes externas. La API REST puede usar datos acerca de la red como dominios, subredes, switches y hosts. Puede proveer una interfaz *flow pusher* para colocar flujos en los switches, donde la última entrada de flujo típicamente es un DROP para descartar los paquetes son coincidencia.

Es posible tener aplicaciones hibridas que sean reactivas y proactivas. En el caso de las aplicaciones proactivas, las aplicaciones basadas en REST pueden estar en el mismo sistema del controlador o en un sistema remoto, gracias a las funciones de HTTP que ofrece REST. En cambio, las aplicaciones reactivas acceden al controlador por APIs nativas.

3.2 Switches basados en Software

A pesar que existen switches que soportan algunas versiones de OpenFlow, y son definidos por el estándar de referencia Wire protocol [34], en este documento se realiza una descripción de los switches basados en software y que serán usados en el desarrollo del presente trabajo.

Hay varios switches en software que pueden ser usados en la actualidad. Pueden ser usados en algún test-bed o para desarrollar aplicaciones de red basadas en OpenFlow. Algunos de estos switches son:

- **Open vSwitch**: Es un switch multicapa open-source con licencia Apache 2.0. Es diseñado para habilitar automatización en la red a través de extensiones y al mismo tiempo soportando protocolos e interfaces de administración como OpenFlow, NetFlow, sFlow, OVSDB, LACP, etc [35].
- **Indigo:** Implementación OpenFlow que funciona sobre switches físicos y usa las características de hardware de un switch Ethernet [36].
- **LINC**: Proyecto Open Source liderado por FlowForwarding. Usa licencia Apache 2.0 y soporta las versiones 1.2,1.3 y 1.4 de OpenFlow [37].
- **OpenWRT**: Convierte un router/Access-point comercial en un switch habilitado para OpenFlow. El módulo OpenFlow debe implementarse como una aplicación de OpenWRT.
- **Of13softswitch**: Compatible con OpenFlow 1.3. Es un switch corriendo en el espacio de usuario [38]. Tiene los siguientes componentes:
    - ofdatapath: Implementación del switch.
    - ofprotocol: Canal seguro entre controlador y switch
    - oflib: Librería como para convertir a OpenFlow 1.3
    - dpctl: Herramienta de configuración desde consola.

Para este trabajo, los dos switches que podrían usarse son Open vSwitch y Of13softswitch al funcionar directamente en software y tener la capacidad de conectar varias máquinas virtuales.

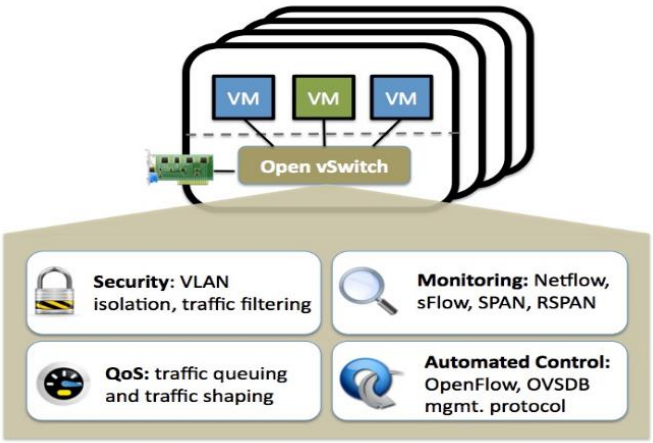


Figura 18. Componentes Open vSwitch

Open vSwitch se compone de tres componentes funcionales que son:

- **ovs-vswitchd:** Demonio que se ejecuta en el espacio de usuario del host anfitrión. Gestiona y controla los switches y se encarga de la comunicación OpenFlow entre los switches y el controlador.
- **ovs kernel module:** Modulo del kernel de Linux usado para reencaminar paquetes de forma rápida que coinciden con alguna de las reglas configuradas por el ovs-vswitchd.
- **ovsdb-server**: Proceso de la base de datos donde se almacena configuración e información de los switches.

3.3 Controlador Floodlight

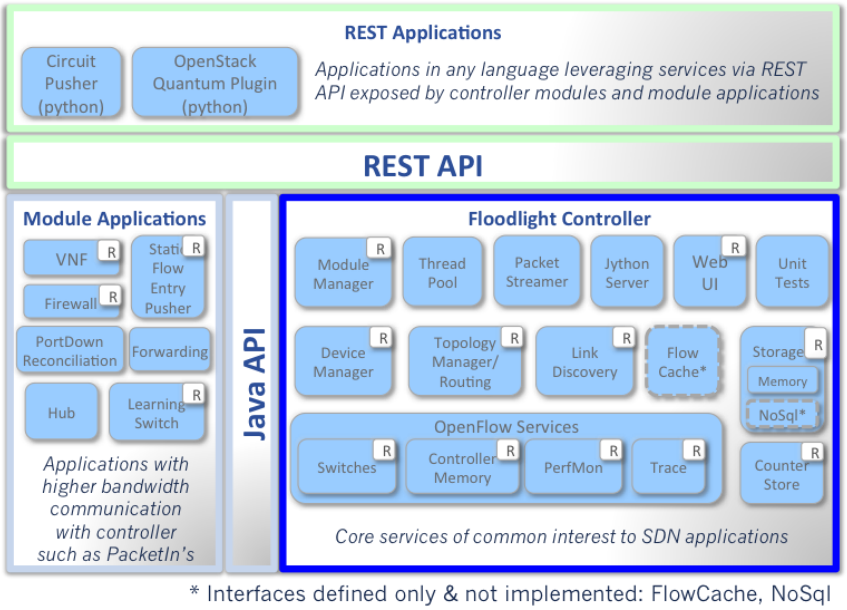


Figura 19. Esquema del controlador Floodlight

Para el desarrollo de este trabajo de fin de master, se ha decidido usar el controlador Floodlight luego de analizar las características expuestas en la **Tabla 3**, revisar las aplicaciones disponibles y la documentación existente.

En la Figura 19 se puede observar la estructura del controlador Floodlight. Se muestran las funciones del núcleo del controlador y para usar las aplicaciones lo hace de dos formas. Puede usar la Java API para usar los módulos de la aplicación o puede usar la REST API y usar aplicaciones que pueden ser llamadas usando servicios REST.

3.3.1 Aplicaciones Floodlight

De detallan las principales aplicaciones utilizadas por el controlador:

Hub:

Es un módulo básico que tiene Floodlight. Permite al switch tener las características de un Hub tal como si se tratara de uno convencional. En el caso que se quiera enviar un paquete, el switch lo envía por todos los puertos menos por el que ingreso.

En el archivo floodlightdefault.properties es necesario realizar lo siguiente antes de compilar nuevamente:

- Eliminar la línea net.floodlightcontroller.forwarding.Forwarding.
- Agregar la línea net.floodlightcontroller.hub.Hub

Learning Switch

Este módulo utiliza las funciones de un switch de capa 2, es decir puede enviar el paquete solo por un puerto y además tiene la posibilidad de almacenar las direcciones MAC de los equipos que están conectados a sus puertos. Además de usarlo como módulo. Esta aplicación puede ser usada con la REST API.

Para habilitarlo es necesario realizar lo siguiente en el archivo floodlightdefault.properties:

- Eliminar línea net.floodlightcontroller.forwarding.Forwarding.
- Agregar línea net.floodlightcontroller.learningswitch.LearningSwitch

Módulo Forwarding

Es una aplicación cargada por default en el controlador para el reenvió de paquetes de forma reactiva. No es necesario realizar cambios en la configuración inicial para habilitar este módulo, a menos que se haya activado anteriormente los módulos de Hub o learning Switch.

En el archive floodlightdefault.properties se especifica los campos en que debe buscar las coincidencias de la siguiente manera:

- net.floodlightcontroller.forwarding.Forwarding.match=vlan, mac, ip, transport

Es posible buscar coincidencias ya sea en la vlan, dirección MAC origen o destino, dirección IP origen o destino y puerto de protocolos de transporte origen o destino

Static Flow Entry Pusher

Esta es una aplicación que funciona de manera proactiva para instalar entradas de flujo específicas a un determinado switch. Es habilitada por default al iniciar el controlador. Es posible usarla con la REST API.

Para añadir una entrada de flujo se utiliza la función curl disponible en las distribuciones Linux. Su modo de uso es la siguiente asumiendo un switch 00:00:00:00:00:00:00:01.

curl -X POST -d '{"switch": "00:00:00:00:00:00:00:01", "name":"flow-mod-1", "cookie":"0", "priority":"32768", "in_port":"1","active":"true", "actions":"output=2"}' http://<controller_ip>:8080/wm/staticflowpusher/json

Para escuchar o aprender las entradas de flujo insertadas de utiliza lo siguiente:

curl http://<controller_ip>:8080/wm/staticflowpusher/list/00:00:00:00:00:00:00:01/json

curl http://<controller_ip>:8080/wm/staticflowpusher/list/all/json

Para limpiar los flujos del switch:

curl http://<controller_ip>:8080/wm/staticflowpusher/clear/00:00:00:00:00:00:00:01/json

curl http://<controller_ip>:8080/wm/staticflowpusher/clear/all/json

Por último, para eliminar una entrada de flujo se realiza lo siguiente:

curl -X DELETE -d '{"name":"flow-mod-1"}' http://<controller_ip>:8080/wm/staticflowpusher/json

3.3.2 Obtener estadísticas de la red usando el controlador

OpenFlow tiene varios mensajes de estadísticas para poder permitir al controlador realizar peticiones a los switches. Entre los mensajes están: flow stats, meter stats, queue stats, aggregate stats, table stats, y port stats.

OpenFlow no provee una forma nativa de obtener el ancho de banda que se está consumiendo. Solo es posible calcular los contadores (por ejemplo de bytes) y realizar el cálculo en base a un tiempo específico.

La separación del plano de datos y control es una desventaja al momento de manejar estadísticas. Su uso dependerá de la aplicación y de la latencia introducida en la red [41]. Es necesario que se actualicen las estadísticas dependiendo el tiempo de la aplicación a usar. Por lo tanto deben tomarse en cuenta estas restricciones:

1. No existe un marcado de la hora en que fueron obtenidas las estadísticas.
2. Latencia del plano de control.
3. Intervalo en que se obtienen las estadísticas

El intervalo en que son obtenidas las estadísticas repercutirá en el nivel de precisión de las mismas. Al obtener las estadísticas en intervalos más cortos se tendrán valores más actualizados pero en cambio se pierde un poco la precisión de las mismas. Al obtener estadísticas en intervalos más grandes de tiempo, no se tienen estadísticas actualizadas pero en cambio se gana en precisión. Es necesario determinar el tiempo correcto dependiendo la aplicación.

En el archivo floodlightdefault.properties se debe realizar lo siguiente:

- En la línea net.floodlightcontroller.statistics.enable=<boolean> cambiar "boolean" por TRUE o FALSE.
- La línea net.floodlightcontroller.statistics.collectionIntervalPortStatsSeconds=<positive-integer> permite ingresar el valor en segundos del tiempo en que se van a obtener las estadísticas.

Es posible obtener los valores de esta forma usando curl y json.tool para tener la información de una manera más legible:

curl http://localhost:8080/wm/statistics/bandwidth/ <switch>/<port>/json -X GET -d '' | python -m json.tool

4 Diseño de escenarios para streaming de video usando SDN

4.1 Escenarios virtuales

4.1.1 Hipervisores

La virtualización es una tecnología actualmente consolidada. Actualmente existe una mayor cantidad de servidores virtuales que físicos. La virtualización de los equipos de red implica que podemos reemplazar los routers/switches en hardware por routers/switches en software.

Los hipervisores permiten a varias máquinas virtuales el uso de los mismos recursos físicos. Estas máquinas virtuales pueden ser migradas de un servidor físico a otro o creadas o destruidas fácilmente.

Existen tres tipos de hipervisores [31]:

- **Tipo 1 (Unhosted):** Un hipervisor de para-virtualización es ejecutado directamente en la plataforma de hardware. Puede soportar sistemas operativos invitados con sus drivers. Ejemplos: VMWare vSphere, VMWare NSX, Microsoft Hyper-V-Server y KVM.
- **Tipo 2 (hosted):** Es un programa que es ejecutado en un Sistema operativo diferente. El sistema operativo invitado es ejecutado sobre el hardware y requiere un emulador a ser ejecutado en el sistema operativo anfitrión. El Sistema operativo invitado no está al tanto que está virtualizado. Ejemplos: Oracle VM VirtualBox, Microsoft Virtual Server, VMWare Workstation y QEMU.
- **A nivel del Sistema Operativo:** Es diferente a los dos tipos anteriores, al correr varias máquinas virtuales simultáneamente. Utiliza un aislador el cual aísla la ejecución de las aplicaciones en un entorno. El kernel de un sistema operativo permite la existencia de múltiples instancias en el espacio de usuario (máquinas virtuales) en lugar de solo una. Son llamados algunas veces como contenedores. No causa sobrecarga pero es muy difícil aislar los entornos. Ejemplos: Linux V-Server, chroot, LXC, BSD, JAIL, Open VZ, Docker.

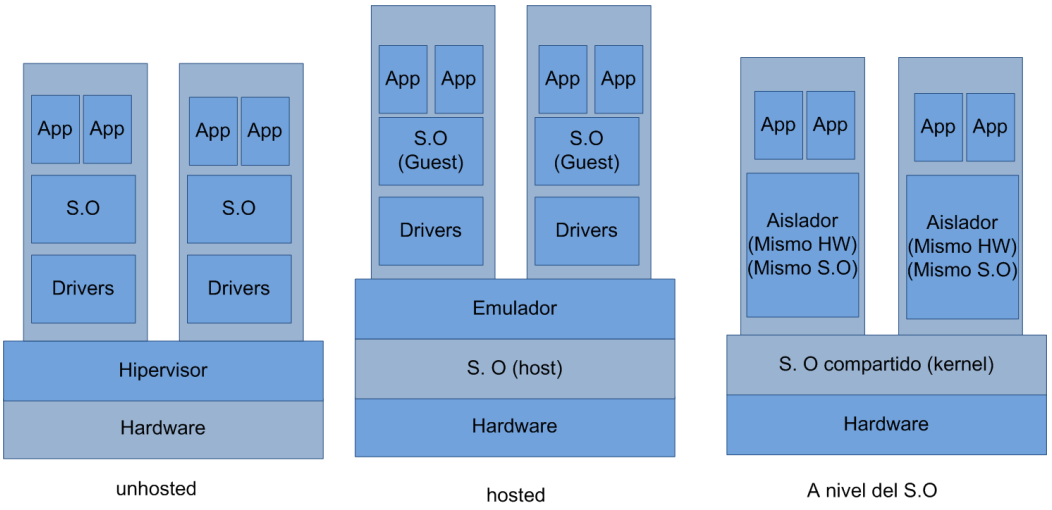


Figura 20. Tipos de virtualización

FlowVisor es una de las primeras tecnologías para virtualización en SDN. Permite a múltiples redes lógicas compartir la misma infraestructura OpenFlow. Puede ser visto como un proxy transparente que intercepta mensajes OpenFlow entre los switches y controladores [22]. Se encarga de particionar el ancho de banda y las tablas de flujo de cada switch, por lo que cada controlador obtiene su propia tabla de flujos virtual en los switches. Su principal objetivo es aislar el tráfico de la red experimental del tráfico de la red de producción [32].

Los contenedores Linux (LXC) son un método de virtualización a nivel del sistema operativo para correr múltiples sistemas Linux aislados con un control en un sistema operativo anfitrión usando el kernel de Linux [33] aprovechando las funciones de cgroups presentes en el mismo.

4.1.2 VNX

VNX (Virtual Networks over Linux) [39] es una herramienta open-source bajo licencia GPL usado para el despliegue de redes virtuales en un equipo físico. Soporta múltiples plataformas de virtualización para Linux como KVM, LXC, XEN, UML, VirtualBox, VMWAre, etc.

VNX facilita realizar pruebas de aplicaciones, servicios y redes usando escenarios compuestos de múltiples máquinas virtuales y redes virtuales desplegadas sobre un sistema operativo anfitrión como Linux. Estas características lo hace una perfecta herramienta para realizar testbeds.

VNX utiliza un archivo XML para crear los escenarios que serán usados para la experimentación. VNX parsea este archivo para crear las máquinas virtuales, redes virtuales y la interconexión requerida. Es posible interactuar en cada máquina virtual por una consola o por SSH. Los usuarios pueden interactuar directamente con las

máquinas virtuales y también se proveen mecanismos para automáticamente ejecutar secuencia de comandos y copia de archivos. Se muestra en la Figura 21 el proceso usado en VNX para el despliegue de un escenario que puede ser usado para experimentación [40].

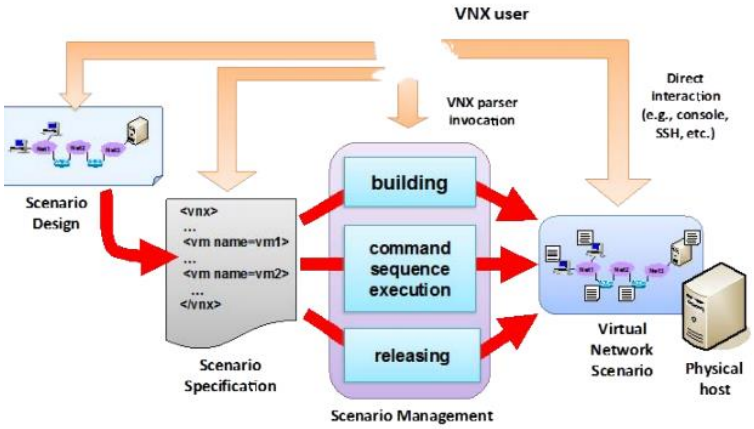


Figura 21. Despliegue de escenario en VNX

4.2 Descripción elementos principales

Es necesario para la realización de cualquier escenario escogido, establecer los siguientes elementos:

- Cliente
- Servidor
- Otros hosts
- Switches
- Controlador SDN

En el escenario realizado utilizando VNX, cada uno de estos elementos será una máquina virtual, y debe tener instalado ciertos elementos, sean programas, servicios o archivos. Algunos pueden estar ya configurados en cada máquina virtual sin necesidad de especificarlo en el archivo XML. Para eso, es necesario modificar los filesystem y agregar los archivos y configurar los programas necesarios. La segunda opción, es cargar los elementos por medio del archivo XML. Esta opción se utilizará para todos los archivos que serán constantemente actualizados en el desarrollo del presente trabajo.

Para todos los hosts se usarán máquinas virtuales de tipo LXC. Este tipo de virtualización es ligera, todos van a compartir el mismo sistema operativo anfitrión. El filesystem usado será un Ubuntu 13.10. Los hosts que estarán incluidos son: clientes, servidores, otros hosts destinados a enviar tráfico por la red y el controlador.

En el caso de los elementos de red se usaran switches Open Vswitch para conectar cada uno de los hosts. Cada switch estará funcionando como un proceso del computador anfitrión. Para emular la red WAN, se utilizará una máquina virtual LXC como tendrá funciones de router.

