UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

SIMULACIÓN DE NODOS DE ACCESO DE BANDA ANCHA, DSLAM

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELECTRICISTA

LUIS EDUARDO ACUÑA VEGA

PROFESOR GUÍA:
SR. ALBERTO CASTRO ROJAS

MIEMBROS DE LA COMISION
SR. CLAUDIO ESTÉVEZ MONTERO
SR. JORGE SANDOVAL ARENAS

SANTIAGO DE CHILE
MAYO 2012

RESUMEN DE LA MEMORIA
PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELECTRICISTA
POR: LUIS ACUÑA VEGA
FECHA: 20/JUNIO/2012
P. GUÍA: SR. ALBERTO CASTRO R.

SIMULACIÓN DE NODOS DE ACCESO DE BANDA ANCHA, DSLAM

El objetivo general de este trabajo es la simulación de un nodo de acceso DSLAM para realizar proyecciones de tráfico para diferentes aplicaciones. Esta problemática surge por parte de la empresa Movistar Chile S.A. que tiene la necesidad de realizar estudios para evaluar sus sistemas de banda ancha fija y como se enfrentan éstos a diversos servicios como lo son IPTV y VoIP. Por esto se requiere de una herramienta capaz de generar tráfico similar al que hay en Internet y con la flexibilidad suficiente para representar estos sistemas y servicios, además de requerimientos como la cuantificación de los clientes que superen un retardo mínimo o el poder generar tráfico temporalmente distribuido siguiendo alguna distribución estadística. También se agregan características como configuración de retardos, tamaños de las colas en la red, división de clientes en grupos para asignarles factores de multiplicación de tráfico y/o velocidad, etc., de forma tal de tener más parámetros posibles de configurar en la red. Por otro lado para hacer más sencilla la utilización del código resultante del trabajo se genera una interfaz gráfica que permite configurar la simulación.

El proceso de desarrollo consiste en modelar la red y cada parte que la integra, definiendo las nuevas funcionalidades, los parámetros configurables y mediciones que se esperan simular. Luego se pasa a la etapa de implementación de los modelos en el simulador OMNeT++, el cual utiliza el lenguaje C++ y el lenguaje NED para la descripción de las redes. Para esta etapa se utilizan las librerías Inet y ReaSE donde la primera contiene los protocolos base utilizados en Internet y la segunda contiene utilidades para el manejo de los flujos de tráfico. Los modelos de red se rehacen para poder satisfacer las necesidades requeridas por Movistar o por parte del código cuando necesita la creación de nuevos módulos de red. Simultáneamente se trabaja en la construcción de la interfaz gráfica la cual debe ser compatible con todos los cambios de la red y con todos los parámetros configurables pudiendo generar los archivos necesarios para ejecutar las simulaciones.

Finalmente se logra un modelo estable de la red, que permite configurar todas las funcionalidades y parámetros que se plantearon como requisitos del simulador y además se crea una interfaz gráfica que permite crear nuevas simulaciones de manera sencilla y rápida, además de una máquina virtual con todos los programas necesarios ya instalados. Los datos proyectados de tráfico son útiles para generar valiosos análisis como se observa en la sección de resultados y son coherentes con los tráficos reales de verificación.

AGRADECIMIENTOS

En primer lugar quiero agradecer a mi profesor guía Alberto Castro por su constante ayuda a través de todo el proceso, dándome ideas, sugerencias y al mismo tiempo su paciencia, permitiéndome continuar a pesar de todos los problemas que se presentaron.

Agradezco a toda mi familia por su apoyo a lo largo de mi vida y especialmente durante la finalización de mis estudios.

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

En la actualidad internet aumenta tanto su extensión como su capacidad, llegando a más usuarios y otorgándoles mayores velocidades a los mismos gracias a sus servicios y a la competencia entre los distintos proveedores de servicios de internet. Este rápido crecimiento requiere de planificación e implementación adecuada de las redes y para esto se hace necesario la evaluación de diferentes escenarios obteniendo así resultados adecuados a las necesidades del proveedor de servicios y a la calidad de servicio requerida. Existe entonces la problemática de qué tecnología utilizar y con qué características dotar a las redes, tanto nuevas como ya existentes. De aquí surgen las interrogantes de cómo evaluar los parámetros de la red, de los nodos de acceso y de los clientes de manera efectiva y a bajo costo. Las alternativas para la evaluación son experimentar sobre el sistema real, experimentación sobre un prototipo construido de red, experimentación con un modelo matemático, y la experimentación sobre un modelo simulable en computador.

La primera alternativa es la de una experimentación directamente en el sistema real ya establecido y ahí medir parámetros de todo tipo, pero se hace inviable poder controlar variables como el tráfico de subida y de bajada por los nodos de acceso además de que interferir el sistema real para la experimentación y medición puede ser de un alto coste. En la segunda alternativa, la confección de un prototipo de red, puede resultar extremadamente difícil además de inviable dado los costos y la imposibilidad de poder emular el tráfico de internet de manera realista, que en este tipo de experimentación puede resultar muy costoso para redes grandes, hacer un laboratorio resulta entonces impráctico e inviable. La tercera alternativa de crear un modelo analítico matemático puede resultar de extremadamente alta complejidad especialmente si se tiene variables aleatorias o cambios de estado en el tiempo como lo son la mayoría de las simulaciones de redes por lo que esta herramienta de modelación queda relegada dada su imposibilidad, aunque cuando es posible, para sistemas más simples, sí es útil.

Por último la simulación de redes en un computador permite tomar en cuenta que el consumo de los clientes cambia en el tiempo, también la aparición de nuevos servicios y prever nuevos posibles problemas. La simulación por computador tiene un bajo costo de implementación y permite fácilmente poder evaluar nuevos conceptos de mercado, su factibilidad y/o sus cambios necesarios. Existen diferentes herramientas de simulación de eventos discretos siendo las de software libre las más utilizadas y muchas veces están más avanzadas que las herramientas comerciales, además éstas permiten aprovechar trabajos previos. Por estos motivos la simulación por computador es la alternativa seleccionada para la realización del presente trabajo. Trabajo que pretende simular las condiciones de tráfico, topológicas y de arquitectura de un nodo de acceso con funcionalidades propias de él. En particular, se simulará el funcionamiento de un IP-DSLAM, notando que los nodos particulares que se verán más adelante en el presente capítulo no vienen por defecto en los paquetes de las herramientas de simulación por lo que un valor agregado de este trabajo es la creación del elemento IP-

DSLAM con funcionalidad similar a la real y además la creación de una interfaz gráfica que pueda crear ambientes de simulación de manera rápida permitiendo así la experimentación de muchos escenarios tanto de tráfico como topológicos, permitiendo variar también una multitud de parámetros de todo tipo permitiendo tráfico realista y también aleatorio.

1.1 Objetivos

El objetivo primario del trabajo es la simulación de un nodo de acceso DSLAM para realizar proyecciones de tráfico para diferentes aplicaciones.

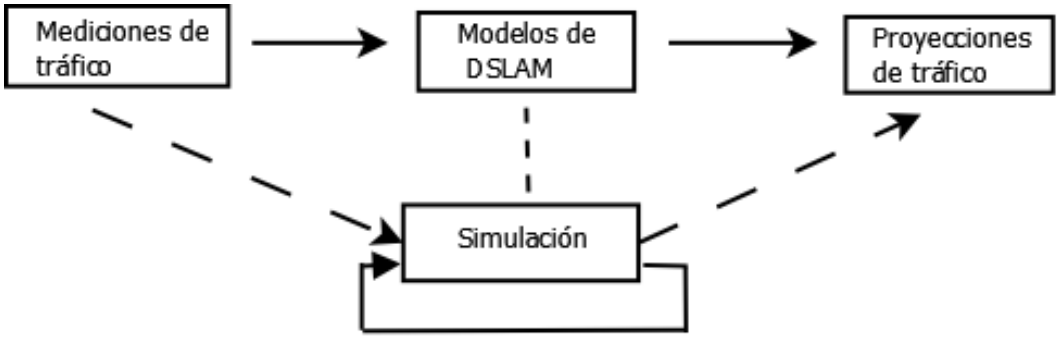


Figura 1.1: Diagrama de caminos para obtener proyecciones desde las mediciones.

Los objetivos secundarios a desarrollarse en el presente trabajo son cinco:

- Comparación entre datos proyectados y datos de verificación.
- Crear interfaz gráfica de simulación.
- Entregar reportes solicitados tales como la pérdida de paquetes, retardos máximos y cantidad de clientes que superan un retardo límite dado. (simulación->Proyección->reportes solicitados).
- Identificar las restricciones de la herramienta de simulación: cantidad máxima de clientes, tráfico, velocidad y otros.
- Facultad de generar tráfico distribuido matemáticamente en el tiempo para realizar diversas pruebas.

1.2 Estructura del documento

El documento consta de cinco capítulos de contenido: introducción, antecedentes, desarrollo, resultados y conclusiones, además de la bibliografía y dos capítulos de anexos.

En el presente capítulo de introducción se explica la problemática a tratarse por parte de los interesados, se muestran las alternativas de trabajo y se explica porque la simulación es la alternativa ideal para el caso. También está la sección de objetivos donde se plantea lo que se debe o espera realizar como requisitos. En el capítulo de introducción se encuentra la presente sub-sección de estructura del documento.

En el capítulo dos de antecedentes se muestran el estado del arte, los desafíos para la simulación, las herramientas de simulación de libre uso, las mediciones, una herramienta de clasificación estadística del tráfico y la estimación de capacidad de enlaces ADSL. En estado del arte se tienen los modelos de nodos de acceso DSLAM, también se tiene modelos de la red y modelos de tráfico. En la sección de los desafíos para la simulación se describe cuando se hace necesaria la simulación computacional y se presentan requerimientos para los modelos de rendimiento y simulación además de las buenas prácticas para el modelado. En herramientas de simulación de libre uso se describen y caracterizan cinco herramientas profundizando en OMNeT++ mostrando algunas de las librerías asociadas a ésta. En la sección de mediciones se muestran las principales mediciones que se esperan lograr. En la sección de herramientas de clasificación estadística del tráfico se presentan básicamente dos casos: aplicando el clasificador en el mismo lugar en que fue entrenado y cuando es aplicado a un lugar diferente al que fue entrenado. En la sección de estimación de capacidad de enlaces ADSL se muestra la herramienta DSLprobe la cual es una herramienta no cooperativa pues no requiere de participación activa de los hosts a medir y que permite estimar las capacidades de subida y de bajada.

El capítulo tres de desarrollo consta de las partes de definición de los modelos a utilizar, de implementación de red básica y de interfaces. En la sección de los modelos a utilizar se encuentran el modelo de red, los modelos de nodos de acceso, el modelo de servidores y el modelo de clientes. En la sección de implementación de red básica se muestra la red implementada y todos los parámetros configurables para cada modelo implementado. En la sección de interfaces se muestra la interacción de las interfaces en presencia de un flujo de tráfico cliente-servidor.

En el capítulo cuarto de resultados se presentan los resultados y los análisis de tres casos. El primero es el de una simulación basada en datos de un DSLAM-IP. El segundo es de un caso de simulación de tráfico con y sin servicios IPTV y VoIP en presencia de pérdida de paquetes. El tercer caso corresponde a la simulación de tráfico con selección de tiempo de inicio distribuido.

El capítulo cinco de conclusiones se subdivide en conclusiones referentes a los objetivos, en conclusiones referentes al desarrollo, en conclusiones referentes a los resultados y en trabajos futuros.

El capítulo seis es el anexo A: Manual de Usuario, donde se explica cómo instalar y como configurar los parámetros para realizar simulaciones y también como obtener los resultados.

El capítulo siete es el anexo B: Cambios a librerías INET y ReaSE, donde se presentan treinta cambios hechos a estas librerías.

2 ANTECEDENTES

2.1 Estado del arte

En la actualidad existen diversos simuladores que permiten la simulación de redes de internet, sus protocolos y sus elementos. Éstos son elementos que no son específicos y al ser necesarios modificarlos quedan fuera del estudio todas las herramientas que no permiten la modificación del código, aunque la mayoría sí lo permite. Se han encontrado trabajos donde se crea tráfico realista y topologías de varios AS [1] con su código disponible y además se han encontrado trabajos donde se agregan protocolos a la suite para OMNeT++ denominada INET [2] y su discusión. Por otro lado no se han encontrado trabajos relacionados con la simulación de un equipo DSLAM en sí, ni de pruebas intensivas de tráfico sobre nodos de acceso. Se ha estudiado acerca de varias herramientas de simulación de redes como lo son OMNeT++, IKR Simulation Library, Open WNS, ns-2 y ns-3. Como se describe más adelante en el presente capítulo la herramienta elegida para el trabajo es OMNeT++ dado que los trabajos relacionados encontrados utilizaban esta herramienta, y ya era conocida por el grupo de trabajo de la empresa Movistar, de hecho en [3] también se ocupa esta herramienta. Existe variada documentación acerca de los protocolos, de modelos de tráfico [4], de topología y de modelos para nodos de acceso. Permitiendo tener información suficiente para el trabajo a realizar. En resumen se tienen para los siguientes aspectos una base para el desarrollo del trabajo:

- Modelo de nodo de acceso (Dslam): Modelo de un router según teoría de colas útil para pasar a DSLAM y varios modelos más complejos de DSLAM.
- Tráfico: Se tiene estudios previos basados en modelos de tráfico tradicionales y de autosimilitud. Y se tiene la implementación en OMNeT del modelo ReaSE de un modelo de tráfico que presenta comportamiento autosimilar. Se tienen también los datos de tráfico de la empresa Movistar.
- Topología de red: Se tiene la información de Movistar sobre cómo es la topología de su red, además se pretende que a nivel de acceso y de usuarios esta topología pueda cambiar en base a parámetros designados mediante una interfaz gráfica.
- Herramientas de simulación como OMNeT++, librerías INET y ReaSE.

2.1.1 Modelos de nodos de accesos DSLAM

Desde una perspectiva de alto nivel, la arquitectura de un ATM DSLAMs, Ethernet DSLAMs e IP-DSLAMs típicamente incluyen un número de tarjetas de línea xDSL que terminan el loop del abonado local y una o más tarjetas uplink ATM OC-3/12/48 o Ethernet/Gigabit Ethernet para el tráfico entre la capa de acceso y la de distribución. Las tarjetas de línea y la tarjeta del uplink están interconectadas por un backplane de agregación de alta capacidad que puede tomar la forma de un puente ATM o Ethernet o switch. La mayoría de los DSLAM modernos son multiservicio y soportan multiples tecnologías DSL, es decir, ADSL, ADSL2, ADSL2+, SDSL y VDSL, etc y por lo tanto estos dispositivos se adaptan a múltiples tipos de tarjetas de línea xDSL (ver [5]).

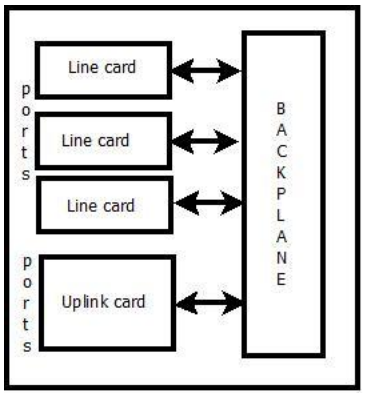


Figura 2.1: Arquitectura de alto nivel de un nodo DSLAM

Desde la perspectiva del procesamiento del tráfico, han emergido dos modelos distintos, el centralizado y el distribuido. En el modelo centralizado todo el procesamiento complejo de tráfico (Clasificación, filtrado, QoS, etc.) es realizado por una única tarjeta de uplink central. Las tarjetas de línea en el modelo centralizado son simples y contienen solo los componentes básicos requeridos para rutear el tráfico hacia la tarjeta del uplink. La arquitectura centralizada es considerada mejor aplicada para DSLAM centrados en la agregación de gran escala y de alta densidad con requerimientos de complejidad moderada sobre el procesamiento del tráfico.

En el modelo distribuido algunos o todos los procesamientos complejos de tráfico son cargados sobre las tarjetas de línea inteligente basadas en procesadores de red programables (Linecard Traffic Processors o LTPs). La tarjeta del uplink in esta arquitectura puede ser tan simple como un switch Ethernet en caso de conexión Ethernet, o también puede tener un procesador completo para escenarios más complejos (IPoMPLS).

El modelo distribuido prevalece en los DSLAM con capacidades de procesamiento de tráfico complejas, como IP-DSLAM con funcionalidades de capa 3 o de QoS.

El modelo de arquitectura distribuida de DSLAM tiene varias ventajas importantes sobre el modelo centralizado tales como capacidad de procesamiento local de tráfico en las tarjetas de línea (multicasting local y peer-to-peer local) y presentan un costo de expansión lineal ya que un uplink barato genera bajo costo de entrada y su capacidad puede ser ampliada a medida que se instalan tarjetas de línea adicionales (paga a medida que creces).

Visto desde el punto de vista simplificador el nodo de acceso puede verse interpretado como un Router y un modelo conveniente se puede encontrar en [6].

Partiendo de un modelo con múltiples links de entradas y múltiples links de salidas utilizando teoría de colas se llega a que como las entradas llegan a un solo procesador éstas pueden ser vistas como una sola cola. Por otro lado las colas de salida pueden tener pérdidas de paquetes por diversas razones, por esto y también por el procesamiento de los mensajes en funciones como diferenciación con QoS se agrega un servidor al final de las colas de salida.

El modelo según teoría de colas:

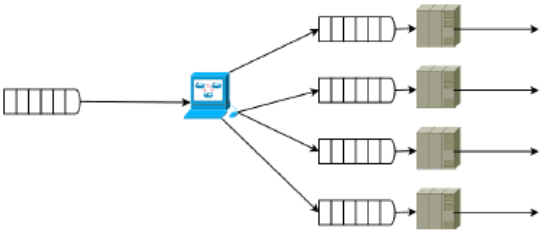


Figura 2.2: Modelo final de Router.

Este modelo, con algunas variaciones, es el que se aplica en el presente trabajo, con límite de bytes para los paquetes en la capa de transporte para la cola de entrada al módulo procesador IP además de un retardo en el procesador y en las colas de salida encapsulamiento PPP con límite de capacidad para la cola de salida dado por un límite en la cantidad de paquetes.

Ahora que se tiene el modelo de DSLAM o nodo de acceso para la simulación es importante conocer las distintas topologías que se tienen en red y cuál es la que se utiliza en el presente trabajo.

2.1.2 Modelos de red

Dentro de la compañía MOVISTAR CHILE un modelo simplificado y útil de su red, desde el punto de vista de un IP-DSLAM, es aquel donde a la MEN se conecta el uplink del IP-DSLAM, el servidor de video y el servidor de voz sobre IP. Además el BRAS le da la conexión a internet a toda la MEN y permite la gestión de todo el tráfico.

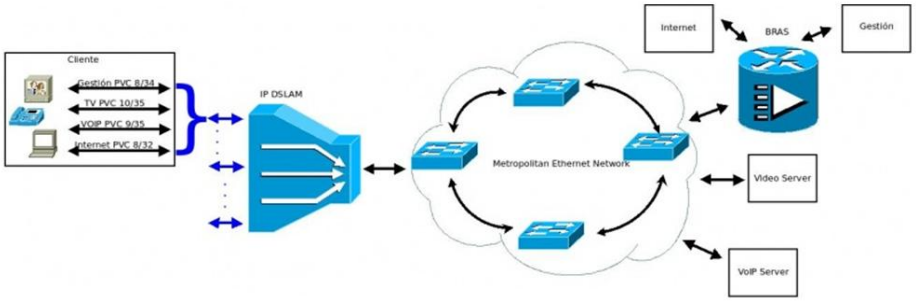


Figura 2.3: Modelo simplificado de Movistar Chile desde el punto de vista de un IP-DSLAM.

Para llevar el anterior modelo a un modelo simulable se reemplaza la MEN por dos routers y el BRAS por un router que se conecta a servidores que generan tráfico como los que uno podría encontrar en Internet con diferentes tipos de servicios.

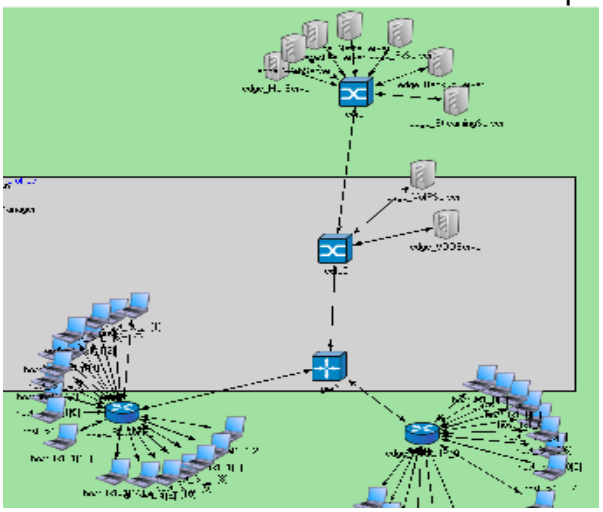


Figura 2.4: Modelo simplificado de red simulable

2.1.3 Modelos de tráfico

Aprovechado el trabajo de tesis [4] se tienen los siguientes modelos para describir el tráfico en redes de datos.

Modelos tradicionales:

Cadenas de Markov.
Procesos de Poisson modulados por Markov.
Procesos de Bernoulli modulados por Markov.
Modelos regresivos.

Modelos basados en Autosimilitud:

Dependencia a Largo Plazo.
Modelos FARIMA.
Movimientos browniano y Gaussiano.
Superposición de fuentes ON-OFF de alta variabilidad.

De los modelos presentados anteriormente es el de superposición de fuentes ON-OFF de alta variabilidad el que se intenta en primera instancia aplicar en el presente trabajo, por lo que se procede a explicar sus características:
Es un modelo que fue sugerido por primera vez en 1986, utilizando el concepto de "*trenes de paquetes"*, que consisten básicamente en una fuente de dos estados, el primer estado "ON" ocurre durante el período durante el cual se emiten paquetes a una tasa arbitraria y el segundo estado "OFF" ocurre cuando no se emiten paquetes. La superposición de numerosas fuentes de este tipo consigue obtener un tráfico agregado que exhibe un comportamiento de dependencia de largo plazo y este fenómeno se denomina como *Efecto Noah*.

Para tener este comportamiento se requiere cumplir con lo siguiente:

- Las fuentes transmiten a una tasa R durante el período ON y no transmitirán durante el período OFF
- El tiempo de permanencia en los períodos ON, es una variable aleatoria independiente para cada fuente con una distribución Pareto de media finita y de varianza infinita.
- Los períodos de permanencia en el estado OFF son variables aleatorias con distribuciones genéricas y de media finita.

Si bien idealmente el tráfico más realista es el autosimilar no es el propósito del presente trabajo que el tráfico simulado lo sea. Por este motivo se puede tener tráfico cliente-servidor con parámetros constantes para cada tipo de aplicación, es decir, sin variar el tamaño y/o tiempo entre las peticiones o respuestas.

Datos de tráfico

Se cuenta además con mediciones de tráfico por parte de la empresa Movistar donde las figuras 2.5 y 2.6 son ejemplos del tipo de datos que se tiene y donde la medición corresponde a la cantidad de tráfico acumulado al momento de desconexión de los usuarios.

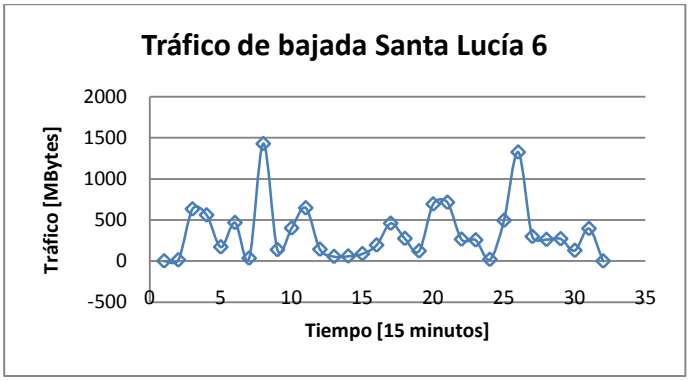


Figura 2.5: Tráfico de bajada por DSLAM Santa Lucía, 1 de Marzo de 2011.

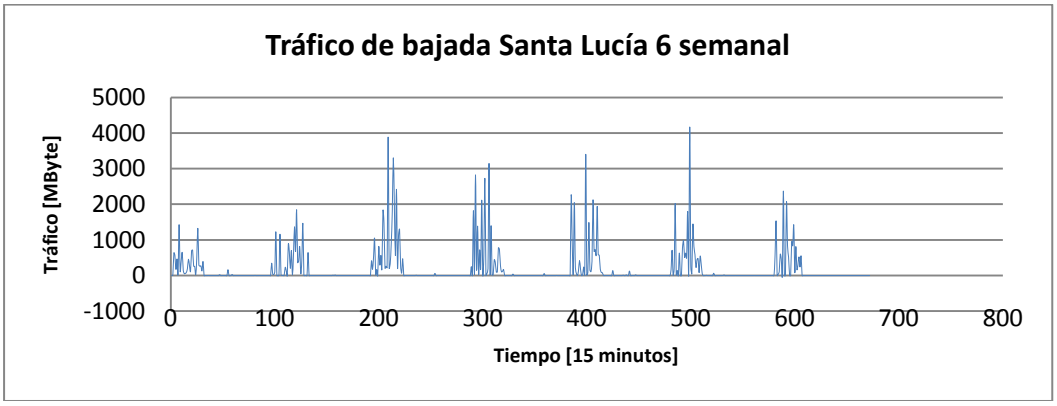


Figura 2.6: Tráfico de bajada por DSLAM Santa Lucía 6, 1 al 7 de Marzo de 2011.

2.2 Desafíos para la simulación

De manera general es importante para la modelación y simulación tener en cuenta los aspectos presentados a continuación.

Simular computacionalmente se hace necesario cuando la solución analítica del problema no es posible o no es factible y cuando ésta sí es posible se puede utilizar la simulación para poder estar seguros que la modelación analítica fue correcta. También se hace necesario para sistemas dinámicos que involucran aleatoriedad y cambios de estado en el tiempo y para sistemas dinámicos complejos, los cuales son tan complejos que requieren demasiadas simplificaciones para analizarlos teóricamente.

Para poder simular es necesario generar un modelo de simulación y para su validación, se debe basar en un modelo de rendimiento, aunque muchas veces algunos programadores consideran que el modelar es básicamente lo mismo que programar el modelo, pero en realidad es importante diferenciar entre estos dos modelos.

Es importante destacar que el modelo depende del propósito de la simulación y no viceversa. A continuación se presentan requerimientos para los modelos de rendimiento y de simulación.

- Por un lado, todos los modelos de rendimiento deberían ser simples, creíbles y bien documentados. Simples como puedan serlo pero no más simples, creíbles para que estén validados y se puedan tomar decisiones en base a él, y bien documentados para poder tener en cuenta las simplificaciones que se realizaron dado el propósito para el cual se construye el modelo de rendimiento, además debe estar claro lo que se dejó de considerar en el modelo y por lo tanto para que propósitos no es útil.
- Los modelos de simulación, aquellos que son programados, son más específicos que los modelos de rendimiento por lo tanto deben preocuparse por ser eficientes, verificados, tener calidad en el código y por estar disponibles. Eficientes con respecto a la implementación del modelo de rendimiento, para que de esta forma se ejecute la simulación de manera acotada (tomando en cuenta la complejidad del modelo de rendimiento) y así poder tener una campaña de investigación completa. Verificados, es decir, que la correspondencia entre el modelo de rendimiento y el modelo de simulación esté verificada por varios métodos. Calidad en el código, se debe mantener un estilo en la programación, se debe usar orientación a objetos en conjunto con documentación suficiente. Debe estar disponible para que se pueda verificar y validar los modelos por parte de otros grupos.

Es importante tener en cuenta las buenas prácticas del modelado para simulación en computador las cuales se presentan a continuación.

- Formulación del problema y definición del sistema/modelo: Las metas de estudio deben definirse primero.
- Elección de las métricas, de los factores y de los niveles: Los factores son parámetros de un sistema o modelo que varían durante el estudio de evaluación, sus valores numéricos durante la evaluación se denominan niveles.
- Recolección de datos y modelación: Se debe tener información del sistema para poder crear el modelo de rendimiento. Se debe identificar las entradas y salidas, las cuales serán utilizadas para validar el modelo de rendimiento. Se describen los elementos y sus interacciones de manera iterativa.
- Elección del ambiente de simulación, implementación del modelo y verificación: La elección de la herramienta de simulación puede depender de los modelos para las diferentes capas, librerías para tráfico, herramientas de depuración, lenguaje en que está escrito, gráficos, etc. basados en esa decisión el modelo se debe implementar y luego verificar.
- Validación y análisis de sensibilidad: El modelo de rendimiento de ser validado con respecto al sistema real en su métrica, factores y niveles, usualmente se realiza comparando las salidas con el sistema real pero cuando éste no está disponible se pueden usar diversas técnicas como comparar con los resultados de un análisis matemático, etc.
- Experimentación, análisis y presentación: una vez que se concluye la validación se procede a la experimentación, su posterior análisis y finalmente sus gráficos.

Para tomar la decisión de dónde simular hay que tomar en cuenta que existen diferentes técnicas de simulación:

Emulación: El proceso de diseñar y construir hardware o firmware (hacer un prototipo) que imita la funcionalidad del sistema real.

Simulación de Monte Carlo: Cualquier simulación que no tiene eje del tiempo, esta simulación se utiliza para modelar fenómenos probabilísticos que no cambian con el tiempo, o para evaluar expresiones no probabilísticas usando técnicas de probabilidades.

Simulación dirigida por traza: Cualquier simulación que utiliza una lista ordenada del mundo real como entrada.

Simulación de eventos continuos: En algunos sistemas el cambio de estado ocurre todo el tiempo, no meramente a tiempos discretos. En estos casos la “simulación continua” es más apropiada, aunque la simulación de eventos discretos puede servir como aproximación.

Simulación de eventos discretos: Tiene dos características. La primera es que para cualquier intervalo de tiempo uno puede encontrar un sub-intervalo en el cual ningún

evento sucede y no cambia ninguna variable de estado. La segunda es que el número de eventos es finito.

A continuación se describe los errores que se deben evitar cometer al simular [7]:

Es importante para la simulación evitar caer en errores como lo son agregar demasiado detalle a los modelos de simulación pues, como estos no se basan en análisis sino que en computación, se puede estar tentado a detallar demasiado, traduciéndose en mayor tiempo de desarrollo y de computación. También existe un trade-off entre los lenguajes a utilizar pues si se elige un lenguaje dedicado éste tomará menos tiempo en el desarrollo pero si se elige un lenguaje de uso general este correrá más rápido.

Dado que las simulaciones son grandes programas estos podrían contener errores de programación o lógicos por esto es importante que estén verificados. Los programas pueden no contener errores pero aún así no representar el comportamiento del sistema real, por esto habrá que revisar los supuestos matemáticos del modelo. Se hace muy importante una apropiada elección de los parámetros iniciales pues en su defecto podría llevar a estados incorrectos. Las simulaciones cortas llevan a valores que dependen fuertemente de los estados iniciales y que pueden ser incorrectas. Elegir bien las semillas del RNG (random number generator) para evitar posibles correlaciones entre procesos de la simulación que invaliden los resultados.

Estimación de tiempo inadecuada, el desarrollo, la implementación y las pruebas requieren de mucho tiempo y esfuerzo. Metas no alcanzables porque estás no fueron puestas en un principio y no se pueden lograr con el modelo final. Mezcla incompleta de habilidades, el proyecto requiere individuos con diversos conocimientos, de estadística y modelado, programadores, etc. Falta de participación del usuario final, para poder lograr éxito debe haber comunicación en reuniones con los usuarios finales. Inhabilidad para manejar proyectos demasiado complejos, se requiere entonces ingeniería de software.

2.3 Herramientas de simulación de libre uso

Comparación [8]:

NS2

Este simulador contiene multitud de módulos que permiten definir escenarios basados en la pila TCP/IP, MPLS, redes inalámbricas, redes de satélite, unicast y multicast, etc. Los escenarios se definen mediante scripts en o TCL (Tool Command Language). Es de código abierto, se incorporan continuamente nuevos módulos y funcionalidades. Los objetos y módulos están programados en C++, es muy extensible. Su alto grado de flexibilidad se ha transformado en una fuente de errores por lo que sus

modelos simulables no son muy confiables. Es ampliamente usado en grupos de investigación y existe muchísima documentación.

NS-3

Fue diseñado desde 2005 para remplazar a ns-2 el cual era extremadamente popular, con cientos de modelos que contribuyeron a al código base de ns-2. Nació de la necesidad de modelar pilas de red, de forma tal que calce de mejor manera con la investigación en las redes. Una de las metas de nd-3 es mejorar el realismo de los modelos, es decir, hacer de los modelos más cercanos en implementación a las implementaciones de software que representan. ns-3 utiliza C++ como código fuente, en parte porque facilita la inclusión de implementaciones basadas en C. Tiene capacidades de emulación que permiten hacer de banco de pruebas con dispositivos reales y aplicaciones, además es de licencia GNU.

Entonces ns-3 se enfoca en:

- Realismo
- Reutilización
- Facilidad de depuración
- Buena mantención
- No hay compatibilidad hacia atrás debido a que ns-2 era demasiado flexible y sus modelos contenían errores.

IKR Simulation Library

La librería de simulación (SimLib) del instituto de redes de comunicaciones e ingeniería de computación (IKR) de la Universidad de Stutgart es una herramienta para la simulación orientada a eventos de sistemas complejos en el área de la ingeniería de las comunicaciones. Su historia viene desde 1980, en 1993 pasó de Pascal a C++. En 2008 se pasó a Java manteniendo los conceptos y mecanismos de la librería existente en C++. Ahora hay dos ediciones de IKR SimLib disponibles: La edición en C++ y la edición en Jaca. IKR está públicamente disponible bajo licencia GNU (LGPL). Esta condición lo hace sumamente útil para quienes solo sabes programar en Java o les conviene más pues es el único simulador cuyas librerías están en java.

Open WNS

Open Source Wireless Network Simulator es una plataforma de simulación de código abierto para sistemas de comunicación móviles wireless y multi-celulares. Su objetivo es desarrollar una plataforma de simulación de nivel de sistema de código abierto (openWNS) para la evaluación y comparación del rendimiento de sistemas de comunicación móviles wireless y multi-celular. La plataforma de simulación ofrecerá una cercana-a-emulación implementación de los respectivos protocolos, incluyendo la implementación detallada de interfaz de modelamiento en escenarios de referencia, modelos de movilidad, generadores de carga de tráfico, métodos de evaluación

estadística y modelos de canales detallados. Al estar, este simulador, orientado a las tecnologías móviles, no es seleccionado entre los útiles para el presente trabajo.

