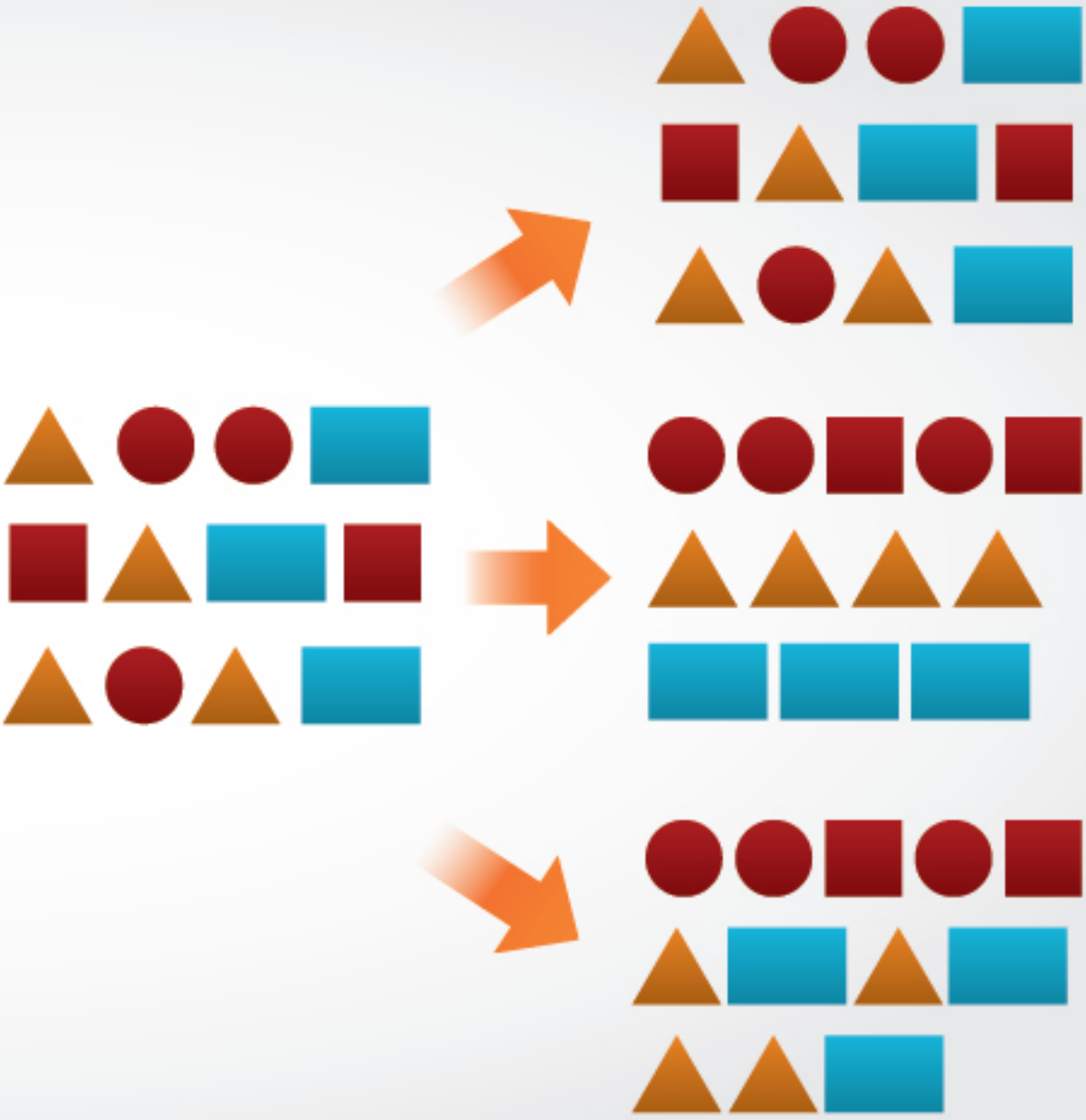
UNIVERSIDAD ICESI

ISBN 978-958-8357-74-4



Simulando con OMNET

Selección de la herramienta y su utilización

Madelayne Morales R. • Manuel A. Calle P. • José Darío Tovar V. • Juan Carlos Cuéllar Q.

Simulando con OMNET

Selección de la herramienta y su utilización

Simulando con OMNET. Selección de la herramienta y su utilización / Madelayne Morales R., Manuel A. Calle P., José Darío Tovar V., Juan Carlos Cuéllar Q.

1 ed. Cali, Colombia. Universidad Icesi, 2013.
92 p., 17x24 cm
ISBN: 978-958-8357-74-4

1.Telemática 2.Simulación 3.Calidad de Servicio I.Tit
004.66 – dc22

Simulando con OMNET. Selección de la herramienta y su utilización
Universidad Icesi

Impreso en Cali - Colombia (Litocencoa SAS)

Universidad Icesi - Facultad de Ingeniería
Calle 18 No.122-135, Cali (Colombia)
Tel: (572) 555 2334
www.Icesi.edu.co
1 ed. Febrero de 2013

Agradecimientos

A Carlos Andrés Muñoz Campiño, estudiante de Ingeniería Telemática de la Universidad Icesi, quién se encargó de organizar toda la documentación de este material en el procesador de texto Latex.

A Andrés Navarro Cadavid, Director del Grupo de Investigación en Informática y Telecomunicaciones (i2T) de la Universidad Icesi, por el apoyo en la gestión para lograr la publicación de este material.

A Jose Ignacio Claros, por la edición del material y el acompañamiento en el proceso de publicación.

Al Dr. Gonzalo Ulloa, Decano de la Facultad de Ingeniería, por el apoyo y la motivación brindada durante la ejecución del proyecto y el proceso de publicación de este material.

Currículos

Madelayne Morales Rodríguez. Ingeniera Telemática de la Universidad Icesi (2012) con interés en las áreas de planeación y gerencia de proyectos en tecnologías sostenibles y servicios interactivos. Desde su último año de carrera es miembro del Grupo de Investigación en Informática y Telecomunicaciones (i2T) de Icesi, en el cual trabaja en la línea de comunicaciones inalámbricas. Forma parte del equipo de investigadores del Proyecto SUCCESS TV [Servicio universal en cooperación Colombia-España para sistemas satélite de televisión] financiado por Colciencias. made1989@gmail.com

Manuel Calle Pérez. Ingeniero Telemático de la Universidad Icesi (2013), con interés en las áreas de gerencia de proyectos en tecnología y mejoramiento de procesos tecnológicos. Desde 2012 trabaja en la Caja de Compensación Familiar del Valle del Cauca [Comfandi], donde realiza revisión y mejoramiento de procesos tecnológicos, y participa para el Proyecto Mujeres Ahorradoras en Acción (PMAA), en el cual desarrolla he implementa la plataforma del PMAA en conjunto con el Departamento para la Prosperidad Social de Colombia. callemanuel@hotmail.com

José Darío Tovar. Ingeniero Telemático de la Universidad Icesi (2013) con interés en las áreas de planeación y gerencia de proyectos, administración y despliegue de servicios. Desde su último año de carrera hace parte del grupo de soporte corporativo de Calipso Comunicaciones S.A, donde labora actualmente. jdariotv@gmail.com

Juan Carlos Cuéllar Quiñonez. Ingeniero Electricista de la Universidad del Valle con especializaciones en Redes y Servicios Telemáticos de la Universidad del Cauca y Redes y Comunicaciones de la Universidad Icesi, y Maestría en Telecomunicaciones de la Universidad Pontificia Bolivariana de Medellín. Es docente, investigador y coordinador del Departamento de Ciencias Físicas y Tecnológicas en la Universidad Icesi y está cursando el Doctorado en Telemática en la Universidad del Cauca. Sus áreas de interés son las redes de próxima generación (NGN) y la configuración de dispositivos de interconectividad, jcuellar@icesi.edu.co

Tabla de contenido

Índice de Tablas 5
Índice de Figuras 7
Presentación 11
Capítulo 1. Introducción 13
Capítulo 2. Elección de una herramienta de simulación 21
2.1. Parámetros para la selección de una herramienta de simulación 23
2.1.1. Uso investigativo 23
2.1.2. Tipo de licencia 24
2.1.3. Curva de aprendizaje 24
2.1.4. Plataformas que lo soportan 25
2.1.5 Interfaz gráfica 25
2.1.6. Graficación de resultados 25
2.1.7. Tecnologías y protocolos de niveles 2 y 3 que soporta 26
2.1.8. Tráfico que permite modelar 26
2.2. Herramientas de simulación 27
2.2.1. NS-2 27
2.2.2. NC-TUNS 29
2.2.3. OPNET Modeler 30
2.2.4. OMNET++ 31
2.2.5. NS-3 33
2.2.6. GNS3 34
2.3. Recomendaciones para la elección de la herramienta 35
Capítulo 3. Instalación de OMNET 37
3.1. Instalación de OMNeT++ 38
3.2. Instalación de INET Framework 44
Capítulo 4. Lenguajes de OMNET++ 49
4.1. Características del lenguaje NED 50
4.2 Módulos simples 51
4.2.1 Simulaciones de eventos discretos 51
4.2.2 Implementación de la FES 51
4.3. Estructura de los componentes y las conexiones 52

4.4. Definir módulos simples en OMNet++ 53
4.5. Constructor 54
Capítulo 5. Ejemplo Tic-Toc 57
Capítulo 6. Simulación de esquemas de encolamiento 71
6.1. Entorno de desarrollo utilizado para la implementación 72
6.2. Escenario de simulación 73
6.2.1. Paquete 74
6.2.2. Generador de tráfico 75
6.2.3. Colas 75
6.2.4. Consumidor 78
6.2.5. Canal 79
6.3. Resultados experimentales 79
6.3.1. Datos obtenidos en las simulaciones 79
6.4. Gráficas del comportamiento de las colas 81
6.4.1. Gráfica FIFO 81
6.4.2. Gráfica PQ 81
6.4.3. Gráfica LLQ 82
6.5.Comentarios finales 82
Referencias 85

