UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

DISEÑO E IMPLEMENTACIÓN DE UNA PLATAFORMA DE ESTUDIO DE CALIDAD DE EXPERIENCIA, EN STREAMING DE VIDEO

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

JAIME ANDRÉS CARRERA TRONCOSO

PROFESOR GUÍA:
SR. JAVIER BUSTOS JIMÉNEZ

MIEMBROS DE LA COMISIÓN:
SR. JOSÉ MIGUEL PIQUER GARDNER
SR. JOSÉ PINO URTUBIA.

SANTIAGO DE CHILE
2015

Resumen

El auge de la distribución de contenido multimedia en los últimos años ha revolucionado la industria de las telecomunicaciones, produciendo consumidores más exigentes, llevando al límite a la actual infraestructura de redes y generando una percepción transversal de calidad sobre los servicios. Como consecuencia de ésto, la tecnología ha debido adaptarse rápidamente a los nuevos requerimientos, generando más y mejores opciones para crear aplicaciones que brinden la calidad solicitada. Ante la demanda de los consumidores por contar con un servicio óptimo, nace la necesidad de medir y comparar la calidad del servicio entregada, y contrastarla con experiencia del usuario al momento de que éste reciba la información, para responder adecuadamente ante los requerimientos planteados.

El presente tema de memoria tiene por objetivo implementar una plataforma de código abierto que facilite la medición de ambos parámetros de manera conjunta. La Calidad de Servicio, que hace referencia a las capacidades técnicas al momento de transmitir datos por una red y la Calidad de Experiencia, que establece el cómo percibe subjetivamente el usuario el contenido.

La solución fue diseñada en base a diferentes componentes. Primero, se diseñó un subsistema que simula los distintos tipos de redes de conexión, a los que un usuario puede estar conectado al momento de consumir contenido multimedia. Luego, se desarrolló un sistema para que un usuario pueda solicitar la creación de contenido multimedia, simulando los escenarios de conexión recién descritos. Finalmente, se estableció una metodología en donde el usuario pueda evaluar su experiencia al momento de consumir el material en cuestión.

Como resultado se obtiene una plataforma web capaz de generar contenido multimedia en demanda, que logra medir la opinión del usuario en base a los parámetros que éste asigna.

La plataforma implementada permite que a futuro se realicen diferentes estudios de manera estandarizada y con posibles mejoras técnicas que ayuden el desarrollo de las pruebas implementadas. Al ser una plataforma de código abierto, se permite que cualquier desarrollador o empresa enriquezca las funciones del actual sistema.

A Sergio Troncoso Magallanes

Agradecimientos

Debo agradecer a quienes de alguna u otra manera me ayudaron o influyeron a lo largo de mis estudios:

A mis profesores de colegio, me dieron las herramientas básicas para enfrentar los primeros desafíos de la carrera. Julia Perez, Pedro Gil, Gilberto Ponce, Fernando Arenas y Carlos Zúñiga.

A Angélica Aguirre y Sandra Gáez por siempre tener excelente disposición a ayudarme con todas mis dudas administrativas que surgieron en este proceso.

A Jo, quién con sus clases me dio ánimos de seguir la senda de las Ciencias de la Computación y luego ayudarme a entrar al mundo de las camaritas.

A mi profesor guía Javier Bustos, por su apoyo durante este trabajo y darme la oportunidad de desarrollar esta memoria. A Rubén Clavería y Hugo Meric por participar del proyecto.

A mis amigos: Gabriel Gayán, Pilar Bianchi, Ronald Poillot, Rodrigo Dueñas, Javier Vásquez, Érika Silva, Hernán Arroyo, Luis Benitez, Jorge Undurraga, Mauricio Campillo, Rodrigo Ojeda, Juan Vila, Macarena Cortinez, María José Stitic, Diego Browne, Renato Valenzuela, Fabián Bravo, Camilo Vergara, Sebastián Godoy, Gustavo Valverde, Roberto Armijo, Pablo Recabal, Mauricio Ortúzar, Cristóbal Gandini, Karim Pichara y muchos otros que acompañan de alguna manera.

A mi hermana y hermanos, por siempre estar dispuestos a recibir un *nerfaso* amigo. A Agustina por alegrar la casa.

A mis abuelos, por ser desde siempre un ejemplo de vida.

A mis padres, por aguantarme y confiar en mis decisiones.

A James, Adam, Sharon, Stephen y Ryan por su incondicional compañía, aunque fuera a la distancia.

A Nina, Carlitos y Glob.

Jaime Carrera Troncoso.
Abril de 2015.

Tabla de Contenido

Índice de cuadros

Índice de figuras

Capítulo 1

Introducción

1.1. Antecedentes generales

La industria de distribución de contenido multimedia, particularmente video, busca constantemente mejorar la calidad de distribución de su contenido.

Existen dos factores relevantes al momento de medir la calidad de la transmisión: Calidad de Servicio (Quality of Service, QoS) y Calidad de Experiencia (Quality of Experiencie, QoE).

La Calidad de Servicio está definida [Uni] como *"la totalidad de las características de un servicio de telecomunicaciones que determinan su capacidad para satisfacer las necesidades explícitas e implícitas del usuario del servicio."*

QoE ha sido estandarizada por la ITU-T [Uni07] como *"la aceptabilidad general de una aplicación o servicio, tal como la percibe subjetivamente el usuario final",* el cual *"incluye los efectos completos del sistema de punto a punto"* y *"puede estar influenciada por las expectativas del usuario y el contexto".*

Los parámetros de QoS como pérdida de paquetes, throughput[1], jitter[2], delay[3] y bitrate[4] son considerados medidas objetivas centradas en la tecnología, en contraste con las usadas por las de QoE, centradas en la opinión del usuario, la cual es posible medir usando tests MOS (Mean Opinion Score) [RS13] [MOS] , donde se establece una escala de 1 a 5; 1 es malo y 5 pasa a ser excelente.

Si bien se puede manejar fácilmente la calidad de un video mediante parámetros definidos

[1]Volumen de trabajo o de información que fluye a través de un sistema
[2]Señal de ruido no deseada
[3]Retraso modulado de una señal sonora
[4]Numero de bits transmitidos por unidad de tiempo

al momento de su construcción, éste se ve afectado camino al usuario final por distintos factores, entre ellos, la calidad de la conexión. La experiencia de reproducir un video en una red EDGE es muy distinta a la obtenida en una red local. En la primera puede haber una conexión inestable y en la segunda, garantía de datos.

La presente memoria tiene por finalidad implementar una plataforma de código abierto que permita realizar las mediciones de experiencia y servicio, especificando qué mediciones se realizarán y bajo qué entornos.

1.2. Motivación

Las herramientas que existen en la actualidad buscan una relación entre QoE y QoS de manera ad-hoc, fijando su estudio en temas como frames perdidos[5] [QOEb] [QOEa], rebuffering[6] [QOEc] y jitter [QOEa]. Consideran las perturbaciones en la red como factores principales al momento de evaluar la calidad de experiencia percibida, pero tratan el entorno como fijo y que no varía durante el estudio.

Hasta hoy, la única plataforma que permite la medición de QoS y QoE emulando distintos escenarios de red existente, que cumple con los requisitos señalados anteriormente es la presentada en su etapa de diseño en [BOX] la cual se implementa y mejora en este trabajo de memoria.

Por consiguiente, el presente tema de memoria tiene por finalidad implantar tal sistema de medición de QoS y QoE de código abierto que resuelve esta problemática, mediante el uso de tres tecnologías que también son de código abierto: ns-3[7], LXC[8] y VLC[9]. Esta plataforma llevará el nombre de BoxingExperience.

La primera tecnología a estudiar, ns-3, trata de un simulador de redes que permite simular entornos variados de uso de red; una red local, WiFi, LTE, entre otros. Permite el ajuste de parámetros de velocidad y latencia propios de diferentes tipos de redes.

[5] en films y videos, un frame o fotograma es una de las imagenes que compone la secuencia completa de imagenes

[6] Volver a cargar datos en el buffer, espacio destinado a prealocar información antes de reproducir contenido multimedia.

[7] `http://www.nsnam.org/overview/what-is-ns-3/`

[8] `https://linuxcontainers.org/lxc/introduction/`

[9] `http://www.videolan.org/`

LXC es un sistema de virtualizacíon[10] que implementa contenedores de espacios de usuario virtuales independientes, virtualización realizada a nivel de sistema operativo en Linux. Por cada contenedor, se puede asignar una tarjeta de red virtual (asignada posteriormente a ns-3) adaptada a las necesidades de medición buscadas.

Cada uno de estos entornos virtuales debería ser capaz de ejecutar VLC. VLC es un reproductor y framework multimedia. Permite transmitir videos (a través de un servidor http) como también su reproducción. De esta manera, en un contenedor es posible tener un servidor VLC y, en otro, un cliente.

Se presenta como desafío estudiar el funcionamiento y configuración de las componentes ns-3, LXC, VLC, tener la capacidad de configurarlas y automatizarlas de forma tal que se pueda desarrollar una cuarta entidad, que facilite el mostrar a los usuarios una batería de videos y permita que evalúen mediante una escala definida la calidad que perciban.

1.3. Objetivo general

El objetivo general de esta memoria es crear e implantar una aplicación que solucione el problema de medir QoS y QoE de manera centralizada, de código abierto, que cumpla la necesidad de emular distintos escenarios (distintos dispositivos, calidad de señal) para medir el cómo se verían ciertas transmisiones de video en tales situaciones.

Los objetivos específicos de este trabajo son los siguientes:

Estudio de herramientas para la automatización de un entorno variable.

Revisión y estudio de herramientas como ns-3 y LXC que permiten la creación de un entorno variable de pruebas.

Crear un subsistema para medir QoS.

Estudio y búsqueda de parámetros relevantes a medir sobre la calidad de servicio de la transmisión de video.

Crear un subsistema para medir QoE.

Estudio y búsqueda de medidas a utilizar en las pruebas de experiencia de usuario.

[10]Explicado con detalle en el capítulo 2

Crear una plataforma de medición QoE/QoS.

Teniendo claro qué parámetros de QoS y QoE considerar, se debe desarrollar una plataforma que permita su utilización.

