ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Grado

Combinación de técnicas de simulación y virtualización para analizar el comportamiento de aplicaciones reales.

(On the combination of simulation and virtualization techniques to analyze the behaviour of real applications)

Para acceder al Titulo de

Graduado en Ingeniería de Tecnologías de Telecomunicación

Autor: Juan Alfonso Arroyo Ruiz
Director: Ramón Agüero Calvo / Luis F. Diez

Julio - 2015

E.T.S. DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACION

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO FIN DE GRADO

Realizado por: Juan Alfonso Arroyo Ruiz
Director del TFG: Ramón Agüero Calvo / Luis F. Diez
Título: "Combinación de técnicas de simulación y virtualización para analizar el comportamiento de aplicaciones reales "
Title: " On the combination of simulation and virtualization techniques to analyze the behaviour of real applications**"**

Presentado a examen el día:
para acceder al Título de

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente (Apellidos, Nombre): Ruiz Robredo, Gustavo.
Secretario (Apellidos, Nombre): Sanchez Gonzalez, Luis.
Vocal (Apellidos, Nombre): Agüero Calvo, Ramón.
Este Tribunal ha resuelto otorgar la calificación de:

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

Fdo.: El Director del TFG
(sólo si es distinto del Secretario)

Vº Bº del Subdirector

Trabajo Fin de Grado Nº
(a asignar por Secretaría)

RESUMEN EJECUTIVO / EXECUTIVE SUMMARY

COMBINACIÓN DE TÉCNICAS DE SIMULACIÓN Y VIRTUALIZACIÓN PARA ANALIZAR EL COMPORTAMIENTO DE APLICACIONES REALES

Cada vez es más habitual analizar el impacto que diferentes técnicas, protocolos y algoritmos tienen sobre servicios reales. Esto es especialmente relevante para aquellos relacionados con el streaming de vídeo. Por otro lado, cuando se usan plataformas de simulación se carece de un modelado realista de algunos aspectos, que es especialmente importante desde el punto de vista de las aplicaciones. Así, es de esperar que sea interesante aprovechar la capacidad que ofrece el simulador de redes ns-3 para utilizar nodos virtuales. Con el fin de utilizar aplicaciones sobre estos nodos, es necesario que mimeticen todas las características de dispositivos reales y, así, analizar las posibilidades que existen. Finalmente llevando a cabo una prueba de concepto sobre un escenario simulado con tecnologías inalámbricas y aplicaciones reales.

ON THE COMBINATION OF SIMULATION AND VIRTUALIZATION TECHNIQUES TO ANALYZE THE BEHAVIOR OF REAL APPLICATIONS

Analyses about the impact on real services that different techniques, protocols and algorithms suppose, are getting their importance increased. This is especially relevant for those related to video streaming. On the other hand, when using simulation platforms, there is a lack on realistic modelling of some aspects, which is especially important from the point of view of applications. Thus, it is expected to have an interest about taking advantage of the capacity the network simulator ns-3 offers for node virtualization. With the purpose of using applications on these nodes, it is necessary that they mimic all the characteristics real devices have and, thus, analyse the existing possibilities. Finally there will be a concept test on a simulated scenario with wireless technologies and real applications.

ÍNDICE

INTRODUCCIÓN

El número de conexiones a internet ha crecido rápidamente en los últimos años, y el número de puntos de acceso globales crecerá de 47.7 millones (Finales de 2014) a más de 340 millones en 2018 [http://www.ipass.com/press-releases/ipass-wi-fi-growth-map-shows-one-public-hotspot-for-every-20-people-on-earth-by-2018/]. Casi un punto de acceso por cada veinte personas en la tierra (Datos de iPass). Como consecuencia aparecen continuamente nuevos servicios orientados a un gran número de usuarios. Algunos ejemplos son los streamings de video, o los sitios para compartir/almacenar archivos. Estos servicios están pensados para servir a un gran número de usuarios, con una tasa muy alta de disponibilidad. Por lo tanto la escalabilidad es un aspecto clave en su diseño. Un cliente que no puede acceder a un servicio, puede perderse, y por lo que es fundamental comprobar el comportamiento del sistema antes de ponerlo en marcha. Esta evaluación incluye pruebas de aplicaciones, arquitectura de red o protocolos para mantener el servicio activo.

La evaluación del comportamiento de nuevos servicios puede llevarse a cabo de diferentes maneras. Una alternativa es la simulación de todo el sistema con modelos de red, equipos y aplicaciones. Esta manera de evaluar el servicio depende en gran medida de la precisión de los modelos y no puede ser llevada a cabo con usuarios reales, ya que no se utilizan aplicaciones ni tiempo reales, y puede requerir la reimplementación de algunas partes, antes de ser implementado. Por otro lado, las llamadas "test-beds" usan equipos y aplicaciones reales que los usuarios pueden utilizar, pero pueden ser muy costosas, ya que requieren equipos reales y arquitecturas de red complicadas. La emulación está entre las simulaciones y los "test-beds", pues combina aplicaciones o dispositivos reales con una topología de red virtualizada. Si, además, funcionan en tiempo real, pueden hacer uso de aplicaciones reales y . Aun así, dependen en gran medida de la precisión de los modelos de red, pero hay un ahorro muy significativo en equipos. Todo esto hace que las emulaciones sean una opción muy atractiva.

Para llevar a cabo las emulaciones en este proyecto, se utiliza el simulador de redes NS3. Es probablemente la alternativa más utilizada, con extensiones que permiten realizar emulaciones, ya que usa un organizador de eventos y dispositivos virtuales, taps, en tiempo real. Estos taps pueden ser asociados con las aplicaciones y los nodos en una red virtual. Así, las aplicaciones pueden intercambiar tráfico sobre una topología simulada.

Sin embargo, la emulación tiene unos límites impuestos por el equipo sobre el que se llevan a cabo y por el mismo NS3. La identificación de estos límites es muy importante para saber lo que es posible emular y saber que los resultados que obtenidos son, o no, los que se podrían esperar en un entorno real.

Otros aspectos clave a conocer a la hora de probar un nuevo servicio son la calidad de servicio QoS y la calidad de experiencia QoE. La ITU define la QoS como las características de un servicio de telecomunicación que influyen en su capacidad para satisfacer las necesidades del usuario del servicio. La QoE se define como la aceptabilidad de una aplicación o servicio al ser percibido, subjetivamente, por el usuario. Así la QoE incluye todos los efectos del sistema (clientes, terminales, red, infraestructura…) y el grado de aceptabilidad que un usuario subjetivamente percibe.

En este trabajo de fin de grado se emulan diferentes escenarios con el simulador NS3 y Linux Containers, en los que se analizan parámetros de QoS, como el jitter, pérdida de paquetes, retardo y rendimiento, y se evalúa la calidad de experiencia (QoE), para comprobar el comportamiento del NS3 con los contenedores de Linux y aplicaciones reales.

ESTRUCTURA DE LA MEMORIA

Esta memoria está estructurada de tal manera que permita conocer la motivación de este proyecto en primer lugar, tras lo que se muestra el estudio que hay hasta el momento en las diferentes tecnologías que se utilizarán. Todo esto sirve el propósito de exponer la base sobre la que se va a trabajar hasta cumplir el objetivo del proyecto.

Posteriormente, aparece el diseño de la solución, parte en la que se documentan todos los pasos dados y las configuraciones de todas las herramientas que se van a utilizar. Por lo que a partir de este punto se muestran las simulaciones y resultados obtenidos, los cuales permiten entender el comportamiento de las herramientas y alcanzar el objetivo marcado en un principio, conocer las limitaciones del ns-3 con aplicaciones reales, reflejadas en las conclusiones.

Finalmente, se reflejan unas líneas futuras que se podrían seguir a la finalización este proyecto.

ESTADO DEL ARTE

INTRODUCCIÓN

En esta sección del trabajo se exponen las diferentes tecnologías/herramientas que se van a usar. Primero se muestran los conceptos detrás de estas y, a medida que se avanza, se puede ver la evolución hacia la parte de estas que más interés tiene para realizar este proyecto.

Así, esta sección se divide en cuatro partes, cada una de las cuales representa un pilar de este proyecto.

VIRTUALIZACIÓN

INTRODUCCIÓN

La virtualización es una tecnología emergente, que ofrece multitud de beneficios. Se ha aplicado en múltiples aspectos de la tecnología de la información, como en sistemas, almacenamiento, redes, seguridad y aplicaciones, utilizándose en pruebas, entrenamiento, desarrollo y entornos de producción.

Para definir la palabra virtualización, se puede decir que se trata de una tecnología que introduce una capa de software para abstraer la capa del hardware del sistema operativo, como las máquinas virtuales guest. Esta capa de abstracción se conoce como VMM (Virtual Machine Monitor) o hipervisor.

La virtualización comenzó en los computadores de los años sesenta para dividir los recursos del sistema entre diferentes aplicaciones. IBM inventó el concepto de máquina virtual, y desde que en 1987 Insignia Solutions demostrara un software emulador llamado SoftPC, que era capaz de correr aplicaciones DOS en estaciones de trabajo UNIX, la virtualización se ha desarrollado enormemente.

Cualquier entorno virtualizado consiste de un VMM o hipervisor, cuyo propósito es asignar los recursos físicos, como la CPU, memoria, red y almacenamiento, a cada SO virtualizado o a cada aplicación ejecutándose en un SO virtualizado. Una vez el hipervisor está activo, emula un dispositivo de hardware para cada SO virtual, y maneja las comunicaciones de cada SO virtual con los recursos físicos.

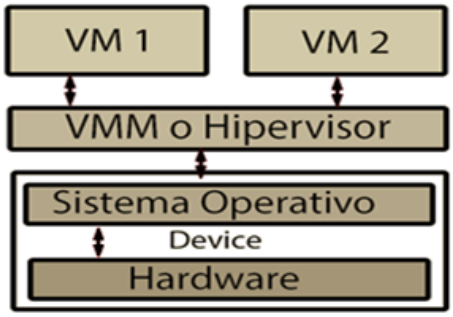


Figura 1. Ilustración entorno virtualizado

La principal característica del hipervisor es que controla directamente los recursos físicos, algo que tradicionalmente hace el SO del host o el hardware físico. Gracias a esto se pueden ejecutar múltiples sistemas operativos en el mismo hardware al mismo tiempo, mientras que son aislados unos de otros. Como resultado, el hardware se particiona en una o más unidades lógicas, conocidas como máquinas virtuales (VM, Virtual Machine).

La virtualización tiene una serie de ventajas e inconvenientes, como cualquier otra tecnología.

Los beneficios son:

1. Concentración: el principal objetivo es combinar y unificar. Los trabajos se combinan en menos plataformas físicas, las que son capaces de soportar la demanda de recursos de computación, como CPU, memoria, I/O. El resultado es una utilización de recursos más eficiente y la optimización del hardware. Adicionalmente, la virtualización ayuda a simplificar la migración de sistemas.
2. Fiabilidad: la virtualización mantiene la funcionalidad y disponibilidad de operaciones gracias a que proporciona un alto aislamiento entre máquinas virtuales. Si una máquina virtual falla, esto no afecta a otras particiones que se están ejecutando en la misma plataforma física. Además, la virtualización permite asignar recursos dinámicamente a una única partición en un momento dado.
3. Seguridad: la virtualización añade el valor de la seguridad a través de la encapsulación y aislamiento de los sistemas operativos de las máquinas virtuales. Si una partición está comprometida, esto detiene la extensión del problema a otras particiones que están en la misma plataforma de virtualización.

Las desventajas de la virtualización:

1. Rendimiento: esto es un aspecto importante.
2. Redundancia: las organizaciones dependen inmensamente de la accesibilidad y el procesado de datos. Necesitan asegurar que este tipo de acceso esté altamente disponible y que en cualquier fallo de hardware, la carga de trabajo se transferirá a otro sistema.
3. Operaciones: los profesionales responsables de entornos virtualizados deben considerar el coste de mantenimiento que acompaña el desarrollo de entornos virtualizados, como licencias, mejoras de software, tiempo de vida de las máquinas virtuales, mantenimiento de hardware...

Existen diferentes tipos de virtualización que son desarrollados en entornos de IT.

Virtualización de hardware.

Se refiere a la creación de una máquina virtual que actúa como un ordenador real con un sistema operativo. El software ejecutado sobre estas máquinas virtuales está separado de los recursos hardware subyacentes.

En la virtualización de hardware, la máquina host (host en adelante) es aquella en la que la virtualización toma lugar, y la guest, es la máquina virtual. Las palabras host y guest se usan para diferenciar el software que corre sobre la maquina física, del software que corre sobre la máquina virtual. El software que crea una máquina virtual en el host es llamado hypervisor o administrador de máquina virtual.

Hay diferentes tipos de virtualización de hardware:

- Virtualización completa: se simula el hardware real, permitiendo que el software, normalmente el sistema operativo del guest, pueda ejecutarse sin ser modificado.
- Virtualización parcial: algunos aspectos del hardware son simulados y ciertos programas del guest necesitan modificaciones para poder ejecutarse en este entorno virtual.
- Paravirtualización: el hardware no es simulado. Sin embargo, los programas del guest son ejecutados en sus propios dominios aislados, como si se estuviesen ejecutando en otro sistema. En este caso, los programas del guest tienen que ser modificados necesariamente.

Virtualización de escritorio.

Es el concepto por el que se separa el escritorio de la máquina física. Puede verse como una forma más avanzada de virtualización de hardware, en la que en lugar de interactuar con el host directamente, se hace a través de otro escritorio, usando una conexión de red.

Virtualización de software.

También conocido como virtualización de aplicaciones, consiste en ejecutar software desde un servidor remoto. De esta manera, el ordenador local no sufre ningún cambio, mientras que el servidor remoto realiza el trabajo. Los beneficios incluyen disponibilidad de más recursos cuando son necesarios, y ahorro en hardware, por incluir algunos.

Virtualización de memoria.

En este tipo de virtualización un equipo puede acceder a un conjunto de memoria RAM virtual, que consta de recursos de memoria RAM de sistemas individuales de un centro de datos.

Virtualización de almacenamiento.

Es la unión de múltiples dispositivos de almacenamiento formando una unidad de almacenamiento.

Virtualización de red.

Es el proceso de combinar todos los recursos hardware y software de red, y sus funcionalidades, en una única entidad administrativa, una red virtual.

CONTENEDORES DE LINUX

LXC, o Linux Containers, es un método de paravirtualización que permite la creación y administración de varios sistemas Linux, aislados entre sí, en un único sistema host.

"LXC es una interfaz en el espacio de usuario para las funciones de contención del kernel de Linux. Permite a los usuarios de Linux crear y administrar sistemas o contenedores de aplicaciones"

En otras palabras, es una tecnología de virtualización muy flexible, que permite crear y administrar sistemas o contenedores de aplicaciones, pero que requiere que los guests compartan el kernel con el host.

Ventajas

- Sobrecarga

  Este método de virtualización normalmente implica una sobrecarga reducida. Esto es debido a que los programas en particiones virtuales usan la interfaz de llamadas al sistema normal del sistema operativo, y no necesitan ejecutarse en una máquina virtual intermedia. Además no requieren de asistencia de hardware para operar eficientemente.

- Aislamiento

  El kernel de Linux está formado por cgroups, para aislar recursos, y espacio de nombres, para aislar completamente aplicaciones, incluyendo árboles de proceso, red, IDs de usuario y sistemas de ficheros.

- Seguridad

  Gracias al aislamiento de los contenedores, ejecutar servicios en contenedores separados asegura que las vulnerabilidades de un contenedor no afecten a los otros.

- Portabilidad

  Los contenedores pueden portarse a cualquier otro host con la misma arquitectura de procesador.

- Limites

  Debido al uso de Linux cgroups, los LXC containers pueden configurarse con limitaciones de recursos. Esto significa que se pueden dar prioridades a algunos contenedores, lo cual es muy útil cuando se trabaja con un gran número de ellos.

Desventajas

- Flexibilidad

  Este método de virtualización no permite tener un guest con un sistema operativo diferente al del host, por lo tanto solo se pueden virtualizar SOs de Linux.

Hay dos tipos de contenedores que pueden crearse:

Con privilegios

Sin privilegios

La principal diferencia entre ambos viene dada por los permisos que tienen con respecto al host, y los que el usuario tiene que tener para trabajar con ellos.

Los contenedores con privilegios usan el user id del root del host, mientras que los contenedores sin privilegios tan solo pueden usar el user id que tengan mapeado; así, resulta más seguro ejecutar aplicaciones en contenedores sin privilegios.

SIMULADOR DE REDES NS3

La función principal de NS3 es simular redes, nodos y el tráfico entre ellos. Para ello, NS3 provee abstracciones para nodos informáticos con aplicaciones para generar tráfico, dispositivos de red y canales físicos para que el tráfico se transmita.

Hay ocasiones en las que resulta interesante mezclar entidades reales y simuladas. Por ejemplo, para validar que una simulación es fiable, comprobando que los resultados son similares a los que se habrían obtenido con hardware real. En otras ocasiones se utiliza debido al coste que la red podría tener en la realidad.

Dadas las abstracciones de nodos informáticos y redes, y el estado físico (real o simulado), se obtienen las configuraciones que se muestran en la tabla 1.

Tabla 1.

	Redes		
Nodos		Real	Simulado
	Real	Tu ordenador y red	NS3-TAP
	Simulado	NS3-EMU	NS3 simulación nativa

Una de las formas en las que las simulaciones de NS3 pueden integrarse con hardware real, es utilizar una red simulada y hosts "reales" para mandar tráfico a la red. A menudo, el número de hosts reales puede ser relativamente grande y, por lo tanto, demasiado caro. El uso de virtualización para crear máquinas virtuales que ejecuten software como si fuesen hosts reales soluciona este problema.

En el caso de este proyecto, los casos que interesan son: nodos y red simulados, y cuando hay nodos reales sobre una red simulada.

Lo que se pretende hacer es crear nodos virtuales y conectarlos a través de una red simulada. Hay varias soluciones de virtualización para trabajar de esta manera. El tipo que interesa es la virtualización de hardware.

La virtualización de hardware se puede subdividir, como se ha visto antes, y también pueden diferenciarse virtualización ligera, paravirtualización, o pesada, virtualización completa.

Cuando se usa virtualización completa, se simulan todos los aspectos de la máquina host, debiendo aislarse todas las máquinas virtuales de las demás.

En paravirtualización, el entorno de la máquina virtual se crea, pero una máquina subyacente no se simula completamente. Una de las formas de hacer paravirtualización consiste en que el sistema operativo del host, hace ver que está virtualizando algunas partes de su hardware, para que él o los guests puedan usarlo.

Este tipo de virtualización es el más ligero y adecuado para ejecutar varias instancias de nodos virtuales en NS3. Ejemplos de este tipo de virtualización son Linux Containers y OpenVZ, que permitirán crear varias máquinas guest Linux sobre el host, Linux también.

MODELO DE ERRORES EN WI-FI

Debido a que en una red Wi-Fi se pueden producir errores en la transmisión de paquetes, es necesario dotar de un modelo de error al simulador NS3 para dar un mayor realismo a las simulaciones.

El modelo de error que se usa en este trabajo usa una tasa de error "FER" arbitraria, y que depende de la distancia del enlace, dentro del rango del nodo Wi-Fi. Para usar el modelo de error hay que indicar la FER que se quiere para cada enlace entre dos nodos.

La FER es un número decimal entre el 0 y el 1, que representa la de que una trama se reciba incorrectamente.

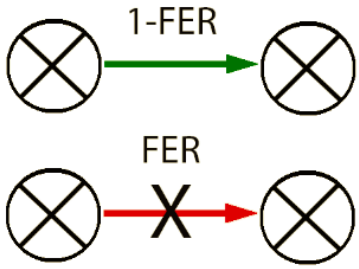


Figura 2.

En 802.11 un paquete se retransmite un número máximo de veces, por lo que la probabilidad de que un paquete no alcance el destino se puede establecer como la probabilidad de que no lo alcance en el primer intento, ni en el segundo, ni en ninguno hasta alcanzar el máximo número de retransmisiones.

$$P(PacketLoss) = FER^N$$

Por lo tanto, la probabilidad de que un paquete llegue a su destino se calcula así:

$$P(Packet\ arrives) = 1 - P(PacketLoss) = 1 - FER^N$$

Donde N es igual a cuatro retransmisiones a lo largo de este trabajo.

VIDEO STREAMING

INTRODUCCIÓN

El streaming es una manera de enviar contenido multimedia a través de una red, para que el usuario pueda reproducirlo en tiempo real. Un usario no necesita descargarse un fichero completamente para reproducirlo. En lugar de esto, el contenido es enviado en un stream continuo y es reproducido en cuanto llega al equipo del usuario. Para esto, el usuario necesita un reproductor, que es un programa especial que descomprime y envía video a la pantalla y música a los altavoces.

Uno de los mayores crecimientos en el área de internet se ha producido en el streaming de vídeo. Ha habido un tremendo interés por parte de los consumidores en el consumo de películas bajo demanda. Se estima que, en el área del entretenimiento, el streaming de video es responsable del treinta porciento del tráfico total de internet [http://www.brookings.edu/~/media/research/files/papers/2014/05/02-video-streaming/west_evolution-of-videostreaming-and-digital-content-delivery_final.pdf].

Cuando se realiza un streaming o se descarga un fichero, los archivos se dividen en paquetes que son enviados independientemente uno despues de otro y, por lo tanto, pueden viajar por rutas distintas, lo cual puede ocasionar problemas que afectan a la calidad con la que se recibe el fichero.

Los problemas que pueden aparecer son:

Pérdida de paquetes.

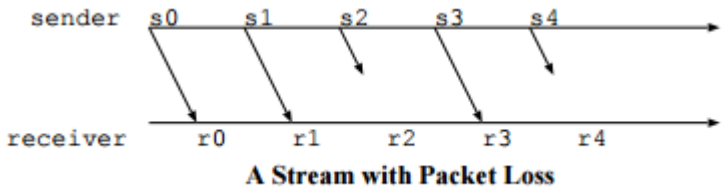


Figure 3.

Cuando se pierden paquetes en un stream, la calidad en recepción se ve afectada. En un streaming de video, esto significaría una pérdida de fotogramas o píxeles. Cuando esto pasa, el reproductor reproduce el siguiente fotograma que ha llegado correctamente, y por lo tanto se pierde información, resultando en una peor calidad de experiencia para el usuario.

- Jitter - Variacion en el tiempo de llegada de los paquetes.

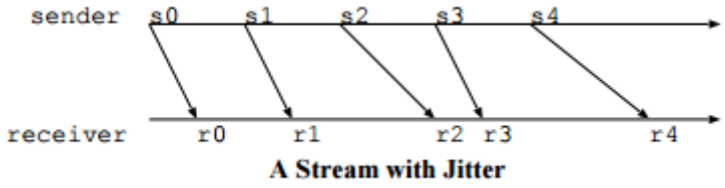


Figure 4.

En la presencia de jitter, los tiempos de llegada de los paquetes varían, lo que puede producir que algunas imágenes se congelen hasta que llega la siguiente, que podría reproducirse por un periodo de tiempo menor que las demás. En el caso de la Figura4, el tercer fotograma llega en r2, lo que hace que el usuario vea el segundo fotograma congelado en la pantalla hasta r2. En ese momento, r2 se mostrará por un breve periodo de tiempo, hasta que el cuarto fotograma llega al receptor en r3.

Algunas de las ventajas del streming de media frente a las descargas, además del hecho de que no haya que esperar a que todo el fichero se descargue para poder reproducirlo, son:

- Uso más eficiente del ancho de banda, porque solo la parte del fichero que se está transfiriendo, es la que se ve.
- Más control sobre la propiedad intelectual, ya que el fichero no es almacenado, sino que se va destruyendo a medida que se reproduce.
- Hace que sea posible recolectar estadísticas sobre lo que los usuarios están viendo y por cuanto tiempo lo ven.

El streaming de media se inicia desde un servidor y es reproducido y mostrado por una aplicación cliente llamada media player. El media player puede ser parte de un navegador, un plug-in, o una aplicación.

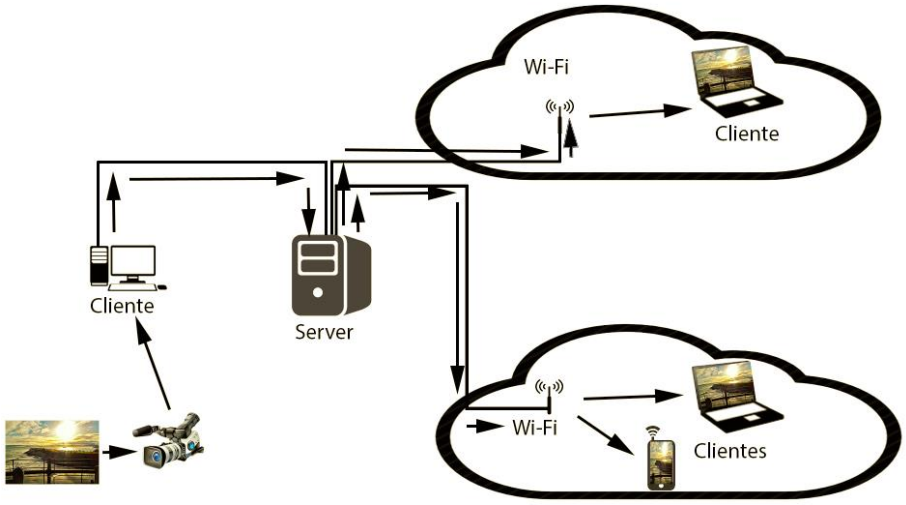


Figura 5. Esquema streaming de video.

QOS

La QoS está definida por la ITU como la totalidad de características de un servicio de telecomunicaciones, que definen su capacidad para satisfacer unas necesidades acordadas e implícitas al usuario del mismo.

Gracias a las características que pueden ser medidas en una transmisión, se puede saber cómo es el comportamiento de un servicio, lo que permite garantizar una determinada QoS. La QoS es especialmente útil para transmisiones de video y multimedia de alto ancho de banda.

La QoS se puede mejorar usando técnicas de modelado de tráfico, cómo la priorización de paquetes.

Habitualmente, la QoS en transmisiones multimedia se suele calcular utilizando medidas de retardo de los paquetes, variaciones entre el tiempo de llegada de los paquetes en el destino, pérdida de paquetes en un enlace y el ancho de banda del mismo.

Al hablar de parámetros de calidad de servicio en streaming de vídeo, se recomienda seguir las siguientes premisas:

- La tasa de pérdida de paquetes no debe ser mayor al 5%.
- El retardo para que un paquete atraviese la red no debe ser mayor de 4 ó 5 segundos, mucho menor si se trata de videoconferencia.
- El jitter de los paquetes no afecta a todos las aplicaciones por igual, en gran medida, el jitter puede afectar más o menos, dependiendo del tamaño del buffer en recepción de la aplicación.
- El ancho de banda garantizado para el streaming depende del formato de codificación del flujo, si el ancho de banda es más pequeño de lo que el streaming necesita se producirá una congestión en la red y, por lo tanto, aumentará el número de paquetes que se pierde y el retardo de los mismos.

QOE

Es una medida subjetiva acerca de la percepción de un usuario percibe un servicio, como navegación por internet o un streaming de video, entre otros.

Al ser una medida subjetiva, dos usuarios no perciben la misma QoE con el mismo servicio, por lo que se puede afirmar que la QoE depende en gran medida de la QoS, pero también del dispositivo del usuario final, de sus expectativas, de su entorno…

Esto hace que la QoE sea una medida muy útil para saber cómo se comporta un servicio, pero a la vez una medida difícil de cuantificar.