Índice de Tablas

Tabla 1. Valores del campo IP *Precedence* 17
Tabla 2. Parámetros para la medición de calidad de servicio - Rec. Y.1540 (ITU-T, 2011b) 18
Tabla 3. Clases de calidad de servicio 19
Tabla 4. Parametrización de las herramientas de simulación 36
Tabla 5. Resumen de los resultados de simulación 80

Índice de Figuras

Figura 1. Datagrama IP - Campo DSCP 16
Figura 2. Escenario - Simulación en NS-2 28
Figura 3. Fragmento de Script - Simulación en NS-2 29
Figura 4. Interfaz gráfica NCTUns 29
Figura 5. Interfaz gráfica OPNET 31
Figura 6. Interfaz gráfica OMNET++ 33
Figura 7. Interfaz gráfica NS3 34
Figura 8. Interfaz gráfica GNS3 35
Figura 9. Contenido de la carpeta *Simulador_OMNeT* 38
Figura 10. Ventana - Consola de MSYS 39
Figura 11. Ventana - Consola de MSYS construcción de las librerías de simulación (1 de 3) 39
Figura 12. Ventana - Consola de MSYS construcción de las librerías de simulación (2 de 3) 39
Figura 13. Ventana - Consola de MSYS construcción de las librerías de simulación (3 de 3) 40
Figura 14. Ventana - Consola de ejecución del ejemplo Dyna (1 de 2) 40
Figura 15. Ventana - Consola de ejecución del ejemplo Dyna (2 de 2) 40
Figura 16. Ventana - *Set up an inifile configuration* (1 de 2) 41
Figura 17. Ventana - *Set up an inifile configuration* (2 de 2) 41
Figura 18. Interfaz gráfica de OMNet++ 41
Figura 19. Ventana - Información el proceso de simulación 41
Figura 20. Ubicación de creación de la carpeta *jre* 42
Figura 21. Contenido de la carpeta *jre* 42
Figura 22. Ventana - Comando de ejecución del IDE de OMNet++ 42
Figura 23. Ventana - Workspace launcher OMNet++ 43
Figura 24. Ventana - Loading 43
Figura 25. Ventana - Bienvenida al simulador OMNet++ 43
Figura 26. Ventana - Documentos 44
Figura 27. Ventana - Importación de INET Framework (1 de 4) 44
Figura 28. Ventana - Importación de INET Framework (2 de 4) 45
Figura 29. Ventana - Importación de INET Framework (3 de 4) 45

Figura 30. Ventana - Importación de INET Framework (4 de 4) 46
Figura 31. Ventana - *Project Explorer* con el paquete INET OK 46
Figura 32. Ventana - Utilidades con los módulos INET cargados 47
Figura 33. Comportamiento FES 51
Figura 34. Representación *cComponent* (OMNeT Community, 2011b) 52
Figura 35. Herencia *cChannel* (OMNeT Community, 2011b) 52
Figura 36. Herencia *cModule* (OMNeT Community, 2011b) 53
Figura 37. Estructura inicial módulo simple 54
Figura 38. Constructor sin argumentos 54
Figura 39. Constructor con argumentos 54
Figura 40. Creación de un proyecto tipo OMNeT++ 58
Figura 41. Ventana de creación de un proyecto tipo OMNeT++ (1 de 2) 58
Figura 42. Ventana de creación de un proyecto tipo OMNeT++ (2 de 2) 59
Figura 43. *Project Explorer* del proyecto TicToc 59
Figura 44. Ventana de utilidades del proyecto 60
Figura 45. Creación del archivo *.ned* 60
Figura 46. Ventana de creación del archivo *.ned* 61
Figura 47. Ventana de diseño del archivo *.ned* 61
Figura 48. TicToc archivo *.ned* 63
Figura 49. Ventana de utilidades del proyecto con el nuevo módulo 63
Figura 50. Creación de la clase *.cc* 64
Figura 51. Ventana de creación de la clase *.cc* 64
Figura 52. Ventana Project Explorer con la clase *.cc* 65
Figura 53. Creación del archivo *.ini* 65
Figura 54. Ventana de la creación del archivo *.ini* 66
Figura 55. Código fuente del archivo *.ini* 66
Figura 56. Inicio de la simulación 67
Figura 57. Mensaje de advertencia 67
Figura 58. Ventana - *Set up an Inifile Configuration* 67
Figura 59. Ventana - Interfaz gráfica de OMNeT++ (1 de 3) 68
Figura 60. Ventana - Interfaz gráfica de OMNeT++ (2 de 3) 68
Figura 61. Ventana - Interfaz gráfica de OMNeT++ (3 de 3) 69
Figura 62. Ventana de información del proceso de simulación del TicToc 69
Figura 63. Presentación archivo *.ned* (OMNeT++ Community, 2012) 73

Figura 64. Escenario de simulación adaptado de simulador OMNeT++ *74*
Figura 65. Generador de tráfico (OMNeT++ Community, 2011a) *75*
Figura 66. Encolamiento FIFO (Frabs, 2008) *77*
Figura 67. Encolamiento PQ (Moorey, 2008) *77*
Figura 68. Encolamiento LLQ (http://www.ccietalk.com/) *78*
Figura 69. Resultados de la simulación FIFO *79*
Figura 70. Comportamiento de la cola FIFO *81*
Figura 71. Comportamiento del encolamiento PQ *82*
Figura 72. Comportamiento del encolamiento LLQ *82*

Presentación

Las herramientas de simulación juegan un papel fundamental en la academia y en los procesos de investigación, ya que permiten recrear escenarios de red con el fin analizar su desempeño o funcionamiento o realizar medidas de alguna variable en particular.

Con respecto de la academia, las herramientas de simulación permiten al estudiante aprender y verificar el funcionamiento de protocolos y/o dispositivos de red, eligiendo una herramienta adecuada para su nivel de aprendizaje; es así como existen desde herramientas de nivel básico, útiles para los primeros semestres de la carrera, hasta herramientas complejas que se utilizan en los últimos semestres.

En cuanto a los procesos de investigación, estas herramientas son vitales porque permiten recrear escenarios reales, que son muy difíciles de implementar en los laboratorios debido a los recursos que demandan. Con su utilización se puede llegar a predecir el comportamiento de la red ante un evento en particular o analizar el funcionamiento que ella podría tener ante una propuesta investigativa de un nuevo protocolo o la configuración de algún dispositivo de red en particular.

Con base en lo anterior, este texto es producto de un proyecto de investigación de convocatoria interna en la Universidad Icesi, el cual fue desarrollado con estudiantes de Ingeniería Telemática. En él, se utilizó la herramienta de simulación OMNET para analizar esquemas de encolamiento en una red de datos, dejando documentado todo proceso realizado, para su uso posterior en ambientes académicos de aprendizaje.

La selección de la herramienta OMNET se hizo después de analizar diferentes herramientas que se encuentran disponibles en el mercado, las cuales se compararon utilizando un conjunto parámetros definidos por los autores. Se recomienda al lector tener en cuenta este grupo de parámetros al momento de elegir una herramienta de simulación, con el fin de elegir la que más se adapte a las necesidades del escenario a simular o el proyecto de investigación a realizar.

El texto está estructurado de la siguiente manera: en la Introducción se presentan conceptos básicos sobre Calidad de Servicio [QoS] y las recomendaciones de Unión Internacional de Telecomunicaciones [UIT-T] que indican los parámetros a tener en cuenta para verificar que se esté ofreciendo QoS a las aplicaciones, y se describen los esquemas de encolamiento más utilizados en las redes de datos. En el Capítulo 2 se presentan las características de las herramientas de simulación más utilizadas en el ambiente investigativo y se entrega el grupo de parámetros a analizar para la selección de una herramienta de simulación. En el capítulo 3 se explica cómo instalar la herramienta de simulación y en el capítulo 4 se presenta el lenguaje con que se debe programar en la herramienta. El capítulo 5 presenta el ejemplo tic-toc, el cual es muy útil para las personas que están en el proceso de aprender el manejo de la herramienta. Para finalizar, en el capítulo 6 se presenta todo el proceso que se debe seguir para simular esquemas de encolamiento con el OMNET.

Capítulo I

Introducción

La diferenciación y el soporte en calidad de servicio es utilizada en muchas arquitecturas de red, técnicas y frameworks, tales como IP, con los esquemas de DiffServ e IntServ, redes NGN, GMPLS (*Generalized Multiprotocol Label Switching*) y redes ópticas, entre otras. Actualmente, la calidad de servicio no es solo una cuestión técnica, sino que se ha convertido en un producto que puede ser visto desde numerosas perspectivas, en las que los clientes, cada día, demandan más servicios, de mayor calidad. Por lo tanto, los proveedores deben mejorar sus implementaciones para permanecer en este reñido mercado (Piotr, Stankiewicz, Cholda, & Jajszczyk, 2011).

En este contexto, las redes actuales deben garantizar la calidad en los servicios sin importar el incremento paulatino de usuarios y dispositivos de interconectividad. La mejor manera de hacerlo no es necesariamente invirtiendo en la infraestructura más moderna que soporte técnicas para este fin. Existen mecanismos que se pueden aplicar sobre infraestructuras de red actuales, las cuales ayudan a manejar el tráfico de manera adecuada, de tal manera que los parámetros de calidad de servicio se mantengan dentro de los límites permisibles establecidos por la Recomendación Y.1541 (UIT, 2011a).

Como una primera instancia para el establecimiento de técnicas apropiadas que permitan la prestación de niveles de calidad de servicio, es necesario conocer el tráfico que circula sobre la red; el primer estudio debe estar enfocado en este sentido, en caracterizar todo el tráfico, la dirección, la cantidad y los dispositivos sobre los que recae la mayor parte del procesamiento. Una vez reconocidos estos aspectos es necesario distinguir los diferentes flujos de tráfico, etiquetando las tramas de cada aplicación, de tal manera que se le pueda dar un tratamiento preferencial a las aplicaciones más sensibles al retardo. Posteriormente, teniendo en cuenta la distinción de tráfico por el marcado de las tramas, se debe elegir e implementar, sobre los dispositivos de enrutamiento, el encolamiento que mejor se adapte, para garantizar la calidad en las operaciones; de este modo es posible garantizar mejores niveles de calidad, sin la necesidad de invertir en nuevos dispositivos.