OMNeT++:

El ambiente de simulación de eventos discretos OMNeT++ ha estado públicamente disponible desde 1997. Se creó teniendo en mente la simulación de redes de comunicaciones, multiprocesadores y con otros sistemas distribuidos en mente como área de aplicación pero en vez de especializarse, OMNeT++ fue diseñado para ser lo más general posible. Esta idea ha probado funcionar y se ha utilizado en redes de colas hasta simulaciones de sistemas inalámbricos, desde simulaciones de procesos de negocios como en redes peer-to-peer, switches ópticos y redes de almacenaje se han simulado en OMNeT++ (ver [9]).
OMNeT++ fue diseñado desde el comienzo para soportar la simulación de redes a gran escala. Este objetivo llevó a los siguientes requerimientos de diseño principales:

- Los modelos de simulación deben ser jerarquizados para permitir simulaciones de gran escala y deben estar construidos de componentes reusables tanto como sea posible.
- La simulación debe facilitar visualización y depuración del modelo de simulación pues esto tradicionalmente tarda un gran porcentaje de los proyectos de simulación.
- El software de simulación en sí mismo debe ser modular, editable y debe permitir introducir simulaciones dentro de otras más grandes como en el caso del software de planificación de redes.
- Las interfaces de datos deben ser abiertas: debe ser posible generar y procesar archivos de entrada y salida con herramientas de software comúnmente disponibles.
- Debe proveer un Ambiente de Desarrollo Integrado que facilite el desarrollo y el análisis de resultados.

Estructura del modelo:

Un modelo en OMNeT++ consiste en módulos que se comunican pasándose mensajes. Los módulos activos se denominan “módulos simples”; ellos están escritos en C++, utilizando la librería de clases de la simulación. Los módulos simples pueden ser agrupados en “módulos compuestos” y el número de niveles jerárquicos no está limitado.

Los módulos compuestos y los módulos simples se pueden interconectara través de compuertas y canales pero finalmente todo el sistema queda inmerso en un gran módulo red el cual no tiene compuertas ni se comunica con otros módulos.

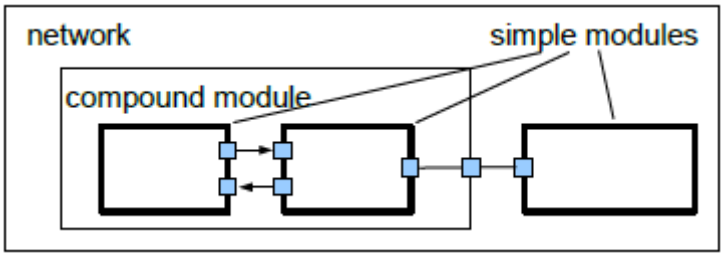


Figura 2.7: Modelo de red con módulos compuestos y simples en OMNeT++.

Los módulos se comunican a través de mensajes los cuales, en adición a la marca de tiempo, pueden contener información arbitraria. Los módulos simples típicamente se comunican a través de compuertas pero también es posible hacerlos llegar directamente al módulo de destino.

Las compuertas (gates) son las interfaces de entrada y salida de los módulos. Las conexiones que van de compuerta a compuerta se denominan canales (channels), estás pueden tener atributos como ancho de banda, ber, etc.

Los módulos también pueden tener parámetros los cuales son utilizados para pasar información de la configuración a los módulos simples y pueden ser accesados desde la parte C++ de los módulos simples. Los parámetros de los módulos pueden ser traspasados a los sus sub-módulos.

Lenguaje NED:

El usuario define la estructura del modelo (los módulos y sus interconexiones) en el lenguaje de descripción de la topología de OMNeT++, NED. Los elementos de una descripción NED son las declaraciones de los módulos simples, las definiciones de los módulos compuestos y las definiciones de la red.

El lenguaje NED ha sido diseñado para ser escalable y lo está logrando pero a medida que crecen los modelos en complejidad, el lenguaje se ha ido adaptando. Las características principales que se han introducido son:

- Herencia
- Interfaces
- Paquetes
- Tipos internos
- Apuntes de metadata

Librería INET de OMNeT++

El sistema INET se construye sobre OMNeT++, es una extensión de éste, y utiliza el mismo concepto: módulos que se comunican pasándose mensajes. Host, routers, switches y otros dispositivos de redes son representados por módulos compuestos de OMNeT++. Estos módulos compuestos son armados desde módulos simples que representan protocolos, aplicaciones y otras unidades funcionales. Una red es un módulo compuesto de OMNeT++ que contiene hosts, routers y otros módulos. Las interfaces externas de los módulos están descritas en los archivos NED. Los archivos NED describen los parámetros y compuertas de los módulos, y también los submódulos y conexiones de los módulos compuestos [2].

Los módulos se organizan dentro de paquetes jerárquicos que se mapean dentro de un árbol de directorios (carpetas), de forma muy similar a los paquetes en Java. Los paquetes en INET están organizados de acuerdo a las capas del modelo OSI. Las subcarpetas dentro de los paquetes usualmente corresponden a protocolos concretos o a familia de protocolos.

La herramienta INET contiene la implementación de los protocolos IPv4, IPv6, TCP, SCTP, UDP,PPP, Ethernet, 802.11 y varios modelos de aplicaciones.

Librería ReaSE de OMNeT++

La mayoría de los proyectos de investigación tanto académicos como industriales, utilizan la simulación para evaluar sus productos. De ahí la importancia de que las simulaciones se basen en un ambiente realista para que los resultados de las pruebas tengan validez, por esto a continuación se muestra cómo la herramienta ReaSE [1] actúa en los diferentes aspectos.

Para crear topologías de nivel AS existen dos enfoques: los basados en observaciones reales y los basados en un generador de topologías aleatorio. El primer enfoque tiene dos problemas: es difícil obtener todos los datos necesarios para una sola simulación y el otro problema es que las topologías van cambiando en el tiempo. Por esto se decide trabajar con la metodología de generación de topologías aleatorias. Otras herramientas como BRITE e Inet siguen la ley de potencias para sus topologías, esto significa que muchos nodos tienen pocas conexiones y solo unos pocos tienen varias, sin embargo BRITE ya no se mantiene e Inet tiene problemas para la creación de topología de nivel de router. De aquí la creación de ReaSE la cual es una herramienta basada en el modelo de preferencia de retroalimentación positiva (PFP) [24]. Este es un modelo iterativo de creación de topologías que sigue la ley de potencias en su distribución de nodos aleatoria.

Para las topologías de nivel de router se hace más difícil aún la obtención de información por parte de los ISP pues estos prefieren no mostrar la topología de sus redes de nivel de router por razones comerciales. Por esto se prefiere utilizar nuevamente generadores de topologías aleatorios.

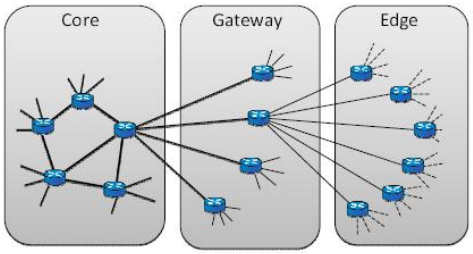


Figura 2.8: Topología de routers jerárquica.

En resumen el ancho de banda aumenta desde los nodos *edge* hasta los nodos *core* mientras que la conectividad disminuye.

En el tema de la generación de tráfico se requiere, para cumplir con condiciones de realismo, que el tráfico sea autosimilar y que provenga de una razonable cantidad de mezclas de diferentes tipos de tráfico. En la implementación de ReaSE se cumple con los dos criterios anteriores utilizando múltiples fuentes de tráfico que son prendidas y apagadas en intervalos distribuidos en larga-cola y utilizando paquetes de tamaño distribuidos en larga-cola con diferentes flujos de tráfico. El tráfico es generado a partir de los hosts que emiten peticiones y reciben respuestas por parte de los servidores. El diagrama de secuencia detallado se muestra en la figura 2.9.

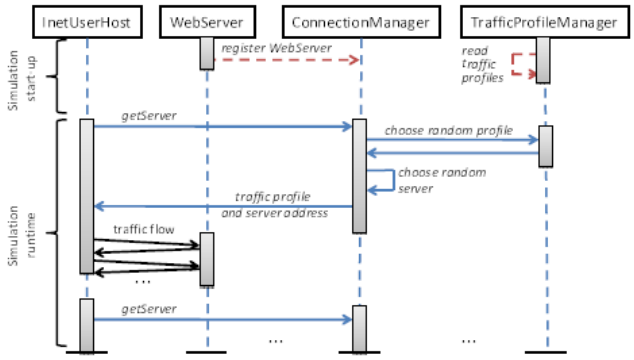


Figura 2.9: Diagrama de secuencia de la administración de las conexiones y el tráfico.

Uno de los aspectos más interesantes es la adición del concepto de usuario a cada host, permitiendo comenzar flujo de tráfico en un instante dado y conectándose con el administrador de conexiones para elegir el tipo de tráfico y el servidor. El modelo de secuencia en la figura 2.9 es sumamente claro y deja ver cómo se comunican los diferentes módulos del sistema para la creación de tráfico a través de la red. Además la

herramienta ReaSE extiende las librerías INET de OMNeT++ para incorporar nodos maliciosos y así simular ataques.

Por sus ventajas de encontrar trabajos relacionados, rapidez, modularidad y fácil aprendizaje, se ocupa OMNeT++ junto con INET y con la ayuda de ReaSE para la creación del tráfico en el presente trabajo.

2.4 Mediciones

Mediciones de tráfico: Se requiere tener la medición de los tráficos de subida y de bajada por un DSLAM antes y después de pasar por éste. De esta forma poder deducir la probabilidad de pérdida de paquetes asociado a tal DSLAM y al respectivo tráfico.

Pérdida de paquetes: Ocurre cuando uno o más de los paquetes de datos viajando a través de una red de computadores no logra llegar a su destino. Es uno de los tres errores más importantes en las comunicaciones digitales.

Retardo máximo: Tiempo máximo que toma a los paquetes la ida, la vuelta o la ida y vuelta. (Ver [10])

Clientes con retardos mayores a un retardo límite: Se mide la cantidad de clientes que superan un retardo límite en un intervalo de tiempo dado.

2.5 Herramienta de clasificación estadística del tráfico

Existe la necesidad, desde la perspectiva de un proveedor de ADSL, de poder identificar el tráfico que cursa por sus redes y hay diferentes motivaciones para la clasificación del tráfico como lo son: Ejecución de reglas internas o nacionales; Mejor entendimiento de aplicaciones actuales o emergentes; Valoración del impacto de aplicaciones con acuerdo de interconexión o el retorno sobre la inversión si existen iniciativas p4p; La posibilidad de ofrecer servicios adicionales basados en la aplicación como por ejemplo la protección de transferencias multimedia.

Usos típicos del clasificador estadístico son: Utilizarlo en PoPs donde no hay DPI disponibles basados en la información obtenida de los PoPs donde sí hay DPI disponibles; Otro uso es el de ayudar a disminuir el porcentaje de tráfico clasificado como desconocido en los PoPs donde sí hay DPI disponibles, el cual usualmente está entre el 8 y el 24%.

De [11] surgen las preguntas: ¿Se puede obtener una alta precisión en la clasificación de tráfico, y esto, para todas las aplicaciones de interés? ¿Pueden los métodos estadísticos ayudar a la minería de datos que el DPI no puede clasificar? ¿Pueden los modelos estadísticos ser representativos de las aplicaciones, es decir, pueden ser entrenados en un sitio y ser ocupados en otro sin ajustes específicos o re entrenamiento? ¿Puede usarse como alternativa a las herramientas comerciales DPI?

Existen dos grandes casos para la clasificación estadística. El caso donde se aplica el clasificador en el mismo lugar en que fue entrenado y el caso donde se aplica el clasificador en un lugar diferente a donde fue entrenado.

Aplicación en el mismo lugar en que fue entrenado.

- La clasificación estadística es lo suficientemente flexible para agrupar el tráfico según aplicación
- La clasificación estadística puede ayudar a revelar el porcentaje de tráfico dejado como *desconocido* dividiendo por un factor de dos esta cantidad.

El clasificador estadístico es utilizado en un sitio diferente a donde fue entrenado.

- El rendimiento promedio es bueno cuando se consideran todos los flujos y aplicaciones pero el resultado se deteriora significativamente cuando se toman como base las aplicaciones. Esto significa que algunas aplicaciones que son correctamente clasificadas en el sitio en el cual el clasificador fue entrenado, se vuelven difíciles de identificar cuando se aplica a un sitio diferente. Esto se debe a que el clasificador aprende características específicas del sitio utilizadas por los usuarios y/o aplicaciones (*overfitting*).

Colección de datos utilizada en el trabajo de investigación [11]:

La base de datos se compone de cuatro mediciones de paquetes recogidos de tres ADSL PoPs diferentes. Los datos fueron tomados usando sondas pasivas bajo un BAS que rutea tráfico hacia y desde los DSLAM al internet.

Algoritmo de clasificación utilizado en el trabajo de investigación [11]:

Se utiliza un algoritmo C4.5 que es un árbol de decisión [12], construye un modelo basado en una estructura de árbol. Se basa entonces en la implementación del algoritmo C4.5 proveído por Weka.

Características:

Se utilizan dos conjuntos de características que se evalúan independientemente. El primer conjunto (conjunto A) utiliza el tamaño y la dirección de los primeros paquetes de datos de una transferencia. El segundo conjunto (conjunto B) utiliza la siguiente información para caracterizar el tráfico:

- Push_pkt_down: Cantidad de paquetes con bandera *Push* de bajada .
- Push_pkt_up: Cantidad de paquetes con bandera *Push* de subida.
- Avg_seg_size_down: Datos en bytes dividido por la cantidad de paquetes.

- Min_seg_size_down: Tamaño de segmento mínimo de bajada.
- Data_pkt_down: Paquetes con carga útil de najada
- Pkt_size_median_up: Mediana del tamaño de paquetes de subida.
- Local port: Puerto TCP local.
- Distant port: Puerto TCP Distante.

Se tienen tres definiciones del flujo:

- S/S: Flujos con negociación en tres pasos.
- SS+4D: Flujos con negociación en tres pasos y con al menos cuatro paquetes de datos.
- SS+FR: Flujos con negociación en tres pasos y con un *flag* FIN o RST al final de la transferencia de datos.

De donde se elige utilizar la definición del flujo SS+4D debido a que SS+FR deja fuera porcentajes muy altos de los datos examinados.

Se utilizan tres métricas para el rendimiento

- *Accuracy:* Corresponde a la fracción de flujos de una clase específica correctamente clasificados. Es la razón de los verdaderos positivos a la suma de los verdaderos positivos con los falsos negativos.
- Precision: Para una clase dada es la razón de los verdaderos positivos a la suma de los verdaderos positivos y los falsos positivos.
- Precision total: Razón de la suma de todos los verdaderos positivos a la suma de todos los verdaderos positivos y falsos positivos.

Clasificación estadística – caso estático:

En los casos estáticos se investiga el rendimiento de la clasificación estadística en cada sitio independientemente de los otros sitios. Se entrena el algoritmo con un 10% de los datos de un sitio. Se realizan 10 experimentos en cada sitio tomando un décimo de los datos cada vez y los resultados son el promedio de estos diez experimentos para cada sitio.

Como se tiene el tamaño de los primeros paquetes de datos como una característica de la clasificación se debe elegir el número de paquetes (k) que se ocupe. Al observar las evoluciones de la precisión y de la *accuracy* para los valores incrementales k de paquetes y se decide un valor de cuatro, lo que está en línea con las recomendaciones [13].

Cuando se corre el clasificador en el mismo sitio en que fue entrenado se obtienen accuracies totales que superan el 90% tanto para el conjunto de características A como para el conjunto de características B. Esto ocurre porque las aplicaciones mayoritarias como WEB e EDONKEY son siempre bien clasificadas pero cuando se tiene una observación desde el punto de vista de las aplicaciones se notan

diferentes porcentajes de precisión y *accuracies*. Aplicaciones como GAMES genera dificultades para el clasificador estadístico, lo que se entiende del hecho de que esta aplicación agrega otras aplicaciones de un gran número de comportamientos.

Para el caso estático se obtienen buenos resultados pero lo que se quiere saber es si se puede utilizar un clasificador estadístico en un lugar diferente al donde se entrenó, para así no depender de un DPI en el lugar que se utilice el clasificador.

Entrenar el clasificador en un sitio y aplicarlo en otro. Puede servir para ISP que entrenen el clasificador en un PoP grande y luego que lo utilicen en otros PoPs. Para el conjunto de características A se tiene como resultado que la similitud espacial es más importante que similitud temporal. Se tiene una clasificación correcta en un 95% de los casos pero cuando se usan sitios cruzados se degradan mucho los resultados debido al *overfitting.*

Los resultados obtenidos para el conjunto de características B son similares a los del A mostrando también el fenómeno de *overfitting* al cambiar, el clasificador, de sitio.

Se puede discernir entonces que: Entrenar un clasificador en un sitio antes de usarlo en otro puede llevar a resultados imprevistos; El estudio de sitios cruzados permite revelar problemas que no podrían haber sido descubiertos de otra manera; Una vez que se entrena un clasificador en un sitio puede usarse por mucho tiempo en el mismo, aunque falta más estudio al respecto pues se tiene una separación de alrededor de solo dos semanas.

Ahora para responder a la pregunta de si se puede utilizar el clasificador estadístico para reducir la cantidad de flujos clasificados como desconocidos por el DPI, se utiliza un método diferente para comparar los resultados el cual consiste en las direcciones ip de los puntos finales y los puertos utilizados. Se obtiene finalmente que, se puede reducir a la mitad los flujos de tráfico desconocidos.

En conclusión se tiene que la herramienta de clasificación estadística: Resulta útil para determinar el tráfico indefinido por las herramientas DPI; Tiene un alto rendimiento al aplicarse en el mismo sitio en el que fue entrenada; Permite identificar aplicaciones más allá de los protocolos; Tiene como punto negativo que puede sufrir de *overffiting* lo que impide la utilización en sitios diferentes al que fue entrenada.

2.6 Estimación de capacidad de enlaces ADSL

Existe, para el caso no colaborativo la herramienta DSLprobe mostrada en [13] y describa a continuación.

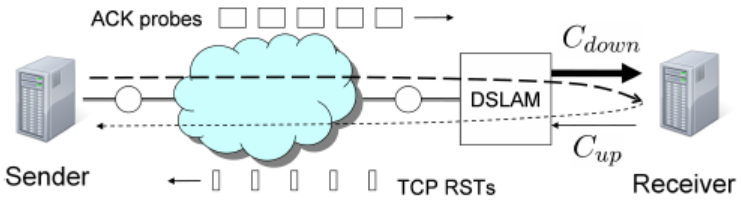


Figura 2.10: Estimación no cooperativa.

En el trabajo [14] se aprovecha características del tráfico TCP para estimar la capacidad de canal de subida y de bajada de un host de forma no colaborativa. En TCP se sabe que los tamaños de los RST son de 40 bytes y es denominado S_{rst} también se tiene que el tamaño de los mensajes ACK puede variar desde los 40 bytes hasta el tamaño del MTU (usualmente 1500 bytes) y se denomina S_{ack}, la proporción entre ambos $S_{A/R}$ y al enviar ACK por parte del transmisor hacia el receptor, se tiene que cada uno de ellos genera un mensaje RST de vuelta. Se tiene entonces que las tasas de transmisión R_{rst} y R_{ack} son iguales cuando se utiliza un S_{ack} = 40 bytes y las dispersiones medidas con el tren de paquetes se ajusta a el enlace de menor capacidad, el cual es usualmente el enlace de subida ADSL. Para medir el enlace de bajada se debe aumentar el tamaño de los mensajes ACK de tal forma de que la carga de los mensajes ACK's excedan la capacidad del enlace de bajada antes de que los RST saturen el enlace de subida $R_{rst} < C_{up}$, Esto no es un problema pues si los mensajes ACK's tienen 1500 bytes y dado que $R_{rst} = {R_{ack}}/{S_{A/R}}$ y se tiene que $S_{A/R}$ = 1500/40 = 37,5 entonces la proporción de las capacidades entre la bajada y la subida no puede ser mayor a 37,5 para tener mediciones correctas, pero en la práctica en los enlaces ADSL esta proporción no es mayor a 16 $C_{down/up} = 16 < S_{A/R} = 37{,}5$ por lo que no sería un problema para las mediciones usuales.

Estimación de capacidad en presencia de tráfico cruzado:

Un gran problema para la correcta estimación de las capacidades es la filtración o reducción de la interferencia producida por el tráfico cruzado. Para obtener buenos resultados se aprovechan las siguientes características:

- Trenes de alta velocidad.
- Identificador IP (IPID)
- Tiempo de arribo entre paquetes (IAT)
- Números en secuencia

Al escoger los trenes de paquetes existe un *trade off* en la cantidad de paquetes a elegir, pues a mayor cantidad de paquetes, menor es la influencia del tráfico cruzado y

para trenes más pequeños se tiene una menor probabilidad de interferencia. Por estas razones se elige trabajar con trenes de 50 ACK, luego si es que hay pérdidas, el tren se reduce a 25, 10 y eventualmente a 5 paquetes [15].

Algoritmo de medición de DSLprobe

Cuando se estima el enlace de subida se detecta el tráfico cruzado de dos formas. La primera forma es verificando que los IPIDs de los RST sean consecutivos. La segunda forma es verificando que la IAT no sea mayor que dos veces la mediana de la IAT del tren.

Cuando se estima el enlace de bajada se pueden tener los casos o problemas que se muestran en la figura 2.11.

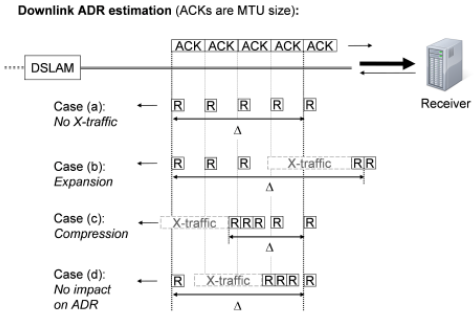


Figura 2.11: Casos posibles cuando las mediciones se enfrentan a tráfico cruzado.

Se utilizan diversos criterios para detectar estos problemas y, de existir, se mide nuevamente con trenes más pequeños y se desprecian las mediciones incorrectas de la medición total filtrando los resultados cuando es posible.

La validación hecha mediante una evaluación con hosts controlados obtiene resultados satisfactorios y errores mínimos de alrededor del 3% en los peores casos de tráfico cruzado para un enlace de bajada de 4,14 Mbps y de 0,62 Mbps para el enlace de subida.

La validación sobre hosts de internet obtuvo buenos resultados para dos tercios de los 1224 hosts estudiados con al menos una medición correcta de las 10 realizadas. El 30% falló debido a congestión en el uplink.

Finalmente se tiene que DSLprobe es una herramienta no cooperativa (no requiere de participación activa de los hosts a medir) que utiliza dos órdenes de magnitud menos tráfico que herramientas similares como MPI para estimar satisfactoriamente las capacidades de los enlaces ADSL de subida y de bajada para hosts en redes IP.

3 DESARROLLO

3.1 Definición de los modelos a utilizar

A continuación se presentan los modelos a utilizar en la simulación y sus diferencias con los modelos reales.

3.1.1 Modelo de red

El modelo de la realidad que se aprecia en la figura 3.1 es complejo y contiene diferentes tipos de servicios, diferentes tipos de nodos de acceso, una MEN y acceso a Internet además de gestión. Es por esto que se realizan algunas simplificaciones que no deberían interferir en los resultados desde el punto de vista de los nodos de acceso, obteniendo el modelo a simular que se presenta en la figura 3.2.

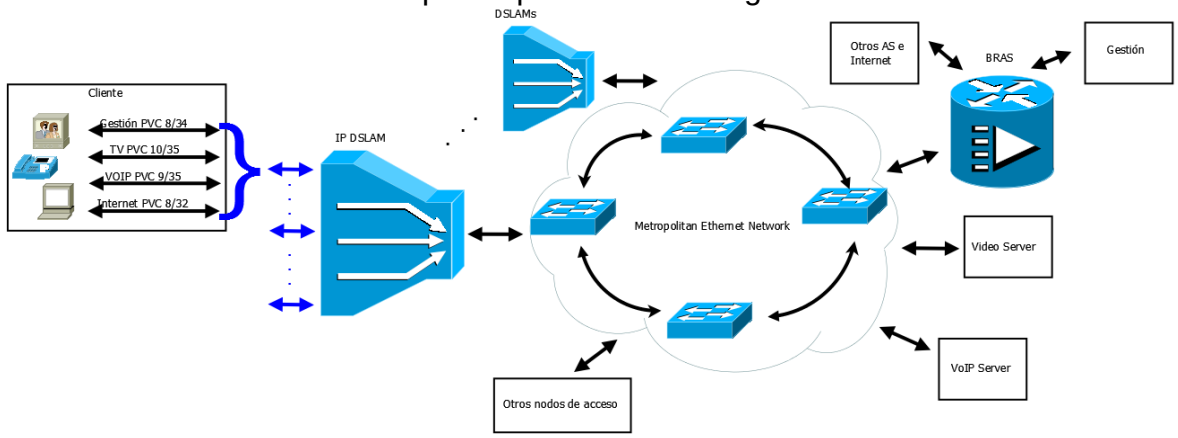


Figura 3.1: Modelo de red real.

Simplificaciones a la red compleja	Descripción
1 AS	Se modela la red de un solo AS.
Solo nodos de acceso DSLAM	Se pueden tener más de un nodo pero, por el momento, solo pueden ser del tipo DSLAM-IP.
Reducción de MEN	La MEN es reemplazada por un router.
Reemplazo de Internet por servidores	Se reemplaza todo el resto de la red por servidores de diferentes aplicaciones.

Tabla 3.1: Simplificaciones de la red.

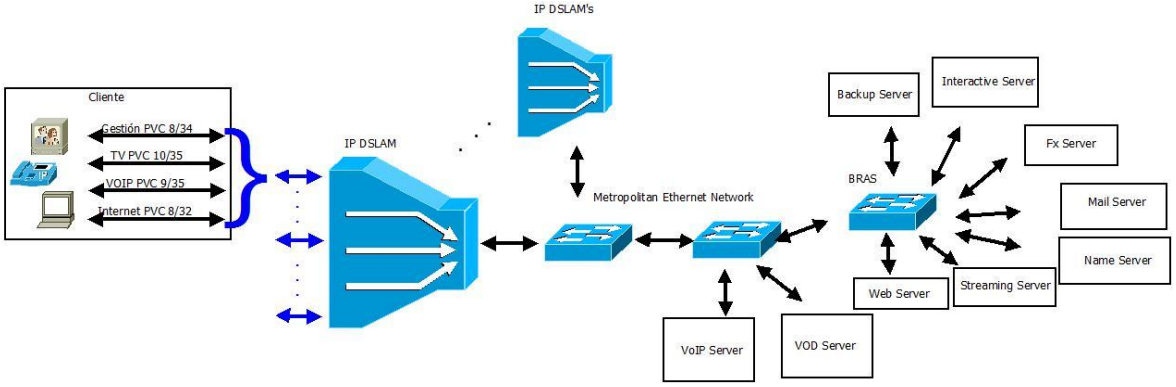


Figura 3.2: Modelo simplificado de red a simular.

Una vez que se tiene como es la red a nivel macro hay que definir el modelo a simular a nivel más específico para los nodos de acceso, servidores y clientes.

3.1.2 Modelos de nodo de acceso

El modelo para el DSLAM típico corresponde a varias tarjetas de línea que cierran el ciclo con el abonado a través de sus puertos y a una tarjeta en el uplink para el caso en estudio.

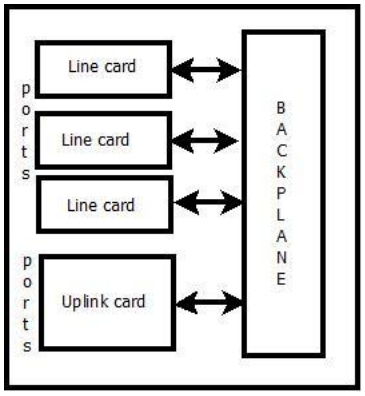
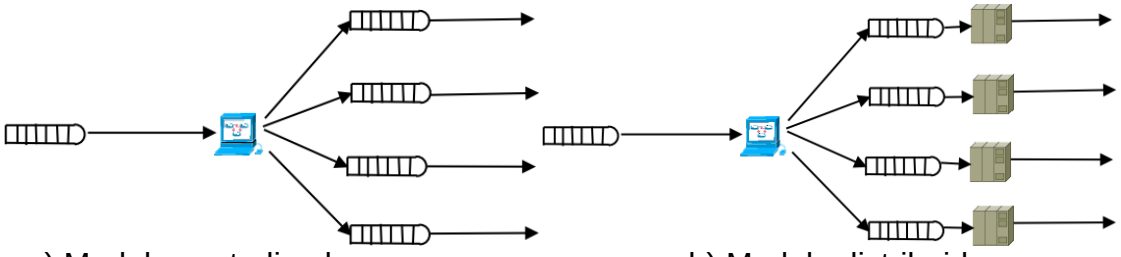


Figura 3.3: Modelo de DSLAM de alto nivel.

El modelo en general desde el punto de vista de teoría de colas se compone de una cola de entrada (ya que todos los paquetes llegan al mismo procesador), de un procesador y de varias colas de salida. Las colas de salida pueden no tener inteligencia o pueden sí tenerla.



a) Modelo centralizado b) Modelo distribuido

Figura 3.4: tipos de modelos de nodos

El modelo elegido el es el de procesamiento distribuido debido a que de esta forma se puede tener servicios diferenciados en cada uno de los puertos. Por otro lado, como se muestra en la figura 3.5, se tiene el servidor de la aplicación IPTV inserta en el módulo del nodo de acceso para así simular la funcionalidad de saturación por conexiones simultáneas de este tipo de servicio que se den dentro de los clientes asociados al nodo. Es importante tomar en cuenta que en la simulación no se tiene un servidor que envíe todos los canales a cada nodo de acceso y solo se simula a modo de poder notar la saturación del procesador por la cantidad de conexiones simultáneas.

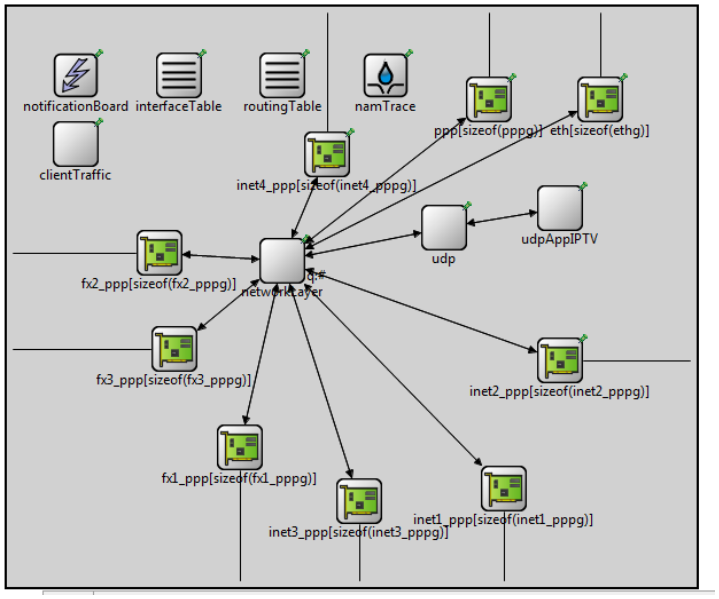


Figura 3.5: Módulo nodo de acceso.

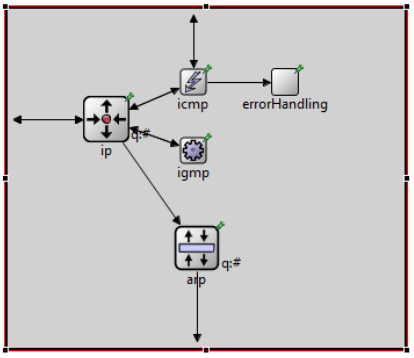
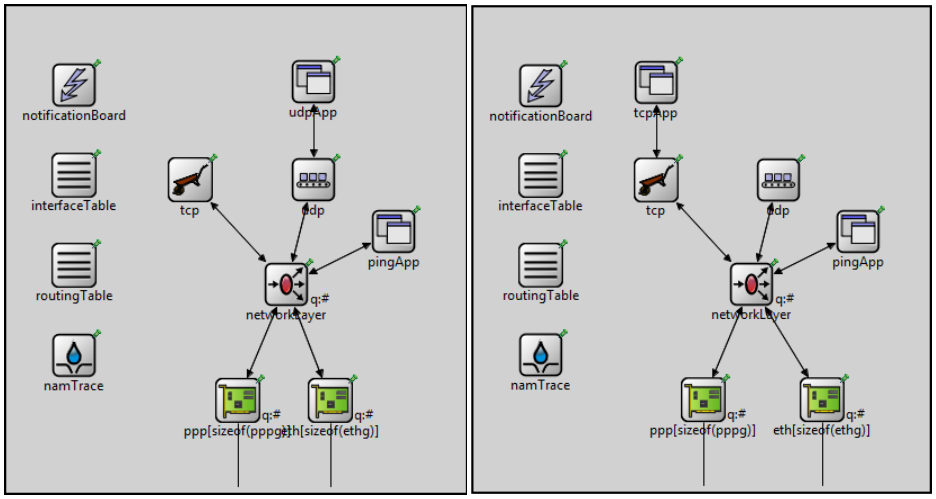


Figura 3.6: Módulo de nivel de red

Todos los paquetes pasan por el módulo IP y se procesan agregando un tiempo de procesamiento.

3.1.3 Modelo de servidores

En las simulaciones existen dos tipos de servidores, los servidores UDP y los servidores TCP donde cada uno responde a las peticiones asociadas a su tipo de servicio y se diferencian estructuralmente en la capa de aplicación donde uno tiene presente un módulo UDP y el otro TCP según corresponda.



a) Modelo de servidor UDP b) Modelo de servidor TCP

Figura 3.7: Modelos de servidores

3.1.4 Modelo de clientes

Los clientes son similares a los servidores salvo que cada uno de estos puede enviar tráfico de ambos tipos TCP y UDP, además estos no responden a los paquetes recibidos y simplemente los descartan. Contiene el módulo inetUser el cual se encarga de encender aplicaciones TCP o UDP iniciando así flujos de tráfico.