1.4. Metodología de Trabajo

El plan de trabajo definido para alcanzar los objetivos planteados es el siguiente:

Instalación y configuración de ns-3, LXC y VLC.

Se monta un entorno de trabajo básico para realizar pruebas ad-hoc. Esto es, instalación de un ambiente mínimo que permita realizar pruebas de concepto y confirmar el correcto funcionamiento del sistema.

Diseño e implementación de la aplicación BoxingExperience.

Se diseña e implementa la aplicación. Por una parte está el código a escribir y por otra parte, el cómo presentar la aplicación al usuario.

Realizar pruebas de BoxingExperience.

Fase de tests de la aplicación donde se prueban los distintos escenarios en que se espera usar BoxingExperience.

Escritura del documento de memoria y documentación de la solución

1.5. Organización de la memoria

El informe está organizado en cuatro capítulos, de los cuales el primero de éstos corresponde al actual.

En el capítulo 2 se presentan diversos antecedentes relacionados con el trabajo realizado, definiciones sobre conceptos utilizados a lo largo de este informe y el resultado de la investigación previa al trabajo.

En el capítulo 3 se establece la metodología a seguir para el diseño e implementación de la plataforma en cuestión de medición de calidad de experiencia.

En el capítulo 4 se presentan los resultados y las conclusiones sobre el trabajo realizado, agregando ideas de posibles aplicaciones y trabajos a realizar en el futuro a partir de la solución creada.

Capítulo 2

Antecedentes

En este capítulo se plasma una investigación bibliográfica realizada acerca de los temas relativos al trabajo, explicando el rol e importancia de cada uno en la plataforma que se implementa.

2.1. Simulación de redes

La simulación de redes es una técnica donde un programa modela el comportamiento de un tipo de red, calculando la interacción entre las distintas entidades de la red a través del uso de modelos matemáticos o reproduciendo fenómenos observados en una red en funcionamiento.

Este trabajo de memoria busca emular distintos escenarios a los que el usuario se pueda ver sometido al momento de consumir material multimedia de la Internet. Por ejemplo, ver un video en la red Wi-Fi de su casa o en la red LTE de su dispositivo móvil; por lo que el uso de un simulador de redes se hace vital.

Se investigó la existencia de simuladores de red actuales, encontrándose tres soluciones. Dos se tratan de software propietario, OPNET[1] y NetSim[2]; mientras una es de código abierto, ns-3.

2.1.1. ns-3

ns-3 es un simulador de distintos escenarios de red en la Internet; de código abierto, basado en eventos discretos. Su uso está enfocado principalmente en ambientes de investigación y

[1] `http://www.riverbed.com/products/performance-management-control/opnet.html`
[2] `http://tetcos.com/`

educativos. Logra simular el comportamiento de las capas físicas y de enlace de distintos equipos de red de manera estricta. Permitiendo simular protocolos Unicast y Multicast; redes cableadas como redes inalámbricas, incluyendo redes móviles.

ns-3 no cuenta con una interfaz gráfica sino que funciona a base de scripts escritos en C++ o en Python (que traduce a C++ con PyBindGen). Los scripts en cuestión establecen una topología de cómo armar la red a simular, el tiempo de ejecución y las posibles tareas a ejecutar. Un ejemplo sería una conversación de *echos* entre dos clientes[3]. Es posible no asignar tarea alguna. La versión actual[4] está diseñada para soportar todo el flujo de trabajo de una simulación desde la configuración hasta la recolección y análisis de tramas.

Es capaz de cumplir la tarea de simular las capas inferiores del modelo TCP/IP: capa física, capa de red y capa de transporte, tal como se ilustra en la figura 2.1.

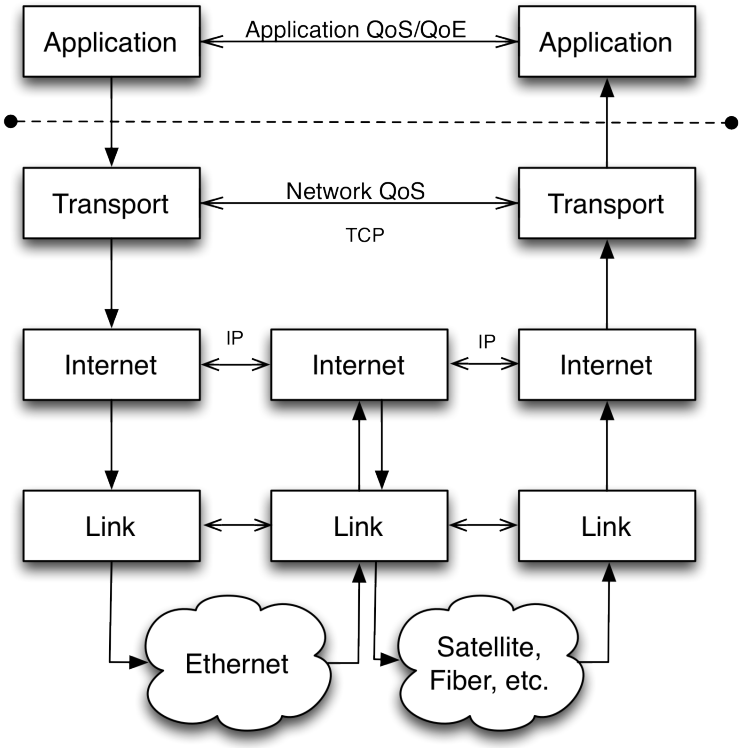


Figura 2.1: Simulación de capas de ns-3. Fuente: elaboración propia

[3] http://www.nsnam.org/docs/release/3.22/tutorial/html/building-topologies.html
[4] https://www.nsnam.org/ns-3-22/

2.2. Entornos Virtuales

Si bien es posible simular el comportamiento de múltiples interfaces de red con ns-3, el siguiente paso es lograr que múltiples máquinas sean capaces de hacer uso de ellas. Un ejemplo es tener un servidor y múltiples clientes conectados a él.

Esta memoria busca simular tal escenario haciendo uso de una sola máquina, por lo que se estudian las posibilidades existentes para cumplir tal propósito.

Para esto, se hace uso de virtualización, que en este caso busca crear una versión virtual del computador que actuaría como cliente o servidor.

Existen diferentes tipos de virtualización:

Virtualización completa : simulación prácticamente completa de una máquina, incluyendo el hardware que será manejado por el sistema operativo a elección que se instalará. Ejemplos comunes son VirtualBox[5] y VMWare Workstation[6].

Virtualización parcial : solo parte del entorno es simulado, como los espacios de direcciones. Es posible compartir recursos y alojar procesos, pero no es posible crear instancias separadas de sistemas operativos.

Paravirtualización : también conocida como virtualización por sistema operativo o liviana. El entorno de hardware no es simulado. De todas maneras, se simula un entorno para que las aplicaciones corran en un entorno aislado.

En este trabajo de memoria se decide trabajar con virtualización a nivel de sistema operativo, paravirtualización; pues no es necesario la asignación de recursos exclusiva que requieren los tipos de virtualización completa o parcial.

2.2.1. LXC

LXC es una implementación de paravirtualización, o virtualización liviana. El kernel del sistema operativo permite la ejecución de múltiples instancias aisladas de usuario en una misma máquina. Provee al usuario su propio espacio de procesos y redes, siendo para éste,

[5] `https://www.virtualbox.org/`
[6] `http://www.vmware.com/products/workstation/`

un sistema operativo aislado. A diferencia de otras tecnologías de virtualización liviana como OpenVZ[7] no requiere de parches adicionales al kernel para ser usada.

La biblioteca liblxc, da acceso a su API con conectores a múltiples lenguajes como Python, Go, Ruby y Haskell. Además de herramientas para la manipulación de los contenedores.

2.2.2. Taps y Bridges

Es posible simular entornos virtuales con LXC y simular el tráfico de interfaces de red con ns-3, por lo que es necesario saber simular interfaces de red y que puedan interactuar entre ellas.

Un *tap* es un dispositivo de red virtual a nivel de kernel simulado por software, al momento de hacer uso de éste, la única diferencia que tiene con un dispositivo común es que está soportado por software en vez de hardware.

Los *bridges* o puentes de red son interfaces (físicas o virtuales) que conectan dos o más interfaces de red independiente del protocolo, redireccionando el tráfico entre éstas. Se comporta como un switch virtual, trabajando de manera transparente para las interfaces en cuestión. En Linux están implementados por el paquete bridge-utils.

2.3. Contenido Multimedia

Como su nombre lo indica, multimedia hace referencia a contenido de múltiples medios. Permite la combinación de texto, audio, imagen y video.

En el siguiente tema de memoria el foco está en que un usuario pueda evaluar contenido multimedia, en este caso, audio y video. En la Internet, hay dos maneras de hacer llegar tal información al usuario, por descarga directa o por streaming (que se explica con mayor detalle posteriormente).

2.3.1. Archivos Multimedia

Por lo general los archivos de audio y video sin comprimir tienen grandes tamaños y se hace poco práctico compartirlos de manera separada. Para solucionar estos problemas, gene-

[7] `http://openvz.org/`

ralmente se comprime cada pista (audio y video) y se empaquetan para que sean entregados. Ambos pasos son explicados a continuación.

2.3.1.1. Codecs

Un codec es un algoritmo que indica el cómo codificar y decodificar un stream o señal. Su nombre viene del inglés, **co**der-**dec**oder.

El uso de un codec permite comprimir y descomprimir cierta señal a un formato específico con el objetivo de que pueda ser transmitida utilizando la menor cantidad de espacio posible, tal que el contenido no pierda calidad.

Usualmente se usa compresión *lossy*, que descarta parcialmente información y usa aproximaciones inexactas con el objetivo de usar menor cantidad de espacio. En contraparte, está la opción de usar compresión *lossless*, que no descarta información; usándose principalmente para almacenar material más que para transmitirlo.

En el caso de contenido multimedia se debe considerar el uso de un codec para video y otro para audio.

Para video, el codec más usado hoy en día es H.264 o MPEG-4 Part 10. Típicamente se usa en modalidad *lossy* y se encuentra en el estandard de discos Blu-ray o en sitios de streaming como Vimeo y YouTube.