El estudio del tipo de tráfico circulante sobre la red, debería ser una tarea relativamente sencilla de realizar, utilizando herramientas de monitoreo (de las cuales hay muchas de licencia libre). El marcado de las tramas lo pueden realizar los mismos dispositivos de enrutamiento dependiendo de su modelo; actualmente, inclusive los dispositivos de usuario final poseen la capacidad de realizar esta labor. Sin embargo, el establecimiento del esquema de encolamiento que mejor se adapte a una arquitectura de red, no es una tarea de *ensayo y error* que se deba realizar sobre la implementación física como tal, ya que en un intento se puede instaurar una técnica inadecuada que podría generar inconsistencia, cargar los dispositivos o los enlaces y, de este modo, congestionar completamente la infraestructura, produciendo exactamente un efecto contrario a lo que se quiere: mantener los niveles de calidad de servicio dentro de un margen permisible.

Por esta razón, la mejor manera de abordar la situación es usando herramientas de simulación, que permiten la evaluación de implementaciones de red a nivel software, sin poner en riesgo la integridad de la implementación física. La variedad de estrategias que pueden ser evaluadas utilizando este medio cubren diversos campos; para este caso, el estudio estará enfocado en las técnicas de encolamiento, las cuales hacen parte de un esquema muy eficiente para ofrecer calidad de servicio. Al finalizar, se espera establecer cuál es el esquema de encolamiento más apropiado para una arquitectura de red, teniendo en cuenta la identificación previa de los aspectos relacionados con el tráfico, mencionados.

Para el cumplimento de este objetivo, se realizará un estudio comparativo de las herramientas de simulación más utilizadas en el medio, con el fin de identificar los pro y los contras de cada una, con base en una serie de parámetros que se debe tener en cuenta en el proceso de selección de la herramienta. Posterior a esto, teniendo en cuenta los resultados del estudio de comparación de las diferentes herramientas de red, se seleccionará un simulador y se realizará un estudio teórico sobre los esquemas de encolamiento más comunes, siempre y cuando la herramienta soporte la implementación. La meta final es establecer, de acuerdo con unas características de tráfico definidas, cuál es el esquema de encolamiento que ofrece mejores resultados y caracterizar la diferencia de desempeño entre cada uno.

La calidad de servicio se puede ver, principalmente, desde dos perspectivas, denominadas clasificación y encolamiento. La clasificación se define como la manera en que el tráfico puede ser identificado y marcado. Por su parte, las colas son *buffers* de memoria de alto desempeño encargadas de procesar adecuadamente los datos; ellas pueden comportarse de diferentes maneras, dependiendo de la clasificación previa a la que el tráfico haya sido sometido. Por lo general, se suelen distinguir por la prioridad a la que despachan cada tipo de información, teniendo en cuenta que aquellos flujos de información más sensibles al retardo, prevalecen en comparación con datos que no son de misión crítica.

De este modo, clasificar, marcar y encolar el tráfico se realiza a través de políticas (*Polices*) establecidas por los administradores de red sobre los dispositivos de conectividad.

Tal como se mencionó en un principio, la medida inicial para brindar calidad de servicio es identificar los datos para proceder a su marcación, para de este modo diferenciar los flujos que circulan sobre la red. Dicho proceso de caracterización se puede realizar utilizando marcas de tramas a nivel 2 o 3 de OSI, por lo cual es posible utilizar alguno de los siguientes tres métodos de clasificación:

- uso de listas de control de acceso (ACL), con el fin de diferenciar los flujos de datos, por medio de la dirección IP, el puerto, etc.;
- clasificación por tipo de aplicación, que consiste en el reconocimiento de los

flujos de datos inspeccionando los paquetes. Este mecanismo generalmente es usado por los dispositivos CISCO; y

- *Switchport*, método en el que intervienen campos de las tramas de nivel 2, utilizando protocolos como el 802.1Q (LAN/MAN Standards Committee ,2006).

Una vez clasificados los datos, se procede a etiquetarlos de acuerdo con las clases de servicio que se hayan establecido. Este proceso de marcado se puede realizar a nivel 3 en la cabecera IP, utilizando el campo DSCP [*Differentiated Services Code Point* o campo de diferenciación de servicio] (Nichols, Blake, Baker, & Black, 1998), como se puede observar en la Figura 1.

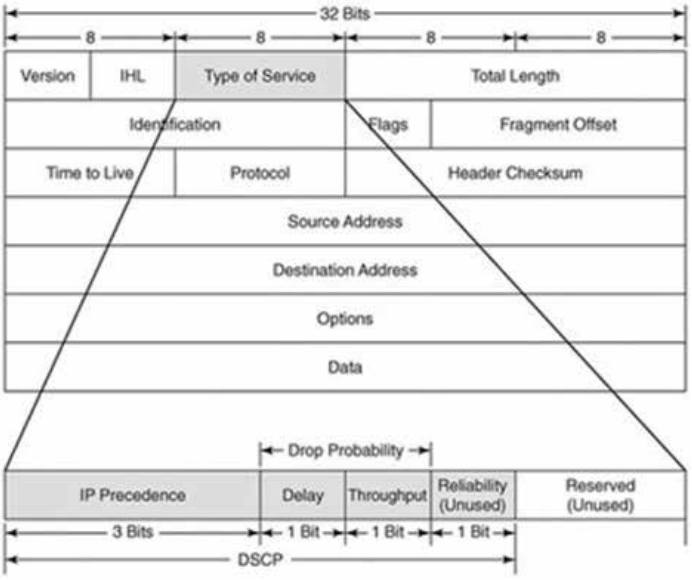


Figura 1. Datagrama IP - Campo DSCP (In Search of..., 2012)

Como se pude apreciar en esta misma figura y tal cual como describe el ISI (1981), RFC del protocolo de internet, el campo DSCP ocupa 6 bits de la cabecera IP: los tres primeros representan la precedencia o prioridad con la que pueden ser catalogados los tipos de tráfico; los tres siguientes el retardo, el *Throughput* y la confiabilidad, codificándose esta última, normal con 0 (cero) y alta con 1 (uno), indicando el incremento en el costo de los servicios a partir de estos parámetros. La Tabla 1 presenta los valores que puede tomar el campo *IP Precedence* inscritos en el RFC 791.

En lo concerniente al encolamiento, es posible la configuración de gran variedad de implementaciones en los dispositivos de interconectividad, generalmente enrutadores. De los tipos de colas, se destacan la cola FIFO (*First In First Out*), las colas de prioridad o PQ (*Priority Queue*) y la cola LLQ (*Low Latency Queuing*).

Las colas de tipo FIFO pueden ser definidas con la analogía de las personas que esperan en la fila de un banco y van siendo atendidas en el orden en que llegaron; el concepto indica que la primera persona en entrar, es la primera persona en salir. En

Código binario	Descripción
111	Network Control
110	Internetwork Control
101	CRITIC/ECP Flash
100	Override Flash
011	Immediate
010	Priority
001	Routine
000	

Tabla 1. Valores del campo IP *Precedence*

las colas PQ, en cambio, cada elemento tiene asignada una *prioridad*; aquellos con la prioridad más alta son atendidos antes que aquellos con una prioridad más baja; si dos elementos tienen la misma prioridad, son procesados de acuerdo con el orden de llegada. Las colas de tipo LLQ, por su parte, funcionan de manera similar a las de prioridad, pero corrigen el riesgo de que grandes flujos de datos de alta prioridad eviten que los de prioridad menor puedan ser atendidos, estableciendo intervalos de tiempo para la atención entre los distintos flujos de tráfico.

Como parte del esquema de desempeño para la prestación de niveles acertados de calidad de servicio, es de vital importancia la conformación del tráfico, (*traffic shaping*), que consiste en controlar el tráfico para evitar que se presenten paquetes descartados o que se sobrepase la capacidad de los flujos que llegan a la cola para ser atendidos. En cierto modo, se evita la congestión, la sobrecarga y el descarte sobre los flujos de datos específicos, controlando el nivel de transferencia y reduciéndolo cuando sobrepasa los límites establecidos. Por ello, esta tarea no se podría realizar efectivamente si al *shaping* no se le agrega políticas que establezcan dichos umbrales.

Hasta este punto se ha descrito la calidad de servicio como un conjunto de técnicas orientadas a garantizar el buen funcionamiento de las aplicaciones y/o servicios a los usuarios de tecnología. Sin embargo, cada día, el nivel de demanda de servicios –y su calidad– se incrementa haciendo que el concepto se extienda y sea visto como un producto, en el cual, el valor que se debe pagar depende del nivel de calidad de servicio que se requiere. Esto se ve reflejado en los niveles de acuerdo de servicio [SLA, *Service Level Agreement*], los cuales constituyen un pilar fundamental en el negocio tecnológico del mundo moderno, ya que buscan el aprovechamiento del medio en un 100 %, con el fin de maximizar las ganancias.

De otro lado, existen estándares que establecen parámetros para medir y evaluar la calidad de los servicios que se prestan. Organismos como la oficina de normalización de la Unión Internacional de Telecomunicaciones [ITU-T] definen normas para el funcionamiento de las redes a nivel mundial; como describe Blandon (2010) dicha organización agrupa sus recomendaciones en veintitrés categorías; de ellas, la categoría Y, define la infraestructura mundial de la información, aspectos del protocolo de

Parámetro	Descripción
IPTD (IP Transfer Delay)	Tiempo que transcurre entre el envío y la recepción de un paquete IP; se mide en un solo paquete, en una sola vía.
IPER (IP Packet Error)	Relación entre paquetes IP con errores y satisfactorios.
IPLR (IP Loss Ratio)	Relación entre el total de paquetes perdidos y el total de paquetes transmitidos.
IPDV (IP Delay Variation)	Variación del retardo: rango de variaciones que puede tener el delay.