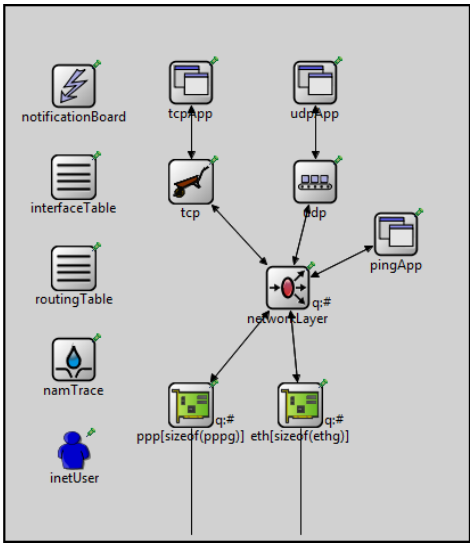


Figura 3.8: Modelo de cliente.

3.2 Implementación de red básica

A continuación se presenta la implementación final del modelo a simular como se ejemplifica en la figura 3.9 y además se explica cómo se integró cada parámetro de red al Simulador OMNeT++ y la interfaz gráfica Genersim como también los cambios a ReaSE.

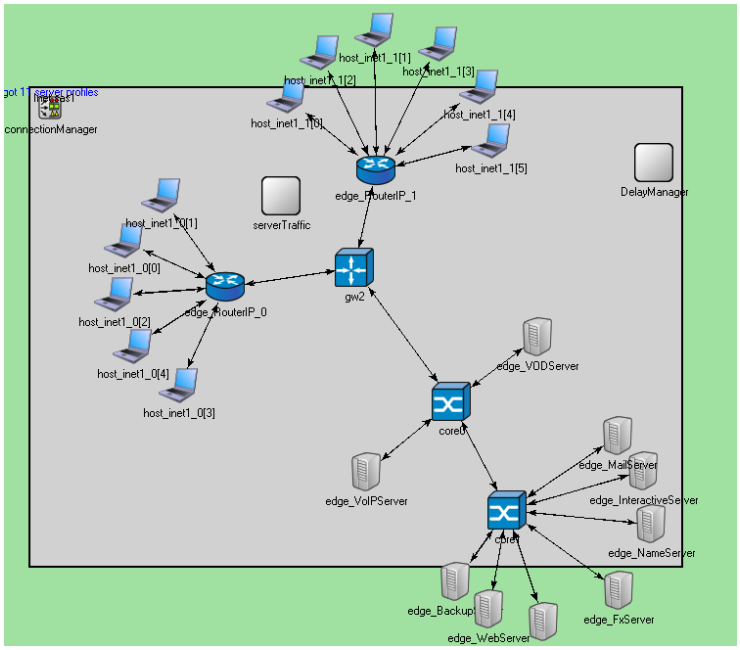


Figura 3.9: red implementada

Para poder crear ambientes de simulación en OMNeT++ se requieren dos tipos de archivo: omnetpp.ini y red.ned. Estos archivos no son modificables de forma trivial y es necesario conocimiento del lenguaje ".ned" y de cómo funciona OMNeT++ para crear simulaciones. Por este motivo se crea Genersim, la cual es una interfaz gráfica que permite la creación de simulaciones de forma rápida y flexible.

3.2.1 Parámetros ajustables de red generales

Número de nodos de acceso: La red simulada puede contar con más de un nodo de acceso DSLAM y esto se ha logrado habilitando la opción en la interfaz gráfica Genersim la cual modifica el archivo red.ned utilizando un arreglo de objetos nodo en java de los cuales se traspasa todos los parámetros de los nodos a el archivo red.ned incluyendo su cantidad.

Ancho de banda puerta de enlace a núcleo, de núcleo a puerta de enlace y de núcleo a núcleo: La interfaz gráfica Genersim modifica el archivo red.ned agregando el ancho de banda específico para cada canal. Estos no dependen de la cantidad de nodos pues están predefinidos en la red.

Paleta de red-parámetros generales	Valor mínimo	Valor máximo	Unidad	Comentario
N° de nodos de acceso	1	100	-	-
Puerta de enlace -> Núcleo	0	sin límite práctico	Mbps	-
Núcleo -> Puerta de enlace	0	sin límite práctico	Mbps	-
Núcleo -> Núcleo	0	sin límite práctico	Mbps	-

Tabla 3.2: Restricciones operacionales para parámetros generales.

3.2.2 Parámetros por cada nodo

Todos los parámetros asociados a un nodo son guardados por genersim en un objeto java denominado Node y con éste se guarda la información de los nodos y se modifican los archivos red.ned y omnetpp.ini

Nombre: Este parámetro no es agregado a ningún archivo salvo los de configuración de la interfaz gráfica al ser guardado y sirve como referencia cuando se tienen redes de más de un nodo de acceso.

Retardo del procesador: Es el tiempo que demora el procesador en procesar un paquete y se agrega a la simulación por medio de la modificación del archivo omnetpp.ini dándole valor al parámetro *procDelay* del módulo IP del nodo correspondiente.

Capacidad de cola uplink out: Es la cantidad de paquetes máxima que se puede encolar en el uplink de salida (en el flujo de tráfico hacia la puerta de enlace). Modifica el archivo omnetpp.ini definiendo el parámetro *frameCapacity* de la cola de la interfaz ppp en la subida.

Tipo de cola uplink out: Permite seleccionar si la cola de subida trabaja con servicios diferenciados o no, para esto define en el archivo omnetpp.ini el parámetro *queueType* de la interfaz ppp de subida.

Capacidad de cola de entrada: Determina la capacidad de la cola de entrada al procesador del DSLAM (todos los paquetes de subida y de bajada). Se agrega a la simulación por medio de la modificación de omnetpp.ini dándole valor al parámetro *limitBytes* del módulo IP del nodo correspondiente.

Ancho de banda desde acceso a puerta de enlace y desde puerta de enlace a acceso: Configura el ancho de banda del canal cuyo flujo va desde el nodo de acceso a la puerta de enlace y del canal cuyo flujo va desde la puerta de enlace a acceso. Se agrega a la simulación por medio de la modificación del archivo red.ned definiendo los anchos de banda para cada nodo.

Compatibilidad IPTV: Define la compatibilidad IPTV de un nodo dado, permitiendo que los clientes conectados al nodo puedan o no conectarse a el nodo de acceso más cercano como su servidor IPTV. Esto se logra modificando información del archivo

red.ned que luego será utilizada por el módulo *inetUser* en cada cliente para saber que sí o que no puede seleccionar el tipo de tráfico IPTV.

Conexiones IPTV simultáneas: Es necesario para los nodos donde sí hay compatibilidad IPTV. De esta forma se puede simular los problemas que se generan por cambios de canal simultáneos que saturen el DSLAM. Se logra al modificar el archivo omnetpp.ini definiendo el parámetro *noThreads* de la aplicación udpApp de cada DSLAM.

Pérdida de paquetes: Configura el porcentaje de pérdida de los paquetes que atraviesan el nodo en particular.

Parámetros del nodo que se está editando	Valor mínimo	Valor máximo	Unidad	Comentario
Retardo proc.	0	2.000.000.000.000	[us]	-
Cap. Cola uplink out	0	2.147.483.647	[Frames]	-
Cap. Cola entrada	0	sin límite práctico	[MB]	-
Acceso -> Puerta de enlace	0	sin límite práctico	[Mbps]	-
Puerta de enlace -> Acceso	0	sin límite práctico	[Mbps]	-
Conexiones IPTV Simultaneas	0	50.000	conexiones	Conexiones por nodo
Pérdida de paquetes	0	100	[%]	-

Tabla 3.3: Restricciones operacionales para parámetros del nodo.

3.2.3 Parámetros ajustables de red por nodo y para cada grupo de clientes.

A cada nodo se le pueden asociar hasta siete grupos diferentes de clientes: cuatro de tipo Inet y tres de tipo distribuido o F(x).

Factor tiempo: Multiplica los tiempos entre los envíos entre paquetes configurados de forma general y se logra hacer esto al modificar el archivo red.ned y definiendo el parámetro *timeFactor*.

Factor data: Multiplica los tamaños de las peticiones y de las respuestas aumentando o disminuyendo el tráfico con respecto a los configurados de forma general si éste es mayor o menor al valor uno. Esto se logra al modificar el archivo red.ned definiendo el parámetro *lengthFactor*.

Ancho de banda desde cliente al nodo de acceso y desde nodo de acceso a cliente: Configuran los anchos de banda que van desde el cliente al nodo de acceso y los que van desde el nodo de acceso al cliente. Esto se logra para cada grupo mediante la modificación de del archivo red.ned definiendo los parámetros *datarate* de los canales para cada uno de los grupos.

Capacidad de colas nodo a cliente: Configura la capacidad de las colas en las tarjetas del nodo de acceso correspondientes al grupo de clientes en particular. Se logra aplicar modificando el archivo omnetpp.ini definiendo el parámetro *capacity* del grupo y del nodo en particular.

Tipo de colas: Configura si las colas nodo a cliente tienen discriminación de servicios o tipos de tráfico. Se logra aplicar modificando el archivo omnetpp.ini definiendo el parámetro *queueType* del grupo y del nodo en particular.

Número de aplicaciones: Simultáneamente pueden convivir diferentes aplicaciones para un mismo cliente con, por ejemplo, diferentes prioridades y características de tráfico. Este parámetro configura la cantidad de aplicaciones que correrán cada uno de los clientes correspondientes al grupo. Se logra modificando el archivo omnetpp.ini definiendo el parámetro nApps para cada cliente de cada grupo de cada nodo, el cual será leído por el *inetUser* de cada cliente.

Grupos Inet , parámetros a y b: Corresponden a los parámetros de una función uniforme (starTime = uniform(a,b)) la cual define el tiempo de partida para cada flujo de tráfico de cada aplicación en los clientes tipo inet. Se logra modificando el archivo omnetpp.ini dándole valor al parámetro *startTime* con la distribución *uniform(a,b).*
Nombre del grupo: Este parámetro no es agregado a ningún archivo salvo los de configuración de la interfaz gráfica al ser guardado y sirve solo como referencia. La interfaz gráfica genersim utiliza el parámetro nombre en los objetos *ParamsInet* y *ParamsFx*.

Parámetros Host:	Valor mínimo	Valor máximo	Unidad	Comentario
n° Host	0	10.000	clientes	Probado con 10.000 clientes totales en 1 nodo.
Factor tiempo	0	100.000	veces	Junto con **Time to respond** y **Time between request** no deben superar los 2.000.000 [s]
Factor data	0	100.000	veces	-
Host -> Acceso	0	sin límite práctico	[Mbps]	-
Acceso -> Host	0	sin límite práctico	[Mbps]	-
Cap.	0	2.147.483.647	[frames]	-
nApps		10	-	Si un cliente se comunica con un mismo servidor con más de una aplicación simultáneamente puede producir más tráfico de respuesta de lo esperado, por esto el límite de 10.
a[s] y b [s]	0	2.000.000	[s]	-

Tabla 3.4: Restricciones operacionales para los parámetros de clientes.

3.2.4 Parámetros ajustables de tráfico

Todos los parámetros ajustables de tráfic que a continuación se presentan son guardados en el archivo *trafficProfile.parameters* con los valores correspondientes para luego ser accedidos desde la simulación en OMNeT++ por el módulo *trafficProfileManager*. Además estos valores son para cada una de las aplicaciones.

Tamaño de la petición: Para cada aplicación se configura el tamaño de la petición.

Tamaño de la respuesta: Tamaño de la respuesta por parte del servidor para el flujo de tráfico entre un cliente (host) y un servidor.

Cantidad de respuestas por petición: Cantidad de respuestas desde el servidor al cliente por cada petición entrante en el flujo cliente (host) y el servidor.

Cantidad de peticiones por flujo: Cantidad de peticiones por cada flujo entre cliente y servidor para una aplicación seleccionada.

Proporción de selección: Configura la proporción de selección de cada aplicación, a mayor valor mayor probabilidad de que la aplicación correspondiente sea utilizada por el simulador.

Tiempo entre peticiones: Corresponde al tiempo de entre envíos de peticiones por parte de los clientes.

Tiempo entre respuestas: Corresponde al tiempo entre el envío de las respuestas por parte del servidor.

Tiempo entre flujos: Tiempo que separa el final de una sesión de flujo por el lado del cliente para dar paso a una nueva selección de aplicación y al inicio de un nuevo a nueva sesión de flujo.

Prioridad DSCP: Corresponde a la prioridad que la aplicación tiene sobre las otras y se utiliza cuando en la simulación hay colas del tipo DropTailQoSQueue la cual le da preferencia a los paquetes según su prioridad.Usa el segundo byte de la cabecera de los paquetes IP.

Tráfico Inet:	Valor mínimo	Valor máximo	Unidad	Comentario
Reply length	0	200.000.000	[Bytes]	probada con un **Replies per request** de 1 y **Factor data** de 1
Replies per request	0	20.000.000	-	Probada con un **Time to respond** de 100 [ms] y **Factor tiempo** de 1
Selection ratio	0	sin límite práctico	-	
Request length	0	200.000.000	[Bytes]	probada con un **Factor data** de 1
Request per flow	0	20.000.000	-	Probada con un **Time between request** de 100 [ms], **Factor tiempo** de 1 y **Replies per request** de 1
Time between requests	0	2.000.000.000	[ms]	
Time to respond	0	2.000.000.000	[ms]	
Time Between flows	0	2.000.000	[s]	
Packet's priority DSCP	1	8	° prioridad	

Tabla 3.5: Restricciones operacionales de parámetros de tráfico inet.

3.2.5 Parámetros ajustables de tráfico distribuido

Existen varios tipos de distribuciones probabilísticas que se pueden seleccionar para cada grupo de clientes F(x) en cada nodo. Cada una de las distribuciones tiene parámetros asociados. Y se utilizan eligiendo un tiempo de partida distribuido para cada aplicación de cada cliente de cada grupo F(x) de cada nodo. Donde al tener varias aplicaciones y varios clientes se va formando una curva de distribución correspondiente.

Se logra modificando el archivo omnetpp.ini definiendo el parámetro *startTime* para cada cliente de cada grupo de cada nodo.

Paleta de Trafico F(x)	Valor mínimo	Valor máximo	Unidad	Comentario
RNG	0	100	-	

Tabla 3.6: Restricciones operacionales de parámetros de tráfico distribuido.

3.2.6 Mediciones posibles

Existen ocho opciones para elegir en las mediciones, donde cada una de las opciones agrupa una cantidad variable de mediciones dependiendo de la cantidad de nodos presentes en la simulación.

- Tráfico por nodo de subida y bajada.
- Tráfico por nodo de subida y bajada diferenciado por tipo de tráfico.
- Tráfico total por servidores de subida y bajada.
- Tráfico por servidores de subida y bajada diferenciado por tipo de servidor.
- Tráfico por clientes de subida y de bajada.
- Tráfico por clientes de subida y de bajada diferenciado por tipo de tráfico.
- Cantidad de clientes que superen un retardo límite diferenciado por tipo de tráfico.
- Retardos máximos totales.

Se logra tener estas opciones de medición al tener un valor booleano para cada opción y transferido a la simulación a través del archivo omnetpp.ini desde donde se lee cada opción en el módulo correspondiente de la simulación, el cual corresponde usualmente al módulo IP de los elementos.

3.3 Interfaces

A continuación se presenta la interacción entre las interfaces en presencia de un flujo de tráfico cliente-servidor.

En un principio los servidores de la red se registran al módulo ConnectionManager. Simultáneamente el módulo TrafficProfileManager lee los parámetros de tráfico para cada aplicación desde el archivo trafficProfile.parameters.

Gracias a genersim uno puede separar los clientes por grupos y por nodos, estos clientes tienen diferentes cantidades de aplicaciones que se utilizarán en la simulación. La cantidad de aplicaciones nApps genera que el módulo de los clientes inetUser elija, para cada una de estas aplicaciones, un tiempo de inicio entonces cada aplicación dentro de un cliente tiene su tiempo de inicio dado y lo que hace al iniciar es pedir un servidor al módulo ConnectionManager donde éste se comunica con el módulo TrafficProfileManager para elegir un tipo de aplicación aleatoriamente y también elige un servidor aleatoriamente.

Se tiene que, como se muestra en la figura 3.8, el módulo cliente se compone, a nivel de usuario, de un módulo inetUser el cuál activa un hilo de ejecución de los módulos de nivel de aplicación TCP y UDP según la aplicación que se elija y este hilo abre un socket con la dirección y puerto de destino. Por medio de este socket se envían los paquetes de la petición hacia el servidor de destino. Los mensajes a nivel de aplicación que se crean en los clientes contienen los parámetros que definen el tráfico de respuesta para que así el servidor responda de acuerdo a estos. Los parámetros son los siguientes:

- Tamaño de la respuesta.
- Cantidad de respuestas por petición.
- Tiempo entre respuestas.
- Valor booleano si es el último.
- Número de paquetes.
- DSCP.
- Identificador de la aplicación.

Los paquetes pasan de la capa de aplicación a la capa de transporte por medio de un socket y luego los segmentos TCP o los paquetes UDP pasan a la capa de red IP donde se determina si el paquete va para una dirección IP externa o a una local preguntando a la tabla de ruteo. Si va para una dirección externa el paquete se envía al módulo ARP que a su vez asocia el paquete a la interfaz PPP de conexión correspondiente (En el presente trabajo se eligen siempre interfaces PPP y no Ethernet pero esto podría eventualmente cambiarse) la cual encola y encapsula los datagramas IP en PPPframes y la cola espera la petición desde el módulo PPP para enviar un paquete a través del canal. Si el paquete va a la dirección local, éste se envía al módulo de capa transporte correspondiente el que puede ser TCP o UDP.

Luego de pasar por el canal los PPPframes llegan a la interfaz PPP del nodo de acceso donde son desencapsulados y dirigidas directamente al módulo del nivel de red IP donde éste encola en una cola FIFO todos los datagramas IP que llegan a él (incluso encola los paquetes que vienen de un nivel superior), ver Figura 3.4, además el tamaño de la cola de los nodos de acceso se puede configurar. Luego se pregunta si el IP de destino del paquete es la IP local para enviarlo al nivel superior, lo que sucede en el caso de que los paquetes sean peticiones de IPTV. Si el IP de destino no es la IP local se envía al módulo ARP que asocia el IP de destino con la interfaz PPP correspondiente. Esta interfaz PPP puede ser inteligente y tener una cola que diferencia el tipo de servicio teniendo prioridad según el valor DSCP con que llegue el mensaje. También nuevamente se encapsula el mensaje y se espera a que el canal pida un mensaje para sacarlo de la cola.

Ahora el camino del tráfico llega al gateway donde se tiene un funcionamiento idéntico al del nodo de acceso pero sin contar con la aplicación IPTV incorporada, sus

colas no diferencian el tipo de servicio y no se puede configurar el tamaño de la cola FIFO del módulo IP el cual es por defecto sin límite, ni de el de sus colas de salida.

Los routers del core son idénticos a los gateway salvo que los primeros están conectados directamente a los servidores de aplicación y gateway.

Los servidores son los módulos que producen las respuestas a las peticiones por parte de los clientes. Pueden atender a múltiples clientes a la vez, esto ha sido configurado a un máximo de alrededor de 50.000. Se tiene que cuando llega un mensaje a la interfaz PPP ésta la desencapsula y la envía al nivel de red donde el módulo IP encola el datagrama y procesa si es IP local preguntando a la tabla de ruteo. De ser éste el caso envía paquete a la interfaz de nivel superior TCP o UDP según corresponda al servidor y al tipo de tráfico. De no ser este el caso se envía el paquete al módulo ARP que envía los paquetes a la interfaz PPP y a través de ésta al resto de la red. Una vez que el mensaje pasa de la capa de transporte a la de aplicación se tiene que obtener los parámetros de tráfico desde el mensaje para configurar las respuestas a la petición y posteriormente se responde según los valores de estos parámetros.
El comportamiento del tráfico de respuesta es análogo al tráfico de petición además hay que agregar que las mediciones se realizan modificando los módulos IP de la librería Inet y agrupando el tráfico que pasa por los clientes, por los servidores y por cada nodo de acceso.

4 RESULTADOS

A continuación se presentan los resultados de las proyecciones de tráfico obtenidas mediante la simulación y además se presentan los resultados esperados, como lo son resultados se tráfico de bajada, tráfico de subida y las variables resultantes relevantes a cada prueba.

Para realizar proyecciones de tráfico se creó Genersim, la cual es la interfaz gráfica tratada en la sección de Desarrollo del presente trabajo. Por esto se hace necesario primero definir todos los parámetros del ambiente en que se quiere simular. Los parámetros utilizados en este capítulo fueron ajustados de forma de que puedan retornar, mediante simulación, valores adecuados a lo que se quiera lograr.

4.1 Simulación basada en datos de un DSLAM-IP

Inicialmente surge la necesidad de poder trabajar con datos provenientes de algún sistema real para así demostrar las capacidades de la herramienta creada y al mismo tiempo poder desarrollar análisis más profundos al tener múltiples formas de aproximarse a los problemas. En este caso se quiere poder recrear el día con mayor volumen de tráfico del DSLAM Santa Lucia 6 para los meses de marzo y abril de 2011. El día seleccionado es el 3 de marzo y su representación gráfica se observa en la figura 4.1.

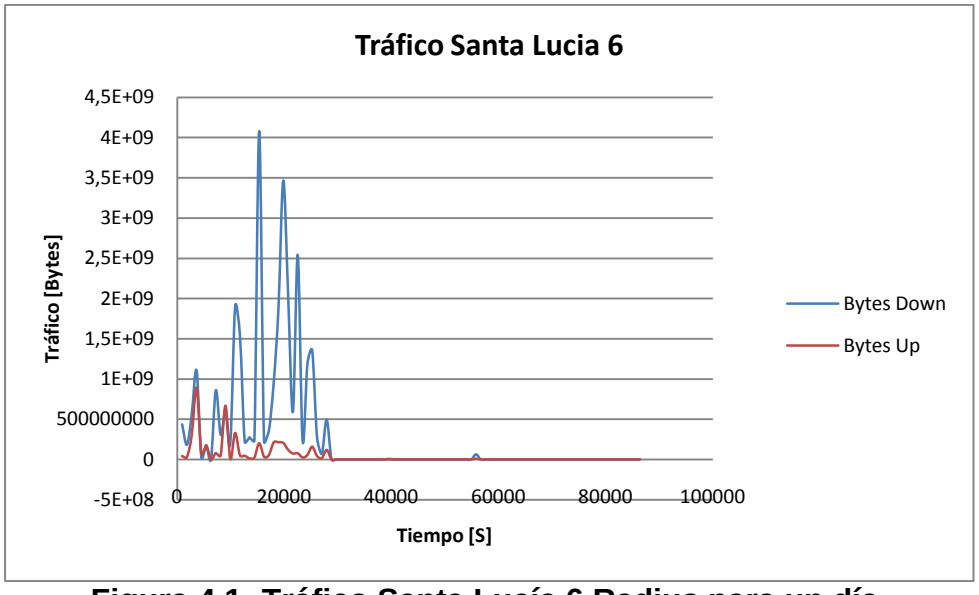


Figura 4.1: Tráfico Santa Lucía 6 Radius para un día.

Como se aprecia en el gráfico 4.1 se tiene que tanto el tráfico de subida (marrón) como el de bajada (azul) se concentran en el primer tercio del día pero esto es solo una ilusión pues lo que sucede es que los datos con los que se trabaja son extraídos de la base de datos Radius de Movistar y ésta graba los datos de tráfico al momento de desconectarse cada cliente y sumado al hecho que se obliga a los clientes a

desconectarse dentro de las primeras 8 horas del día, se genera este fenómeno. Sin embargo, para realizar algunas pruebas, se simulará teniendo tráfico en el primer tercio del día pero siempre tomando en cuenta que el volumen diario es la variable más importante que se puede extraer de los datos de la base de datos Radius.

Volumen de bajada	28.432.013.099 Bytes
Volumen de subida	4.382.378.378 Bytes
Razón bajada/subida	6,487804258

Tabla 4.1: Información del volumen de tráfico en DSLAM Santa Lucía para día de mayor tráfico.

Configurando los parámetros de Genersim, se logra obtener una simulación que genera los resultados que se muestran en la figura 4.2 la cual contiene el gráfico del tráfico de bajada medido en el lado de los servidores (esto es para evitar que afecte la pérdida de paquetes de la red, por esto se mide el tráfico a penas es creado) y contiene el tráfico de subida medido en el lado de los clientes (por la misma razón que el tráfico de bajada). Por otra parte, el día con mayor tráfico del DSLAM Santa Lucía 6, fue expandido para generar una simulación de cinco días con igual carga de tráfico en cada día.

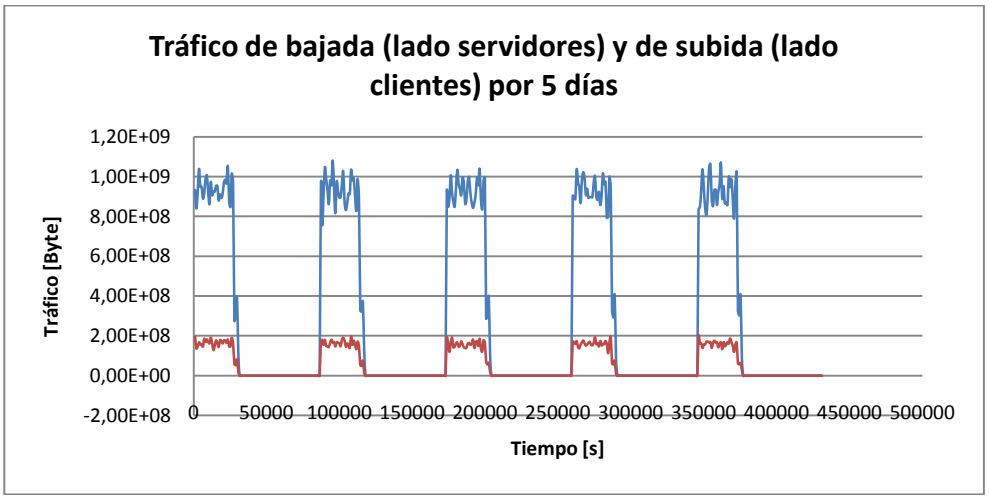


Figura 4.2: Tráfico simulado de bajada (azul) y subida (marrón).

Se observa a simple vista cinco períodos de tráfico, esto se hizo intencionalmente para simular la característica de tráfico que se observa en el tráfico de la base de datos Radius (Figura 4.1) la cual se expresa durante las primeras ocho horas de cada día.

Volumen de bajada medido en servidores por día	29.225.628.228
Volumen de subida medido en clientes por día	5.145.415.990
Razón bajada /subida	5,6799239721

Tabla 4.2: Información del volumen de tráfico en DSLAM simulado.

Como se observa en las tablas 4.1 y 4.2 existe una gran similitud en el tráfico de bajada logrando el objetivo inicial de poder simular datos que tengan un origen en la realidad, además se tiene que el tráfico de subida resultante es del orden del tráfico de subida original y se logra una razón de bajada/subida similar.

4.2 Simulación de tráfico con y sin servicios IPTV y VoIP.

Esta simulación consta de dos partes importantes, la primera es la que supone un cambio en la pérdida de paquetes configurada para el DSLAM, y la segunda es la de diferenciación de los casos cuando los clientes eligen aplicaciones como IPTV y VoIP o cuando no las eligen.

Para acortar el nombre de cada escenario se utilizará la nomenclatura definida en la tabla 4.3.

E1: pérdida de paquetes de 1% con servicios IPTV y VoIP activados.
E2: pérdida de paquetes de 1% sin servicios IPTV y VoIP activados.
E2: pérdida de paquetes de 2% con servicios IPTV y VoIP activados.
E2: pérdida de paquetes de 2% sin servicios IPTV y VoIP activados.

Tabla 4.3: Distintos escenarios considerados en las simulaciones

Para realizar un análisis junto con la visualización de los resultados gráficamente, los resultados numéricos finales se muestran primero en la tabla 4.4 donde se distingue entre cada uno de los escenarios.

Escenarios	%Pérdidas up	%Pérdidas down	Tráfico up [Bytes]	Tráfico down [Bytes]	Down/Up
E1	1,00453	1,00030	1.989.964.590	18.666.734.082	9,3804353
E2	1,00391	1,00697	521.108.744	13.973.693.000	26,8153109
E3	1,98011	1,99969	1.990.130.840	18.584.744.530	9,33845361
E4	1,98318	2,00001	521.292.264	13.833.742.700	26,5374026

Tabla 4.4: Resultados finales obtenidos de los datos simulados.

4.2.1 Escenario de pérdida de paquetes de 1% con servicios IPTV y VoIP activados.

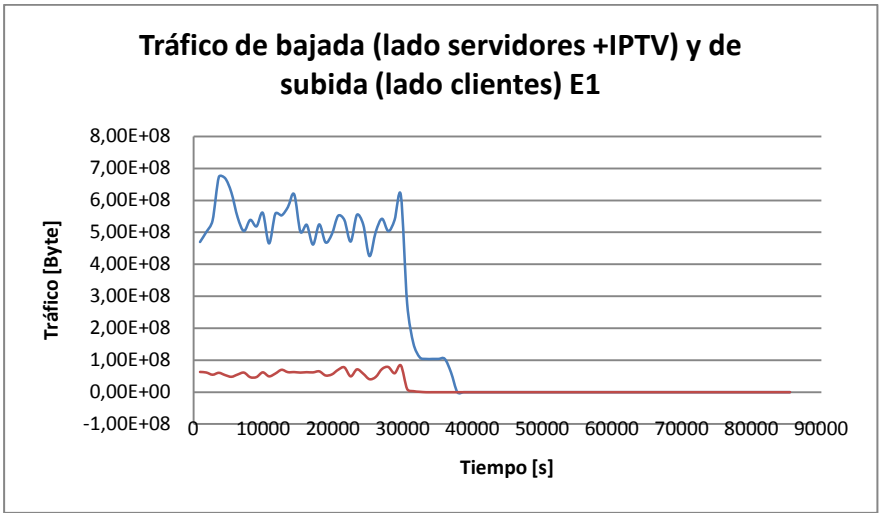


Figura 4.3: Tráfico de bajada (azul) y de subida (marrón) para escenario E1.

En la figura 4.3 y en la tabla 4.4 se observa que el tráfico de bajada supera al tráfico de subida en 9,4 veces, esto sucede debido a la configuración de los perfiles de tráfico utilizados incluyendo los perfiles VoIP e IPTV. Además, como se puede observar en la tabla 4.4, se tiene un tráfico de bajada mayor a todos los demás casos y esto se debe a que, por un lado, al comparar los escenarios E1 con E2 se tiene que la utilización de perfiles de tráfico (servicios) VoIP e IPTV en E1 genera una diferencia de caso 5 GB con respecto a E2 por lo que los resultados tienen coherencia con lo esperado ya que estos servicios utilizan mayor carga en el tráfico up y en el tráfico down comparativamente a los demás perfiles de tráfico por defecto. Por otro lado, al comparar el escenario E1 con E3 que tiene una misma configuración de parámetros para los perfiles de tráfico (ambos con VoIP e IPTV) se tiene que existe una leve diferencia a favor en el tráfico down para E1 pues este tráfico de bajada se mide en por el lado servidores, lugar donde llegan las peticiones para generar un flujo de paquetes de bajada, y estas peticiones en su camino de subida pasan por el nodo de acceso el cual está configurado con una pérdida de 1% en E1 y de 2% en E3 lo que explica la diferencia a favor en el tráfico de bajada de E1.

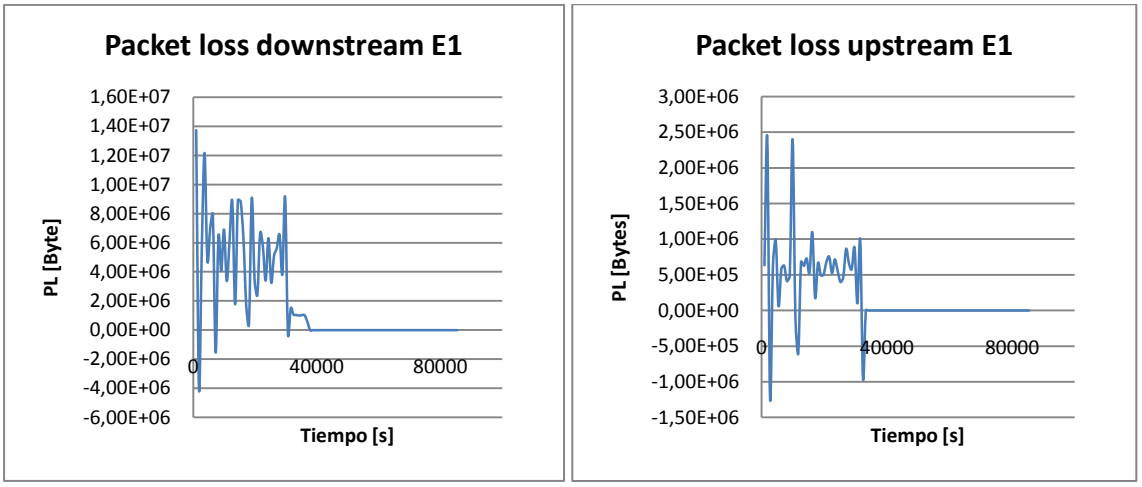


Figura 4.4: Pérdidas de paquetes en cada intervalo de medición, escenario E1.

En la figura 4.4 se tienen las pérdidas de paquetes medidas como la diferencia entre los tráficos de la subida en el lado cliente y servidor (upstream) o entre los tráficos de bajada en el lado servidor y cliente (downstream). La definición de estas observaciones de packet loss no es precisa pues medir en un intervalo el tráfico que llega en dos partes instantáneamente puede provocar errores si el intervalo de tiempo es muy pequeño, si hay mucho retardo en la red y si no hay mucha pérdida de paquetes real en el intervalo. Tomando esto en consideración se esperaría que en la mayor parte de los instantes los resultados de pérdida de paquetes según la definición sean positivos, lo cual es cierto y se corrobora con las observaciones pero aún existen observaciones de diferencias de tráfico negativa las cuales se explican por las fuentes de error mencionadas. Aún así se tiene que la pérdida de paquetes total, la suma de todos los intervalos, medida de esta manera es muy precisa por lo que el análisis a continuación es válido. Se tiene una pérdida de paquetes total de 1,0003% en la bajada y 1,0045% en la subida, es decir, valores muy cercanos al 1% configurado en el nodo de acceso. Las diferencias en las pérdidas de paquetes son mínimas y reflejan el buen funcionamiento del algoritmo que descarta paquetes en el nodo de acceso, además, dada una configuración con colas grandes en toda la red no se esperan pérdidas de paquetes en otros sectores.