Para audio se usa principalmente Opus (de muy bajo bitrate, aplicaciones en tiempo real como Skype), AAC y Dobly Digital (mayor bitrate y calidad que Opus). Todos algoritmos *lossy*.

Una vez que el contenido fue procesado, se genera un archivo, o stream de datos; que debe ser empaquetado de alguna manera para que llegue al usuario final.

2.3.1.2. x264

x264 es una biblioteca de software libre para codificar videos al format H.264/MPEG-4 AVC. Esta liberada bajo la licencia pública GNU GPL[8].

[8] `http://www.gnu.org/licenses/gpl-2.0.html`

2.3.1.3. AAC

Advanced Audio Coding es un estandard de codificación de audio *lossy*. Usualmente obtiene mejor calidad que MP3 a bitrates similares [Bra].

2.3.1.4. Contenedores

Un contenedor es un meta archivo capaz de envolver (mux) múltiples tipos de datos. En el caso de contenido multimedia, la especificación del contenedor describe la forma en que se organiza la información de audio y video en un solo archivo. Actualmente se usan principalmente:

MPEG-2 : MTS o TS, usado principalmente en transmisión de televisión y radio, como lo son DVB, ATSC e IPTV. Su variante M2TS es usada para encapsular los streams de discos Blu-ray.

MP4 : es uno de los contenedores más usados y con mayor compatibilidad. Ciertas de sus funcionalidades están patentadas y en ciertos países que reconocen las patentes de algoritmos en software se debe pagar cierto royalty por su uso[9]

Matroska mkv, estandard abierto y gratis. Puede contener una cantidad ilimitada de video, audio, imágenes e incluso subtítulos dentro de un solo archivo. Busca ser un formato universal para el almacenamiento de contenido multimedia. Lamentablemente no es compatible con muchos sistemas de reproducción multimedia.

2.3.2. Streaming

A diferencia del método de descarga directa, donde el usuario descarga un archivo completo antes de reproducirlo; el streaming es un mecanismo de transmisión de contenido donde lo que es recibido y presentado al usuario por parte de un proveedor, se hace de manera continua e idealmente sin interrupciones. Funciona mediante el uso de un buffer de datos que almacena lo que se descarga para ser presentado al usuario; a diferencia del sistema de descarga de archivos, donde se requiere que el usuario descargue la totalidad del archivo para poder acceder a éste.

[9] `http://www.mpegla.com/main/programs/M4S/Pages/PatentList.aspx`

Esta modalidad requiere que el usuario posea una conexión de al menos el mismo ancho de banda que la tasa de recepción esperada, para que pueda ver el contenido de manera continua y sin interrupciones; de no cumplirse esta condición, habrá escasez de información y tendrá que esperar a que vuelva a estar cargado el buffer o solicitar al proveedor contenido que use menos ancho de banda. El streaming multimedia se realiza principalmente a través de dos protocolos de transmisión, RTP y RTSP; el primero corre sobre UDP mientras que el segundo, sobre TCP, siendo éste el más usado actualmente.

2.3.3. Reproductores Multimedia

Un reproductor multimedia es un programa o dispositivo que permite al usuario reproducir el contenido en cuestión, ya sea un archivo almacenado de alguna manera o la recepción de un stream. Puede ser capaz de reproducir un disco Blu-ray o DVD, un disco de audio, archivos multimedia o incluso decodificar la señal de televisión.

2.3.3.1. VLC

VLC es un reproductor multimedia gratis y de código abierto. Puede ser usado como servidor y cliente, por lo que se ajusta a las necesidades de este trabajo.

VLC usa principalmente codecs que provee la biblioteca libavcodec del proyecto FFmpeg[10]

Al momento de realizar streaming VLC es capaz de codificar en tiempo real el contenido multimedia a entregar a los clientes.

2.4. Calidad de Servicio y Experiencia

2.4.1. QoS

En 1994 la ITU [Uni] define el concepto de Calidad de Servicio para el campo de la telefonía. Comprende las necesidades de todos los aspectos de una conexión, como el tiempo de respuesta del servicio, pérdida de paquetes, relación señal-ruido, diafonía, eco, entre otras.

En las redes de computadores y telecomunicaciones en general, Calidad de Servicio es la capacidad de ofrecer diferentes prioridades a las diferentes aplicaciones, usuarios, o flujos de datos tal de garantizar cierto nivel de rendimiento. Se considera dentro de estas prioridades,

[10] `https://wiki.videolan.org/FFmpeg/`

por ejemplo, bitrate, delay, jitter, tasa de bits erroneos garantizada o probabilidad de paquetes perdidos. Estos parámetros son de importancia al momento de realizar streaming multimedia, dado que la mayoría de las veces requieren un bitrate mínimo y por la naturaleza de su contenido, son sensibles a pérdida de paquetes y retardos. Un ejemplo es la pérdida de paquetes en la transmisión de un video por la Internet. Si se relaja el control de errores, es posible que al momento de ver el contenido se vean artefactos e interferencias en la imagen por falta de cuadros.

Al momento de medir la calidad del servicio de tráfico de una conexión, se consideran las siguientes variables de desempeño:

2.4.1.1. Delay

Cuando una interfaz recibe más tráfico del que puede manejar, experimenta congestión. Al haber más trafico un paquete se puede atrasar por las largas colas de espera o debe tomar una ruta menos directa para evitar la congestión en cuestión. El delay o latencia, es el intervalo de tiempo que pasa entre un estímulo y la respuesta a éste, en este caso, el tiempo que toma enviar un paquete a cierto lugar y que éste acuse su llegada.

La ITU [ITUa] define la aceptabilidad de delay en tres rangos:

Rango en milisegundos [ms]	**Descripción**
0-150	Aceptable para la mayoría de las aplicaciones
150-400	Aceptable mientras los administradores de la aplicación sepan del impacto que tiene en la calidad de transmisión.
400+	No aceptable para la mayoría de los casos.

Cuadro 2.1: Especificaciones de delay. Fuente: Recomendación ITU-T G.114

2.4.1.2. Throughput

Throughput es la tasa de paquetes que logran entregarse con éxito. Usualmente se mide en bits por segundo (bps) o paquetes por segundo.

Cuando muchos usuarios hacen uso de recursos de una red, la interfaz destino recibe más tráfico del que puede manejar, experimentando congestión. De esta manera, no todos los paquetes se entregan con éxito, disminuyendo el throughput máximo en ésta.

2.4.1.3. Jitter

Los paquetes llegan a su destino con diferente delay, pues no se puede predecir el viaje que hará cada uno camino a su destino. Esta variación de retardos recibe el nombre de Jitter.

Por lo general, para compensar el posible retardo de los paquetes, se usa un buffer de reproducción. Este buffer almacena los paquetes y luego son leídos por la aplicación que los necesite.

2.4.2. QoE

QoE está estandarizada por la ITU-T [Uni07] como la aceptabilidad general de una aplicación o servicio, tal como la percibe subjetivamente el usuario final, la cual incluye los efectos completos del sistema de punto a punto. Puede estar influenciada por las expectativas del usuario y el contexto.

Es necesario adoptar una metodología de trabajo que permita modelar el servicio y de este modo establecer un indicador de desempeño de la calidad de experiencia percibida por el usuario. La ITU-T [ITUc] recomienda el siguiente proceso para estimar la calidad de funcionamiento de un servicio punto a punto:

1. **Evaluación de la calidad de funcionamiento de la red.**
   Existen dos fuentes principales de información sobre la calidad de funcionamiento de la red, las mediciones y el modelamiento.

   a) Las mediciones permiten tratar la red como una caja negra y dan información útil para el resto del modelamiento. Se considera que la prueba no debe alterar el medio de la aplicación. Las mediciones en una red no cargada no son particularmente útiles.

   b) El cómo está modelada la red puede proporcionar la caracterización de la calidad de funcionamiento necesaria cuando la red no esté construida aún o cuando no puedan alcanzarse las consideraciones esenciales para las mediciones.

2. **Evaluación de la calidad de funcionamiento de la aplicación.**
   Los modelos de aplicación utilizan como entradas las estimaciones de la calidad de

funcionamiento de la red y dan como resultado una o varias métricas de calidad de funcionamiento de la aplicación.

3. **Evaluación de la calidad percibida como un valor de percepción del usuario.**

Con estos tres puntos que consideran información de la red, aplicación y percepción del usuario, se pueden obtener el o los indicadores de calidad de experiencia buscados.

2.4.2.1. KPI y MOS

Siguiendo la recomendación E.419 de la ITU-T [ITUb], se busca describir en detalle indicadores clave de desempeño o Key Performance Indicator (KPI) de la experiencia, tal de establecer una correlación sólida entre los factores tradicionales que sirven de indicadores para la gestión de red y el ënfoque comercial"que impera en la industria de las telecomunicaciones.

El Puntaje de Opinión Media (Mean Opinion Score, MOS) es una prueba que sigue la especificación de la ITU-T en su recomendación P.800 [ITUd] y ha sido usada en las últimas décadas para medir la experiencia del usuario en redes de telefonía.

En multimedia, el MOS proporciona una medida numérica de la calidad percibida por el usuario del contenido que está recibiendo; por ejemplo, el streaming de video de un concierto en vivo. El MOS se expresa como un número entero en el rango de 1 a 5, donde 1 es la más baja calidad percibida y 5 es la mejor. El MOS es generado por un promedio de los resultados de una serie de pruebas a un grupo de usuarios, donde cada usuario responde de acuerdo al siguiente esquema

Mean Opinion Score		
MOS	Calidad	Percepción del error
5	Excelente	Imperceptible
4	Buena	Perceptible, pero no molesta
3	Suficiente	Ligeramente molesta
2	Pobre	Molesta
1	Mala	Muy molesta

Cuadro 2.2: Valores de MOS. Fuente: Recomendación ITU-T E.419

2.5. Aplicación de mediciones

Para automatizar el proceso de creación de entornos de prueba y para que el mismo usuario sea quien los configure, se hace conveniente desarrollar una interfaz para que se pueda hacer uso de la aplicación. Para esto, se decide el uso de una aplicación web que controle remotamente la máquina donde se generan y ejecutan las simulaciones.