Tabla 2. Parámetros para la medición de calidad de servicio - Rec. Y.1540 (ITU-T, 2011b)

internet y las redes NGN. En lo referente a calidad de servicio, la ITU-T cuenta con dos recomendaciones: la Y.1540 y la Y.1541 (ITU-T, 2011a; 2011b).

La primera de ellas define parámetros útiles para especificar y evaluar la calidad de servicio, teniendo en cuenta aspectos como la velocidad, la exactitud, la seguridad y la disponibilidad, e indicando también donde se puede medir cada uno de ellos (i.e., extremo a extremo o a tramos de la red). La definición formal de dichos parámetros se presenta en la Tabla 2.

Esta Recomendación está dirigida tanto a proveedores, como a fabricantes de equipos y usuarios. A los proveedores, con el fin de estimular el desarrollo de servicios que satisfagan a los usuarios, en términos de calidad; a los fabricantes de equipos, para influir en el diseño de sus dispositivos; y a los usuarios, por ser los encargados de evaluar el desempeño de los servicios.

La segunda, la recomendación Y.1541, define los valores límite de cada uno de los parámetros de calidad de servicio definidos en la Y.1540, los cuales varían dependiendo de cada clase de servicio (i.e., multimedia, video, *streaming* y datos), es decir, la prioridad en transferencia del tráfico (ver Tabla 3).

Generalmente, la evaluación de las distintas técnicas de prestación de calidad de servicio mencionadas, se realiza sobre software de simulación de red, lo cual garantiza que en el momento de realizar configuraciones sobre los dispositivos de red reales, se tendrá claridad sobre el esquema de calidad de servicio a configurar, con el fin de cometer la menor cantidad de errores en redes operativas.

En la actualidad, existen diversos tipos de software capaces de recrear escenarios de red sobre los que es posible implementar gran cantidad de protocolos, inyectar diversos tipos de tráfico y caracterizar parámetros de funcionalidad. Dichos simuladores son necesarios, tanto en la academia, para aprendizaje y el estudio de diversos conceptos, como en entornos empresariales, puesto que realizar análisis preliminares sobre la forma, el comportamiento y la funcionalidad de los protocolos, garantiza en gran medida el correcto funcionamiento de la implementación física.

Tabla 3. Clases de calidad de servicio

Parámetro de calidad de funcionamiento de red	Tipo de objetivo de calidad de funcionamiento	Clase de QoS					
		0	1	2	3	4	5 no especificada
IPTD	Límite superior en el IPTD medio	100 ms	400 ms	100 ms	400 ms	1 s	U
IPDV	Límite superior en el cuantil 1 – 10-3 de IPTD menos el IPTD mínimo	50 ms	50 ms	U	U	U	U
IPLR	Límite superior en la probabilidad de pérdida de paquetes	$1 X 10^{-3}$	$1 X 10^{-3}$	$1 X 10^{-3}$	$1 X 10^{-3}$	$1 X 10^{-3}$	U
IPER	Límite superior					$1 x 10^{-4}$	U

Capítulo 2

Elección de una herramienta de simulación

Hoy en día, las herramientas de simulación son un componente fundamental para el diseño, la implementación y el monitoreo de redes de comunicación, porque permiten predecir el comportamiento de diferentes eventos que pueden afectar el desempeño de la red y degradar la calidad de las aplicaciones y los servicios.

Como presenta Cuéllar (2011) las herramientas de simulación juegan un papel importante para evaluar el comportamiento de parámetros como el retardo y el *jitter* en una red, porque permiten la recreación de escenarios reales con el fin de analizar su desempeño, sin tener que implementar infraestructura física. Además, las simulaciones permiten tener en cuenta numerosas variables y son un método eficaz para la enseñanza y la investigación.

La simulación no es un concepto nuevo; siempre se ha buscado la manera de evaluar sistemas complejos y tal como define Phillips (2007), la simulación es la ejecución de un modelo representado por un programa de computadora que permite recrear entornos de red, ahorrando tiempo y dinero. Comercialmente existen simuladores de tiempo continuo y tiempo discreto; la simulación en tiempo discreto modela sistemas que cambian en el tiempo de acuerdo con los diferentes estados que una variable puede tener, algo muy útil para sistemas de comunicación. Los simuladores de tiempo continuo, por su parte, avanzan en el tiempo y constantemente revisan si ha ocurrido algún evento, con el fin de actualizar las variables correspondientes, para, solo en ese caso, realizar la modificación de valores.

En el área de investigación de redes es muy costoso desplegar una implementación completa, con múltiples computadores, dispositivos de interconectividad y enlaces para verificar los protocolos o algoritmos de red. La simulación en este sentido permite el ahorro de dinero y tiempo, y hace posible el diseño de redes complejas a partir de módulos simples. Sin embargo, no solo en el mundo de las redes de comunicaciones es útil simular, este concepto aplica a diferentes ciencias e ingenierías, entre otros campos (Pan, 2008).

Goldstein, Leisten, Stark, y Tickle (2005) por ejemplo, plantean el uso de herramientas de simulación de red como parte de un método pedagógico de enseñanza que permite a los estudiantes entender los diferentes conceptos de una manera más clara, sencilla y representada de una manera tangible, basándose en la premisa de que los estudiantes aprenden con mayor facilidad si disfrutan usando una herramienta educativa. Es así como Pérez-Hardy (2003) expone diferentes vías en las que un simulador de red puede ser usado para enseñar a los estudiantes tanto los conceptos básicos como los avanzados, el diseño y los principios de desempeño. De esta manera, aunque la simulación de red es una práctica que se utiliza en la academia, se busca su aplicación también en la industria, ya que ofrece los beneficios descritos por Breslau (2000), que se listan a continuación:

» mejora la validación del comportamiento de protocolos existentes;

» permite el desarrollo de nuevos protocolos;
» brinda la oportunidad de estudiar protocolos a gran escala; y
» permite comparar resultados entre diferentes implementaciones de red.

En la academia, diferentes autores abordan el tema. Vom Lehn, Weingartner, y Wehrle (2008) y Hlupic (2000) presentan los conceptos de simulación y hacen comparaciones entre varias herramientas. Los resultados obtenidos muestran tanto la experiencia de los usuarios con el uso de las herramientas, como también, si fue o no necesario realizar cambios en su código. Además, muestran como en la mayoría de los casos, usar las herramientas de simulación no constituye un proceso dispendioso, aunque algunas son limitadas para su uso en problemas muy complejos o no estandarizados. Uno de los factores cruciales en la elección de una herramienta es el costo de la licencia, cuando ésta es de tipo comercial.

A continuación, se presenta una serie de parámetros de comparación que sirve como guía para evaluar algunas de las herramientas de simulación más usadas en el ámbito investigativo; así el usuario final cuenta con la fundamentación necesaria para elegir la herramienta que mejor se adapte a sus necesidades, como han hecho Bragg (2000) y García, Escobar, Navarro y Vásquez (2011).

2.1. Parámetros para la selección de una herramienta de simulación

Los siguientes parámetros permiten elegir una herramienta de simulación para un uso específico, en determinada área de telecomunicaciones. En cada caso, se define el parámetro y la manera cualitativa para su medición.

2.1.1. Uso investigativo

Este parámetro tiene como finalidad establecer niveles para comparar el uso de las herramientas de simulación en áreas académicas e investigativas, con el fin de explorar los aportes que han logrado las diferentes comunidades respecto de su uso, y hacer un primer acercamiento que permita identificar las principales implementaciones de red en que son usadas.

Los niveles de medida para constituir puntos de comparación entre el uso investigativo de un simulador de red u otro son:

» *Alto*. Indica que la herramienta ha sido utilizada en un alto número de referencias científicas y en proyectos de investigación.
» *Medio*. Este nivel cataloga las herramientas que son usadas frecuentemente en la comunidad investigativa, pero que debido a algunas de sus características o especificaciones técnicas, no han logrado la proliferación deseada.
» *Bajo*. Esta clasificación indica poco uso investigativo del simulador, debido a las

limitaciones de sus características, por lo cual las referencias en los grupos de investigación es mínima.

2.1.2. Tipo de licencia

Para efectos prácticos, dependiendo del tipo de licencia de la herramienta, ella puede tener habilitadas todas sus funciones o solo un grupo de ellas; esto también puede ir ligado con el valor que se cobre por la licencia. Con base en lo anterior, la forma de categorizar los tipos de licencias es:

Libre. A partir de esta forma de cuantificación se busca evaluar si las diferentes herramientas de simulación cumplen con las características definidas en Free Software Foundation (2012) para software libre, donde se plantea que el software libre no es necesariamente una herramienta gratuita, sino que, en realidad, lo que la hace libre es la posibilidad que ofrece a los usuarios para editarla, copiarla, ejecutarla, distribuirla, estudiarla y mejorarla. Desde la perspectiva de la *Free Software Fundation* y la *GNU Operation System*, los usuarios de programas libres tienen cuatro *libertades* esenciales:

- ejecutar el programa para cualquier propósito;
- estudiar cómo funciona el programa y adecuarlo a la forma que se desee;
- distribuir el código de un usuario a otro; y
- distribuir el código modificado a otros.

El software libre debe contar con estas cuatro libertades; el hecho de que se cobre algún valor por un programa no inhibe que el programa pueda ser libre o utilizado según las mismas pautas.

Comercial. Se refiere a las licencias que tienen restricciones para el usuario, teniendo en cuenta que su comercialización, costo, duración de uso, edición y libertad de permisos, son controlados y definidos por los propietarios de la herramienta. Este tipo de licencia también es conocida como licencia propietaria, en contrapartida al licenciamiento libre o software libre.