DISEÑO DE LA SOLUCIÓN

INTRODUCCIÓN

Para desarrollar este proyecto se pretende ejecutar aplicaciones en contenedores de Linux, que estén aislados los unos de los otros y tengan un espacio de direcciones diferente. Además, hay que conectarlos a nodos del simulador NS3, de tal manera que los contenedores a través del NS3.

Una vez conseguido esto, y para obtener estadísticas de funcionamiento del NS3 cuando tiene conectados contenedores de aplicaciones generando tráfico real, hay que utilizar algún servicio que permita enviar y recibir tráfico, y visualizar que el sistema está funcionando. La aplicación VLC, que nos permite enviar flujos de vídeo desde un contenedor que actuará como servidor, hasta otro contenedor o contenedores, que serán los clientes, y reproducirán el streaming de vídeo que les llegue. De esta manera se podrá percibir de manera directa la calidad del servicio.

Para recoger las estadísticas de funcionamiento del NS3 hay que modificar sus clases y registrar los datos necesarios en ficheros, para poder ver parámetros de calidad de servicio (QoS), los tiempos de salida y llegada de los paquetes de/a los dispositivos TapBridge, su longitud, el throughput de transmisión y recepción, y el jitter del propio NS3 (diferencia entre tiempo de sistema y tiempo de simulación).

CONTENEDORES LXC

Hay dos tipos de contenedores de Linux, los contenedores con privilegios, y los que no los tienen.

Los contenedores sin privilegios presentan algunos beneficios, como la seguridad. Este tipo de contenedores son más seguros que los otros ya que se enmascara la identidad de usuario (UserID de ahora en adelante) del host, algo deseable cuando se utilizan contenedores para ejecutar un servidor web, por ejemplo. Los contenedores con privilegios no ofrecen este tipo de seguridad porque el usuario root del contenedor, puede tener dicho acceso root en el host.

En contenedores sin privilegios, incluso aunque un usuario puede aparecer como root en el contenedor, tiene un UserID diferente mapeado en el host.

Esto tiene sus desventajas también, ya que si un usuario quiere usar un contenedor sin privilegios, necesitará asegurarse de que habrá disponibles suficientes subuids y subgids, y necesita permitir además que los usuarios puedan conectar contenedores a bridges.

El uso de contenedores con privilegios es más sencillo cuando se tiene acceso root al host, y ofrece más funcionalidades. La conexión de los contenedores con privilegios a los bridges es mucho más sencilla. Además, solo se pueden añadir librerías del host al contenedor si este tiene privilegios, permitiendo la ejecución de aplicaciones instaladas en el host desde el contenedor.

Considerando todo esto, y sabiendo el uso que van a tener los contenedores en este proyecto, el tipo de contenedor más apropiado es el que tiene privilegios.

CONFIGURACIÓN DE LXC

Lo primero que hay que hacer antes de crear contenedores LXC es configurar el host para que tenga todas las herramientas necesarias para utilizarlos.

Para configurar el entorno de LXC es necesario utilizar el script configLXC.sh.

El script completo se puede encontrar en el apéndice. A continuación se explican los comandos más relevantes en la instalación, configuración y funcionamiento de LXC containers.

```
sudo apt-get install lxc uidmap
```

LXC es un conjunto de herramientas, plantillas y librerías para el funcionamiento y contención de los contenedores.

Por su parte, Uidmap es un programa que ofrece a usuarios sin privilegios utilizar mapas uid y gid.

El comando siguiente es para que el NS3 pueda usar los contenedores. El paquete Bridge-utils contiene una utilidad necesaria para crear y administrar bridges.

Además, Uml-utilities es un puerto del Kernel de Linux hacia su propia interfaz de llamadas al sistema. Ofrece una especie de máquina virtual que ejecuta el sistema operativo como si fuese un proceso de usuario bajo otro Kernel de Linux.

```
sudo apt-get install bridge-utils uml-utilities
```

Usando usermod y chmod lo que se hace es asignar al propio equipo, una serie de uids y gids, para que en un futuro se puedan usar los contenedores como contenedores sin privilegios.

```
sudo usermod --add-subuids 100000-165536 $USER
sudo usermod --add-subgids 100000-165536 $USER
sudo chmod +x $HOME
```

Para crear los contenedores se necesita un fichero de configuración, y que se crea en el directorio `~/.config/lxc/`

```
mkdir -p ~/.config/lxc/
cat > ~/.config/lxc/default.conf << EOF

lxc.network.type = veth
lxc.network.link = lxcbr0
lxc.network.flags = up
lxc.network.hwaddr = 00:16:3e:xx:xx:xx
lxc.id_map = u 0 100000 65536
lxc.id_map = g 0 100000 65536
EOF
```

Lxc.network.type especifica la clase de virtualización de red se va a usar para el contenedor. La opción VETH significa que un dispositivo virtual Ethernet con un entorno asignado al contenedor y el otro conectado al bridge especificado en lxc.network.link.

Lxc.network.flags = up, activa la interfaz de red del contenedor.

Lxc.network.hwaddr normalmente es aleatorio, pero en algunos casos puede ser necesario fijarlo.

Lxc.id_map es usado para ejecutar el contenedor en un espacio de usuario privado, mapeando user id y group id.

CREACIÓN DEL PRIMER CONTENEDOR

Después de haber configurado LXC, todo está preparado para crear el primer contenedor.

Para ello, se ha creado el script firstContainer.sh, que lo instancia y le instala VLC e IPERF, dos aplicaciones que se van a usar durante este trabajo. Además de instalar estas aplicaciones, se necesita una carpeta compartida entre el host y los guests (contenedores), en la que ubicará el video que los contenedores transmiten mediante VLC.

El objetivo de este contenedor no es otro que servir como base para crear otros, de manera que el proceso sea lo más dinámico posible. Mientras que la creación de un contenedor lleva alrededor de 25 minutos, clonar uno y cambiarle las direcciones MAC e IP, añadiendo la dirección IP del Gateway, es un proceso que dura 1 minuto y 20 segundos, con una ganancia de aproximadamente 19 veces.

El script completo se puede encontrar en el apéndice. A continuación se explican los comandos más relevantes en la creación y configuración del primer contenedor.

```
lxc-create -t ubuntu -n $name -- -a i386
```

Creación del contenedor con el template Ubuntu y la arquitectura i386 (32 bis).

```
mkdir -p /home/juan/project/shared
chmod 7777 /home/juan/project/shared
```

Mkdir se usa para crear la carpeta que el host va a compartir con el contenedor, y chmod configura los permisos para que pueda acceder a ella.

```
lxc.mount.entry = /dev/dri dev/dri none bind,optional,create=dir
lxc.mount.entry = /dev/snd dev/snd none bind,optional,create=dir
lxc.mount.entry = /tmp/.X11-unix tmp/.X11-unix none bind,optional,create=dir
lxc.mount.entry = /dev/video0 dev/video0 none bind,optional,create=file
lxc.mount.entry = /home/juan/project/shared root none bind.ro 0.0
```

Todo esto se escribe en el fichero de configuración del contenedor y sirve para integrarlo con el escritorio del host, añadiendo las librerías y carpetas para que el contenedor pueda usar los archivos y aplicaciones compartidos. Además se monta la carpeta compartida en la carpeta root del contenedor, así se puede acceder a ella desde el contenedor.

```
cp /home/juan/project/setup-pulse.sh /var/lib/lxc/$name/setup-pulse.sh
chmod +x /var/lib/lxc/$name/setup-pulse.sh
```

El script setup-pulse.sh indica a pulseaudio en el host que tiene que unir /home/Ubuntu/.pulse_socket al contenedor, otorgandole permisos de ejecución.

A continuación se actualiza el contenedor y se instalan las aplicaciones VLC e IPERF.

```
lxc-start -n $name -d
lxc-attach -n $name -- umount /tmp/.X11-unix
lxc-attach -n $name -- apt-get update
lxc-attach -n $name -- apt-get dist-upgrade -y
lxc-attach -n $name -- apt-get install wget ubuntu-artwork dmz-cursor-theme
ca-certificates pulseaudio –y
##APPS
lxc-attach -n $name -- dpkg -i /root/vlc.deb
lxc-attach -n $name -- apt-get -f install -y
lxc-attach -n $name -- apt-get install iperf
lxc-attach -n $name -- sudo -u ubuntu mkdir -p /home/ubuntu/.pulse/
echo "disable-shm=yes" | lxc-attach -n $name -- sudo -u ubuntu tee
/home/ubuntu/.pulse/client.conf
```

El script setup-pulse.sh se puede encontrar en el apéndice tal y como se ha utilizado.

CREACIÓN DE CONTENEDORES

Una vez se tienen el primer contenedor, se puede ir clonando para obtener más contenedores. El script creado para ello se llama createContainer.sh, y clona el primer contenedor con toda su configuración. Una vez clonado, se modifica la configuración del contenedor para cumplir con los requisitos del usuario.

El script tiene tres argumentos de entrada:

- $1 nombre del contenedor
- $2 ip
- $3 gateway

El script completo se puede encontrar en el apéndice. A continuación se explican los comandos más relevantes a la clonación y configuración individual de los contenedores.

```
lxc-clone -o firstcontainer -n $name
```

Lxc-clone es el comando que se usa para clonar contenedores. En este caso clona el contenedor firstcontainer y crea el contenedor $name, donde $name es una variable con el nombre que se le haya pasado al script.

```
sed -i '/lxc.network.ipv4/d' /var/lib/lxc/$name/config
sed -i '/lxc.network.ipv4.gateway/d' /var/lib/lxc/$name/config
echo "lxc.network.ipv4 = $ip/24" >>/var/lib/lxc/$name/config
echo "lxc.network.ipv4.gateway = $gwip">>/var/lib/lxc/$name/config
```

Con estas líneas de código elimina la configuración con la dirección IP del contenedor y del Gateway, para así introducir nuevas direcciones, a partir de los argumentos que se le han pasado al script.

Ahora se crean los dispositivos Tap que el NS3 va a usar para recibir/transmitir paquetes dentro y fuera de la red simulada.

```
sudo tunctl -t tap-$name
sudo ifconfig tap-$name 0.0.0.0 promisc up
sudo brctl addif lxcbr0 tap-$name
```

Por último, los taps se añaden al bridge lxcbr0 para hacer el proceso más rápido, ya que no es necesario crear un bridge para cada TAP.

NOTAS

Algo que es importante destacar es que una vez que los contenedores están creados y se ha modificado su dirección IP y Gateway, ya no dispondrán de una conexión a internet, por lo que es importante que tengan instalado todos los componentes necesarios antes de que esto pase.

Se modifica la dirección del Gateway para que los contenedores puedan mandar tráfico dentro de la red del NS3, en lugar utilizar el bridge LXCBR0 al que están conectados.

El motivo por el que todos los contenedores están conectados al bridge LXCBR0 es para agilizar el proceso de creación/configuración e inicialización de los contenedores.

APLICACIONES

El motivo por el que se han creado los contenedores de Linux es para así poder ejecutar aplicaciones reales, que estén en redes diferentes y completamente aisladas unas de otras, necesitando emplear el simulador de redes NS3 para interconectarlas.

Para poder observar el efecto real de las comunicaciones sobre las aplicaciones se utilizará VLC. Con ella se realizan streamings de video, utilizando un contenedor como servidor, y otros como clientes del flujo.

VLC EN LOS CONTENEDORES

Ya hemos visto cómo se crean los contenedores, pero aún no se ha explicado que hay que hacer para que las aplicaciones, concretamente el VLC, funcionen correctamente y con la interfaz gráfica necesaria en el contenedor. A continuación se detalla este proceso.

El script firstcontainer.sh que se ha visto antes, crea el primer contenedor, con toda la configuración necesaria para poder clonarlo y que el resto tengan todos los elementos que necesitan. Pero para utilizar el VLC y hacer streaming de video, hay que llevar a cabo una serie de pasos adicionales para ejecutar una aplicación en un contenedor.

Como ejecutar una aplicación en un contenedor no es en todos los casos tan sencillo como en un host, el script execute.sh se encarga de mandar el comando adecuado al contenedor para que este llame a pulse-setup.sh, y ejecute VLC como servidor, con un video que enviar, unos parámetros de configuración de

streaming y el destino. El script encargado de ejecutar el VLC en el lado cliente, será cli-execute.sh. Utilizan los comandos que se recogen a continuación.

```
lxc-attach --clear-env -n $CONTAINER -- sudo -u ubuntu -i env
QT_X11_NO_MITSHM=1 DISPLAY=$DISPLAY PULSE_SERVER=$PULSE_SOCKET vlc --
configuración
```

Es muy importante fijarse en el siguiente parámetro.

```
QT_X11_NO_MITSHM=1
```

Evita que se intente usar memoria compartida, ya que sino no se podría obtener interfaz gráfica apropiada para el usuario.

Ambos scripts pueden consultarse en el apéndice, tal y cómo se han usado en las primeras simulaciones llevadas a cabo.

STREAMING VLC

Para entender cómo funciona el streaming de VLC hay que echar un vistazo a la línea de comando del servidor en primer lugar. Para iniciar el streaming hay que definir el vídeo que se quiere transmitir;

File:///root/video.avi

Después hay que definir algunos parámetros para la codificación del video, de manera que se pueda enviar el streaming de una manera más eficiente. El código que se encarga de esto es:

```
sout="#transcode{vcodec=h264,fps=20,scale=Auto,acodec=mpga,ab=128,channels=2,sampl
erate=44100}:udp{dst=$IPDST:1234}"
```

La parte “udp{dst=$IPDST:1234}” sirve para especificar el protocolo de transmisión (UDP), la dirección IP del destino del flujo y el puerto al que se envía.

El cliente tiene que recibir el flujo, para lo que se configura el VLC para abrir un network stream en la dirección del mismo contenedor, ya que es hacia donde el servidor lo envía.

Comando:

```
vlc udp://@20.0.$1.2:1234
```

$1 es un parámetro de entrada al script, usado para identificar el contenedor en el que se abre el flujo, y también para fijar la IP de dicho contenedor. El número que aparece después de la dirección IP pone “:1234”, es el puerto para recibir el vídeo.

ESQUEMA FUNCIONAMIENTO

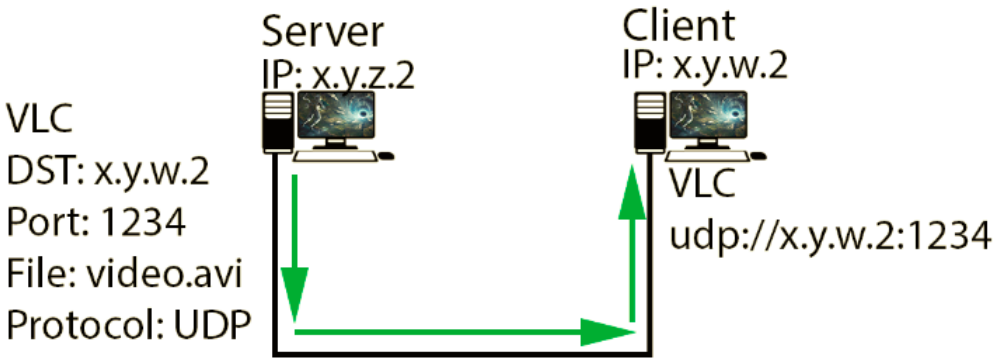


Figura 6. Esquema streaming VLC

NS3

INTRODUCCIÓN

El NS3 es un simulador de redes, desarrollado con el objetivo de proveer con una herramienta de código abierto y que lo permitiese crecer, para fines de investigación sobre redes y educación. Se usa, principalmente, debido a que permite estudiar el comportamiento de un sistema en un entorno altamente controlado y reproducible.

Además, es especialmente interesante en este proyecto debido a su capacidad para emular dispositivos en nodos virtuales y, así, comprobar el comportamiento de aplicaciones reales dentro del simulador.

En la sección a continuación se dan a conocer las posibilidades que existen.

TAP-BRIDGE

INTRODUCCIÓN

La conexión entre el NS3 y los contenedores es posible gracias al modelo TapBridge implementado en el simulador, que conecta las entradas y salidas de un dispositivo de red del NS3 con las entradas y salidas de un dispositivo de red del sistema operativo Linux llamado tap. Un tap emula un dispositivo de la capa de enlace y opera con paquetes de dicho nivel, como tramas Ethernet. Este dispositivo se usa para crear bridges.

El resultado de esta conexión es la integración de un host "real" o, en el caso de este trabajo los contenedores LXC, que soporta dispositivos tap, con las simulaciones sobre NS3.

MODELOS

Hay básicamente tres modelos operacionales de este dispositivo. El funcionamiento es muy similar, diferenciándose en la manera en la que se crean y se configuran. Además, difieren en los dispositivos que pueden estar en cada lado del bridge. Los modelos son:

- ConfigureLocal

  En este caso es el TapBridge el que tiene que crear y configurar los taps.
  La información sobre la configuración se obtiene de un dispositivo en el NS3, creando uno automáticamente. Parece, por tanto, que el host está conectado directamente a una red simulada en el NS3.
  Se puede decir, por tanto, que el dispositivo de red del NS3 reemplaza el TAP en el host Linux, ya que éste tiene las direcciones IP y MAC del dispositivo del NS3 simulado. La conexión realizada entre el dispositivo de red del NS3 y el tap, se realiza a través de un enlace denominado "IPC", que actúa como un bridge convencional, aunque los dispositivos conectados tienen la misma dirección IP y MAC.

- UseLocal

  El usuario proporciona una configuración y el TapBridge se adapta a ella.
  La dirección MAC preconfigurada del dispositivo tap (MAC X) no coincide con la del dispositivo de red del NS3 (MAC Y). Esto se hace dado que algunos dispositivos de red (especialmente los inalámbricos) no soportan el comando SendFrom(). Esto significa que el tráfico saliente del tap en el host irá al dispositivo de red del NS3, que cambiará la dirección MAC de X a Y, y viceversa. Por este motivo, solo puede haber un dispositivo tap conectado al bridge IPC.

- UseBridge

  De nuevo el usuario proporciona una configuración, y el TapBridge se adapta a ella.
  Se crea un TapBridge con la configuración de cada dispositivo tap. Una vez el nombre del tap se asigna al TapBridge, usando el atributo "DeviceName", éste extiende el bridge del sistema operativo del host, para unirlo con el nodo de red del NS3.
  Este modo permite que haya más de un dispositivo de red de Linux en el lado del host (no NS3). Así, el dispositivo de red del NS3 tiene que ser capaz de funcionar con varias direcciones, lo que se consigue con la función SendFrom(), que no es soportada por todos los nodos, por ejemplo los dispositivos Wi-Fi.

CONCLUSIÓN

En conclusión, la mejor opción dadas las necesidades de nuestro trabajo es el modo UseBridge. La opción UseLocal podría ser adecuada, pero solo en el caso de tener un único contenedor. Haciendo pruebas se ha comprobado que la opción UseLocal no funciona correctamente cuando se intenta comunicar varios contenedores LXC en un escenario de red sencillo.

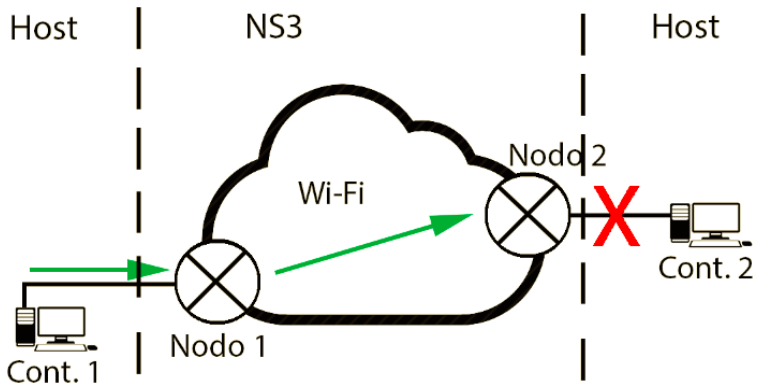


Figura 7. Esquema funcionamiento UseLocal

El problema que se manifiesta, pese a la capacidad del contenedor de hacer pasar los paquetes a través del tap y posteriormente atravesar la red simulada con el NS3, es que los paquetes alcanzaban el nodo "ghost" que contenía el otro contenedor; posteriormente, una vez los paquetes llegaban a ese punto, el dispositivo de red del NS3 no sabía cómo actuar para hacer llegar los paquetes a su destino fuera del NS3.

Gracias al modo UseBridge, que implementa la función SendFrom(), y haciendo algunos cambios en el escenario de red, los contenedores LXC se pueden comunicar satisfactoriamente.

La modificación consiste en crear un segmento de red CSMA/CD con dos nodos, actuando uno de ellos como Gateway para el contenedor. El otro es el nodo Ghost. Así se soluciona la limitación de los nodos Wireless de no soportar la función SendFrom().

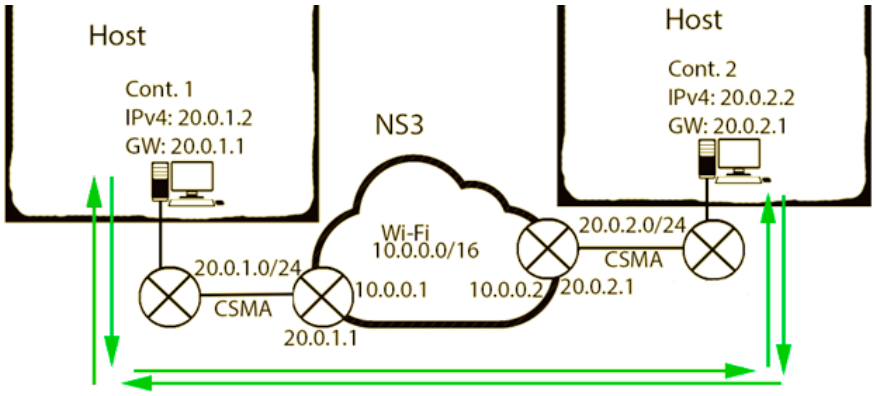


Figura 8. Esquema UseBridge TapBridge

Con esta configuración, el contenedor es capaz de enviar paquetes a la red CSMA/CD del NS3, para después ser enviados a la red Wi-Fi mediante el Gateway. Los paquetes, una vez atraviesan la red Wi-Fi y llegan al otro Gateway, pasan a la otra red CSMA/CD, en la que está el nodo Ghost, el cual redirige el tráfico hacia el tap del host (fuera del NS3), para que finalmente el contenedor pueda recibir el tráfico. El mecanismo funciona igual en sentido contrario.

REAL TIME

Para integrar NS3 con sistemas reales y transmitir/recibir paquetes, hace falta un planificador de tiempo real para sincronizar la simulación con el reloj de la máquina. Este planificador se llama "RealTime scheduler".

El objetivo del RealTime scheduler es conseguir que el reloj de la simulación vaya a la misma velocidad que el del ordenador, manteniendo el sincronismo entre ambos.

Cuando no se usa el RealTime scheduler, el simulador incrementa el tiempo de simulación hasta el siguiente evento programado. Durante la ejecución del evento, el tiempo de simulación se para. Con el RealTime scheduler, el comportamiento es simular durante la ejecución de un evento, pero entre ellos, el simulador intenta mantener el reloj de simulación sincronizado con el del ordenador.

Cuando un evento se termina de ejecutar, el RealTime scheduler chequea el tiempo de simulación del siguiente evento y lo compara con el del ordenador. Si es mayor, el simulador permanece a la espera hasta que alcanza el tiempo real, ejecutando el siguiente evento.

Puede suceder que, debido al procesado de los eventos, el simulador no pueda mantenerse sincronizado con el tiempo real. En este caso, el usuario puede configurar la respuesta que se produce. El parámetro ns3::RealTimeSimulatorImpl::SynchronizationMode puede tomar dos valores. "BestEffort", en el que el simulador intenta sincronizarse con el tiempo real ejecutando eventos hasta que alcanza un punto en el que el siguiente evento está programado en el tiempo real futuro, o la simulación finaliza. Así, entonces, la simulación puede llevar más tiempo que el reloj del ordenador. La otra opción es "HardLimit", en la que la simulación se detiene al superar un umbral de tolerancia.

Muchas de las veces en las que se llevaba a cabo una simulación con un valor de HardLimit inferior al que viene por defecto (100ms), esta se detenía. Por lo tanto, y para comprender mejor la influencia de los valores de HardLimit, se ha modificado la clase RealTimeSimulatorImpl para, en lugar de detener la simulación, recoger la evolución temporal de la diferencia entre el tiempo de reloj en el host y en la simulación.

Implementación del RealTime en el simulador NS3:

```
GlobalValue::Bind("SimulatorImplementationType",
StringValue("ns3::RealtimeSimulatorImpl"));

GlobalValue::Bind("ChecksumEnabled",                    BooleanValue(true));
Config::SetDefault("ns3::RealtimeSimulatorImpl::SynchronizationMode",
StringValue("HardLimit"));

Config::SetDefault("ns3::RealtimeSimulatorImpl::HardLimit",
TimeValue(Time(timevar)));
```

Siendo timevar una variable con el HardLimit que se quiera fijar, y que normalmente actuaría como umbral para parar la simulación.

INTERCONEXIÓN

Antes de ver como se realiza la interconexión de todos los elementos, es preciso detallar parte de la configuración de uno de los contenedores, que se encuentra en /var/lib/lxc/cont1/config

ENTENDIENDO LA CONFIGURACIÓN

De la configuración el parámetro más relevante para realizar la interconexión es:

- lxc.utsname = cont1

  Este es el nombre del contenedor, información necesaria para operar con él. Además, el nombre del dispositivo tap en el host, tendrá asignado como nombre el nombre de "tap-nombre del contenedor", en este caso tap-cont1. El nombre del tap se utiliza para indicar al ns3 qué tap tiene que usar para coger los paquetes que vienen del host hacia la red simulada.

- lxc.network.link = lxcbr0

  lxcbr0 es el nombre del bridge al que se conecta el contenedor. Además, este bridge tiene añadidos los taps que introducirán/cogerán paquetes en/del NS3.

- lxc.network.hwaddr = 00:16:3e:6c:53:c1

  Dirección MAC del contenedor, útil no solo para que el NS3 sepa a qué entidad dirigir los paquetes, sino que también para filtrar los paquetes que interesan a la hora de recoger estadísticas para caracterizar el comportamiento del NS3.

- lxc.network.ipv4 = 20.0.1.2/24

  Dirección IPv4 del contenedor. Será la misma que la del nodo ghost del NS3 vinculado al contenedor en cuestión, que adquiere al usar la clase TapBridge con el modo UseBridge.

- lxc.network.ipv4.gateway = 20.0.1.1

  Dirección a la que el contenedor envía los paquetes en el simulador NS3. Esta es la dirección del nodo que actúa como Gateway para el contendor, y que está situado en la intersección del segmento CSMA/CD con la red Wi-Fi.

CREACIÓN/CONFIGURACIÓN TAPS

A continuación se explica el proceso de creación de dispositivos tap en el host y como se añaden al bridge. Para ello se utiliza el script attachTap.sh.

El script completo se puede encontrar en el apéndice. A continuación se explican los comandos más relevantes a la creación y puesta a punto del primer contenedor.