2.5.1. Django

Django[11] es un framework de desarrollo web de código abierto, escrito en Python[12]. Fomenta un desarrollo claro, rápido y de diseño pragmático usando el paradigma Modelo Vista Controlador.

2.5.2. Celery

Celery[13] es un gestor asíncrono de colas. Permite la creación de trabajos de manera asíncrona y en tiempo real. Escrito en Python, está pensado para ser usado en aplicaciones escritas en tal lenguaje, como lo son las aplicaciones web de Django.

2.5.3. SQLite

SQLite[14] es un sistema de gestión de bases de datos relacional. Es una biblioteca relativamente pequeña escrita en C. En contraste con otros sistemas de bases de datos, SQLIte no está implementado de manera que deba ejecutarse como un proceso separado, sino que es parte del programa que necesite hacer uso de él. Su código de fuente es de dominio público.[15]

Python posee soporte nativo para SQLite (desde la versión 2.5)

[11] https://www.djangoproject.com/
[12] https://www.python.org/
[13] http://www.celeryproject.org/
[14] http://www.sqlite.org/
[15] http://www.sqlite.org/copyright.html

Capítulo 3

Diseño e Implementación

En este capítulo se presentan los detalles referentes al diseño e implementación del presente tema de memoria.

Primero se resume cómo funciona la aplicación por el lado del usuario y posteriormente se detalla la implementación de ésta por el lado del servidor.

3.1. Diseño y uso de la aplicación

En el capítulo Introducción[1] se indica el objetivo general: una plataforma que permita al usuario simular distintos entornos de red para medir el cómo se verían ciertas transmisiones de videos en situaciones variables.

Para esto, se decide la implementación de una plataforma en que tanto los clientes como servidores se conectan a una red virtualizada por ns-3. Si bien hay un usuario final, se decide agregar clientes extras para generar tráfico en la red y poder estudiar el cómo afecta a la calidad del video entregado.

La arquitectura en cuestión se genera dinámicamente dependiendo de los parámetros que el usuario considere, con tal de definir sus propiedades. Esto, a través de una aplicación Web, como se ilustra en la figura 3.1

[1] Para mayor información véase el capítulo 1.

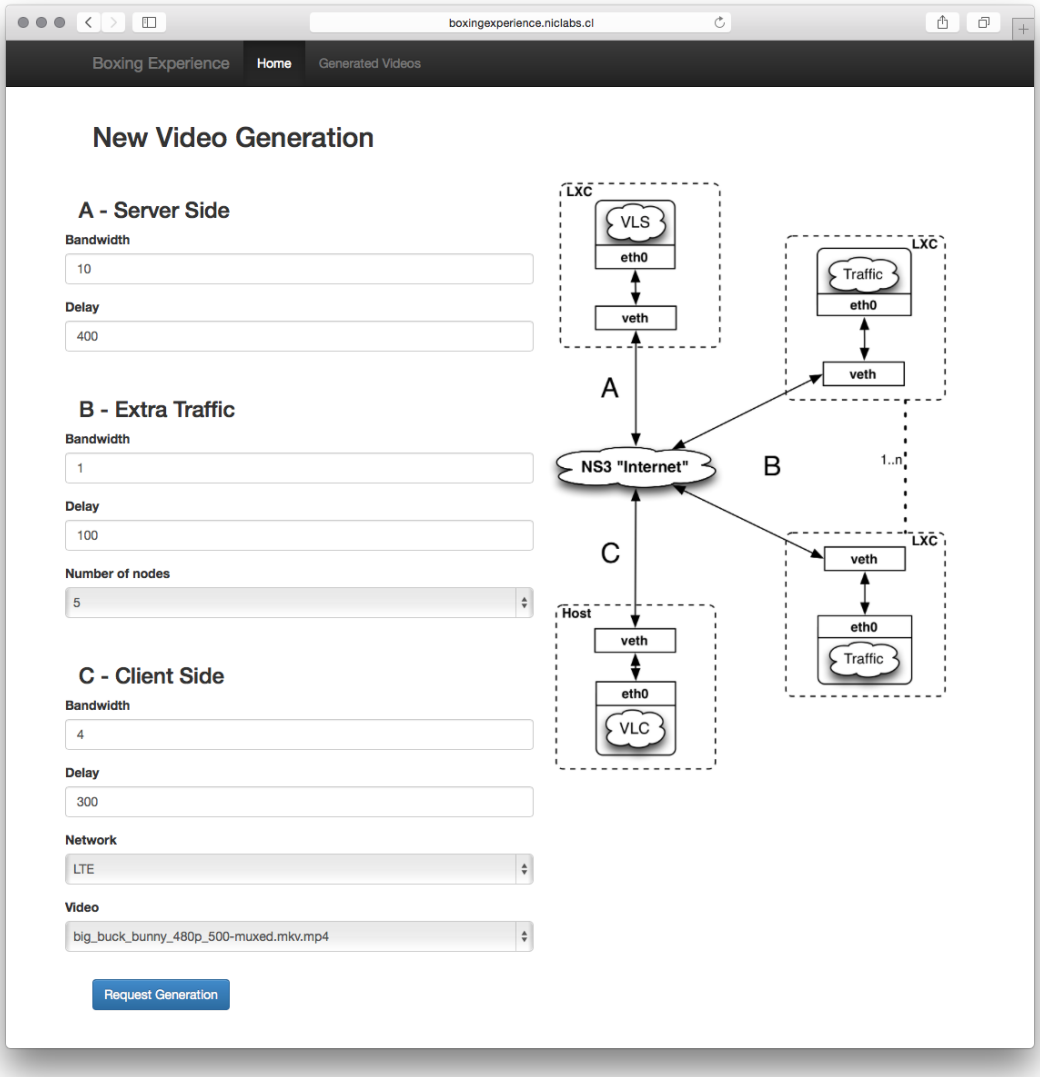


Figura 3.1: Home de interfaz web. Formulario de parámetros que el usuario debe ingresar. Fuente: elaboración propia.

La figura 3.2 indica un esquema de la configuración del servidor, clientes y las conexiones entre éstos.

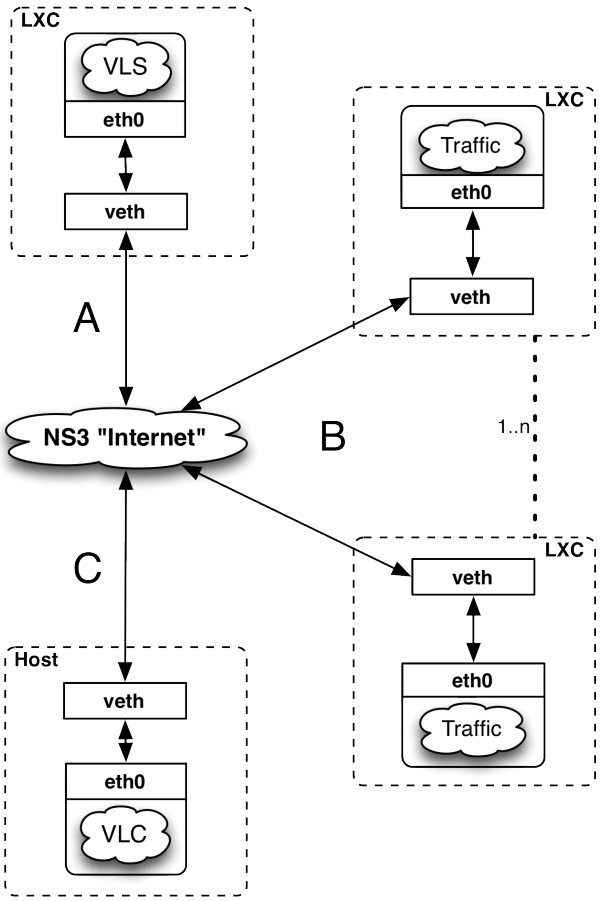


Figura 3.2: Esquema de BoxingExperience. Las distintas entidades de la solución, sus conexiones y el cómo interactúan entre ellas. Fuente: elaboración propia.

Se observa que está dividida en tres zonas:

Zona A El lado del servidor. Un contenedor LXC encapsula la máquina que se encargará de realizar streaming de video. Se conecta a una de las unidades de red virtuales que facilita ns-3. Se le puede indicar el ancho de banda (en Mbps) y delay (en ms).

Zona B Muestra una cantidad variable de clientes, cada uno en su propio contenedor. La única tarea de cada cliente es generar tráfico en la red, solicitando el material que el servidor está entregando. Al igual que el servidor, toma como parámetros el ancho de banda y delay. Se agrega la cantidad de nodos a considerar en la simulación.

Zona C El cliente final, entidad encargada de mostrar el contenido multimedia al usuario. Aparte de ancho de banda y delay, se le indica el tipo de red que debe simular y el video a solicitar.

Una vez que el cliente ingresa los parámetros que desea, la aplicación genera un nuevo video donde se ven reflejadas las posibles fallas que puede haber tenido la conexión simulada.

Se le informa al usuario que se generará el video, de donde podrá descargarlo y luego evaluarlo, como se muestra en la figura 3.3 y 3.4.