4.2.2 Escenario de pérdida de paquetes de 1% sin servicios IPTV y VoIP activados.

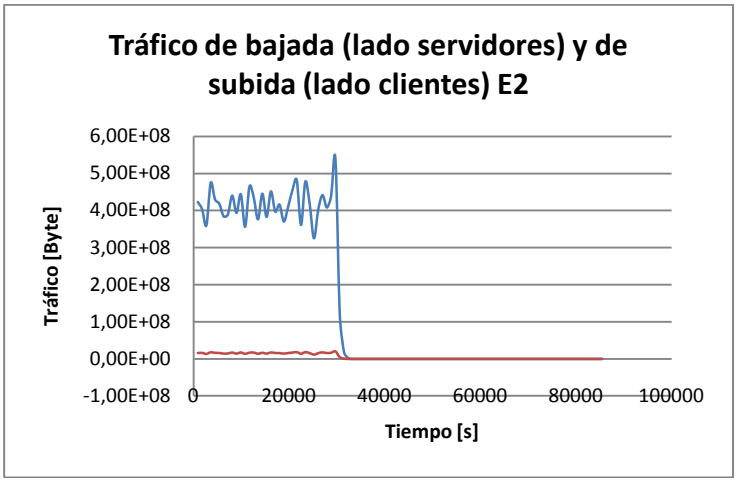


Figura 4.5: Tráfico de bajada (azul) y de subida (marrón) para escenario E2.

Como se observa en la figura 4.5 existe una gran diferencia entre el tráfico de bajada y el tráfico de subida dado por una razón de 26,8 veces lo cual es muy diferente al escenario E1 que tiene una razón de 9,3 veces. Esta diferencia se explica principalmente por los distintos servicios o aplicaciones que se prestan pues mientras los perfiles de tráfico configurados como base e IPTV tienen una razón alta, el servicio de VoIP tiene una razón de 1 lo que genera que el escenario E1, que conlleva servicios de VoIP, produzca una razón bajada versus subida mucho menor que E2, por lo tanto, la diferencia entre subida y bajada en E2 se explica a que no corre aplicaciones que tengan una razón siquiera similar entre la bajada y la subida. Por otro lado se tiene que las diferencias que existen en los tráficos de bajada entre E1 y E2 se explican por el servicio IPTV que corre en la red y agrega grandes cantidades de tráfico de bajada con respecto a otras aplicaciones de la configuración base.

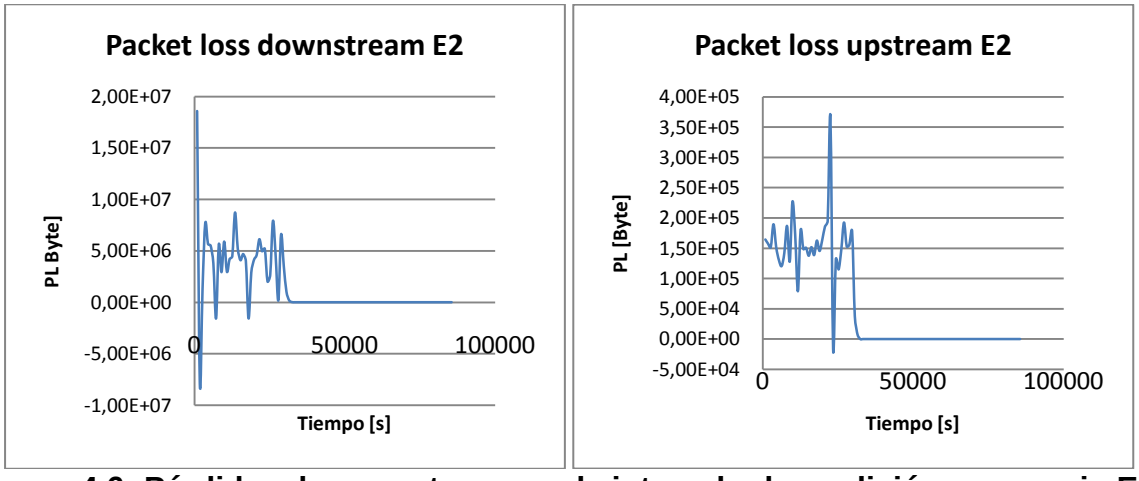


Figura 4.6: Pérdidas de paquetes en cada intervalo de medición, escenario E2.

Tal como se explica para el caso E1 en pérdidas de paquetes (ver figura 4.4) los valores de estas mediciones tienen un pequeño margen de error pero la suma total es lo más relevante. En la figura 4.6 se tiene que los valores de pérdida de paquetes en el escenario E2 son mayoritariamente positivos, es más su suma también lo es. Como en E2 se tiene configurado el nodo de acceso a tener una pérdida de paquetes total en torno al 1% y dado que los resultados de E1 fueron extremadamente similares al 1%, es decir no hubo mayor contribución de pérdida de paquetes por el resto de la red, se tiene que como para el caso E2 se tiene la misma red y niveles de tráfico menores, se puede esperar que tampoco haya contribución de la red y la pérdida de paquetes total esté dada solo por las pérdidas en el nodo de acceso y por lo tanto tener una pérdida de paquetes en torno al 1%. Las pérdidas de paquetes obtenidas fueron de 1,0045% en la subida y de 1,00697 en la bajada, por lo que todo indica que lo esperado según el análisis se corrobora con las observaciones.

4.2.3 Escenario de pérdida de paquetes de 2% con servicios IPTV y VoIP activados.

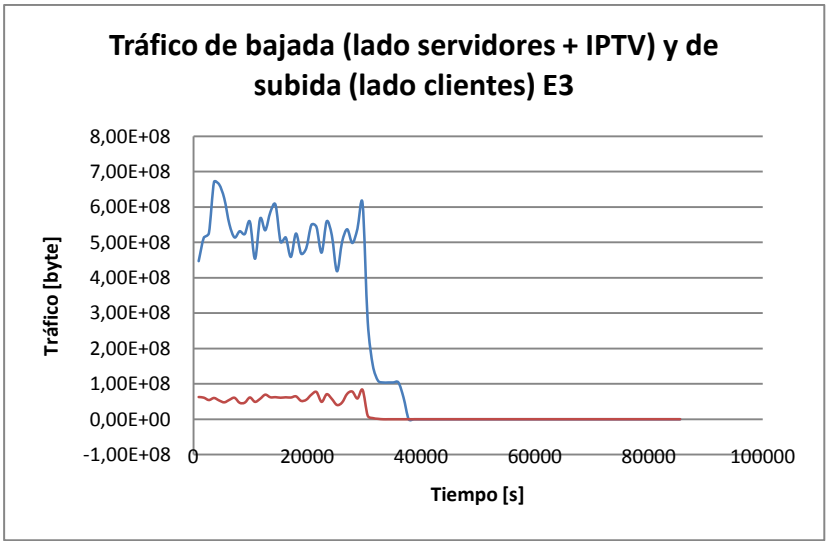


Figura 4.7: Tráfico de bajada (azul) y de subida (marrón) para escenario E3.

En la figura 4.7 se observa una diferencia entre el tráfico de subida y de bajada mucho menor que en E2 y muy similar a la observada en el caso E1 esto se debe a que las configuraciones para la red y tráfico entre E1 y E3 son idénticas salvo por la configuración de 2% de pérdida de paquetes en el nodo de acceso. Se esperan entonces tráficos de subida medidos en el lado clientes muy similares a los del caso E1 pues estos tráficos no han sido afectados por las diferentes pérdidas de paquetes en el nodo de acceso. Se tiene que los resultados de la simulación arrojan un tráfico de subida en E1 de 1.989.964.590 bytes mientras que para E3 se tienen 1.990.130.840 bytes por lo que se aprecia una inesperada diferencia de aproximadamente un 0.008% en las observaciones que a pesar de ser mínima sí es inesperada. Por otro lado se tiene que la diferencias en los tráficos de bajada entre los escenarios E1 y E3 (ver tabla 4.4) deben estar dados por las diferencias en las pérdidas de paquetes configuradas para el nodo de acceso.

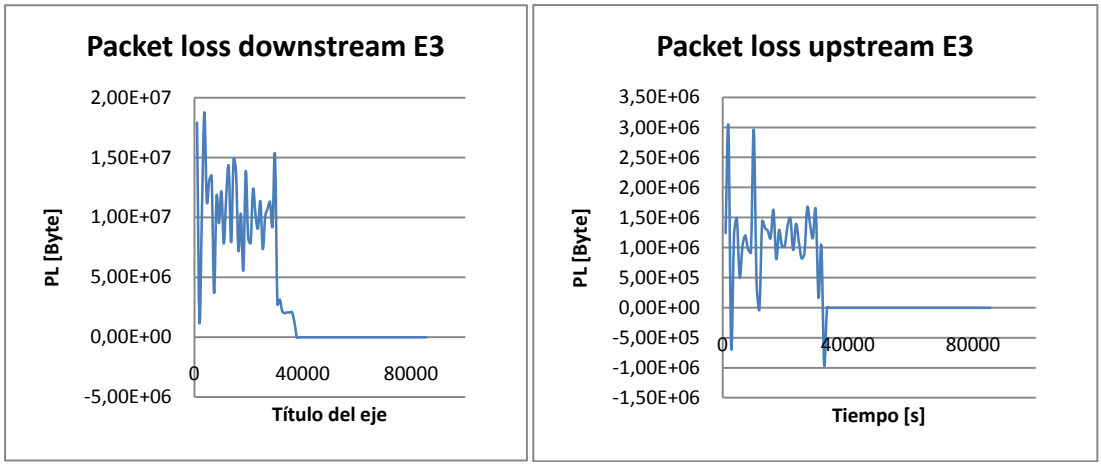


Figura 4.8: Pérdidas de paquetes en cada intervalo de medición, escenario E3.

En la figura 4.8 se muestran las pérdidas de paquetes obtenidas en cada intervalo de medición de 15 minutos, al igual que en los escenarios E1 y E2 (E4 también) se espera que si bien pueden haber resultados negativos, la mayoría y la suma total de estas mediciones debe ser positiva, es más, la suma total corresponde a pérdida de paquetes total de la simulación, items que se cumplen en las observaciones de la figura 4.8. Para este caso, como el nodo de acceso está configurado al 2% de pérdida de paquetes se espera que la pérdida de paquetes total esté en torno al 2% en el tráfico de bajada medido por el lado de los servidores. Los valores obtenidos (ver tabla 4.4) fueron 1,98011% de pérdida de paquetes en el tráfico de subida y de un 1,99969% en el tráfico de bajada. Estos valores resultantes de la simulación indican que las pérdidas fueron mayores que en el escenario E1 y que básicamente se ajustaron al valor configurado en las pérdidas de paquetes del nodo, quedando explicas las diferencias de tráfico de baja entre los escenarios E1 y E3.

4.2.4 Escenario de pérdida de paquetes de 2% sin servicios IPTV y VoIP activados.

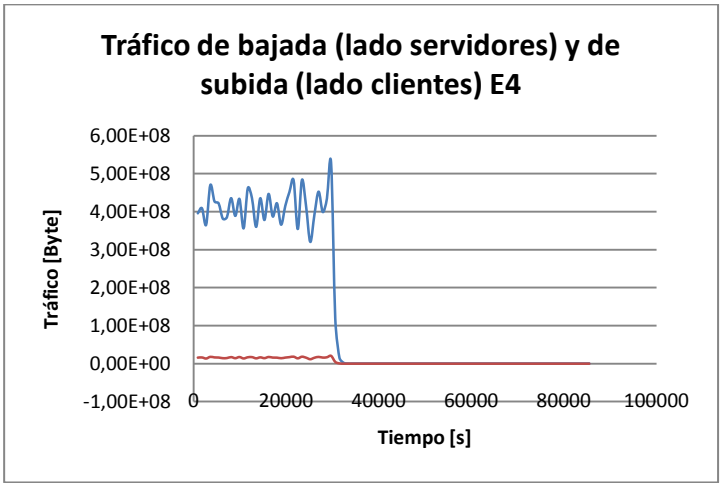


Figura 4.9: Tráfico de bajada (azul) y de subida (marrón) para escenario E4.

Nuevamente se tiene que, tal como se presenta en la figura 4.5 del escenario E2, la figura 4.9 del escenario E4 presenta una gran diferencia entre los tráficos de bajada y de subida, esto es propio de las configuraciones en las cuales no se tienen aplicaciones con proporciones entre sus flujos de bajada y subida en rangos menores o más cercanos a 1 y en este escenario no se tienen tales aplicaciones (este escenario no posee el servicio VoIP de razón igual a 1) por lo que el gráfico presente en la figura 4.9 es esperable con una razón bajada versus subida de 26,54 (ver tabla 4.4). Las diferencias entre las razones bajada versus subida y tráficos de bajada entre los escenarios E2 y E4 se explican por las diferencias en las pérdidas de paquetes configuradas.

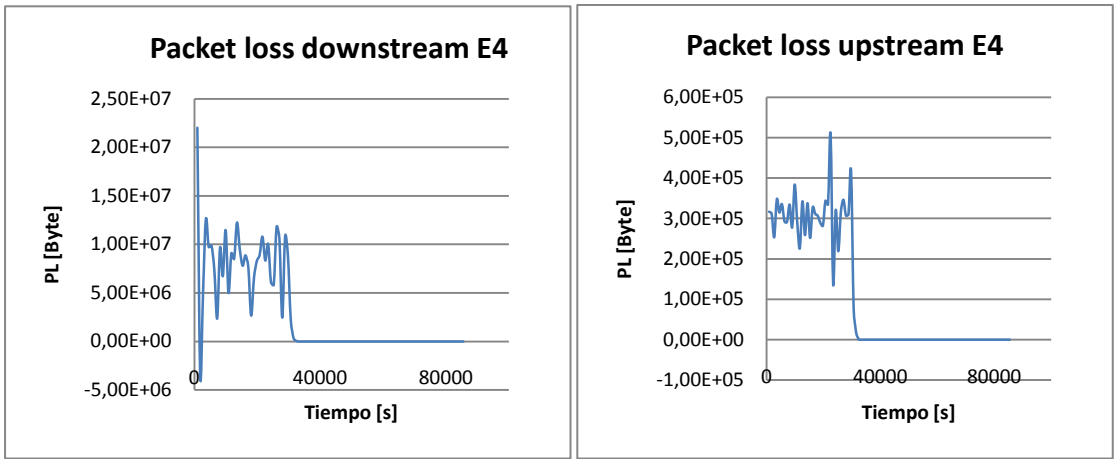


Figura 4.10: Pérdidas de paquetes en cada intervalo de medición, escenario E4.

En la figura 4.10 se tienen las pérdidas de paquetes para cada intervalo de tiempo, las cuales, al igual que para el análisis en E1, E2 y E3 son en su gran mayoría positivas y la suma también lo es. La pérdida de paquetes total medidas debería ser muy similar a la configurada de un 2% en el nodo de acceso ya que como hemos visto en los casos distintos escenarios, el resto de la red no aporta pérdida de paquetes y los resultados son (ver tabla 4.4) de 1,98318% en el tráfico de subida y de 2,0001% en el tráfico de bajada, resultados que confirman lo esperado. El hecho que la pérdida de paquetes sea el doble en E4 con respecto a E2 explica las diferencias en el tráfico de bajada entre ambos escenarios mas no explica la diferencia de un 0.035% entre los tráficos de subida pre nodo de acceso entre ambos, aquí se tiene una diferencia mínima, tal vez despreciable pero inesperada ya que ambas configuraciones son iguales excepto en el nodo de acceso.

4.3 Simulación de tráfico con selección de tiempo de inicio distribuido.

Simulaciones con tráfico de un día distribuido exponencialmente en el tiempo con carga de bajada de 10 GB por día variando la carga de subida y cambiando cada vez el RNG. Con carga de subida variando desde una carga mínima de 200 MB hasta los 5 GB diarios además la función de densidad de probabilidad, asociada a los tiempos de inicio de cada sesión de cada aplicación, se configura con una esperanza de 43.200 lo que equivale al beta de la función exponencial.

$$f(x,\beta) = \begin{cases} \frac{1}{\beta} e^{-x/\beta}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

Por esto se crea un estimador de beta calculando la esperanza muestral y se denomina beta_up y beta_down a estos valores a razón de compararlos con β (43.200 [s]), esperanza real de la distribución exponencial, configurado previamente en la simulación.

Para tener una gran cantidad de datos se realizan varias simulaciones, que sumándolas son 27, donde se tiene que las primeras 25 tienen un tiempo de simulación del doble (86.400 [s]) de la esperanza configurada (43.200 [s]). y las últimas dos se realizan con un tiempo de simulación de 864.000 [s], es decir, 20 veces la esperanza configurada. El tamaño de las muestras fue de 6.000 ya que se configura el simulador con 2.000 clientes ejecutando cada uno 3 aplicaciones. La distribución matemática de las esperanzas muestrales, en estricto rigor, corresponde a una distribución gamma que surge de la suma de 6.000 observaciones de variables aleatorias exponencialmente distribuidas. Pero el tamaño muestral de 6.000 es lo suficientemente grande (ver [16]) para que la distribución de la esperanza muestral sea aproximada, gracias al teorema del límite central, a una distribución normal. Con esto se procede a definir un intervalo de confianza del 95% para evaluar los resultados de las esperanzas muestrales beta_down's y beta_up's y que estos no se escapen, en su mayoría, del intervalo.

Si los supuestos de las simulaciones son correctos, la diferencia entre las esperanzas muestrales de la real, deberían estar acotadas, con un intervalo de confianza del 95%, entre los 42.106,9 [s] y los 44.293,1[s] .

A continuación se presentan los resultados de 25 simulaciones, todas con un tiempo de simulación de 84.600[s].

N° Simulación	Razón down/up	RNG	beta_down	beta_up
1	2,00	0	29.600	29.598
2	2,08	1	29.970	29.969
3	2,17	2	30.209	30.208
4	2,27	3	30.104	30.102
5	2,38	4	29.646	29.645
6	2,50	5	29.729	29.727
7	2,63	6	29.438	29.437
8	2,78	7	29.962	29.962
9	2,94	8	29.826	29.825
10	3,12	9	30.158	30.157
11	3,33	10	30.125	30.125
12	3,57	11	30.130	30.130
13	3,85	12	30.598	30.598
14	4,17	13	29.996	29.994
15	4,55	14	29.666	29.665
16	5,00	15	30.132	30.132
17	5,56	16	30.213	30.213
18	6,25	17	29.911	29.910
19	7,14	18	30.051	30.051
20	8,33	19	30.041	30.040
21	10,00	20	29.456	29.457
22	12,50	21	29.724	29.724
23	16,67	22	30.184	30.185
24	25,06	23	30.584	30.600
25	50,00	24	30.211	30.210

Tabla 4.5: Información sobre simulaciones con tiempos exponenciales para 25 iteraciones.

En la tabla 4.5 Se observa que las razones up/down son coherentes con lo esperado pues son los resultados de variar, a pasos constantes de 200 MB, la subida desde los 5GB hasta los 200 MB para un tráfico de bajada constante de 10 GB. La columna de los RNG simplemente indica que se utilizaron diferentes semillas para el *random number generator*. Las columnas beta down y beta up corresponden a las esperanzas muestrales pero claramente ninguno de estos 50 valores está dentro del intervalo entre 42.106,9 y 44.293,1 definido anteriormente, resultado que indica que, la distribución inicial de las variables con las cuales se crearon las muestras no eran una exponencial centrada en 43.200. Se deduce entonces que algo provoca una distorsión en la distribución inicial. Esta distorsión se debe a que una característica integrada al sistema, el truncado de los valores que están fuera de los tiempos de simulación y su reasignación hasta tomar valores que estén dentro, provoca que para tiempos de simulación “pequeños” se pierda la distribución original y se transforme en otra

totalmente diferente. Esta característica de truncamiento se creó con la finalidad de que cada aplicación que inicia cada cliente tenga un espacio para ejecutar y generar flujos de tráfico dentro del tiempo de simulación. Para probar que éste es el problema original se toman en cuenta los casos extremos del tráfico de subida no cambiando ningún parámetro en la simulación excepto por el tiempo de simulación. En los 25 casos anteriores se tiene un tiempo de simulación 2 veces mayor a la esperanza y a continuación se pueden observar los resultados de los nuevos casos a simular con tiempos de simulación de 20 veces la esperanza.

N° Simulación	Razón down/up	RNG	Beta_down	Beta_up
1b_extendida	2,00	0	43.250	43.248
25b_extendido	50,00	24	44.027	44.027

Tabla 4.6: Información para extensiones de tiempo de simulación.

En los resultados de la tabla 4.6 se tiene que las 4 esperanzas muestrales están dentro del intervalo de confianza definido para la esperanza real, por lo que se puede observar que al aumentar el tiempo de simulación el efecto de truncamiento pierde su relevancia.

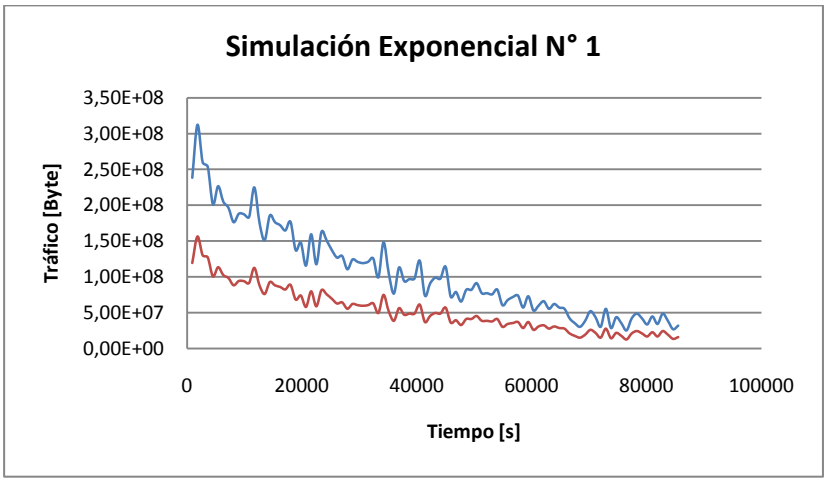


Figura 4.11: Tráfico de bajada (azul) y de subida (marrón) sim. N°1

En la Figura 4.11 se observa que la razón entre el tráfico de bajada y el de subida es efectivamente 2 (ver tabla 4.5). Por otro lado a simple vista no se puede determinar la distribución y aunque pareciese una exponencial (decrece cada vez más lento) los resultados muestran claramente el error. El valor de esperanza muestral en este caso es de 29.600 en la bajada y de 29.598 en la subida.

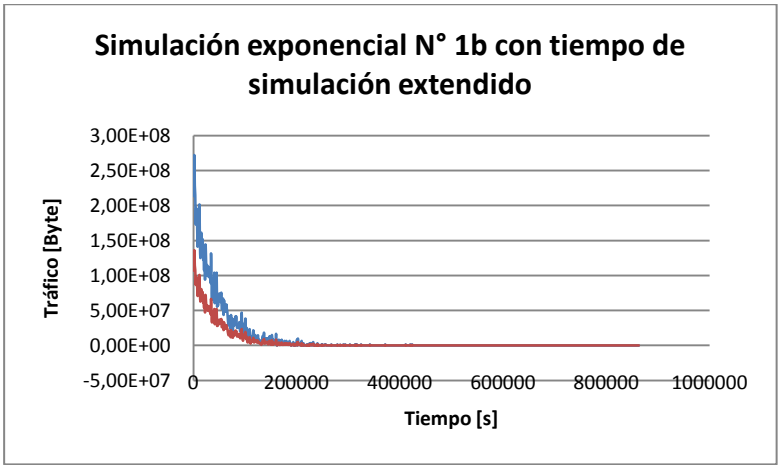


Figura 4.12: Tráfico de bajada (azul) y de subida (marrón) sim. N° 1b

En la figura 4.12 se observa el caso en que el tiempo de la simulación es 20 veces el tamaño de la muestra si bien las curvas pareciesen estar más cercanas una de la otra los resultados muestran que no pues el tráfico de bajada es el doble del tráfico de subida (ver tabla 4.6). El valor de la esperanza muestral en este caso es de 43.250 en la bajada y de 43.248 en la subida siendo idéntico al caso simulación exponencial N°1 salvo por el tiempo de simulación, por lo que se observa que aumentar el tiempo de simulación, en este caso, corrige el problema.

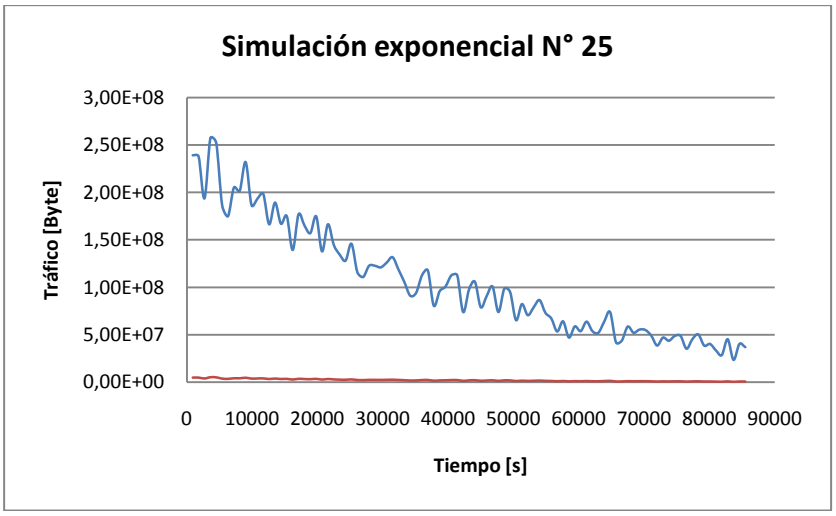


Figura 4.13: Tráfico de bajada (azul) y de subida (marrón) sim. N°25

A simple vista la figura 4.13 muestra un gráfico con tráficos que tienen, entre sí, una diferencia más grande en su magnitud comparativamente con la Figura 4.11 de hecho, la razón de tráfico de bajada versus tráfico de subida es de 50. Las esperanzas muestrales para la bajada y la subida son 30.211 y 30.210 respectivamente.

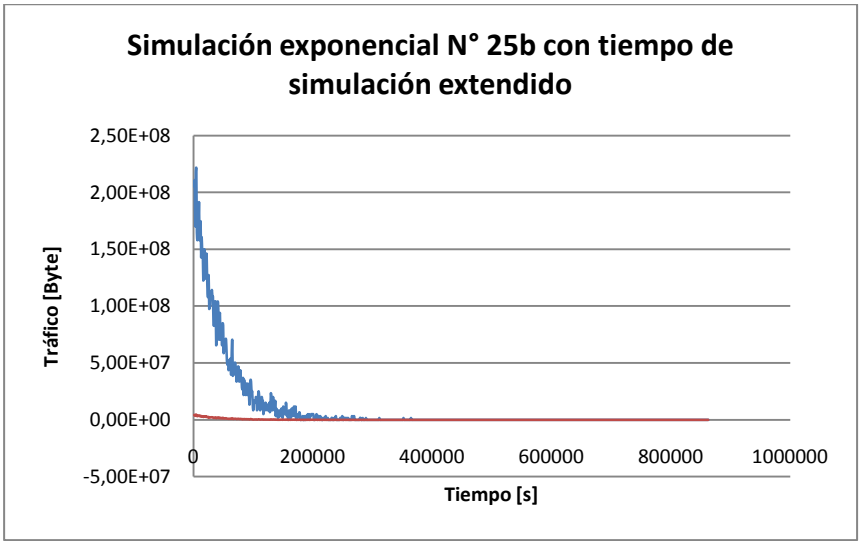


Figura 4.14: Tráfico de bajada (azul) y de subida (marrón) sim. N°25b

En la figura 4.14 se muestra el caso 25b que tiene una configuración en la simulación idéntica, salvo por el tiempo de simulación, al caso 25 de la figura 4.13 por esto se observa que se mantiene una razón tráfico de bajada versus tráfico de subida de 50 veces pero este caso le permite a la curva exponencial expresarse sin truncarla a diferencia de lo que pasa en el caso 25.Esto se corrobora por los números obteniendo una esperanza muestral de 44.027 para la bajada y la subida lo que está dentro del intervalo de confianza y por lo tanto soluciona el problema de los resultados insatisfactorios.

5 CONCLUSIONES

5.1 Objetivos

La gran cantidad de parámetros que se tienen para configurar y simular la red, incluyendo al nodo de acceso, los anchos de banda y los clientes, y la también gran cantidad de parámetros que se tienen para configurar el tráfico, permitiéndose la mezcla de hasta 10 tipos de aplicaciones típicas de internet diferentes, cada una con sus respectivas configuraciones y al mismo tiempo poder tener clientes que ejecuten aplicaciones de tráfico de distintas distribuciones estadísticas en uno o más nodos, otorga una enorme cantidad de posibilidades para la creación de escenarios de simulación, permitiendo así la proyección de tráfico para cada uno de estos escenarios pudiéndose obtener resultados para la aparición de nuevas aplicaciones o servicios o para cambios permitidos en la estructura de la red y el comportamiento de esta. Además el nodo DSLAM modelado e implementado es sumamente genérico lo que permite cambios en su configuración y funcionamiento, como puede ser cambiar su retardo de procesamiento, anchos de banda asociados, comportamiento y tamaño de colas, compatibilidad con la aplicación IPTV o la pérdida de paquetes extra que se quiere tener frente a condiciones normales. También se puede tener más de un nodo, cada uno con su cantidad de clientes y mediciones independientes. Por estas razones se logra tener simulaciones que presentan gran flexibilidad en cuanto a caracterización del nodo, tráfico y red, superando las expectativas y generando proyecciones de tráfico a discreción del usuario.

Los datos fuente son generalmente asociados al tráfico de subida y bajada a través de un DSLAM y a características de la red como cantidad de clientes, etc. En el caso del presente trabajo esos son los datos fuentes utilizados en la sección 4.1 del capítulo de resultados, además, en esa sección, se comparan los datos proyectados de la simulación con los datos fuentes obteniendo resultados bastante similares con lo que se cumple así con la necesidad de representar escenarios de acuerdo a datos reales, su posterior simulación y la obtención de datos proyectados coherentes con los datos de verificación. Por otro lado las características del proyecto realizado permiten también la comparación con datos de retardo, tráfico en diversos lugares de la red y pérdida de paquetes si es que estos datos fuente estuviesen disponibles también.

Uno de los principales retos del trabajo es la concreción de una interfaz gráfica que por un lado sea “amigable” al usuario, por otro lado logre tener disponible la configuración de todos los parámetros de la red los cuales se presentan en el desarrollo y además que logre crear código del lenguaje descriptivo de la red propio del simulador OMNeT++ y archivos necesarios para las simulaciones. Se lograron implementar todos los requisitos pero además de eso se puede guardar y cargar archivos que guardan configuraciones de un nodo en particular permitiendo compartir configuraciones de nodos entre diferentes escenarios de simulación y también permite guardar y cargar archivos que contienen la configuración de todos los parámetros para una simulación dada.

Con respecto a la posibilidad de generar reportes a partir de la simulación y proyección, el simulador tiene dos partes, la primera es que a medida que avanza el tiempo de simulación se va guardando los datos previamente seleccionados en la paleta de mediciones de la interfaz gráfica. Para eso se trabajo en el código C++ de los módulos IP de cada nodo de acceso y también se crearon varios módulos más y metodologías para la recolección de los datos para los clientes, nodos y servidores. Permitiendo así guardar en vectores los tráficos, la cantidad de clientes con retardos que superen un límite configurable y los retardos máximos globales. Cabe destacar que en la medición de tráficos estos se realizan por el lado clientes, por el lado servidores y en cada nodo con vectores diferentes para cada nodo. La segunda parte del simulador pasa los valores de los vectores a documentación manejable como lo son los datos en csv y esto se logra ya que OMNeT++ realiza el trabajo pues simplemente hay que realizar un click sobre cada vector y se exporta como csv. Este método es el que se realiza para trabajar los datos necesarios para creación de la sección de resultados. Se tiene entonces que efectivamente se satisface la necesidad de crear reportes de diferentes mediciones.

Las restricciones que se presentan para el funcionamiento de todo el proyecto en su conjunto resultan en la limitación que pueden tener ciertos parámetros. Teniendo así un conjunto de valores mínimos y máximos aproximados para cada parámetro pues si un parámetro baja le da espacio para que otro suba o viceversa, por esto los límites de operación se presentan en la sección de desarrollo en tabla 3.2, tabla 3.3, tabla 3.4, tabla 3.5 y tabla 3.6. Estos valores fueron obtenidos por inspección del código fuente y en casos de no ser esto factible se obtuvieron por prueba y error pero dan un marco para la utilización adecuada de la herramienta creada y denominada Genersim.

Una de las capacidades más destacadas del proyecto es la posibilidad de generar tráfico que sigue una función de densidad de probabilidad. La distribución es en el tiempo y se puede elegir entre 13 tipos de distribuciones diferentes donde a cada una de ellas se le pueden configurar sus parámetros. Es importante destacar que se puede elegir una distribución diferente para cada grupo de clientes de este tipo en cada nodo y se puede tener hasta 3 grupos de estos por nodo permitiendo la posibilidad de crear nuevas distribuciones temporales a partir de las combinaciones de diferentes tipos de tráfico. Un ejemplo de una distribución exponencial se encuentra realizado en el presente documento en la sección 4.3 de resultados.

5.2 Desarrollo

En la sección de desarrollo se muestran la gran cantidad de parámetros necesarios para tener una herramienta funcional y con la flexibilidad que se requiere para poder producir nuevas pruebas y nuevas simulaciones, esto implica tanto inspección para entender cómo funciona el código de librerías ya implementadas e identificar así el lugar preciso de donde insertar o quitar funciones como la creatividad

para modificarlo e insertar nuevas funcionalidades sin alterar el comportamiento previo como también la creación de módulos completos y compatibles con los demás. También se muestra parte del proceso que se realiza para hacer un modelo de las diferentes partes de la red.