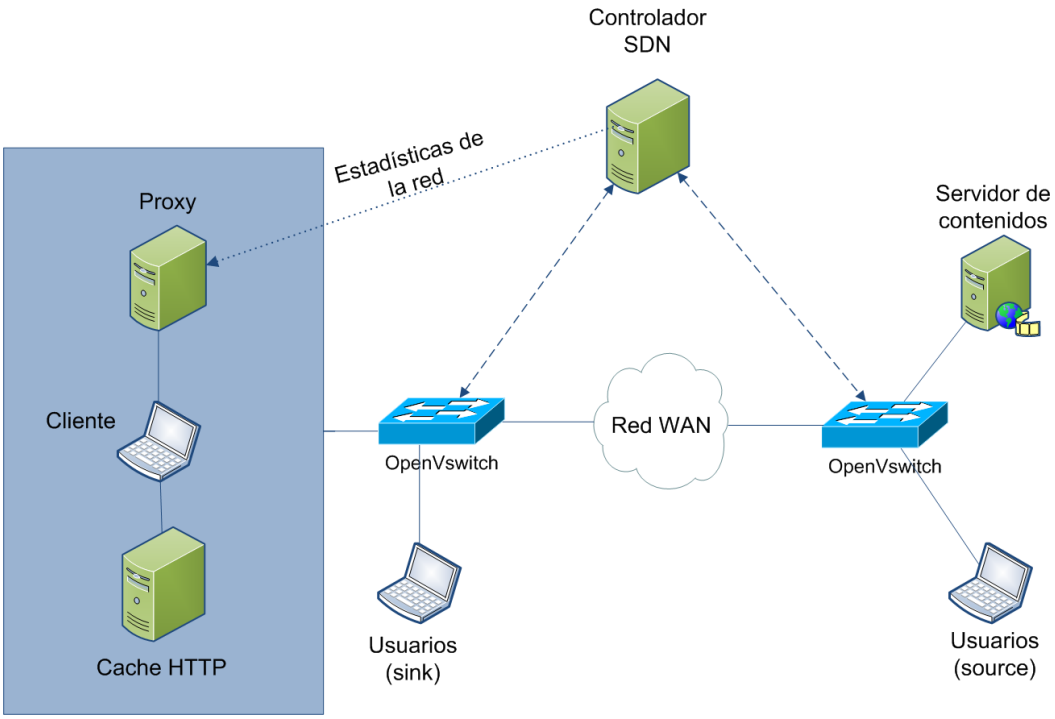


Figura 22. Escenario principal para pruebas de Streaming multimedia

4.2.1 Configuración de los hosts

Cliente

El cliente debe tener los siguientes elementos configurados:

- Servidor Apache: Se debe tener configurado un servidor apache para cuando se el cliente funcione como cache HTTP. Al iniciar el escenario, hay que habilitar el servicio, por lo tanto es necesario agregar ese comando en el archivo XML. Se debe tener almacenado los segmentos de video, aunque en este caso solo se necesitará almacenar para una sola calidad.
- Squid3: El proxy servirá para modificar las peticiones realizadas desde el reproductor DASH en base a la información brindada por el controlador SDN y por un fichero que debe indicar cuando se debe solicitar segmentos al servidor, y cuando al Cache HTTP. El programa para redirigir las peticiones está escrito en Python.

Para visualizar el reproductor DASH, se lo hará desde un navegador google Chrome desde el host anfitrión. Debido a este motivo, el host debe tener conectividad con el cliente, al cual usara como un proxy. El reproductor puede observarse en la Figura 23. Se puede habilitar el navegador por el terminal y luego agregar el manifiesto http://server.ltextreme.com/bbbh/bbbh.mpd.

google-chrome --user-data-dir="/root/chrome-profile/" --disable-web-security --proxy-server="client.ltextreme.com:8080" http://server.ltextreme.com/CUSUM-dashjs13/

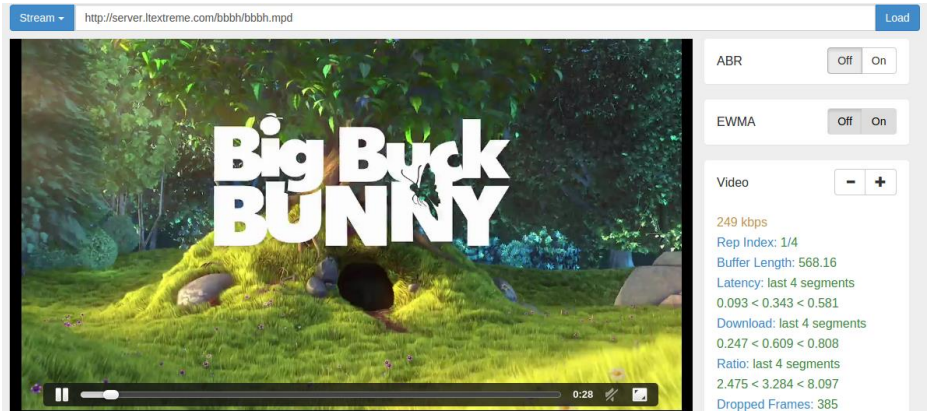


Figura 23 . Reproductor DASH

```
<!--DASH CLIENT - PROXY -->
      <vm name="client" type="lxc">
            <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc_ubuntu_13</filesystem>
            <if id="1" net="Net0" >
                  <ipv4>10.0.0.2/24</ipv4>
            </if>
            <if id="2" net="Net2">
                  <ipv4>10.0.100.5/24</ipv4>
            </if>
            <if id="3" net="MgmtNet">
                  <ipv4>10.0.2.5/24</ipv4>
            </if>
            <route type="ipv4" gw="10.0.0.1">10.0.0.0/16</route>
            <forwarding type="ip"/>
            <filetree seq="on_boot" root="/tmp/">conf/hosts</filetree>
            <filetree seq="on_boot" root="/root/proxy/">conf/proxy/</filetree>
            <filetree seq="on_boot" root="/root/">conf/stats</filetree>
            <exec seq="on_boot" type="verbatim" ostype="system">
            rm -r -f /etc/hosts;
            rm -r -f /etc/squid3/squid.conf;
            mv root/proxy/dash_redirect_bw.py /etc/squid3/;
            mv root/proxy/module_sdn.py /etc/squid3/;
            mv root/proxy/__init__.py /etc/squid3/;
            mv root/proxy/error.txt /etc/squid3/;
            </exec>
            <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
            <exec seq="on_boot" type="verbatim">cat /root/proxy/squid.conf >> /etc/squid3/squid.conf</exec>
            <exec seq="on_boot" type="verbatim" ostype="system">
                  service squid3 start;
                  service apache2 start;
                  rm -r -f /var/www/sintel;
                  rm -r -f /var/www/mpd;
                  rm -r -f /var/www/bbbh;
                  <!--rm -r -f /var/www/bbbhm;-->
                  rm -r -f /var/www/bbbhewma;
                  rm -r -f /var/www/dashjs-backup;
                  rm -r -f /var/www/dashjs;
                  rm -r -f /var/www/dashjs13;
                  squid3 -k reconfigure;
                  service apache2 start;
```

Figura 24. Configuración del cliente en VNX

En la Figura 24, se observa la configuración del cliente para el escenario en VNX. Se subraya en la elipse roja los archivos que deben ser actualizados. El archivo de hosts contiene información del direccionamiento IP y los nombres de cada máquina virtual. Se actualiza el archivo squid.conf con el programa escrito en Python dash_redirect_bw.py. El programa en Python debe solicitar información de las

estadísticas de la red como el ancho de banda disponible al controlador, por lo tanto se agrega el fichero module_sdn.py e _init_.py. El fichero error.txt es necesario para simular el handover entre multicast y unicast. Los segmentos que estén detallados en el fichero, deber solicitados al servidor de contenidos y para los segmentos que no se encuentren se realizarán las peticiones al cache HTTP.

En la elipse roja se borran directorios con contenidos que no son necesarios para la ejecución de los escenarios. Además se inicializa el servidor Apache y se reconfigura el squid que se utiliza para el proxy.

Servidor de contenidos

Para el servidor es necesario tener configurado un servidor Apache, y de la misma manera que el cache HTTP del cliente, debe inicializarse el servicio al empezar el escenario. El servidor debe tener almacenados los segmentos de video, para lo cual se utilizarán 4 calidades de video distintas: 200 kbps, 500 kbps, 1000 kbps y 2000 kbps. En la Figura 25 se observa un esquema resumido del archivo XML usado en el servidor. Se utiliza el dataset de Big Buck Bunny utilizando un video con codificación AVC de 1280 X 720. El video tiene una duración de casi 10 minutos y se utilizan 299 segmentos de dos segundos cada uno.

```
<?xml version="1.0"?>
<!-- MPD file Generated with GPAC version 0.5.1-DEV-rev4193 -->
<MPD type="static" xmlns="urn:mpeg:DASH:schema:MPD:2011" minBufferTime="PT1.5S" mediaPresentationDuration="PT0H9M56.46S"
profiles="urn:mpeg:dash:profile:isoff-main:2011">
 <ProgramInformation moreInformationURL="http://gpac.sourceforge.net">
 <Period start="PT0S" duration="PT0H9M56.46S">
  <AdaptationSet segmentAlignment="true" bitstreamSwitching="true" maxWidth="1280" maxHeight="720" maxFrameRate="24" par="16:9">
   <ContentComponent id="1" contentType="video"/>
   <SegmentList>
   <Representation id="1" mimeType="video/mp4" codecs="avc1.4d401f" width="1280" height="720" frameRate="24" sar="1:1"
   startWithSAP="1" bandwidth="249341" accessTech="unicast">
    <SegmentList timescale="1000" duration="2000">
   </Representation>
   <Representation id="2" mimeType="video/mp4" codecs="avc1.4d401f" width="1280" height="720" frameRate="24" sar="1:1"
   startWithSAP="1" bandwidth="497236" accessTech="multicast">
    <SegmentList timescale="1000" duration="2000">
   </Representation>
   <Representation id="3" mimeType="video/mp4" codecs="avc1.4d401f" width="1280" height="720" frameRate="24" sar="1:1"
   startWithSAP="1" bandwidth="973441" accessTech="unicast">
   <Representation id="4" mimeType="video/mp4" codecs="avc1.4d401f" width="1280" height="720" frameRate="24" sar="1:1"
   startWithSAP="1" bandwidth="1604353" accessTech="unicast">
    <SegmentList timescale="1000" duration="2000">
   </Representation>
  </AdaptationSet>
 </Period>
```

Figura 25. Esquema archivo MPD utilizado

Puede observarse en la Figura 26 como se configura el escenario para VNX. Se puede observar cómo se actualiza el archivo de hosts de acuerdo al direccionamiento IP usado en el escenario y como se deshabilita el squid al no ser necesario en el servidor y se inicializa el servicio Apache.

```
<!-- CONTENT SERVER -->
        <vm name="server" type="lxc">
                <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc_ubuntu_13</filesystem>
                <if id="1" net="Net1">
                        <ipv4>10.0.1.2/24</ipv4>
                </if>
                <route type="ipv4" gw="10.0.1.1">10.0.0.0/16</route>
                <forwarding type="ip" />
                <filetree seq="on_boot" root="/tmp/">conf/hosts</filetree>
                <filetree seq="on_boot" root="/root/">conf/stats</filetree>
                <exec seq="on_boot" type="verbatim" ostype="system">
                rm -r -f /etc/hosts;
                </exec>
                <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
                <exec seq="on_boot" type="verbatim" ostype="system">
                        service squid3 stop;
                        service apache2 start;
                </exec>
```

Figura 26. Configuración del servidor en el escenario de VNX

Otros hosts

Se configuran dos hosts, fuente (*source*) y sumidero (*sink*) que servirán para inyectar tráfico a la red que servirá para realizar pruebas de congestión y poder evaluar el algoritmo de adaptación. Es necesario que las ambas máquinas virtuales tengan configurado iperf para realizar las pruebas. La configuración de las máquinas virtuales en VNX se detalla en la Figura 27 Debido a que el trafico viene desde el servidor al cliente (tráfico de descarga), entonces el source debe ir del lado del servidor y el sink del lado de la red del cliente para que afecte al tráfico en el mismo sentido.

```
<vm name="sink" type="lxc">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc_ubuntu_13</filesystem>
    <if id="1" net="Net0">
      <ipv4>10.0.0.3/24</ipv4>
    </if>
    <route type="ipv4" gw="10.0.0.1">default</route>
    <filetree seq="on_boot" root="/tmp/">conf/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="on_boot" type="verbatim">iperf -s -u -p 5001 &amp;</exec>
  </vm>

<vm name="source" type="lxc">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc_ubuntu_13</filesystem>
    <if id="1" net="Net1">
      <ipv4>10.0.1.3/24</ipv4>
    </if>
    <route type="ipv4" gw="10.0.1.1">default</route>
    <filetree seq="on_boot" root="/tmp/">conf/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="send3000" type="verbatim">iperf -c 10.0.0.3 -p 5001 -u -t 60 -b 3000k</exec>
    <exec seq="send500" type="verbatim">iperf -c 10.0.0.3 -p 5001 -u -t 60 -b 500k</exec>
    <exec seq="send1000" type="verbatim">iperf -c 10.0.0.3 -p 5001 -u -t 60 -b 1000k</exec>
    <exec seq="send1500" type="verbatim">iperf -c 10.0.0.3 -p 5001 -u -t 60 -b 1000k</exec>
  </vm>
```

Figura 27. Configuración de hosts para realizar pruebas de congestión

Controlador

Para la configuración del controlador, es necesario antes configurar el fichero floodlightdefault.properties de acuerdo a los módulos que se necesiten para la ejecución del escenario. En el caso del escenario utilizando un solo switch para la red del cliente y para la red del servidor se debe mantener el módulo Forwarding tal como fue explicado en la sección 3.3.1. Se habilita el módulo de estadísticas y se configura el tiempo en que va a recolectar los datos en 5 segundos. Finalmente se debe compilar el código y los siguientes ficheros deben ser copiados para que al momento de la ejecución del escenario se carguen en la máquina virtual del controlador:

- floodlightdefault.properties: Módulos del controlador.
- floodlight.jar : Archivo de java que permitirá ejecutar el controlador.
- logback.xml: Fichero para configurar como se muestran los logs en la consola.

Para iniciar el controlador, se puede hacerlo desde el host por medio de comandos de VNX de acuerdo a lo mostrado en la Figura 28 con el comando start_ctrl: ***vnx –f ESCENARIO.XML –v –x start_ctrl***, o se puede realizarlo directamente desde el controlador de la siguiente forma:

java -Dlogback.configurationFile=/root/floodlight/logback.xml -jar /root/floodlight/floodlight.jar -cf /etc/floodlight/floodlightdefault.properties

Por consola se pueden revisar los logs del controlador de acuerdo a lo configurado en el fichero logback.xml. Se pueden definir niveles de logs como ERROR, INFO, WARN, DEBUG, y TRACE. En la Figura 29 se puede observar los logs que aparecen al iniciar el controlador. En la figura se han separado los tipos de mensajes, se observa primero como se habilita el módulo de Forwarding y luego el módulo de recolección de estadísticas con un tiempo de 5 segundos. Así mismo se habilita un thread que estará realizando esta tarea mientras este ejecutándose el controlador. También se observa cómo se habilita el módulo OFSwitchManager que sirve para gestionar las conexiones de todos los switches OpenFlow que se conectan al controlador.

```
<!-- Controlador SDN (floodlight) -->
<vm name="sdnctrl" type="lxc" arch="x86_64">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc_ubuntu64</filesystem>
    <if id="1" net="MgmtNet">
      <ipv4>10.0.2.2/24</ipv4>
    </if>
    <filetree seq="on_boot" root="/tmp/">conf/hosts</filetree>
    <filetree seq="on_boot" root="/root/floodlight/">conf/sdnctrl/</filetree>

    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="on_boot" type="verbatim">
       mkdir -p /etc/floodlight;
       ln -s /root/floodlight/floodlightdefault.properties /etc/floodlight/;
    </exec>
    <exec seq="on_boot" type="verbatim">chmod +x /root/floodlight/start-floodlight</exec>
    <!--exec seq="on_boot,start_ctrl" type="verbatim">/root/floodlight/start-floodlight</exec-->
    <exec seq="start_ctrl" type="verbatim">
    java -Dlogback.configurationFile=/root/floodlight/logback.xml -jar /root/floodlight/floodlight.jar -cf /etc/floodlight/floodlightdefault.properties &amp;
    </exec>
    <exec seq="stop_ctrl" type="verbatim">pkill -f "java -Dlogback.configurationFile"</exec>

</vm>
```

Figura 28. Configuración controlador en escenario VNX

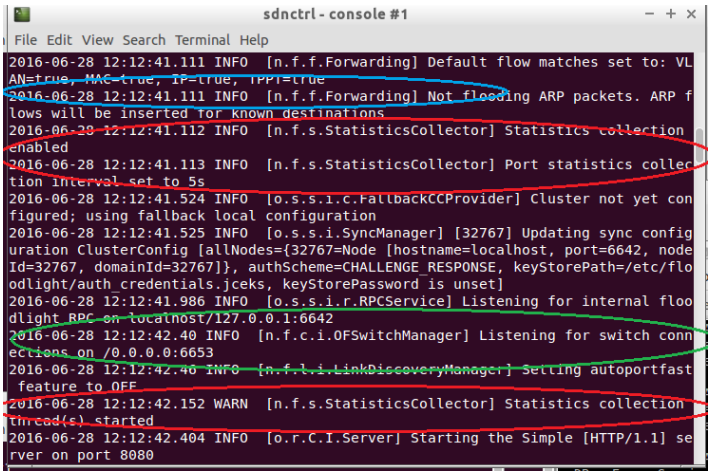


Figura 29. Logs del controlador

Switches:

Para la configuración de los switches en el escenario con VNX es necesario habilitar lo siguiente:

- Nombre: Nombre en que será descrito en el escenario.
- Modo: En el caso de los switches a las que se van a conectar el cliente, servidor y los hosts usados para inyectar tráfico, se debe especificar el uso del Open Vswitch al tener que conectarse con el controlador. En el caso de elementos de red se puede especificar la conexión con 'virtual_bridge' que configurará un Bridge de Linux.
- Hardware Address: Se configura una dirección MAC con la que será identificado el switch.
- Controlador: SE especifica la dirección IP del controlador y el puerto TCP en que está escuchando. En el caso de floodlight debe especificarse el puerto 6653.
- Fail Mode: En este caso debe especificarse el modo 'secure' para que únicamente el controlador pueda instalar reglas a los switches. El otro caso sería 'standalone' donde las reglas se instalan automáticamente sin necesidad del controlador, siendo la opción por defecto.

La configuración final debe quedar de la siguiente manera, si se utiliza un solo switch por red:

<net name="Net0" mode="openvswitch" hwaddr="00:00:00:00:00:01" controller="tcp:10.0.2.2:6653" fail_mode='secure' of_version="OpenFlow13" />

<net name="Net1" mode="openvswitch" hwaddr="00:00:00:00:00:02" controller="tcp:10.0.2.2:6653" fail_mode='secure' of_version="OpenFlow13" />

Emulación red WAN

Al realizar la implementación del escenario en un ambiente virtualizado donde cada máquina virtual esta interconectado, se están usando características de red ideales, es decir sin considerar ancho de banda, latencia o jitter.

Para la realización de este escenario no se han considerado valores que coincidan con un medio de acceso específico, pero se ha considerado emular estos valores para poder sacar mejores conclusiones en las simulaciones realizadas. Es posible cambiar estos parámetros a valores dependientes de la red de acceso posteriormente.

En las simulaciones se ha definido una máquina virtual que se la ha nombrado Router1. Además de funcionar como router de capa 3 y separar las redes del cliente y del servidor, se lo utiliza para modelar la red WAN.