El número de contenedores que hay en la carpeta /var/lib/lxc se conocen gracias a unas líneas de código, y a partir de ahí se crean y configuran los dispositivos Tap.

```
sudo tunctl -t tap-$var
sudo ifconfig tap-$var 0.0.0.0 promisc up
sudo brctl addif lxcbr0 tap-$var
```

Indicar que la variable $var, puede sustituirse por el nombre del contenedor, por ejemplo cont1, puesto que es una variable que precisamente recoge este valor contenedor de la carpeta en la que están guardados los contenedores creados.

- sudo tunctl -t tap-$var
  Tunctl se usa para crear y configurar interfaces tap. Tunctl –t especifica el nombre de la interfaz.
- sudo ifconfig tap-$var 0.0.0.0 promisc up
  Ifconfig es el comando para configurar interfaces. Este paso es necesario antes de añadir los tap al bridge. Con él también se activan los tap.
- sudo brctl addif lxcbr0 tap-$var
  Brctl se usa para configurar, mantener y monitorizar la configuración del bridge Ethernet en el kernel de Linux. Con este comando se añade el interfaz tap-$var al bridge lxcbr0.

EJECUTAR CONTENEDORES

Una vez que se han creado, configurado y activado los taps, se pueden iniciar los contenedores. Para ello se puede usar el script startcontbg.sh, que tiene como parámetro de entrada el número de contenedores que se quieren iniciar en segundo plano.

El script completo se puede encontrar en el apéndice. A continuación se explica el comando más importante.

```
lxc-start -n cont$i -d
```

Lxc-start inicia el contenedor cont$i, en el que $i es un número de 1 a X, siendo X el parámetro de entrada que se le ha pasado al script. La opción –d determina que el contenedor se ejecute en segundo plano, lo que permite lanzar comandos al contenedor de una manera dinámica, posibilitando a su vez realizar simulaciones con scripts, sin que sean necesarias interrupciones en las que el usuario tenga que escribir los comandos manualmente. En definitiva, la ejecución en segundo plano permite automatizar las simulaciones.

NS3

A continuación se configuran los TapBridge del escenario para que reciban y transmitan paquetes del host a la red simulada en NS3/hacia los taps del host.

TAPBRIDGE

Para eso, como se ha visto antes, se usa la clase TapBridge con el modo UseBridge, tal y como se ve a continuación.

```
TapBridgeHelper tapBridge;
tapBridge.SetAttribute("Mode", StringValue("UseBridge"));
tapBridge.SetAttribute("DeviceName", StringValue("tap-cont1"));
tapBridge.Install(nleft.Get(1), dleft.Get(1));
```

SEGMENTO CSMA/CD

Para crear el segmento de red CSMA/CD necesario para que el NS3 y los contenedores sean completamente compatibles, se siguen los siguientes pasos.

Parte 1:

Crear un contenedor de nodos, con los Gateway que va a haber en toda la red, que coincidirá, obviamente, con el número de contenedores que se conectan a la red.

```
NodeContainer nodeGW;
nodeGW.Create(num_contenedores);
```

Parte 2:

Crear un contenedor de nodos en la parte del segmento de CSMA/CD. Se genera un nodo ghost y se añade un nodo Gateway.

Ejemplo:

```
NodeContainer nleft;
nleft.Add(nodeGW.Get(0));
nleft.Create(1);
InternetStackHelper sleft;
sleft.Install(nleft.Get(0));
CSMA/CDHelper cleft;
cleft.SetChannelAttribute("DataRate", DataRateValue(DataRate(20000000)));
cleft.SetChannelAttribute("Delay", TimeValue(MilliSeconds(0)));
NetDeviceContainer dleft;
dleft = cleft.Install(nleft);
Ipv4AddressHelper aleft;
aleft.SetBase("20.0.1.0","255.255.255.0");
Ipv4InterfaceContainer ileft;
ileft = aleft.Assign(dleft.Get(0));
```

Como se puede observar, la dirección IP que se asigna es la del Gateway, ya que el nodo ghost usa el modo UseBridge, que adquiere la configuración que le dice el TAP.

Después de crear los segmentos de red CSMA/CD necesarios se instancia la red Wi-Fi, en la que se añaden los Gateway de los diferentes segmentos.

Se tiene que mencionar la excepción que se da cuando se usa un modelo de error, ya que hay que instanciar los nodos que se van a usar únicamente en la red Wi-Fi justo después de crear los Gateway CSMA/CD. Esto se debe a que el modelo de error está configurado de tal manera que empieza a procesar los nodos según el orden en que fueron instanciados en el escenario.

WI-FI

Al crear la red Wi-Fi hay que añadir los Gateway de los segmentos de red CSMA/CD para que los contenedores puedan comunicarse.

Al ser una red Wi-Fi, hay que especificar además la posición de los nodos, ya que podrían moverse.

En el NS3 se puede configurar la red Wi-Fi utilizando diferentes versiones del IEEE 802.11. En las simulaciones se usa 802.11b. Además, también se pueden añadir errores al canal, gracias a un modelo de errores que se ha integrado modificando las clases yans-wifi-phy y error-model. Para ello hay que añadir RangePropagationRangeModel, y definir la FER (Frame Error Rate) entre cada pareja de nodos, pudiendo fijar una FER por defecto para todos los enlaces.

ENCAMINAMIENTO

Entre las alternativas para emplear encaminamiento sobre redes multi-salto, la más inmediata es utilizar los protocolos implementados ya en el NS3, como AODV y OSLR.

Se ha optado por utilizar AODV, Ad-hoc On demand Distance Vector. Este protocolo de encaminamiento reactivo realiza el descubrimiento de rutas y su posterior mantenimiento. Presenta una operación salto a salto, empleando números de secuencia, y beacons periódicos. Usa números de secuencia para prevenir la formación de ciclos en las rutas. Ofrece una respuesta rápida cuando la topología de la red ad-hoc es cambiante.

Debido a lo anterior, se selecciona AODV para llevar a cabo las simulaciones, aunque se produce un conflicto al construir las rutas cuando se añaden los Gateway de los contenedores (que tienen una interfaz Wi-Fi y otra CSMA/CD). Así se puede ver en una captura realizada con Wireshark, en la que aparecen constantemente paquetes de AODV con el mensaje “error de ruta” en ellos.

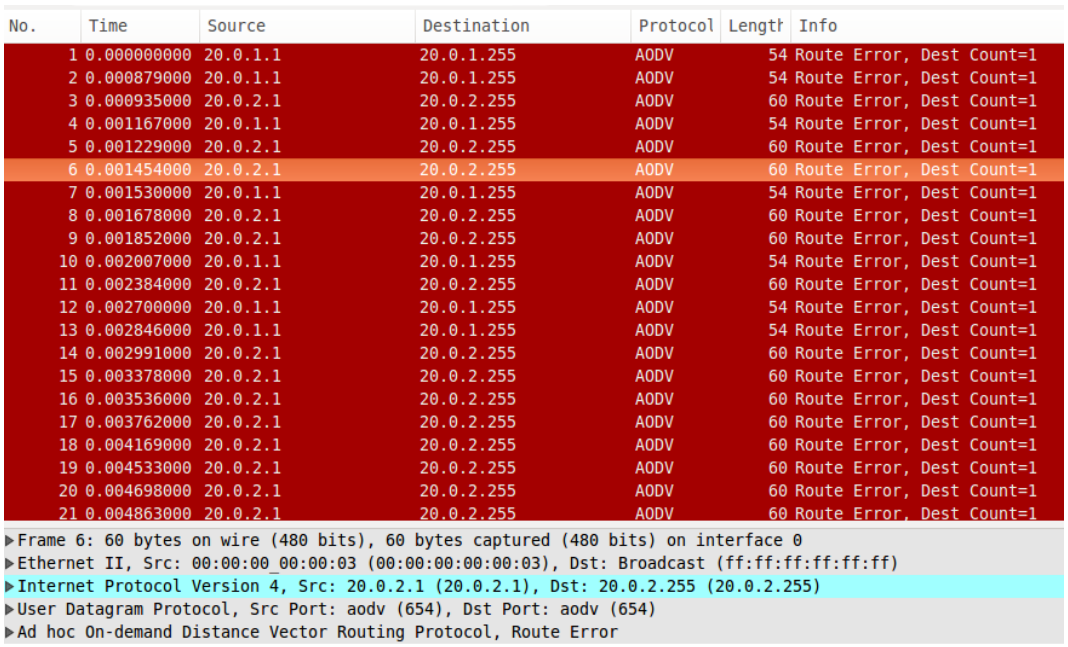


Figura 9. Captura Wireshark AODV

Ya que no es posible utilizar contenedores y el protocolo AODV en la misma red, la alternativa que se utilizará es la configuración manual de tablas de rutas.

Una ventaja que presenta esta alternativa, es que se conoce el camino que los paquetes en todo momento, y así se facilita trabajar con errores y topologías de red de manera sistemática, aunque se pierde el realismo de la operación tradicional AODV.

Para asignar las rutas manualmente se tienen que dar los pasos descritos seguidamente.

STATICROUTING

Se pretende definir la tabla de rutas del nodo 0, de manera que los destinos pueden ser bien la dirección “20.0.2.2”, que está en el otro lado de la red Wi-Fi, o bien la dirección “20.0.1.2”, que está en el extremo de la red CSMA/CD. Como el nodo 0 tiene una interfaz en ambas redes, se identificará la interfaz Wi-Fi con el número 2, y la interfaz CSMA/CD con el 1.

Con el siguiente comando se obtiene una referencia al objeto IPv4 del nodo, donde se añaden las rutas posteriormente.

```
Ptr<Ipv4> ipv4A = nodes.Get(0)->GetObject<Ipv4> ();
```

Con esto, ya se puede pasar a definir las rutas.

```
Ipv4StaticRoutingHelper staticRoutingHelper;
Ptr<Ipv4StaticRouting> staticRoutingA = staticRoutingHelper.GetStaticRouting(ipv4A);
staticRoutingA->AddHostRouteTo(Ipv4Address("20.0.2.2"),Ipv4Address("10.0.0.3"),2);
staticRoutingA->AddHostRouteTo(Ipv4Address("20.0.1.2"),Ipv4Address("20.0.1.2"),1);
```

La función AddHostRouteTo(Destination,NextHop),IfaceNumber) recibe los argumentos enumerados a continuación.

- Destination: Es el nodo/contenedor al que se pretende llegar.
- NextHop: Es el siguiente nodo al que hay que reenviar los paquetes para alcanzar el destino.
- IfaceNumber: Es el identificador de la interfaz de red por la que hay que transmitir el paquete hacia el siguiente salto. El orden de creación de los interfaces es el que se usa para numerarlos. Así, si un nodo tiene dos interfaces de red, CSMA/CD y Wi-Fi por ejemplo, y se ha creado primero el correspondiente a la red CSMA/CD, su IfaceNumber será 1, mientras que el de la interfaz Wi-Fi será 2.

ESTADÍSTICAS

A pesar de que es posible observar variación de la QoE en la reproducción del video que se recibe en streaming, es necesario observar la evolución de otros parámetros que determinan la QoS y el rendimiento de la conexión, así como el comportamiento del NS3.

REAL TIME JITTER

Diferencia entre el tiempo de reloj del host y el de simulación.

MOTIVO

El motivo para analizar la evolución de este jitter es conocer en qué casos el comportamiento del NS3, con tráfico y tiempo reales, es aceptable. En ocasiones se verá como este jitter va aumentando a medida que el experimento avanza, hasta unos niveles que dejan de ser aceptables para trabajar con tráfico real. Sin embargo en otras medidas pasa lo contrario, y tras una fase de inicialización, o una situación puntual en la red, este jitter se recupera a valores aceptables.

La interpretación de estos valores permitirá conocer mejor el rendimiento del NS3 con contenedores y tráfico de aplicaciones reales.

MODIFICACIÓN CLASES

Para obtener estas estadísticas fueron necesarias algunas modificaciones en la clase realtime-simulator-impl.cc. Se elimina la parte de código que detiene la simulación cuando se excede el Hard-Limit establecido y se añade una función que determina el valor máximo de este jitter cada 0.1 segundos, y que llama a una función del escenario, que recibirá este valor (tiempo real – tiempo simulación en dicho intervalo), y lo escribirá en un fichero, para su posterior interpretación y representación.

REALTIME-SIMULATOR-IMPL.CC

Solo se muestra la función que ha sido modificada en mayor medida; hay otras modificaciones en la clase, pero simplemente llevan a cabo la inicialización de variables, y permiten acceso a funciones de la clase, para que se puedan llamar desde el escenario.

La clase completa se puede encontrar en el apéndice, tal y como se ha utilizado en las simulaciones llevadas a cabo.

A continuación se muestra la modificación que se encarga de recoger la diferencia entre los tiempos de reloj de host y de simulación.

```
if (m_synchronizationMode == SYNC_HARD_LIMIT) {
uint64_t tsFinal = m_synchronizer->GetCurrentRealtime();
uint64_t tsJitter;
if (tsFinal >= m_currentTs) {
      tsJitter = tsFinal - m_currentTs;
} else {
      tsJitter = m_currentTs - tsFinal;
}
if (tsJitter > maxJitter) {
      maxJitter = tsJitter;
}
if ((tsFinal-lasttime)>= static_cast<uint64_t>(m_tracestep.GetTimeStep ())){
std::stringstream ss;
ss << tsFinal/1e9 << " " << (double(maxJitter) / 1e6); //maxJitter-ms //and
tsFinal-Sec
m_result(ss.str());
ss.str("");
ss.clear();
lasttime=tsFinal;
maxJitter=0;
}
}
```

Como se puede ver en el código, se va registrando el valor máximo de tsjitter tras cada intervalo de monitorización, delimitado por m_tracestep.

La parte del código que detenía la simulación cuando se alcanzaba una diferencia entre tiempo real y tiempo de simulación igual al hard limit, es esta:

```
if (tsJitter > static_cast<uint64_t>(m_hardLimit.GetTimeStep ()))
{
    NS_FATAL_ERROR ("RealtimeSimulatorImpl::ProcessOneEvent (): "
```

```
    "Hard real-time limit exceeded (jitter = " << tsJitter << ")");
}
```

Para evitar que detenga las simulaciones simplemente, se comenta en el código.

TAPBRIDGE

En el código correspondiente el TapBridge es donde hay que registrar los parámetros de QoS, ya que es la parte más cercana a los contenedores dentro del simulador NS3 y en ellos se pueden ver los paquetes que entran y salen de los contenedores LXC.

Cuando un paquete se transmite, la función m_PktTx devuelve dos parámetros:

- Time_Tx: Tiempo en el que el paquete sale del TapBridge hacia la red simulada.
- PacketSize: el tamaño del paquete en bytes.

Y la función m_ThroughputTx recoge la siguiente métrica:

- BufferTx: Throughput instantáneo de transmisión medido cada segundo en Bytes/segundo.

Por otra parte, en recepción, la función m_PktRx permite:

- SRC: Dirección MAC de la fuente del paquete.
- DST: Dirección MAC del destino del paquete.
- Time_Tx: Tiempo en el que el paquete salió del TapBridge del contenedor emisor del paquete.
- Time_Rx: Tiempo en el que el paquete se recibió en el TapBridge del receptor.
- Delay: Diferencia entre los tiempos de recepción y de transmisión del paquete, esto es el tiempo que tarda el paquete en atravesar toda la red.
- PktSize: Tamaño del paquete.

Y finalmente la función m_ThroughputRx registra dos parámetros:

- DST: Dirección MAC del destino final de los paquetes, el contenedor que los recibe.
- Throughput: El throughput de recepción instantáneo, medido cada segundo, en bytes/s.

IDENTIFICACIÓN SRC Y DST

Antes de pasar a explicar la modificación de las clases, es necesario describir el direccionamiento utilizado por el NS3.

EXPLICACIÓN DIRECCIONAMIENTO

Observando el esquema de la Figura 10 se pueden distinguir las direcciones MAC de los contenedores y de los dispositivos de red del NS3.

Cuando se realiza una transmisión de CONT1 a CONT2, la dirección MAC origen de los paquetes es la del contenedor 1, externo al NS3, y la dirección MAC destino es la del Gateway, dright(0), del contenedor 2.

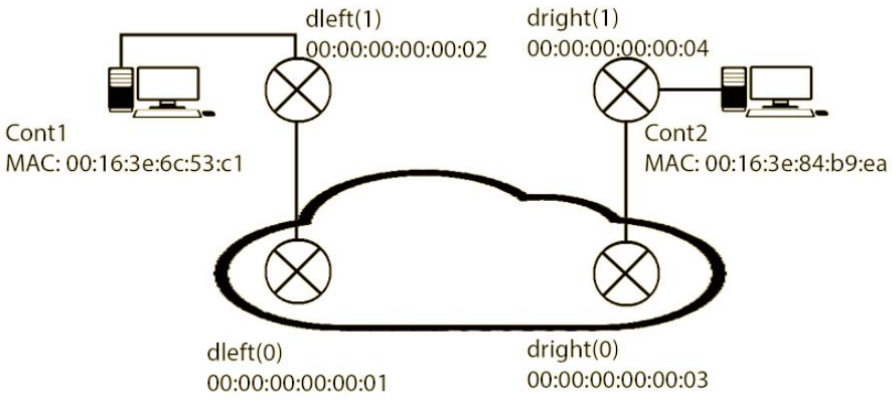


Figura 10. Esquema direccionamiento TapBridge

Gracias a esto se pueden identificar los paquetes que salen de cada contenedor y el correspondiente destino que tienen. Además, cuando los paquetes son transmitidos desde el nodo Gateway, su dirección SRC pasa a ser la MAC del nodo Gateway, manteniendo la dirección destino como la dirección del contenedor al que se quiere mandar el paquete.

Un ejemplo de cómo serían las direcciones MAC SRC y DST en una transmisión de CONT1 a CONT2 se muestra:

CONT1 → CONT2:

- Primero se transmite al Gateway.
  CONT1 → dright(0)
  SRC: 00:16:3e:6c:53:c1
  DST: 00:00:00:00:00:03
- Segundo, el Gateway reenvía al destino.
  Dright(0) → CONT2
  SRC: 00:00:00:00:00:03
  DST: 00:16:3e:84:b9:ea

MODIFICACIÓN CLASES

Las modificaciones en este caso se realizan solo en la clase tap-bridge.cc, que se puede encontrar en el apéndice tal y como se ha utilizado en las simulaciones llevadas a cabo.

A continuación se muestran los cambios que han sido necesarios.

TRANSMISIÓN

```
if (m_mode == USE_BRIDGE)
{
DelayTag myTag_;
uint64_t tsend = m_pktsynchronizer->GetCurrentRealtime();
myTag_.SetNanos (tsend);
packet->AddPacketTag(myTag_);
m_bridgedDevice->SendFrom (packet, src, dst, type);
```

myTag es una etiqueta que se añade al paquete, para indicar al TapBridge receptor el tiempo en el que el paquete se ha enviado desde el TapBridge emisor.

A la hora de transmitir los paquetes, interesa saber que se envían desde un contenedor concreto, en este caso Cont1, y de que se dirigen al Gateway correcto. Para hacer esto, se comparan las direcciones SRC y DST del paquete, con las direcciones de dichas entidades.

```
// if src = 02-06-00:16:3e:6c:53:c1 && dst = 02-06-00:00:00:00:00:01
ss << src;
std::string tmp = ss.str();
ss.str("");
ss.clear();
const char* csrc = tmp.c_str();
ss << dst;
tmp = ss.str();
ss.str("");
ss.clear();
const char* cdst = tmp.c_str();
if((strcmp(csrc,"02-06-00:16:3e:6c:53:c1")==0)&&(strcmp(cdst,"02-06-
00:00:00:00:00:01")==0)){
```

Para tener estadísticas de QoS, se recoge el tiempo de envío del paquete y su tamaño, pasándoselo al escenario con la función m_PktTX.

```
ss << myTag_.GetNanos() << " " << packet->GetSize() ;
m_PktTX(ss.str());
ss.str("");
ss.clear();
```

Finalmente, para obtener el throughput de transmisión cada segundo desde el contenedor, se utiliza el siguiente código.

```
BufferTx=BufferTx+packet->GetSize();
if((tsend-t_old)>=1000000000){ // 1 second
m_ThroughputTx(BufferTx);
BufferTx = 1;
t_old = tsend;
}
```

RECEPCIÓN

```
NS_LOG_LOGIC ("Writing packet to Linux host");
NS_LOG_LOGIC ("Pkt source is " << header.GetSource ());
```

```
NS_LOG_LOGIC ("Pkt destination is " << header.GetDestination ());
NS_LOG_LOGIC ("Pkt LengthType is " << header.GetLengthType ());
NS_LOG_LOGIC ("Pkt size is " << p->GetSize ());
//Jaar
//RX Packets, from here they go to the Linux Container.
DelayTag myTag2_;
packet->PeekPacketTag(myTag2_);
```

Lo primero que se tiene que hacer es recuperar la etiqueta con la información del tiempo de envío del paquete, a través de la función packet->PeekPacketTag(myTag2_), que sitúa en myTag2_ el tiempo en el que el paquete fue transmitido.

Esta etiqueta se utiliza posteriormente para calcular el tiempo que tardó el paquete en atravesar toda la red, para lo que se necesita también el tiempo en el que el paquete se ha recibido.

```
uint64_t trecv = m_pktsynchronizer->GetCurrentRealtime();
uint64_t pkt_retardo = trecv - myTag2_.GetNanos();
```

También se tiene que comprobar que el destinatario es el adecuado.

```
std::stringstream ss;
std::string tmp;
ss << header.GetDestination ();
tmp = ss.str();
ss.str("");
ss.clear();
const char* cdst = tmp.c_str();
if((strcmp(cdst,"00:16:3e:84:b9:ea")==0)|| (strcmp(cdst,"00:16:3e:c5:01:9c")==0)||
(strcmp(cdst,"00:16:3e:18:67:cc")==0)||(strcmp(cdst,"00:16:3e:67:7a:42")==0)||(str
cmp(cdst,"00:16:3e:ee:5c:7b")==0)){
ss << header.GetSource () << " " << header.GetDestination () << " " <<
myTag2_.GetNanos() << " " << trecv << " " << pkt_retardo << " " << p->GetSize ();
m_PktRX(ss.str());
ss.str("");
ss.clear();
//Calculate Throughput
BufferRx=BufferRx+p->GetSize();
if(trecv-t_old>=1000000000){ // 1 second
ss << header.GetDestination () << " " << BufferRx;
m_ThroughputRx(ss.str());
BufferRx = 0;
t_old = trecv;
}
}
```

Las MACs que se ven en la condición (if) se corresponden con las de los contenedores LXC en el host.

Gracias a las funciones m_PktRX y **m_ThroughputRx** se pueden caracterizar los datos que interesan para estudiar la QoS en el escenario, que los vuelca en ficheros para su posterior procesado y representación.

SIMULACIONES Y RESULTADOS

INTRODUCCIÓN

El objetivo de esta sección es presentar los resultados acerca del funcionamiento del NS3 y la calidad de las transmisiones del streaming de vídeo.

Para ello se hacen una serie de experimentos en los que se obtienen unas estadísticas de rendimiento que dependen de diferentes factores.

El objetivo del primer escenario es identificar el factor que en mayor medida afecta el rendimiento del planificador de tiempo real del NS3, y para ello se va modificando el número de dispositivos físicos en una red y el número de flujos de datos. Una vez realizado este análisis, se lleva a cabo otro estudio que pretende obtener estadísticas tanto de la diferencia entre tiempos de reloj de simulación y de host, como de la QoS de los flujos de video. Se pretende analizar cómo reacciona NS3 cuando trabaja con varios streamings de video reales.

Una vez caracterizado el comportamiento del NS3 con y sin contenedores, se introducirán errores de transmisión en el canal radio, incrementando el realismo del análisis.

El último de los estudios llevados a cabo se basará en un escenario realista, para analizar la variación de la QOS en entornos reales simulados con presencia de errores.

CONFIGURACIÓN DE LAS SIMULACIONES

Todas las simulaciones se han llevado a cabo en un portatil ASUS S500CA, con un procesador Intel Core i5-3317U @ 1.7GHz y 8GB de memoria RAM. Se ha utilizado Ubuntu 14.04 LTS (64 bits) como sistema operativo. Pasos previos a dar antes de comenzar con las simulaciones sib kis descrutis seguidamente.

- Instalar NS3 3.21 y las librerías que necesita para funcionar, utilizando el script installns3.sh.
- Instalar LXC para crear contenedores; script configLXC.sh.
- Crear un primer contenedor con VLC y la carpeta compartida para acceder al video para hacer streaming.
- Crear tantos contenedores como sea necesario para llevar a cabo las simulaciones, empleando el script createContainer.sh. Posteriormente será necesario arrancarlos en background antes de simular, aunque antes se requiere activar los tap devices y vincularlos con el bridge LXCBR0, lo que facilita el script attachTAP.sh.
- Para poder observar los resultados de las simulaciones se requiere modificar las clases del NS3, tal y como se ha indicado previamente.
- Para realizar las simulaciones solo queda configurar los escenarios del NS3 y automatizar el proceso con scripts, para así obtener los resultados necesarios.

- Finalmente, el último paso es la interpretación de los resultados, mediante el uso de la herramienta MATLAB.

SIMULACIÓN SIN CONTENEDORES

Este primer estudio se lleva a cabo sin contenedores, ni aplicaciones reales, para observar el comportamiento del simulador cuando al variar el número de nodos de la red. Para ello se utilizará AODV como protocolo de encaminamiento, y se activarán varios flujos de datos haciendo uso de la application de NS3 "OnOff".

La aplicación OnOff es un generador de tráfico, que utiliza un patrón de tráfico sencillo con una tasa constante. Así, cuando la aplicación está en "On", manda tráfico CBR, a una tasa que se ha configurado a 11Mbps en este caso.

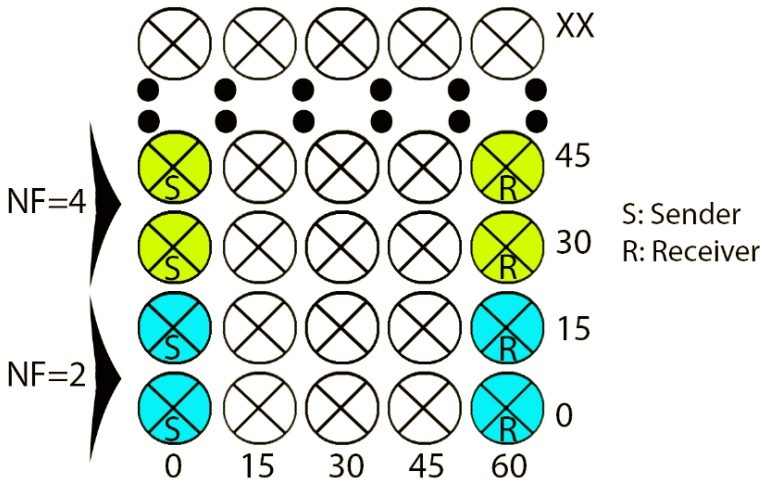


Figura 11. Esquema Simulación Sin Contenedores

RESULTADOS

Estos resultados han sido obtenidos mediante la utilización del script "nocontsimulation.sh", que ejecuta el escenario en el NS3 con el comando "./waf --run="nocontsim --nadd=$i --tsim=$timesim --nflujos=$j --nsim=$k --range=15"".

- Nadd es el número de nodos que hay en el escenario Wi-Fi, que varía de 50 a 300 nodos.
- Tsim es el tiempo de simulación, en este caso, 60 segundos.
- Nflujos, número de flujos de datos que se activan en el escenario [2,4,6,8].
- Nsim es el número de simulación, se utiliza para poder guardar estadísticas en diferentes ficheros.
- Range, el rango de cobertura de los nodos.