2.1.3. Curva de aprendizaje

Por medio de este parámetro, se busca catalogar el nivel de exigencia de la herramienta para lograr su manejo adecuado. Es un parámetro de suma importancia. Su medición considera diversos aspectos esenciales:

Conocimientos previos. Puesto que el contexto son las redes de comunicación, es de vital importancia conocer aspectos básicos en lo referente a protocolos y estándares de red para enfrentarse a una herramienta potente. Por otra parte, dado que las herramientas de alto rendimiento permiten la introducción de código, es importante tener conceptos básicos de programación.

Uso didáctico de la herramienta. Existe una gran variedad de simuladores que, en términos de interacción, son sumamente amigables con el usuario, ya que permiten

configurar elementos de red de una manera intuitiva, permitiendo observar solo las características básicas del contexto de red; por esta razón, son usados como herramientas didácticas para la iniciación de los estudiantes en los cursos de redes.

De acuerdo con las características planteadas, los niveles definidos para efectuar la medición de este parámetro en cada uno de los simuladores que se presentarán son:

» *Alto*. Herramienta en la que es muy importante una fundamentación solida. Conocimientos avanzados en redes y programación, puesto que el manejo de la herramienta exige programación de los dispositivos por modificación de su código fuente.

» *Medio*. A este nivel se encuentran las herramientas que no requieren una gran cantidad de conocimientos previos en programación o redes y donde toda la carga de aprendizaje se centra en la configuración de los dispositivos de red mediante comandos de consola.

» *Bajo*. Esta categoría corresponde a las herramientas de uso didáctico, en las que la interacción con el usuario hace que la configuración de simulaciones de red se realice de manera intuitiva.

2.1.4. Plataformas que lo soportan

Este parámetro es descriptivo. Lista los diferentes sistemas operativos en los cuales la herramienta puede correr sin ningún problema. Los sistemas operativos que se tendrán en cuenta son: Windows, Linux, Mac OS y Solaris.

2.1.5 Interfaz gráfica

Con este parámetro se busca definir la cercanía que tiene la herramienta con el usuario y las facilidades que le presta. La medida de este parámetro tendrá en cuenta tres rangos:

» *Alto*. Requiere un nivel de programación mínimo, ya que la herramienta tiene la disposición de trabajar desde todas sus perspectivas con una interfaz gráfica.

» *Medio*. Implementa una interfaz gráfica que facilita su uso, pero lo hace de forma limitada; algunas de sus implementaciones deben definirse mediante programación.

» *Bajo*. La herramienta no cuenta con interfaz gráfica o ella no es muy amigable con el usuario, lo cual implica la programación de cada elemento dentro de una simulación para su ejecución final.

2.1.6. Graficación de resultados

Las herramientas de simulación se utilizan para recrear el funcionamiento de la red de la manera más real posible. Para ello es necesario realizar medidas de ciertas variables de red, con el fin de realizar un análisis posterior de los datos y comprender así el comportamiento de la red ante diferentes eventos o posibles configuraciones. Una manera de interpretar y analizar los datos de las variables medidas es graficándolos

de diversas formas. Dependiendo qué tan potente o amigable sea la herramienta de simulación para realizar esta tarea se han definido los siguientes rangos:

» *Buena*. La herramienta posee extensiones o módulos propios para la generación graficas estadísticas las cuales pueden ser manipuladas desde el mismo simulador o ser exportadas a un procesador especializado.

» *Aceptable*. Aquellos simuladores que pueden generar datos estadísticos, pero que necesitan de una herramienta externa, para generarlos, procesarlos adecuadamente y presentar la información ordenada al usuario.

» *Limitada*. Aquellas herramientas que no cuentan con un módulo propio o extensión para la generación de gráficos; la información estadística puede estar representada en archivos de texto que necesitan de herramientas diferentes a la de simulación para su organización y presentación.

2.1.7. Tecnologías y protocolos de niveles 2 y 3 que soporta

Parámetro de índole descriptivo en el cual se listan las tecnologías y protocolos de nivel 2 y 3 del Modelo OSI que soporta. Algunas herramientas no soportan todos los protocolos, lo que hace necesaria su implementación generando código o adaptando componentes preexistentes. De este modo se quiere clasificar los simuladores de acuerdo con la variedad de protocolos que permiten simular, con base en los siguientes criterios:

» *Alto*. Que permite la implementación de gran cantidad de tecnologías/protocolos de red, ya que posee módulos propios con la arquitectura necesaria para que sean soportados y desplegados de manera correcta, con el fin de acercarse a implementaciones reales.

» *Medio*. Simuladores que no permiten realizar implementaciones de un gran número de tecnologías/protocolos, puesto que no poseen los módulos necesarios o en su defecto es necesario modificar el código fuente de alguno de sus módulos para lograr simular el protocolo deseado.

» *Bajo*. Aquellas herramientas que no poseen los módulos de las tecnologías/protocolos desarrollados o en su defecto es necesario conseguir los módulos por separado.

2.1.8. Tráfico que permite modelar

Este es un parámetro muy importante porque permite establecer los tipos de aplicaciones, servicios o protocolos que la herramienta está en capacidad de simular. Esto va muy ligado con el comportamiento de la aplicación, en el sentido de cómo genera los datos a utilizar en la simulación, lo cual está ligado a una distribución estadística que se puede parametrizar en la herramienta, tanto para las aplicaciones que ella dispone, como para aplicaciones que el usuario pueda ajustar dependiendo de sus necesidades. Este aspecto puede ser medido de acuerdo con los siguientes criterios:

» *Alto*. Herramientas con la capacidad de generar tráfico de gran variedad de aplicaciones de acuerdo con diferentes distribuciones estadísticas y la capacidad de recibir la inyección de tráfico proveniente de analizadores de tráfico (*Sniffers*) para realizar análisis estadísticos.

» *Medio*. Herramientas que permiten generar tráfico de las aplicaciones más comunes, donde las distribuciones estadísticas se pueden configurar de manera básica.

» *Nulo*. Herramientas en las que no es posible configurar distribuciones de tráfico que permitan análisis académicos profundos o, en su defecto, que no cuentan con módulos o extensiones para realizar esta labor.

2.2. Herramientas de simulación

Una vez definidos los parámetros de comparación, se procede a su evaluación en las diferentes herramientas de simulación, con el fin de establecer un cuadro comparativo que le permita al usuario visualizar, de una manera clara, las ventajas y desventajas de elegir una herramienta u otra.

2.2.1 NS-2

Como presenta Mahasweta (2008), NS2 es una de las herramientas de código abierto más confiables y de mayor uso para la implementación de proyectos en simulación, tal como se muestra en los trabajos de Mehta, Ullah, Kabir, Sultana, y Sup (2009), Bateman y Bhatti (2010) y Fan y Taoshen (2009), ya que sus posibilidades de uso, su disponibilidad al público y, adicional a esto, sus características, permiten que una gran cantidad de usuarios puedan realizar diversos tipos de trabajo sobre ella.

En relación con la libertad de disposición que tiene el simulador, en Bateman y Bhatti (2010), se argumenta la disponibilidad del código fuente tanto para su inspección y modificación, como la libertad para la aplicación por parte de cualquier usuario. Esto ha promovido la generación de módulos, al interior de la comunidad, que permiten la creación de nuevos protocolos y sistemas para ser simulados. A su vez, su característica de licencia libre ha permitido expresar tanto las preocupaciones sobre la herramienta, como sobre su influencia positiva.

Para enfocar una característica más del simulador NS2, es preciso divisar los dos niveles de simulación posibles, explicados por Fan y Taoshen (2009) y Qun y Wang-Jun (2008); uno se basa en la configuración y construcción de Otcl –en referencia a la programación orientada a *O*bjetos con extensión tcl– opción en la cual, es posible utilizar algunos elementos de redes existentes en la herramienta, para realizar una simulación escribiendo scripts en Otcl, sin la necesidad de modificar NS2; el otro, se refiere a la programación en lenguaje C++ y Otcl, utilizada cuando los módulos requeridos para las implementaciones no existen, lo que obliga a actualizar la herramienta con los nuevos elementos y, en ciertas circunstancias, agregar una clase

programada en C++ y una clase en Otcl y luego programar un script Otcl para la implementación de la simulación; esto, genera la necesidad de un gran conocimiento previo para el uso de la herramienta. Adicionalmente, como muestran Marquez, Placido y Sampaio (2009), es necesario, para la correcta ejecución de la herramienta, tener experiencia, lo que conlleva a la necesidad de usar la herramienta por largos periodos de tiempo para lograr su implementación eficaz. NS2, en consecuencia, se cataloga con un nivel alto en su curva de aprendizaje.

NS2 es un simulador con limitadas características gráficas. Como se planteó, está orientado a programación (Márquez, Placido, & Sampaio, 2009), aunque se han hecho intentos de simulación gráfica para aprovechar la potencia de la herramienta, intentando graficar las rutas de los paquetes, como explican Qun y Wang-Jun (2008), implementando la herramienta NAM. Para realizar la simulación de un escenario como el que se plantea en la página oficial de la herramienta –y se observa en la Figura 2– se debe realizar un *script* similar al fragmento que presenta la Figura 3. Por esta razón, NS2 es una herramienta catalogada con un bajo nivel de características gráficas.