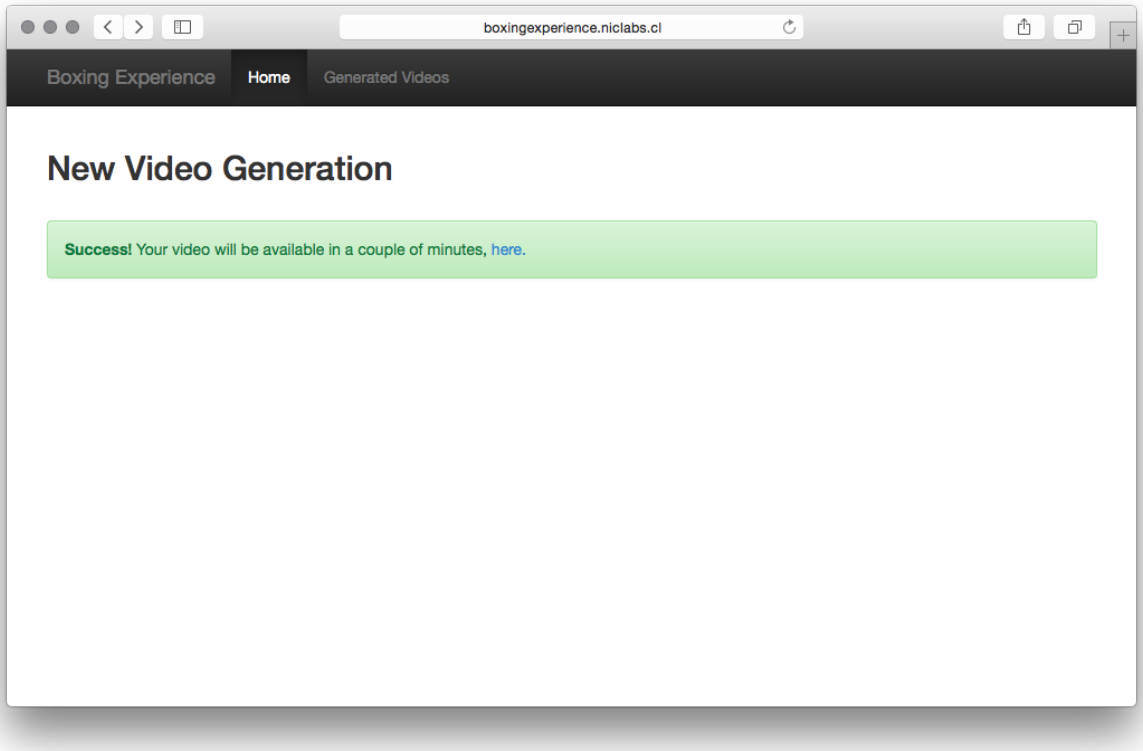


Figura 3.3: Notificación de video generándose. Fuente: elaboración propia.

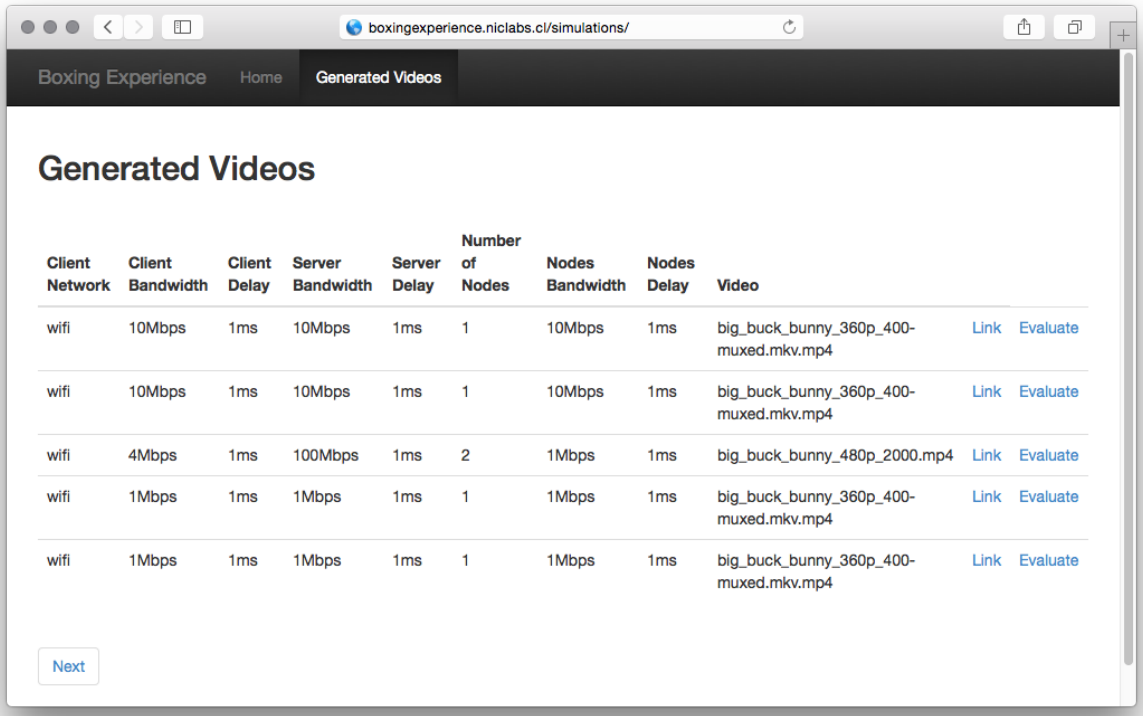


Figura 3.4: Lista de videos generados. Fuente: elaboración propia.

Al momento de evaluar, el usuario visualiza el archivo generado por la simulación e indica su opinión respecto a lo que ve, como se muestra en la figura 3.5

3.2. Implementación

El desarrollo del trabajo presentado se realiza en una máquina corriendo Linux en su variación Ubuntu 14.04.1 de 64bits[2].

3.2.1. Esquema de red

El primer paso a seguir es tener claro el esquema de red para la plataforma a desarrollar. Cada nodo virtualizado por LXC tendrá su propia dirección IP a la cual se le asignará un gateway (puerta de enlace) por donde debe ir su tráfico. Estos gateways son controlados por ns-3, quien se encarga de *rutear* los paquetes en su red virtualizada.

[2] `https://www.ubuntu.com/`

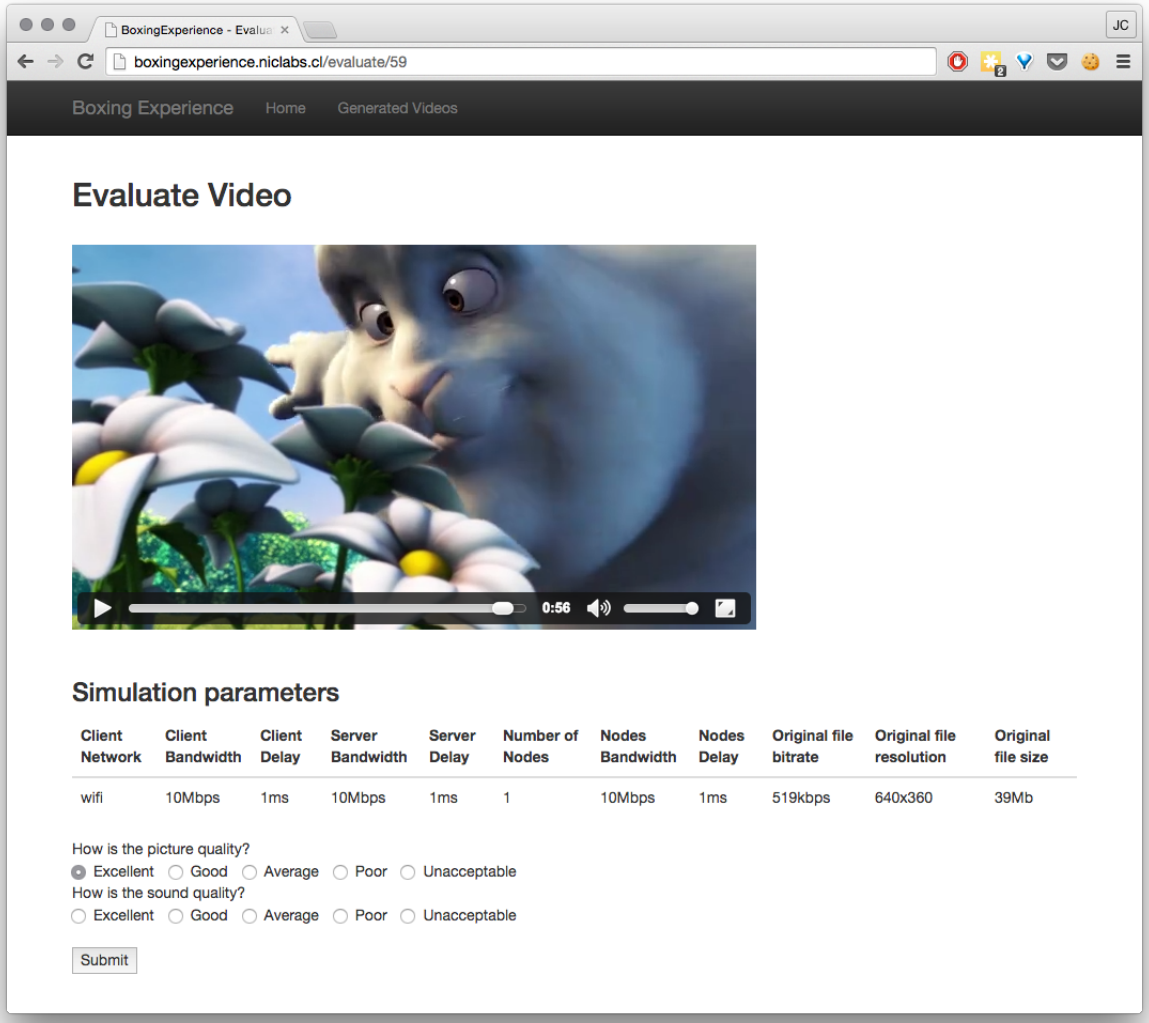


Figura 3.5: Evaluación de video generado. Fuente: elaboración propia.

La red se separa en cuatro partes. Servidor, cliente, nodos y enlace entre servidor y clientes.

Servidor : al servidor se le asigna la IP 10.100.2.2. Por el lado de ns-3, el gateway será 10.100.2.1

Cliente : el cliente recibe la IP 10.100.3.2 y gateway 10.100.3.1. En otra interfaz de red recibe la IP 10.0.3.100. Esta interfaz conversa con la máquina que lo ejecuta, tal de redireccionar el flujo de tráfico que se solicite en esa puerta a la red de ns-3, esto para posteriormente poder capturar el video que se lea.

Nodos : los nodos, como máximo 10, recibirán IPs en 10.100.4x.1 y Gateway 10.100.4x.2 donde $x \in \{0 \dots 9\}$.

Routers : para conectar las redes de clientes con la red del servidor, se usa un par de routers con IPs 10.100.1.1 y 10.100.1.2.

El esquema se puede aclarar en la figura 3.6, donde las líneas en azul indican que su tráfico es afectado por ns-3.