Por lo tanto, teniendo en cuenta la duración de cada simulación (~15 seg.), el número de repeticiones que se hará cada configuración del escenario (20 veces), el número de configuraciones que se analizan (x6x4), se puede estimar la duración total de este primer análisis.

20x6x4x15segundos = 7200 segundos = 2 horas.

Para representar los resultados se procesarán los ficheros con MATLAB.

A continuación se muestran una serie de resultados que representan la evolución temporal de la diferencia entre los tiempos de reloj de simulación y del host para una única repetición de la simulación, así como la CDF () obtenido sobre todas las repeticiones del experimento.

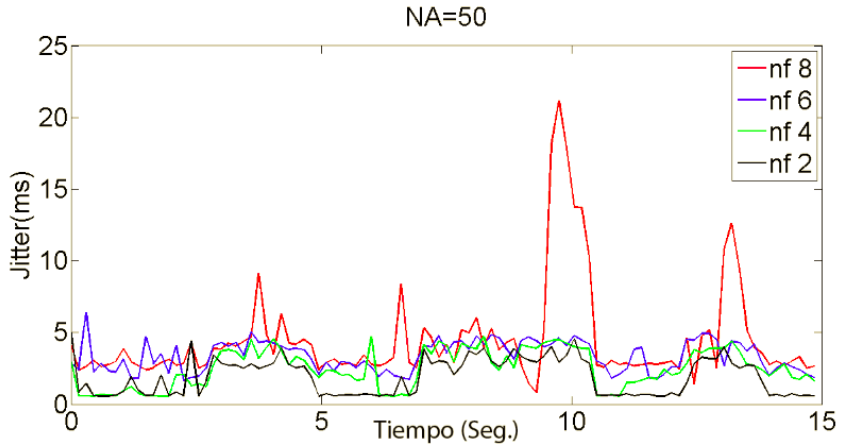


Figura 12a. Hard Limit jitter – 50 nodos

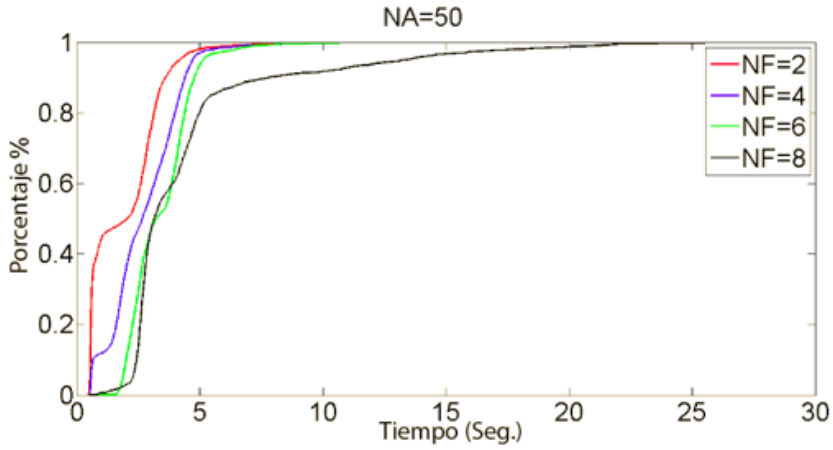


Figura 12b. CDF Hard Limit jitter – 50 nodos

Las figuras Figura12a y Figura12b se corresponden con el caso con 50 nodos. En las gráficas se puede observar una línea por cada número de flujos en el escenario (de 2 a 8).

Cómo se había indicado, en el escenario se utiliza AODV como protocolo de encaminamiento y, debido a que cada segundo se envían paquetes de HELLO para mantener la conectividad, se puede observar "unos picos" cada segundo, para posteriormente recuperarse.

Se puede ver, como es lógico ver que los valores de jitter aumentan con el número de flujos activos en el escenario. Sin embargo con 50 nodos en el escenario, las diferencias son muy pequeñas. Como se puede apreciar en la CDF, en un 80% de las veces el valor del jitter es menor de 5 ms, y cuando hay 8 flujos de datos el jitter podría aumentar hasta los 22 ms, mientras que para 2, 4 y 6 flujos, se mantiene menor o igual a 5ms en un 97% de las veces, con picos que llegan a los 7-8 ms. Como se ha dicho antes estas subidas son consecuencia de los mensajes de HELLO del AODV, con el comportamiento de la red es apropiado.

Por lo tanto se podría predecir que no habría problemas con tráfico y aplicaciones reales en este escenario.

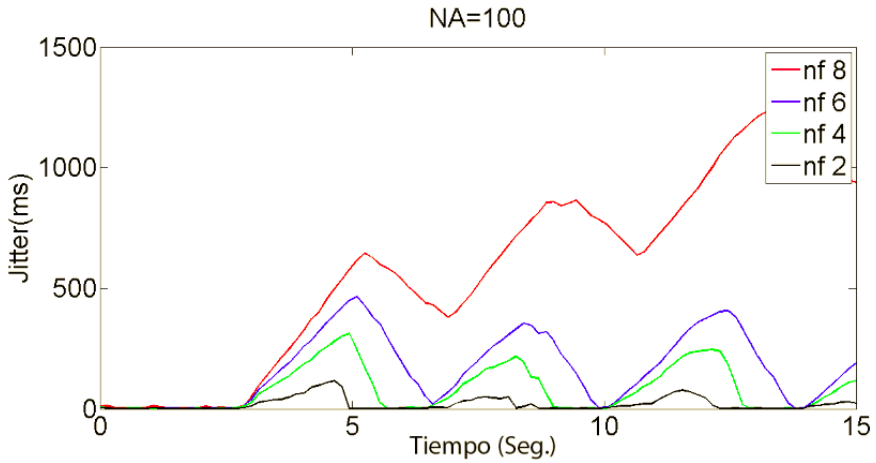


Figura 13a. Hard Limit jitter – 100 Nodos

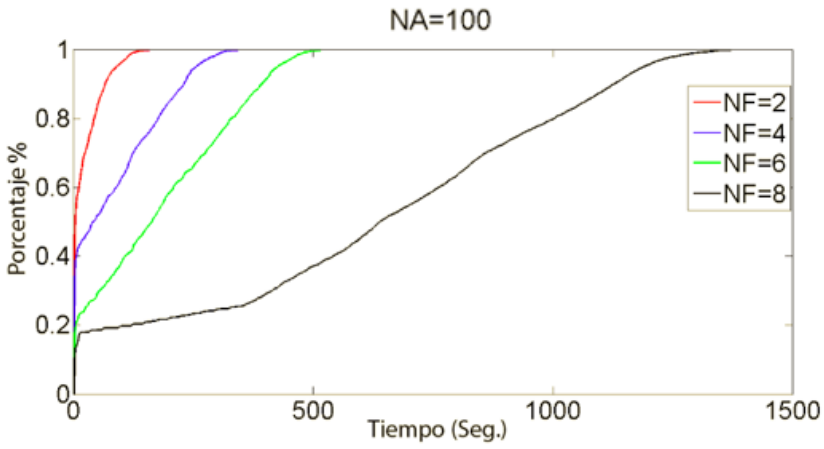


Figure 13b. CDF Hard Limit jitter – 100 Nodos

En las figuras Figura13a y Figura13b se ha considerado 100 nodos en el escenario, y es fácil ver que cuando hay 8 flujos activos, el tiempo de reloj de simulación diverge del correspondiente al host, lo cual pondría un límite para trabajar con tiempo real, e incluso supondría un problema en cuanto a la rapidez de las simulaciones.

Al observar el jitter con NF=6, se ve que se llegan a valores de 500 ms y, aunque estos valores se recuperan eventualmente, es cierto que en un 60% de los casos el jitter se sitúa entre 150 ms y 500 ms, lo que podría suponer un problema a la hora de trabajar con aplicaciones reales.

Con NF=4 el jitter alcanza los 350 ms, pero en el 60% de las veces se mantiene por debajo de 100 ms, un valor aceptable, así, en un 40% de las veces el jitter no sería aceptable para trabajar en tiempo real. Se pone nuevamente de manifiesto una lenta recuperación después de los picos que se producen.

En el último de los escenarios se activan 2 flujos de datos; los valores de jitter son menores de 100ms en un 95% de las veces, lo que indica que sería plausible trabajar sobre este escenario con aplicaciones reales. Además el valor máximo del jitter observado es de 150 ms, recuperándose tras alcanzar dichos máximos.

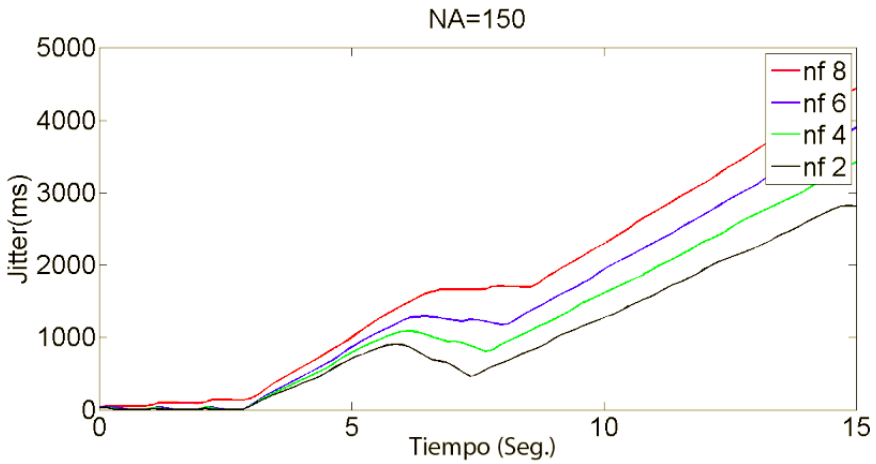


Figura 14a. Hard Limit jitter – 150 Nodos

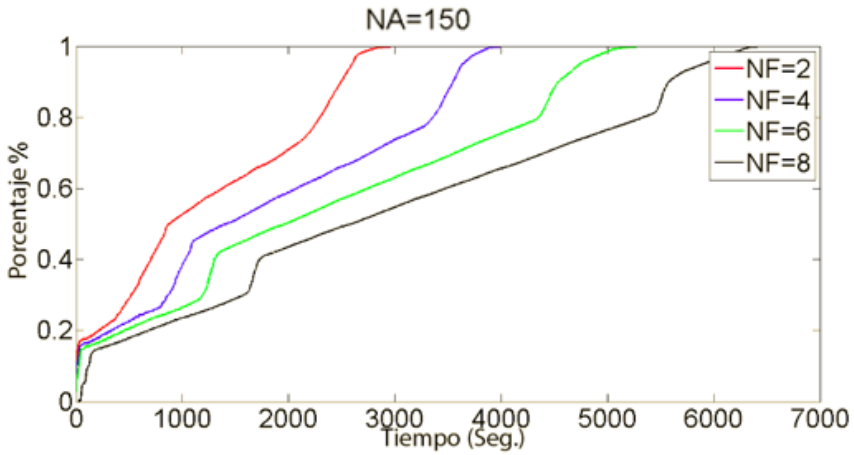


Figure 14b. CDF Hard Limit jitter – 150 Nodos

En las figuras Figura14a y Figura14b se aprecia fácilmente la imposibilidad de trabajar con aplicaciones reales sobre un escenario con 150 nodos. La diferencia entre los tiempos real y de simulación diverge en todos los casos, alcanzando valores máximos que se sitúan entre 3 y 6.25 segundos.

Como cabría esperar, en escenarios con más de 150 nodos, la diferencia entre ambos tiempos es mayor y, por lo tanto, no son apropiados para trabajar en tiempo real.

En conclusión, el aspecto con mayor repercusión sobre el rendimiento del NS3 es el número de dispositivos que hay en la red simulada, lo que se explica, entre otros motivos, por el hecho de que el tráfico AODV crece por cada nodo que se añade a la red.

EFECTO CONTENEDORES

Para conocer el impacto del número de contenedores recibiendo el flujo de vídeo simultáneamente, se utiliza un escenario con parejas de contenedores a una distancia superior al área de cobertura evitando que haya interferencia entre ellos; 60 metros, se hacen streamings de video en paralelo, utilizando 1, 2, 3 y 4 parejas.

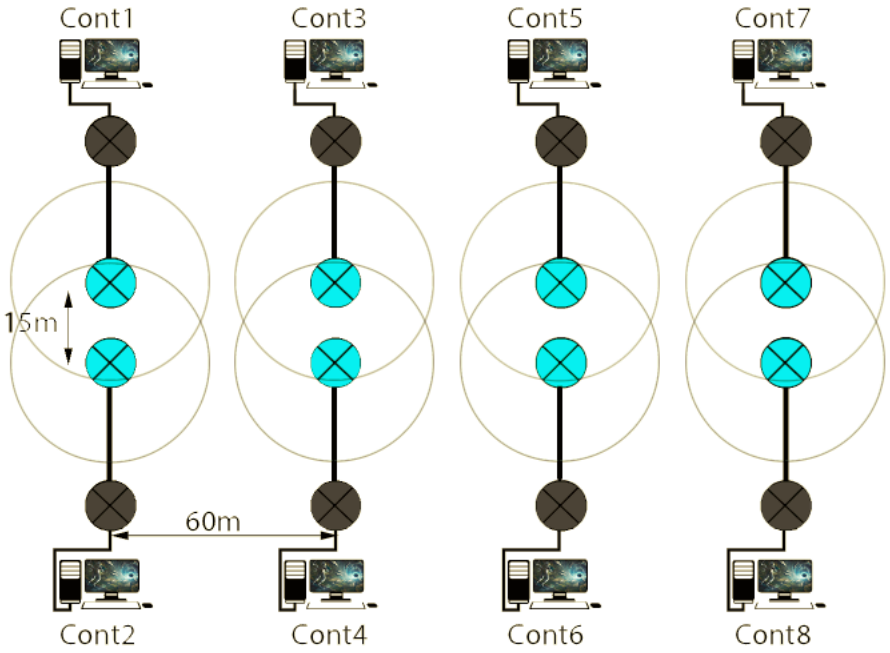


Figura 15. Esquema Simulación Efecto Contenedores

RESULTADOS

Los resultados han sido obtenidos mediante la utilización del script "mul_stream_0_router.sh", que ejecuta el escenario en NS3 con el comando "./waf --run="mul_stream_0_router --ncont=$numcont --tsim=$timesim --nsim=$numsim"".

- Ncont es el número de contenedores que se consideran: [2,4,6,8].
- Tsim es el tiempo que el escenario se va a estar simulando, fijando en este caso 60 segundos.
- Nsim es el contador del experimento, que facilita almacenar estadísticas en diferentes ficheros.

El tiempo total que este grupo de simulaciones llevan se puede calcular como sigue:

60 segundos de simulación + 15 segundos de inicialización = 75 segundos

20 repeticiones por cada configuración

De 2 a 8 contenedores, en saltos de 2, hacen 4 escenarios.

Tiempo total=(75seg)x4x20= 6000 segundos = 100 minutos

HARD LIMIT JITTER

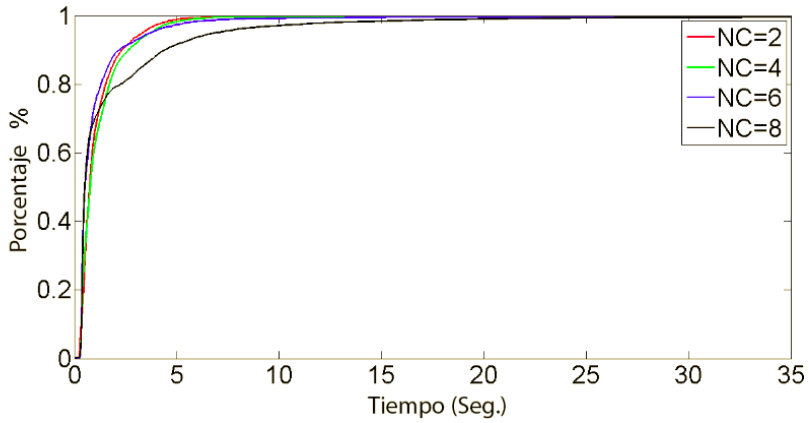


Figura 16. Hard Limit Jitter – CDF

La Figura16 representa la CDF del jitter, considerando todas las repeticiones realizadas.

Se ve que la diferencia entre los tiempos de simulación y real es muy pequeña en cualquiera de los casos. Con dos, cuatro y seis contenedores no se aprecian diferencias significativas (menores de 5 ms en el 98% de las ocasiones), con un ligero incremento en el caso que se consideran 8 contenedores.

Se vuelve a poner de manifiesto, por tanto, que lo que más afecta al rendimiento del NS3 es el número de dispositivos; hay que recordar que en este escenario solo se despliegan 8 nodos en el NS3.

EVOLUCIÓN DEL JITTER ENTRE PAQUETES

El jitter entre paquetes se mide a través de la diferencia entre los instantes de llegada de un paquete y del anterior. Esta medida permite conocer si la red simulada se ve afectada por el tráfico que está siendo transmitido a través de ella, y cuál es la diferencia al aumentar el número de flujos.

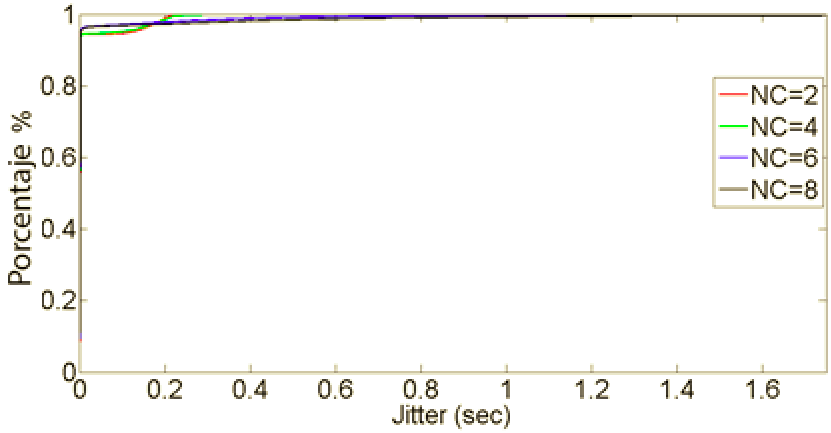


Figura 17. Packet Jitter – CDF

En esta ocasión, la Figura17 muestra la CDF del jitter entre paquetes en el primer, el contenedor “cont2”.

El 92.5% de los paquetes tienen un jitter menor a 2.3ms, lo que se corresponde una calidad de servicio muy elevada.

THROUGHPUT DE TRANSMISIÓN – CONT-1

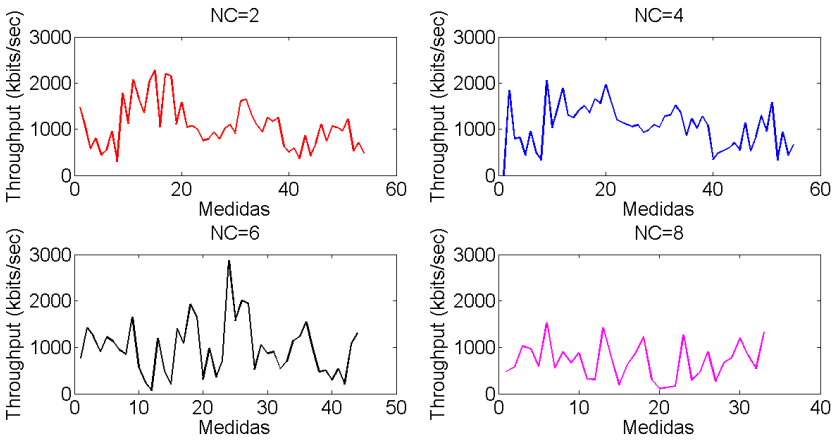


Figura 18. Throughputs de transmisión del contenedor “cont-1”

La Figura18 muestra el throughput de transmisión del contenedor “cont-1”, con diferente número de contenedores transmitiendo a la vez. De esta manera se puede entender mejor el comportamiento del equipo y del NS3 al trabajar con varios contenedores ejecutando el VLC y realizando streaming de video simultáneamente.

El throughput se muestra muy variable porque la tasa de envío es 20 fotogramas por segundo (fps), con un tamaño que va cambiando dinámicamente; en un streaming de VLC se envían paquetes que son más importantes que otros, contienen más información y son más largos.

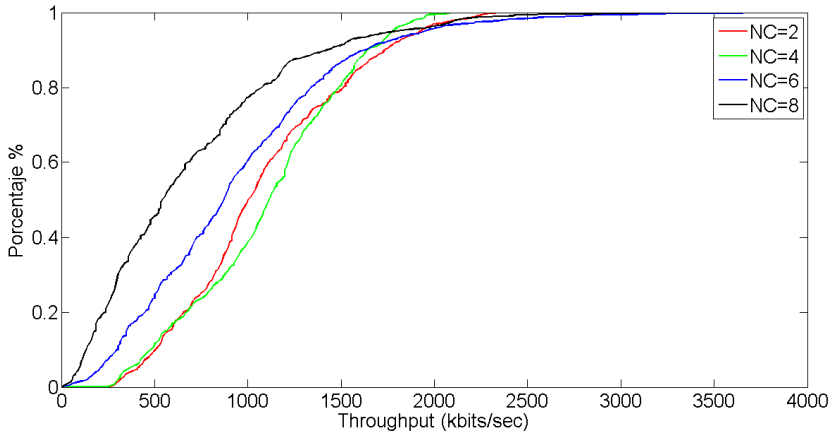


Figura 19. CDF de los throughputs de transmisión del contenedor “cont1”

Se ve que el throughput de transmisión es muy similar cuando hay dos y cuatro contenedores, mientras que se reduce cuando se incrementan a seis, y más aún hasta ocho activos al mismo tiempo. Como el throughput de transmisión se obtiene en el dispositivo TapBridge del NS3, se mide antes de que el tráfico atraviese la red simulada, por lo que es razonable pensar que la limitación en este caso está en el equipo utilizado. Al ejecutarse el mismo número de VLC que contenedores, y estando la mitad transmitiendo un flujo de vídeo de elevada calidad, el número de paquetes que pasan a los taps es elevado, y da la impresión que el ordenador no puede funcionar adecuadamente con todos a la vez, en los casos de seis y ocho contenedores.

Se puede concluir que para trabajar óptimamente con el equipo mencionado, con el NS3 y VLC ejecutándose en contenedores, se debe limitar a un máximo de cuatro contenedores, o dos flujos de video.

THROUGHPUT DE RECEPCIÓN

Con este parámetro se analiza el throughput en el TapBridge receptor, dentro de la red del NS3, y una vez que el tráfico ha atravesado toda la red simulada.

En este caso el análisis se centra más en la influencia del NS3, con el tráfico que se está transmitiendo.

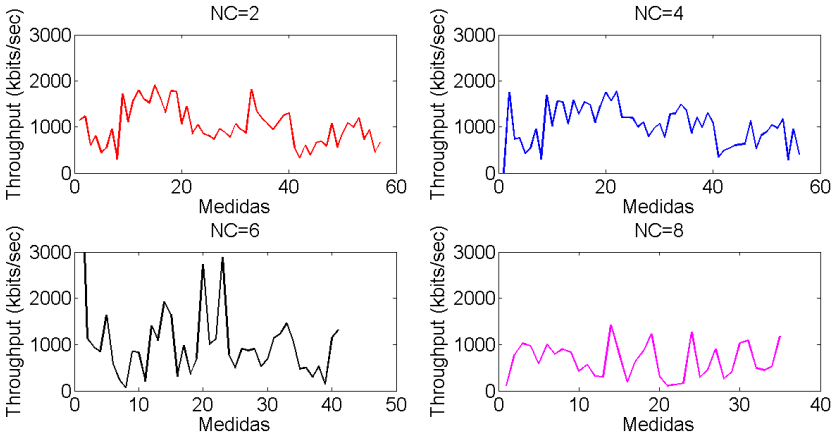


Figura 20. Throughputs de recepción del contenedor "cont-2"

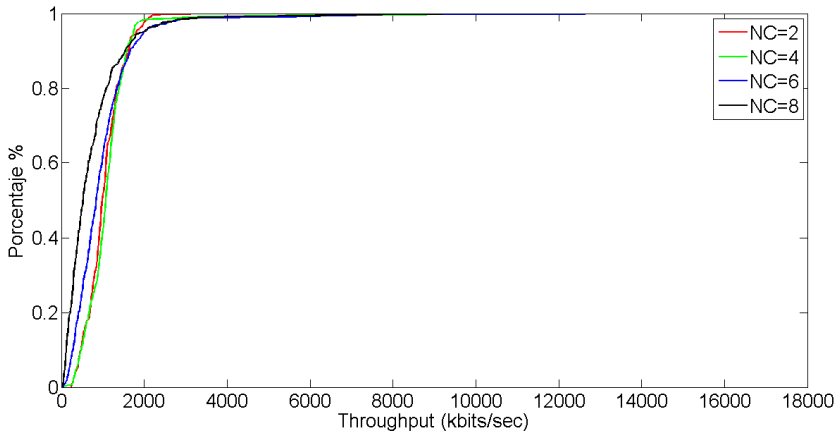


Figura 21. CDF de los throughputs de transmisión del contenedor "Cont2"

El throughput de recepción se comporta de manera similar el throughput de transmisión, lo que supone que el simulador no introduce problemas en cuanto a la cantidad de tráfico viajando a través de él.

THROUGHPUTS HISTOGRAMA

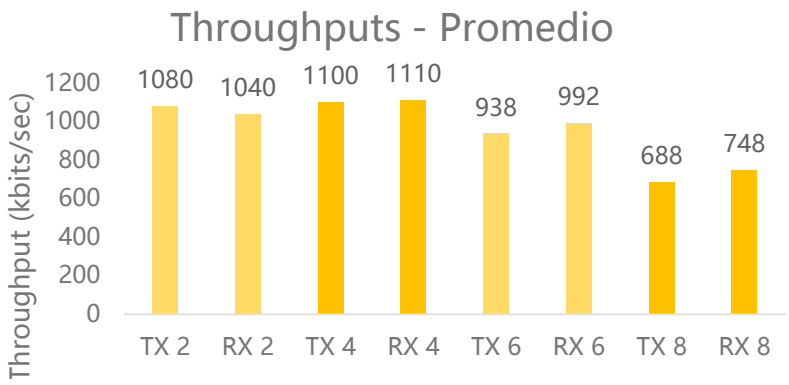


Figura 22.

La Figura22 muestra los valores promedio del throughput de transmisión (contenedor "cont1") y del throughput de recepción (contenedor "cont2"), para un número de contenedores variable [2,4,6,8].

Se concluye de la Figura22 que cuando hay 6 y 8 contenedores, se produce un estancamiento en el ordenador, que no puede transmitir tantos paquetes desde los contenedores a la red, por lo que en este caso, no se podrían utilizar más de 4 contenedores activos simultáneamente.