5.3 Resultados

Para los resultados de la sección 4.1 se tiene que hay características, no necesariamente triviales que se pueden simular, por ejemplo, la concentración del tráfico durante el primer tercio del día y volver a tener la misma concentración al siguiente día, es decir se puede repetir las cargas de tráfico para una separación arbitraria entre días, además se logran cargas de tráfico proyectado similares a los datos de verificación. Se concluye entonces que se pueden simular condiciones singulares como la concentración temporal y la repetición dentro de una misma simulación.

En la sección 4.2 donde se presentan 4 escenarios de simulación diferenciándose por porcentaje de pérdida de paquetes y por servicios, a modo general se concluye que las pérdidas de paquetes configuradas en el nodo de acceso calzan de manera adecuada con los datos proyectados por lo que se deduce que prácticamente no hay pérdidas de paquetes en la red y solo se crean pérdidas en el nodo de acceso ya que para los casos de mayor tráfico las pérdidas fueron iguales a las configuradas en el nodo.

En la sección 4.2 se tienen escenarios que comparan los tráficos para el caso en que se simula con servicios IPTV y VoIP activados y cuando no, resultando en un claro aumento del tráfico cuando sí se tienen estos servicios pues son configurados con sus valores reales y superan en carga a las aplicaciones configuradas como base. Además se tiene que cuando estos servicios están activas existe una disminución en la razón tráfico de bajada versus tráfico de subida y esto se entiende notando el hecho de que las aplicaciones VoIP tienen un equilibrio entre el tráfico de bajada y la subida, lo que disminuye la razón global cuando este servicio está presente.

En la sección 4.3, donde se trabaja con una distribución temporal exponencial del tráfico, se tiene que los resultados proyectados no son los que se esperan desde la teoría, entonces se deduce que la distribución no era exponencial pues el proyecto trunca los valores de los flujos de tráfico que comienzan fuera del tiempo de simulación y obliga a las aplicaciones a tomar nuevos valores hasta que sí estén dentro, lo que deforma la función y cambia su esperanza y tipo. Por esto se realizaron simulaciones con mayor tiempo de simulación y el problema se suprime de donde se concluye que la herramienta sí es válida para hacer análisis estadísticos pero se tiene que utilizar tomando las precauciones necesarias.

5.4 Trabajos futuros

Los trabajos futuros podrían abarcar una amplia gama de posibilidades, como por ejemplo:

- La implementación de IPv6.
- La ampliación de la red para sistemas inalámbricos.
- La agrupación de las mediciones de retardo por nodo.
- Ofrecer una mayor cantidad de grupos de clientes.
- Insertar una mayor cantidad de servidores y aplicaciones.
- Crear nuevas topologías e interconexiones.
- Aumentar la eficiencia del código para tener simulaciones más rápidas.
- Portar el código desde OMNeT++ 4.0 a la versión 4.2.
- Replicar los conceptos en otro simulador y comparar rendimientos.
- Integrar nuevas funcionalidades con aplicaciones que utilicen los protocolos de multicast o broadcast.

BIBLIOGRAFIA

[1] GAMER T. y SCHARF M., “Realistic Simulation Environments for IP-based Networks”, Mayo 2008, 7p.
[2] Equipo de desarrollo de INET, [en línea] <http://inet.omnetpp.org/doc/INET/neddoc/index.html> [consulta: 21/10/2011]
[3] URRUTIA P., “Modelación y optimización de redes IP usando herramientas de inteligencia computacional”, Agosto 2007, 128p. Facultad de Ciencias Físicas y Matemáticas, Universidad de Chile.
[4] GONZÁLEZ M., “Modelamiento de tráfico en redes de datos”, Noviembre 2007, 111p. Facultad de Ciencias Físicas y Matemáticas, Universidad de Chile.
[5] Agilent Technologies, “Understanding DSLAM and BRAS Access Devices”, 2006, 16p.
[6] URRUTIA P., “Modelación y optimización de redes IP usando herramientas de inteligencia computacional”, p.78-80, Agosto 2007. Facultad de Ciencias Físicas y Matemáticas, Universidad de Chile.
[7] GUIZANI M. ET AL, “Network Modeling and Simulation a Practical Perspective”, Abril 2010, 281p.
[8] CACHINERO J., “Análisis y modelado de “Multicast” Interdominio para el soporte de servicios de video”, Septiembre 2009, 268p.
[9] VARGA A., HORNING R., “An overview of the OMNeT++ simulation environment”, 2008, 10p.
[10] SRINIVAS V., “IP Quality of Service”, 2001, 343p.
[11] PIETRZYK M. ET AL, “Challenging Statistical Classification for Operational Usage: the ADSL Case”, Mayo 2009, 27p.
[12] QUINLAN J. R., “C4.5: Program for Machine Learning”, 2003, 302p.
[13] BERNAILLE L, TEIXEIRA R., SALAMATIAN K., ”Early Application Identification”, Diciembre 2006, 12p.
[14] CROCE D. ET AL, “Capacity Estimation of ADSL links”, 2008, 12p.
[15] DOVROLIS C., RAMANATHAN P., MOORE D., “Packet-dispersion techniques and a capacity-estimation”, Diciembre 2004, 15p.
[16] CORNWELL P., “Necessary Sample Size for Good Central Limit Theorem Approximation”, Marzo 2011, 14p.

6 Anexo A: Manual de Usuario

Genersim

Simulador de nodos de acceso de banda ancha

DSLAM-IP/OLT

Manual de Usuario

Santiago, Mayo 2012

Autor: Luis Acuña [luis.acuna@playsip.org]

6.1 Introducción

En este documento se presenta el manual para la utilización de la herramienta de generación de ambientes de simulación, equipos, tráfico y medición de variables de tráfico, Genersim.

En el contexto de la planificación de redes surge la necesidad de prever y proyectar tráficos dependiendo de las condiciones cambiantes de las redes y para poder verificar como algunos cambios en los servicios o tráfico afectan el desempeño de equipos de red como lo son DSLAM-IP/OLT. Para verificar estos desempeños antes de fabricar prototipos o antes de implementarlos directamente en la red se utiliza la simulación. En este caso se utiliza el simulador OMNeT++ el que requiere de cierta preparación previa para su utilización y de ahí la necesidad de un facilitador como Genersim, que provee de una interfaz gráfica capaz de suministrarle al usuario una herramienta con la que puede cambiar los parámetros de una red tipo de uno o más DSLAM-IP/OLT de manera rápida y fácil al mismo tiempo de poder configurar el tráfico (de tipo aleatorio o realista) y las mediciones a realizarse.

El uso de Genersim permite la creación de redes con uno o más nodos de acceso y con cada nodo de acceso configurado con varios parámetros modificables como lo son el retardo del procesamiento, anchos de banda, pérdida de paquetes, tipos de cola y compatibilidad IPTV. Además en la red se puede configurar la cantidad de clientes conectados, su ancho de banda, tiempos de inicio de sesiones de tráfico, etc. El tipo de aplicaciones de las sesiones de tráfico también es configurable permitiendo flexibilidad en el tráfico resultante de la agregación de estas aplicaciones.

6.2 Instalación

Para comenzar el proceso de instalación se hace necesario revisar los archivos que se tienen en el dvd. Estos corresponden a:

- Disco duro genersim.vmdk con compatibilidad probada para VirtualBox
- Archivo de instalación de Oracle VM VirtualBox para Windows (VirtualBox-4.1.8-75467-Win).
- El presente Manual (Manual.pdf).

La instalación del simulador se realiza mediante la carga de un disco duro virtual trabajando con el programa Oracle VM VirtualBox, donde lo primero que debe hacerse es copiar el disco duro en su computador. Una vez copiado el disco duro se procede con el segundo paso.
El segundo paso es instalar el programa VirtualBox siguiendo las etapas requeridos por el instalador y luego ejecutarlo.

Al ejecutarlo, presionar el botón Nueva, tal como se aprecia en la Figura 6.1.

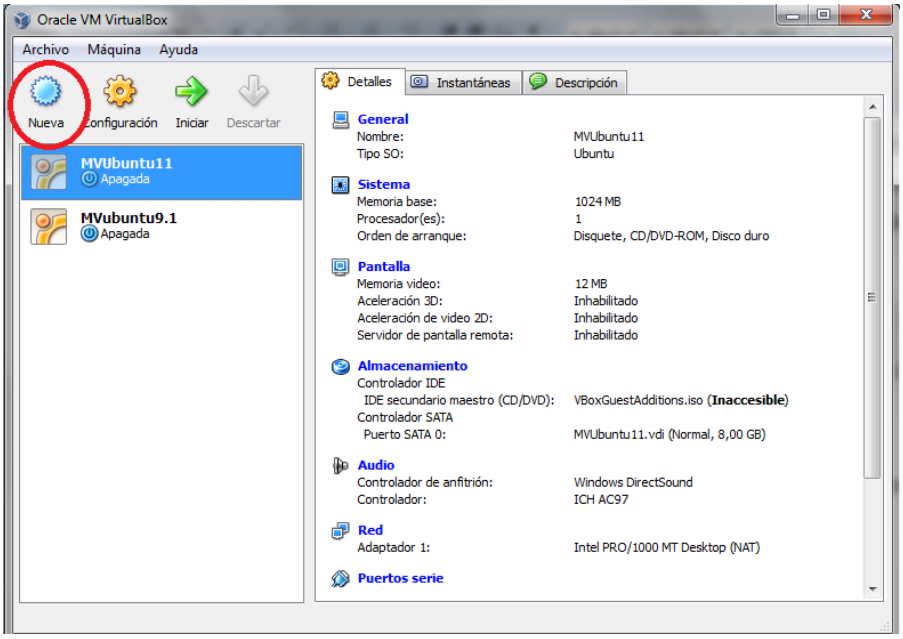


Figura 6.1: Pantalla inicial de Oracle VM VirtualBox

Luego del mensaje de bienvenida aparecerá la Figura 6.2 donde se debe configurar el nombre (se recomienda Genersim) y el tipo de sistema operativo que en este caso debe configurarse con Sistema operativo: **Linux** y Versión: **Ubuntu**.

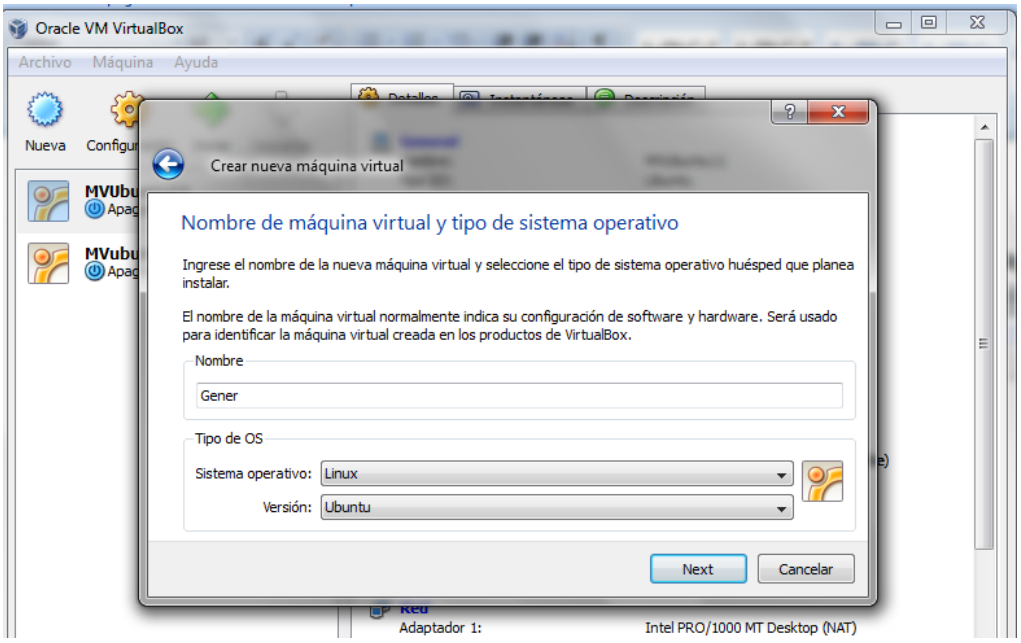


Figura 6.2: Elección de nombre y sistema operativo de nueva Máquina virtual.

Luego (Figura 6.3) se configura la memoria RAM, para esto se recomienda al menos 512 MB, pero se puede modificar más adelante a discreción del usuario.

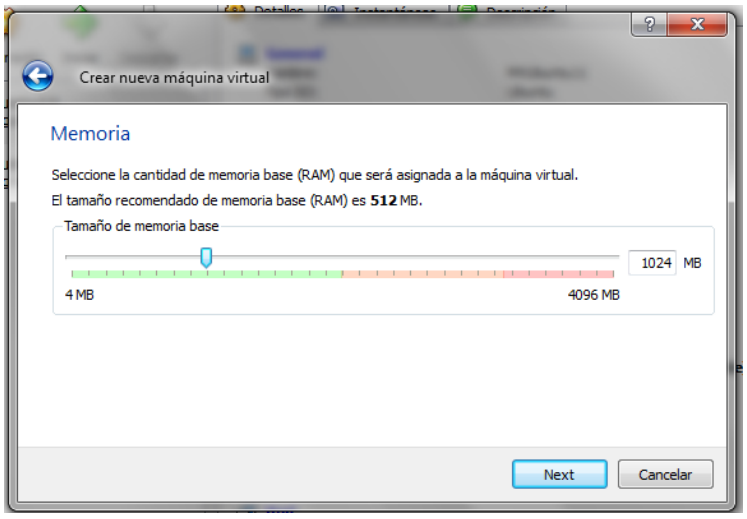


Figura 6.3: Elección de memoria en nueva máquina virtual.

En la figura 6.4 se elige el disco duro a utilizar, si es nuevo o uno ya existente. Para el caso en estudio hay que elegir *Usar disco duro existente* y hacer clic en la carpeta del círculo rojo.

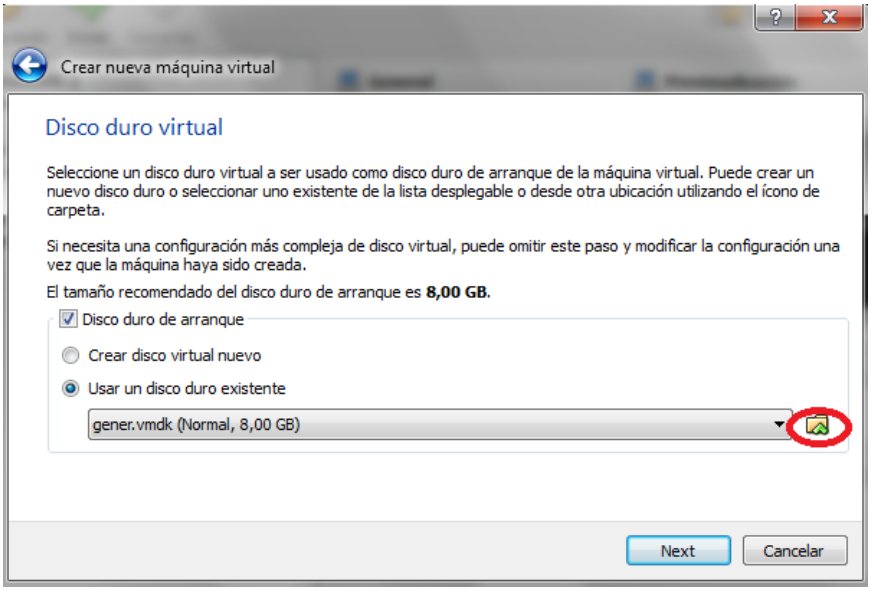


Figura 6.4: Elección del disco duro genersim.vmdk.

Una vez superados estos pasos se está listo para arrancar desde el disco Genersim.

Al arrancar Genersim se llega al sistema operativo Ubuntu 10.04 donde se han instalado los programas necesarios para las simulaciones.

6.3 Conociendo la interfaz gráfica Genersim

Una vez arrancada la máquina virtual, iniciar sesión con el usuario genersim e ingresar la contraseña suministrada por playSIP [www.playsip.org]

Figura 6.5: Inicio de sesión genersim.

Para abrir la interfaz gráfica se debe seleccionar y hacer doble clic sobre el ícono de nombre Genersim.

Figura 6.6: Lanzador de GUI Genersim

Con esto se abrirá la interfaz gráfica del proyecto para configurar parámetros de la topología, del tráfico y de las mediciones.

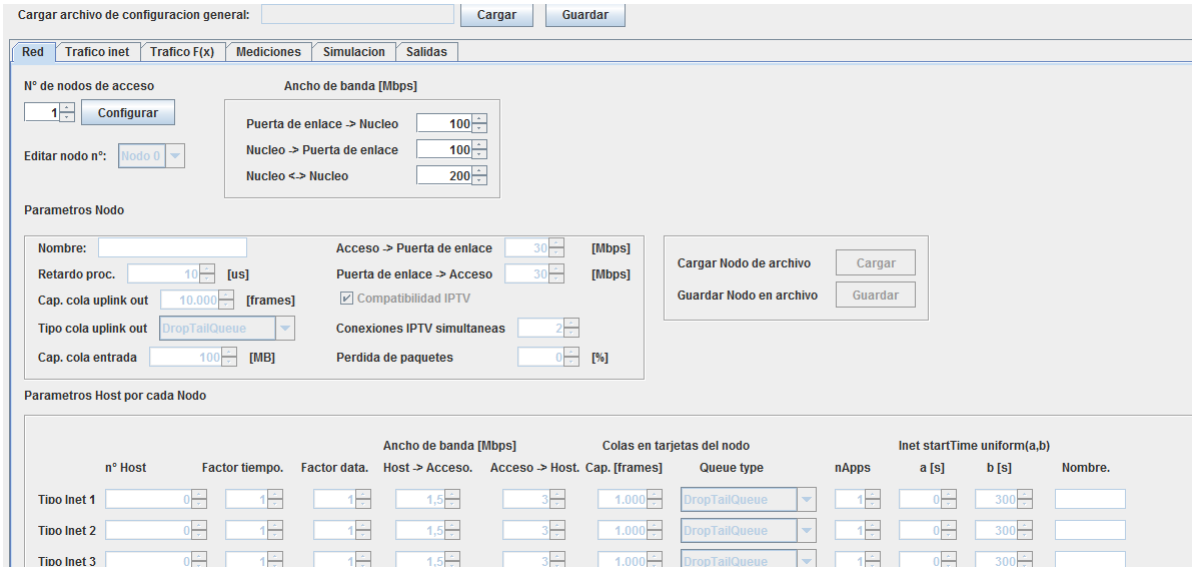


Figura 6.7: Pantalla inicial de Genersim.

Lo primero qe debe decidirse, para comenzar a llenar los parámetros, es la cantidad de nodos de acceso a simularse para luego presionar el botón Configurar, habilitando así, la edición de los demás parámetros. A continuación se explica la utilidad de cada parámetro en orden según la paleta en la que pertenece.

6.3.1 Paleta de red

Para configurar los parámetros de red se debe verificar que se está en la paleta de red, lo mismo sucede con las otras paletas (ver Figura 6.8)

Figura 6.8: Menú de paletas con la paleta red seleccionada.

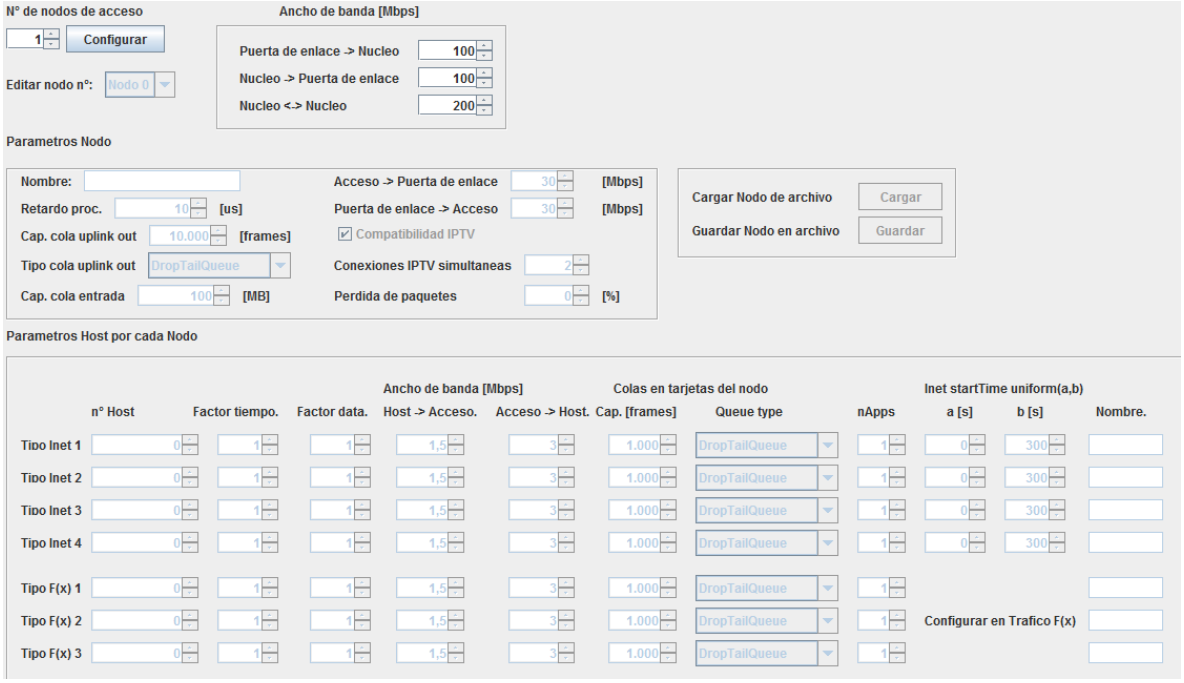


Figura 6.9: Todos los campos de la paleta red de la interfaz Genersim.

A continuación se muestran cada uno de los parámetros a configurar con su descripción.

6.3.1.1 Parámetros generales:

- **Puerta de enlace -> Núcleo:** Se configura el ancho de banda del canal que va desde la puerta de enlace al núcleo en [Mbps].

- **Núcleo -> Puerta de enlace:** Se configura el ancho de banda del canal que va desde el núcleo a la puerta de enlace en [Mbps].

- **Núcleo -> Núcleo:** Se configura el ancho de banda de los canales entre núcleos en [Mbps]

- **N° de nodos de acceso:** Para iniciar la configurar una simulación se comienza por seleccionar la cantidad de nodos de acceso a reproducirse en la red.

- **Editar nodo n°:** Luego de configurar la cantidad de nodos de acceso se elige cual nodo editar.

6.3.1.2 Parámetros del nodo que se está editando:

- **Nombre:** El nombre de referencia del nodo de acceso que se está editando.

- **Retardo proc.:** Retardo de procesamiento de los mensajes del nodo en [us].

- **Cap. Cola uplink:** Configura la cantidad de frames que se pueden tener en cola en la tarjeta que controla el flujo que va desde el nodo de acceso a la puerta de enlace.

- **Tipo cola uplink out:** Tipo de cola en la tarjeta que controla el flujo que va desde el nodo de acceso a la puerta de enlace. El tipo de cola se puede elegir como DropTailQueue o DropTailQoSQueue, donde el primero encola y bota los paquetes del final sin discriminación mientras que el segundo le da preferencia a los paquetes de mayor prioridad y luego bota los del final de la cola (Tail).

- **Cap. Cola entrada:** Determina la capacidad de la cola de entrada al procesador del DSLAM-IP/OLT (todos los paquetes de subida y de bajada), medido en [MB]

- **Acceso -> Puerta de enlace:** Configura el ancho de banda del canal cuyo flujo va desde el nodo de acceso a la puerta de enlace en [Mbps].

- **Puerta de enlace -> Acceso:** Configura el ancho de banda del canal cuyo flujo va desde la puerta de enlace al nodo de acceso en [Mbps].

- **Compatibilidad IPTV:** Configura si se puede realizar tráfico del tipo IPTV por un nodo en particular provocando que los hosts asociados al nodo en particular puedan, o no, comenzar un flujo de este tipo. No se realiza el servicio de multicast. Se utiliza el nodo de acceso como servidor IPTV y respondiendo a peticiones unitarias por parte de los clientes conectados a él, esto se realizó así para no simular el tráfico IPTV que va desde el servidor IPTV hacia el nodo de acceso.

- **Conexiones IPTV Simultaneas:** Configura la cantidad máxima de socket asociados a IPTV en un nodo en particular.

- **Pérdida de paquetes:** Configura el porcentaje de pérdidas de los paquetes que atraviesan el nodo con rango [0%,100%].

Los siguientes parámetros corresponden a diferentes tipos de host agrupados según sus características, donde se pueden tener hasta cuatro grupos de hosts tipo, con tráfico inet (El tráfico inet es aquel que se configura en la segunda paleta y se explica en la sección correspondiente) por nodo y hasta tres grupos de host tipo F(x) por nodo.
A continuación Parámetros host para cada Nodo:

6.3.1.3 Parámetros Host:

- **n° Host:** Corresponde a la cantidad de hosts o clientes que pertenecen al grupo para un nodo dado.

- **Factor tiempo:** Corresponde a un factor que multiplica los tiempos entre envíos de paquetes de peticiones y también a los tiempos de envío entre los paquetes de respuesta provocando mayor o menor velocidad en el flujo dependiendo de si el factor es menor o mayor a uno, todo esto con respecto a lo que se configura en las paletas de tráfico. Este factor al ser un multiplicador no posee unidad de medida.

- **Factor data:** Corresponde a un factor que multiplica los tamaños de las peticiones como de las respuestas aumentando el flujo de paquetes si éste es mayor a uno y disminuyendo el flujo si éste es menor a uno, todo esto con respecto a lo que se configura en las paletas de tráfico. Este factor al ser un multiplicador no posee unidad de medida.

- **Host -> Acceso [Mbps]:** Configura el ancho de banda del canal que corresponde al flujo que va desde el host al nodo de acceso medido en [Mbps].

- **Acceso -> Host [Mbps]:** Configura el ancho de banda del canal que corresponde al flujo que va desde el nodo de acceso al host medido en [Mbps].

- **Cap. [frames]:** Configura la capacidad de las colas en las tarjetas del nodo de acceso correspondientes al grupo de hosts en particular.

- **Queue type:** Configura el tipo de cola asociado a las tarjetas del nodo de acceso correspondientes al grupo de hosts en particular pudiendo elegirse DropTailQueue o DropTailQoSQueue donde el primero encola y bota los paquetes del final sin discriminación mientras que el segundo le da preferencia a los paquetes de mayor prioridad y luego bota los del final de la cola (Tail).

- **nApps:** Este parámetro configura la cantidad de aplicaciones que correrán cada uno de los hosts correspondientes al grupo. Simultáneamente pueden convivir diferentes aplicaciones para un mismo host con, por ejemplo, diferentes prioridades y características de tráfico. Por ejemplo un cliente puede al mismo tiempo correr una aplicación de Streaming UDP con prioridad 1° y una aplicación Web Traffic con prioridad en las colas de 8°. Estas aplicaciones son independientes y pueden realizar sesiones totalmente separados temporalmente.

- **a[s] y b [s]:** Corresponden a los parámetros de una función uniforme (starTime = uniform(a,b)) la cual define el tiempo de partida para cada flujo de tráfico de cada aplicación en los host tipo inet.

- **Nombre:** Corresponde al nombre del grupo, por ejemplo, ADSL, VDSL o ADSL2+.

6.3.2 Paleta de Tráfico Inet

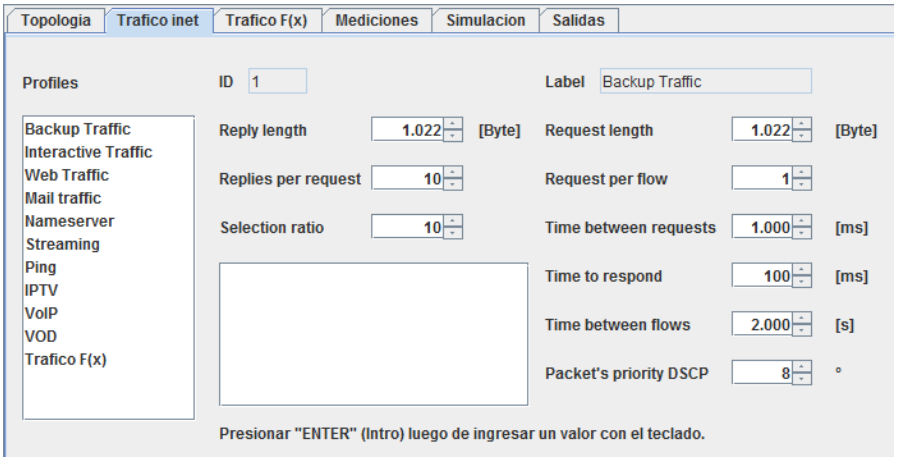


Figura 6.10: Campos para una aplicación en la paleta Tráfico inet.

En esta paleta, que se aprecia en la Figura 6.10, se configuran los diferentes tipos de tráfico que serán utilizados por los clientes (hosts) de los grupos inet. A continuación se presenta el significado de cada parámetro de tráfico:

- **Reply length:** Tamaño de la respuesta por parte del servidor para el flujo de tráfico entre un cliente (host) y un servidor medido en [Byte]

- **Replies per request:** Cantidad de respuestas desde el servidor al cliente por cada petición entrante en el flujo cliente (host) y el servidor.

- **Selection ratio:** Razón de selección entre los distintos perfiles de tráfico, a mayor la razón mayor la probabilidad de elegir la aplicación (o perfil) con respecto a las demás cada vez que se comienza un flujo.

- **Request length:** Tamaño de las peticiones asociadas a la aplicación dada, medido en [Byte].

- **Request per flow:** Cantidad de peticiones por cada flujo entre cliente y servidor para una aplicación seleccionada.

- **Time between requests:** Corresponde al tiempo de envío entre peticiones por parte de los clientes. Se mide en [ms]

- **Time to respond:** Corresponde al tiempo de envío entre las respuestas por parte del servidor. Se mide en [ms]

- **Time Between flows:** Tiempo que separa el final de una sesión de flujo por el lado del cliente para dar paso a una nueva selección de aplicación y al inicio de una nueva sesión de flujo. Se mide en [s]

- **Packet's priority DSCP:** Corresponde a la prioridad que la aplicación tiene sobre las otras y se utiliza cuando en la simulación hay colas del tipo DropTailQoSQueue la cual le da preferencia a los paquetes según su prioridad.

6.3.3 Paleta de Trafico F(X):

Este tipo de tráfico permite generar tráfico distribuido aleatoriamente en el tiempo permitiendo configurar diversas distribuciones matemáticas para esto. El funcionamiento del tráfico se logra a través de la distribución del tiempo de inicio de cada flujo entre un cliente F(x) y el servidor.

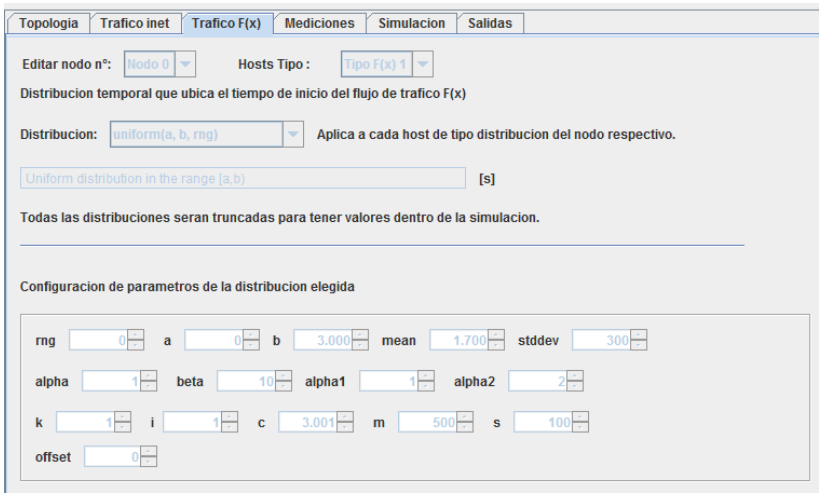


Figura 6.11: Paleta de configuración Tráfico F(x).

Como se aprecia en la Figura 6.11 se tiene que, para configurar los parámetros de cada grupo del tráfico F(x), hay 3 menús desplegables como se presentan a continuación:

- **Editar nodo n°:** Este menú muestra los nodos disponibles según la configuración permitiendo elegir el nodo de cuyos grupos de hosts F(x) se quiere editar.

- **Hosts Tipo:** En este menú se elige el grupo del nodo, que se va a editar permitiendo la edición de hasta tres grupos por nodo, cada uno con sus respectivas distribuciones matemáticas.

- **Distribucion:** Permite la selección de la distribución a ser utilizada por el grupo aplicándose a cada host del tipo F(x). Al ser seleccionada una distribución, inmediatamente después, se habilitan los campos asociados a tal distribución para así poder editarlos.

6.3.4 Paleta de Mediciones:

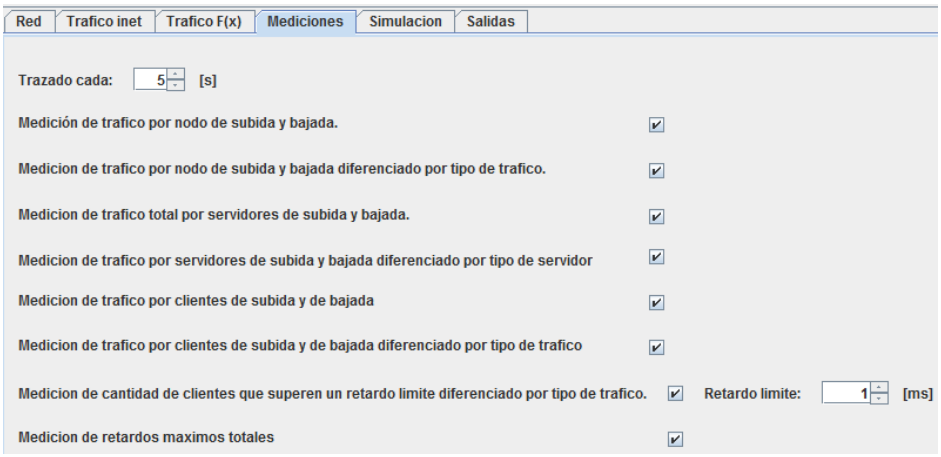


Figura 6.12: Paleta de mediciones.

En la paleta Mediciones, que se aprecia en la Figura 6.12, se tiene un parámetro:

Trazado: corresponde al intervalo de tiempo entre las tomas de medidas para un tipo de medición dado y se aplica a todos los tipos de mediciones seleccionados del sistema.