El stack de protocolos de comunicaciones de kernel de Linux provee capacidades de control de tráfico y de shapping. El paquete ***iproute2*** tiene el comando **tc** que nos ayudará a realizar la emulación de red WAN deseada. Para la configuración del shapping del ancho de banda se utiliza Hierarchical Token Bucket (HTB) para limitar la cantidad de ancho de banda de la siguiente manera:

$TC qdisc add dev $IF root handle 1: htb default 30
$TC class add dev $IF parent 1: classid 1:1 htb rate $DNLD
$TC class add dev $IF parent 1: classid 1:2 htb rate $UPLD

La primera línea crea un qdisc (discípula de encolamiento) para HTB en la interfaz de la variable IF y especifica que se usará como default la clase 30. La siguientes dos líneas línea configuran las clases tanto para el tráfico de bajada como de subida de acuerdo a las variables DNLD y UPLD respectivamente.

Para configurar el retardo y la pérdida de paquetes se utiliza NETEM [42]. Se configura el retardo en la variable d1 con una variación de ±dv. Debido a que típicamente el retardo en una red no es uniforme, se utiliza una distribución normal para modelarla. En lo que se refiere a la perdida de paquetes, se utiliza la variable l1 con una variación del 25%. Las siguientes dos líneas especifican los comandos usados:

$TC qdisc add dev $IF parent 1:1 netem delay $d1 $dv distribution normal loss 0.3% 25%
$TC qdisc add dev $IF parent 1:2 netem delay $d1 $dv distribution normal loss 0.3% 25%

Por último, se especifica las direcciones IP origen y destino por medio de la variable IP

$U32 match ip dst $IP/16 flowid 1:1
$U32 match ip src $IP/16 flowid 1:2

4.2.2 Escenario de pruebas utilizando un solo cliente

En la Figura 22 que fue mostrada al inicio de este capítulo, se especificó el escenario usado para realizar las pruebas de la distribución de contenido multimedia. Se pueden observar todos los elementos descritos en la anterior sección. El controlador es el encargado de insertar las tablas de flujo a los switches, y debe estar escuchando ya sea las peticiones que estos elementos pueden realizarle al no tener una acción establecida en sus tablas de flujo o también para tomar las estadísticas de los switches. Estas estadísticas deben ser pasadas al proxy, el cual está configurado en la máquina virtual del cliente y debe redirigir las peticiones solicitadas por el reproductor al servidor. Dentro de la máquina virtual del cliente también debe estar un servidor HTTP que almacenera el contenido multimedia en cache.

La red WAN se encarga se separar ambas redes por medio de un router y es emulada para tener características reales de la red. Los usuarios que deben conectarse a los switches tienen como misión enviar tráfico, el mismo que se originará desde la red del servidor (source) hasta el host de la red del cliente (sink). La distribución de los switches puede cambiarse dependiendo las pruebas. En caso que solo se necesite realizar pruebas de distribución de contenido sin modificar los caminos (paths) entonces solo se necesita un switch por red. Caso contrario, puede cambiarse la distribución para usar una isla de switches OpenFlow en otro escenario.

Aunque no está en la gráfica, el host anfitrión participa también en el escenario. En este caso se puede decir que emula el computador colocado en el cliente, ya que desde el host vamos a usar el navegador Google Chrome para abrir el reproductor de video dash.js [43]. Para que el controlador tenga conectividad con el cliente, se configuró un enlace entre ellos por medio de un *virtual bridge.* Hay que tener en cuenta que el controlador usa la misma red del host. El direccionamiento IP usado para cada uno de estos elementos se describe en la Tabla 4.

Tabla 4. Direccionamiento IP usado

Elemento	*Dirección IP*
Cliente	10.0.0.2/24
sink	10.0.0.3/24
Router (eth1)	10.0.0.1/24
Router (eth2)	10.0.1.1/24
Servidor	10.0.1.2/24
Source	10.0.1.3/24
Controlador SDN	10.0.2.2/24
Host anfitrión	10.0.2.3/24
Cliente (Conexión a Controlador)	10.0.2.5/24
Cliente (Conexión a host)	10.0.100.5/24
Host (Conexión a cliente)	10.0.100.4/24

4.2.3 Escenario de pruebas utilizando varios clientes

Para la realización de este escenario se usaron los elementos descritos en la anterior sección con excepción de los hosts usados para generar tráfico como sink y source. En su lugar, se colocarán dos clientes DASH adicionales con sus respectivos proxys. El objetivo de este escenario es evaluar equidad (*fairness*) entre los distintos clientes que comparten recursos de la red y un mismo cuello botella para obtener un contenido determinado. Los tres clientes en este escenario deben conectarse a un switch, el mismo que se conectará a la red WAN. El servidor de contenidos será único para la realización de las pruebas. En la Figura 30 se puede observar la descripción de los elementos involucrados.

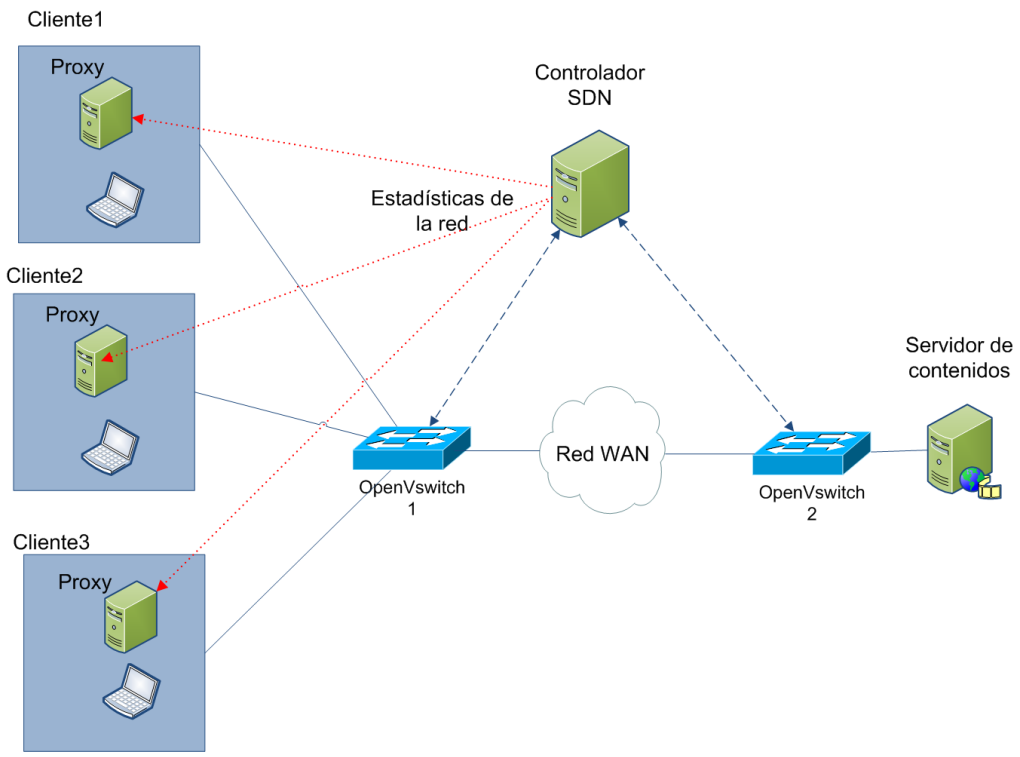


Figura 30. Escenario de pruebas utilizando tres clientes

El algoritmo de adaptación usado será el mismo en cada uno de los clientes y escogerá los segmentos dependiendo del ancho de banda de la interfaz del switch que se conecta a la WAN. Por lo tanto dependerá del tráfico combinado del video de los tres usuarios.

4.2.4 Escenario de pruebas usando diferentes caminos de red

Para este escenario se utilizarán las características que tiene el controlador SDN para insertar las entradas de flujo a los switches de forma proactiva. En los escenarios anteriores se insertaban las entradas de flujo cuando llegaba un paquete al switch y este no sabía a qué puerto reenviarlo, por lo que lo enviaba al controlador por medio de un mensaje PACKET_IN y este le reenviaba por medio de un mensaje PACKET_OUT. Ahora se va a proceder a insertar las tablas de flujo en los switches antes que lleguen los paquetes al switch.

En lugar se borrar el módulo Forwarding del fichero floodligthdefautproperties, se va a proceder a crear entradas de flujo con la prioridad más alta para que sean esas reglas las que tengan que usar cada switch.

Para insertar los flujos en los switches, se usara el módulo *Static Flow Entry Pusher* el cual fue descrito su uso en la sección 3.3.1. Es posible usar un código de Python tal como se usó para el cálculo de estadísticas en la máquina virtual del cliente con module_sdn.py, pero en este caso se usará *curl* para solicitar al controlador que realice los cambios en base a un script que primero deberá borrar las entradas de flujo actuales y luego crear las nuevas.

Se configuraron dos scripts, uno para cada camino que se sigue desde el cliente hasta el router que separa las dos redes. El usuario puede ejecutar cada uno de los scripts y validar si se han insertado las entradas de flujo en cada switch, y a continuación proceder a validar la distribución de contenidos, teniendo también la posibilidad de agregar tráfico adicional entre *source* y *sink* por medio del comando iperf.

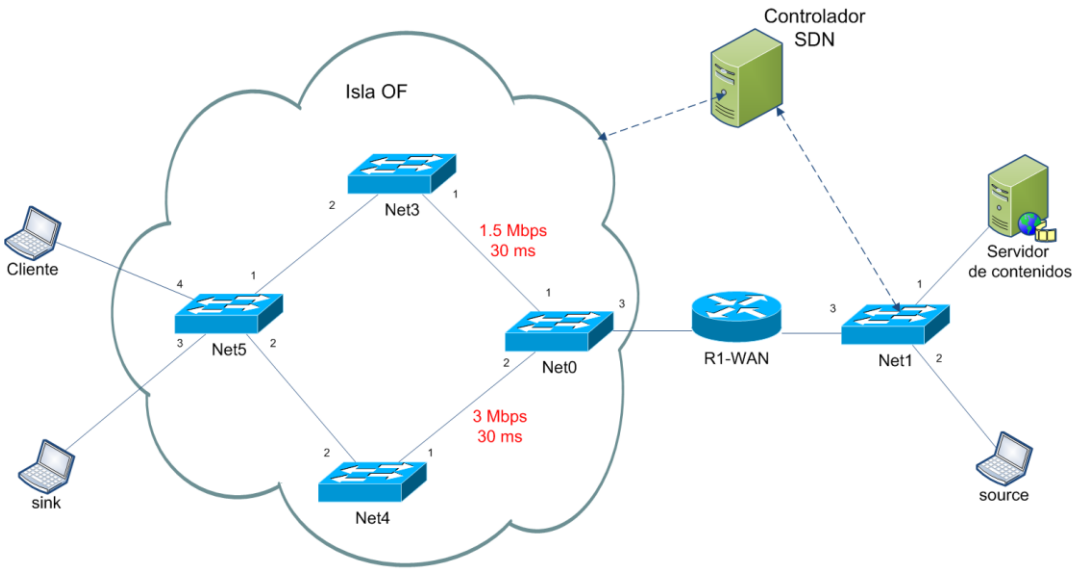


Figura 31. Escenario de pruebas usando diferentes caminos de red

En la Figura 31 se puede observar el diseño del escenario, usando los nombres de switch y números de puerto tal como están configurados en VNX. Las direcciones MAC para identificar cada switch son los siguientes:

Net0: 00:00:00:00:00:01
Net1: 00:00:00:00:00:02
Net3: 00:00:00:00:00:03
Net4: 00:00:00:00:00:04
Net5: 00:00:00:00:00:05

Los 4 switches agrupados en la red del cliente de streaming forman una isla OpenFlow y las reglas de flujo están diseñadas para que siga el camino (path1) Net5-Net3-Net1 o el camino (path2) Net5-Net4-Net1. Utilizando el mismo script utilizado para emular la red WAN en los escenarios anteriores, se procede a configurar valores preestablecidos de ancho de banda y retardo en el puerto 1 y puerto 2 del switch Net0. Se utilizan las interfaces del host net0-3-0 y net0-4-0 para cada caso. El segundo camino tendrá el doble del ancho de banda disponible. De esta manera el camino path2 tendrá mejores condiciones para pasar el streaming de video.

Al realizar el cambio del camino de red, hay que realizar un cambio en el programa que ejecuta la redirección al servidor dash_redirect_bw.py para que refleje ahora las mediciones en los puertos del switch antes mencionados.

4.3 Conectividad del escenario

4.3.1 Pruebas de conectividad

Antes de comenzar las pruebas de streaming de video, es necesario validar el correcto funcionamiento de la red utilizada. Luego de validar los logs del controlador de la Figura 29, y validar que se hayan cargado los módulos necesarios, se procede a realizar pruebas ICMP entre el cliente y el servidor. Después de observar que el ping se realizó exitosamente, se procede a revisar la captura realizada con el software wireshark. En la Figura 32 se muestra la captura filtrada para los mensajes OpenFlow. Se observa que el primer mensaje PACKET_IN es el enviado por el switch al controlador al no tener en su tabla de flujo la acción a realizar con el paquete. El controlador responde el ICMP REQUEST al switch con un mensaje PACKET_OUT con una acción que le indica que si el paquete viene del puerto 3, debe ser reenviado al puerto 2 del switch. Con esta información el switch reenvía el paquete al puerto respectivo, el cual lo enviara al servidor, por lo que tendrá que pasar por la red WAN y por el switch OVS2, por lo que se repetirá en este switch lo expuesto anteriormente. Al

enviar su respuesta el servidor, el paquete pasa al switch OVS2 y este envía el mansaje al controlador que responderá con un mensaje que será insertado en la tabla de flujo del switch con una acción que indique que si viene del puerto 2 el paquete, entonces debe reenviarse al puerto 2, tal como se aprecia en el mensaje OpenFlow de la figura. Lo mismo se repetirá en el OVS1 para el ICMP REPLY y con eso ya se tendrán insertadas las entradas en la tabla de flujo de cada switch.

Las entradas de flujo de cada switch pueden ser consultadas con el siguiente comando de Open Vswitch: **ovs-ofctl -O OpenFlow13 dump-flows Net0.** Net0 vendría a ser el nombre asignado en el escenario de VNX. De la misma manera se pueden consultar las tablas de flujo en la interfaz web del controlador de la siguiente manera: http://10.0.2.2:8080/ui/index.html. Esta información puede observarse en la Figura 33, luego de que se ha enviado también tráfico entre las máquinas virtuales sink y source. Otra forma de revisar el contenido de las tablas de flujo, es realizando una consulta vía la REST API al controlador utilizando lo siguiente: **curl http://10.0.2.2:8080/wm/core/switch/all/flow/json | python -m json.tool.**

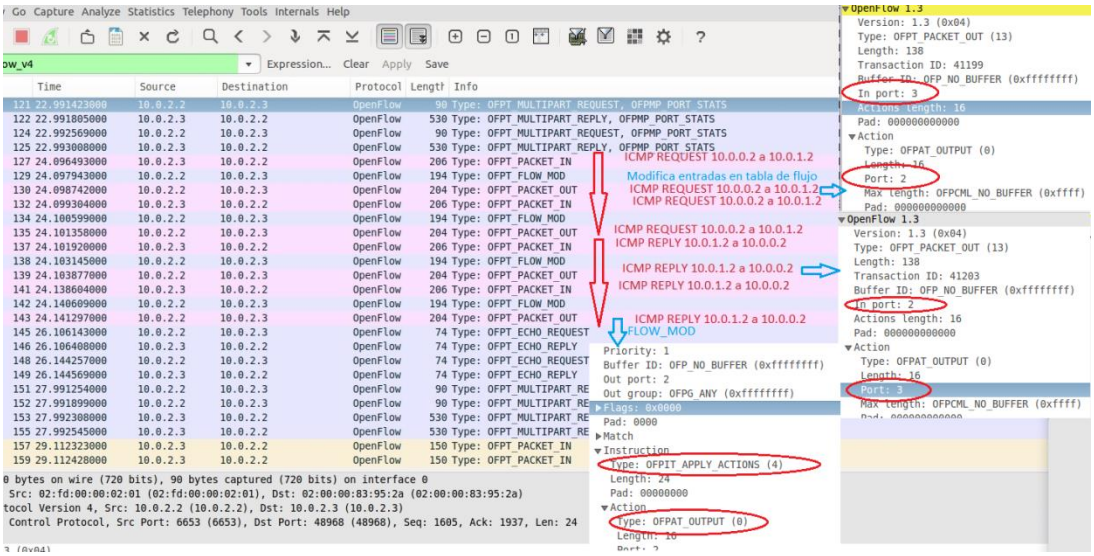


Figura 32. Captura ICMP entre cliente y servidor

Switch OVS1

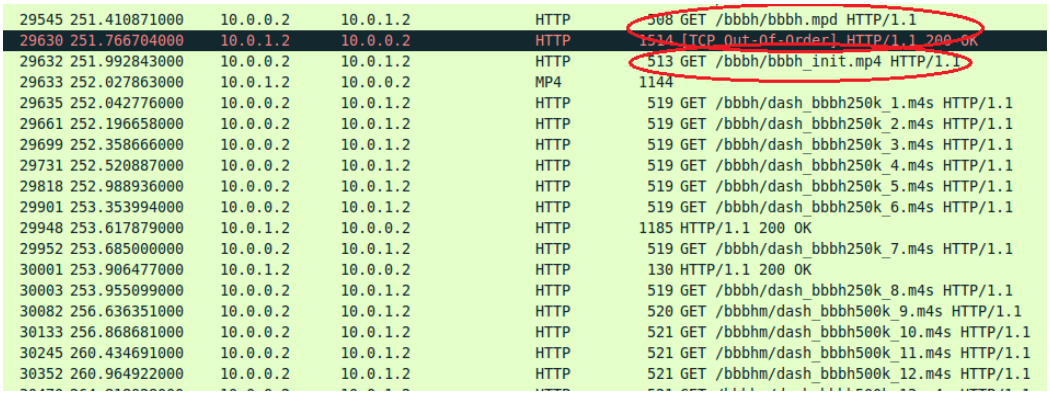
Cookie	Table	Priority	Match	Apply Actions	Write Actions	Clear Actions	Goto Group	Goto Meter	Write Metadata	Experimenter	Packets	Bytes	Age (s)	Timeout (s)
0	0x0	0		actions:output=controller	---	---	---	---	---	---	3744	3445121	25083	0
9007199254740992	0x0	1	in_port=3 eth_dst=02:fd:00:00:04:01 eth_src=02:fd:00:00:05:01 eth_type=0x0x800 ipv4_src=10.0.0.2 ipv4_dst=10.0.1.2	actions:output=2	---	---	---	---	---	---	420	41160	421	5
9007199254740992	0x0	1	in_port=2 eth_dst=02:fd:00:00:00:01 eth_src=02:fd:00:00:04:01 eth_type=0x0x800 ipv4_src=10.0.1.3 ipv4_dst=10.0.0.3	actions:output=1	---	---	---	---	---	---	12	1176	12	5
9007199254740992	0x0	1	in_port=2 eth_dst=02:fd:00:00:05:01 eth_src=02:fd:00:00:04:01 eth_type=0x0x800 ipv4_src=10.0.1.2 ipv4_dst=10.0.0.2	actions:output=3	---	---	---	---	---	---	420	41160	421	5
9007199254740992	0x0	1	in_port=1 eth_dst=02:fd:00:00:04:01 eth_src=02:fd:00:00:00:01 eth_type=0x0x800 ipv4_src=10.0.0.3 ipv4_dst=10.0.1.3	actions:output=2	---	---	---	---	---	---	12	1176	12	5

Switch OVS2

Cookie	Table	Priority	Match	Apply Actions	Write Actions	Clear Actions	Goto Group	Goto Meter	Write Metadata	Experimenter	Packets	Bytes	Age (s)	Timeout (s)
0	0x0	0		actions:output=controller	---	---	---	---	---	---	3744	3445121	25083	0
9007199254740992	0x0	1	in_port=3 eth_dst=02:fd:00:00:04:01 eth_src=02:fd:00:00:05:01 eth_type=0x0x800 ipv4_src=10.0.0.2 ipv4_dst=10.0.1.2	actions:output=2	---	---	---	---	---	---	420	41160	421	5
9007199254740992	0x0	1	in_port=2 eth_dst=02:fd:00:00:00:01 eth_src=02:fd:00:00:04:01 eth_type=0x0x800 ipv4_src=10.0.1.3 ipv4_dst=10.0.0.3	actions:output=1	---	---	---	---	---	---	12	1176	12	5
9007199254740992	0x0	1	in_port=2 eth_dst=02:fd:00:00:05:01 eth_src=02:fd:00:00:04:01 eth_type=0x0x800 ipv4_src=10.0.1.2 ipv4_dst=10.0.0.2	actions:output=3	---	---	---	---	---	---	420	41160	421	5
9007199254740992	0x0	1	in_port=1 eth_dst=02:fd:00:00:04:01 eth_src=02:fd:00:00:00:01 eth_type=0x0x800 ipv4_src=10.0.0.3 ipv4_dst=10.0.1.3	actions:output=2	---	---	---	---	---	---	12	1176	12	5

Figura 33. Tablas de flujo de cada switch

Usando wireshark también podemos revisar las peticiones realizadas desde el cliente/proxy al servidor de video. De esta manera se puede validar el correcto funcionamiento del proxy y que se envíen correctamente las peticiones de cada segmento de video DASH. Las capturas realizas por wireshark nos han ayudado para validar problemas que han ocurrido en el desarrollo de este trabajo, como han sido tener solicitudes repetidas, en desorden o cuando el servidor no ha estado enviando los segmentos solicitados. Por último, una manera de confirmar las peticiones que le llega al servidor es revisar los logs en el servidor Apache, ya sea en el servidor de contenidos o en el cache HTTP del cliente.