RETARDO PAQUETES

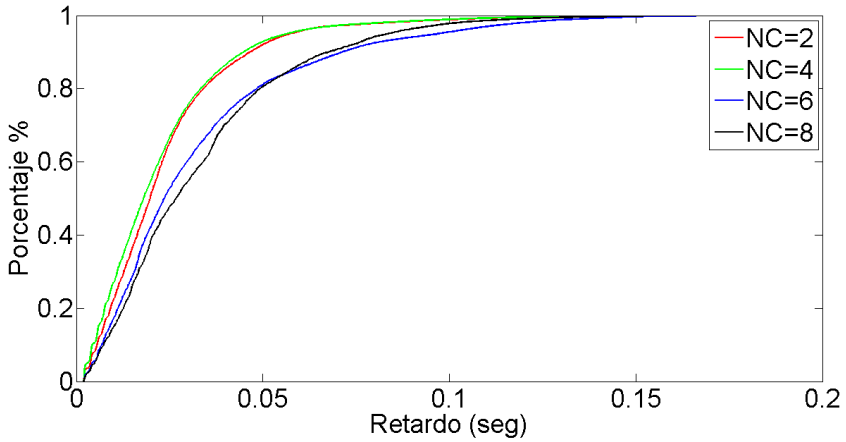
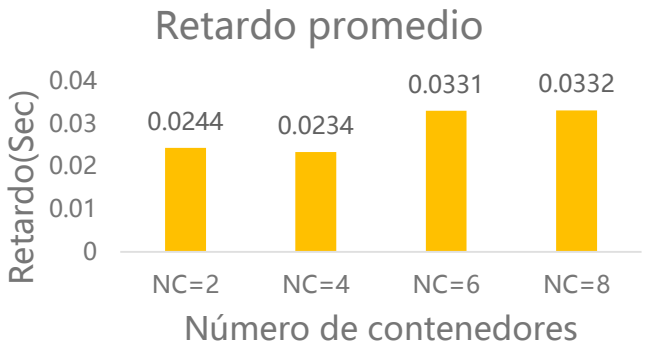


Figura 23. Retardo de los paquetes

La Figura23 representa la CDF del retardo entre paquetes, observándose que el 90% de los que llegan al contenedor "cont-2", cuando solo hay dos y cuatro contenedores, tienen un retardo inferior a los 0.05 segundos, mientras que en el caso de seis y ocho contenedores este porcentaje se reduce en un 10%. Se concluye, que la red simulada por NS3 comienza a tener problemas por el gran número de paquetes que tiene que transmitirse desde un extremo de la red hasta el otro. Se puede deducir que, debido al diseño de la red Wi-Fi en el NS3, los paquetes generados con la configuración en el VLC de 20 fps, una tasa de muestras de 44100 Hz y velocidad binaria de 128 kb/segundo, por el número de contenedores transmitiendo, causa congestión en la red, y esto es lo que deriva en el aumento del retardo, aunque no parece que sea un problema para la red.

Hay que aclarar que, pese a que los nodos Wi-Fi no interfieren unos con otros, comparten el mismo canal 802.11.

Este efecto se puede apreciar mejor comparando los valores promedio de retardo de los paquetes recibidos en el contenedor "cont2".



TASA DE PÉRDIDA DE PAQUETES

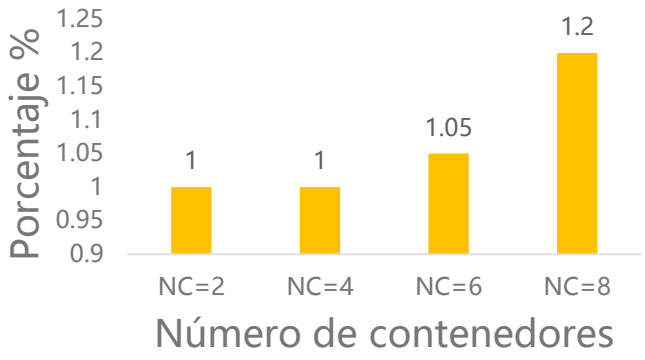


Figura 24.

Se observa un aumento de la tasa de pérdida al pasar a seis y ocho contenedores, lo que, junto con el aumento del retardo, hace pensar que empiezan a aparecer algunas limitaciones en el NS3 cuando se utilizan seis contenedores, y éstas se van acentuando a medida que se añaden más.

CONCLUSIÓN

Para tener un comportamiento óptimo por parte del NS3 y del equipo, no pueden realizarse simulaciones con más de cuatro contenedores o dos flujos de vídeo. Es posible sin embargo, que al utilizar un vídeo de peor calidad, los valores de QoS indiquen la posibilidad de activar más streamings simultáneamente.

SIMULACIÓN CON ERRORES EN EL CANAL

Para comprobar el funcionamiento del modelo de errores sobre enlaces inalámbricos, se genera un escenario simple en el que se envía un streaming entre dos contenedores, y se varía la FER (Frame Error Rate) del enlace entre los Gateway de los contenedores.

Se pretende estudiar la pérdida de paquetes en función de la FER.

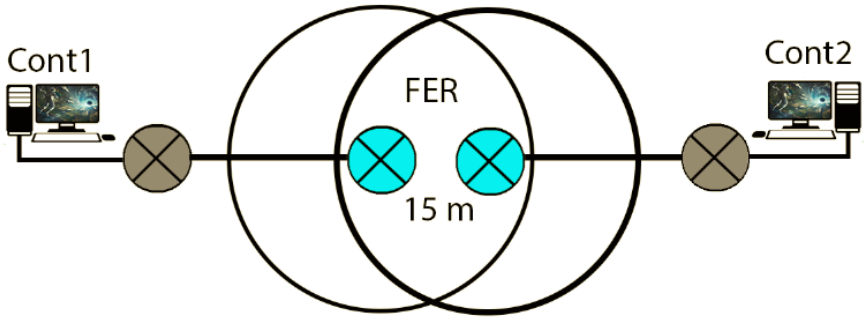


Figura 25. Esquema Simulación Prueba Errores

RESULTADOS

Estos resultados han sido obtenidos mediante el script “fer_error.sh”, que ejecuta el escenario en el NS3 con el comando “../waf --run=”fer_error --tsim=$timesim --nsim=$numsim --err1=$e1 ”.

- Tsim es el tiempo que el escenario se va a estar ejecutando; en este caso, 60 segundos.
- Nsim es el número de simulación, y facilita la recogida de resultados en múltiples ficheros.
- Err1 es un número del 0 al 10, que establece la FER del canal, entre 0 y 1, en pasos de 0.1.

El tiempo de ejecución total del script se estima tal y cómo sigue.

(60 + 6 segundos)*(20 repeticiones)*(11 casos de error) = 14520 segundos = 4.03 horas.

REALTIME JITTER

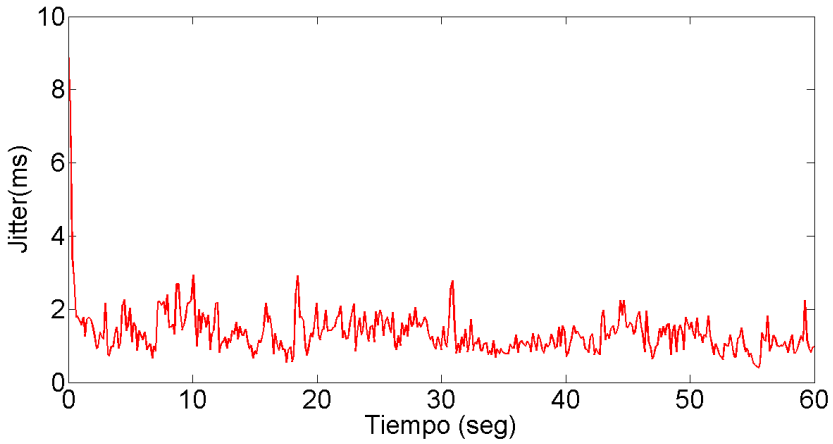


Figura 26. Diferencia tiempos simulación y host.

Puesto que no se observan cambios significativos en la diferencia entre los tiempos real y simulación al modificar la tasa de error en el canal, solo se muestra una única gráfica, representativa de todas ellas. Esta gráfica es Figura26 y es la que resulta de usar una FER igual a 0.4.

Del análisis anterior se concluye que la tasa de error no influye en la diferencia entre los tiempos del host y de simulación.

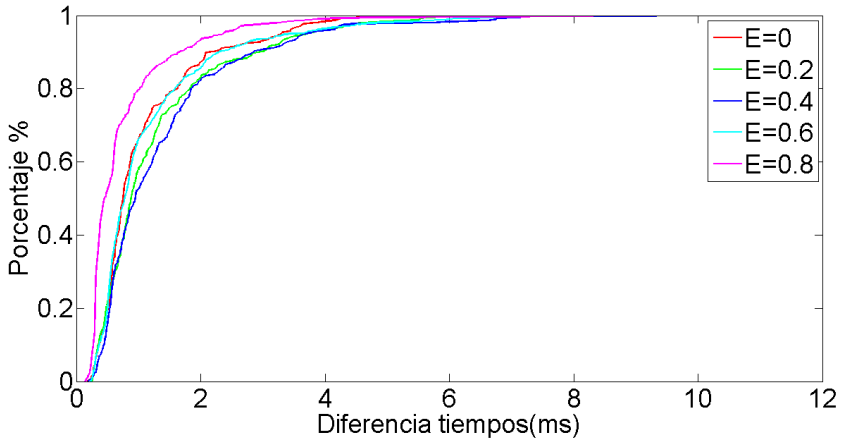


Figura 27. CDF de la diferencia entre tiempos de simulación y host.

En la CDF (Figura27) se observa claramente el gran parecido del jitter para los diferentes valores de FER, y también que la diferencia entre tiempos es realmente pequeña en este escenario, estando, en el 80% de las veces, por debajo de 2 ms (el porcentaje aumenta hasta el 95 % para un jitter de 4 ms).

TASA DE PÉRDIDA DE PAQUETES

En este caso, la medida de mayor interés es la tasa de error de paquetes (ver Figura28).

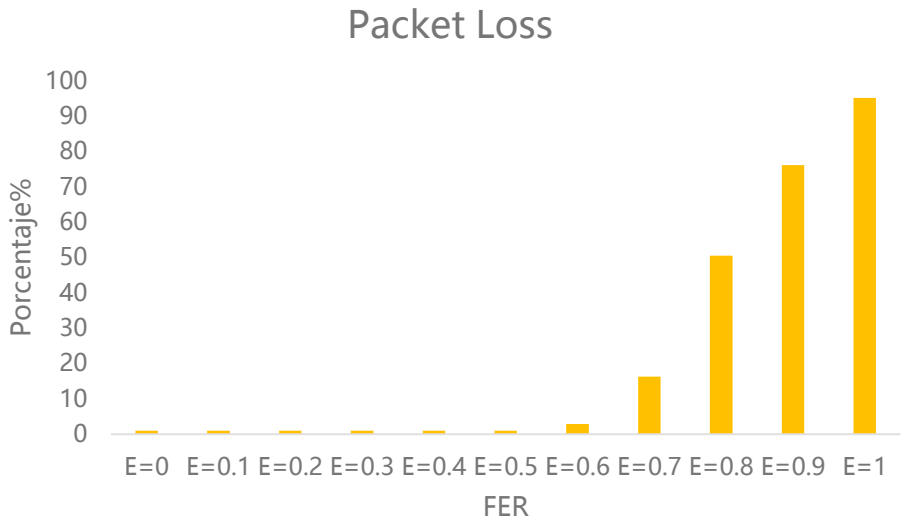
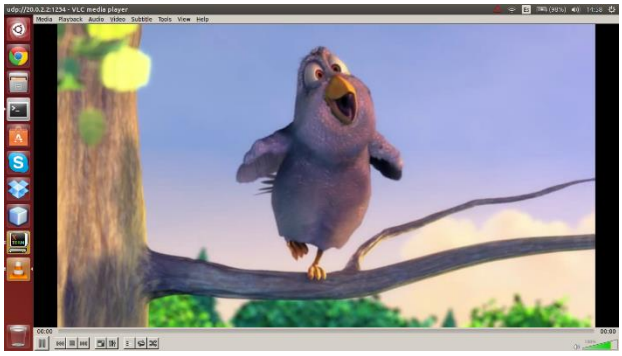


Figura 28. Tasa de pérdida de paquetes.

Se ve que, por ejemplo, introducir una FER del 50%, resulta en una pérdida de paquetes alrededor del 4%. Hay que tener en cuenta que, según el estándar 802.11, un paquete se retransmite hasta en cuatro ocasiones antes de darle por perdido y que, además, el modelo de canal que se ha utilizado asume que los paquetes que sean más cortos de 1000 bytes siempre llegan correctamente.

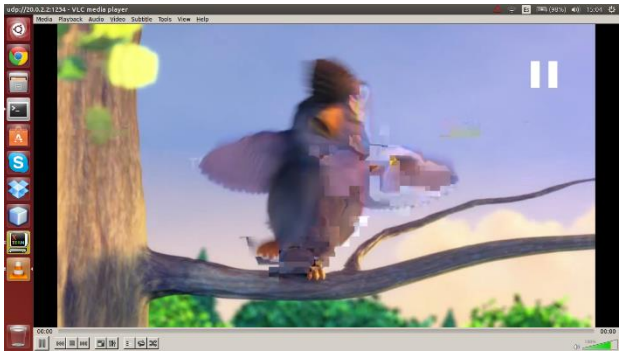
CAPTURAS DE PANTALLA

FER=0

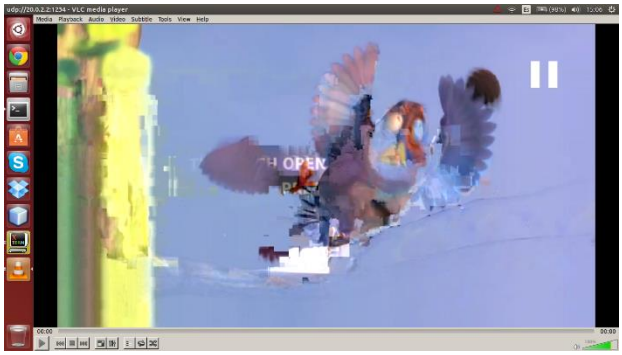


FER=0.4

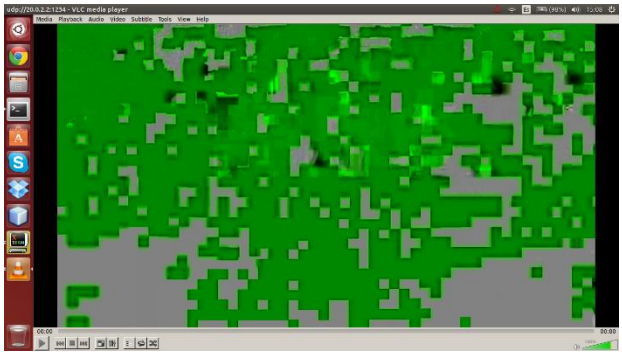
FER=0.5



FER=0.7



FER=0.9



SIMULACIÓN COMPLETA

A partir de los resultados obtenidos en las simulaciones previas, se identifican los escenarios que pueden funcionar adecuadamente.

Por lo tanto, se procede a describir el análisis de un escenario real, que cumpla con el requisito de jitter para analizar su comportamiento en cuanto a la QoS. Además se observará el resultado del streaming en los clientes.

El escenario simulado se muestra en la Figura29. Se consideran tres contenedores, uno de ellos hace las veces de servidor de video (cont1), y los otros dos son los clientes (cont2 y cont3). Los dos streamings siguen rutas diferentes. El camino a Client 1 tiene cuatro saltos, mientras que para llegar a Client 2 solo se requieren dos saltos. La FER en cada enlace se muestra en Figura29.

El servidor envía un flujo con una frecuencia de muestras de 44100 Hz, tasa binaria de 128 Kb/s, y se configura a 15 fps.

Los resultados que se podrían esperar es que la QoS en el cliente 2 sea mejor que la del cliente 1, y que, por tanto, la calidad del video sea mejor. Además, como la ruta hacia el cliente 1 necesita el doble de saltos, es de esperar un incremento del retardo equivalente.

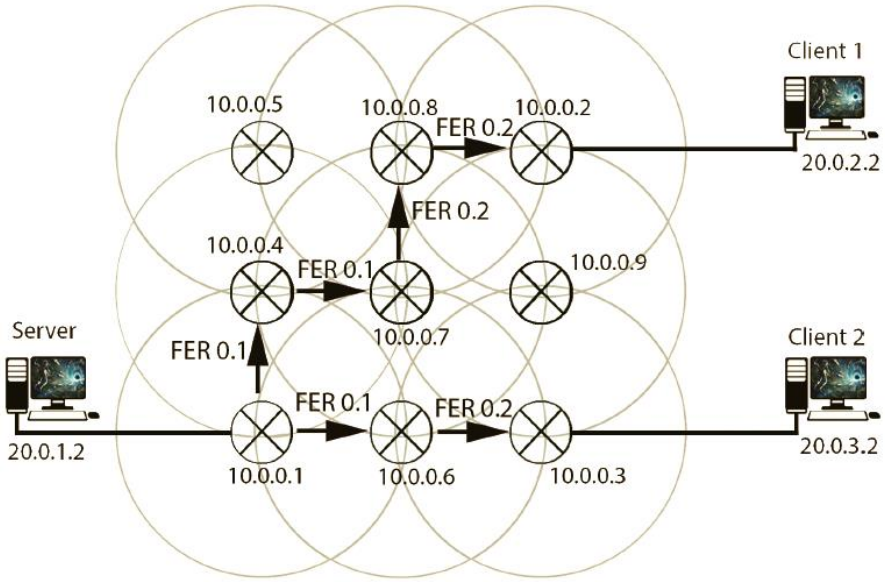


Figura 29. Esquema Simulación Completa

RESULTADOS

REALTIME JITTER

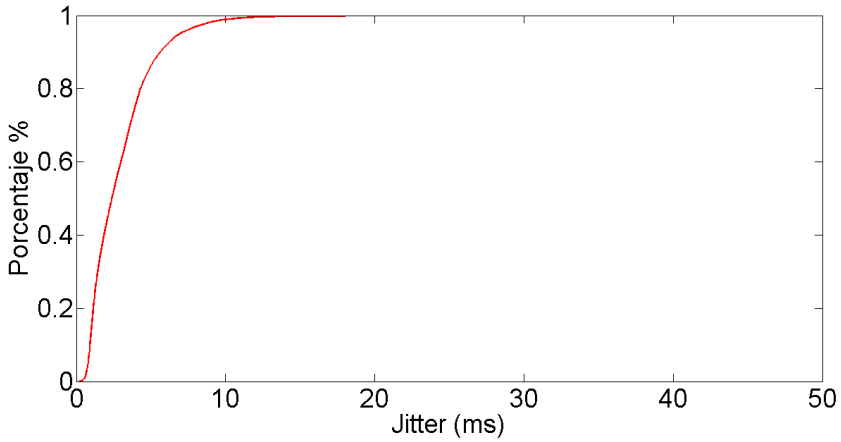


Figura 30. Hard Limit Jitter NS3 – CDF

Como era de esperar tras haber realizado las simulaciones anteriores, los valores de la diferencia entre los tiempos de reoj en simulación y en el host es pequeña, siendo menor de 10 ms en un 96 % de los casos. Cabe resaltar que no sería razonable imponer un “hard-limit” que parase la simulación, pues se produce un máximo puntual, pero que se recupera de manera rápida.

JITTER ENTRE PAQUETES

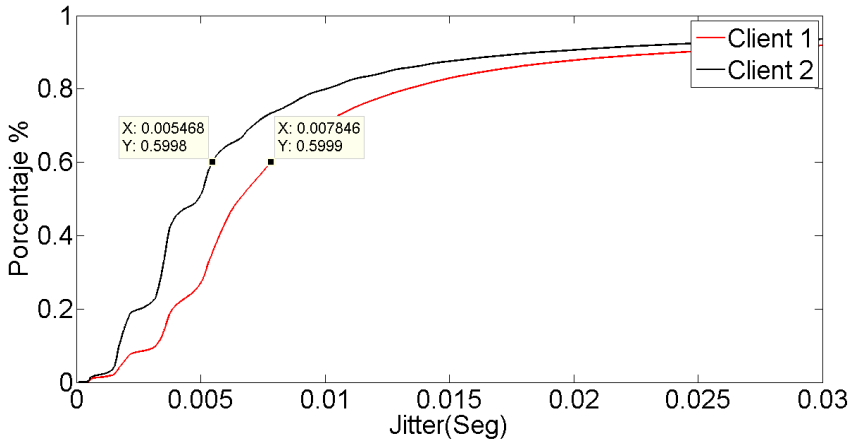


Figura 31. Jitter de los paquetes en clientes 1 y 2 – CDF

Comparando las CDF del jitter de los paquetes en ambos clientes se ve que son similares, con unos valores apropiados, en un 80% de los casos menor a 10 y 15 ms en los clientes 2 y 1 respectivamente, reduciéndose a 5 ms y 8 ms para un 60 % de los valores medidos.

Se cumple que el jitter en el cliente 1 es ligeramente superior al del cliente 2, con casi 5 ms de diferencia entre ambos. En el caso de este último es de esperar que tenga cierto impacto en la calidad del video.

El cliente 1 tiene unos valores de jitter que se corresponden con una muy buena QoS. El cliente 2 tiene unos valores de jitter peores, lo que se percibe a la hora de ver el vídeo en el cliente.

RETARDO

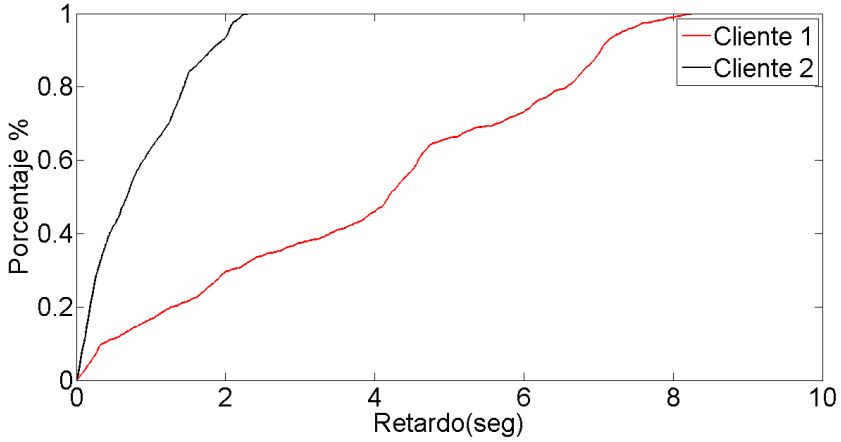


Figura 32.Retraso paquetes – CDF

La Figura32 representa la CDF del retardo de los paquetes en ambos clientes, para analizar la diferencia existente en ambos casos.

Como la ruta hacia el cliente 1 es más larga, era esperable que el retardo fuera mayor que el correspondiente al cliente 2. La consecuencia sobre el streaming de video es que empezaría a reproducirse más tarde que en el cliente 2, pero gracias a los buffers y a que el jitter entre paquetes consecutivos es adecuado (como se ha visto antes), no se produce una disminución aparente de la calidad de servicio del video.

Este parámetro de retardo podría ser más relevante a la hora de realizar videoconferencias o aplicaciones con interacción con el usuario, puesto que el tiempo de espera sería muy elevado.

PAQUETES RECIBIDOS EN CADA CLIENTE

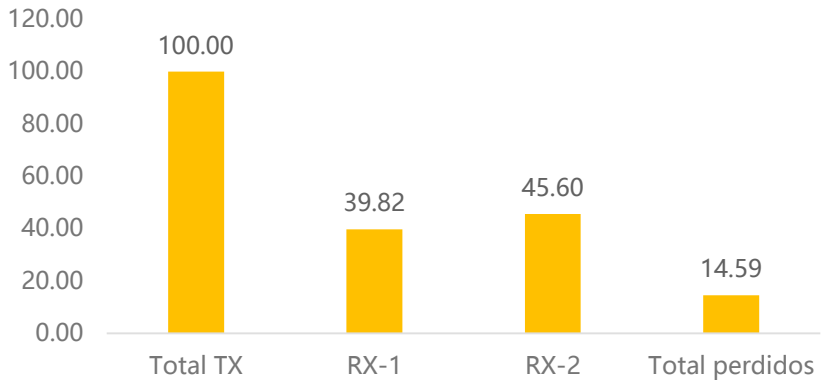


Figura 33. Porcentaje de paquetes recibidos en cada cliente.

En la Figura33 se representa el total de paquetes que se envían al cliente 1 y al cliente 2, en la columna Total TX. La columna RX-1 se corresponde con el porcentaje de paquetes que recibe el cliente 1, y RX-2 refleja el porcentaje que recibe el cliente 2. La última columna representa el total de paquetes perdidos en total (cliente 1 y cliente 2).

Teniendo en cuenta que un streaming de video podría soportar pérdidas menores al 5%, se puede decir que la calidad del servicio en el cliente 1 podría ser no apropiada.

En efecto, se ve que la calidad en el cliente 1 no es adecuada, y el vídeo se congela en numerosas ocasiones, con una calidad con la que un cliente no usaría el servicio de streaming. También se observa que la calidad en el cliente 2 es muy buena la mayor parte del video, habiendo sin embargo momentos en los que algunas partes de la imagen no se aprecian correctamente.

TROUGHPUTS

Las siguientes figuras muestran la evolución del rendimiento durante la simulación y sus CDF correspondientes. La línea azul representa el throughput de transmisión del contenedor “Cont1”, mientras que las líneas roja y negra se corresponden con los throughputs de recepción en los contenedores “Cont2” y “Cont3”, o cliente 1 y cliente 2, respectivamente.

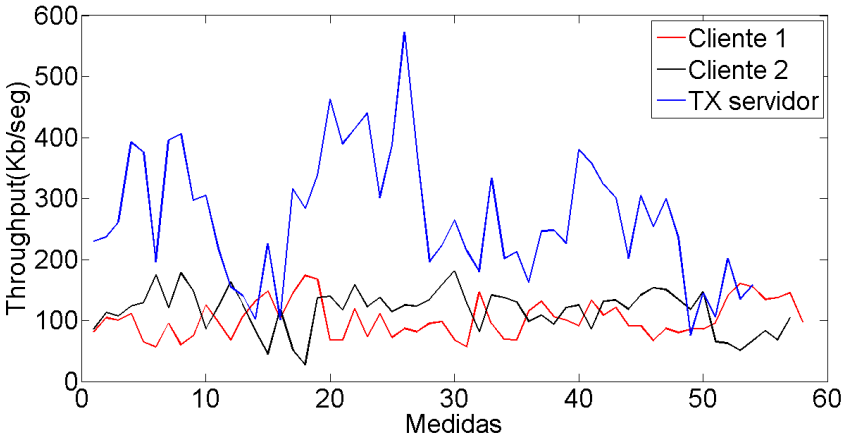


Figura 34. Throughputs – Evolución temporal

El comportamiento a picos que se observa se debe al tamaño variable de los fotogramas que se transmiten y reciben, como ya se había mencionado al analizar el Throughput de transmisión – Cont-1 (Figura18). En este caso también hay que destacar la influencia de la pérdida de paquetes en la red.