Además se tienen diferentes mediciones que se pueden realizar. Éstas son:

- **Tráfico por nodo de subida y bajada:** Corresponde a dos mediciones, la primera corresponde a la cantidad de bytes que se procesan por un nodo en la bajada y la segunda corresponde a la cantidad de bytes que se procesan en un nodo en la subida. Ambas mediciones se realizan sobre cada nodo dentro de la simulación. Incluye al tipo de tráfico IPTV y correspondería a las mediciones que en MOVISTAR se consideran del UPLINK.

- **Tráfico por nodo de subida y bajada diferenciado por tipo de tráfico:** Corresponde a dos mediciones por cada tipo de tráfico, las primera corresponden a la cantidad de Bytes que se procesan por un nodo en la bajada del tipo de tráfico correspondiente y las segundas corresponden a la cantidad de bytes que se procesan por un nodo en la subida del tipo de tráfico correspondiente. Ambas mediciones por cada tipo de tráfico se realizan sobre cada nodo dentro de la simulación. Ejemplos de tipos de tráfico: VoIP y Streaming.

- **Tráfico total por servidores de subida y bajada:** Corresponde a las cantidades de tráfico de subida y de bajada agregadas de todos los servidores en bytes por cada traza de tiempo, excluyendo IPTV pues el servidor se encuentra en el nodo. No se tienen servidores IPTV para ahorrar la carga de tráfico que esto significa entre el servidor IPTV y el nodo, por esto se optó por poner el servidor dentro del nodo entregando el servicio a los usuarios desde ahí.

- **Tráfico por servidores de subida y bajada diferenciado por tipo de servidor:** Corresponden a dos mediciones por cada servidor, la primera corresponde a la cantidad de bytes que se envían (downstream) y la segunda corresponde a la cantidad de bytes que llegan al servidor (upstream). Ejemplos de tipo de servidor: BackupServer, MailServer y VODServer.

- **Tráfico por clientes de subida y de bajada:** Corresponde a la cantidad total de tráfico transcurrido en cada traza o intervalo de tiempo por el lado de los clientes medido en bytes.

- **Tráfico por clientes de subida y bajada diferenciado por tipo de tráfico:** Corresponde a las mediciones de tráfico (upstream y downstream) por cada tipo de tráfico desde la perspectiva de los clientes. Ejemplos de tipo de tráfico pueden ser VoIP y Streaming.

- **Cantidad de clientes que superan un retardo límite diferenciado por tipo de tráfico:** Corresponde a la cantidad total de clientes (hosts) que superan un retardo límite dado y también a la cantidad de aplicaciones separados por cada tipo de tráfico que superan un retardo límite dado dentro de la simulación.

- **Retardo límite:** Corresponde al límite de retardo mínimo, el cual de ser superado, se agrega a un contador en cada intervalo de tiempo. Esto para las diferentes aplicaciones existentes. Se mide en [ms].

- **Retardos máximos totales:** Registra los retardos máximos totales que se presentan en la simulación en cada uno de los intervalos dados por el tiempo de trazado.

6.3.5 Paleta de Simulacion:

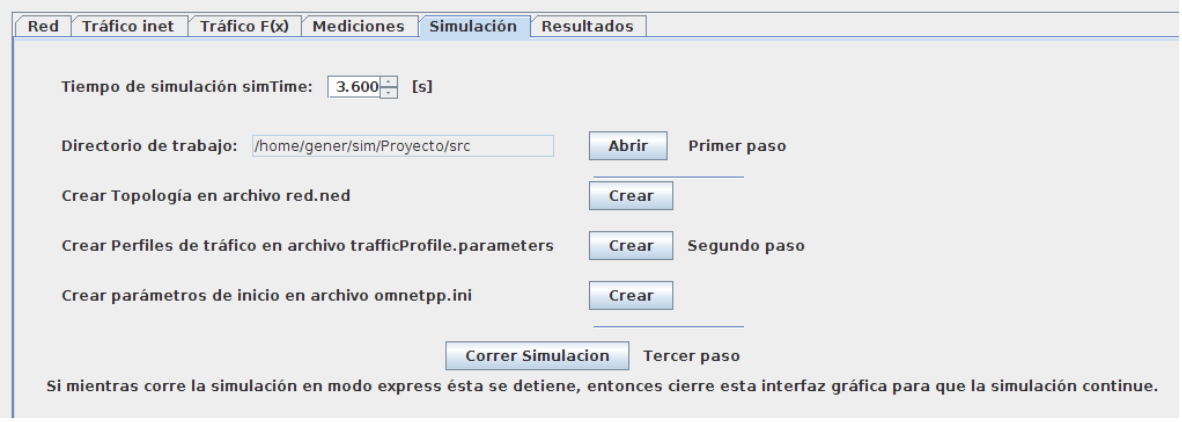


Figura 6.13: Paleta Simulación.

En la Figura 6.13 se aprecia la paleta de **Simulación**, y una vez configuradas las paletas anteriores, se realizan tres pasos. En un primer paso se elige el directorio de trabajo en el cual se debe encontrar el ejecutable del proyecto que por defecto es la dirección */home/genersim/sim/Proyecto/src* y que a priori no se debe cambiar.

Además, en un segundo paso, se crean los archivos con los parámetros de la simulación los cuales son:

- **Crear topología en archivo red.ned:** Corresponde al archivo que contiene la información de la red en cuanto a la posición de los gateways, de los servidores, de los nodos de acceso, del núcleo de la red y de los hosts y como estos se interconectan a través de canales. Es también el archivo donde se separan los grupos de hosts según su tipo. Este archivo es completamente modificable desde la interfaz gráfica y no es necesaria su verificación explicita por parte del usuario. Por esto no se entrega mayor información como un documento anexo. Las topologías permitidas son las que se pueden realizar con la interfaz, es decir, las que se pueden lograr con la siguiente estructura de la Figura 3.9. con uno o más DSLAM-IP/OLT y diferentes cantidades de clientes por DSLAM-IP/OLT. El resto es estático.

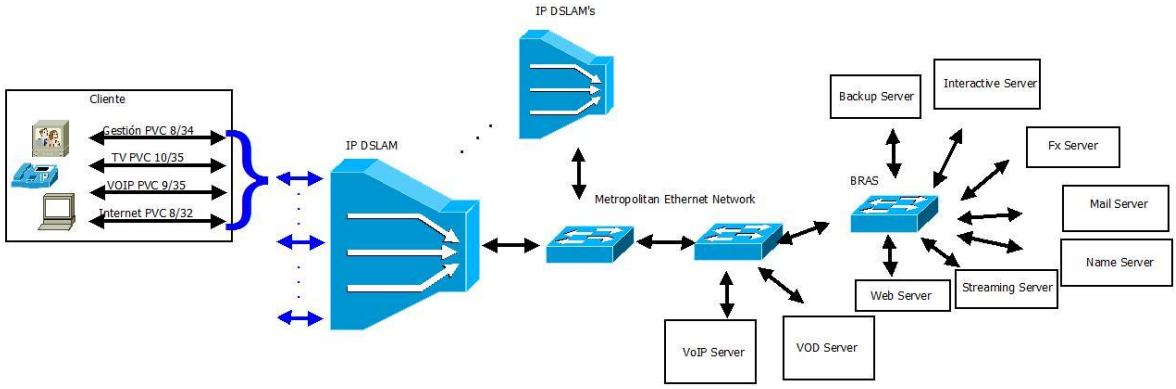


Figura 6.14 Modelo simplificado de red a simular.

- **Crear Perfiles de tráfico en archivo trafficProfile.parameters:** crea el archivo que contiene toda la información que fue editada en la paleta de tráfico Inet y que durante la simulación es leído por el módulo TrafficProfileManager el cual contendrá la información e interactuará con los clientes y servidores para enviarles la información de tráfico necesaria. Este archivo es completamente modificable desde la interfaz gráfica y no es necesaria su verificación explicita por parte del usuario. Por eso no se entrega mayor información como ir en un documento anexo.
- **Crear parámetros de inicio en archivo omnetpp.ini:** Es el archivo con parámetros de inicio como tiempo simulación, tipo de colas, clase clasificadora de mensajes y otros parámetros. Este archivo es completamente modificable desde la interfaz gráfica y no es necesaria su verificación explicita por parte del usuario. Por eso no se entrega mayor información como ir en un documento anexo.

El tercer paso corresponde a apretar el botón para correr simulación con interfaz donde luego de unos segundos debe aparecer la ventana siguiente:

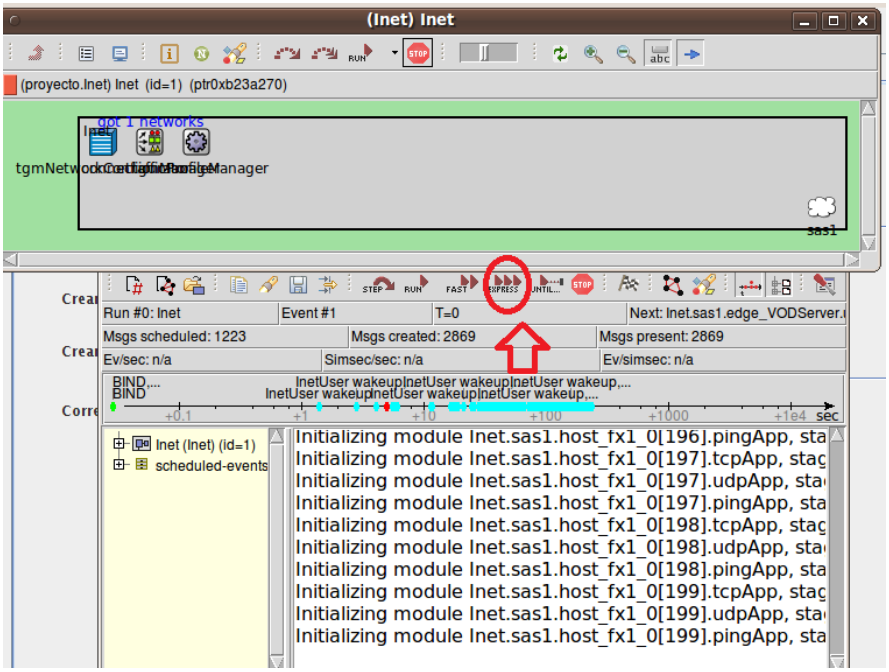


Figura 6.15: Ventanas que son parte de la simulación señalando el botón para comenzar.

Donde la flecha que apunta hacia arriba en la Figura 6.15 indica el botón que debe presionar el usuario de la interfaz para así tener una simulación en el modo más rápido (en modo **express,** donde no se tiene el output de texto mientras se simula) y antes de presionar este botón se recomienda cerrar la interfaz gráfica Genersim para evitar que se detenga la simulación inesperadamente. Luego de finalizada la simulación y si no hay errores, aparece la ventana de la Figura 6.16:

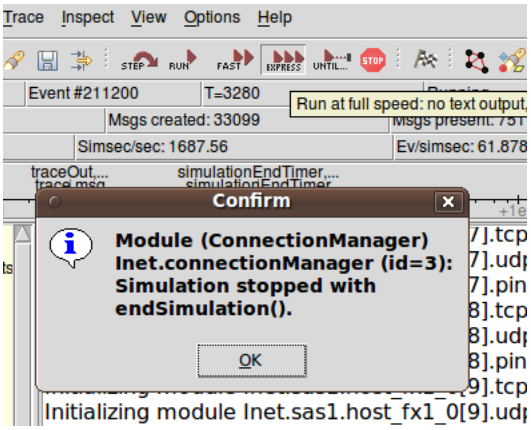


Figura 6.16: Ventana de finalización de una simulación.

Luego de presionar el botón OK se debe pasar a la paleta siguiente de Salidas.

6.3.6 Paleta de Resultados:

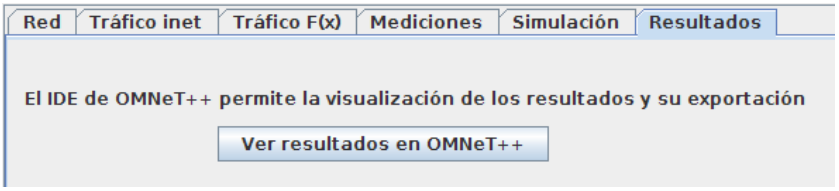


Figura 6.17: Paleta Resultados de Genersim.

Una vez finalizada la simulación se procede a abrir OMNeT++ para dirigirse a los datos obtenidos, para esto, ir a la paleta Resultados como se aprecia en la Figura 6.17 y presionar el botón **Ver resultados en OMNeT++** de la interfaz gráfica Genersim o sino presionar el ícono **OMNeT++ 4.0** del escritorio de la máquina Virtual.

Para facilitar proceso de visualización de resultados se presenta, en el capítulo siguiente, un ejemplo de uso donde se ven los resultados en detalle.

6.4 Ejemplo de uso

6.4.1 Introducción

Se tiene la información sobre los datos de tráfico de un nodo de acceso real de nombre *Santa Lucia 6* perteneciente a la empresa Movistar y corresponde a un DSLAM-IP Huawei 5300. Los datos de tráfico se tienen para los meses de Marzo-Abril del año 2011. Donde al ver los datos se observa claramente una regularidad diaria.

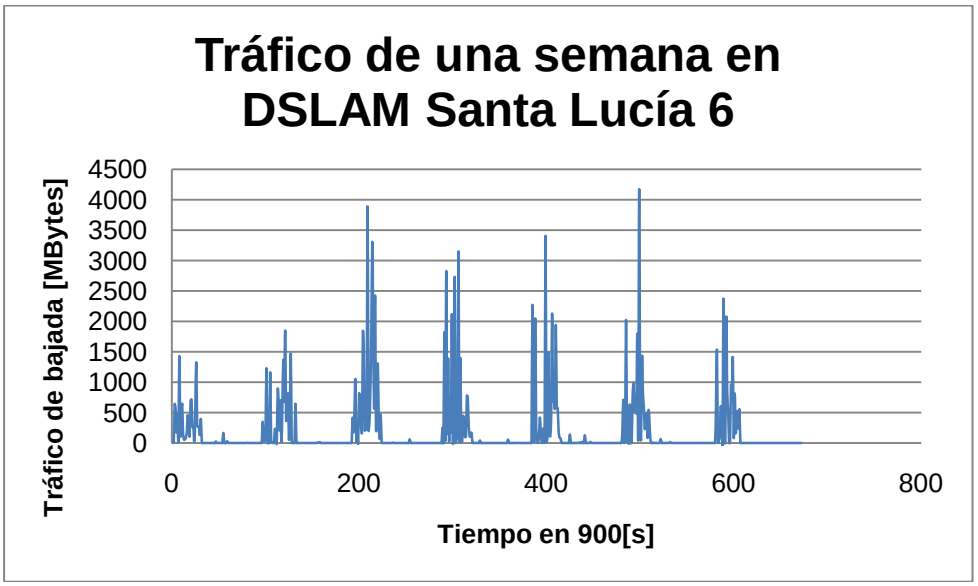


Figura 6.18: Tráfico de bajada de una semana del DSLAM Santa Lucía 6

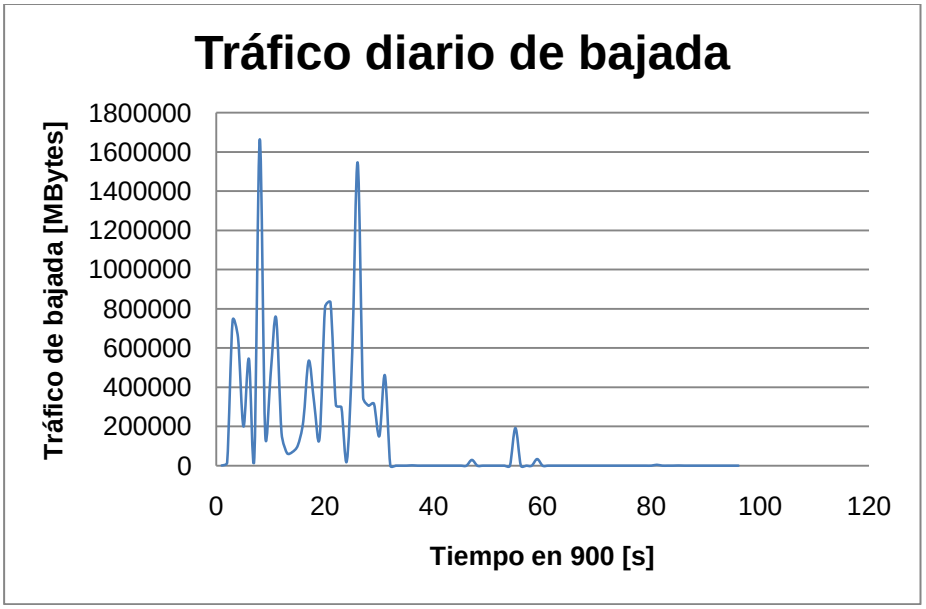


Figura 6.19: Tráfico del día con más tráfico de bajada del DSLAM Santa Lucía 6.

Donde, como se aprecia en la Figura 6.19, el tráfico de datos se concentra principalmente en el primer tercio del día. Este fenómeno es factible de simular en OMNeT++ con INET y Proyecto Genersim, ajustando los parámetros necesarios, es importante destacar que los datos con que se trabaja son de la base de datos Radius de Movistar y ésta almacena los datos al cerrar las sesiones de los clientes, de ahí el hecho que el tráfico se concentre durante las primeras ocho horas pero igualmente se utilizan estos datos como fuente.
Ahora se procederá a mostrar un ejemplo de uso donde se apreciará paso por paso la realización de una simulación. Para esto se comienza abriendo la máquina virtual con el usuario y se realiza doble clic en el ícono del lanzador de Genersim tal como se observa en la Figura 6.5 del presente manual.

A continuación, en la sección 6.4.2, se presentan los pasos para llenar los parámetros de la interfaz gráfica Genersim, pero se puede ahorrar los pasos de llenar uno a uno los parámetros cargando directamente el archivo **parametros1** al hacer clic en el botón **Cargar** de la parte superior de la interfaz gráfica Genersim y seleccionando el archivo **parametros1.** Si se hace esto puede saltarse la sección de configuración de parámetros y continuar en la sección 6.4.3 del presente manual.

6.4.2 Configuración de parámetros

En la paleta de Red se mantiene el N° de nodos de acceso en valor uno que viene por defecto y que corresponde al caso que se está tratando, y luego se presiona configurar como se aprecia en la Figura 6.20.

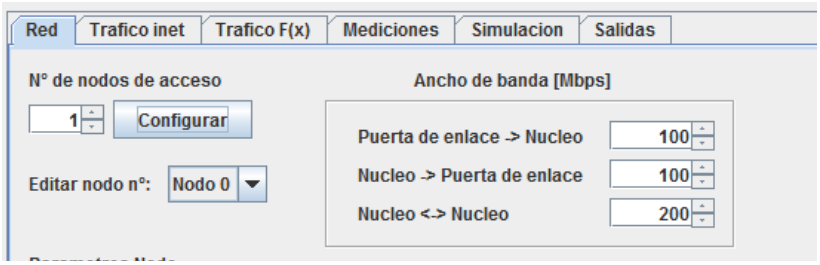


Figura 6.20: Configuración de parámetros generales

Ahora se procede a llenar los datos y parámetros del nodo de la manera en que se presenta en la Figura 6.21, notando que se utilizará una pérdida de paquetes del 5%:

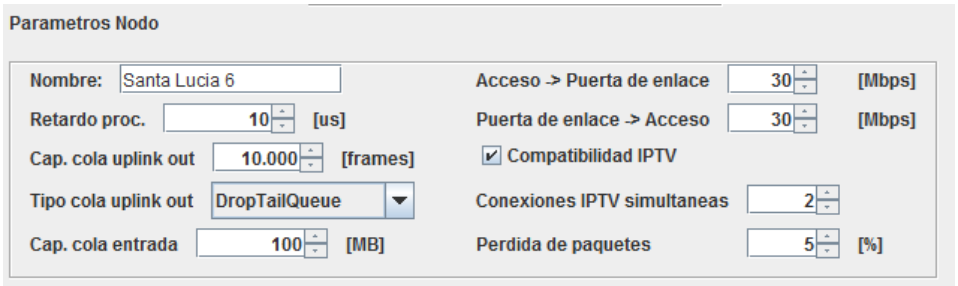


Figura 6.21: Configuración de parámetros nodo.

Los datos para los parámetros asociados a los clientes son: clientes = 648; Velocidades = 3Mbps; Cliente empresa 24; Clientes estándar 624.

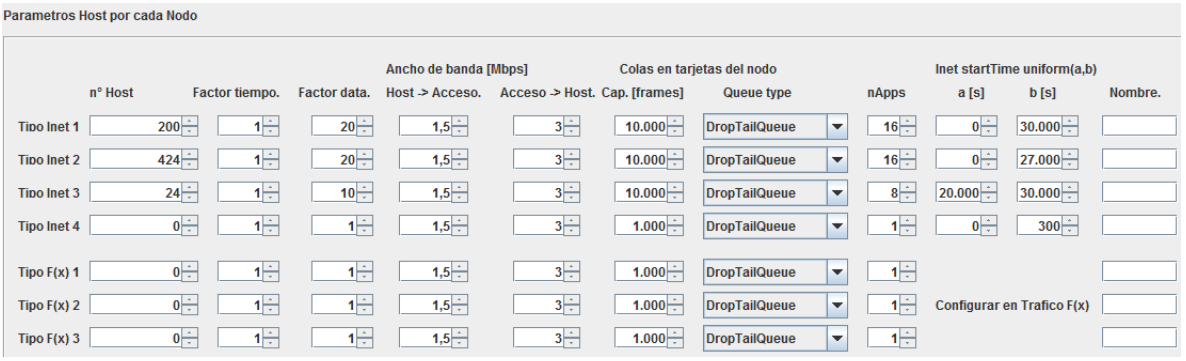


Figura 6.22: Configuración de parámetros Host.

Ahora que se tienen configurado el Nodo 0 con sus respectivos host asociados, se pasa a configurar la paleta de tráfico inet donde se configuran las distintas aplicaciones del tráfico inet que aparecen en la lista en la izquierda de la Figura 6.23.

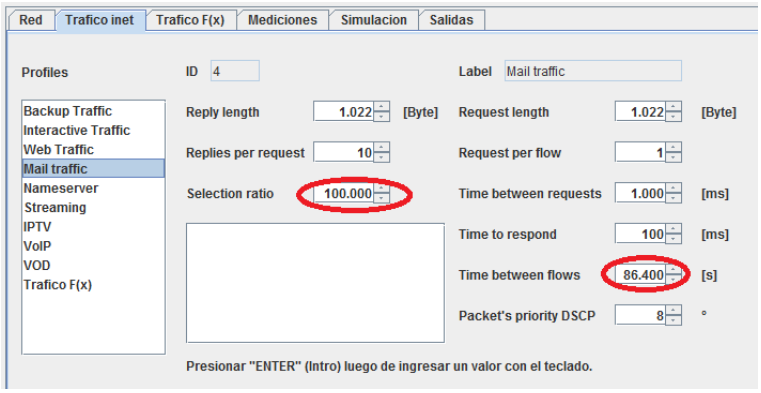


Figura 6.23. Configuración de parámetros de tráfico inet para aplicación *Mail traffic*.

En la Figura 6.23 se configuran los parámetros de tráfico para las aplicaciones Mail traffic mientras que en la Figura 6.24 se configuran los parámetros de tráfico para la aplicación Streaming notando en óvalos los cambios a los valores por defecto.

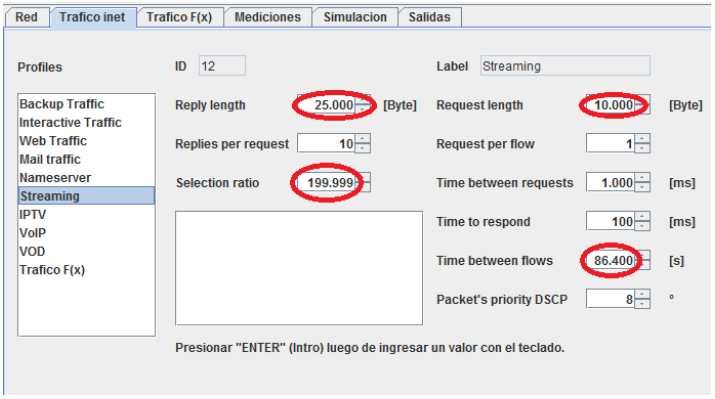


Figura 6.24: Configuración de parámetros de tráfico inet para aplicación *Streaming*.

Es importante notar que el tiempo entre flujos se configura en 86.400 en ambas aplicaciones, para así obtener que cada un día se repitan los flujos.

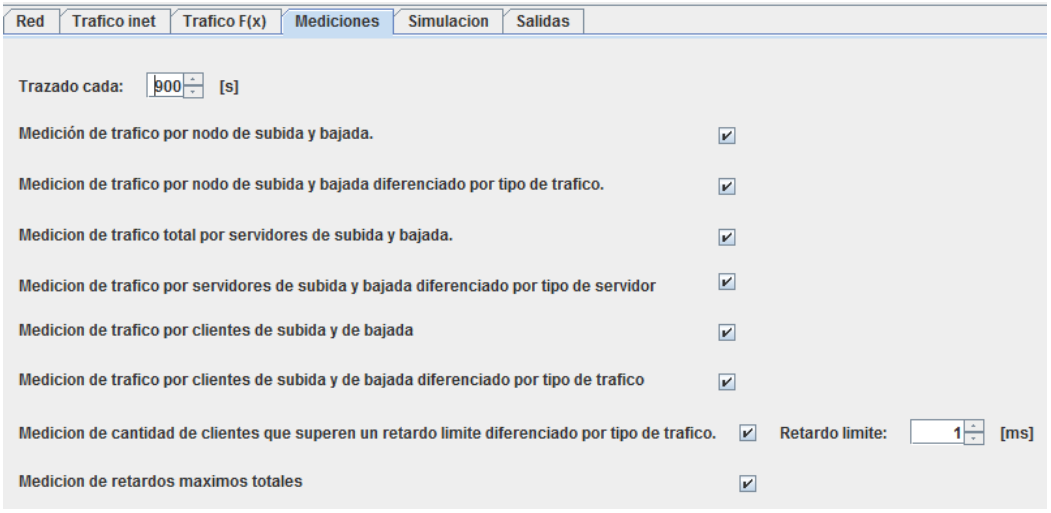


Figura 6.25: Configuración de paleta Mediciones.

En la paleta de mediciones (Figura 6.25) se eligen todas las mediciones disponibles además de un trazado cada 900 segundos. Posteriormente, en este manual, se revisará como quedan guardadas cada una de las mediciones hechas.

6.4.3 Simulación

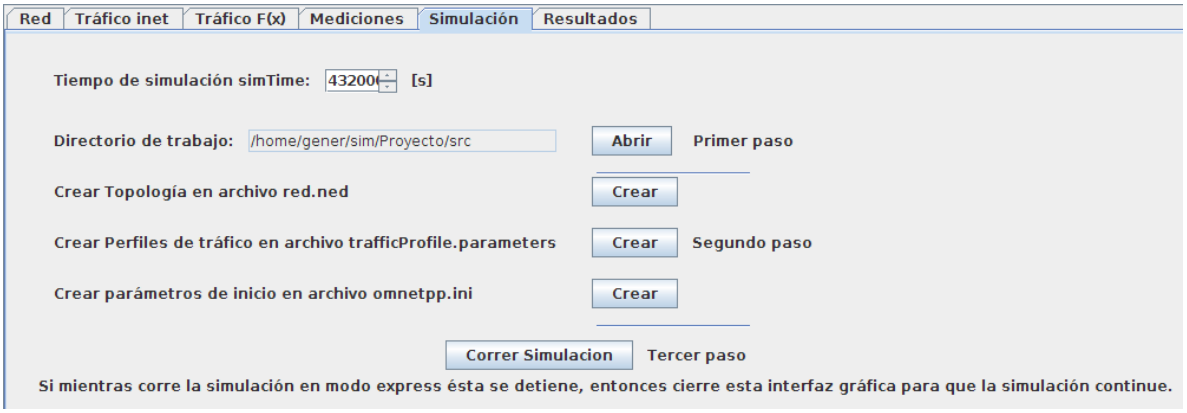


Figura 6.26: Paleta de simulación.

En la paleta de **Simulación** (Figura 6.26) se elige el tiempo de simulación de 5 días lo que es equivalente a 432.000 [s] luego, en el primer paso, se abre el directorio del ejecutable que por defecto ya viene configurado por lo que no es necesario cambiarlo. No es necesario cambiar la dirección pues ya está configurada correctamente en la carpeta donde se ejecutará la simulación. En el segundo paso se crean los tres archivos necesarios y en el tercer paso se presiona el botón Correr Simulación. Esto puede tardar unos segundos en abrirse y minutos o incluso horas en simular. Para el ejemplo de uso presente se utilizó un computador CPU Mobile DualCore Intel Core 2 Duo T5450, 1666 MHz (10 x 167) con 4GB de memoria RAM y la simulación duró aproximadamente 30 horas corriendo en modo express.

Para simular es necesario presionar el botón que se muestra en la Figura 6.27 el cual permite simular en modo express (el más rápido).

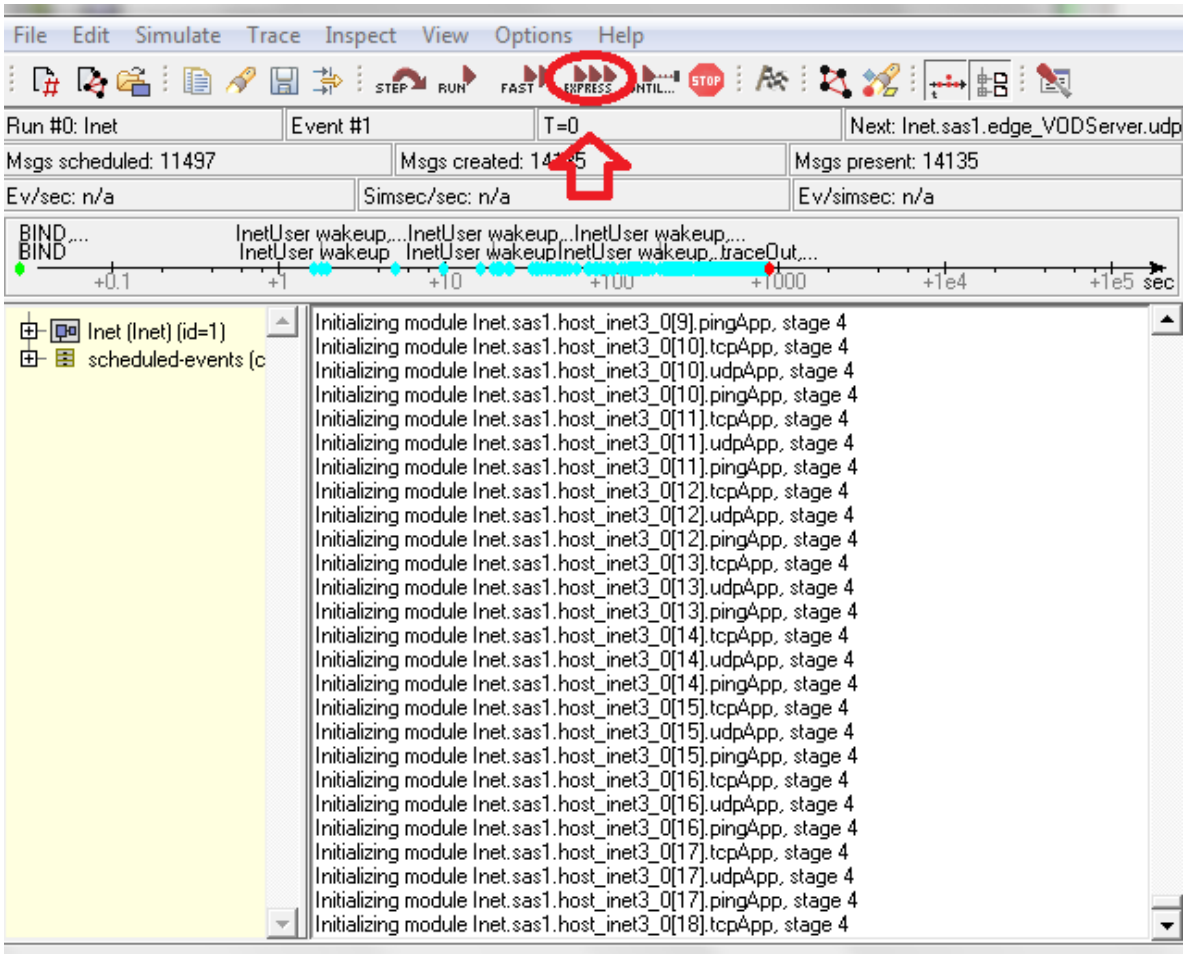


Figura 6.27: Selección de botón express en ventana de simulación.

Si la simulación se detiene sin razón aparente o si no quiere que esto suceda, entonces cierre la interfaz gráfica Genersim (Ventana java donde se configuran los parámetros). Esto permite la continuación de la simulación y si todo sale bien, ésta concluye con el mensaje que se aprecia en la Figura 6.28 y donde es necesario presionar el botón en OK.

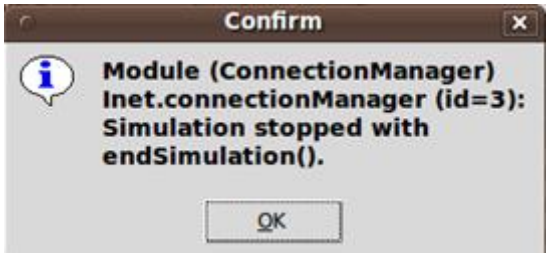


Figura 6.28: Ventana de finalización de una simulación.