Aunque el software no contiene una herramienta que retorne gráficas de una manera eficiente, existen otras herramientas que permiten hacerlo, lo cual minimiza el problema y permite aprovechar todos los valores que arroja el simulador de las tecnologías manejadas a nivel 2 y 3 de OSI, como lo describen las prácticas presentadas por Shin, Jang y Kim (2009), quienes definen tecnologías que la herramienta puede simular (i.e.,TCP/IP, UDP, FTP, RTP, SRN, GPRS, mobile IPv6, RSRV, MPLS), así como redes Ah Hoc, WLAN, Mobile-IP, UMTS y Wireless. Un aspecto a favor de la herramienta de simulación según ISI (2011) y Bateman y Bhatti (2010) es su capacidad

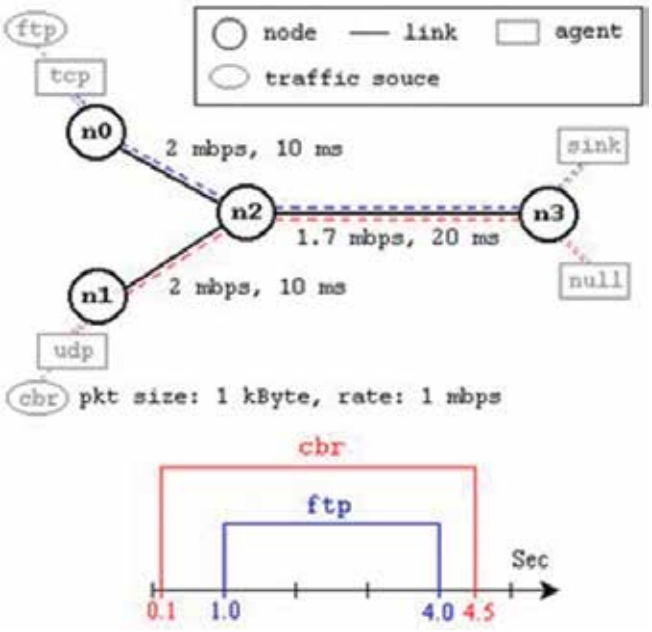


Figura 2. Escenario - Simulación en NS-2

```
#Create a simulator object
set ns [new Simulator]

#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red

#Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Define a 'finish' procedure
proc finish () {
        global ns nf
        $ns flush-trace
        #Close the NAM trace file
        close $nf
        #Execute NAM on the trace file
        exec nam out.nam &
        exit 0
}
```

Figura 3. Fragmento de Script - Simulación en NS-2

de instalación en los diversos entornos y plataformas (i.e., FreeBSD, Linux, SunOs, Solaris, Windows y Mac OS X).

2.2.2. NC-TUNS

National Chiao Tung University Network Simulator, es un simulador altamente usado como lo presentan NSTU (2010) y Wang & Chou (2009); en la comunidad investigativa y en la documentación de la IEEE Xplore se puede encontrar gran diversidad de *papers* realizados sobre este simulador. La Figura 4 muestra la interfaz gráfica de la versión 4 del software.

La herramienta de simulación NCTUns se comercializa en la actualidad con el nombre EstiNet 7.0. Esta versión implementa todas las mejores aplicaciones de NCTUns 6.0. Esta herramienta es open-source y se plantea como uno de los mejores simuladores en las comunidades networking en cuanto a su alta fiabilidad, como se presenta en Wang, Chou, Lin, & Huang (2010).

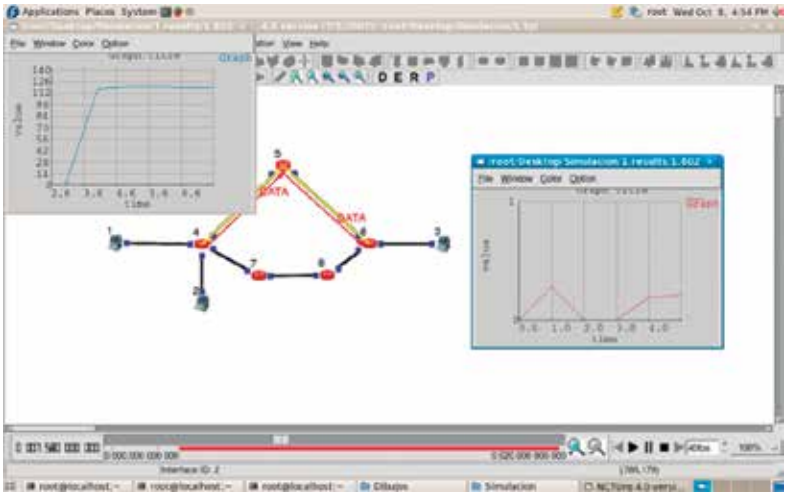


Figura 4. Interfaz gráfica NCTUns

Wang, Wang, Li, y Lau (2011) describen el alto potencial que se encuentra en la herramienta, ya que utilizan la implementación de VANET y describen como la herramienta permite, por medio de una interfaz gráfica, hacer un esquema de simulación muy fácilmente. Por otro lado, Khan, Zaman, y Reddy (2008) indican que el software de simulación NCTUns provee una interfaz de usuario integrada y profesional; por todo ello, la herramienta es catalogada con un alto nivel de interfaz gráfica.

Respecto del manejo que requiere la herramienta de simulación, se debe precisar que es necesario un conocimiento previo de tecnologías de red, ya que las implementaciones a las que está enfocada se orientan a comunicaciones en movimiento (e.g., VANETs y MANETs).

Por otro lado, la herramienta de simulación NCTUns, con todas sus propiedades, tiene una limitante: sólo se puede implementar en Linux (Wang, Chou, Lin, & Huang, 2010), y esto ocurre, como se muestra en Belzarena y González-Barbone (2006), porque el simulador aprovecha la pila de protocolos TCP/IP de Linux para lograr alta confiabilidad en la respuesta de las simulaciones.

Gracias a la facilidad de utilización y al gran desarrollo que ha tenido la herramienta, según Cruz, Camara, y Guardia (2009), Wang y Chou (2009), Khan, Zaman, Reddy, Reddy, y Harsha (2008), y Wang y Bao (2005)], el software tiene la posibilidad de simular Wimax, VANET, MANET, Internet, Wireless LANS, GPRS Networks, Optical Networks, Personal AP, Real TCP/IP, UDP/IP. También, tiene la característica especial de simular CBR TCP, aplicaciones reales (generando datos reales) como presenta Abusubaih (2010).

2.2.3. OPNET Modeler

Esta es una herramienta de simulación que permite flexibilidad y escalabilidad en modelos jerárquicos – los cuales representan estructuras de redes reales– y puede ser implementada en sistemas operativos tipo Unix o Windows. Dichos modelos, están divididos en tres dominios, denominados Red, Nodo y Procesos, los cuales están escritos en C++ y poseen su propio editor, como explican Guo y Zeng (2009).

Gracias a sus numerosas ventajas y a su poderosa interfaz gráfica (Figura 5), su uso, por parte de grupos académicos es alto, puesto que la manera de simular es muy intuitiva; sin embargo, necesita de gran cantidad de conocimientos previos en redes y programación, por lo que su curva de aprendizaje es alta. De otro lado, aunque la licencia del simulador es comercial, existe una versión de este software que no genera cargos económicos, aunque su uso está limitado a actividades netamente académicas.

Puesto que OPNET proporciona los mecanismos necesarios para el desarrollo fluido de una simulación, permitiendo arrastrar componentes para conformar topologías de red, Xia, Li, y Wan (2008) configuraron una solución de MPLS VPN activando los diferentes parámetros desde un cuadro de diálogo que ofrece una lista de ellos. Adicionalmente Xia, Li, y Wan (2008) hicieron uso de la herramienta de

Figura 5. Interfaz gráfica OPNET (Aranud, 2012)

depuración de procesos, la cual representa un componente importante para el trabajo con OPNET.

OPNET cuenta con características que permiten el estudio de los resultados de simulación. Como específica Chang (1999), el servicio de análisis consiste en mostrar la información de los resultados de simulación en forma de grafos, presentados dentro de áreas rectangulares llamadas paneles de análisis, los cuales tienen propósitos específicos para presentar o transformar la información. Estos paneles, básicamente, contienen gráficas cartesianas que describen la relación entre diferentes variables, como lo efectuaron para su implementación David, Sanmartin, y Marquez (2010).

En cuanto a las tecnologías de nivel 2 y 3, hay varias que la herramienta soporta. Guo y Zeng (2009 usaron la herramienta para comparar el desempeño de varios algoritmos de encolamiento (i.e., WF2Q, WFQ, PGPS y SCFQ) y Hammoodi, Stewart, Kocian, y McMeekin (2009) realizaron las implementaciones de tecnologías WSN (*Wireless Network Sensor*), aunque el simulador no soporta modelos de energía, importantes para este tipo de redes. Adicionalmente, en su trabajo, Kulgachev y Jasani (2010), evalúan el desempeño del protocolo RTS/CTS en redes con tecnologías como 802.11g y 802.11b.

2.2.4. OMNET++

Esta es una herramienta de simulación de eventos discretos. Tal como especifican Varga y Hornig (2008), la motivación para el desarrollo de OMNET fue producir una herramienta *open-source* poderosa que pudiera ser usada por la academia, en educación o investigación, y que fuera una alternativa a las herramientas comerciales, para ser implementada en sistemas tipo Unix, MAC y Windows; incluso, haciendo uso de Cygwin o el compilador de Microsoft Visual C++, es posible portar la herramienta en otros sistemas con menor esfuerzo.

Esta herramienta se conforma por módulos escritos en C++ que se comunican entre sí a través de mensajes, donde módulos simples pueden conformar módulos compuestos y los niveles jerárquicos no tienen límites.

Este software ha estado disponible desde septiembre de 1997 y, fuera de las descargas anónimas, alrededor de cuarenta universidades a nivel mundial han obtenido la herramienta para su uso, indicando áreas de aplicación que van desde tecnologías móviles, inalámbricas hasta ATM y redes ópticas, realizando proyectos relacionados con el desarrollo de modelos de TCP/IP en la universidad Karlsruhe (Alemania).

Se debe tener en cuenta que, a pesar de que la herramienta de simulación presenta muchos beneficios, los módulos que la conforman no están del todo desarrollados, lo cual implica que los programadores deben modificar el código existente o realizar implementaciones de módulos nuevos para cubrir componentes de red que aún no están especificados dentro del paquete OMNET; esto hace que la curva de aprendizaje de la herramienta sea alta, pues modificar el código fuente de los componentes de red no es una tarea sencilla.

La interfaz gráfica que ofrece la herramienta (ver Figura 6), como ya se mencionó, presenta características de depuración superiores al de otras herramientas; adicional a esto, Lessmann, Janacik, Lachev, y Orfanus (2008) destacan que OMNET es un simulador con visualización en línea, lo que permite que los usuarios puedan pausar la simulación e inspeccionar –o cambiar– los valores de los módulos que conforman una red, incluso cambiar la apariencia de los nodos y datos que arroja como resultado, para ser analizados con herramientas estadísticas que se pueden encontrar en el repositorio de OMNET en la Web. Sin embargo, si el módulo no está desarrollado en alguna librería de OMNET, será necesario generarlo mediante código; la interfaz gráfica, en ese caso, se reduce a un editor de texto.