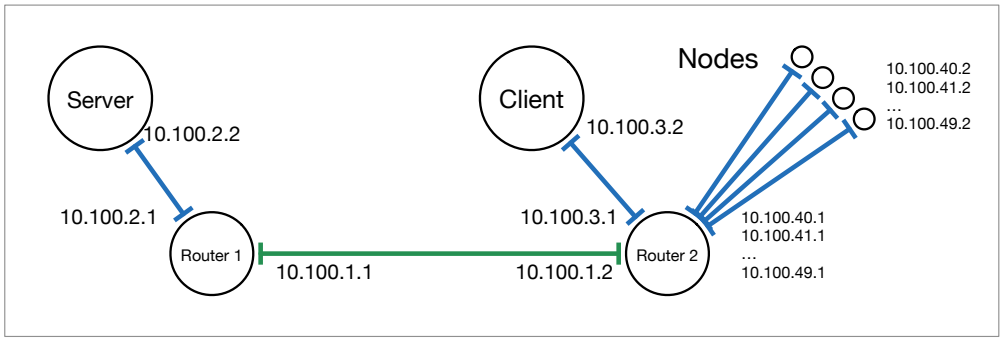


Figura 3.6: Esquema de red de Boxing Experience con IPs. Fuente: elaboración propia

3.2.2. LXC

Una vez que se tiene el esquema de red planificado, se procede a crear los contenedores descritos en la figura 3.2. Esta configuración se realiza una sola vez, pues serán constantes en el tiempo. Para la Zona B, correspondiente a los clientes que generan tráfico, se considera el uso de 10 clientes.

La forma básica de crear un contenedor en linux es

```
sudo lxc-create --name nombrecontenedor -t ubuntu
```

Donde -t ubuntu denota que es una máquina Ubuntu.

Para entrar a un contenedor basta ejecutar:

```
sudo lxc-start --name nombrecontenedor
```

Creados los contenedores, se configuran como si fueran computadores nuevos, por lo que es necesario instalar VLC en cada uno.

```
sudo apt-get update
sudo apt-get install vlc
```

Actualizados los contenedores e instaladas las aplicaciones necesarias, se procede a configurar las interfaces de red como se describió en la subsección anterior.

Dado que LXC y ns-3 tienen sus propias interfaces de red virtualizadas, para que haya comunicación entre ellas se crean puentes en la máquina que ejecuta las aplicaciones.

A modo de ejemplo, un puente para la interfaz virtual de servidor:

```
brctl addbr serverbridge
tunctl -t servertap
ifconfig servertap 0.0.0.0 promisc up
brctl addif serverbridge servertap
ifconfig serverbridge up
```

Cuando se ejecuten las pruebas ns-3 se encarga de asignar las IPs que corresponda a sus interfaces

Finalmente, es importante considerar que es la máquina principal, la que corre todos los servicios, la que se encargará de capturar el video que el contenedor Cliente recibe.

Tal objetivo se cumple utilizando el enlace explicado anteriormente, 10.0.3.100. En el contenedor cliente se agrega una regla con iptables[3] para redirigir el tráfico del puerto 8080 al servidor. Finalmente el contenedor Cliente actúa como proxy.

```
iptables -t nat -A PREROUTING -j DNAT -d 10.0.3.100 -p tcp
   --dport 8080 --to 10.100.2.2
iptables -t nat -A POSTROUTING -j MASQUERADE
```

3.2.3. ns-3

ns-3 funciona a base de archivos de configuración creados en C++ o Python, que definen la topología de la red a simular. El archivo de configuración recibe parámetros que establecen el tipo de enlace, velocidad y delay de cada entidad.

El archivo de configuración en cuestión se invoca utilizando la herramienta *waf* que provee ns-3, seguida del nombre del script y los parámetros que éste pueda tomar.

Por omisión las simulaciones corren por un tiempo predeterminado en el script, pues son trabajos pequeños que intentan mostrar el comportamiento de una tarea puntual. Abusando de esa propiedad, se indica que corra por tiempo eterno, tal que solo se detenga la simulación de redes cuando se le indique.

[3] `http://ipset.netfilter.org/iptables.man.html`

La topología a correr corresponde a la red indicada con anterioridad. Un servidor, diez nodos que generan tráfico (que no necesariamente se usan) y un cliente final.

3.2.4. Ejecución de las pruebas

Para ejecutar la ronda completa de simulaciones se escribe un script en bash que recibe los siguientes parámetros:

SERVER_BANDWIDTH	ancho de banda del servidor que realiza streaming
VIDNAME	nombre del video a *stremear* por el servidor
SERVER_DELAY	delay del servidor
NODES_NUMBER	numero de nodos que generan tráfico
NODES_BANDWIDTH	ancho de banda de los nodos
NODES_DELAY	delay de los nodos
CLIENT_BANDWIDTH	ancho de banda del cliente
CLIENT_DELAY	delay del cliente
CLIENT_NETWORK	tipo de red del cliente
OUTPUTVIDEO	nombre del video generado por la simulación

El script en cuestión, aparte de levantar el servicio de simulación de redes de ns-3, se encarga de ejecutar VNC en todas las máquinas.

En el servidor corre el servidor de videos:

```
sudo -u ubuntu vlc -vvv -I dummy /home/ubuntu/vid/'$VIDNAME'
   --sout "#standard{access=http,mux=ts,dst=@:8080}"
```

En los nodos que generan tráfico, se invoca a VLC para que consuma el archivo generado sin mostrar el video ni reproducir su audio:

```
sudo -u ubuntu cvlc --novideo --noaudio http://10.100.2.2:8080
   vlc://quit
```

Es ideal mostrar el mismo resultado a distintos usuarios para que sea evaluado múltiples veces. En vez de emular cada vez el escenario de red, se grabará el video resultante en un archivo único. En la máquina que ejecuta las pruebas, se invoca a ffmpeg para que capture la señal tal como llega; de esta manera queda grabado tal cual como un usuario lo vería si la red se comportara con esos parámetros:

```
ffmpeg -re -i http://10.0.3.100:8080 -timelimit 50 -t 50 -vsync
   0 -y -c copy $OUTPUTVIDEO
```

Como se dijo anteriormente, en la máquina cliente se redirecciona el tráfico que viene de 10.0.3.100 a 10.0.2.2. El stream de datos corre por la red virtualizada de ns-3 y se logra la simulación deseada.

3.2.5. Aplicación Web

La experiencia del usuario se vería potencialmente disminuida al eventualmente solicitarle que ejecutara comandos con los parámetros indicados anteriormente, por lo que se desarrolla una aplicación web que automatiza la ejecución de las simulaciones. Ésta se desarrolla utilizando el framework de desarrollo web Django. Para la persistencia de datos se aprovecha el uso de la base de datos que provee el mismo framework por omisión, SQLite.

Dado que no es posible ejecutar más de una simulación en paralelo, pues habría que haber encapsulado a su vez el entorno de trabajo en otro entorno virtual, lo que se escapaba del alcance de este trabajo; se encolan las tareas en una cola de trabajos que tiene un solo worker, usando Celery.

3.2.5.1. Videos de prueba

Uno de los parámetros más importantes al momento de ejecutar la simulación es qué archivo usar para ésta. Si bien es posible codificar en tiempo real videos al usar VLC en su modalidad de servidor, para este tema de memoria se usan videos estándar y con distintas resoluciones producidos previamente.

Dos videos populares y de código abierto son usados (producciones desarrolladas bajo el proyecto Open Video Project, con licencia Creative Commons Attribution 3.0[4]). Ambos son usualmente utilizados para este tipo de pruebas: Big Buck Bunny[5] y Elephants Dream[6].

Tomando las versiones sin comprimir de los videos de prueba, se generó una batería de nuevas versiones basándose en las recomendaciones de YouTube[7] para distintas resoluciones y bitrates.

Para video se utiliza H264 codificado con x264 utilizando Profile 4.1. Para las distintas resoluciones, las siguientes propiedades:

[4] http://creativecommons.org/licenses/by/3.0/
[5] https://peach.blender.org/
[6] https://orange.blender.org/
[7] https://support.google.com/youtube/answer/2853702?hl=en