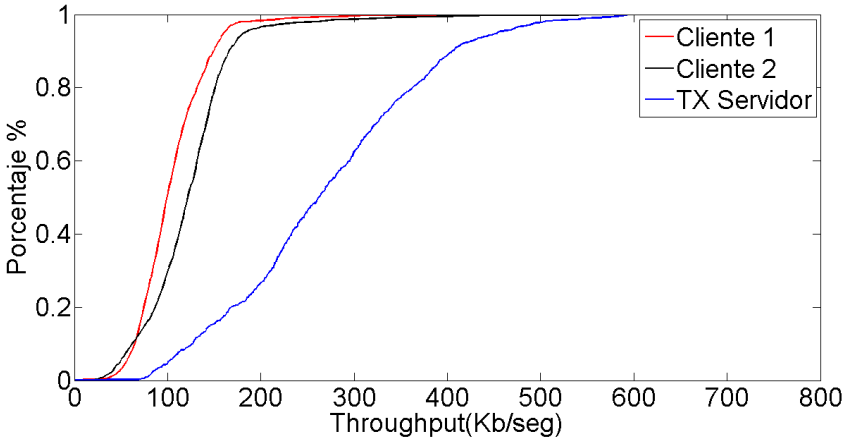


Figura 35. Throughputs – CDF

En la Figura35 se observan las CDFs pudiéndose observar fácilmente que el throughput en recepción del cliente 1 es menor que el del cliente 2, debido a la mayor pérdida de paquetes producida en el camino desde el servidor hasta el cliente 1.

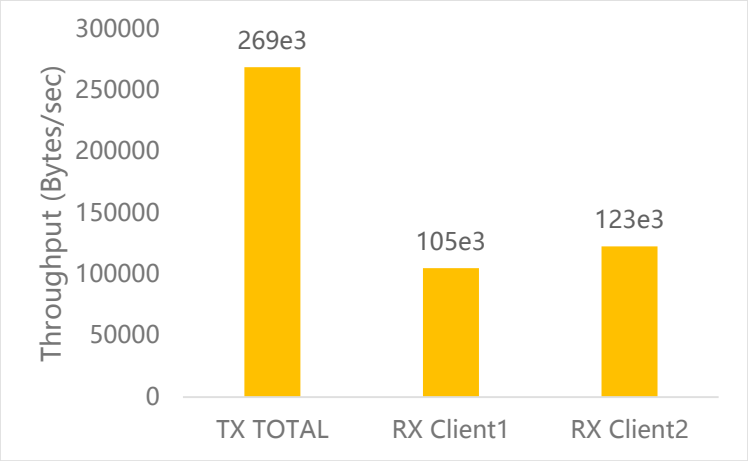


Figura 36. Valores promedio de rendimiento.

En la Figura36 se representan los valores promedio de rendimiento tanto en el servidor (transmisión), como en los clientes (recepción), por lo que sirve para afianzar los datos representados en la Figura35.

CAPTURAS DE PANTALLA

En la Figura37 y Figura38 se han representado capturas representativas de ambos streaming, mostrando que el vídeo tiene mejor calidad en el cliente 2 que en el cliente 1.

También se puede observar que hay fases en las que la calidad del vídeo es peor, y correspondiéndose con los valores mayores de jitter y pérdida de paquetes, mientras que otras veces el servicio es adecuado.

Cabe destacar además que la calidad del vídeo mejora a medida que pasa el tiempo, debido al buffering utilizado por VLC, que permite paliar los efectos del jitter de los paquetes.

En Figura37 y Figura38 se pueden diferenciar los clientes 1 y 2, a la izquierda y derecha de la pantalla respectivamente.

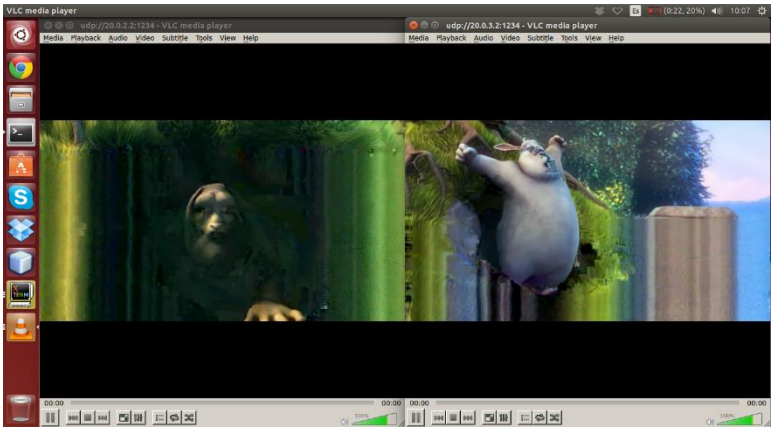


Figura 37. Captura 1 – Simulación completa

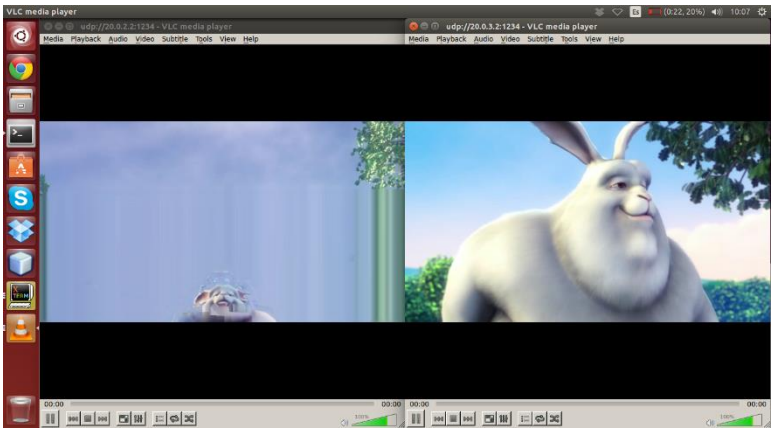


Figura 38. Captura 2 – Simulación completa

CONCLUSIONES

El objetivo de este proyecto es conocer las limitaciones del simulador de redes ns-3 con nodos virtuales que envían tráfico de aplicaciones reales. Lo que se puede conocer gracias a los resultados de las simulaciones. Estos límites solo se corresponden con un ordenador de características similares al mencionado anteriormente.

Uno de los límites se produce al pasar de cuatro contenedores activos, transmitiéndose vídeo por ns-3. Por lo que la limitación indica una carencia de velocidad de proceso en el ordenador. Además de esta limitación, se producen estancamientos en la red cuando trabaja con un gran número de paquetes, lo que limita el rendimiento del simulador.

En cuanto a los contenedores de Linux, se puede decir que permiten realizar una simulación muy completa de un dispositivo, y facilitan en gran medida la emulación realizada con el ns-3 y aplicaciones reales.

Se puede concluir que, el ns-3 con contenedores, usando un ordenador personal, puede ser utilizado para emular redes con un cierto grado de complejidad y no muchos contenedores activos simultáneamente. A la hora de emular redes de mayor complejidad y un número de contenedores mayor, como podría ser el caso de un servicio de vídeo bajo demanda para un número elevado de clientes, se necesitarían equipos con mayor potencia de procesado, pudiéndose trabajar con varios equipos en paralelo. En ambos casos, representa un beneficio muy interesante para probar al comportamiento de técnicas, protocolos, algoritmos y aplicaciones con la finalidad de funcionar sobre servicios reales.

Debido a los resultados obtenidos y al temario del trabajo, se han presentado dos artículos a dos conferencias, la JITEL (ámbito nacional) y la MONAMI (ambito internacional). Habiendo sido aceptado el artículo para la JITEL y quedando pendiente de calificación el enviado a la MONAMI.

LÍNEAS FUTURAS

QOE

Ya se ha comentado la importancia que tiene la QoE en servicios como el streaming de vídeo y, en general, en cualquier aplicación que implique cierta interacción con el usuario.

Medir la QoE no es una tarea sencilla y por eso en muchas ocasiones se recurre al MOS, que consiste en un test, que se ha estado llevando a cabo durante décadas en servicios como la telefonía, para recoger información sobre la opinión subjetiva del usuario acerca de la calidad del servicio.

Este test, por tanto, requiere de intervención humana para completarse, y es subjetivo. Sería muy interesante desarrollar una metodología para medir la QoE que un usuario experimenta, de una manera más sistemática y objetiva.

Con esto, se tendría una idea mucho mejor de cómo afectan los distintos tipos de redes y topologías a la manera en la que los usuarios perciben el servicio, y permitiría, junto con la QoS, el desarrollo de mejores redes y protocolos para los usuarios.

LTE

LTE, es una abreviación de Long-Term Evolution, es un estándar de comunicación inalámbrica de alta velocidad que aparece como la evolución natural de las comunicaciones celulares (GSM y UMTS). Gracias al LTE los usuarios pueden disponer de velocidades de hasta 100Mbps en sus terminales, y las previsiones le auguran una presencia muy notable.

Debido a esto, una de las líneas de futuro a partir de la propuesta presentada en esta memoria consiste en analizar el comportamiento del NS3 simulando redes LTE, utilizando contenedores y aplicaciones reales.

DEMOSTRACIÓN

INTRODUCCION

Con esta demonstración se quiere dar a conocer el trabajo desde un punto de vista práctico y dinámico. En la Figura39 se puede ver la topología de la red con la que se van a realizar dos streamings simultáneos.

El enlace hasta el cliente 2 es ideal, mientras que para el cliente 1 uno de los dos enlaces tiene un error que se modificará durante la demostración y el otro es ideal.

Durante la simulación se podrá ver que la calidad del video empeora en ambos casos a medida que se aumenta el error del enlace. Esto se debe a:

1. El enlace hacia el cliente 1 tiene errores, hay paquetes que se pierden y retransmisiones.

2. En el cliente 2 la calidad del video se ve afectada porque todos los enlaces comparten canal, y se dividen los 5Mbits/segundo que puede haber de throughput total, por lo que se produce contienda entre los paquetes.

Tanto el escenario cómo el script BASH se pueden encontrar en el apéndice tal y como se han usado en la demostración.

ESQUEMA ESCENARIO

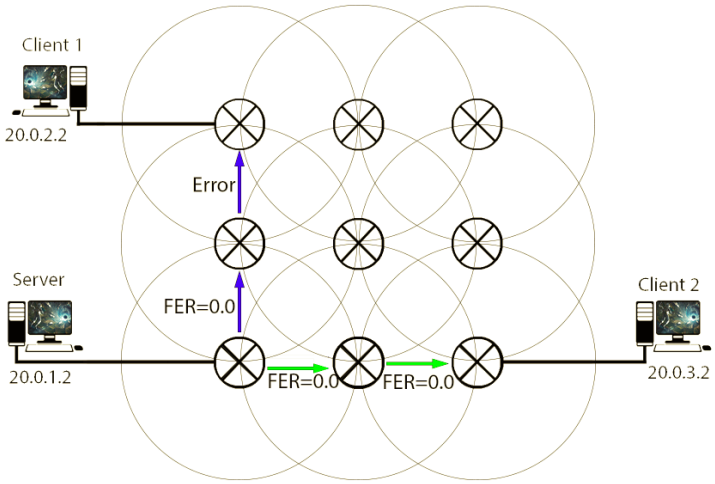


Figura 39. Esquema demonstración.

APÉNDICE

CONFIGURACIÓN LXC

CONFIGLXC.SH

```
#!/bin/bash
#Installing the required packages
######
#First update the host OS
######
sudo apt-get update
sudo apt-get dist-upgrade
######
#Required packages to run LXC containers.
######
sudo apt-get install lxc systemd-services uidmap
############################
#This is not for the container but for the ns3 to use the containers
#The bridge-utils package contains an utility needed to create and manage
bridge devices
#uml-utilities: User-mode Linux is a port of the Linux kernel to its own
system call #interface. It provides a kind of virtual machine, which runs
Linux as a user process #under another Linux kernel.
######
sudo apt-get install bridge-utils uml-utilities
############################
######
#Assigning myself a set of uids and gids (just like with unprivileged
containers). In case
#unprivileged containers would be used in a future.
######
sudo usermod --add-subuids 100000-165536 $USER
sudo usermod --add-subgids 100000-165536 $USER
sudo chmod +x $HOME
######
#Create the directory ~/.config/lxc/ if it is not created yet, and in it, the
file default.conf which #is going to be the configuration file to create
containers.
######
mkdir -p ~/.config/lxc/
cat > ~/.config/lxc/default.conf << EOF
######
#lxc.network.type: specify what kind of network virtualization to be used for
the container.
#It has the option VETH, which means that a virtual ethernet pair device is
created with one #side assigned to the container and the other side attached
to a bridge specified by the #lxc.network.link option.
######
lxc.network.type = veth
lxc.network.link = lxcbr0
######
#lxc.network.flags = up, activates the network interface.
######
lxc.network.flags = up
```

```
######
#lxc.network.hwaddr is normally random allocated but in some cases is
necessary. Any x #situated here will be replaced by random value.
######
lxc.network.hwaddr = 00:16:3e:xx:xx:xx
######
#id_map is used to start the container in a private user namespace with user
and group id #mappings.
######
lxc.id_map = u 0 100000 65536
lxc.id_map = g 0 100000 65536
EOF

echo "Now type this and it'll be done."
echo "echo \"juan veth lxcbr0 10\" > /etc/lxc/lxc-usernet"

sudo su
```

FIRSTCONTAINER.SH

```
#!/bin/bash
############
#Name the container will have
############
name=firstcontainer
############
#It is necessary to be root to create privileged containers.
#This if checks if the user is root.
############
if [ $(id -u) -eq 0 ]
then

echo "Creating container.."
############
#Creation of the container. -t : template, indicates that is going to be an
ubuntu template and #-a: architecture i386, indicates that the architecture
of the container will be 32bits.
############
lxc-create -t ubuntu -n $name -- -a i386
echo "Done."
############
#Creating shared folder and applying permissions with the setuid, setgid, and
sticky flags.
#setuid: access to user id
#setgid: access to group id
#sticky: protects the files of a directory
############
mkdir -p /home/juan/project/shared
chmod 7777 /home/juan/project/shared
############
#Mounting shared libraries and folders so the container can use the shared
files
#and applications.
############
echo "Modifying container.."
```

```
echo "lxc.mount.entry = /dev/dri dev/dri none bind,optional,create=dir
lxc.mount.entry = /dev/snd dev/snd none bind,optional,create=dir
lxc.mount.entry = /tmp/.X11-unix tmp/.X11-unix none bind,optional,create=dir
lxc.mount.entry = /dev/video0 dev/video0 none bind,optional,create=file
lxc.mount.entry = /home/juan/project/shared root none bind.ro 0.0
echo "Done."
###########
#This script will tell pulseaudio on the host to bind
/home/ubuntu/.pulse_socket in the #container. And giving executable
permissions to it.
###########
echo "Adding setup-pulse.sh to the container's folder.."
cp /home/juan/project/setup-pulse.sh /var/lib/lxc/$name/setup-pulse.sh
chmod +x /var/lib/lxc/$name/setup-pulse.sh
echo "Done."
###########
#With this the container will be updated and will have the applications we
need for the #project
###########UPDATED###########
lxc-start -n $name -d
lxc-attach -n $name -- umount /tmp/.X11-unix
lxc-attach -n $name -- apt-get update
lxc-attach -n $name -- apt-get dist-upgrade -y
lxc-attach -n $name -- apt-get install wget ubuntu-artwork dmz-cursor-theme
ca-certificates pulseaudio -y
########################
######## APPS ##################
#lxc-attach -n $name -- dpkg -i /root/google.deb
lxc-attach -n $name -- dpkg -i /root/vlc.deb
lxc-attach -n $name -- apt-get -f install -y
lxc-attach -n $name -- apt-get install iperf
lxc-attach -n $name -- sudo -u ubuntu mkdir -p /home/ubuntu/.pulse/
echo "disable-shm=yes" | lxc-attach -n $name -- sudo -u ubuntu tee
/home/ubuntu/.pulse/client.conf
##############################
lxc-stop -n $name
echo "Container created, you can clone it."

else
echo "You have to be root"
fi
```

CREATECONTAINER.SH

```
#!/bin/bash
# $1 name
# $2 ip
# $3 gateway
######
name=$1
```

```
ip=$2
gwip=$3
############
#It is necessary to be root to create privileged containers.
#This if checks if the user is root.
############
if [ $(id -u) -eq 0 ]
then
############
#clone the (-o) original container “firstcontainer” in (-n) the new container
$name
############
echo "Creating container.."
lxc-clone -o firstcontainer -n $name
echo "Done."
############
#sed -i ‘/abc/d’ file: deletes all lines containing ‘abc’ in file
#echo “abc” >> file: adds abc at the end of the file
#So with this what we get is to put a new ipv4 and a new gateway in the
container’s #configuration file.
############
echo "Modifying container.."
sed -i '/lxc.network.ipv4/d' /var/lib/lxc/$name/config
sed -i '/lxc.network.ipv4.gateway/d' /var/lib/lxc/$name/config
echo "lxc.network.ipv4 = $ip/24" >>/var/lib/lxc/$name/config
echo "lxc.network.ipv4.gateway = $gwip">>/var/lib/lxc/$name/config
echo "Done."
############
#Creating the tap device the ns3 will use to get the packets in and out the
container.
#And adding the tap device to the bridge “lxcbr0” which is already created in
the system.
############
echo "Creating the tap device the ns3 will use to get the packets from the
container."
sudo tunctl -t tap-$name
sudo ifconfig tap-$name 0.0.0.0 promisc up
sudo brctl addif lxcbr0 tap-$name
echo "Done."
############
#Starts again with the custom configuration
lxc-start -n $name -d
lxc-stop -n $name
lxc-start -n $name -d
echo "Container running in background.."

else
   echo "You need to be root."
   sudo ./createContainer.sh $1 $2 $3
fi
############
#End of createContainer.sh script
############
```

SETUP-PULSE.SH

```
#!/bin/sh
PULSE_PATH=$LXC_ROOTFS_PATH/home/ubuntu/.pulse_socket
if [ ! -e "$PULSE_PATH" ] || [ -z "$(lsof -n $PULSE_PATH 2>&1)" ]; then
    pactl load-module module-native-protocol-unix auth-anonymous=1 \
        socket=$PULSE_PATH
fi
```

APLICACIONES

EXECUTE.SH

```
#!/bin/bash
# $1 Container's name
# Commands to execute a program with the graphic mode in the container
CONTAINER=cont${1}
IPDST=$2
PULSE_SOCKET=/home/ubuntu/.pulse_socket
lxc-attach --clear-env -n $CONTAINER -- sudo -u ubuntu -i env QT_X11_NO_MITSHM=1 DISPLAY=$DISPLAY PULSE_SERVER=$PULSE_SOCKET vlc file:///root/video.avi --sout="#transcode{vcodec=h264,fps=20,scale=Auto,acodec=mpga,ab=128,channels=2,samplerate=44100}:udp{dst=$IPDST:1234}"
```

CLI-EXECUTE.SH

```
#!/bin/bash
# $1 Container's name
# Commands to execute a program with the graphic mode in the container
CONTAINER=cont${1}
PULSE_SOCKET=/home/ubuntu/.pulse_socket
lxc-attach --clear-env -n $CONTAINER -- sudo -u ubuntu -i env QT_X11_NO_MITSHM=1 DISPLAY=$DISPLAY PULSE_SERVER=$PULSE_SOCKET vlc udp://@20.0.$1.2:1234
```

INTERCONEXIÓN

ATTACHTAP.SH

```
#!/bin/bash
```

```
vector=`ls /var/lib/lxc`  #The contents of the folder go into a variable
#It's not an array, so:
nelements=`ls /var/lib/lxc | wc -l` #Number of elements in folder
for i in `seq 1 $nelements`; do
      var=$(echo $vector | awk "{print \$$i}")
      if [ $var == "firstcontainer" ] || [ $var == "config" ] || [ $var ==
"setup-pulse.sh" ]
      then
            echo "----"
            #NOTHING CAUSE WE DON?T WANT THEM
            #WE JUST WANT THE CONTAINERS
      else
            echo $var
            sudo tunctl -t tap-$var
            sudo ifconfig tap-$var 0.0.0.0 promisc up
            sudo brctl addif lxcbr0 tap-$var
      fi
done
echo "Now all the containers have their own tap device and they're connected
to the bridge lxcbr0"
brctl show
```

STARTCONTBG.SH

```
#!/bin/bash
############################
# This script starts the containers between cont1 and
# contX
############################
number=$1 #number of containers to start in background
if [ $(id -u) -eq 0 ]
then
      for i in `seq 1 $number`; do
            lxc-start -n cont$i -d
      done
      lxc-ls --active
else
      echo "Be root"
```

```
fi
```

ESTADÍSTICAS

REALTIME-SIMULATOR-IMPL.CC

```

```

TAP-BRIDGE.CC

SIMULACIONES

SIMULACIÓN SIN CONTENEDORES

ESCENARIO

```
#include <iostream>
#include <fstream>
#include <string.h>
#include <stdlib.h>
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/mobility-module.h"
#include "ns3/wifi-module.h"
#include "ns3/internet-module.h"
#include "ns3/CSMA-module.h"
#include "ns3/applications-module.h"
#include "src/internet/helper/internet-stack-helper.h"
#include "src/core/model/simulator.h"
#include "src/network/helper/node-container.h"
#include "src/CSMA/helper/CSMA-helper.h"
#include "src/network/helper/net-device-container.h"
#include "src/internet/helper/ipv4-address-helper.h"
#include "src/internet/helper/ipv4-static-routing-helper.h"
#include "src/internet/model/ipv4-static-routing.h"
#include "src/network/utils/ipv4-address.h"
#include "src/core/model/global-value.h"
```

```
#include "src/core/model/string.h"
#include "src/core/model/config.h"
#include "src/aodv/helper/aodv-helper.h"
#include "src/wifi/helper/yans-wifi-helper.h"
#include "src/network/helper/trace-helper.h"
#include "src/applications/helper/on-off-helper.h"
#include "src/network/helper/application-container.h"
#include "src/applications/helper/packet-sink-helper.h"
using namespace ns3;
int na;
int nf;
int ns;
int numfluj;
void Foo(std::string i) {
std::stringstream sstm;
// Writing results in a file
sstm.str("");
sstm.clear();
sstm << "/home/juan/project/stats/nocontsimulation/" << na << "-" << nf << "-
" << ns << "-" << numfluj << ".txt";
std::string ss=sstm.str();
std::ofstream myfile;
myfile.open(ss.c_str(), std::ofstream::app);
myfile << i << "\n";
myfile.close();
sstm.str("");
sstm.clear();
////////////////////////////////////
}
int main(int argc, char *argv[]) {
int nadd=20;    //Additional nodes in which there is background traffic
int range=5; //Distance range of the node
int nsim=0;  //Simulation number, just affects to the file name
int tsim=20; //Simulation time, after this it stops
int nflujos=2;
std::string timevar = "100ms";
CommandLine cmd;
```

```
cmd.AddValue("nadd", "Number of additional nodes", nadd);
cmd.AddValue("range", "Range of routers", range);
cmd.AddValue("nsim", "Simulation number", nsim);
cmd.AddValue("timevar", "Hard limit time", timevar);
cmd.AddValue("tsim", "Simulation stop time", tsim);
cmd.AddValue("nflujos", "Number of data flows in network", nflujos);
cmd.Parse(argc, argv);
na=nadd;
ns=nsim;
numfluj=nflujos;
//Time::SetResolution (Time::NS);
GlobalValue::Bind("SimulatorImplementationType",
StringValue("ns3::RealtimeSimulatorImpl"));
GlobalValue::Bind("ChecksumEnabled", BooleanValue(true));
Config::SetDefault("ns3::RealtimeSimulatorImpl::SynchronizationMode",
StringValue("HardLimit"));
// Default time difference is 100 ms
Config::SetDefault("ns3::RealtimeSimulatorImpl::HardLimit",
TimeValue(Time(timevar)));
ns3::Config::SetDefault("ns3::RangePropagationLossModel::MaxRange",
ns3::DoubleValue(range));
Simulator::GetImplementation()->TraceConnectWithoutContext("JitterAndTime",
MakeCallback(&Foo));
NodeContainer routersadd;
routersadd.Create(nadd);
AodvHelper aodv;
InternetStackHelper routersaddstack;
routersaddstack.SetRoutingHelper(aodv);
routersaddstack.Install(routersadd);
NodeContainer nodes; //Before the for
nodes.Add(routersadd);
Ptr<ListPositionAllocator>                  positionAlloc                =
CreateObject<ListPositionAllocator > ();
int init=0;
int nfiles=nadd/5;
for(init=0;init<(nfiles);init++){
positionAlloc->Add(Vector(init*range, 0.0, 0.0));
positionAlloc->Add(Vector(init*range, init*range, 0.0));
```

```
positionAlloc->Add(Vector(init*range, 2*init*range, 0.0));
positionAlloc->Add(Vector(init*range, 3*init*range, 0.0));
positionAlloc->Add(Vector(init*range, 4*init*range, 0.0));
}
//WIFI
////////////////////////////////////////////////////////////////////
WifiHelper wifi = WifiHelper::Default();
wifi.SetStandard(WIFI_PHY_STANDARD_80211b);
wifi.SetRemoteStationManager("ns3::ConstantRateWifiManager",     "DataMode",
StringValue("DsssRate11Mbps"));
NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default();
wifiMac.SetType("ns3::AdhocWifiMac");
// Configure the physcial layer.
YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default();
YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default();
wifiChannel.SetPropagationDelay("ns3::ConstantSpeedPropagationDelayModel");
wifiChannel.AddPropagationLoss("ns3::RangePropagationLossModel");
wifiPhy.SetChannel(wifiChannel.Create());
// Install the wireless devices onto our ghost nodes.
NetDeviceContainer devices = wifi.Install(wifiPhy, wifiMac, nodes);
// We need location information since we are talking about wifi
MobilityHelper mobility;
mobility.SetPositionAllocator(positionAlloc);
mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
mobility.Install(nodes);
Ipv4AddressHelper ipv4;
ipv4.SetBase("10.0.0.0", "255.255.0.0");
Ipv4InterfaceContainer interfaces = ipv4.Assign(devices);
//JAAR
std::stringstream sf;
std::string varonoff;
std::string varsink;
int port1=9;
for(init=0;init<nflujos;init++){
sf << "onoff" << init;
varonoff=sf.str();
```

```
OnOffHelper varonoff ("ns3::UdpSocketFactory", InetSocketAddress (interfaces.GetAddress (init*5 ), port1));
sf.str("");
sf.clear();
varonoff.SetConstantRate (DataRate ("11Mbps"));
ApplicationContainer apps = varonoff.Install (nodes.Get ((init*5)+4));
apps.Start (Seconds (0.0));
apps.Stop (Seconds (tsim));
sf << "sink" << init;
varsink=sf.str();
PacketSinkHelper varsink ("ns3::UdpSocketFactory", InetSocketAddress (Ipv4Address::GetAny (),port1));
sf.str("");
sf.clear();
varsink.Install (nodes.Get (init*5));
apps.Start (Seconds (0.0));
apps.Stop (Seconds (tsim));
}
wifiPhy.EnablePcapAll ("NOCONT",devices.Get(0));
Simulator::Stop(Seconds(tsim));
//    Simulator::Stop();
Simulator::Run();
Simulator::Destroy();
}
```

SCRIPT

```
#!/bin/bash
nadd=300 #number of additional nodes
na_step=50  #Step for the loop
na_start=50
timesim=15
nsim=20
mkdir -p stats/nocontsimulation
cd ns-allinone-3.21/ns-3.21/
./waf build
```

```
for i in `seq $na_start $na_step $nadd`; do
for j in `seq 2 2 8`; do # flujos
for k in `seq 0 $nsim`; do
echo ----------------------------
echo NA - FLUJO - SIM
echo ${i}-${j}-${k}
echo ----------------------------
./waf --run="nocontsim --nadd=$i --tsim=$timesim --nflujos=$j --nsim=$k --
range=15"
done
done
done
```