29545	251.410871000	10.0.0.2	10.0.1.2	HTTP	508 GET /bbbh/bbbh.mpd HTTP/1.1
29630	251.766704000	10.0.1.2	10.0.0.2	HTTP	1514 [TCP Out-Of-Order] HTTP/1.1 200 OK
29632	251.992843000	10.0.0.2	10.0.1.2	HTTP	513 GET /bbbh/bbbh_init.mp4 HTTP/1.1
29633	252.027863000	10.0.1.2	10.0.0.2	MP4	1144
29635	252.042776000	10.0.0.2	10.0.1.2	HTTP	519 GET /bbbh/dash_bbbh250k_1.m4s HTTP/1.1
29661	252.196658000	10.0.0.2	10.0.1.2	HTTP	519 GET /bbbh/dash_bbbh250k_2.m4s HTTP/1.1
29699	252.358666000	10.0.0.2	10.0.1.2	HTTP	519 GET /bbbh/dash_bbbh250k_3.m4s HTTP/1.1
29731	252.520887000	10.0.0.2	10.0.1.2	HTTP	519 GET /bbbh/dash_bbbh250k_4.m4s HTTP/1.1
29818	252.988936000	10.0.0.2	10.0.1.2	HTTP	519 GET /bbbh/dash_bbbh250k_5.m4s HTTP/1.1
29901	253.353994000	10.0.0.2	10.0.1.2	HTTP	519 GET /bbbh/dash_bbbh250k_6.m4s HTTP/1.1
29948	253.617879000	10.0.1.2	10.0.0.2	HTTP	1185 HTTP/1.1 200 OK
29952	253.685000000	10.0.0.2	10.0.1.2	HTTP	519 GET /bbbh/dash_bbbh250k_7.m4s HTTP/1.1
30001	253.906477000	10.0.1.2	10.0.0.2	HTTP	130 HTTP/1.1 200 OK
30003	253.955099000	10.0.0.2	10.0.1.2	HTTP	519 GET /bbbh/dash_bbbh250k_8.m4s HTTP/1.1
30082	256.636351000	10.0.0.2	10.0.1.2	HTTP	520 GET /bbbhm/dash_bbbh500k_9.m4s HTTP/1.1
30133	256.868681000	10.0.0.2	10.0.1.2	HTTP	521 GET /bbbhm/dash_bbbh500k_10.m4s HTTP/1.1
30245	260.434691000	10.0.0.2	10.0.1.2	HTTP	521 GET /bbbhm/dash_bbbh500k_11.m4s HTTP/1.1
30352	260.964922000	10.0.0.2	10.0.1.2	HTTP	521 GET /bbbhm/dash_bbbh500k_12.m4s HTTP/1.1

Figura 34. Captura de peticiones HTTP realizadas desde el cliente

Como último método para validar la conectividad y el correcto funcionamiento del streaming multimedia, se han habilitado varios mensajes de logging. Se muestra en la Figura 35 un ejemplo de la reproducción de un escenario utilizando varios caminos de red. Se puede observar el terminal de la izquierda donde están los mensajes de log que se sacan del programa que ejecuta el proxy para solicitar los segmentos. Se muestran datos relevantes como el número de segmento, la URL del reproductor antes de pasar por el proxy, el contador de cambios de calidad, la URL solicitada el servidor de contenidos y el ancho de banda medido en la red por el controlador SDN. En el terminal de la derecha se tienen los mensajes sacados de las medidas de ancho de banda del switch Net0 y Net5 que serán de ayuda para poder comprobar cuando se cambia el camino de red y para diferenciar el trafico enviado de video y el que se envía por iperf. Por ultimo en la parte inferior se tienen los mensajes de logs del reproductor dash.js. De esa información se puede sacar el nivel de buffer que se tiene mientras progresa la reproducción.

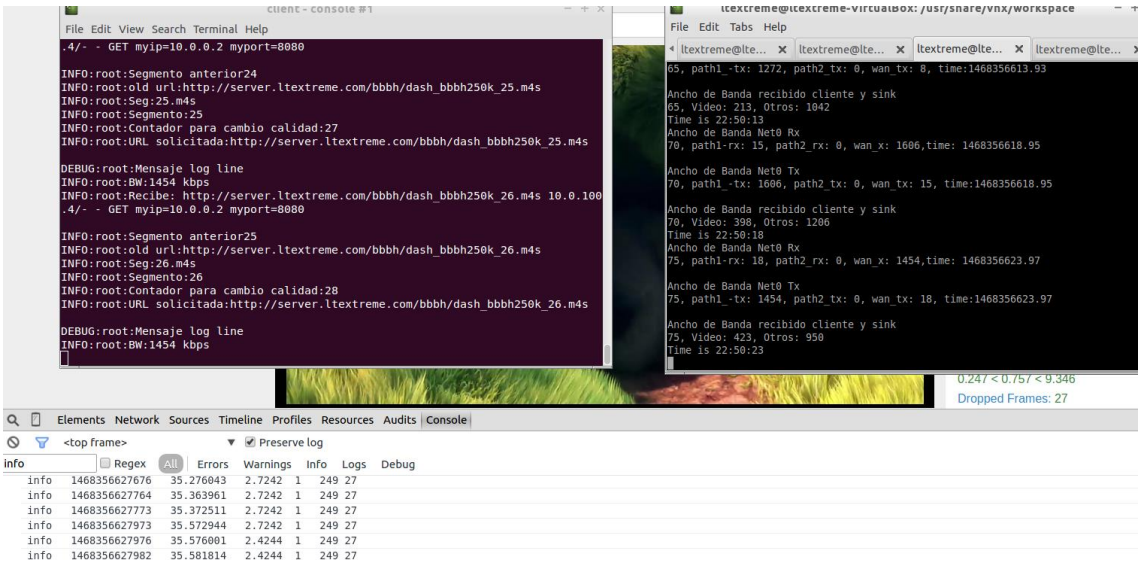


Figura 35. Mensajes de Log durante la reproducción

4.3.2 Calculo de estadísticas

Se utiliza el modulo descrito en la sección 3.3.2 para obtener el valor del ancho de banda medido en la red. Para la realización del escenario en que se utiliza un switch por red, es necesario tomar las medidas del switch de la red del lado del cliente para medir el nivel de congestión que está experimentando. Por lo tanto, es necesario medir el ancho de banda en la conexión del switch con la red WAN. Al estarse realizando streaming de multimedia desde el servidor, es necesario tomar las medidas solo del enlace de bajada.

En la Figura 36 se detalla cómo se toman las estadísticas del switch y cuál es su función. Al tomar la medida de ancho de banda de bajada del puerto 2 (el que está conectado a la red WAN), es posible saber el nivel de congestión de la red, ya que por ese enlace está pasando el tráfico de todos los equipos de la red conectada al switch. Al tener la ventaja que el controlador tiene una imagen de la topología de la red, es posible gracias a este equipo de control, tomar decisiones de acuerdo al nivel de congestión que se está experimentado.

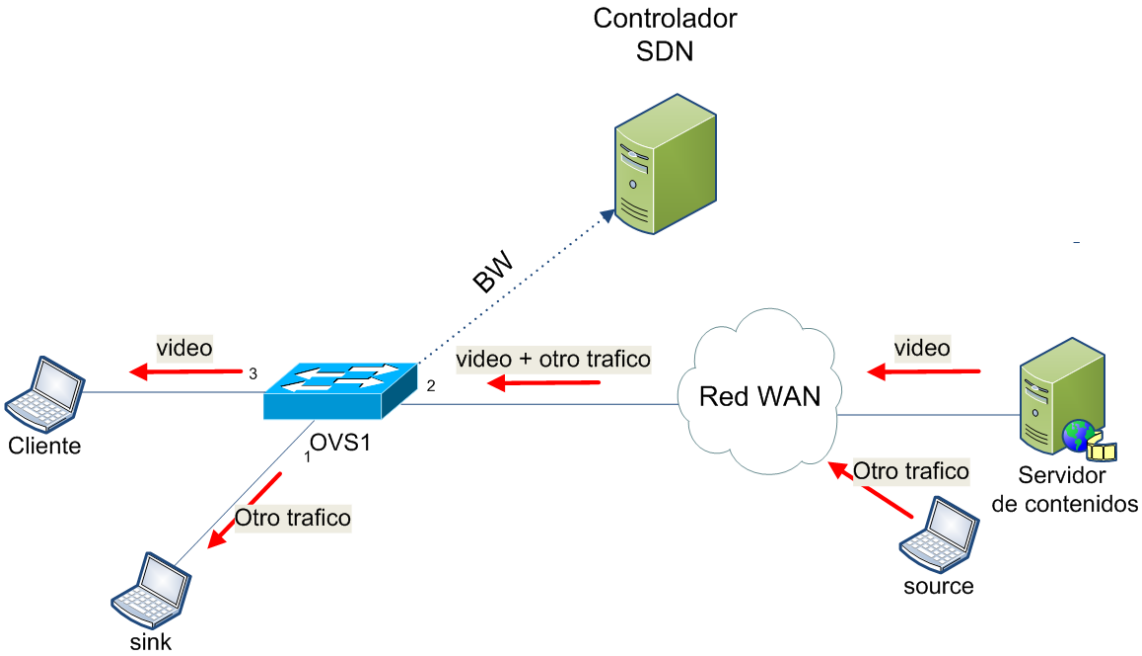


Figura 36. Recolección de estadísticas del switch OpenvSwitch1

Para tener una imagen rápida de las estadísticas, se utiliza la información brindada por la REST API Además se decodifica la información para que sea mostrada en formato JSON. Un ejemplo del comando es el siguiente:

```
$ curl http://10.0.2.2:8080/wm/statistics/bandwidth/00:00:00:00:00:00:00:01/2/json -X GET -d '' | python -m json.tool

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current Dload  Upload   Total   Spent    Left  Speed
```

100 573 0 573 0 0 45679 0 --:--:-- --:--:-- --:--:-- 52090 [{

"bits-per-second-rx": "0",
"bits-per-second-tx": "0"
"dpid": "00:00:00:00:00:00:00:01",
"port": "1"
"updated": "Tue Jun 21 15:35:50 UTC 2016"
},
{
"bits-per-second-rx": "0",
"bits-per-second-tx": "0",
"dpid": "00:00:00:00:00:00:00:01",
"port": "local",
"updated": "Tue Jun 21 15:35:50 UTC 2016" },
{
"bits-per-second-rx": "1888724",
"bits-per-second-tx": "44348",
"dpid": "00:00:00:00:00:00:00:01",
"port": "2",
"updated": "Tue Jun 21 15:35:50 UTC 2016"
},
{
"bits-per-second-rx": "44348",
"bits-per-second-tx": "1888724",
"dpid": "00:00:00:00:00:00:00:01",
"port": "3",
"updated": "Tue Jun 21 15:35:50 UTC 2016"
}
]

Con la información mostrada es posible conocer la medida del ancho de banda de la red. Esta información será actualizada cada 5 segundos. Se ha elegido este valor para tener una mejor precisión y al mismo tiempo que sean valores actualizados. Además que este valor es correcto con correspondencia al intervalo en que deben cambiarse la calidad de los segmentos solicitados, tal como será analizado con el algoritmo de adaptación.

Se configura una clase en Python llamada CollectStats con métodos para obtener las estadísticas con el switch dado. Luego de utilizar el método GET, se obtienen las estadísticas de cada puerto. El valor que se va a retornar es una lista de Python, cuyos elementos son diccionarios como se muestran a continuación:

{u'bits-per-second-rx': u'851', u'bits-per-second-tx': u'851', u'updated': u'Wed Jun 29 15:04:26 UTC 2016', u'port': u'1', u'dpid': u'00:00:00:00:00:00:00:01'}

***{u'bits-per-second-rx': u'1635'**, u'bits-per-second-tx': u'1635', u'updated': u'Wed Jun 29 15:04:26 UTC 2016', u'port': u'2', u'dpid': u'00:00:00:00:00:00:00:01'}*

{u'bits-per-second-rx': u'784', u'bits-per-second-tx': u'784', u'updated': u'Wed Jun 29 15:04:26 UTC 2016', u'port': u'3', u'dpid': u'00:00:00:00:00:00:00:01'}

Ahora solo se necesita obtener los valores del diccionario. El valor subrayado es el ancho de banda medida en la bajada del puerto 2 (conexión a la WAN).

4.4 Algoritmo de adaptación de acuerdo a estadísticas obtenidas usando SDN

En el diseño utilizado en este trabajo, se utiliza un proxy para redirigir las peticiones realizadas por el reproductor DASH ya sea al cache HTTP o al servidor de contenidos. De la misma manera el proxy se encargará de modificar la URL solicitada por el reproductor dependiendo el valor del ancho de banda de la red.

Se debe tomar en cuenta que se pueden recibir archivos que pueden ser el manifiesto MPD, el archivo de inicialización que envía un .mp4 y los segmentos DASH que tienen extensión m4s. Tomando en cuenta esta información, hay que cargar el programa escrito en Python en el archivo squid.conf como se muestra a continuación:

```
acl dash urlpath_regex -i \.mpd$
acl dash urlpath_regex -i \.m4s$
acl dash urlpath_regex -i \.mp4$
cache deny dash
url_rewrite_access allow dash
url_rewrite_program /usr/bin/python /etc/squid3/dash_redirect_bw.py    url_rewrite_children 5
```

Para el diseño del algoritmo, se deben tomar en cuenta las estrategias a considerar para mejorar la QoE tal como fue mostrado en la sección 2.4.2. Para este trabajo se consideró lo siguiente:

1. Frecuencia del cambio de calidad de video: Cambios muy frecuentes pueden resultar molestos para el usuario, por lo que se ha decidido realizar el cambio de calidad pasando 5 segmentos.
2. El buffer debe llenarse lo más pronto posible: Es importante para disminuir el retardo inicial se tenga prioridad de llenar el buffer. Para eso, los primeros 5 segmentos se envían con la calidad más baja sin importar el ancho de banda que está siendo utilizado en la red.
3. Aumento y disminución escalonada de calidad: Usuarios pueden considerar molesto el cambio de una calidad bien alta a una calidad muy baja. Por lo tanto se decide realizar el cambio de segmentos con tasa de bits de 2000 kbps a 500 kbps y de la misma manera al realizar un aumento se realizará de manera escalonada de tasas de bits 250 kbps a 500 kbps.

La adaptación del video debe realizarse solo cuando no hay segmentos en el cache HTTP. En los escenarios propuestos en este trabajo se simulará el handover entre el streaming multicast y unicast. En un fichero llamado "error.txt" se indicaran los segmentos que deben ser descargados por unicast, ese decir, que deben ser solicitados al servidor de contenidos y utilizará el algoritmo de adaptación de DASH propuesto en este trabajo. En el caso que no se encuentre el número de segmento en el archivo, quiere decir que debe solicitar el segmento al cache HTTP, que en este caso está instalado en la misma máquina virtual del cliente.

Para seleccionar la calidad solicitada dependiendo el ancho de banda, se comparará con un valor constante definido como BW_MAX. En el caso que se use menos de la mitad del ancho de banda total de la red, se puede solicitar el segmento de mayor calidad. Si el ancho de banda medido esta entre el 50% y 75% se solicita el siguiente nivel de calidad, que tiene una tasa de bits de 1000 kbps. Si se está usando entre el 75% y el 100%, entonces se solicita una tasa de bits de 500 kbps. Por ultimo al pasarse de este valor límite, se debe solicitar la más baja calidad, una tasa de bits de 250 kbps.

En el programa escrito en Python también se almacenarán en un fichero valores que servirán para realizar la evaluación del escenario. Estos valores son: El número de segmento, el ancho de banda tomado en la consulta al controlador, la representación usada, el número de cambios de calidad y el tiempo en que fue realizada la solicitud. En la Figura 37 se describe el algoritmo utilizado.