1280x720

- Maximo 4000 Kbps
- Recomendado 2500 Kbps
- Mínimo 1500 Kbps

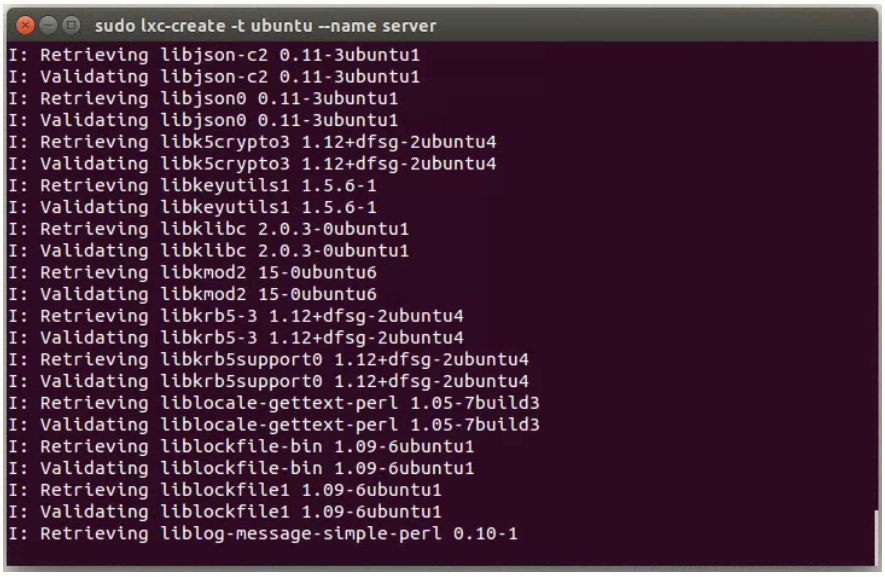
854x480

- Maximo 2000 Kbps
- Recomendado 1000 Kbps
- Mínimo 500 Kbps

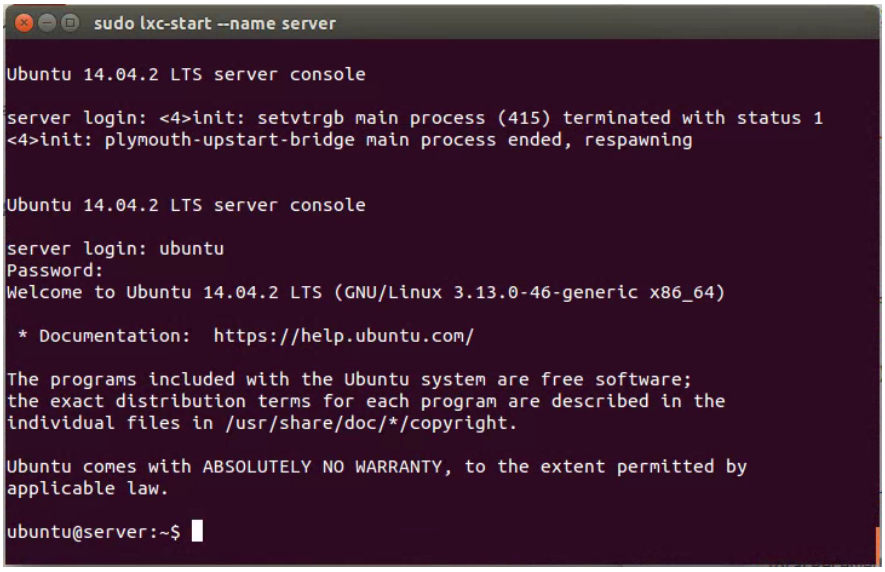
640x360

- Maximo 1000 Kbps
- Recomendado 750 Kbps
- Mínimo 400 Kbps

Para audio se utilizó AAC-LC a 48khz con factor de calidad q0.6 (aproximadamente 192kbps)



(a) Creación de un nuevo contenedor.



(b) Arranque del nuevo contenedor.

Figura 3.7: Creación y arranque de un nuevo contenedor. Fuente: elaboración propia.

Figura 3.8: Frame de video de prueba Big Buck Bunny. Fuente: peach.blender.org

Capítulo 4

Conclusiones y Trabajo Futuro

4.1. Conclusiones

Al término de este trabajo de memoria se logró cumplir con el objetivo de ésta: crear e implantar una plataforma innovadora que soluciona el problema de medir QoS y QoE de manera centralizada.

Actualmente el software permite simular distintos tipos de red y generar archivos multimedia que son transmitidos por tales medios simulados.

El usuario puede simular una configuración de red y finalmente emitir su opinión de manera exitosa, guardando los resultados de su opinión en una base de datos, tal de poder ser analizados con posterioridad.

El uso de un sistema integrado de medición de calidad de servicio y experiencia de usuario, permite una innovación importante en el ámbito de mejorar la entrega de un servicio óptimo, con respecto al escenario actual de mediciones aisladas de ambos componentes.

Dado que la calidad de la conexión de las redes es altamente variable entre distintos lugares, el contar con una herramienta integrada que simule esta variabilidad permite realizar mejores pruebas de laboratorio, sin necesidad de contar con una infraestructura técnica para generar estos casos, lo cual se traduce en un ahorro en tiempo y equipamiento para quien quiera realizar las pruebas establecidas.

4.2. Trabajo Futuro

El trabajo desarrollado durante el semestre entregó como resultado la plataforma buscada donde se puede medir QoS y QoE de manera simultánea. A partir del trabajo desarrollado, se presentan diversas opciones que pueden mejorar la solución implementada, las cuales son descritas a continuación.

4.2.1. Estandarización de pruebas

La plataforma permite que el usuario realice las pruebas en cualquier máquina. Puede darse la situación en que el usuario esté en un computador que no cumpla con las condiciones necesarias para reproducir bien un video y altere su percepción de los hechos.

En [ITUe] se propone el proceso de pruebas estandarizadas. Se estudia previamente a los sujetos que evaluarán; se define un orden estratégico para mostrar los videos y establece un lugar adecuado para que se realice el proceso en cuestión.

4.2.2. Servidor de Streaming

Actualmente existen alternativas de pago y gratuitas para realizar streaming de video. Como lo son Mist[1] y Wowza[2]. Sería prudente revisar su funcionamiento y el cómo se comportan con la plataforma desarrollada.

4.2.3. Simulaciones en tiempo real

Se consideró que el usuario final no estaría en la máquina donde se generan los videos y las redes simuladas.

Cuando se captura el video con ffmpeg[3] no se replica de manera exacta la experiencia de ver un video en tiempo real, pues es un video ya grabado. El reproductor hará su mejor esfuerzo por reconstruirlo y mostrarlo de la mejor manera.

[1] `https://mistserver.org`
[2] `http://wowza.com`
[3] Revisar Ejecución de las Pruebas en la sección 3.2.4

4.2.4. Extensiones sobre herramientas usadas

Dada la versatilidad de las herramientas utilizadas para la construcción de BoxingExperience, es de interés realizar en el futuro, extensiones sobre éstas. Por ejemplo, agregar otro tipo de topologías en ns-3 o distintos tipos de redes, simulando incluso la posición de antenas y el tráfico que existe sobre ellas en una red LTE; incluir más nodos generadores de tráfico; generar estadísticas en tiempo real acerca del contenido que se está viendo, como lo es la captura de FPS (cuadros por segundo), pérdida de paquetes, etc.

4.2.5. Correlación de QoS y QoE

Sería de interés realizar pruebas que involucraran múltiples sujetos de pruebas y experimentos, que permitieran estudiar los resultados de sus opiniones, tal de establecer relaciones claras entre los parámetros establecidos por QoS y los resultados obtenidos por QoE.

Apéndices

A . ns-3

A .1. topology.cc

Código de la topología utilizada en ns-3

```
#include <fstream>
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/csma-module.h"
#include "ns3/applications-module.h"
#include "ns3/tap-bridge-module.h"
#include "ns3/realtime-simulator-impl.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"

#define FRAME_KB 24
#define FRAME_BIT 1024*8*FRAME_KB

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("Topology");

int main (int argc, char *argv[])
{

    float stop = -1.0;
    std::string sTap = "servertap";
    std::string cTap = "clienttap0";
    std::string datarate01 = "5Mbps";
    std::string datarate02 = "200Mbps";
    std::string datarate13 = "5Mbps";

    std::string delay01 = "0.1ms";
    std::string delay13 = "0.1ms";
```

```
31
32      std::string extradatarate = "5Mbps";
33      std::string extradelay = "0.2ms";
34
35      // CommandLine Variables
36      CommandLine cmd;
37      cmd.AddValue("duration", "Simulation length, in seconds", stop);
38      cmd.AddValue("datarate01", "Link bandwith, for example \"5Mbps\" or
           \"1800Kbps\"", datarate01);
39      cmd.AddValue("datarate02", "Link bandwith, for example \"5Mbps\" or
           \"1800Kbps\"", datarate02);
40      cmd.AddValue("datarate13", "Link bandwith, for example \"5Mbps\" or
           \"1800Kbps\"", datarate13);
41      cmd.AddValue("nodesBW", "Link bandwith, for example \"5Mbps\" or
           \"1800Kbps\"", extradatarate);
42      cmd.AddValue("serverdelay", "Link bandwith, for example \"5Mbps\"
           or \"1800Kbps\"", delay01);
43      cmd.AddValue("clientdelay", "Link bandwith, for example \"5Mbps\"
           or \"1800Kbps\"", delay13);
44      cmd.AddValue("nodesdelay", "Link bandwith, for example \"5Mbps\" or
           \"1800Kbps\"", extradelay);
45      cmd.Parse (argc, argv);
46
47      // TimeSetResolution
48      Time::SetResolution (Time::NS);
49      GlobalValue::Bind ("SimulatorImplementationType", StringValue
           ("ns3::RealtimeSimulatorImpl"));
50      GlobalValue::Bind ("ChecksumEnabled", BooleanValue (true));
51
52      // This will stop ns-3 if it can't keep up with the traffic to
           simulate.
53      Config::SetDefault
           ("ns3::RealtimeSimulatorImpl::SynchronizationMode",
           StringValue("HardLimit"));
54      // Default time difference is 100 ms
55      Config::SetDefault ("ns3::RealtimeSimulatorImpl::HardLimit",
           TimeValue(Time("100ms")));
56
57
58      // TOPOLOGY
59      // Creating Nodes
60      NodeContainer c;
61      c.Create (4);
62      /* 0: ROUTER 0; 1: ROUTER 1; 2: SERVER; 3: CLIENT; 4: SOURCE; 5:
           SINK */
```

```
63      // shared traffic: router0-router1
64      NodeContainer n0n1 = NodeContainer (c.Get (0), c.Get (1));
65      // server-router0
66      NodeContainer n0n2 = NodeContainer (c.Get (0), c.Get (2));
67      // client-router1
68      NodeContainer n1n3 = NodeContainer (c.Get (1), c.Get (3));
69
70
71      // Creating Channels
72      CsmaHelper csma;
73      csma.SetChannelAttribute("DataRate", StringValue (datarate01));
74      csma.SetChannelAttribute("Delay", StringValue(delay01));
75      NetDeviceContainer d0d1 = csma.Install (n0n1);
76      csma.SetChannelAttribute("DataRate", StringValue (datarate02));
77      csma.SetChannelAttribute("Delay", StringValue("0.0ms"));
78      NetDeviceContainer d0d2 = csma.Install (n0n2);
79      csma.SetChannelAttribute("DataRate", StringValue (datarate13));
80      csma.SetChannelAttribute("Delay", StringValue(delay13));
81      NetDeviceContainer d1d3 = csma.Install (n1n3);
82
83      // Nodes for extra clients
84      NetDeviceContainer deviceContainer;
85      NodeContainer cc;
86      cc.Create (10);
87
88      for (uint32_t i=0; i < 10; i++)
89      {
90          NodeContainer cnodes;
91          cnodes.Add (c.Get (1));
92          cnodes.Add (cc.Get (i));
93          csma.SetChannelAttribute("DataRate",
                StringValue(extradatarate));
94          csma.SetChannelAttribute("Delay", StringValue(extradelay));
95          deviceContainer.Add (csma.Install (cnodes));
96      }
97
98          // TAP-BRIDGE Settings
99          // Connect pre-configured tap devices to each ghost node:
100     TapBridgeHelper tapBridge;
101     tapBridge.SetAttribute ("Mode", StringValue ("UseBridge"));
102         // - Server
103     tapBridge.SetAttribute ("DeviceName", StringValue (sTap));
104     tapBridge.Install (c.Get (2), d0d2.Get (1));
105         // - Client
106     tapBridge.SetAttribute ("DeviceName", StringValue (cTap));
```

```
107     tapBridge.Install (c.Get (3), d1d3.Get (1));
108
109         // Taps & bridges for extra clients
110     for (uint32_t i=0; i < 10; i++)
111     {
112         std::stringstream tapname;
113         tapname << "extratap" << i;
114         tapBridge.SetAttribute ("DeviceName", StringValue (tapname.str
                ().c_str ()));
115         tapBridge.Install (cc.Get (i), deviceContainer.Get (2*(i)+1));
116     }
117
118         // It's necesary to assign IPs to each device that is not
                connected to a pre-configured tap device.
119     InternetStackHelper stack;
120     stack.Install (c.Get(0));
121     stack.Install (c.Get(1));
122
123         // SETTING IP ADDRESSES
124     Ipv4AddressHelper ipv4;
125     ipv4.SetBase ("10.100.1.0", "255.255.255.0");
126     Ipv4InterfaceContainer i0i1 = ipv4.Assign (d0d1);
127     ipv4.SetBase ("10.100.2.0", "255.255.255.0");
128     ipv4.Assign(d0d2.Get (0));
129     ipv4.SetBase ("10.100.3.0", "255.255.255.0");
130     ipv4.Assign(d1d3.Get (0));
131
132         // IP addresses for extra clients
133     Ipv4AddressHelper addressHlpr;
134
135     for (uint32_t i=0; i < 10; i++){
136         std::stringstream sIP;
137         sIP << "10." << 100 << ".4" << i << ".0";
138         addressHlpr.SetBase(sIP.str ().c_str (), "255.255.255.0");
139         addressHlpr.Assign(deviceContainer.Get (2*(i)));
140     }
141
142     Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
143
144     Simulator::Stop (Seconds(stop));
145     NS_LOG_UNCOND ("[Time (s): " << Simulator::Now ().GetSeconds () <<
            "\t ] \t Simulation starts");
146     Simulator::Run ();
147     NS_LOG_UNCOND ("[Time (s): " << Simulator::Now ().GetSeconds () <<
            "\t ] \t Simulation has finished");
```

```
148     Simulator::Destroy ();
149
150     return 0;
151
152 }
```