EFECTO CONTENEDORES

ESCENARIO

```
#include <iostream>
#include <fstream>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <sstream>
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/mobility-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/wifi-module.h"
#include "ns3/internet-module.h"
#include "ns3/CSMA-module.h"
#include "ns3/applications-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/ipv4-static-routing-helper.h"
#include "ns3/tap-bridge-module.h"
#include "src/internet/helper/internet-stack-helper.h"
```

```
#include "src/core/model/simulator.h"
#include "src/network/helper/node-container.h"
#include "src/CSMA/helper/CSMA-helper.h"
#include "src/network/helper/net-device-container.h"
#include "src/internet/helper/ipv4-address-helper.h"
#include "src/internet/helper/ipv4-static-routing-helper.h"
#include "src/internet/model/ipv4-static-routing.h"
#include "src/network/utils/ipv4-address.h"
#include "src/tap-bridge/model/tap-bridge.h"
#include "src/core/model/global-value.h"
#include "src/core/model/string.h"
#include "src/core/model/config.h"
#include "src/aodv/helper/aodv-helper.h"
#include "src/wifi/helper/yans-wifi-helper.h"
#include "src/network/helper/trace-helper.h"
#include "ns3/error-model.h"
using namespace ns3;
int ns;
int nc;
void F_tx(uint64_t i){
//    std::cout << "Throughput TX " << i << std::endl;
std::stringstream sstm1;
// Writing results in a file
sstm1.str("");
sstm1.clear();
sstm1  <<  "/home/juan/project/stats/mul_stream_0_router/sender/throughput-"
<< nc << "-" << ns << ".txt";
std::string ss1=sstm1.str();
std::ofstream myfile1;
myfile1.open(ss1.c_str(), std::ofstream::app);
myfile1 << i << "\n";
myfile1.close();
sstm1.str("");
sstm1.clear();
}
void F_rx(std::string i){
//    std::cout << "Throughput RX " << i << std::endl;
```

```
std::stringstream sstm1;
// Writing results in a file
sstm1.str("");
sstm1.clear();
sstm1 << "/home/juan/project/stats/mul_stream_0_router/receiver/throughput-"
<< nc << "-" << ns << ".txt";
std::string ss1=sstm1.str();
std::ofstream myfile1;
myfile1.open(ss1.c_str(), std::ofstream::app);
myfile1 << i << "\n";
myfile1.close();
sstm1.str("");
sstm1.clear();
}
void Foo1(std::string i) { //CONT 1
//    std::cout << "tap1 " << i << std::endl;
std::stringstream sstm1;
// Writing results in a file
sstm1.str("");
sstm1.clear();
sstm1 << "/home/juan/project/stats/mul_stream_0_router/sender/" << nc << "-"
<< ns << ".txt";
std::string ss1=sstm1.str();
std::ofstream myfile1;
myfile1.open(ss1.c_str(), std::ofstream::app);
myfile1 << i << "\n";
myfile1.close();
sstm1.str("");
sstm1.clear();
}
void Foo2(std::string i) { //CONT 2
//    std::cout << "tapX rx " << i << std::endl;
std::stringstream sstm2;
// Writing results in a file
sstm2.str("");
sstm2.clear();
```

```
sstm2 << "/home/juan/project/stats/mul_stream_0_router/receiver/" << nc << "-
" << ns << ".txt";
std::string ss2=sstm2.str();
std::ofstream myfile2;
myfile2.open(ss2.c_str(), std::ofstream::app);
myfile2 << i << "\n";
myfile2.close();
sstm2.str("");
sstm2.clear();
}
void Foo(std::string i) {
std::stringstream sstm;
// Writing results in a file
sstm.str("");
sstm.clear();
sstm << "/home/juan/project/stats/mul_stream_0_router/ns3jitter/" << nc << "-
" << ns << ".txt";
std::string ss=sstm.str();
std::ofstream myfile;
myfile.open(ss.c_str(), std::ofstream::app);
//      myfile << tv << " " << (double(i) / 1e6) << "\n";
myfile << i << "\n";
myfile.close();
sstm.str("");
sstm.clear();
////////////////////////////////////
}
int main(int argc, char *argv[]) {
//    int nrouters=0; //Number of jumps between the containers
int range=15; //Distance range of the node
int tsim=100; //Simulation time, after this it stops
int nsim=1111;
int ncont=2;
std::string timevar = "100ms";
CommandLine cmd;
//    cmd.AddValue("nrouters", "Number of Routers", nrouters);
cmd.AddValue("ncont", "Number of Containers", ncont);
```

```
cmd.AddValue("range", "Range of routers", range);
cmd.AddValue("timevar", "Hard limit time", timevar);
cmd.AddValue("tsim", "Simulation stop time", tsim);
cmd.AddValue("nsim", "Simulation stop time", nsim);
cmd.Parse(argc, argv);
ns=nsim;
nc=ncont;
//Time::SetResolution (Time::NS);
GlobalValue::Bind("SimulatorImplementationType",
StringValue("ns3::RealtimeSimulatorImpl"));
GlobalValue::Bind("ChecksumEnabled", BooleanValue(true));
Config::SetDefault("ns3::RealtimeSimulatorImpl::SynchronizationMode",
StringValue("HardLimit"));
// Default time difference is 100 ms
Config::SetDefault("ns3::RealtimeSimulatorImpl::HardLimit",
TimeValue(Time(timevar)));
ns3::Config::SetDefault("ns3::RangePropagationLossModel::MaxRange",
ns3::DoubleValue(range));
Simulator::GetImplementation()->TraceConnectWithoutContext("JitterAndTime",
MakeCallback(&Foo));
////////////////////////////////
NodeContainer nodeGW;
nodeGW.Create(ncont);
NodeContainer nodes;
TapBridgeHelper tapBridge;
tapBridge.SetAttribute("Mode", StringValue("UseBridge"));
//installation inside the for
int init;
std::string noderes;
std::string stackres;
std::string CSMAres;
std::string devres;
std::string addressres;
std::string ipres;
std::string interres;
std::string tapres;
std::stringstream sstm;
std::stringstream sstmptr;
```

```
std::string tappointer;
for (init = 1; init <= ncont; init++) {
//        printf("Round %i\n", init);
//Creating node containers
sstm << "nodescont" << init;
noderes = sstm.str();
//        std::cout << "Nodes: " << noderes << std::endl;
NodeContainer noderes;
noderes.Add(nodeGW.Get(init-1));
noderes.Create(1);
sstm.str("");
sstm.clear(); //Clear state flags.
//Installing internet stacks
sstm << "stack" << init;
stackres = sstm.str();
//        std::cout << "Stack: " << stackres << std::endl;
InternetStackHelper stackres;
stackres.Install(noderes.Get(0));
sstm.str("");
sstm.clear(); //Clear state flags.
//CSMA helper
sstm << "CSMA" << init;
CSMAres = sstm.str();
//        std::cout << "CSMA: " << CSMAres << std::endl;
CSMAHelper CSMAres;
CSMAres.SetChannelAttribute("DataRate", DataRateValue(DataRate(20000000)));
CSMAres.SetChannelAttribute("Delay", TimeValue(MilliSeconds(0)));
sstm.str("");
sstm.clear(); //Clear state flags.
//Create a device container to hold the nodes that belong to the subnetwork
sstm << "dev" << init;
devres = sstm.str();
//        std::cout << "Devices: " << devres << std::endl;
NetDeviceContainer devres = CSMAres.Install(noderes);
sstm.str("");
sstm.clear(); //Clear state flags.
```

```
//Configure the subnet address and mask
sstm << "address" << init;
addressres = sstm.str();
//        std::cout << "Address: " << addressres << std::endl;
Ipv4AddressHelper addressres;
sstm.str("");
sstm.clear(); //Clear state flags.
sstm << "20.0." << init << ".0";
ipres = sstm.str();
//        std::cout << "Ip: " << ipres << std::endl;
addressres.SetBase(ipres.c_str(), "255.255.255.0");
sstm.str("");
sstm.clear(); //Clear state flags.
//Create an interface container to hold the ipv4 interfaces created
sstm << "inter" << init;
interres = sstm.str();
//        std::cout << "Interfaces: " << interres << std::endl;
Ipv4InterfaceContainer interres = addressres.Assign(devres.Get(0));
sstm.str("");
sstm.clear(); //Clear state flags.
//NodeContainer nodes; //Before the for
nodes.Add(noderes.Get(0));
//Installing tap-devices
sstm << "tap-cont" << init;
tapres = sstm.str();
//        std::cout << "Tap-devices: " << tapres << std::endl;
tapBridge.SetAttribute("DeviceName", StringValue(tapres));
sstmptr << "tap" << init;
tappointer=sstmptr.str();
Ptr<NetDevice> tappointer= tapBridge.Install(noderes.Get(1), devres.Get(1));
if(init%2!=0){
if(init==1){
tappointer->TraceConnectWithoutContext("PktTX", MakeCallback(&Foo1));
tappointer->TraceConnectWithoutContext("ThroughputTx", MakeCallback(&F_tx));
}else{}
}else{
```

```
//            std::cout << "aaa " << init << std::endl;
tappointer->TraceConnectWithoutContext("PktRX", MakeCallback(&Foo2));
tappointer->TraceConnectWithoutContext("ThroughputRx", MakeCallback(&F_rx));
}
sstmptr.str("");
sstmptr.clear();
sstm.str("");
sstm.clear(); //Clear state flags.
}
Ptr<ListPositionAllocator>                  positionAlloc                  =
CreateObject<ListPositionAllocator > ();
positionAlloc->Add(Vector(0, 0, 0));
positionAlloc->Add(Vector(15, 0, 0));
if(ncont>=4){
positionAlloc->Add(Vector(0, 60, 0));
positionAlloc->Add(Vector(15, 60, 0));
if(ncont>=6){
positionAlloc->Add(Vector(0, 120, 0));
positionAlloc->Add(Vector(15, 120, 0));
if(ncont==8){
positionAlloc->Add(Vector(0, 180, 0));
positionAlloc->Add(Vector(15, 180, 0));
}
else{}
}
else{}
}
else{}
//WIFI
/////////////////////////////////////////////////////////////////////
WifiHelper wifi = WifiHelper::Default();
wifi.SetStandard(WIFI_PHY_STANDARD_80211b);
wifi.SetRemoteStationManager("ns3::ConstantRateWifiManager",     "DataMode",
StringValue("DsssRate11Mbps"));
NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default();
wifiMac.SetType("ns3::AdhocWifiMac");
```

```
// Configure the physical layer.
YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default();
YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default();
wifiChannel.SetPropagationDelay("ns3::ConstantSpeedPropagationDelayModel");
wifiChannel.AddPropagationLoss("ns3::RangePropagationLossModel");
//ERROR FER
Ptr<MatrixErrorModel> error = CreateObject<MatrixErrorModel> ();
//    std::cout << nodes.Get(0)->GetId() << " " << nodes.Get(1)->GetId() <<
std::endl;
error->SetFer (nodes.Get(0)->GetId() ,nodes.Get(0)->GetId(), 0.0);
error->SetFer (nodes.Get(1)->GetId() ,nodes.Get(1)->GetId(), 0.0);
error->SetFer (nodes.Get(1)->GetId() ,nodes.Get(0)->GetId(), 0.0);
error->SetFer (nodes.Get(0)->GetId() ,nodes.Get(1)->GetId(), 0.0);
//    error->SetDefaultFer (0.0);
wifiPhy.SetErrorModel(error);
wifiPhy.SetChannel(wifiChannel.Create());
// Install the wireless devices onto our ghost nodes.
NetDeviceContainer devices = wifi.Install(wifiPhy, wifiMac, nodes);
// We need location information since we are talking about wifi
MobilityHelper mobility;
mobility.SetPositionAllocator(positionAlloc);
mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
mobility.Install(nodes);
Ipv4AddressHelper ipv4;
ipv4.SetBase("10.0.0.0", "255.255.0.0");
Ipv4InterfaceContainer interfaces = ipv4.Assign(devices);
Ptr<Ipv4> ipv4A = nodes.Get(0)->GetObject<Ipv4> ();
Ptr<Ipv4> ipv4B = nodes.Get(1)->GetObject<Ipv4> ();
//    Static Routing
Ipv4StaticRoutingHelper staticRoutingHelper;
Ptr<Ipv4StaticRouting>                    staticRoutingA                    =
staticRoutingHelper.GetStaticRouting(ipv4A);
staticRoutingA-
>AddHostRouteTo(Ipv4Address("20.0.2.2"),Ipv4Address("10.0.0.2"),2);
staticRoutingA-
>AddHostRouteTo(Ipv4Address("20.0.1.2"),Ipv4Address("20.0.1.2"),1);
Ptr<Ipv4StaticRouting>                    staticRoutingB                    =
staticRoutingHelper.GetStaticRouting(ipv4B);
```

```
staticRoutingB-
>AddHostRouteTo(Ipv4Address("20.0.2.2"),Ipv4Address("20.0.2.2"),1);

staticRoutingB-
>AddHostRouteTo(Ipv4Address("20.0.1.2"),Ipv4Address("10.0.0.1"),2);

if(ncont>=4){

Ptr<Ipv4> ipv4C = nodes.Get(2)->GetObject<Ipv4> ();

Ptr<Ipv4> ipv4D = nodes.Get(3)->GetObject<Ipv4> ();

Ptr<Ipv4StaticRouting>                    staticRoutingC                    =
staticRoutingHelper.GetStaticRouting(ipv4C);

staticRoutingC-
>AddHostRouteTo(Ipv4Address("20.0.4.2"),Ipv4Address("10.0.0.4"),2);

staticRoutingC-
>AddHostRouteTo(Ipv4Address("20.0.3.2"),Ipv4Address("20.0.3.2"),1);

Ptr<Ipv4StaticRouting>                    staticRoutingD                    =
staticRoutingHelper.GetStaticRouting(ipv4D);

staticRoutingD-
>AddHostRouteTo(Ipv4Address("20.0.4.2"),Ipv4Address("20.0.4.2"),1);

staticRoutingD-
>AddHostRouteTo(Ipv4Address("20.0.3.2"),Ipv4Address("10.0.0.3"),2);

if(ncont>=6){

Ptr<Ipv4> ipv4E = nodes.Get(4)->GetObject<Ipv4> ();

Ptr<Ipv4> ipv4F = nodes.Get(5)->GetObject<Ipv4> ();

Ptr<Ipv4StaticRouting>                    staticRoutingE                    =
staticRoutingHelper.GetStaticRouting(ipv4E);

staticRoutingE-
>AddHostRouteTo(Ipv4Address("20.0.6.2"),Ipv4Address("10.0.0.6"),2);

staticRoutingE-
>AddHostRouteTo(Ipv4Address("20.0.5.2"),Ipv4Address("20.0.5.2"),2);

Ptr<Ipv4StaticRouting>                    staticRoutingF                    =
staticRoutingHelper.GetStaticRouting(ipv4F);

staticRoutingF-
>AddHostRouteTo(Ipv4Address("20.0.6.2"),Ipv4Address("20.0.6.2"),1);

staticRoutingF-
>AddHostRouteTo(Ipv4Address("20.0.5.2"),Ipv4Address("10.0.0.5"),2);

if(ncont>=8){

Ptr<Ipv4> ipv4G = nodes.Get(6)->GetObject<Ipv4> ();

Ptr<Ipv4> ipv4H = nodes.Get(7)->GetObject<Ipv4> ();

Ptr<Ipv4StaticRouting>                    staticRoutingG                    =
staticRoutingHelper.GetStaticRouting(ipv4G);

staticRoutingG-
>AddHostRouteTo(Ipv4Address("20.0.8.2"),Ipv4Address("10.0.0.8"),2);
```

```
staticRoutingG-
>AddHostRouteTo(Ipv4Address("20.0.7.2"),Ipv4Address("20.0.7.2"),1);

Ptr<Ipv4StaticRouting>                    staticRoutingH                    =
staticRoutingHelper.GetStaticRouting(ipv4H);

staticRoutingH-
>AddHostRouteTo(Ipv4Address("20.0.8.2"),Ipv4Address("20.0.8.2"),1);

staticRoutingH-
>AddHostRouteTo(Ipv4Address("20.0.7.2"),Ipv4Address("10.0.0.7"),2);

}}}

Simulator::Stop(Seconds(tsim));

//    Simulator::Stop();

Simulator::Run();

Simulator::Destroy();

}
```

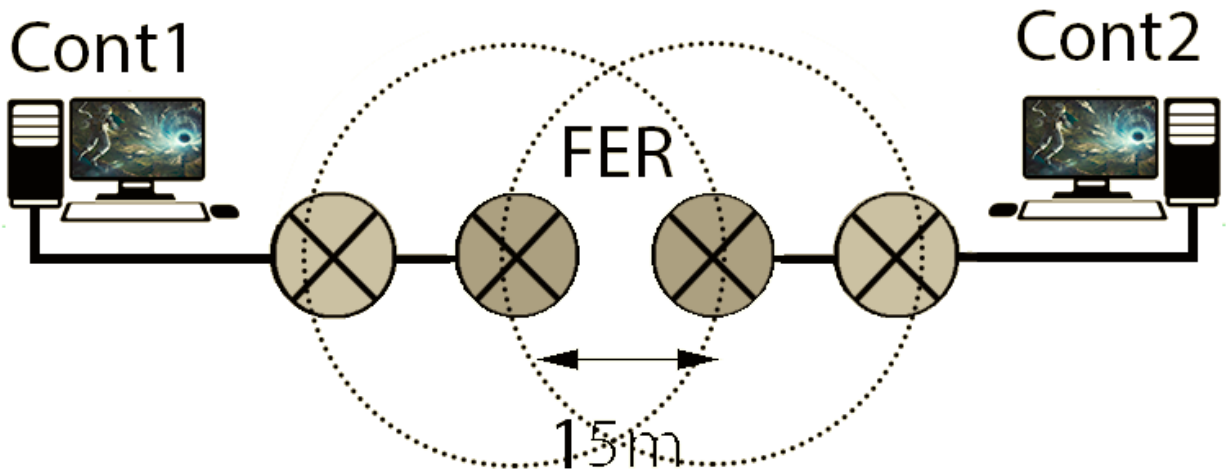
SCRIPT

```
#!/bin/bash

timesim=60

mkdir -p /home/juan/project/stats/mul_stream_0_router

mkdir -p /home/juan/project/stats/mul_stream_0_router/receiver

mkdir -p /home/juan/project/stats/mul_stream_0_router/sender

cd ns-allinone-3.21/ns-3.21/

./waf build

for numcont in `seq 2 2 8`; do

for numsim in `seq 1 1 20`; do

echo "$numcont - $numsim"

if [ $numcont -ge 2 ]; then

xterm -hold -e "sh /home/juan/project/execute.sh 1 20.0.2.2" &

xterm -hold -e "sh /home/juan/project/cli-execute.sh 2" &

fi

if [ $numcont -ge 4 ]; then

xterm -hold -e "sh /home/juan/project/execute.sh 3 20.0.4.2" &

xterm -hold -e "sh /home/juan/project/cli-execute.sh 4" &

fi
```

```
if [ $numcont -ge 6 ]; then
xterm -hold -e "sh /home/juan/project/execute.sh 5 20.0.6.2" &
xterm -hold -e "sh /home/juan/project/cli-execute.sh 6" &
fi
if [ $numcont -ge 8 ]; then
xterm -hold -e "sh /home/juan/project/execute.sh 7 20.0.8.2" &
xterm -hold -e "sh /home/juan/project/cli-execute.sh 8" &
fi
sleep 15
./waf   --run="mul_stream_0_router   --ncont=$numcont   --tsim=$timesim   --
nsim=$numsim"
killall xterm
echo
juju=`ps -A | grep xterm`
while [ ${#juju} -ne 0 ]; do
sleep 1
killall xterm
juju=`ps -A | grep xterm`
done
done
done
```

SIMULACIÓN PRUEBA ERRORES

ESQUEMA ESCENARIO



ESCENARIO

```
#include <iostream>
#include <fstream>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <sstream>

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/mobility-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/wifi-module.h"
#include "ns3/internet-module.h"
#include "ns3/CSMA-module.h"
#include "ns3/applications-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/ipv4-static-routing-helper.h"
#include "ns3/tap-bridge-module.h"
#include "src/internet/helper/internet-stack-helper.h"
#include "src/core/model/simulator.h"
#include "src/network/helper/node-container.h"
#include "src/CSMA/helper/CSMA-helper.h"
#include "src/network/helper/net-device-container.h"
#include "src/internet/helper/ipv4-address-helper.h"
```

```
#include "src/internet/helper/ipv4-static-routing-helper.h"
#include "src/internet/model/ipv4-static-routing.h"
#include "src/network/utils/ipv4-address.h"
#include "src/tap-bridge/model/tap-bridge.h"
#include "src/core/model/global-value.h"
#include "src/core/model/string.h"
#include "src/core/model/config.h"
#include "src/aodv/helper/aodv-helper.h"
#include "src/wifi/helper/yans-wifi-helper.h"
#include "src/network/helper/trace-helper.h"
#include "ns3/error-model.h"

using namespace ns3;

int ns;
//int nc;
float ierr1;

void F_tx(uint64_t i){
    std::stringstream sstm1;
    // Writing results in a file
      sstm1.str("");
      sstm1.clear();
      sstm1 << "/home/juan/project/stats/fer_error/sender/throughput-" << ns
<< "-" << ierr1 << ".txt";
      std::string ss1=sstm1.str();
      std::ofstream myfile1;
      myfile1.open(ss1.c_str(), std::ofstream::app);
      myfile1 << i << "\n";
      myfile1.close();
      sstm1.str("");
      sstm1.clear();
}
void F_rx(std::string i){
    std::stringstream sstm1;
    // Writing results in a file
      sstm1.str("");
```

```
      sstm1.clear();
      sstm1 << "/home/juan/project/stats/fer_error/receiver/throughput-" <<
ns << "-" << ierr1 << ".txt";
      std::string ss1=sstm1.str();
      std::ofstream myfile1;
      myfile1.open(ss1.c_str(), std::ofstream::app);
      myfile1 << i << "\n";
      myfile1.close();
      sstm1.str("");
      sstm1.clear();
}

void Foo1(std::string i) { //CONT 1
    std::stringstream sstm1;
    // Writing results in a file
      sstm1.str("");
      sstm1.clear();
      sstm1 << "/home/juan/project/stats/fer_error/sender/" << ns << "-" <<
ierr1 << ".txt";
      std::string ss1=sstm1.str();
      std::ofstream myfile1;
      myfile1.open(ss1.c_str(), std::ofstream::app);
      myfile1 << i << "\n";
      myfile1.close();
      sstm1.str("");
      sstm1.clear();
}
void Foo2(std::string i) { //CONT 2
    std::stringstream sstm2;
    // Writing results in a file
      sstm2.str("");
      sstm2.clear();
      sstm2 << "/home/juan/project/stats/fer_error/receiver/" << ns << "-" <<
ierr1 << ".txt";
      std::string ss2=sstm2.str();
      std::ofstream myfile2;
      myfile2.open(ss2.c_str(), std::ofstream::app);
```

```
      myfile2 << i << "\n";
      myfile2.close();
      sstm2.str("");
      sstm2.clear();
}

void Foo(std::string i) {

    std::stringstream sstm;
    // Writing results in a file
      sstm.str("");
      sstm.clear();
      sstm << "/home/juan/project/stats/fer_error/ns3jitter/" << ns << "-" <<
ierr1 << ".txt";
      std::string ss=sstm.str();
      std::ofstream myfile;
      myfile.open(ss.c_str(), std::ofstream::app);
      myfile << i << "\n";
      myfile.close();
      sstm.str("");
      sstm.clear();
    ////////////////////////////////////
}

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

    int range=15; //Distance range of the node
    int tsim=100; //Simulation time, after this it stops
    int nsim=1111;
    int ncont=2;
    float err1 = 0.0;
    std::string timevar = "100ms";
    CommandLine cmd;
    cmd.AddValue("ncont", "Number of Containers", ncont);
    cmd.AddValue("range", "Range of routers", range);
    cmd.AddValue("timevar", "Hard limit time", timevar);
    cmd.AddValue("tsim", "Simulation stop time", tsim);
```

```
    cmd.AddValue("nsim", "Simulation number", nsim);
    cmd.AddValue("err1", "error first link", err1);
    cmd.Parse(argc, argv);

    ns=nsim;
//    nc=ncont;

    err1=err1*0.1; //Porque se pasa seq 0 1 10
    ierr1=err1;

    //Time::SetResolution (Time::NS);
    GlobalValue::Bind("SimulatorImplementationType",
StringValue("ns3::RealtimeSimulatorImpl"));
    GlobalValue::Bind("ChecksumEnabled", BooleanValue(true));

    Config::SetDefault("ns3::RealtimeSimulatorImpl::SynchronizationMode",
StringValue("HardLimit"));
    // Default time difference is 100 ms
    Config::SetDefault("ns3::RealtimeSimulatorImpl::HardLimit",
TimeValue(Time(timevar)));

    ns3::Config::SetDefault("ns3::RangePropagationLossModel::MaxRange",
ns3::DoubleValue(range));
    Simulator::GetImplementation()-
>TraceConnectWithoutContext("JitterAndTime", MakeCallback(&Foo));

    ////////////////////////////////

    NodeContainer nodeGW;
    nodeGW.Create(ncont);

    NodeContainer nodes;



//LEFT
    NodeContainer nleft;
    nleft.Add(nodeGW.Get(0));
```

```
    nleft.Create(1);

    InternetStackHelper sleft;
    sleft.Install(nleft.Get(0));

    CSMAHelper cleft;
    cleft.SetChannelAttribute("DataRate",
DataRateValue(DataRate(20000000)));
    cleft.SetChannelAttribute("Delay", TimeValue(MilliSeconds(0)));

    NetDeviceContainer dleft;
    dleft = cleft.Install(nleft);

    Ipv4AddressHelper aleft;
    aleft.SetBase("20.0.1.0","255.255.255.0");

    Ipv4InterfaceContainer ileft;
    ileft = aleft.Assign(dleft.Get(0));

    ////////////////////////////
    //Right
    NodeContainer nright;
    nright.Add(nodeGW.Get(1));
    nright.Create(2);

    InternetStackHelper sright;
    sright.Install(nright.Get(0));

    CSMAHelper cright;
    cright.SetChannelAttribute("DataRate",
DataRateValue(DataRate(20000000)));
    cright.SetChannelAttribute("Delay", TimeValue(MilliSeconds(0)));

    NetDeviceContainer dright;
    dright = cright.Install(nright);

    Ipv4AddressHelper aright;
```

```
    aright.SetBase("20.0.2.0","255.255.255.0");

    Ipv4InterfaceContainer iright;
    iright = aright.Assign(dright.Get(0));

    ////////////////////////////

    nodes.Add(nleft.Get(0));
    nodes.Add(nright.Get(0));

    Ptr<ListPositionAllocator>                positionAlloc                =
CreateObject<ListPositionAllocator > ();
    positionAlloc->Add(Vector(0, 0, 0));
    positionAlloc->Add(Vector(15, 0, 0));

    //WIFI
//////////////////////////////////////////////////////////////////////
    WifiHelper wifi = WifiHelper::Default();
    wifi.SetStandard(WIFI_PHY_STANDARD_80211b);
    wifi.SetRemoteStationManager("ns3::ConstantRateWifiManager", "DataMode",
StringValue("DsssRate11Mbps"));

    NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default();
    wifiMac.SetType("ns3::AdhocWifiMac");



    // Configure the physical layer.
    YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default();
    YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default();

wifiChannel.SetPropagationDelay("ns3::ConstantSpeedPropagationDelayModel");
    wifiChannel.AddPropagationLoss("ns3::RangePropagationLossModel");

    //ERROR FER
    Ptr<MatrixErrorModel> error = CreateObject<MatrixErrorModel> ();
    error->SetFer (nodes.Get(0)->GetId() ,nodes.Get(0)->GetId(), 0.0);
    error->SetFer (nodes.Get(1)->GetId() ,nodes.Get(1)->GetId(), 0.0);
    error->SetFer (nodes.Get(0)->GetId() ,nodes.Get(1)->GetId(), ierr1);
```

```
    error->SetFer (nodes.Get(1)->GetId() ,nodes.Get(0)->GetId(), ierr1);
    error->SetDefaultFer (0.0);



    wifiPhy.SetErrorModel(error);
    wifiPhy.SetChannel(wifiChannel.Create());

    // Install the wireless devices onto our ghost nodes.
    NetDeviceContainer devices = wifi.Install(wifiPhy, wifiMac, nodes);

    // We need location information since we are talking about wifi
    MobilityHelper mobility;
    mobility.SetPositionAllocator(positionAlloc);
    mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
    mobility.Install(nodes);

    Ipv4AddressHelper ipv4;
    ipv4.SetBase("10.0.0.0", "255.255.0.0");
    Ipv4InterfaceContainer interfaces = ipv4.Assign(devices);



    Ptr<Ipv4> ipv4A = nodes.Get(0)->GetObject<Ipv4> ();
    Ptr<Ipv4> ipv4B = nodes.Get(1)->GetObject<Ipv4> ();
    //    Static Routing
    Ipv4StaticRoutingHelper staticRoutingHelper;
    Ptr<Ipv4StaticRouting>                  staticRoutingA                  =
staticRoutingHelper.GetStaticRouting(ipv4A);
    staticRoutingA-
>AddHostRouteTo(Ipv4Address("20.0.2.2"),Ipv4Address("10.0.0.2"),2);
    staticRoutingA-
>AddHostRouteTo(Ipv4Address("20.0.1.2"),Ipv4Address("20.0.1.2"),1);


    Ptr<Ipv4StaticRouting>                  staticRoutingB                  =
staticRoutingHelper.GetStaticRouting(ipv4B);
    staticRoutingB-
>AddHostRouteTo(Ipv4Address("20.0.2.2"),Ipv4Address("20.0.2.2"),1);
    staticRoutingB-
>AddHostRouteTo(Ipv4Address("20.0.1.2"),Ipv4Address("10.0.0.1"),2);
```

```
    TapBridgeHelper tapBridge;
    tapBridge.SetAttribute("Mode", StringValue("UseBridge"));
    tapBridge.SetAttribute("DeviceName", StringValue("tap-cont1"));
    Ptr<NetDevice> tap_1=tapBridge.Install(nleft.Get(1), dleft.Get(1));
    tapBridge.SetAttribute("DeviceName", StringValue("tap-cont2"));
    Ptr<NetDevice> tap_2=tapBridge.Install(nright.Get(1), dright.Get(1));

    tap_1->TraceConnectWithoutContext("PktTX", MakeCallback(&Foo1));
    tap_1->TraceConnectWithoutContext("ThroughputTx", MakeCallback(&F_tx));
    tap_2->TraceConnectWithoutContext("PktRX", MakeCallback(&Foo2));
    tap_2->TraceConnectWithoutContext("ThroughputRx", MakeCallback(&F_rx));

    Simulator::Stop(Seconds(tsim));
//    Simulator::Stop();
    Simulator::Run();
    Simulator::Destroy();
}
```

SCRIPT

Script usado para la automatización de las simulaciones con los escenarios. En este caso, el script se va a encargar de ejecutar el escenario del NS3, pasándole los parámetros que el escenario necesita para ir pasando por todos los casos que nos interesan.