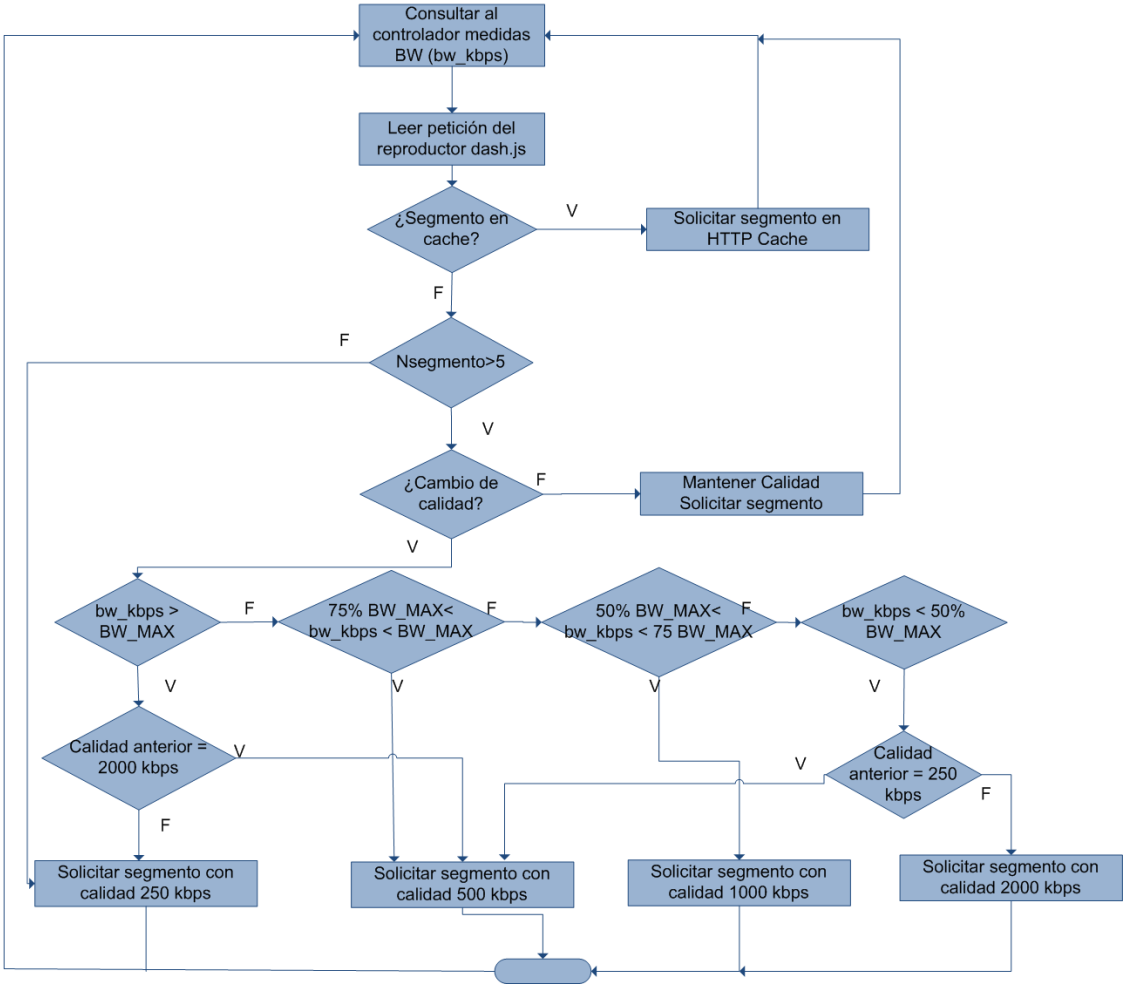


Figura 37. Algoritmo de adaptación de DASH tomando estadísticas de la red

5 Experimentación y análisis de resultados

5.1 Configuración de experimentos

5.1.1 Configuración experimentos usando un solo cliente

Para realizar los experimentos se programaron algunos scripts para realizar la simulación deseada de acuerdo al escenario usado. Es necesario que se considere lo siguiente para la cada prueba:

- Abrir navegador con el proxy activado. Se debe abrir una nueva sesión del navegador para cada proxy.
- Activar monitoreo del tráfico que pasa por las interfaces del switch.
- Asignar archivo error.txt en caso que sea tráfico solo unicast o una simulación de un handover multicast/unicast.
- Congestión utilizando iperf de acuerdo al experimento seleccionado.

Estos experimentos pueden realizarse de las siguientes maneras:

- Sin congestión.
- Con congestión variable.
- Congestión alta.
- Sin adaptación

Al momento de cada experimento se va a extraer información que nos servirá para poder analizar cada caso. Del programa escrito en Python usado para el proxy se obtiene información de cada petición realizada el servidor. Los datos que se obtienen con cada petición HTTP son el número de segmento, el ancho de banda obtenido por las estadísticas del controlador SDN, la representación usada, el número de cambio de representaciones, el tiempo y si la petición es realizada al servidor de contenidos (unicast) o al cache HTTP (multicast).

Del switch usado en la red del cliente, se obtiene el tráfico de transmisión y recepción de cada interfaz medida por el controlador. Con esta información podemos obtener el tráfico que hay en cada interfaz del switch en cada experimento y además tener una comparación del tráfico de video y del tráfico de otros usuarios de la red.

Para obtener información que como ha variado el nivel del buffer en el reproductor, tenemos que revisar los logs de la consola del navegador Chrome, la cual debe ser copiada en un fichero. La información que se tiene es el tiempo en milisegundos de experimento, el tiempo en que se está reproduciendo el video sin contar paradas, y por último se tiene el nivel del buffer.

Por último, al tener estos datos al finalizar cada experimento, se obtiene las siguientes gráficas:

- Calidad de video por ancho de banda recibido de la red.
- Histograma de número de calidades de video solicitadas.
- Ancho de banda transmisión switch.
- Ancho de banda recepción switch.
- Comparación tráfico cliente (video) y de otros usuarios (iperf).
- Nivel del buffer.

Para la realización de los experimentos se seleccionan los siguientes parámetros en la WAN:

- Ancho de banda limitado a 2500 Mbps.
- Retardo de 30 ± 5 ms.
- Perdida de paquetes del 0.3%.

5.1.2 Configuración de experimentos usando varios clientes

Para este escenario, se mantienen algunos datos del experimento anterior como los datos de la WAN. En este escenario se tienen 3 clientes DASH y se eliminan los usuarios que generan tráfico usando iperf.

Los pasos para realizar este experimento son los siguientes

- Abrir navegador con el proxy activado. Se debe abrir una nueva sesión del navegador para cada proxy.
- Activar monitoreo del tráfico que pasa por las interfaces del switch.
- Asignar archivo error.txt para cada cliente como si fuera tráfico unicast (copiar todos los segmentos).
- Comenzar a reproducir cada video con un intervalo de 10 segundos cada uno.

Para abrir una sesión nueva para cada proxy se procede de la siguiente manera:

sudo google-chrome --new-window --user-data-dir="/root/chrome-profile/" --disable-web-security --proxy-server="client.ltextreme.com:8080" http://server.ltextreme.com/CUSUM-dashjs13/

sudo google-chrome --new-window --user-data-dir="/tmp2" --disable-web-security --proxy-server="client2.ltextreme.com:8080" http://server.ltextreme.com/CUSUM-dashjs13/

sudo google-chrome --new-window --user-data-dir="/tmp3" --disable-web-security --proxy-server="client3.ltextreme.com:8080" http://server.ltextreme.com/CUSUM-dashjs13/

En cada reproductor se debe seleccionar como archivo de manifiesto el que se tiene en esta ubicación: http://server.ltextreme.com/bbbh/bbbh.mpd

5.1.3 Configuración de experimentos variando los caminos de red

Para estas pruebas pueden usarse algunas variaciones. Se puede hacer la prueba utilizando solo el streaming de video o se puede también agregar tráfico adicional por el comando iperf.

Se deben tener dos scripts, uno por cada camino de red, donde lo primero que debe hacer es borrar las tablas de flujo existentes. Luego de ejecutar el script se insertarán las tablas de flujos y se debe actualizar el programa dash_redirect_bw.py del proxy del cliente. En las pruebas realizadas se realiza el cambio del primer camino al segundo luego de 300 segundos.

5.2 Resultados

5.2.1 Un solo cliente sin congestión

Para realizar este experimento, se solicitan todos los segmentos al servidor de contenidos y no se envía tráfico adicional entre el source y el sink. De esta manera se puede validar el algoritmo de adaptación con las estadísticas SDN usadas solo con presencia del video. Hay que tener en cuenta que se tiene un valor muy limitado del ancho de banda (2500 kbps), por lo tanto no será posible solicitar siempre segmentos de la más alta calidad.

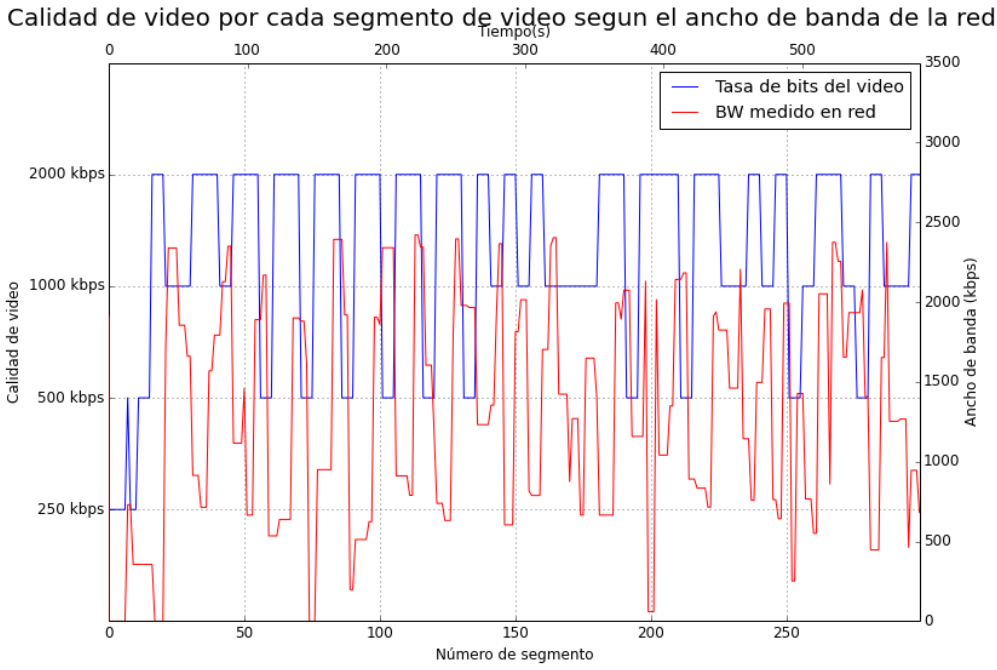


Figura 38. Sin congestión: Calidad de video por segmento de video

En la Figura 38 podemos observar las calidades de video solicitadas y el ancho de banda medido en función del número de segmento solicitado. El ancho de banda en este caso debe ser solo del video que se está recibiendo. De la misma manera se observa en el eje horizontal superior el tiempo en que fue solicitado. En la gráfica se puede observar que la mayoría de las representaciones solicitadas son de alta calidad, con lo que se está aprovechando que no se está compitiendo con otro flujo. Esto lo podemos comprobar en la Figura 39.

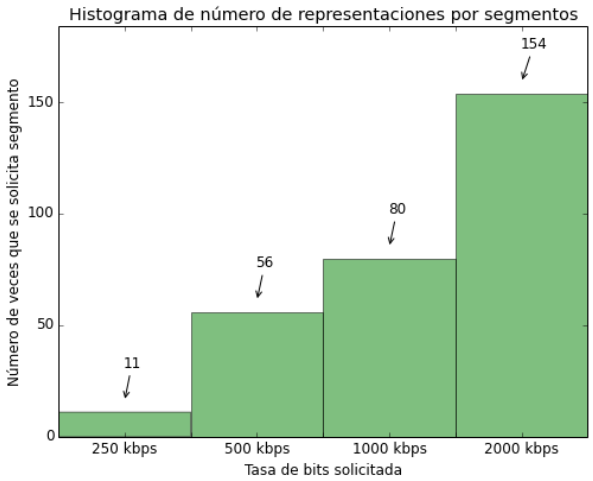


Figura 39. Sin congestión: Distribución de calidades de video solicitados.

El reproductor estará mostrando el video siempre que tenga contenido almacenado en su buffer. Para que se produzca una parada se debe tener una cantidad menor que el tamaño del segmento, que para este trabajo se han escogido segmentos de dos segundos. En la Figura 40 se observa cómo ha evolucionado el nivel del buffer con respecto al tiempo de la reproducción.

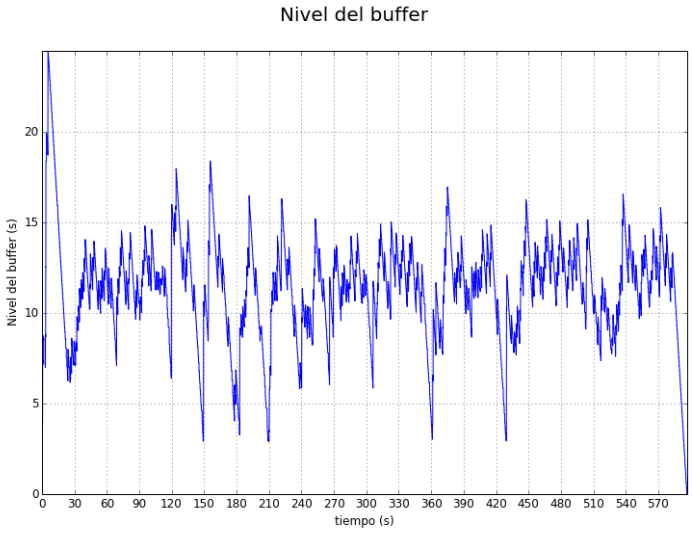


Figura 40. Sin congestión: Nivel del buffer

5.2.2 Un solo cliente con Congestión variable

Para este experimento se comenzará a enviar tráfico entre source y sink de manera variable de utilizando el comando iperf.

En la Figura 41 se observa la evolución de la calidad del video conforme aumenta la reproducción. A diferencia del caso sin congestión, ahora se puede observar como varían los segmentos solicitados. Igual que el caso sin congestión se empieza solicitando segmentos de baja calidad para poder llenar el buffer para después ir variando la tasa de bits recibida. Al existir otros flujos, el ancho de banda disponible estará variando en el transcurso de la reproducción.

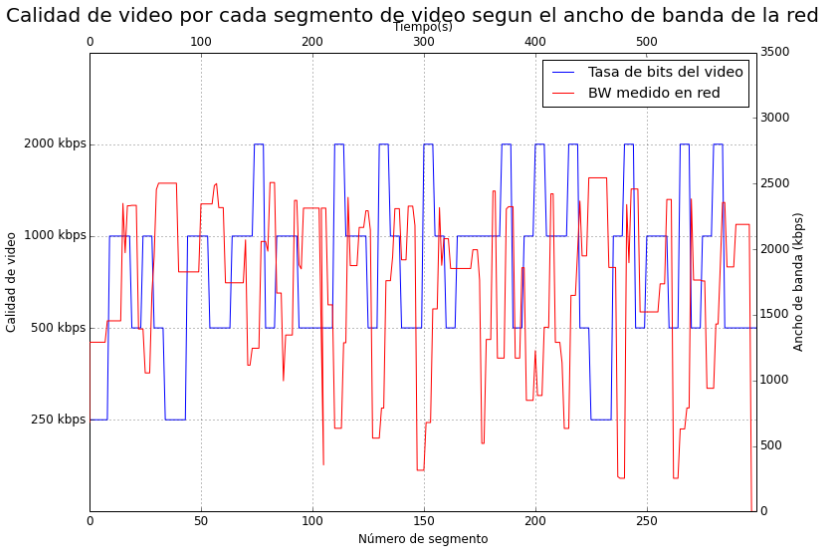


Figura 41. Congestión variable: Calidad de video por segmento de video

En la Figura 42 se comprueba que en este caso se han solicitado más veces las calidades con tasa de bits de 500 kbps y de 1000 kbps. Hay que recordar que de acuerdo al algoritmo de adaptación, cuando debe bajar la calidad de video por la congestión, se debe realizar de manera escalonada, por lo que antes de recibir segmentos de 250 kbps se debe recibir el de 500 kbps. En la Figura 43 se puede observar como es el tráfico del video y del tráfico generado por iperf en función del tiempo. De esta manera se puede observar cómo se comporta el video en presencia de otro tráfico que este congestionando la red.

Es importante que el algoritmo reaccione a los cambios generados en el tráfico de red donde se esté generando el cuello de botella. Es importante en situaciones de

extrema congestión poder variar entre calidades bajas e intermedias para que el usuario pueda percibir una mejor QoE.

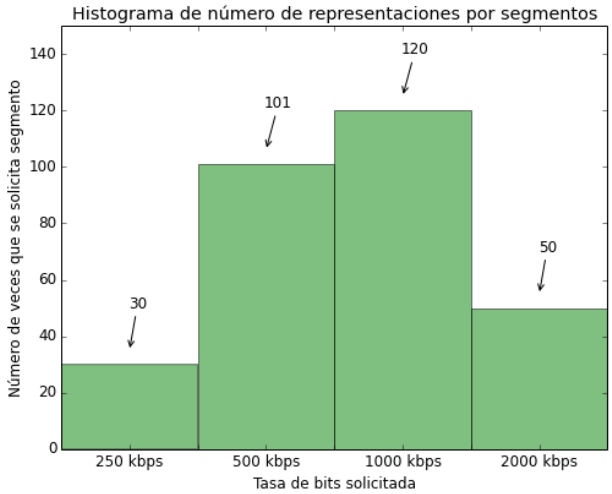


Figura 42. Congestión variable: Distribución de calidades de video solicitadas

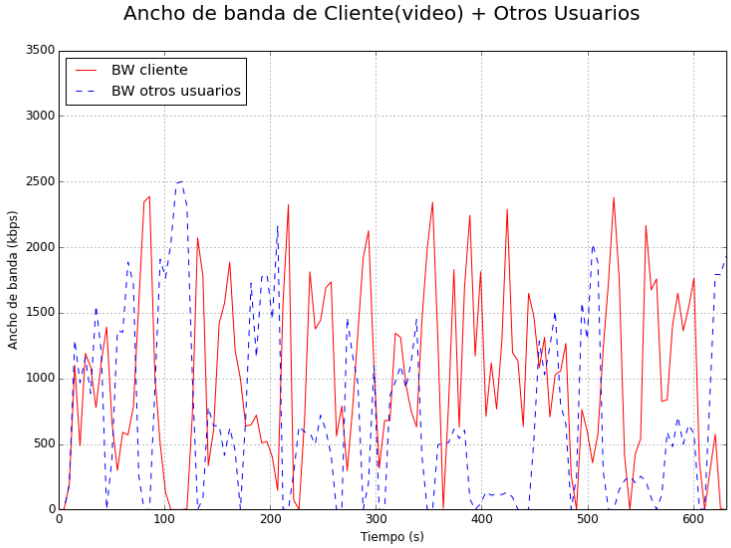


Figura 43. Congestión variable: Trafico video y de otros usuarios

En la Figura 44 se puede observar la evolución del nivel del buffer. A diferencia del caso del experimento sin congestión, se observa que varias veces en la reproducción se tienen valores mínimos. Se tienen 3 paradas de menos de 1 segundo y 6 de más de 1 segundo de duración. En la gráfica pueden visualizarse los intervalos en que queda

vacío el buffer y usando las gráficas del tráfico de video con las de los otros usuarios, es posible discernir el motivo de esas paradas.

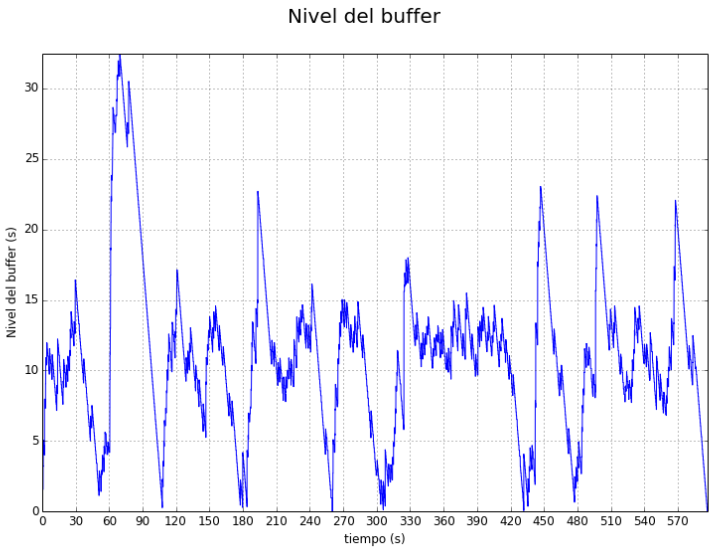


Figura 44. Congestión variable: Nivel del buffer

Como último resultado de este experimento se tiene la variación del ancho de banda medido por el controlador SDN en cada puerto del switch y que se muestra en la Figura 45. En este caso el switch está recibiendo el tráfico combinado que viene de la WAN del video y el tráfico generado por iperf. El tráfico de transmisión del switch se puede observar a la izquierda de la trafica y es el tráfico que se divide al cliente y al receptor del tráfico iperf (sink).