Respecto de la graficación de resultados, el análisis realizado por Sameh, Wagih, y Salama (2010) permite apreciar que las opciones de graficación de parámetros son adecuadas para mostrar los resultados obtenidos de manera clara.

Con relación a las tecnologías de nivel 2 y nivel 3 que soporta la herramienta, diversas publicaciones muestran diferentes protocolos e implementaciones realizadas. En primera instancia, Rhee, Cho, Xianshu, y Han (2009) presentan el diseño de un esquema de enrutamiento específico que pueda proporcionar los niveles de calidad de servicio [QoS] con base en los acuerdos de servicio y administración de recursos con el suscriptor. Por otra parte, El-Darieby, Petriu y Rolia (2012), proponen un nuevo protocolo de distribución jerárquica para proporcionar túneles basados en MPLS, permitiendo mecanismos para QoS e ingeniería de tráfico. Finalmente, Zhu, Dreibholz Rathgeb y Zhou (2008), presentan el diseño de un dispositivo de QoS para descartar paquetes de flujos que no se vean *tan* alterados por la pérdida de los mismos, evitando así que el descarte se haga aleatoriamente, sobre cualquier flujo, inyectando para esto tráfico MPEG, H323 y MP3, midiendo el retardo de la transmisión de cada

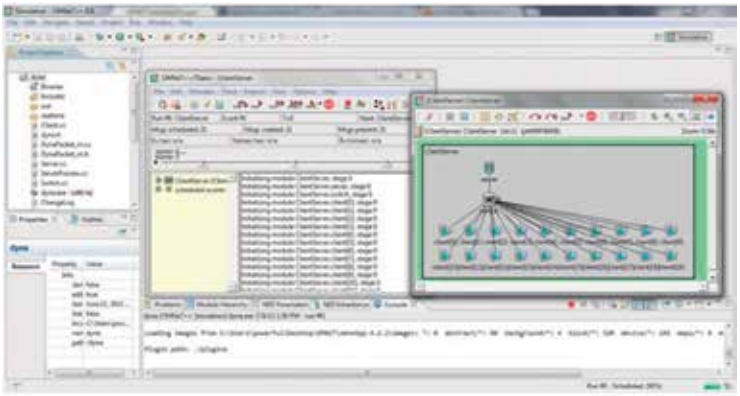


Figura 6. Interfaz gráfica OMNET++

flujo y la tasa de paquetes perdidos, generando, por medio de gráficas, estadísticas detalladas.

Adicionalmente, Chen, Li y Chen (2010) presentan un caso interesante de televisión digital para usuarios residenciales con conexiones a Internet de manera asíncrona y usando NAT, con el fin de que compartan datos directamente con otro. La simulación realizada en OMNET contó con 1000 *peers*, los cuales, al iniciar, se unieron a un pseudo sistema P2PTV en tan sólo 5 minutos, obteniendo resultados que mostraron que la implementación incrementa la tasa de compartimiento entre *peers* que usan NAT en un 50 %, lo cual hace más escalable la arquitectura y reduce alrededor de 20 a 24 % el tiempo de descarga de paquetes.

2.2.5. NS-3

Esta herramienta es un simulador de eventos discretos de red, que tiene como principales objetivos lograr un mayor énfasis en los niveles 2 y 4 del OSI y que su uso sea principalmente educativo; cabe mencionar que no se encuentran publicaciones referentes a la herramienta tan fácilmente. En un principio la compatibilidad con NS-2 no es un objetivo del proyecto, por lo que NS-3 no es una actualización de NS-2 sino un proyecto diferente. Esta herramienta es *open-source* y permite la inclusión de otro software *open-source*; es escalable, modular y emulador (ns-3, 2012).

En relación con la usabilidad y adaptabilidad, NS-3 está escrito solo en C++ lo que lo hace más fácil de depurar (Brugge, Paquereau y Heegaard, 2010). Las plataformas que soporta esta herramienta, tanto de escritorio como de servidor, son: i386, x86-64, Linux, OS X, freeBSD, Solaris y Windows.

La interfaz gráfica de esta herramienta (Figura 7), como indican Henderson, Roy, Floyd, y Riley (2006), soporta algunas formas de animación visual para todo o parte de una simulación. Es una herramienta útil para depurar y mostrar la simulación a terceros. Usa NAM para la animación del escenario de red, que es parte de NS-2 desde el principio, y ha logrado hacer un extendido en ambientes Wireless. Para la generación de resultados, Henderson et al., (2006) indican que NS-3 proporciona objetos de

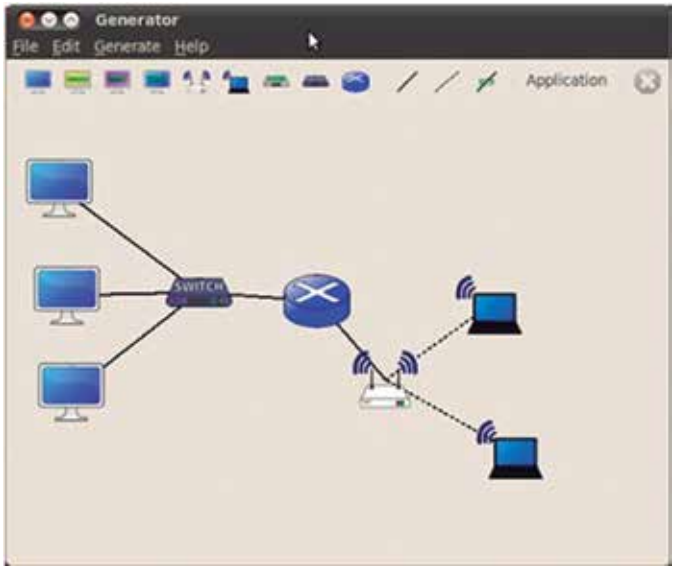


Figura 7. Interfaz gráfica NS3 (ns3, 2010)

soporte para facilitar la recolección de datos durante la ejecución de la simulación, una característica también incluida desde NS-2. NS-3 se extiende incluyendo objetos para histogramas, rastreo, funciones de distribución y secuencias versus tiempo.

Respecto a las tecnologías, NS-3 permite simulaciones sobre IPv4 e IPv6, Wireless (e.g., WiFi, Wimax) y algoritmos de enrutamiento (e.g., OLSR y AODV) (Wang & Liu, 2010).

2.2.6. GNS3

Software gráfico de simulación de red que permite la emulación de redes complejas. Trabaja de manera similar a reconocidos emuladores como Vmware o Virtual Box, emulando los IOS de los dispositivos de interconectividad de Cisco. *Dynamips*, el *core* de este software, es el que permite esta tarea. La herramienta corre en ambientes Linux, Windows y Mac.

GNS3 no toma el lugar de un *router* real, pero es una herramienta para el aprendizaje y preparación para certificaciones Cisco como CCNA, CCNP y CCIE. Su interfaz gráfica es muy intuitiva para el usuario, como se observa en la Figura 8.

La licencia de esta herramienta es de libre descarga, pero necesita de las imágenes de los dispositivos Cisco para emular su comportamiento, imágenes que se deben adquirir directamente con el fabricante (GNS3, 2012b).

En cuanto a las tecnologías que soporta el software, el caso que presentan Djenane, Benaouda y Harous (2009) muestra la configuración de la red de Algerie Telecom para la simulación de VPN sobre MPLS usando IOS de enrutadores Cisco, advirtiendo que, de no usar este tipo de dispositivos en la implementación real, el principio de funcionamiento es básicamente el mismo. La configuración de los dispositivos incluye los protocolos RIP, BGP, OSPF.

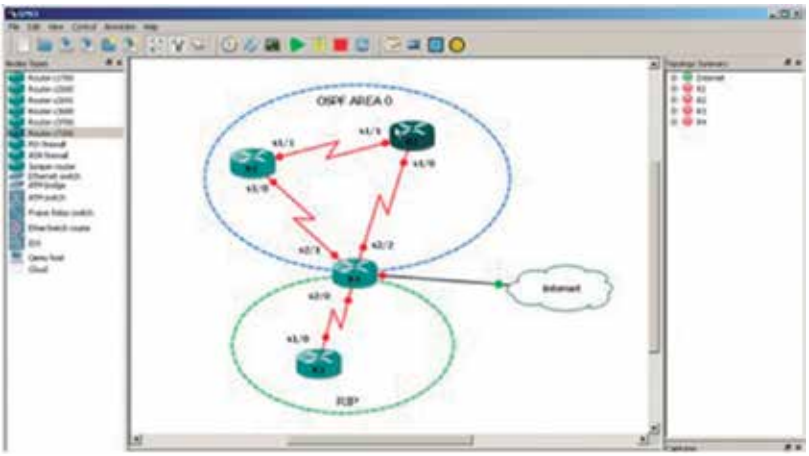


Figura 8. Interfaz gráfica GNS3 (GNS3, 2012a)

2.3. Recomendaciones para la elección de la herramienta

En la realización de proyectos donde esté involucrado el uso de simuladores de red, es de gran importancia invertir tiempo y estudio en la elección de la herramienta adecuada; para hacerlo se debe tener claro cuáles son las características más relevantes que dicho software debe proporcionar, ya que de una adecuada selección depende la efectividad de la realización de las pruebas en el proyecto. La Tabla 4 muestra la comparación entre las diferentes herramientas de simulación, teniendo en cuenta los parámetros definidos y permite guiar al usuario en el proceso de selección y toma de la decisión que mejor se ajuste a sus requerimientos.