6.4.4 Resultados

Una vez realizada la simulación cierre las ventanas asociadas a ésta y proceda a ver los resultados. Para esto presionar el botón **Ver resultados en OMNeT++** de la interfaz gráfica Genersim o presionando el ícono **OMNeT++ 4.0** del escritorio de la máquina Virtual.
Una vez en OMNeT++ 4.0, para poder observar los resultados hay que, en el explorador del proyecto, ir a la carpeta **Proyecto -> src** y luego hacer doble clic sobre el archivo **General.anf**

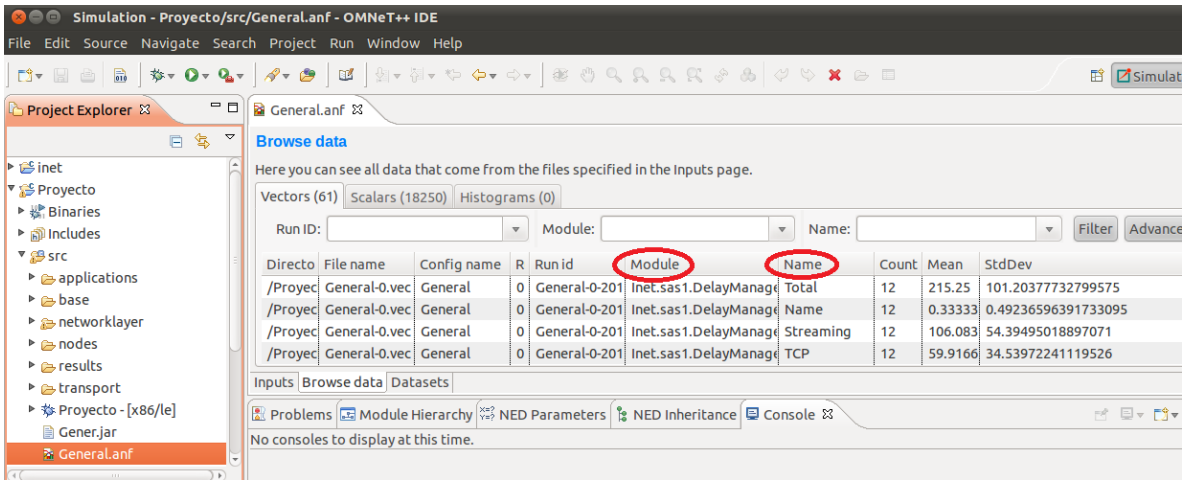


Figura 6.29: Archivo General.anf y destaque de columnas de identificación relevantes

En la parte derecha de la Figura 6.29 se visualiza el contenido del archivo General.anf donde se ven todas las mediciones realizadas durante la simulación las cuales pueden ser tanto vectores como valores escalares. En los vectores se guardan las mediciones configuradas previamente desde la interfaz con su trazado de tiempo (abscisas) y variable a medir (ordenadas) correspondiente. La columna **Module** que se observa en la Figura 6.29 más la columna **Name** contienen la información para dilucidar la medición asociada, además y a continuación, se muestra la Tabla 6.1 explicativa de las asociaciones entre la columna **Module**, la columna **Name** y la medición configurada desde la interfaz. Donde **X** e **Y** corresponden a las diferentes aplicaciones cuyas mediciones se tienen y **N** corresponde a las enumeraciones de todos los nodos de la simulación, más específicamente:

X ∈ {TCP, Name, Streaming, IPTV, VoIP, VOD, Fx}
Y ∈ { Backup, Interactive, Web, Mail, Name, Streaming, VoIP, VOD, Fx}
N ∈ {0, ..., número de nodos total - 1}

Module	Name	Descripción
Inet.sas1.DelayManager	Total	Cantidad total de clientes superando retardo límite.
Inet.sas1.DelayManager	**X**	Cantidad de clientes que han superado retardo límite con tráfico **X.**
Inet.sas1.DelayManager	Maximos	Registra el retardo máximo en cada intervalo de tiempo, medido en [s].
Inet.sas1.serverTraffic	Downstream	Registra tráfico de bajada total desde los servidores.
Inet.sas1.serverTraffic	Upstream	Registra el tráfico de subida total hacia los servidores.
Inet.sas1.edge_**Y**Server.networkLayer.ip	Downstream	Registra el tráfico de bajada desde el servidor **Y** en [Bytes].
Inet.sas1.edge_**Y**Server.networkLayer.ip	Upstream	Registra el tráfico de subida en el servidor **Y** en [Bytes].
Inet.sas1.RouterIP_**N**.clientTraffic	Downstream	Registra el tráfico de bajada total hacia los clients.
Inet.sas1.RouterIP_**N**.clientTraffic	Upstream	Registra el tráfico de subida total desde los clients.
Inet.sas1.RouterIP_**N**.clientTraffic	**X** up	Registra el tráfico de subida desde los clientes para aplicaciones **X**.
Inet.sas1.RouterIP_**N**.clientTraffic	**X** down	Registra el tráfico de bajada hacia los clientes para aplicaciones **X**.
Inet.sas1.RouterIP_**N**.networkLayer.ip	Downstream	Registra el tráfico total proveniente del gateway y que pasa por el nodo **N** en [Bytes].
Inet.sas1.RouterIP_**N**.networkLayer.ip	Upstream	Registra el tráfico total que va desde el Nodo **N** al gateway en [Bytes].

Module	Name	Descripción
Inet.sas1.RouterIP_**N**.networkLayer.ip	**X** up	Registra el tráfico de tipo **X** que viene desde los clientes y pasa por el nodo **N** en [Bytes].
Inet.sas1.RouterIP_**N**.networkLayer.ip	**X** down	Registra el tráfico de tipo **X** que viene desde el gateway y pasa por el nodo **N** en [Bytes].

Tabla 6.1: Descripción de todos los vectores de resultados.

A continuación se muestran los vectores graficados para cada tipo de medición para el ejemplo en revisión.

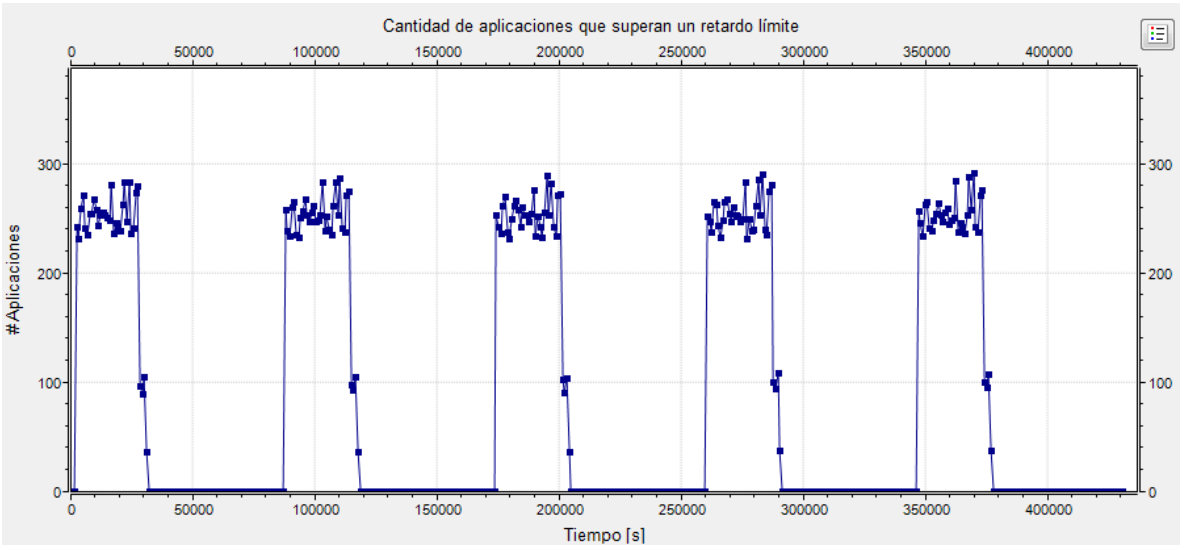


Figura 6.30: Cantidad de aplicaciones que superan un retardo límite (1 [ms])

La Figura 6.30 muestra la cantidad de aplicaciones que superaron un retardo límite mínimo, en este caso de 1 [ms], durante cada intervalo de tiempo trazado cada 900 segundos.

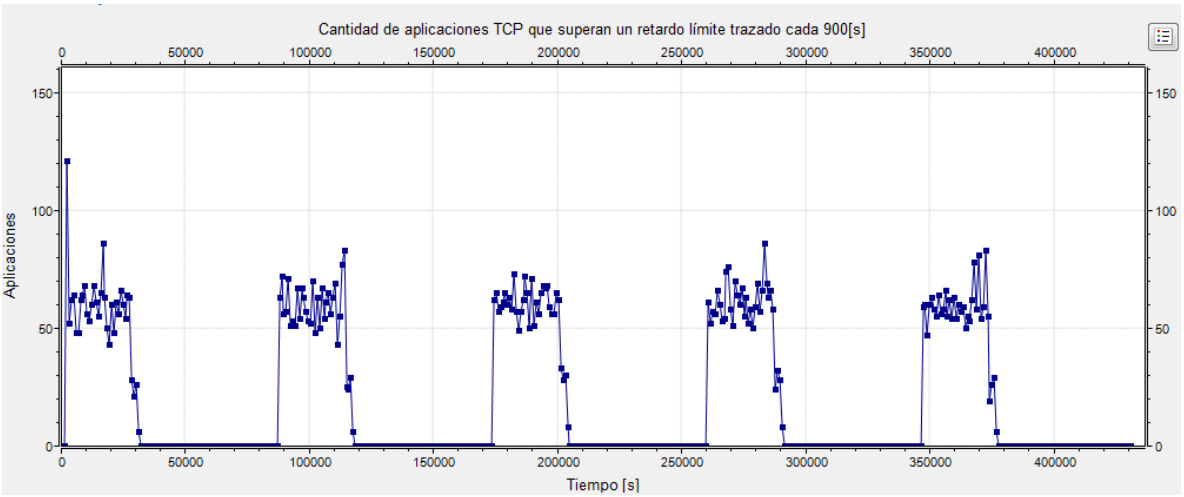


Figura 6.31: Cantidad de aplicaciones TCP que superan un retardo límite.

La Figura 6.31 muestra la cantidad de aplicaciones (de los host) que superaron un retardo mínimo ocupando los tipos de tráfico TCP durante el intervalo de tiempo trazado cada 900 segundos.

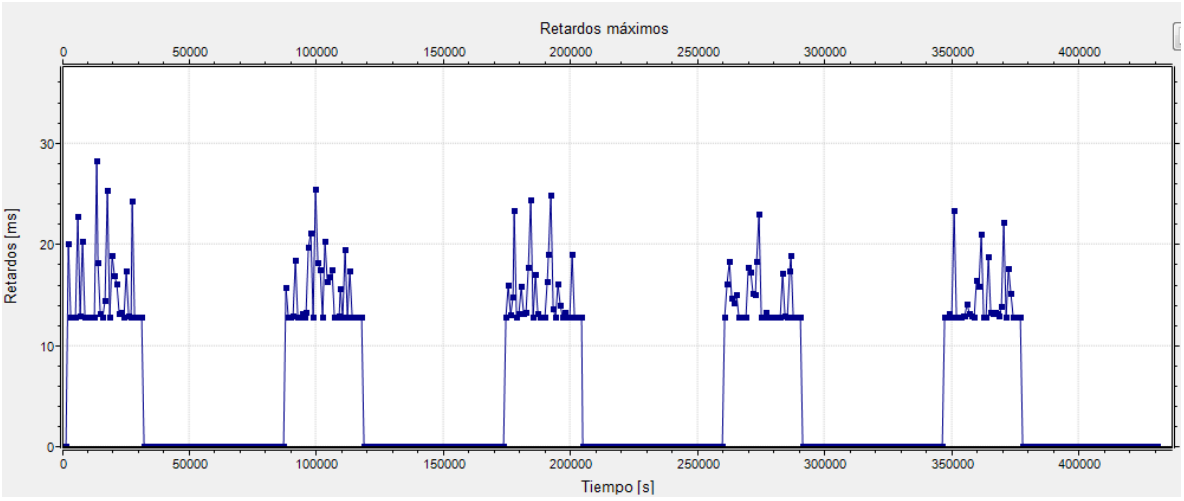


Figura 6.32: Retardos máximos en cada intervalo de 900[s]

La Figura 6.32 muestra el gráfico del valor del retardo máximo medido en milisegundos que sucede en cada intervalo de tiempo trazado en 900 segundos.

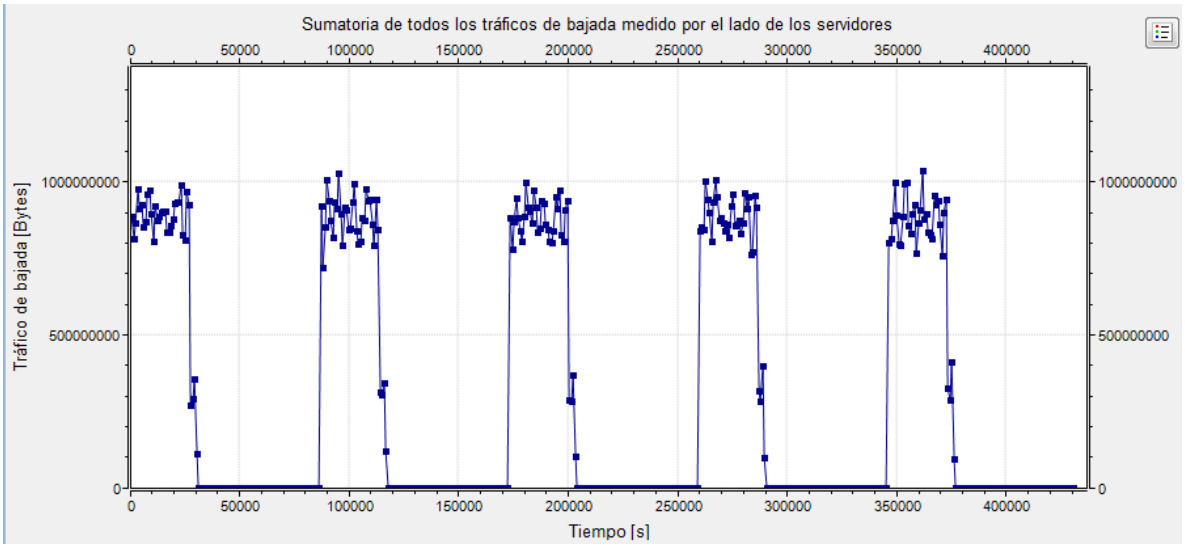


Figura 6.33: Sumatoria de todos los tráficos de bajada medido por el lado de los servidores.

La Figura 6.33 muestra la cantidad de tráfico de bajada medida en Bytes para la suma desde todos los servidores.

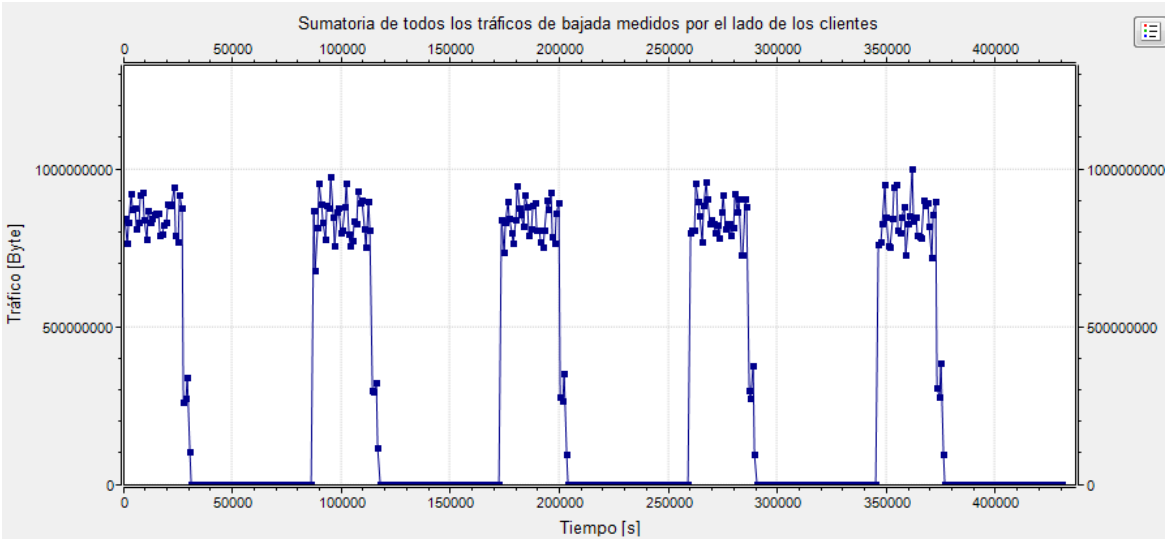


Figura 6.34: Sumatoria de todos los tráficos de bajada medidos por el lado de los clientes.

La Figura 6.34 muestra la cantidad de tráfico de bajada medido en Bytes para la suma de todos los tráficos en el lado de los clientes. Notar que la forma es similar a la de la Figura 6.33 y serían iguales también en magnitud si no hubiese pérdida de paquetes pero al comparar ambas notamos que existe una diferencia entre lo que se envía y lo que llega para el tráfico de bajada como se muestra en la Figura 6.36.

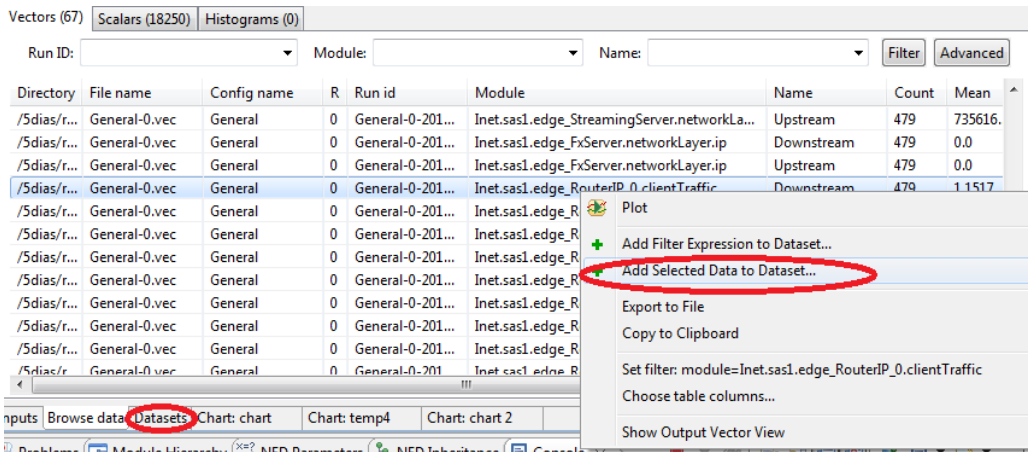


Figura 6.35: Selección de un vector para agregarlo a un dataset.

Para juntar más de una curva en un mismo gráfico se selecciona cada vector y se agrega a un Dataset luego en la pestaña Datasets se agrega un chart y al hacerle doble clic aparece el gráfico con todos los datos agregados.

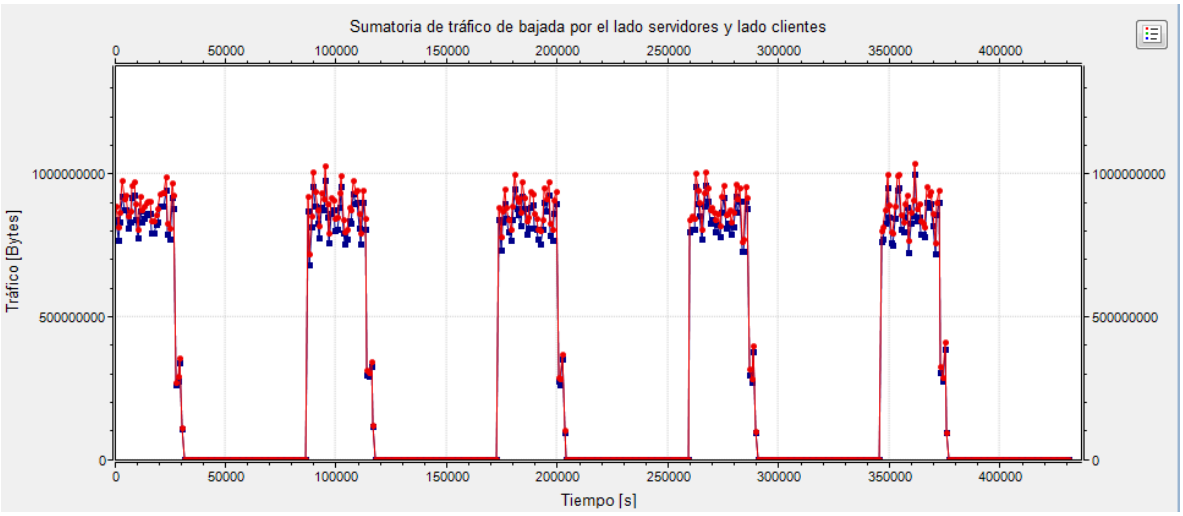


Figura 6.36: Sumatoria del tráfico de bajada lado servidores y lado clientes.

La Figura 6.36 muestra la cantidad de tráfico de bajada por el lado de los servidores (círculos en rojo) y por el lado de los clientes (cuadrados en azul) mostrando que existe una pérdida de paquetes, dado que el tráfico de bajada medido en el lado de los servidores es siempre superior al medido en el lado de los clientes.

Ahora se agregará los datos de subida a ver si existe diferencia entre estos.

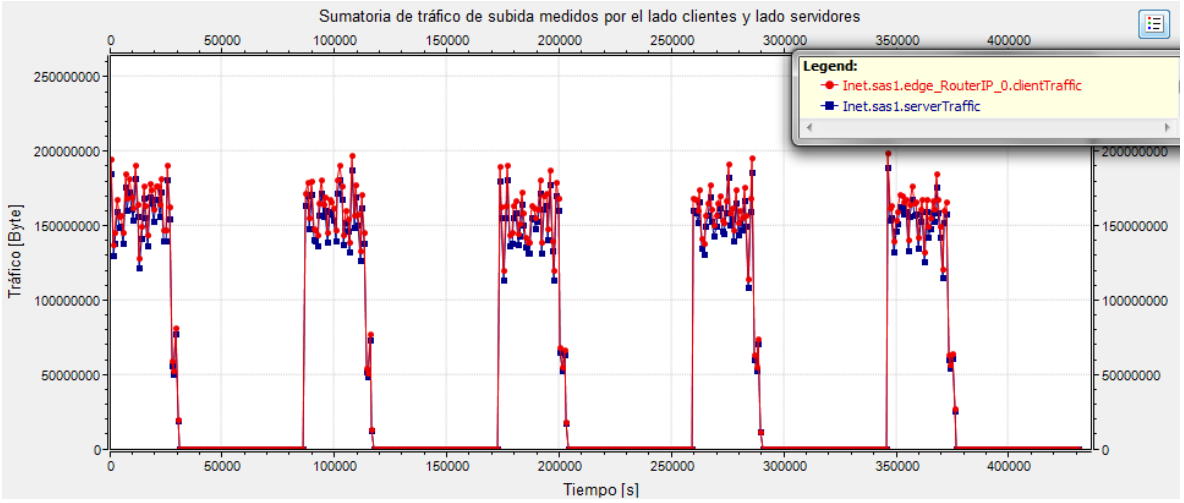


Figura 6.37: Sumatoria del tráfico de subida lado servidores y lado clientes.

Como se aprecia en la Figura 6.37 los tráficos de subida (lado clientes en círculos rojos y lado servidores en cuadrados azules) tienen una diferencia por lo que sí existe una apreciable pérdida de paquetes en éstos (Configurada al 5%).

Juntando los datos de subida total y de bajada total ambos por el lado de los clientes y por el lado de los servidores se obtiene el gráfico de la Figura 6.38.

Figura 6.38: Sumatorias de tráfico de bajada y de subida para lado servidores y lado clientes.

De esta forma se pueden generar gráficos con diferentes datos.

Finalmente los datos de un vector o de un dataset se pueden exportar a archivos .csv al hacer clic con el botón derecho sobré aquél y seleccionar la opción Export to file. Esto permite su manejo con, por ejemplo, Excel de Microsoft Office o una herramienta similar mediante la importación de datos de texto csv.

6.5 Ejemplo de uso con tráfico aleatorio

De manera similar a la sección 6.4, la configuración de parámetros se realiza de igual forma, salvo que en vez de elegir, en la paleta Red, clientes tipo inet se eligen clientes tipo F(x), tal como se muestra en la figura 6.39.

Figura 6.39: Configuración de cliente F(x).

Además, y a diferencia del caso anterior, se debe configurar la paleta F(x) hecha específicamente para configurar este tipo de tráfico donde se elige el nodo a editar, el grupo de tipo de clientes y la distribución matemática elegida.

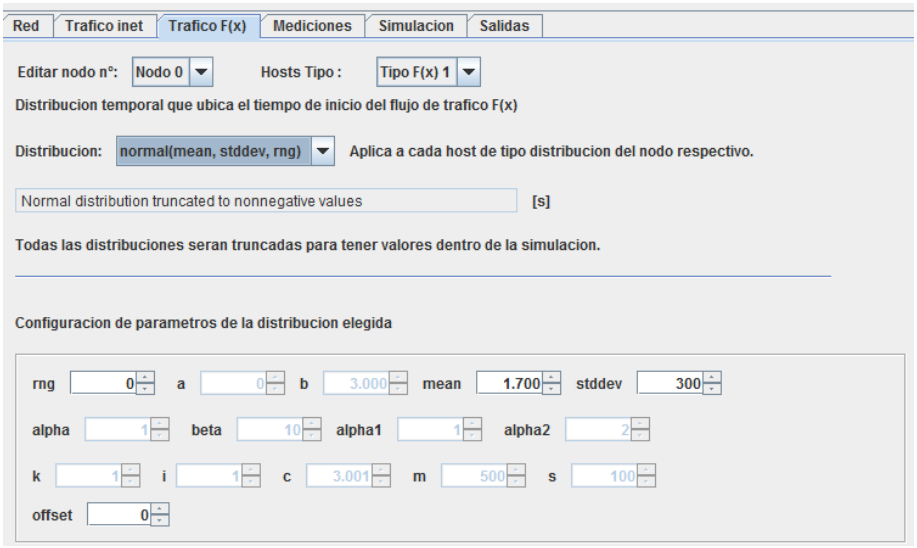


Figura 6.40: Paleta de Trafico F(x)

El resto de las configuraciones es idéntico a las realizadas en el punto 6.4 del presente documento cambiando los parámetros según lo que la simulación necesite.

6.6 Rangos de operación de los parámetros de Genersim

Paleta de red-parámetros generales	Valor mínimo	Valor máximo	Unidad	Comentario
N° de nodos de acceso	1	100	-	-
Puerta de enlace -> Núcleo	0	sin límite práctico	Mbps	-
Núcleo -> Puerta de enlace	0	sin límite práctico	Mbps	-
Núcleo -> Núcleo	0	sin límite práctico	Mbps	-

Parámetros del nodo que se está editando	Valor mínimo	Valor máximo	Unidad	Comentario
Retardo proc.	0	2.000.000.000.000	[us]	-
Cap. Cola uplink out	0	2.147.483.647	[Frames]	-
Cap. Cola entrada	0	sin límite práctico	[MB]	-
Acceso -> Puerta de enlace	0	sin límite práctico	[Mbps]	-
Puerta de enlace -> Acceso	0	sin límite práctico	[Mbps]	-
Conexiones IPTV Simultaneas	0	50.000	conexiones	Conexiones por nodo
Pérdida de paquetes	0	100	[%]	-

Parámetros Host:	Valor mínimo	Valor máximo	Unidad	Comentario
n° Host	0	10.000	clientes	Probado con 10.000 clientes totales en 1 nodo.
Factor tiempo	0	100.000	veces	Junto con **Time to respond** y **Time between request** no deben superar los 2.000.000 [s]
Factor data	0	100.000	veces	-
Host -> Acceso	0	sin límite práctico	[Mbps]	-
Acceso -> Host	0	sin límite práctico	[Mbps]	-
Cap.	0	2.147.483.647	[frames]	-
nApps		10	-	Si un cliente se comunica con un mismo servidor con más de una aplicación simultáneamente puede producir más tráfico de respuesta de lo esperado,

				por esto el límite de 10.
Parámetros Host:	**Valor mínimo**	**Valor máximo**	**Unidad**	**Comentario**
a[s] y b [s]	0	2.000.000	[s]	-

Tráfico Inet:	**Valor mínimo**	**Valor máximo**	**Unidad**	**Comentario**
Reply length	0	200.000.000	[Bytes]	probada con un **Replies per request** de 1 y **Factor data** de 1
Replies per request	0	20.000.000	-	Probada con un **Time to respond** de 100 [ms] y **Factor tiempo** de 1
Selection ratio	0	sin límite práctico	-	
Request length	0	200.000.000	[Bytes]	probada con un **Factor data** de 1
Request per flow	0	20.000.000	-	Probada con un **Time between request** de 100 [ms], **Factor tiempo** de 1 y **Replies per request** de 1
Time between requests	0	2.000.000.000	[ms]	
Time to respond	0	2.000.000.000	[ms]	
Time Between flows	0	2.000.000	[s]	
Packet's priority DSCP	1	8	° prioridad	

Paleta de Trafico F(x)	**Valor mínimo**	**Valor máximo**	**Unidad**	**Comentario**
RNG	0	100	-	

Trabajar en los rangos de operación máximo o superiores puede requerir de más memoria RAM asociada a la máquina virtual, puede provocar que la simulación se tome un tiempo en abrir además de posiblemente correr de forma lenta.

7 Anexo B: Cambios a librerías Inet y Rease

Cambios a las librerías Inet y ReaSE

1. En Inet

    1.1. Se agrega los mensajes genéricos de nivel de aplicación.
    1.2. Se agrega el parámetro DSCP a los mensajes genéricos de nivel de aplicación.
    1.3. Se modifica TCP y UDP para que pasen el valor DSCP del nivel de aplicación al nivel de transporte.
    1.4. Se quitan las grabaciones de vectores con información de colapso de las colas DropTailQueue y DropTailQoSQueue.

2. En ReaSE

    2.1. Creación de móduloss Cliente, RouterIP, FxServer, VODServer, VoIPServer.
    2.2. Se definen las cantidades de interfaces por cada tipo de cliente en el módulo RouterIP.
    2.3. Cambio a las interfaces de red de cada nodo para así identificar y tener diferentes funcionamientos dependiendo de si el nodo es un cliente, un router IP, o un servidor.
    2.4. Se implementó un límite a la cola de entrada de nivel IP de los nodos RouterIP para que si se supera éste límite los paquetes se pierdan.
    2.5. Se corrigieron pequeños errores de la clase TrafficProfileManager.cc
    2.6. Se agrega la capacidad de configurar y de obtener el valor DSCP modificando el archivo TransmissionConfig.h.
    2.7. Se implementó un mecanismo de pérdida de paquetes configurable a un cierto porcentaje para el nodo RouterIP.
    2.8. Se agregó la capa de aplicación y de transporte UDP a los nodos RouterIP para que estos se puedan comportar como servidores IPTV con un número configurable de conexiones simultáneas antes de bloquearse.
    2.9. Se creó la clase ProyectoDSCPClassifier para implementar la clasificación en las colas siguiendo las indicaciones DSCP con ocho posibles valores de prioridades. Esto en las colas DropTailQoSQueue.
    2.10. Se agrega la medición en vectores del tráfico por parte del nivel de red IP del nodo RouterIP diferenciado por tipo y midiendo tanto el tráfico de subida como el de bajada.
    2.11. Se crea la clase ClientTraffic la cual permite la grabación, en vectores, del tráfico diferenciado por tipo mediante métodos que se llaman desde el nivel de red de los nodos Cliente.
    2.12. Se agrega clientTraffic como submódulo del nodo RouterIP para así poder asociar las mediciones de tráfico de los clientes con el nodo RouterIP al cual están conectados.
    2.13. Se modifica el módulo IP de los nodos Cliente para de esta forma pueda enviar información del tráfico de subida y de bajada que pasa por él hacia el módulo clientTraffic del nodo RouterIP al cual pertenece.
    2.14. Se modifica el módulo IP de los servidores para que guarden en vectores el tráfico de subida y de bajada diferenciado por tipo.
    2.15. Se crea y agrega la clase ServerTraffic para poder tener guardado en vectores el tráfico agregado de todos los servidores tanto en la subida como en la bajada.
    2.16. Se modifica el nivel de red de los servidores para poder enviar información del tráfico a la clase ServerTraffic.
    2.17. Se crea y agrega el módulo DelayManager para poder guardar en vectores los retardos máximos que se dan en la simulación además de guardar diferenciado por tipo de aplicación la cantidad de aplicaciones que superan un retardo mínimo configurable.

2.18. Se modifica el módulo IP de los nodos Cliente para que estos envíen la información de los retardos medidos hacia el módulo DelayManager.
2.19. Se reutiliza el módulo inetUser para generar tráfico en el nodo Cliente.
2.20. Se modifica inetUser para poder iniciar varias aplicaciones en vez de una sola, de forma independiente y pudiéndose configurar la cantidad externamente con la interfaz gráfica Gener.
2.21. Se modificó inetUser para poder tener desfase configurable en los tiempos de inicio de cada simulación.
2.22. Se modifica inetUser para poder configurar un factor de tamaño y otro de tiempo en las peticiones como en las respuestas en cada grupo de usuarios.
2.23. Se modifica inetUser para poder tener obligatoriamente una aplicación de tipo Fx cuando el Cliente es de tipo Fx lo que se configura desde la interfaz gráfica Gener.
2.24. Se modifica ConnectionManager.cc agregando un método para poder obligar a buscar la dirección IP para el servidor Fx y también se modifica TrafficProfileManager.cc para tener el perfil de tráfico deseado.
2.25. Se modifica inetUser para asociar el perfil de tráfico de IPTV con la exclusión de la selección del mismo si es que el nodo Cliente, al cual pertenece, forma parte de un nodo sin la habilitación de IPTV.
2.26. Se cambiaron la capa de transporte de todos los Nodos servidores de tipo TCP dejando de usar el original de ReaSE para pasar a utilizar el original de Inet dado que se presentaban problemas de conexión.

Código de los cambios en el código fuente de Inet y ReaSE

1.1

Archivo GenericApplicationMessage.ned

```
packet GenericApplicationMessage
{
    int replyLength; //bytes
    int replyPerRequest; //how often respond to incoming request
    double timeToRespond; //reply after this many seconds
    bool last; //this indicates the last packet --> close socket
    int packetNumber; //increasing number for each packet
    int app; // application ID
}
```

1.2

Se agrega el siguiente código a GenericApplicationMessage.msg:19

```
    int DSCP; // for DiffServ code point usage
```

1.3

Archivos modificados agregando las siguientes líneas de código.