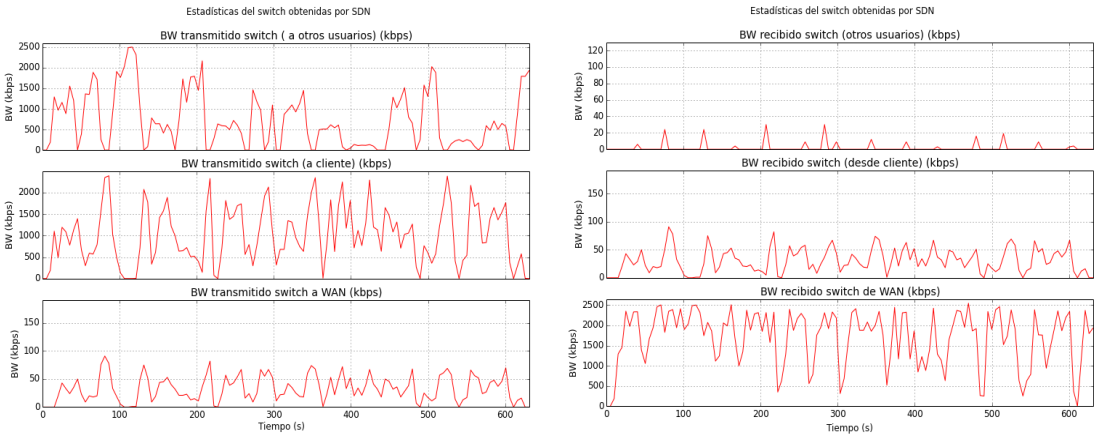


Figura 45. Congestión variable: Estadísticas de cada puerto del switch

5.2.3 Un solo cliente con congestión total

Para este experimento en lugar de tener una congestión variable, se decide usar congestión fija utilizando iperf y enviando 1500 kbytes por la duración de la reproducción.

De acuerdo a la Figura 46 y Figura 47 ahora se tiene la mayoría de representaciones de video se encuentran en las tasas de bits de 250 kbps y especialmente de 500 kbps. De la misma manera no se tiene calidades altas (2000 kbps) en este experimento. Que se tenga mayoría de solicitudes de 500 kbps es una ventaja al no tener que estar solicitando calidades inferiores a pesar de tener congestión alta todo el tiempo. Hay que tener en cuenta que aunque se realiza un iperf constante, de todas maneras el ancho de banda medido por los iperf varía entre 1500 kbps y tiene picos de 2500 kbps como se observa en la Figura 48 donde se combina el tráfico que recibe el cliente y el que reciben otros usuarios en la red.

En la Figura 46 también se observa otro detalle al observar el eje superior del tiempo. En este caso la reproducción llego a los 700 segundos, lo que indica que el video se detuvo varias veces, considerando que tiempo del video es de 600 segundos.

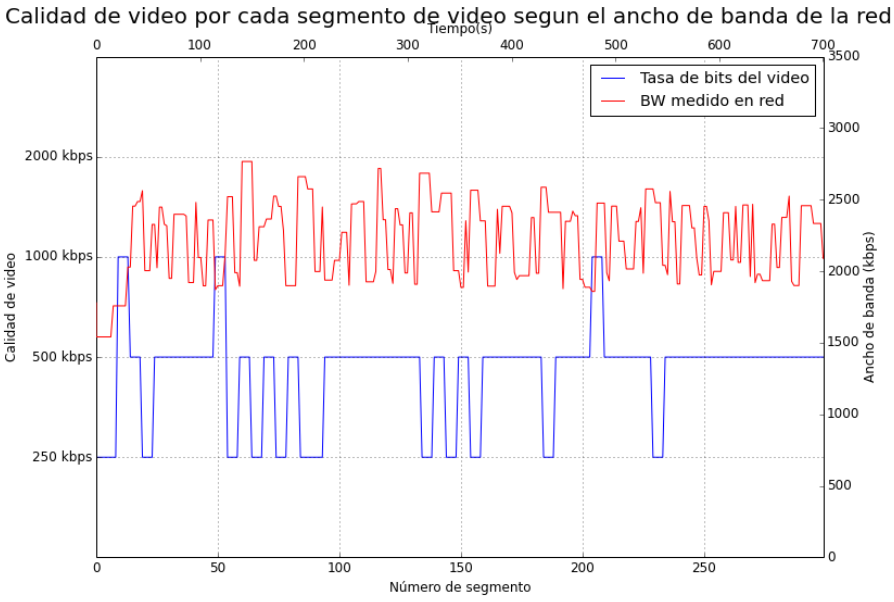


Figura 46. Congestión total: Calidad de video por segmento de video

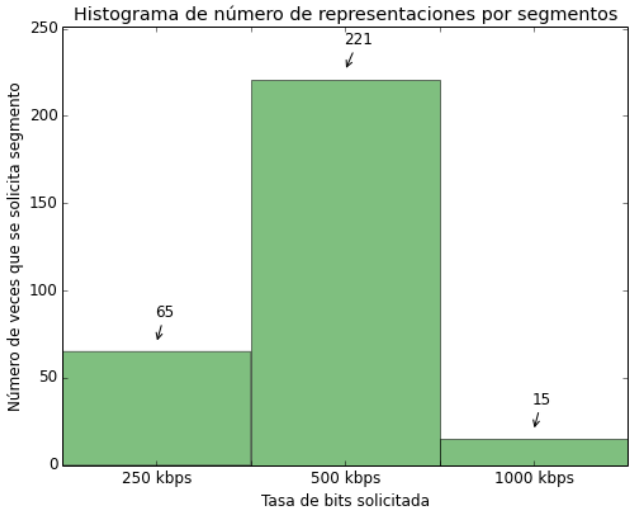


Figura 47. Congestión total: Distribución de calidades de video solicitadas

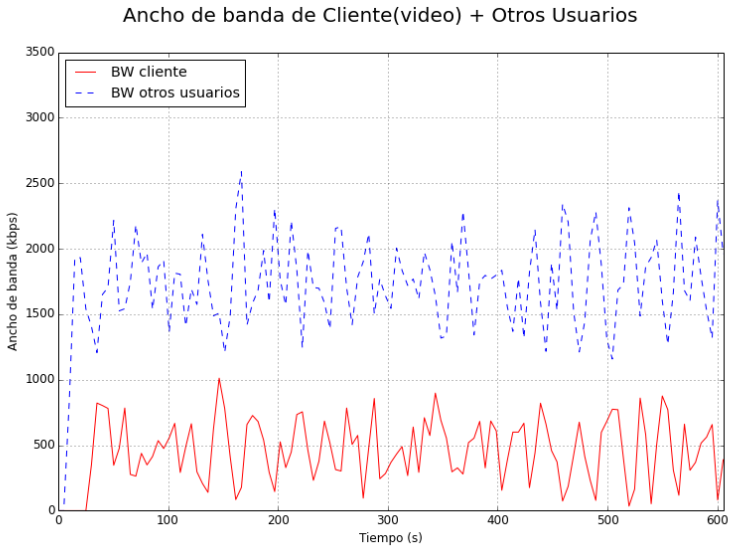


Figura 48. Congestión total: Trafico video y de otros usuarios

Se observa en la Figura 49 que el comportamiento del buffer no es óptimo en estos casos. En los cálculos realizados se tiene que existieron 11 paradas de menos de 1 segundo que podrían despreciarse, pero así mismo se tienen 24 paradas de más de un segundo. Esto ha ocasionado que el tiempo que se demoré la reproducción sea mayor. El tiempo más alto ha sido de 9.68 segundos, siendo un valor que tiene resultados negativos en la calidad que percibe el usuario.

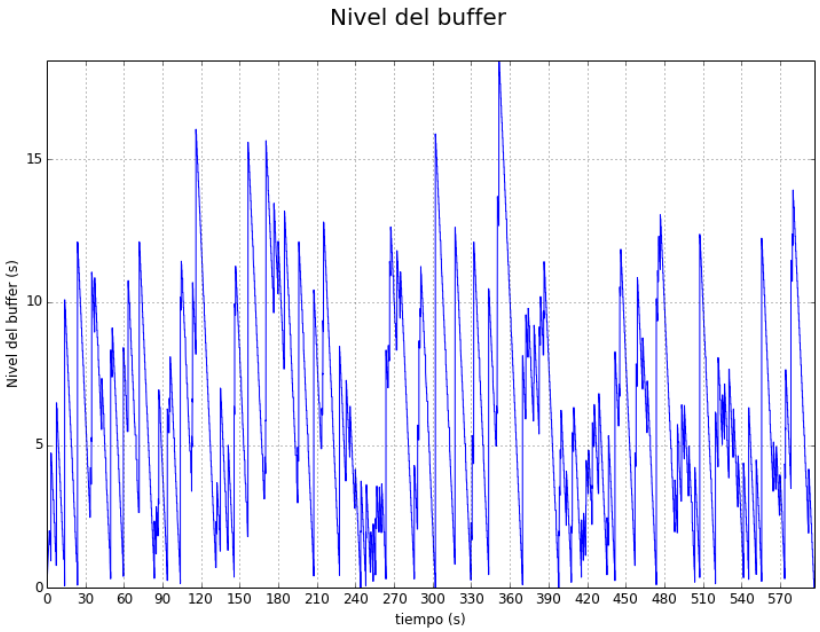


Figura 49. Congestión variable: Nivel del buffer

5.2.4 Simulación Multicast/ Unicast con congestión variable

En este experimento se va a simular el caso del handover entre multicast y unicast. En el caso de multicast se debe seleccionar el contenido del cache HTTP que está en el cliente y en caso que se simule unicast se deberá seleccionar al servidor de contenidos tal como se ha realizado en los anteriores experimentos. Para el caso de multicast, se asume que el contenido ya está en un cache, por lo tanto no va a pasar por alguna red WAN, lo que significa que no se tendrán las consecuencias de las limitantes del ancho de banda y el retardo. Para el caso que se utilice unicast, se podrá usar el algoritmo de adaptación para seleccionar la calidad de video de acuerdo a las condiciones de la red en ese momento.

Para realizar este experimento se debe seleccionar la opción de multicast al momento de ejecutar el script. De esta manera se copiará un fichero a error.txt con el número de segmentos que deben ser solicitados al servidor de contenidos. En el caso que un segmento no aparezca, se lo seleccionará al cache. El 40 % de segmentos deben ser seleccionados en unicast y el resto por multicast.

El trafico unicast, será el mismo usado para unicast con congestión variable, aunque en el caso de que se solicite al cache HTTP este tráfico no tendrá importancia. El tráfico debe ser considerado cuando se cambie a unicast, por lo tanto siempre el controlador debe estar capturando las estadísticas.

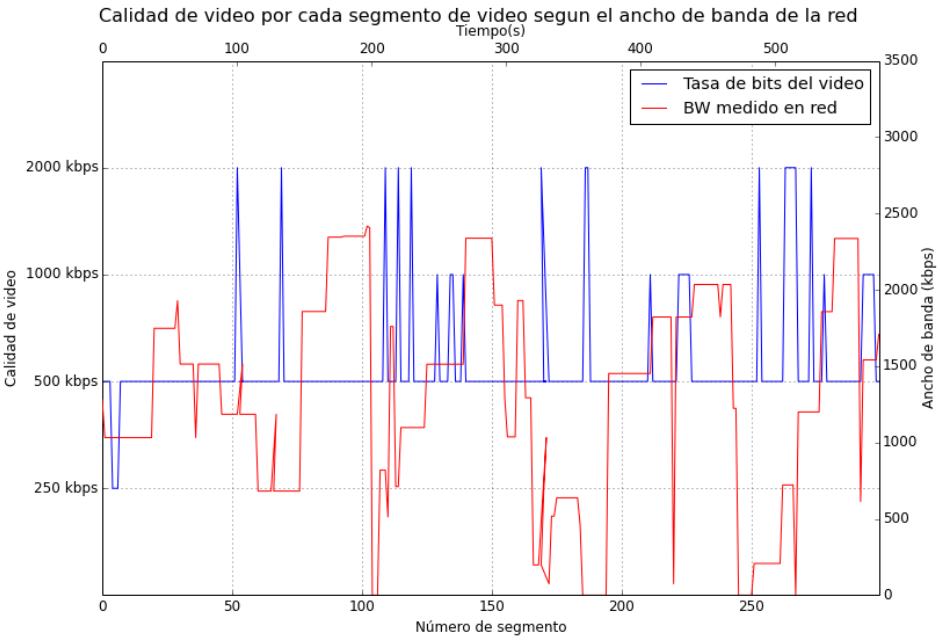


Figura 50. Multicast/ Unicast: Calidad de video por segmento de video

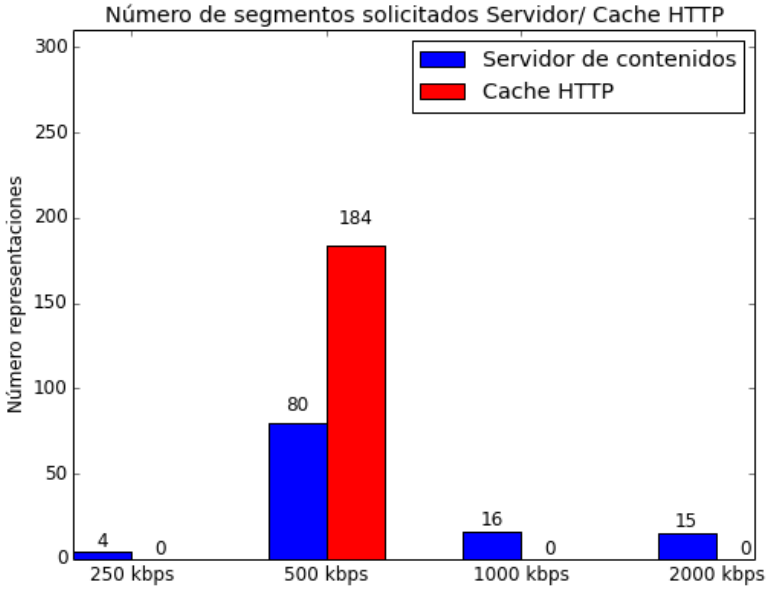


Figura 51. Multicast/ Unicast: Número de segmentos solicitados

En la Figura 50 se observa como varían las representaciones solicitadas por cada segmento dependiendo el ancho de banda medido en el cuello de botella de la red. Debido a que ahora se están solicitando muchos segmentos al cache que se encuentra en el cliente, se está ahorrando mucho ancho de banda lo que ocasiona que en muchas ocasiones el valor medido sea muy bajo, casi siempre solo son el tráfico generado por iperf. Como es de esperar, la calidad de 500 kbps tiene mayoría al ser esta la única calidad que se recibe por multicast y que está almacenada en el cache del cliente.

Cuando se está simulando el trafico unicast, se tiene una mayoría de representaciones solicitadas de 500 kbps. La explicación que se tiene es que el algoritmo de adaptación no realiza cambios de calidad instantáneos, sino que lo hace cada 5 segmentos. En muchos casos cuando puede querer hacer un cambio ya sea a una mayor o menor calidad, ya se ha hecho el cambio a una solicitud al cache. De todas maneras se observa que se han seleccionado 16 segmentos de 1000 kbps y 15 de 2000 kbps. En la Figura 51 se puede comparar los segmentos solicitados para cada caso.

Finalmente, se realiza la gráfica de la variación del nivel del buffer mientras se está reproduciendo el video para el caso cuando se realiza el cambio entre unicast y multicast. Al solicitar muchos segmentos al cache HTTP, se observa una mejora en está grafica al no tener que pasar muchas veces por la WAN. En este caso no se detuvo el video en ninguna ocasión. También se observan niveles altos del buffer en ciertos casos, al estar obteniendo todos los segmentos del servidor ubicado en el mismo cliente. Un cambio futuro que puede considerarse, es agregar la parte de la solicitud de los segmentos por el canal e-MBMS y considerar las pérdidas del canal.

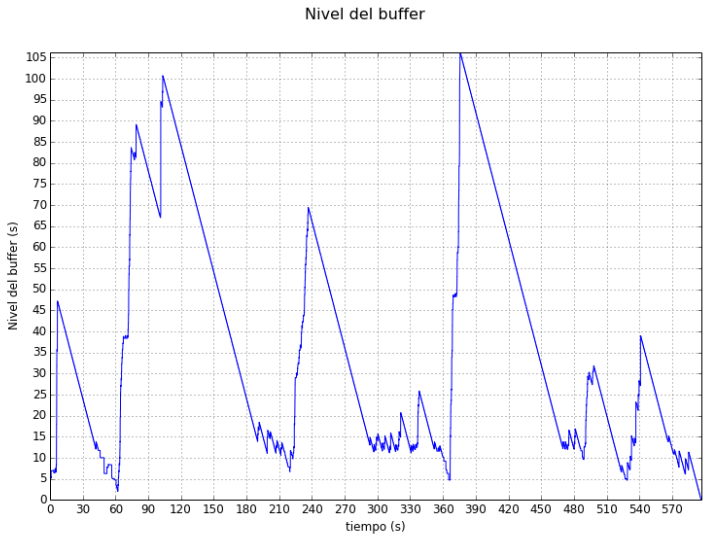


Figura 52. Multicast/ Unicast: Nivel del buffer

5.2.5 Resumen de los experimentos utilizando un cliente DASH

En la Tabla 5 se han resumido los resultados obtenidos en cada uno de los experimentos. Los parámetros a considerar son los siguientes:

- Cambios de calidad de video en la reproducción.
- Tiempo total de la reproducción.
- Retardo Inicial.
- Número de paradas (veces que se detiene el video).
- Parada más larga.

Todos estos parámetros afectan la QoE que experimentan los usuarios y son los que se deben tener en cuenta al momento del desarrollo de un sistema que utilice DASH.

Se observa que cuando hay poca congestión se tienen muchos cambios de calidad lo cual puede ocasionar problemas a los usuarios, aunque se tiene la ventaja que muchos de estos cambios han sido para poder aumentar a una mayor calidad. Por lo tanto a nivel del usuario debe manejar este *trade-off*, cosa que puede soportar al disfrutar de una mejor calidad. Cuando se tiene mucha congestión, como se observa en las figuras anteriores, no es posible tener la mejor calidad de video, y al tener casi siempre un ancho de banda medido fijo, se tiene una menor cantidad de cambios.

En lo que se refiere al tiempo de reproducción del video, contando el tiempo de paradas, se tiene que el mayor tiempo fue cuando no se usó adaptación, enviando siempre la más alta calidad. También a considerar que usando congestión variable se demoró casi 100 segundos menos que teniendo congestión fija en las pruebas realizadas. Ambos casos han podido comprobar cómo funciona el algoritmo de adaptación implementado.

En el caso de las paradas que se han tenido en el transcurso del video, se observa como han aumentado en cada escenario de pruebas, siendo el más alto cuando no se usa adaptación. De la misma manera en la tabla se expresan cuál ha sido la parada más larga en cada experimento. Paradas en la reproducción más largas tienen un impacto aún más negativo en la QoE. En la tabla se han especificado las paradas menores a 1 segundo y las mayores a 1 segundo para separar las que tienen mayor afectación para el usuario.