- ./waf --run="fer_error --tsim=$timesim --nsim=$numsim --err1=$e1 "
- Tsim: El tiempo que el escenario se va a simular. En este caso 60 segundos.
- Nsim: Número de repetición de la simulación, de 1 a 20.
- Err1: Número usado para pasar la tasa de errores que el canal va a tener.

En el script se puede observar que hay un comando llamado XTERM. XTERM es el emulador de terminal estándar, permite que hayan más de un terminal emulado abierto y funcionando a la vez, y en este script se usa para abrir el VLC tanto en servidor como en cliente, y tantos como contenedores se usen en la simulación.

Al final de cada simulación se destruyen, para que así cada simulación esté en igualdad de condiciones, y los resultados no se vean comprometidos.

```
#!/bin/bash
timesim=60
cd ns-allinone-3.21/ns-3.21/
./waf build
./waf --run="fer_error --tsim=10"

for e1 in `seq 0 1 10`; do
for numsim in `seq 34 1 34`; do
xterm -hold -e "sh /home/juan/project/execute.sh 1 20.0.2.2" &
xterm -hold -e "sh /home/juan/project/cli-execute.sh 2" &
sleep 6
./waf --run="fer_error --tsim=$timesim --nsim=$numsim --err1=$e1 "
killall xterm
echo "Error $e1 -nsim $numsim"
juju=`ps -A | grep xterm`
while [ ${#juju} -ne 0 ]; do
sleep 0.5
killall xterm
juju=`ps -A | grep xterm`
done
done
done
```

SIMULACIÓN COMPLETA

ESCENARIO

```
#include <iostream>
#include <fstream>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <sstream>

#include "ns3/core-module.h"
#include "ns3/network-module.h"
```

```
#include "ns3/mobility-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/wifi-module.h"
#include "ns3/internet-module.h"
#include "ns3/CSMA-module.h"
#include "ns3/applications-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/ipv4-static-routing-helper.h"
#include "ns3/tap-bridge-module.h"
#include "src/internet/helper/internet-stack-helper.h"
#include "src/core/model/simulator.h"
#include "src/network/helper/node-container.h"
#include "src/CSMA/helper/CSMA-helper.h"
#include "src/network/helper/net-device-container.h"
#include "src/internet/helper/ipv4-address-helper.h"
#include "src/internet/helper/ipv4-static-routing-helper.h"
#include "src/internet/model/ipv4-static-routing.h"
#include "src/network/utils/ipv4-address.h"
#include "src/tap-bridge/model/tap-bridge.h"
#include "src/core/model/global-value.h"
#include "src/core/model/string.h"
#include "src/core/model/config.h"
#include "src/aodv/helper/aodv-helper.h"
#include "src/wifi/helper/yans-wifi-helper.h"
#include "src/network/helper/trace-helper.h"
#include "ns3/error-model.h"
#include "src/network/model/node.h"

using namespace ns3;

int ns;
//int nc;
float ierr1;

void F_tx(uint64_t i){
    std::stringstream sstm1;
```

```
    // Writing results in a file
      sstm1.str("");
      sstm1.clear();
      sstm1 << "/home/juan/project/stats/fer_errors/sender/throughput-" << ns
<< ".txt";
      std::string ss1=sstm1.str();
      std::ofstream myfile1;
      myfile1.open(ss1.c_str(), std::ofstream::app);
      myfile1 << i << "\n";
      myfile1.close();
      sstm1.str("");
      sstm1.clear();
}
void F_rx(std::string i){
    std::stringstream sstm1;
    // Writing results in a file
      sstm1.str("");
      sstm1.clear();
      sstm1 << "/home/juan/project/stats/fer_errors/receiver/throughput-" <<
ns << ".txt";
      std::string ss1=sstm1.str();
      std::ofstream myfile1;
      myfile1.open(ss1.c_str(), std::ofstream::app);
      myfile1 << i << "\n";
      myfile1.close();
      sstm1.str("");
      sstm1.clear();
}

void Foo1(std::string i) { //CONT 1
    std::stringstream sstm1;
    // Writing results in a file
      sstm1.str("");
      sstm1.clear();
      sstm1 << "/home/juan/project/stats/fer_errors/sender/" << ns << ".txt";
      std::string ss1=sstm1.str();
      std::ofstream myfile1;
```

```
      myfile1.open(ss1.c_str(), std::ofstream::app);
      myfile1 << i << "\n";
      myfile1.close();
      sstm1.str("");
      sstm1.clear();
}
void Foo2(std::string i) { //CONT 2
    std::stringstream sstm2;
    // Writing results in a file
      sstm2.str("");
      sstm2.clear();
      sstm2  <<  "/home/juan/project/stats/fer_errors/receiver/"  <<  ns  <<
".txt";
      std::string ss2=sstm2.str();
      std::ofstream myfile2;
      myfile2.open(ss2.c_str(), std::ofstream::app);
      myfile2 << i << "\n";
      myfile2.close();
      sstm2.str("");
      sstm2.clear();
}

void Foo(std::string i) {

    std::stringstream sstm;
    // Writing results in a file
      sstm.str("");
      sstm.clear();
      sstm  <<  "/home/juan/project/stats/fer_errors/ns3jitter/"  <<  ns  <<
".txt";
      std::string ss=sstm.str();
      std::ofstream myfile;
      myfile.open(ss.c_str(), std::ofstream::app);
      myfile << i << "\n";
      myfile.close();
      sstm.str("");
      sstm.clear();
```

```
    ////////////////////////////////////
}

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

    int range=15; //Distance range of the node
    int tsim=100; //Simulation time, after this it stops
    int nsim=1111;
    int ncont=3;
    std::string timevar = "100ms";
    CommandLine cmd;
    cmd.AddValue("ncont", "Number of Containers", ncont);
    cmd.AddValue("range", "Range of routers", range);
    cmd.AddValue("timevar", "Hard limit time", timevar);
    cmd.AddValue("tsim", "Simulation stop time", tsim);
    cmd.AddValue("nsim", "Simulation number", nsim);
    cmd.Parse(argc, argv);

    ns=nsim;

    //Time::SetResolution (Time::NS);
    GlobalValue::Bind("SimulatorImplementationType",
StringValue("ns3::RealtimeSimulatorImpl"));
    GlobalValue::Bind("ChecksumEnabled", BooleanValue(true));

    Config::SetDefault("ns3::RealtimeSimulatorImpl::SynchronizationMode",
StringValue("HardLimit"));
    // Default time difference is 100 ms
    Config::SetDefault("ns3::RealtimeSimulatorImpl::HardLimit",
TimeValue(Time(timevar)));

    ns3::Config::SetDefault("ns3::RangePropagationLossModel::MaxRange",
ns3::DoubleValue(range));
    Simulator::GetImplementation()-
>TraceConnectWithoutContext("JitterAndTime", MakeCallback(&Foo));

    ////////////////////////////////
    int nrouters = 6;
```

```
    NodeContainer nodeGW;
    nodeGW.Create(ncont);
    NodeContainer nodeWifi;
    nodeWifi.Create(nrouters);
    NodeContainer nodes;



//server
    NodeContainer nserver;
    nserver.Add(nodeGW.Get(0));
    nserver.Create(1);

    InternetStackHelper sserver;
    sserver.Install(nserver.Get(0));

    CSMAHelper cserver;
    cserver.SetChannelAttribute("DataRate",
DataRateValue(DataRate(20000000)));
    cserver.SetChannelAttribute("Delay", TimeValue(MilliSeconds(0)));

    NetDeviceContainer dserver;
    dserver = cserver.Install(nserver);

    Ipv4AddressHelper aserver;
    aserver.SetBase("20.0.1.0","255.255.255.0");

    Ipv4InterfaceContainer iserver;
    iserver = aserver.Assign(dserver.Get(0));

    ////////////////////////////
    //C1
    NodeContainer nc1;
    nc1.Add(nodeGW.Get(1));
    nc1.Create(2);
```

```
InternetStackHelper sc1;
sc1.Install(nc1.Get(0));

CSMAHelper cc1;
cc1.SetChannelAttribute("DataRate", DataRateValue(DataRate(20000000)));
cc1.SetChannelAttribute("Delay", TimeValue(MilliSeconds(0)));

NetDeviceContainer dc1;
dc1 = cc1.Install(nc1);

Ipv4AddressHelper ac1;
ac1.SetBase("20.0.2.0","255.255.255.0");

Ipv4InterfaceContainer ic1;
ic1 = ac1.Assign(dc1.Get(0));

////////////////////////////
////////////////////////////
//C2
NodeContainer nc2;
nc2.Add(nodeGW.Get(2));
nc2.Create(2);

InternetStackHelper sc2;
sc2.Install(nc2.Get(0));

CSMAHelper cc2;
cc2.SetChannelAttribute("DataRate", DataRateValue(DataRate(20000000)));
cc2.SetChannelAttribute("Delay", TimeValue(MilliSeconds(0)));

NetDeviceContainer dc2;
dc2 = cc2.Install(nc2);

Ipv4AddressHelper ac2;
ac2.SetBase("20.0.3.0","255.255.255.0");
```

```
    Ipv4InterfaceContainer ic2;
    ic2 = ac2.Assign(dc2.Get(0));

    ////////////////////////////



    NodeContainer routers;
    routers.Add(nodeWifi);

    InternetStackHelper stackrout;
    stackrout.Install(routers);



    nodes.Add(nserver.Get(0));
    nodes.Add(nc1.Get(0));
    nodes.Add(nc2.Get(0));
    nodes.Add(routers);


    Ptr<ListPositionAllocator>                positionAlloc                =
CreateObject<ListPositionAllocator > ();
    positionAlloc->Add(Vector(0, 0, 0)); //10.0.0.1
    positionAlloc->Add(Vector(30, 30, 0)); //10.0.0.2
    positionAlloc->Add(Vector(30, 0, 0)); //10.0.0.3
    positionAlloc->Add(Vector(0, 15, 0)); //10.0.0.4
    positionAlloc->Add(Vector(0, 30, 0)); //10.0.0.5
    positionAlloc->Add(Vector(15, 0, 0)); //10.0.0.6
    positionAlloc->Add(Vector(15, 15, 0)); //10.0.0.7
    positionAlloc->Add(Vector(15, 30, 0)); //10.0.0.8
    positionAlloc->Add(Vector(30, 15, 0)); //10.0.0.9

    //WIFI
//////////////////////////////////////////////////////////////////////////
    WifiHelper wifi = WifiHelper::Default();
    wifi.SetStandard(WIFI_PHY_STANDARD_80211b);
```

```
    wifi.SetRemoteStationManager("ns3::ConstantRateWifiManager", "DataMode",
StringValue("DsssRate11Mbps"));

    NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default();
    wifiMac.SetType("ns3::AdhocWifiMac");



    // Configure the physical layer.
    YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default();
    YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default();

wifiChannel.SetPropagationDelay("ns3::ConstantSpeedPropagationDelayModel");
    wifiChannel.AddPropagationLoss("ns3::RangePropagationLossModel");

    //ERROR FER
    Ptr<MatrixErrorModel> error = CreateObject<MatrixErrorModel> ();
    error->SetFer (nodes.Get(0)->GetId() ,nodes.Get(0)->GetId(), 0.0);
    error->SetFer (nodes.Get(1)->GetId() ,nodes.Get(1)->GetId(), 0.0);
    error->SetFer (nodes.Get(2)->GetId() ,nodes.Get(2)->GetId(), 0.0);
    error->SetFer (nodes.Get(3)->GetId() ,nodes.Get(3)->GetId(), 0.0);
    error->SetFer (nodes.Get(4)->GetId() ,nodes.Get(4)->GetId(), 0.0);
    error->SetFer (nodes.Get(5)->GetId() ,nodes.Get(5)->GetId(), 0.0);
    error->SetFer (nodes.Get(6)->GetId() ,nodes.Get(6)->GetId(), 0.0);
    error->SetFer (nodes.Get(7)->GetId() ,nodes.Get(7)->GetId(), 0.0);
    error->SetFer (nodes.Get(8)->GetId() ,nodes.Get(8)->GetId(), 0.0);
    error->SetFer (nodes.Get(9)->GetId() ,nodes.Get(9)->GetId(), 0.0);

    // 1st stream
    error->SetFer (nodes.Get(0)->GetId() ,nodes.Get(3)->GetId(), 0.1);
    error->SetFer (nodes.Get(3)->GetId() ,nodes.Get(0)->GetId(), 0.1);

    error->SetFer (nodes.Get(3)->GetId() ,nodes.Get(6)->GetId(), 0.1);
    error->SetFer (nodes.Get(6)->GetId() ,nodes.Get(3)->GetId(), 0.1);

    error->SetFer (nodes.Get(6)->GetId() ,nodes.Get(7)->GetId(), 0.2);
    error->SetFer (nodes.Get(7)->GetId() ,nodes.Get(6)->GetId(), 0.2);
```

```
    error->SetFer (nodes.Get(7)->GetId() ,nodes.Get(1)->GetId(), 0.2);
    error->SetFer (nodes.Get(1)->GetId() ,nodes.Get(7)->GetId(), 0.2);
    // 2nd stream
    error->SetFer (nodes.Get(0)->GetId() ,nodes.Get(5)->GetId(), 0.1);
    error->SetFer (nodes.Get(5)->GetId() ,nodes.Get(0)->GetId(), 0.1);

    error->SetFer (nodes.Get(5)->GetId() ,nodes.Get(2)->GetId(), 0.2);
    error->SetFer (nodes.Get(2)->GetId() ,nodes.Get(5)->GetId(), 0.2);

    error->SetDefaultFer (0.1);

    wifiPhy.SetErrorModel(error);
    wifiPhy.SetChannel(wifiChannel.Create());

    // Install the wireless devices onto our ghost nodes.
    NetDeviceContainer devices = wifi.Install(wifiPhy, wifiMac, nodes);

    // We need location information since we are talking about wifi
    MobilityHelper mobility;
    mobility.SetPositionAllocator(positionAlloc);
    mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
    mobility.Install(nodes);

    Ipv4AddressHelper ipv4;
    ipv4.SetBase("10.0.0.0", "255.255.0.0");
    Ipv4InterfaceContainer interfaces = ipv4.Assign(devices);



    Ptr<Ipv4> ipv4A = nodes.Get(0)->GetObject<Ipv4> ();
    Ptr<Ipv4> ipv4B = nodes.Get(1)->GetObject<Ipv4> ();
    Ptr<Ipv4> ipv4C = nodes.Get(2)->GetObject<Ipv4> ();
    Ptr<Ipv4> ipv4D = nodes.Get(3)->GetObject<Ipv4> ();
//    Ptr<Ipv4> ipv4E = nodes.Get(4)->GetObject<Ipv4> ();
    Ptr<Ipv4> ipv4F = nodes.Get(5)->GetObject<Ipv4> ();
    Ptr<Ipv4> ipv4G = nodes.Get(6)->GetObject<Ipv4> ();
    Ptr<Ipv4> ipv4H = nodes.Get(7)->GetObject<Ipv4> ();
```

```
//    Ptr<Ipv4> ipv4I = nodes.Get(8)->GetObject<Ipv4> ();


    //    Static Routing
    Ipv4StaticRoutingHelper staticRoutingHelper;
    Ptr<Ipv4StaticRouting>                  staticRoutingA                  =
staticRoutingHelper.GetStaticRouting(ipv4A);
    staticRoutingA-
>AddHostRouteTo(Ipv4Address("20.0.2.2"),Ipv4Address("10.0.0.4"),2);
    staticRoutingA-
>AddHostRouteTo(Ipv4Address("20.0.3.2"),Ipv4Address("10.0.0.6"),2);
    staticRoutingA-
>AddHostRouteTo(Ipv4Address("20.0.1.2"),Ipv4Address("20.0.1.2"),1);


    Ptr<Ipv4StaticRouting>                  staticRoutingB                  =
staticRoutingHelper.GetStaticRouting(ipv4B);
    staticRoutingB-
>AddHostRouteTo(Ipv4Address("20.0.2.2"),Ipv4Address("20.0.2.2"),1);
    staticRoutingB-
>AddHostRouteTo(Ipv4Address("20.0.1.2"),Ipv4Address("10.0.0.8"),2);


    Ptr<Ipv4StaticRouting>                  staticRoutingC                  =
staticRoutingHelper.GetStaticRouting(ipv4C);
    staticRoutingC-
>AddHostRouteTo(Ipv4Address("20.0.3.2"),Ipv4Address("20.0.3.2"),1);
    staticRoutingC-
>AddHostRouteTo(Ipv4Address("20.0.1.2"),Ipv4Address("10.0.0.6"),2);


    Ptr<Ipv4StaticRouting>                  staticRoutingD                  =
staticRoutingHelper.GetStaticRouting(ipv4D);
    staticRoutingD-
>AddHostRouteTo(Ipv4Address("20.0.2.2"),Ipv4Address("10.0.0.7"),1);
    staticRoutingD-
>AddHostRouteTo(Ipv4Address("20.0.1.2"),Ipv4Address("10.0.0.1"),1);




    Ptr<Ipv4StaticRouting>                  staticRoutingF                  =
staticRoutingHelper.GetStaticRouting(ipv4F);
    staticRoutingF-
>AddHostRouteTo(Ipv4Address("20.0.3.2"),Ipv4Address("10.0.0.3"),1);
    staticRoutingF-
>AddHostRouteTo(Ipv4Address("20.0.1.2"),Ipv4Address("10.0.0.1"),1);
```

```
    Ptr<Ipv4StaticRouting>                  staticRoutingG                  =
staticRoutingHelper.GetStaticRouting(ipv4G);
    staticRoutingG-
>AddHostRouteTo(Ipv4Address("20.0.2.2"),Ipv4Address("10.0.0.8"),1);
    staticRoutingG-
>AddHostRouteTo(Ipv4Address("20.0.1.2"),Ipv4Address("10.0.0.4"),1);


    Ptr<Ipv4StaticRouting>                  staticRoutingH                  =
staticRoutingHelper.GetStaticRouting(ipv4H);
    staticRoutingH-
>AddHostRouteTo(Ipv4Address("20.0.2.2"),Ipv4Address("10.0.0.2"),1);
    staticRoutingH-
>AddHostRouteTo(Ipv4Address("20.0.1.2"),Ipv4Address("10.0.0.7"),1);









    TapBridgeHelper tapBridge;
    tapBridge.SetAttribute("Mode", StringValue("UseBridge"));
    tapBridge.SetAttribute("DeviceName", StringValue("tap-cont1"));
    Ptr<NetDevice> tap_1=tapBridge.Install(nserver.Get(1), dserver.Get(1));
    tapBridge.SetAttribute("DeviceName", StringValue("tap-cont2"));
    Ptr<NetDevice> tap_2=tapBridge.Install(nc1.Get(1), dc1.Get(1));
    tapBridge.SetAttribute("DeviceName", StringValue("tap-cont3"));
    Ptr<NetDevice> tap_3=tapBridge.Install(nc2.Get(1), dc2.Get(1));

    tap_1->TraceConnectWithoutContext("PktTX", MakeCallback(&Foo1));
    tap_1->TraceConnectWithoutContext("ThroughputTx", MakeCallback(&F_tx));

    tap_2->TraceConnectWithoutContext("PktRX", MakeCallback(&Foo2));
    tap_2->TraceConnectWithoutContext("ThroughputRx", MakeCallback(&F_rx));

    tap_3->TraceConnectWithoutContext("PktRX", MakeCallback(&Foo2));
    tap_3->TraceConnectWithoutContext("ThroughputRx", MakeCallback(&F_rx));

    Simulator::Stop(Seconds(tsim));
```

```
    Simulator::Run();
    Simulator::Destroy();
}
```

SCRIPT

```
#!/bin/bash
timesim=60
cd ns-allinone-3.21/ns-3.21/
./waf build

for numsim in `seq 1 1 30`; do
echo "Nsim $numsim"
xterm -hold -e "sh /home/juan/project/execute_2.sh" &
xterm -hold -e "sh /home/juan/project/cli-execute.sh 2" &
xterm -hold -e "sh /home/juan/project/cli-execute.sh 3" &
sleep 7
./waf --run="fer_errors_2 --tsim=$timesim --nsim=$numsim"
killall xterm
juju=`ps -A | grep xterm`
while [ ${#juju} -ne 0 ]; do
sleep 1
killall xterm
juju=`ps -A | grep xterm`
done
done
```

GLOSARIO

Cgroups

Es una característica inicialmente desarrollada para limitar el uso de recursos en el kernel de Linux. Pero puede hacer mucho más, como ajustar el uso de memoria, ancho de banda y CPU, también sirve para denegar el acceso a recursos del sistema.

Uids

El espacio de usuario es una parte de la memoria del Sistema en la que procesos de usuario se ejecutan. El espacio de usuario sirve para diferenciar procesos, y que así el Kernel pueda evitar que estos interfieran unos con otros.

Gids

En UNIX, un grupo es una colección lógica de usuarios en un sistema, su principal uso es asignar permisos de grupo a ficheros y directorios. Los grupos son independientes unos de otros.

Tap

Dispositivo que simula un dispositivo de capa de red y funciona con paquetes de capa dos, cómo datagramas Ethernet. Los paquetes enviados por un sistema operativo vía TAP, son entregados a un programa en espacio de usuario que está adherido al dispositivo. También un programa de espacio de usuario pasa paquetes al dispositivo TAP. En este caso, el TAP inyecta los paquetes a la pila de protocolos de red del sistema operativo.

Bridge

Dispositivo de interconexión de redes de ordenadores que opera en la capa 2 (nivel de enlace) del modelo OSI.

Beacons

Paquetes de datos que envían los puntos de acceso para hacer funcionar una red.

CBR traffic

Constant Bit Rate traffic, manda tráfico a una tasa fija.

CDF

Cumulative Density Function, función que muestra la probabilidad de que un valor sea menor o igual a un porcentaje dado.

https://www.nsnam.org/docs/tutorial/html/conceptual-overview.html

https://www.videolan.org/doc/streaming-howto/en/ch04.html

https://www.nsnam.org/wiki/index.php/HOWTO_make_ns-3_interact_with_the_real_world#ns-3_TapBridge

https://www.nsnam.org/docs/release/3.13/models/html/tap.html

https://www.nsnam.org/wiki/HOWTO_Use_Linux_Containers_to_set_up_virtual_networks

https://www.stgraber.org/2014/02/09/lxc-1-0-gui-in-containers/

https://www.suse.com/documentation/sles11/singlehtml/lxc_quickstart/lxc_quickstart.html

http://www.ipass.com/press-releases/ipass-wi-fi-growth-map-shows-one-public-hotspot-for-every-20-people-on-earth-by-2018/

http://www.ciscopress.com/articles/article.asp?p=357102&seqNum=2

http://www.certusdigital.com/index.php?option=com_content&view=article&id=58:why-is-jitter-important&catid=34:questions-a-answers&Itemid=29

https://www.nsnam.org/docs/release/3.12/manual/html/realtime.html

http://linux.die.net/man/8/tunctl

http://www.thegeekstuff.com/2009/03/ifconfig-7-examples-to-configure-network-interface/

http://linuxcommand.org/man_pages/brctl8.html

http://moment.cs.ucsb.edu/AODV/

http://www.everythingvm.com/content/history-virtualization

http://en.wikipedia.org/wiki/Virtualization

https://www.stgraber.org/2013/12/20/lxc-1-0-your-first-ubuntu-container/

http://infonomics-society.org/IJISR/Paper%201.pdf

http://searchunifiedcommunications.techtarget.com/definition/QoS-Quality-of-Service

http://www.brookings.edu/~/media/research/files/papers/2014/05/02-video-streaming/west_evolution-of-videostreaming-and-digital-content-delivery_final.pdf

http://www.linfo.org/kernel_space.html

https://kb.iu.edu/d/aeqw