B . Creación de redes

B .1. `create_interfaces.sh`

Script para creacion de bridges

```
1  #!/bin/bash
2  clientbridge=clientbridge0
3  clienttap=clienttap0
4  serverbridge=serverbridge
5  servertap=servertap
6
7  create_interfaces()
8  {
9      # SERVER
10     brctl addbr $serverbridge
11     tunctl -t $servertap
12     ifconfig $servertap 0.0.0.0 promisc up
13     brctl addif $serverbridge $servertap
14     ifconfig $serverbridge up
15
16     # CLIENT
17     brctl addbr $clientbridge
18     tunctl -t $clienttap
19     ifconfig $clienttap 0.0.0.0 promisc up
20     brctl addif $clientbridge $clienttap
21     ifconfig $clientbridge up
22
23     # NODES
24     bridgename=extrabridge
25     tapname=extratap
26     for (( i=0; i < 10; i++ ))
27     do
28         brctl addbr $bridgename${i}
29         tunctl -t $tapname${i}
30         ifconfig $tapname${i} 0.0.0.0 promisc up
31         brctl addif $bridgename${i} $tapname${i}
32         ifconfig $bridgename${i} up
```

```
    done

}

create_interfaces
```

C . Archivo de vistas Django

C .1. views.py

Archivo correspondiente a la capa controlador del proyecto Django de BoxingExperience

```
import json
import os
import sys
import re
import tempfile

from django.shortcuts import render
from django.views.decorators.csrf import csrf_exempt
from django.http import HttpResponseRedirect, HttpResponse,
   HttpResponseNotAllowed, HttpResponseBadRequest
from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger

from web.models import Simulation, Video, Evaluation
from worker.tasks import add_task

def index(request):

    videos = Video.objects.all()
    context = {
        'request': request,
        'videos': videos,
        }
    return render(request, 'web/index.html', context)

def simulations(request):
    if request.method == 'GET':
        context = {'request': request}
        simulations = Simulation.objects.all().order_by('-created_at')
        paginator = Paginator(simulations, 5)
        page = request.GET.get('page')
        try:
            simulations = paginator.page(page)
```

```
32          except PageNotAnInteger:
33              simulations = paginator.page(1)
34          except EmptyPage:
35              simulations = paginator.page(paginator.num_pages)
36          context['simulations'] = simulations
37
38      return render(request, 'web/simulations.html', context)
39
40  def request_video(request):
41      if request.method == 'POST':
42          response_data = {}
43
44          bandwidth_server = request.POST.get('bandwidth_server', None)
45          delay_server = request.POST.get('delay_server', None)
46          bandwidth_traffic = request.POST.get('bandwidth_traffic', None)
47          delay_traffic = request.POST.get('delay_traffic', None)
48          nodes_traffic = request.POST.get('nodes_traffic', None)
49          bandwidth_client = request.POST.get('bandwidth_client', None)
50          delay_client = request.POST.get('delay_client', None)
51          network_client = request.POST.get('network_client', None)
52          video_id = request.POST.get('video', None)
53
54          if bandwidth_server and delay_server and bandwidth_traffic and
               delay_traffic and nodes_traffic and bandwidth_client and
               delay_client and network_client and video_id:
55              video = Video.objects.get(pk=video_id)
56              output =
                    bandwidth_server+'sbw'+delay_server+'sd'+bandwidth_traffic+'nbw'+de
57              simu_kwards = {
58                  'bandwidth_server': bandwidth_server,
59                  'delay_server': delay_server,
60                  'bandwidth_traffic': bandwidth_traffic,
61                  'delay_traffic': delay_traffic,
62                  'nodes_traffic': nodes_traffic,
63                  'bandwidth_client': bandwidth_client,
64                  'delay_client': delay_client,
65                  'network_client': network_client,
66                  'video': video,
67                  'output': output,
68              }
69              simulation = Simulation.objects.create(**simu_kwards)
70              if simulation:
71                  command_line = 'sudo
                        /home/jcarrera/Desktop/BoxingExperience/WebApp/run_test.sh'+'
                        '+bandwidth_server+'Mbps '+delay_server+'
```

```
                    '+bandwidth_traffic+'Mbps '+delay_traffic+'
                    '+nodes_traffic+' '+bandwidth_client+'Mbps
                    '+delay_client+' '+network_client+'
                    '+video.file_name+' '+output+' '+video.resolution+'
                    &'
72                  print command_line
73                  add_task.delay(command_line)
74                  status = 'success'
75                  response_data['output'] = output
76              else:
77                  status = 'fail'
78          else:
79              status = 'fail'
80
81          response_data['status'] = status
82
83          return HttpResponse(json.dumps(response_data),
                content_type="application/json")
84      else:
85          return HttpResponseNotAllowed(['POST'])
86
87  def evaluate(request, simulation_id=None):
88      context = {'request': request, 'simulation_id': simulation_id,}
89      if request.method == 'GET':
90          try:
91              simulation = Simulation.objects.get(id=simulation_id)
92              context['simulation'] = simulation
93          except:
94              context['fail'] = True
95      if request.method == 'POST':
96          simulation_id = request.POST.get('simulation_id', None)
97          score_1 = request.POST.get('score_1', None)
98          score_2 = request.POST.get('score_2', None)
99          if simulation_id and score_1:
100             simulation = Simulation.objects.get(id=simulation_id)
101             Evaluation.objects.create(simulation=simulation,
                    score_1=score_1, score_2=score_2)
102     else:
103         return HttpResponseNotAllowed(['POST', 'GET'])
104
105     return render(request, 'web/evaluate.html', context)
```

Bibliografía

[BOX] Boxing experience: Measuring qos and qoe of multimedia streaming using ns3, lxc and vlc.

[Bra] Karlheinz Brandenburg. Mp3 and aac explained.

[ITUa] International telephone connections and circuits – general recommendations on the transmission quality for an entire international telephone connection.

[ITUb] Itu-t e.419 recommendation. business oriented key performance indicators for management of networks and services. international telecommunication union, telecommunication standardization sector.

[ITUc] Itu-t g.1030 recommendation. quality of service and performance – generic and user related aspects. estimating end-to-end performance in ip networks for data applications. international telecommunication union, telecommunication standardization sector.

[ITUd] Itu-t p.800 recommendation. methods for subjective determination of transmission quality".

[ITUe] Itu-t p.910 subjective video quality assessment methods for multimedia applications.

[MOS] Última visita en abril de 2014.

[QOEa] J.pokhrel, b.wehbi, a.morais, a.cavalli and e.allilaire, "estimation of qoe of video traffic using a fuzzy expert system," in consumer communications and networking conference (ccnc), 2013 ieee. ieee, 2013, pp. 224–229.

[QOEb] K. piamrat, c. viho, j. bonnin, and a. ksentini, "quality of experience measurements for video streaming over wireless networks," in informa- tion technology: New generations, 2009. itng'09. sixth international conference on. ieee, 2009, pp. 1184–1189.

[QOEc] R. k. mok, e. w. chan, and r. k. chang, "measuring the quality of experience of http video streaming," in integrated network management (im), 2011 ifip/ieee international symposium on. ieee, 2011, pp. 485–492.

[RS13] L. Janowski S. Egger R. Schatz, T. Hoßfeld. From packets to people: quality of experience as a new measurement challenge. page 219–263, 2013.

[Uni] I.T. Union. Quality of telecommunication services: concepts, models, objectives and dependability planning – terms and definitions related to the quality of telecommunication services. In *ITU-T Recommendation ITU-T E.800*.

[Uni07] I.T. Union. Vocabulary and effects of transmission parameters on customer opinion of transmission quality, amendment 2. In *ITU-T Recommendation P.10/G.100, Tech Rep.*, 2007.