También se realizaron mediciones del retardo inicial para cada experimento. En este caso depende que tan rápido se descarguen segmentos del buffer, aunque no se puede tener resultados contundentes hasta tener más pruebas en que la red ya se encuentre congestionada al momento de empezar la reproducción.

Tabla 5. Resultados de los experimentos con un cliente DASH

Nombre	*Sin congestión*	*Congestión variable*	*Congestión fija*	*Sin Adaptación*	*Multicast/ Unicast*
Cambios de calidad	41	41	26	n/a	37
Tiempo reproducción	597 s	612 s	714 s	739 s	577 s
Retardo inicial	1.02 s	1.06 s	1.2 s	0.93 s	0.82 s
Paradas menores 1 s	0	3	11	25	0
Paradas mayores 1 s	0	6	24	33	0
Parada más larga	n/a	3.55 s	9.68 s	14.6 s	n/a

5.2.6 Resultado pruebas usando varios clientes DASH

Para este este experimento se utilizó la configuración del escenario especificado en 4.2.3 y se realizan los pasos especificados en 5.1.2 para obtener los datos necesarios que nos permitan obtener las gráficas y poder sacar conclusiones de la implementación realizada y al mismo tiempo poder estudiar cómo funciona la entrega de contenidos usando DASH.

En la Figura 53 se observa una comparación de los tres clientes DASH, en la que se muestra como varía la tasa de bits de cada segmento solicitado de acuerdo al ancho de banda recibido. En cada caso puede observarse que el mayor porcentaje de tasa de bits solicitada es de 500 kbps, lo que puede comprobarse en la Figura 54 donde se muestra la distribución de los tres clientes DASH. Se debe considerar que el eje de tiempo es diferente en cada figura porque la reproducción de cada video se hizo de manera secuencial. El primer cliente empieza en 25 segundos, el cliente 2 en 45 segundos y el cliente 3 en 75 segundos. Se lo realizó de esta manera para poder comprobar el efecto que se tenía en la equidad de los recursos de red cuando un cliente DASH se une antes al sistema que otro usuario.

Como ventaja al usar el algoritmo implementado es que no se tiene muchos cambios a la menor calidad de video (250 kbps) y se logra aumentar a la mayor calidad cuando otro usuario está descargando una calidad inferior. Tal como se puede observar, todos los clientes DASH están compitiendo por los recursos. Hay que tener en cuenta que se realizó este experimento con limitante de ancho de banda de 2500 kbps en la WAN, lo cual no permite que dos usuarios puedan tener al mismo tiempo la mayor tasa de bits (2000 kbps) en la reproducción.

Se observa también en la gráfica que el cliente 2 se demoró más en solicitar el último segmento del video (633.14 s) comparado al cliente 1 (619.18 s) y al cliente 3 (612.95 s). Cada cliente DASH realizó un total de 26 cambios de calidad, lo que confirma aún más el grado de competitividad que se tuvo para los tres clientes.

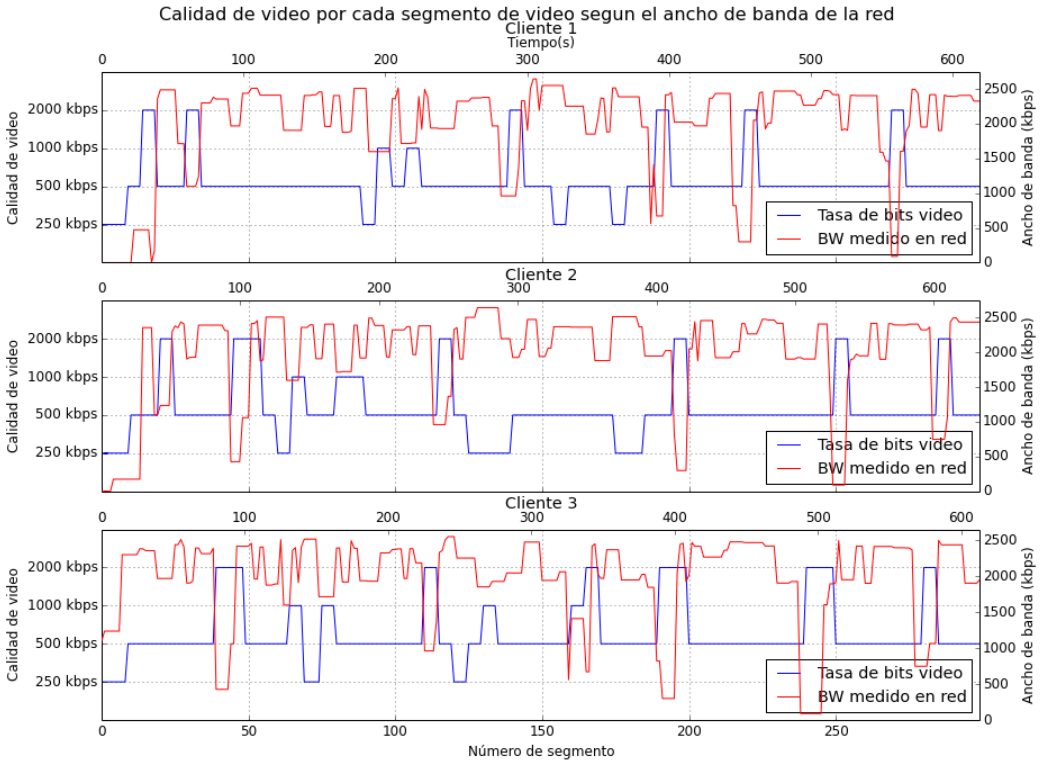


Figura 53. Calidad de video de acuerdo al ancho de banda recibido para tres clientes DASH

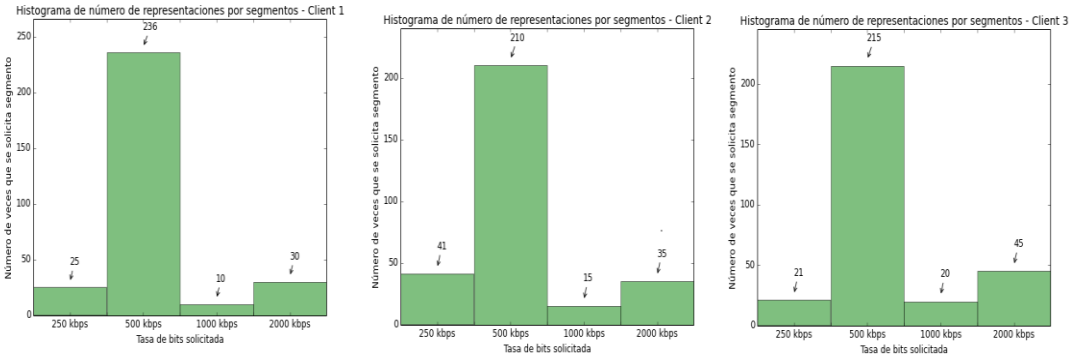


Figura 54. Número de segmentos recibidos para los tres clientes DASH

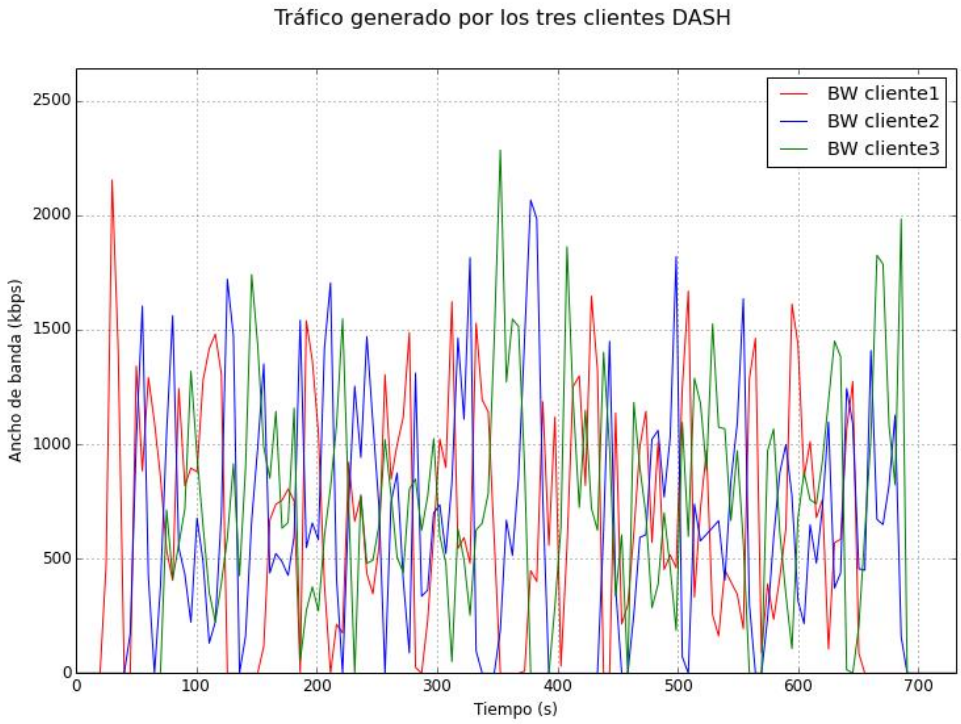


Figura 55. Trafico usado por los tres clientes DASH

En la Figura 55 se observa el tráfico de cada uno de los tres clientes DASH del escenario, con el eje de tiempo empezando desde que se empezó a correr el script que inicia la simulación. Se puede observar el tiempo en que empieza cada reproducción. Los valores de tiempo en que empieza cada reproducción son obtenidos exactamente con los scripts realizados para ejecutar los experimentos. Se puede observar como disminuye el tráfico del cliente 1 cuando llega el cliente 2, el cual inicialmente tiene una mayor tasa de bits que el cliente 1. Esto ocurre hasta que llega al sistema el cliente 3, el cual de la misma manera inicialmente está recibiendo una mejor calidad. Al no entrar más usuarios al sistema, los tres clientes comienzan a competir por tener una mejor calidad de video, y se puede observar como el aumento de tráfico para cada cliente se realiza de manera alternada entre los tres clientes.

Se observa la diferencia de los tiempos en que terminan de solicitar los segmentos al final de la gráfica. Como era de esperarse el cliente 1 termina de solicitar los segmentos antes que el resto.

Por último, se tiene la gráfica del nivel del buffer para cada cliente DASH en el experimento realizado. Como era de esperarse se tienen varias paradas en la reproducción aunque la mayoría son de menos de 1 segundo.

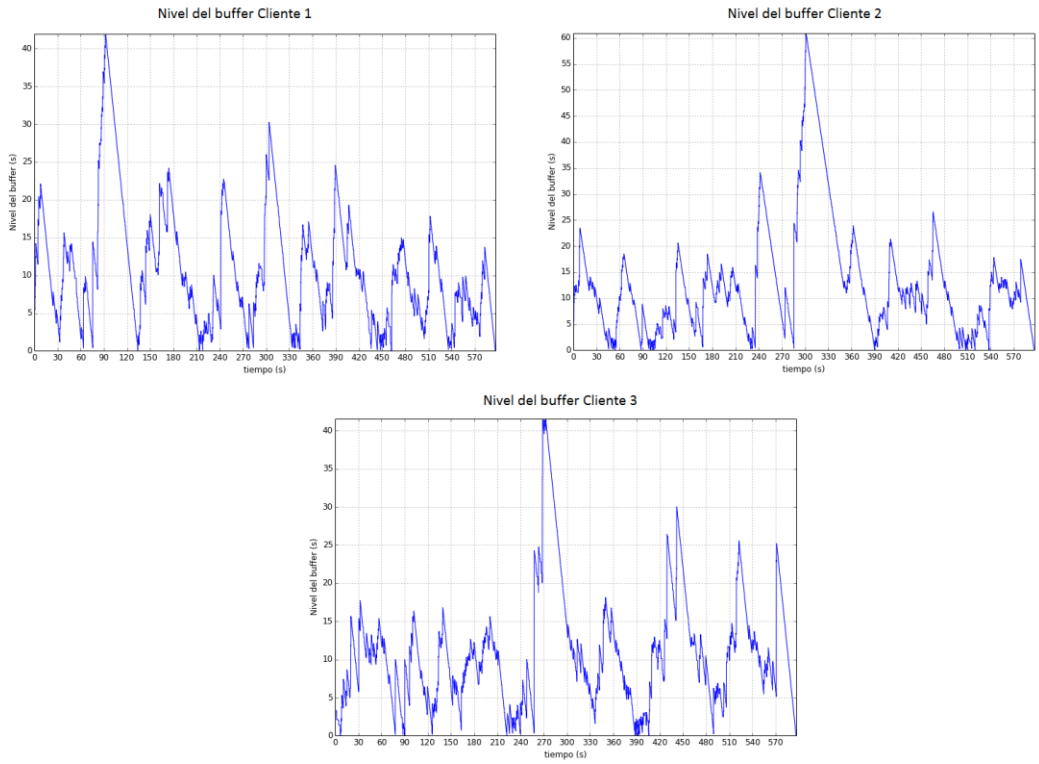


Figura 56. Nivel del buffer para los tres clientes DASH

Por poner ejemplo de un cliente de streaming, se analiza el nivel del buffer del cliente 3 en la Figura 56. Se observa que en 7 intervalos de tiempo existieron problemas de paradas por haberse agotado el buffer. Si se contabilizan el número exacto de paradas se tiene que hay 10 paradas de menos de 1 segundo y 8 paradas de más de 1 segundo, focalizadas en los intervalos de tiempo donde se observa que el buffer ha caído a niveles muy bajos, es decir menos de la longitud de los segmentos.

En la Tabla 6 se muestra un resumen de las métricas obtenidas al realizar el experimento con tres clientes DASH. Se muestra el tiempo en que empieza la reproducción, los cambios de calidad, el tiempo total de la reproducción, el retardo inicial, las paradas menores a 1 segundo y las mayores a 1 segundo, la longitud de la parada más larga y el instante de tiempo en que ocurrió. Se puede observar que las peores métricas las tiene el cliente 2, donde no solo es el último video en terminar la reproducción (a pesar de empezar segundo), también tiene el mayor tiempo en el retardo inicial, además del mayor número de paradas. Su parada más larga fue cuando tuvo su primera caída en el buffer y fue de 8.01 segundos. El cliente 3 en cambio tiene la parada más larga, siendo de 12.2 segundos y ocurre en su quinta bajada de nivel del

buffer. La parada más larga para el cliente 1 fue casi al final de la reproducción, cuando experimentaba su novena bajada de nivel.

Tabla 6. Métricas del experimento usando tres clientes DASH

Nombre	*Cliente 1*	*Cliente 2*	*Cliente 3*
Tiempo PLAY	*25.11 s*	*45.18 s*	*75.37 s*
Cambios de calidad	*22*	*22*	*22*
Tiempo reproducción	*635.79*	*654.47 s*	*638.1 s*
Retardo inicial	*0.59*	*3.06 s*	*1.74 s*
Paradas menores a 1 segundo	*11*	*7*	*10*
Paradas mayores a 1 segundo	*6*	*17*	*8*
Parada más larga	*8.2 s*	*8.01 s*	*12.2 s*
Instante de tiempo parada más larga	*535.82 s*	*45.73 s*	*225.72 s*

5.2.7 Pruebas usando múltiples caminos de red

Se realizan estas pruebas enviando tráfico con iperf al mismo tiempo que se realiza la reproducción. Se desea evaluar si el cambio de camino de red tiene mejores resultados en la distribución del contenido multimedia.

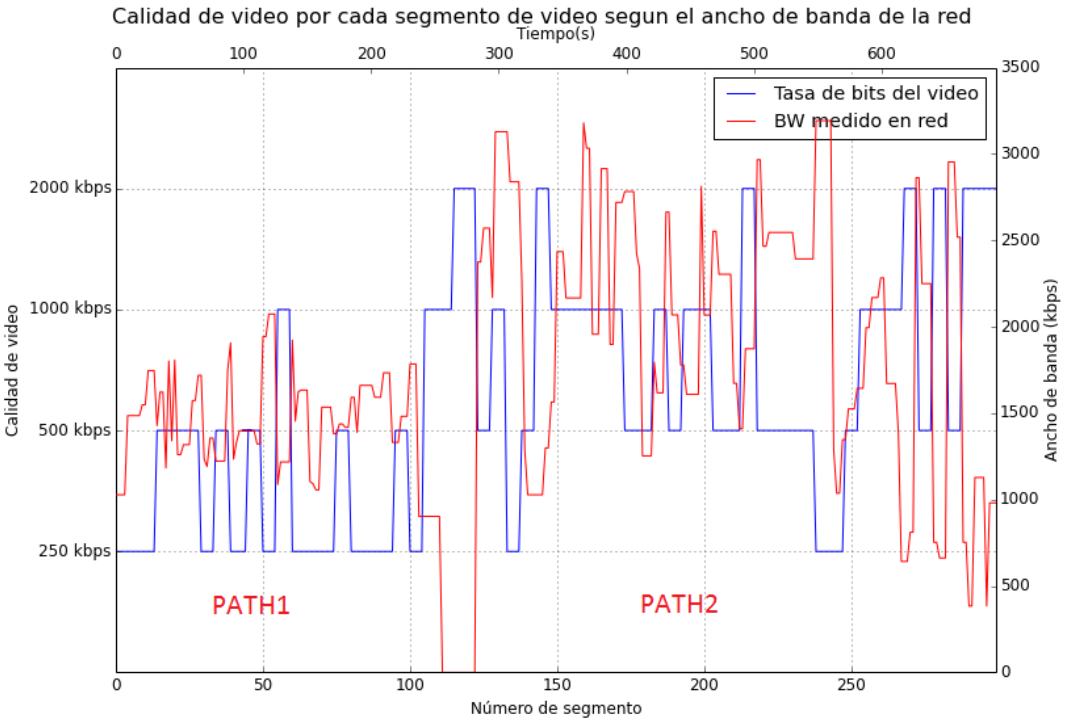


Figura 57. Calidad de video solicitada por dos diferentes caminos de red

En la Figura 57 se muestran los resultados luego de terminar la reproducción. Se muestra la tasa de bits solicitada para cada segmento y el ancho de banda medido en la red. Se especifica en la gráfica que antes de los 300 segundos corresponde al contenido multimedia que pasa por el primer camino de red (path1) y luego de los 300 segundos cambio al camino 2 (path2). El path1 tiene la limitante de ser de solo 1500 kbps, por lo en conjunto al tráfico generado por otros usuarios, no será posible entregar la mejor calidad. Por el contrario, al pasar a un camino de red que tiene el doble de ancho de banda (3000 kbps), será posible poder solicitar tasas de bits mayores. En la Figura 58 se observa el tráfico de video y el generado por iperf usado en las pruebas.