TCPConnection.h:321

```
int DSCP;
```

TCPConnection.h:27

```
include "GenericApplicationMessage_m.h"
```

TCPConnectionBase.cc:167

```
DSCP = 0;
```

TCPConnectionBase.cc:188

```
DSCP = 0
```

TCPConnectionBase.cc:359

```
if (dynamic_cast<GenericApplicationMessage*> (msg)){
DSCP = (check_and_cast<GenericApplicationMessage*> (msg))->getDSCP();
  }
```

TCPConnectionUtil:227

```
controlInfo->setDiffServCodePoint(DSCP);
```

UDP.h:31

```
#include "GenericApplicationMessage_m.h"
```

UDP:92

```
DSCP = 0;
```

UDP.cc:210

```
if (dynamic_cast<GenericApplicationMessage*> (msg)){
                DSCP = (check_and_cast<GenericApplicationMessage*> (msg))->getDSCP();
```

```
  }
```

UDP.cc:509

```
ipControlInfo->setDiffServCodePoint(DSCP);
```

UDP.h:76

```
int DSCP; //protected
```

Además se agrega GenericApplicationMessage.msg en *inet/src/application/generic*

1.4

Se borran las líneas siguientes del archivo DropTailQueue.cc

DropTailQueue.cc:30

```
qlenVec.setName("queue length");
```

DropTailQueue.cc:31

```
dropVec.setName("drops");
```

DropTailQueue.cc:45

```
dropVec.record(1);
```

DropTailQueue.cc:51

```
qlenVec.record(queue.length());
```

DropTailQueue:64

```
qlenVec.record(queue.length());
```

2.1

Archivo de Módulo Cliente: Cliente.ned

Se diferencia de los demás módulos agregando los siguientes sub-módulos.

```
inetUser: InetUser {
```

```
            parameters:
                lengthFactor = lengthFactor;
                timeFactor = timeFactor;
                offset = offset;
                profileId = profileId;
                @display("p=60,390;i=abstract/person");

        }
        networkLayer: NetworkLayerClient {
            parameters:
                proxyARP = false;
                @display("p=248,247;i=block/fork;q=queue");

            gates:
                ifIn[sizeof(pppg)+sizeof(ethg)];
                ifOut[sizeof(pppg)+sizeof(ethg)];
        }
```

Se reemplaza código de archivo de módulo RouterIP: RouterIP.ned

```
networkLayer: NetworkLayer_hack {
            parameters:
                @display("p=200,141;q=queue");
                tracingOn = true;
                spoofingOn = true;
```

Los servidores son modificados con el siguiente código.

```
networkLayer: NetworkLayerServer {
            parameters:
                proxyARP = false;
                @display("p=248,247;i=block/fork;q=queue");

            gates:
                ifIn[sizeof(pppg)+sizeof(ethg)];
```

```
                ifOut[sizeof(pppg)+sizeof(ethg)];
        }
```

2.2

Archivo RouterIP.ned:112

```
connections allowunconnected:
        // connections to network outside
        for i=0..sizeof(pppg)-1 {
            pppg[i] <--> ppp[i].phys;
            ppp[i].netwOut --> networkLayer.ifIn[i];
            ppp[i].netwIn <-- networkLayer.ifOut[i];
        }

        for i=0..sizeof(inet1_pppg)-1 {
            inet1_pppg[i] <--> inet1_ppp[i].phys;
            inet1_ppp[i].netwOut --> networkLayer.ifIn[sizeof(pppg)+i];
            inet1_ppp[i].netwIn <-- networkLayer.ifOut[sizeof(pppg)+i];
        }

        for i=0..sizeof(inet2_pppg)-1 {
            inet2_pppg[i] <--> inet2_ppp[i].phys;
            inet2_ppp[i].netwOut -->
networkLayer.ifIn[sizeof(pppg)+sizeof(inet1_pppg)+i];
            inet2_ppp[i].netwIn <--
networkLayer.ifOut[sizeof(pppg)+sizeof(inet1_pppg)+i];
        }

        for i=0..sizeof(inet3_pppg)-1 {
            inet3_pppg[i] <--> inet3_ppp[i].phys;
            inet3_ppp[i].netwOut -->
networkLayer.ifIn[sizeof(pppg)+sizeof(inet1_pppg)+sizeof(inet2_pppg)+i];
            inet3_ppp[i].netwIn <--
networkLayer.ifOut[sizeof(pppg)+sizeof(inet1_pppg)+sizeof(inet2_pppg)+i];
        }
```

```
        for i=0..sizeof(inet4_pppg)-1 {
            inet4_pppg[i] <--> inet4_ppp[i].phys;
            inet4_ppp[i].netwOut -->
networkLayer.ifIn[sizeof(pppg)+sizeof(inet1_pppg)+sizeof(inet2_pppg)+sizeof(in
et3_pppg)+i];
            inet4_ppp[i].netwIn <--
networkLayer.ifOut[sizeof(pppg)+sizeof(inet1_pppg)+sizeof(inet2_pppg)+sizeof(i
net3_pppg)+i];
        }
        for i=0..sizeof(fx1_pppg)-1 {
            fx1_pppg[i] <--> fx1_ppp[i].phys;
            fx1_ppp[i].netwOut -->
networkLayer.ifIn[sizeof(pppg)+sizeof(inet1_pppg)+sizeof(inet2_pppg)+sizeof(in
et3_pppg)+sizeof(inet4_pppg)+i];
            fx1_ppp[i].netwIn <--
networkLayer.ifOut[sizeof(pppg)+sizeof(inet1_pppg)+sizeof(inet2_pppg)+sizeof(i
net3_pppg)+sizeof(inet4_pppg)+i];
        }

        for i=0..sizeof(fx2_pppg)-1 {
            fx2_pppg[i] <--> fx2_ppp[i].phys;
            fx2_ppp[i].netwOut -->
networkLayer.ifIn[sizeof(pppg)+sizeof(inet1_pppg)+sizeof(inet2_pppg)+sizeof(in
et3_pppg)+sizeof(inet4_pppg)+sizeof(fx1_pppg)+i];
            fx2_ppp[i].netwIn <--
networkLayer.ifOut[sizeof(pppg)+sizeof(inet1_pppg)+sizeof(inet2_pppg)+sizeof(i
net3_pppg)+sizeof(inet4_pppg)+sizeof(fx1_pppg)+i];
        }

        for i=0..sizeof(fx3_pppg)-1 {
            fx3_pppg[i] <--> fx3_ppp[i].phys;
            fx3_ppp[i].netwOut -->
networkLayer.ifIn[sizeof(pppg)+sizeof(inet1_pppg)+sizeof(inet2_pppg)+sizeof(in
et3_pppg)+sizeof(inet4_pppg)+sizeof(fx1_pppg)+sizeof(fx2_pppg)+i];
            fx3_ppp[i].netwIn <--
networkLayer.ifOut[sizeof(pppg)+sizeof(inet1_pppg)+sizeof(inet2_pppg)+sizeof(i
net3_pppg)+sizeof(inet4_pppg)+sizeof(fx1_pppg)+sizeof(fx2_pppg)+i];
        }
```

```
        for i=0..sizeof(ethg)-1 {
            ethg[i] <--> eth[i].phys;
            eth[i].netwOut -->
networkLayer.ifIn[sizeof(pppg)+sizeof(inet1_pppg)+sizeof(inet2_pppg)+sizeof(in
et3_pppg)+sizeof(inet4_pppg)+sizeof(fx1_pppg)+sizeof(fx2_pppg)+sizeof(fx3_pppg
)+i];
            eth[i].netwIn <--
networkLayer.ifOut[sizeof(pppg)+sizeof(inet1_pppg)+sizeof(inet2_pppg)+sizeof(i
net3_pppg)+sizeof(inet4_pppg)+sizeof(fx1_pppg)+sizeof(fx2_pppg)+sizeof(fx3_ppp
g)+i];
        }
```

2.3

Para servidores en archivo NetworkLayerServer.ned:

```
ip: IPserver {
            parameters:
                timeToLive = 125;
                multicastTimeToLive = 32;
                fragmentTimeout = 60s;
                protocolMapping =
"6:0,17:1,1:2,2:3,46:4,89:5,132:6,254:7,135:7";
                @display("p=85,95;i=block/routing;q=queue");
            gates:
                transportIn[8];
                transportOut[8];
                queueIn[sizeof(ifIn)];
        }
```

Para RouterIP en archive NetworkLayer_hack.ned:

```
ip: IP_hack {
            parameters:
                timeToLive = 125;
                multicastTimeToLive = 32;
```

```
                fragmentTimeout = 60s;
                protocolMapping =
"6:0,17:1,1:2,2:3,46:4,89:5,132:6,254:7,135:7";

                @display("p=85,95;i=block/routing;q=queue");
            gates:
                transportIn[8];
                transportOut[8];
                queueIn[sizeof(ifIn)];
        }
```

Para archivo Cliente.ned:

```
ip: IPclient {
            parameters:
                timeToLive = 125;
                multicastTimeToLive = 32;
                fragmentTimeout = 60s;
                protocolMapping =
"6:0,17:1,1:2,2:3,46:4,89:5,132:6,254:7,135:7";
                @display("p=85,95;i=block/routing;q=queue");
            gates:
                transportIn[8];
                transportOut[8];
                queueIn[sizeof(ifIn)];
        }
```

2.4

En módulo IP_hack.ned:

```
double limitBytes;
```

En archivo IP_hack.cc:537

```
void IP_hack::arrival(cPacket *msg)
{
      if(queue.getByteLength()+msg->getByteLength()<limit){
```

```
            queue.insert(msg);
      }
      else{
            delete msg;
            packetdrop=packetdrop+1;
      }

}
```

2.5

Archivo TrafficProfileManager.cc

```
if (status == 1)
            {
                  if (strstr(buf, "<Id>"))
                        t->profileID = atoi(GET_OFFSET(buf));
                  else if (strstr(buf, "<Label>"))
                  {
                        t->label = "";
                        t->label += GET_OFFSET(buf);
                        t->label += "\0";
                  }
                  else if (strstr(buf, "<RequestLength>"))
                        t->requestLength = atoi(GET_OFFSET(buf));
                  else if (strstr(buf, "<RequestsPerSession>"))
                        t->requestsPerSession = atoi(GET_OFFSET(buf));
                  else if (strstr(buf, "<ReplyLength>"))
                        t->replyLength = atoi(GET_OFFSET(buf));
                  else if (strstr(buf, "<ReplyPerRequest>"))
                        t->replyPerRequest = atoi(GET_OFFSET(buf));
                  else if (strstr(buf, "<TimeBetweenRequests>"))
                        t->timeBetweenRequests = atof(GET_OFFSET(buf));
                  else if (strstr(buf, "<TimeToRespond>"))
                        t->timeToRespond = atof(GET_OFFSET(buf));
                  else if (strstr(buf, "<TimeBetweenSessions>"))
                        t->timeBetweenSessions = atof(GET_OFFSET(buf));
                  else if (strstr(buf, "<Ratio>"))
                        t->probability = atof(GET_OFFSET(buf));
```

```
                else if (strstr(buf, "<WANRatio>"))
                        t->WANprob = atof(GET_OFFSET(buf));
                else if (strstr(buf, "<DSCP>"))
                        t->DSCP = atoi(GET_OFFSET(buf));
                else if (strstr(buf, "<Port>"))
                        t->ownPort = atoi(GET_OFFSET(buf));
                else if (strstr(buf, "<HopLimit>"))
                        t->hopLimit = atoi(GET_OFFSET(buf));
                else if (strstr(buf, "</Profile>"))
```

2.6

Archivo TransmissionConfig.h

TransmissionConfig.h:74

```
DSCP = 0; //Best effort by default
```

TransmissionConfig.h:95

```
this->DSCP = o.DSCP;
```

TransmissionConfig.h:192

```
int getDSCP(){
            return DSCP;
      }
```

2.7

```
if (msg->getArrivalGate()->isName("transportIn"))
      {
            if(rand()>PL*RAND_MAX){                    //Packet loss
                  handleMessageFromHL(msg);
            }else{
                  packetdrop=packetdrop+1;
                  delete msg;
            }
      }
      else if (dynamic_cast<ARPPacket *> (msg))
```

```
      {
            // dispatch ARP packets to ARP
            handleARP((ARPPacket *) msg);

      }
      else if (msg->getArrivalGate()->isName("queueIn"))
      {
            if(rand()>PL*RAND_MAX){                    //Packet loss
                  IPDatagram *dgram = check_and_cast<IPDatagram *> (msg);
                  handlePacketFromNetwork(dgram);
            }else{
                  packetdrop=packetdrop+1;
                  delete msg;
            }
      }
```

2.8

```
udpAppIPTV: GenericUDPApplication {
            parameters:
                @display("p=307,244");
                isServer = true;
                profileNumber = 120;
                port = 6246;
                @display("i=block/app;p=272,67");
        }
```

2.9

```
#include "ProyectoDSCPClassifier.h"

#include <omnetpp.h>
#include "BasicDSCPClassifier.h"
#include "IPDatagram.h"
#ifndef WITHOUT_IPv6
#include "IPv6Datagram.h"
#endif
```

```
Register_Class(ProyectoDSCPClassifier);

#define BEST_EFFORT 7

int ProyectoDSCPClassifier::getNumQueues()
{
    return 8;
}

int ProyectoDSCPClassifier::classifyPacket(cMessage *msg)
{
    if (dynamic_cast<IPDatagram *>(msg))
    {
        // IPv4 QoS: map DSCP to queue number
        IPDatagram *datagram = (IPDatagram *)msg;
        int dscp = datagram->getDiffServCodePoint();
        return classifyByDSCP(dscp);
    }
#ifndef WITHOUT_IPv6
    else if (dynamic_cast<IPv6Datagram *>(msg))
    {
        // IPv6 QoS: map Traffic Class to queue number
        IPv6Datagram *datagram = (IPv6Datagram *)msg;
        int dscp = datagram->getTrafficClass();
        return classifyByDSCP(dscp);
    }
#endif
    else
    {
        return BEST_EFFORT; // lowest priority ("best effort")
    }
}

int ProyectoDSCPClassifier::classifyByDSCP(int dscp)
{
    // DSCP is 6 bits, mask out all others
    dscp = (dscp & 0x3f);
```

```
    // DSCP format:
    //    xxxxx0: used by standards; see RFC 2474
    //    xxxxx1: experimental or local use
    // source: Stallings, High-Speed Networks, 2nd Ed, p494

    // all-zero DSCP maps to Best Effort (lowest priority)
    if (dscp==0)
        return BEST_EFFORT;

    // assume Best Effort service for experimental or local DSCP's too
    if (dscp & 1)
        return BEST_EFFORT;

    // from here on, we deal with non-zero standardized DSCP values only
    int upper3bits = (dscp & 0x3c) >> 3;
    //int lower3bits = (dscp & 0x07);  -- we'll ignore this

    // rfc 2474, section 4.2.2: at least two independently forwarded
classes of traffic have to be created
    if (upper3bits == 7)
        return 0; // highest priority
    else if (upper3bits == 6)
        return 1;
    else if (upper3bits == 5)
        return 2;
    else if (upper3bits == 4)
        return 3;
    else if (upper3bits == 3)
        return 4;
    else if (upper3bits == 2)
        return 5;
    else if (upper3bits == 1)
        return 6;
    else
        return 7; // low priority (best effort)
}
```

2.10

```
void IP_hack::handlePacketFromNetwork(IPDatagram *datagram)
{
      // in case of tracing update packet count
      if (tracingOn)
      {

            if(datagram->getArrivalGateId()==3145728){//funciona con más
de un nodo también

                  downt=downt+datagram->getByteLength();//downstream

                  if(dynamic_cast<GenericApplicationMessage*>(datagram-
>getEncapsulatedMsg()->getEncapsulatedMsg())){



      switch(dynamic_cast<GenericApplicationMessage*>(datagram-
>getEncapsulatedMsg()->getEncapsulatedMsg())->getApp()){
                              case 11:
                                    namedownt=namedownt+datagram-
>getByteLength();
                                    break;
                              case 12:
                                    streamdownt=streamdownt+datagram-
>getByteLength();
                                    break;
                              case 121:
                                    voipdownt=voipdownt+datagram-
>getByteLength();
                                    break;
                              case 122:
                                    voddownt=voddownt+datagram-
>getByteLength();
                                    break;
                              case 123:
                                    fxdownt=fxdownt+datagram-
>getByteLength();
                                    break;
                        }
```

```
                  }else{
                        switch (datagram->getTransportProtocol())
                        {
                              case IP_PROT_TCP:
                                    TCPdownt=TCPdownt+datagram-
>getByteLength();
                                    break;
                        }
                  }
            }else{

                  upt=upt+datagram->getByteLength();//upstream
                  if(dynamic_cast<GenericApplicationMessage*>(datagram-
>getEncapsulatedMsg()->getEncapsulatedMsg())){


      switch(dynamic_cast<GenericApplicationMessage*>(datagram-
>getEncapsulatedMsg()->getEncapsulatedMsg())->getApp()){
                        case 11:
                              nameupt=nameupt+datagram->getByteLength();
                              break;
                        case 12:
                              streamupt=streamupt+datagram-
>getByteLength();
                              break;
                        case 120:
                              iptvupt=iptvupt+datagram-
>getByteLength();// en el up va, en el down no va
                              break;
                        case 121:
                              voipupt=voipupt+datagram->getByteLength();
                              break;
                        case 122:
                              vodupt=vodupt+datagram->getByteLength();
                              break;
                        case 123:
                              fxupt=fxupt+datagram->getByteLength();
                              break;
                        }
```

```
                    }else{
                          switch (datagram->getTransportProtocol())
                          {
                                case IP_PROT_TCP:
                                      TCPupt=TCPupt+datagram-
>getByteLength();
                                      break;
                          }
                    }
             }
```

2.11

```
Define_Module(ClientTraffic);

void ClientTraffic::initialize()
{
      tracingInterval = par("tracingInterval");

      traceOn1 = par("traceOn1").boolValue();
      traceOn2 = par("traceOn2").boolValue();

      downt=0, upt=0,nameupt=0,namedownt=0,streamupt=0,streamdownt=0,TCPupt=0,
      TCPdownt=0,iptvupt=0,iptvdownt=0,voipupt=0,voipdownt=0,vodupt=0,voddownt
=0,fxupt=0,fxdownt=0;

      if(traceOn1)
      {
            down.setName("Downstream"), up.setName("Upstream");
      }
      if(traceOn2){
            nameup.setName("Nameserver up"),namedown.setName("Nameserver
down"),streamup.setName("Streaming up"),streamdown.setName("Streaming
down"),TCPup.setName("TCP up"),TCPdown.setName("TCP
down"),iptvup.setName("IPTV up"),iptvdown.setName("IPTV
down"),voipup.setName("VoIP up"),voipdown.setName("VoIP
```

```
down"),vodup.setName("VOD up"),voddown.setName("VOD down"),fxup.setName("Fx
up"),fxdown.setName("Fx down");
	}

	traceMsg = new cMessage("traceOut");
	scheduleAt(simTime() + tracingInterval, traceMsg);

}
void ClientTraffic::handleMessage(cMessage *msg){
	if (msg->isSelfMessage())
	{
		if (msg == traceMsg)
		{
			if (traceOn1)
			{
				down.recordWithTimestamp(simTime(), downt);
				up.recordWithTimestamp(simTime(), upt);
				downt=0, upt=0;
			}
			if(traceOn2){
				nameup.recordWithTimestamp(simTime(), nameupt);
				namedown.recordWithTimestamp(simTime(), namedownt);
				streamup.recordWithTimestamp(simTime(), streamupt);
				streamdown.recordWithTimestamp(simTime(),
streamdownt);
				TCPup.recordWithTimestamp(simTime(), TCPupt);
				TCPdown.recordWithTimestamp(simTime(), TCPdownt);
				iptvup.recordWithTimestamp(simTime(), iptvupt),
				iptvdown.recordWithTimestamp(simTime(), iptvdownt),
				voipup.recordWithTimestamp(simTime(), voipupt),
				voipdown.recordWithTimestamp(simTime(), voipdownt),
				vodup.recordWithTimestamp(simTime(), vodupt),
				voddown.recordWithTimestamp(simTime(), voddownt),
				fxup.recordWithTimestamp(simTime(), fxupt),
				fxdown.recordWithTimestamp(simTime(), fxdownt);

				nameupt=0,namedownt=0,streamupt=0,streamdownt=0,
TCPupt=0,
```

```
      TCPdownt=0,iptvupt=0,iptvdownt=0,voipupt=0,voipdownt=0,vodupt=0,voddownt
=0,fxupt=0,fxdownt=0;
                  }
            }
            scheduleAt(simTime() + tracingInterval, msg);
      }
}
void ClientTraffic::putDownBytes(int64 bytes,int tipo){

      downt+=bytes;

      switch(tipo){
                  case 11:
                        namedownt+=bytes;
                        break;
                  case 12:
                        streamdownt+=bytes;
                        break;
                  case 210:
                        TCPdownt+=bytes;
                        break;
                  case 120:
                        iptvdownt+=bytes;
                        break;
                  case 121:
                        voipdownt+=bytes;
                        break;
                  case 122:
                        voddownt+=bytes;
                        break;
                  case 123:
                        fxdownt+=bytes;
                        break;
                  default:
                        break;
            }
}
void ClientTraffic::putUpBytes(int64 bytes,int tipo){
```

```
      upt+=bytes;

      switch(tipo){
                  case 11:
                        nameupt+=bytes;
                        break;
                  case 12:
                        streamupt+=bytes;
                        break;
                  case 210:
                        TCPupt+=bytes;
                        break;
                  case 120:
                        iptvupt+=bytes;
                        break;
                  case 121:
                        voipupt+=bytes;
                        break;
                  case 122:
                        vodupt+=bytes;
                        break;
                  case 123:
                        fxupt+=bytes;
                        break;
                  default:
                        break;
            }
}
```

2.12

```
  submodules:

        clientTraffic: ClientTraffic{
        }
        namTrace: NAMTraceWriter {
            parameters:
                namid = -1; // auto
```

```
                @display("p=330,60");
        }
```

2.13

Archivo IPclient.cc:119

```
if (tracingOn)
                  {
                        ((DelayManager *)(this->getParentModule()-
>getParentModule()->getParentModule()->getSubmodule("DelayManager")))-
>putDelay(dtotal,0);
                        ((DelayManager *)(this->getParentModule()-
>getParentModule()->getParentModule()->getSubmodule("DelayManager")))-
>putDelay(dname,11);
                        ((DelayManager *)(this->getParentModule()-
>getParentModule()->getParentModule()->getSubmodule("DelayManager")))-
>putDelay(dstream,12);
                        ((DelayManager *)(this->getParentModule()-
>getParentModule()->getParentModule()->getSubmodule("DelayManager")))-
>putDelay(dTCP,210);
                        ((DelayManager *)(this->getParentModule()-
>getParentModule()->getParentModule()->getSubmodule("DelayManager")))-
>putDelay(diptv,120);
                        ((DelayManager *)(this->getParentModule()-
>getParentModule()->getParentModule()->getSubmodule("DelayManager")))-
>putDelay(dvoip,121);
                        ((DelayManager *)(this->getParentModule()-
>getParentModule()->getParentModule()->getSubmodule("DelayManager")))-
>putDelay(dvod,122);
                        ((DelayManager *)(this->getParentModule()-
>getParentModule()->getParentModule()->getSubmodule("DelayManager")))-
>putDelay(dfx,123);

                       dtotal =
0,dname=0,dstream=0,dTCP=0,diptv=0,dvoip=0,dvod=0,dfx=0;
                }
```

Archivo IPclient.cc:251

```
if(dynamic_cast<GenericApplicationMessage*>(datagram-
>getEncapsulatedMsg()->getEncapsulatedMsg())){
            int64 databytes = datagram->getByteLength();
            int tipo = dynamic_cast<GenericApplicationMessage*>(datagram-
>getEncapsulatedMsg()->getEncapsulatedMsg())->getApp();
            ((ClientTraffic *)(this->getParentModule()-
>getParentModule()->gate("pppg$o",0)->getNextGate()->getOwnerModule()-
>getSubmodule("clientTraffic")))->putUpBytes(databytes,tipo);
      }else{
            switch (datagram->getTransportProtocol())
            {
            case IP_PROT_TCP:
            int64 databytes = datagram->getByteLength();
            int tipo = 210;
            ((ClientTraffic *)(this->getParentModule()-
>getParentModule()->gate("pppg$o",0)->getNextGate()->getOwnerModule()-
>getSubmodule("clientTraffic")))->putUpBytes(databytes,tipo);
            break;
            }
      }
```

2.14

IPServer.cc:219

```
if (tracingOn)
      {
            downt+=datagram->getByteLength();
      }
```

IPServer.cc:309

```
if (tracingOn)
      {
            upt+=datagram->getByteLength();
      }
```

2.15

Archivo ServerTraffic.cc:

```
#include <omnetpp.h>
#include "ServerTraffic.h"



Define_Module(ServerTraffic);

void ServerTraffic::initialize()
{
	tracingInterval = par("tracingInterval");

	traceOn = par("traceOn").boolValue();

	downt=0, upt=0;
	if(traceOn)
	{
		down.setName("Downstream"), up.setName("Upstream");
	}

	traceMsg = new cMessage("traceOut");
	scheduleAt(simTime() + tracingInterval, traceMsg);

}
void ServerTraffic::handleMessage(cMessage *msg){
	if (msg->isSelfMessage())
	{
		if (msg == traceMsg)
		{
			if (traceOn)
			{
				down.recordWithTimestamp(simTime(), downt);
				up.recordWithTimestamp(simTime(), upt);
				downt=0, upt=0;
			}
		}
		scheduleAt(simTime() + tracingInterval, msg);
	}
}
```

```
void ServerTraffic::putDownBytes(int64 downBytes){
	if(traceOn){
		downt+=downBytes;
	}
}
void ServerTraffic::putUpBytes(int64 upBytes){
	if(traceOn){
		upt+=upBytes;
	}
}
```

2.16

IPserver.cc:311

```
((ServerTraffic *)(this->getParentModule()->getParentModule()->getParentModule()->getSubmodule("serverTraffic")))->putUpBytes(datagram->getByteLength());
```

IPserver.cc:221

```
((ServerTraffic *)(this->getParentModule()->getParentModule()->getParentModule()->getSubmodule("serverTraffic")))->putDownBytes(datagram->getByteLength());
```

2.17

DelayManager.cc:

```
#include <omnetpp.h>
#include "DelayManager.h"


Define_Module(DelayManager);
```

```
void DelayManager::initialize()
{
	traceOn1 = par("traceOn1").boolValue();
	traceOn2 = par("traceOn2").boolValue();
	limite = par("limitdelay");
	tracingInterval = par("tracingInterval");
	totalTrace = 0;
	namet=0,streamt=0,TCPt=0,iptvt=0,voipt=0,vodt=0,fxt=0,maximo=0;

	total.setName("Total");
	name.setName("Name");
	stream.setName("Streaming");
	TCP.setName("TCP");
	iptv.setName("IPTV");
	voip.setName("VoIP");
	vod.setName("VOD");
	fx.setName("Fx");
	max.setName("Maximos");


	double startTime = par("traceStartTime");
	traceMsg = new cMessage("trace msg");
	if(startTime <=0)
	scheduleAt(simTime() + tracingInterval, traceMsg);
	else
	scheduleAt(simTime() + startTime, traceMsg);



}
void DelayManager::handleMessage(cMessage *msg)
{
	if(msg->isSelfMessage()){
		if(traceMsg == msg){//Cantidad de clientes que superan limite
máximo
			if(traceOn1){
			total.recordWithTimestamp(simTime(),totalTrace);
			name.recordWithTimestamp(simTime(),namet);
			stream.recordWithTimestamp(simTime(),streamt);
```

```
                    TCP.recordWithTimestamp(simTime(),TCPt);
                    iptv.recordWithTimestamp(simTime(),iptvt);
                    voip.recordWithTimestamp(simTime(),voipt);
                    vod.recordWithTimestamp(simTime(),vodt);
                    fx.recordWithTimestamp(simTime(),fxt);
                    }
                    if(traceOn2){
                          max.recordWithTimestamp(simTime(),maximo);
                    }
                    totalTrace = 0;
                    maximo =0;
                    namet=0,streamt=0,TCPt=0,iptvt=0,voipt=0,vodt=0,fxt=0;

                    scheduleAt(simTime() + tracingInterval, traceMsg);
              }
        }
}

void DelayManager::putDelay(double maxdelay,int tipo){
        if(maxdelay >= maximo){
              maximo = maxdelay;
        }
        if(maxdelay >= limite){

        switch(tipo){
              case 0:
                    totalTrace++;
                    break;
              case 11:
                    namet++;
                    break;
              case 12:
                    streamt++;
                    break;
              case 210:
                    TCPt++;
                    break;
              case 120:
                    iptvt++;
```

```
                    break;
              case 121:
                    voipt++;
                    break;
              case 122:
                    vodt++;
                    break;
              case 123:
                    fxt++;
                    break;

       }
       }else{}
}
```

2.18

IPclient.cc:105

```
if (tracingOn)
                    {
                          ((DelayManager *)(this->getParentModule()->getParentModule()->getParentModule()->getSubmodule("DelayManager")))->putDelay(dtotal,0);
                          ((DelayManager *)(this->getParentModule()->getParentModule()->getParentModule()->getSubmodule("DelayManager")))->putDelay(dname,11);
                          ((DelayManager *)(this->getParentModule()->getParentModule()->getParentModule()->getSubmodule("DelayManager")))->putDelay(dstream,12);
                          ((DelayManager *)(this->getParentModule()->getParentModule()->getParentModule()->getSubmodule("DelayManager")))->putDelay(dTCP,210);
                          ((DelayManager *)(this->getParentModule()->getParentModule()->getParentModule()->getSubmodule("DelayManager")))->putDelay(diptv,120);
                          ((DelayManager *)(this->getParentModule()->getParentModule()->getParentModule()->getSubmodule("DelayManager")))->putDelay(dvoip,121);
```

```
                        ((DelayManager *)(this->getParentModule()-
>getParentModule()->getParentModule()->getSubmodule("DelayManager")))-
>putDelay(dvod,122);
                        ((DelayManager *)(this->getParentModule()-
>getParentModule()->getParentModule()->getSubmodule("DelayManager")))-
>putDelay(dfx,123);

                        dtotal =
0,dname=0,dstream=0,dTCP=0,diptv=0,dvoip=0,dvod=0,dfx=0;

                  }
```

IPClient:368

```
if (tracingOn)
      {
            double delaybeta = (simTime()-datagram-
>getCreationTime()).dbl();
            if(delaybeta>=dtotal)
                  dtotal = delaybeta;
            if(dynamic_cast<GenericApplicationMessage*>(datagram-
>getEncapsulatedMsg()->getEncapsulatedMsg())){


      switch(dynamic_cast<GenericApplicationMessage*>(datagram-
>getEncapsulatedMsg()->getEncapsulatedMsg())->getApp()){
                  case 11:
                        if(delaybeta>=dname)
                              dname = delaybeta;
                        break;
                  case 12:
                        if(delaybeta>=dstream)
                              dstream = delaybeta;
                        break;
                  case 120:
                        if(delaybeta>=diptv)
                              diptv = delaybeta;
                        break;
                  case 121:
                        if(delaybeta>=dvoip)
```

```
                              dvoip = delaybeta;
                        break;
                  case 122:
                        if(delaybeta>=dvod)
                              dvod = delaybeta;
                        break;
                  case 123:
                        if(delaybeta>=dfx)
                              dfx = delaybeta;
                        break;
                  }
            }else{
                  if(delaybeta>=dTCP)
                        dTCP = delaybeta;
            }

      }
```

2.19

Archivo Cliente.ned:53

```
        inetUser: InetUser {
            parameters:
                lengthFactor = lengthFactor;
                timeFactor = timeFactor;
                offset = offset;
                profileId = profileId;
                @display("p=60,390;i=abstract/person");

        }
```

2.20

Archivo InetUser.cc:212

```
double startTime[nApps];
      for(int j = 0;j<nApps;j++){
      // activate the InetUser at startTime
      cMessage *startMessage = new cMessage("InetUser wakeup");
      startMessage->setKind(MSGKIND_START);
```

```
      startTime[j] = par("startTime");
      double simulationDuration = par("simulationDuration");
      int i;
      for(i=0;i<1000;i++){
            if(startTime[j]<0){
                  startTime[j] = par("startTime");
            }else{
                  if(startTime[j] > simulationDuration){
                        startTime[j] = par("startTime");
                  }else{
                        break;
                  }
            }
      }
      if(i==1000)
            opp_error("couldn't get startTime into the simulation time");

      //seting offset
      startTime[j] =
simulationDuration*((startTime[j]+offset)/simulationDuration -
(int)((startTime[j]+offset)/simulationDuration));
      scheduleAt((simtime_t) startTime[j], startMessage);
      }
```

2.21

Archivo InetUser.cc:365

```
startTime[j] =
simulationDuration*((startTime[j]+offset)/simulationDuration -
(int)((startTime[j]+offset)/simulationDuration));
```

2.22

Archivo InetUser.cc:

```
if (curTrafficProfile.profileID != 120 &&
!curTargetInfo.address.isUnspecified()){
            curTrafficProfile.timeBetweenRequests =
timeFactor*curTrafficProfile.timeBetweenRequests;
            curTrafficProfile.timeToRespond =
```

```
timeFactor*curTrafficProfile.timeToRespond;
            curTrafficProfile.requestLength =
(int)(lengthFactor*curTrafficProfile.requestLength);
            curTrafficProfile.replyLength =
(int)(lengthFactor*curTrafficProfile.replyLength);


      }
```

2.23

Archivo InetUser:105

```
if(currentProfileId==123){
                  cm->getGivenServer(curTargetInfo, curTrafficProfile,
currentProfileId);
            }else{
                  cm->getServer(curTargetInfo, curTrafficProfile,
exceptId, getId());
            }
```

2.24

ConnectionManager:480

```
void ConnectionManager::getGivenServer(TargetInfo &ti,TrafficProfile &tp,
int profileId)
{
      serverMap::iterator it = servers.find(profileId);
      serverVector *vec = it->second;
      ti.address = (*vec)[0]->address;
      ti.port = (*vec)[0]->port;

      tpm->getTrafficProfile(tp,profileId);
}
```

TrafficProfileManager:198

```
void TrafficProfileManager::getTrafficProfile(TrafficProfile &t, int
profileId)
{
      unsigned int i = 0;
      for (; i < (profiles.size() - 1); i++)
```

```
      {
            if (profiles[i]->profileID==profileId)
                  break;
      }
      t = *profiles[i];
      EV << " ID DE PERFIL DE TRAFICO ELEGIDA: " << t.profileID;
}
```

2.25

InetUser.cc:170

```
currentProfileId = par("profileId");
```

2.26

Archivos BackupServer.ned, InteractiveServer.ned, MailServer.ned, WebServer.ned

```
            tcp: TCP {
                  parameters:
                        @display("p=163,154;i=block/wheelbarrow");
            }
```