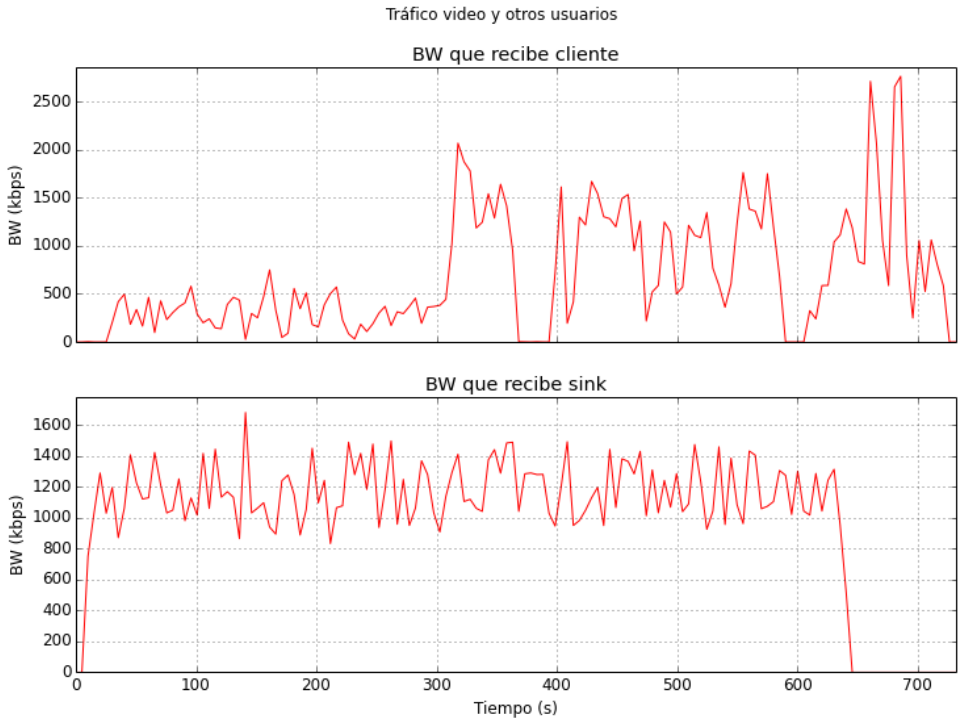


Figura 58. Trafico de video y de otros usuario recibido

En la Figura 59 se observa el ancho de banda medido en el switch Net0. Es posible observar en las gráficas de la derecha cuando se realiza el cambio entre el path1 y el path2. Al mismo tiempo se observa que se trasmite un mayor ancho de banda al cambiar de camino de red.

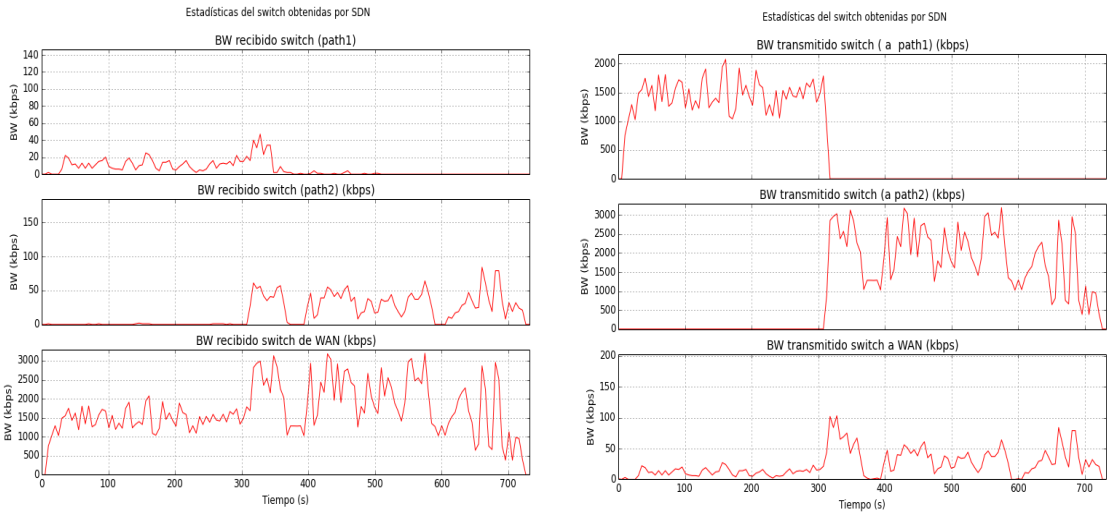


Figura 59. Estadísticas de red en switch Net0

Para culminar la revisión de resultados de esta prueba, se valida la gráfica del nivel del buffer. Se puede observar claramente cómo crece el nivel del buffer al cambiarse a un camino de red que tiene un ancho de banda mayor y por lo tanto permite un throughput mayor para la reproducción del video.

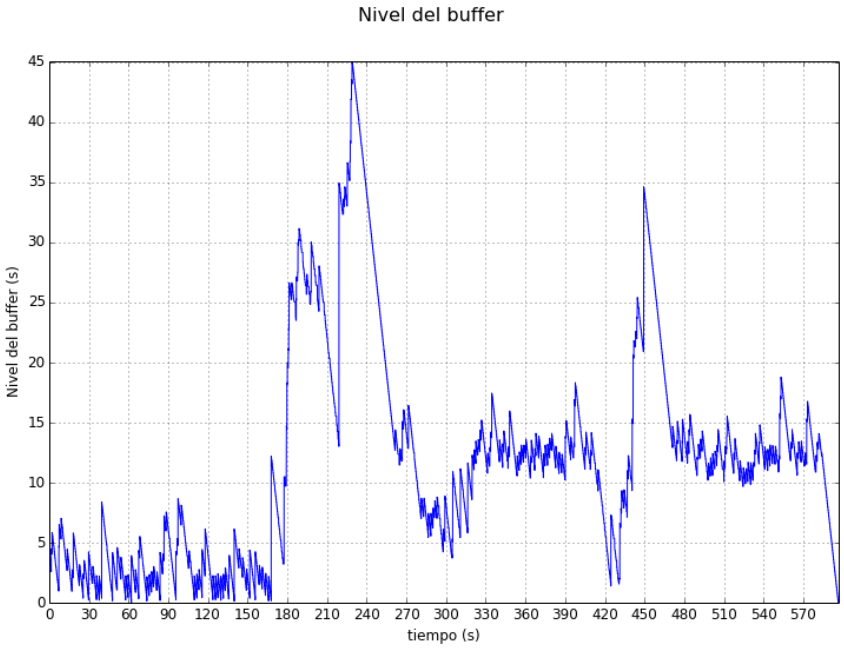


Figura 60. Nivel del buffer al usar escenario de varios caminos de red

6 Conclusiones

Se ha analizado en esta memoria los principales protocolos de streaming, ya sea utilizando UDP o basados en HTTP. Se han descrito las ventajas al usar streaming basado en HTTP al no tener limitantes al pasar por firewalls y también porque se tiene la capacidad de realizar prefetching, es decir puede intentar descargar el video a una tasa más alta que la tasa que está usando y almacenar las tramas en un buffer TCP. Se puede ir un paso más y utilizar un algoritmo de adaptación, utilizando un protocolo como DASH que tiene ventajas como transmisión confiable sobre TCP, reúso de servidores existentes HTTP e infraestructura de cache y compatibilidad con NATs y firewalls.

Utilizando un archivo de manifiesto MPD es posible especificar todas las representaciones que el cliente puede solicitar al servidor de contenidos, teniendo cada representación los mismos segmentos pero con diferente tasa de bits, siendo la tasa de bits más alta la que represente la mayor calidad.

Así como se analizó el uso de protocolos basados en HTTP, los cuales utilizan unicast, también se analiza el uso de multicast. Se describe un escenario que puede ser realizado como experimentación y pruebas para validar el handover entre el streaming multicast al streaming unicast.

Se analizó también los parámetros a considerar para medir la calidad percibida por los usuarios. Se determinó que los parámetros que afectan la QoE de los usuarios son las paradas en la reproducción, cambios bruscos en la tasa de bits recibida, el retardo de inicio y cuando hay muchos cambios de calidad mientras dura la reproducción. Aspectos externos que pueden influir en la calidad son la interacción con otras entidades de red y la equidad (fairness) que se debe considerar cuando varios clientes de streaming están compitiendo por los recursos de red.

Se realizó un análisis de las características fundamentales de las redes definidas por software, donde se llega a la conclusión que su uso puede incrementar la innovación en las redes y servicios, gracias a su flexibilidad y al desarrollo de aplicaciones que pueden controlar los equipos de red como switches y routers de manera centralizada. Se utilizaron los módulos de forwarding para manejar los flujos reactivos y el módulo *static flow pusher* para los flujos proactivos. Otra aplicación importante en este trabajo fue la de recolección de estadísticas. Gracias a esta información es posible brindar la información real del tráfico en el cuello de botella de la red al cliente DASH, sin que este tenga que realizar estimaciones del throughput al servidor.

Para la implementación se realizaron escenarios en VNX utilizando un solo cliente DASH con otros usuarios inyectando tráfico simulando la congestión de la red. También se realizó un escenario donde se tienen tres clientes DASH que están compitiendo entre sí para utilizar los recursos de red disponibles. Por último se realizan pruebas con un escenario que permite usar las características de las SDNs para modificar los caminos de red por donde pasa el streaming multimedia.

En este trabajo se utiliza un proxy que redirige las peticiones de los segmentos realizadas por el reproductor. El proxy fue diseñado para elegir si los segmentos deben ser solicitados por unicast o por multicast gracias a un fichero de texto que lo especifica. Este fichero nos ayudó a emular el comportamiento del cliente en una transmisión multicast. La redirección realizada por el proxy, involucra realizar la adaptación de calidad, utilizando las recomendaciones dadas y analizadas en este documento para tener una mejor QoE.

Gracias a las pruebas realizadas se obtuvieron gráficos del comportamiento y rendimiento dependiendo del nivel de congestión y se evaluaron las métricas de QoE necesarias para saber cuándo el usuario puede estar percibiendo una mejor calidad. Se demostró que el algoritmo propuesto logra mejorar el desempeño comparándolo a las pruebas realizadas sin el algoritmo de adaptación. En estas pruebas se tenía un valor de ancho de banda en la red WAN bien limitado, lo que no permitía al cliente siempre recibir la más alta calidad. Aunque muchas veces se tienen varios cambios de calidad, se ha procurado que los cambios sean escalonados, que nunca se pase de la más alta a la más baja calidad y viceversa, además de establecer un rango de segmentos solicitados para realizar el cambio de calidad de ser necesario.

Luego de culminar este trabajo, he podido comprobar cómo se realiza la distribución de contenido desde cada elemento. Empezando por el cliente que funciona como elemento principal al ser el encargado de solicitar los segmentos de video, continuando con el servidor que debe tener almacenado el video con una cierta codificación. He podido entender cómo funcionan las SDNs y los beneficios que pueden aportar para el desarrollo de nuevos servicios. En este caso fue de ayuda no solo para crear las tablas de flujo en los switches, sino que también lo fue para la implementación del cliente de streaming gracias a que el controlador tiene una imagen completa de la red y el cliente puede comunicarse con el controlador por medio de llamadas REST. Gracias a las funcionalidades de cambios proactivos en la red realizados por el controlador, he podido comprobar la flexibilidad de las SDN y su increíble potencial en el futuro.

Como trabajos futuros, los escenarios diseñados en el presente trabajo de fin de master pueden usarse para entender cómo funciona el streaming multimedia además

de entender cómo funcionan las SDNs. Gracias a los scripts realizados, es posible obtener datos y graficarlos automáticamente y poder analizar los resultados. Es decir, es posible cambiar los parámetros del algoritmo de adaptación y evaluar el impacto que tienen en las métricas de QoE al visualizar los resultados.

Cambios en la red WAN pueden realizarse para evaluar la distribución de contenido multimedia para una red de acceso en particular. Para eso se tendrían que cambiar los parámetros de ancho de banda o retardo o incluso hacer que estos parámetros varíen con el tiempo.

Utilizando SDNs es posible realizar aplicaciones en que se puedan optimizar los caminos que se siguen de acuerdo al tipo de contenido enviado. De la misma manera conociendo el ancho de banda utilizado en la red es posible cambiar a un camino más óptimo y de esa manera optimizar el uso de la red.

Otra línea futura de investigación involucraría cambiar la codificación de los segmentos de video en el servidor de contenidos. Se podría cambiar de la codificación AVC usada en este trabajo, a SVC o HEVC y evaluar los cambios que se tienen al realizar el streaming.

Bibliografía

[1] Cisco, "Cisco Visual Networking Index: Forecast and Methodology, 2014–2019," pp. 2014–2019, 2016.

[2] A. Rao, A. Legout, Y. Lim, D. Towsley, C. Barakat, and W. Dabbous, "Network characteristics of video streaming traffic," *Proc. Seventh Conf. Emerg. Netw. Exp. Technol.*, pp. 1–12, 2011.

[3] Kurose and Ross, *Computer Networking - A Top-Down Approach*. 2013.

[4] "DASH Industry Forum." [Online]. Available: http://dashif.org/faq/#3.

[5] T. Stockhammer, "Dynamic Adaptive Streaming over HTTP: Standards and Design Principles," *Proc. Second Annu. ACM Conf. Multimed. Syst.*, vol. 2014, no. i, pp. 133–144, 2011.

[6] I. Sodagar and A. Vetro, "Industry and Standards The MPEG-DASH Standard for Multimedia Streaming Over the Internet," *IEEE Multimed.*, vol. 18, no. 4, pp. 62–67, 2011.

[7] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the Scalable Video Coding Extension of the H . 264 / AVC Standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 9, pp. 1103–1120, 2007.

[8] G. J. Sullivan, J. R. Ohm, W. J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649–1668, 2012.

[9] V. K. Adhikari, Y. Guo, F. Hao, M. Varvello, V. Hilt, M. Steiner, and Z. Zhang, "Unreeling Net fl ix : Understanding and Improving Multi-CDN Movie Delivery," *Ieee Infocom*, pp. 1620–1628, 2012.

[10] H. Y. H. Yin, X. L. X. Liu, G. M. G. Min, and C. L. C. Lin, "Content delivery networks: a bridge between emerging applications and future IP networks," *IEEE Netw.*, vol. 24, no. August, pp. 52–56, 2010.

[11] J. Liu, B. Li, and Y. Q. Zhang, "Adaptive video multicast over the internet," *IEEE Multimed.*, vol. 10, no. 1, pp. 22–33, 2003.

[12] D. Lecompte and F. Gabin, "Evolved multimedia broadcast/multicast service (eMBMS) in LTE-advanced: Overview and Rel-11 enhancements," *IEEE Commun. Mag.*, vol. 50, no. 11, pp. 68–74, 2012.

[13] N. Paila T, "FLUTE - File Delivery over Unidirectional Transport - RFC 6726," 2012. [Online]. Available: https://tools.ietf.org/html/rfc6726.

[14] L. Bellido, C. M. Lentisco, M. Aguayo, and E. Pastor, "Supporting Handover between LTE Video Broadcasting and Unicast Streaming," *2015 9th Int. Conf. Next Gener. Mob. Appl. Serv. Technol.*, pp. 329–334, 2015.

[15] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hossfeld, and P. Tran-gia, "A

Survey on Quality of Experience of HTTP Adaptive Streaming," *Ieee Commun. Surv. Tutorials*, vol. 17, no. 1, pp. 469–492, 2015.

[16] T. Hoßfeld, M. Seufert, M. Hirth, T. Zinner, P. Tran-Gia, and R. Schatz, "Quantification of YouTube QoE via crowdsourcing," *Proc. - 2011 IEEE Int. Multimedia, ISM 2011*, pp. 494–499, 2011.

[17] P. Ni, R. Eg, A. Eichhorn, C. Griwodz, and P. Halvorsen, "Flicker effects in adaptive video streaming to handheld devices," *Proc. 19th ACM Int. Conf. Multimed. - MM '11*, p. 463, 2011.

[18] T. Kupka, P. Halvorsen, and C. Griwodz, "An evaluation of live adaptive HTTP segment streaming request strategies," *Proc. - Conf. Local Comput. Networks, LCN*, pp. 604–612, 2011.

[19] C. Müller, S. Lederer, and C. Timmerer, "An Evaluation of Dynamic Adaptive Streaming over HTTP in Vehicular Environments," *Proc. 4th Work. Mob. Video. ACM, 2012.*, pp. 37–42, 2012.

[20] A. Mansy, M. Fayed, and M. Ammar, "Network-layer fairness for adaptive video streams," *Proc. 2015 14th IFIP Netw. Conf. IFIP Netw. 2015*, 2015.

[21] J. Gettys, "Bufferbloat: Dark buffers in the Internet," *IEEE Internet Comput.*, vol. 15, no. 3, 2011.

[22] F. Ieee, C. E. Rothenberg, M. Ieee, S. Azodolmolky, S. M. Ieee, S. Uhlig, and M. Ieee, "Software-Defined Networking : A Comprehensive Survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14 – 76, 2015.

[23] "Open Network Install Enviroment (ONIE)." [Online]. Available: http://onie.org/.

[24] Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks [white paper]," *ONF White Pap.*, pp. 1–12, 2012.

[25] "ONF." [Online]. Available: https://www.opennetworking.org/.

[26] E. Yang, Y. Ran, S. Chen, and J. Yang, "A multicast architecture of SVC streaming over OpenFlow networks," *2014 IEEE Glob. Commun. Conf. GLOBECOM 2014*, pp. 1323–1328, 2014.

[27] Slamak Azodolmolky, *Software Defined Networking with OpenFlow*. Packt, 2013.

[28] P. Göransson, *Software Defined Networks A Comprehensive Approach [2014]*. 2014.

[29] B. Heller, "OpenFlow Switch Specification version 1.3.5," *Current*, vol. 0, pp. 1–36, 2009.

[30] "ONF Northbound Interfaces." [Online]. Available: https://www.opennetworking.org/technical-communities/areas/services/1916-northbound-interfaces.

[31] Guy Pujolle, *Software Networks- Virtualization, SDN, 5G and Security*, vol. 53, no. 9. 2013.

[32] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer, "Survey on Network Virtualization Hypervisors for Software Defined Networking," *going Pap.*, no. c, pp. 1–27, 2015.

[33] "Linux Containers LXC." [Online]. Available: https://linuxcontainers.org/lxc/introduction/.

[34] B. Pfaff, B. Lantz, B. Heller, C. Barker, D. Cohn, D. Talayco, D. Erickson, E. Crabbe, G. Gibb, G. Appenzeller, J. Tourrilhes, J. Pettit, K. Yap, L. Poutievski, M. Casado, M. Takahashi, M. Kobayashi, N. McKeown, P. Balland, R. Ramanathan, R. Price, R. Sherwood, S. Das, T. Yabe, Y. Yiakoumis, and Z. L. Kis, "OpenFlow Switch Specification version 1.3 Wire Protocol," vol. 0, pp. 0–105, 2012.

[35] "Open vSwitch." [Online]. Available: http://vswitch.org/.

[36] "Indigo." [Online]. Available: http://www.projectfloodlight.org/indigo/.

[37] "LINC." [Online]. Available: https://github.com/FlowForwarding/LINC-Switch.

[38] "Ofsoftswitch13." [Online]. Available: https://github.com/CPqD/ofsoftswitch13.

[39] "VNX (Virtual Networks over Linux)." [Online]. Available: http://web.dit.upm.es/vnxwiki/index.php/Main_Page.

[40] D. Fernández, L. Bellido, J. Ruiz, O. Walid, V. Mateos, and V. Villagra, "Virtual Networks over linuX (VNX)," pp. 105–110, 2012.

[41] "How to Collect Switch Statistics (and Compute Bandwidth Utilization)." [Online]. Available: https://floodlight.atlassian.net/wiki/pages/viewpage.action?pageId=21856267.

[42] "https://wiki.archlinux.org/index.php/Advanced_traffic_control." [Online]. Available: https://wiki.archlinux.org/index.php/Advanced_traffic_control.

[43] "Dash.js." [Online]. Available: https://github.com/Dash-Industry-Forum/dash.js/wiki.