Aunque todos los parámetros presentados en la Tabla 4 son importantes en el proceso de selección, el tipo de licencia de las herramientas es determinante, puesto que, aquellas con licencia libre generalmente son la primera opción para los usuarios que no tienen la posibilidad de costear una. Sin embargo, el hecho de que una herramienta posea licencia libre se debe valorar con respecto a la documentación y soporte, para asegurar el apoyo de la comunidad académica, acudiendo a la revisión de errores o problemas a los que otros autores se han enfrentado y han corregido. Diferente a lo que ocurre con los simuladores de licencia comercial, que brindan el soporte durante el tiempo de validez de la licencia.

Otro parámetro importante a tener en cuenta es el uso específico que se le ha dado a cada simulador. Aunque la mayoría son capaces de simular diferentes tipos de redes, las referencias en un tema específico hacen que la cantidad de información, módulos y desarrollo en el tema sea mayor, lo cual permite establecer, en un primer análisis, si la herramienta se ajusta a los requerimientos del usuario o no.

Finalmente, lo referente a la generación y presentación de resultados estadísticos en una herramienta de simulación es vital, puesto que, con base en ellos, el investigador puede hacer aportes a las comunidades investigativa o académica. Algunas

Tabla 4. Parametrización de las herramientas de simulación

	OPNET	OMNET	NS-3	GNS3	NS-2	NC-TUNS
Uso investigativo	Alto	Alto	Medio	Bajo	Alto	Alto
Tipo de licencia	Comercial	Libre	Libre	Libre/ Comercial	Libre	Libre
Curva de aprendizaje	Alto	Alto	Alto	Bajo	Alto	Alto
Plataformas que soporta	Windows, Unix	Windows, Unix	Windows, Mac, Unix	Linux, Mac Windows.	Windows, Mac, Unix.	Linux
Interfaz gráfica	Alto	Medio	Medio	Alto	Bajo	Alto
Graficación de resultados	Buena	Aceptable	Aceptable	Limitada	No tiene	Aceptable
Tecnologías de nivel 2 y nivel 3 que soporta	Alto	Alto	Medio	Bajo	Alto	Alto
Tráfico que permite modelar	Alto	Medio	Medio	Nulo	Alto	Alto

herramientas poseen mejores especificaciones que otras en este aspecto y permiten generar este tipo de resultados sin la necesidad de usar aplicaciones externas o realizar procesamientos extensos para la presentación de los datos.

Capítulo 3

Instalación de OMNET

En este capítulo se explica la instalación de OMNET y del INET Framework en ambiente Windows 7, para un sistema operativo de 64 bits. Para un ambiente diferente –o para mayor información– se recomienda consultar el manual de instalación de OMNeT ++ (Varga, 2011), publicado en http://omnetpp.org/doc/omnetpp/InstallGuide.pdf (actualmente la versión disponible es la 4.2.2).

3.1. Instalación de OMNeT++

Previo a la instalación del simulador es necesario tener instalado *Java Runtime Environment* [JRE]. La guía de instalación oficial recomienda descargar la versión 5.0 de Java o una posterior. Adicionalmente al instalar OMNET en Windows 7 de 64 bits, fue necesario desinstalar cualquier Java VM del computador. Por esta razón, se recomienda descargar el JRE para 32 bits.

Para iniciar, se debe descargar OMNeT ++ de la página oficial y verificar que sea la distribución para Windows. Hecho esto, se procede a descomprimir el archivo.

El archivo se puede ubicar en cualquier lugar; sin embargo, se debe tener en cuenta que la carpeta en la que se descomprima el archivo no puede tener un nombre con espacios entre las palabras (e.g., Archivos de Programa). En este caso la ubicación quedó dada por la ruta:

```
C:/Users/Mon.Investigación/Documents/Simulador_OMNeT
```

Una vez descomprimido el archivo, en la carpeta se encontrará una serie de carpetas y archivos como los que muestra la Figura 9.

El siguiente paso es iniciar el archivo mingwenv.cmd Un clic sobre él desplegará una consola como la que muestra la Figura 10.

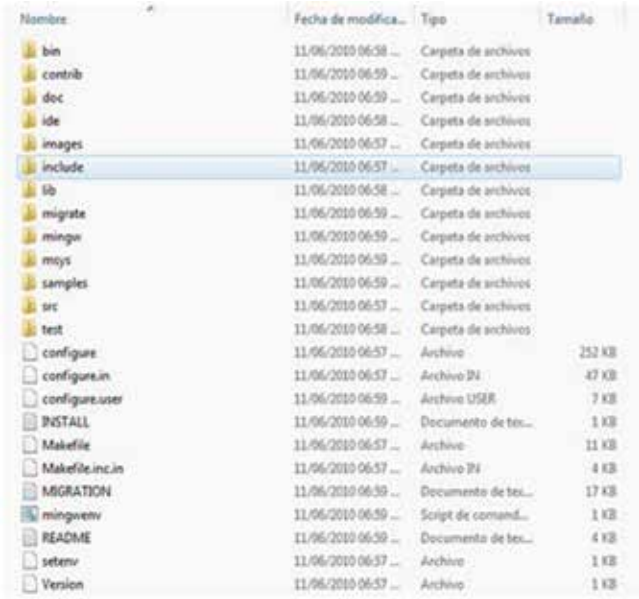


Figura 9. Contenido de la carpeta *Simulador_OMNeT*

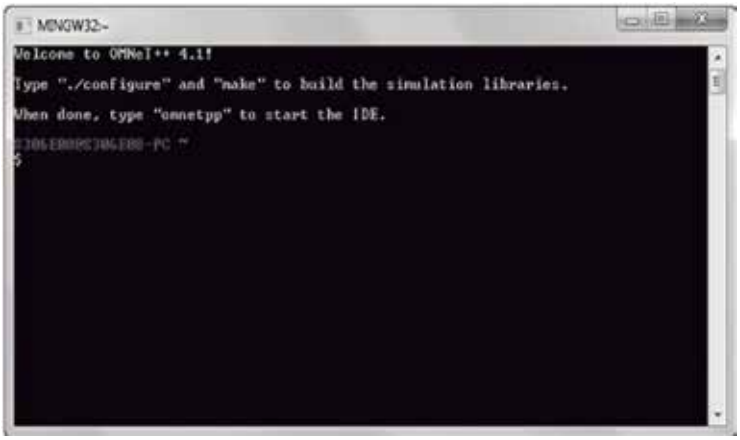


Figura 10. Ventana - Consola de MSYS

La guía de instalación oficial (Varga, 2011) recomienda analizar el archivo configure. user, para determinar si se requiere hacer modificaciones en el archivo; en este caso no se va a realizar ninguna. Para terminar la configuración del simulador de manera correcta, se requiere digitar una serie de comandos para construir los archivos que van a servir para depurar y liberar procesos. Con este paso se construye la librería de simulación. Las Figuras 11 y 12 presentan los comandos a digitar.

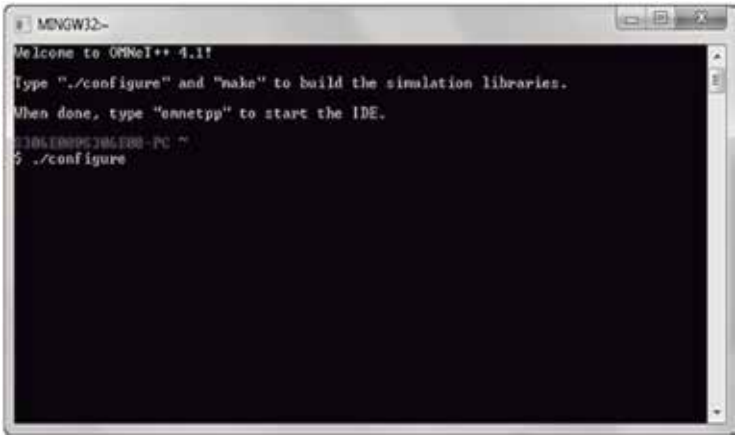


Figura 11. Ventana - Consola de MSYS construcción de las librerías de simulación (1 de 3)

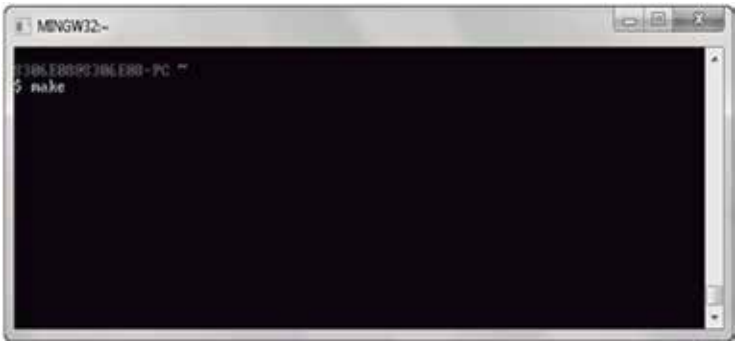


Figura 12. Ventana - Consola de MSYS construcción de las librerías de simulación (2 de 3)

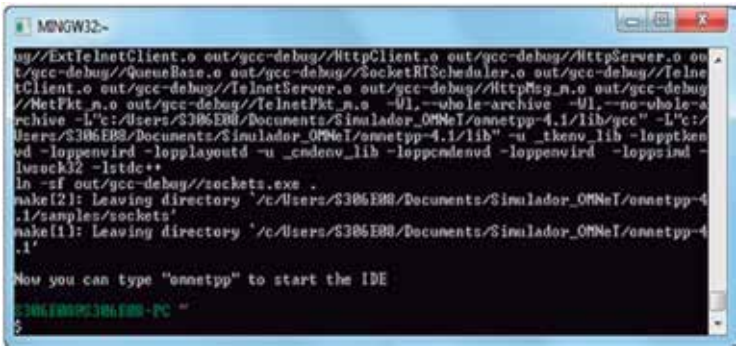


Figura 13. Ventana - Consola de MSYS construcción de las librerías de simulación (3 de 3)

Al finalizar la ejecución de estos dos comandos, la consola va a presentar un resultado como el que muestra la Figura 13; esto indica la configuración ha sido exitosa y que se puede iniciar el IDE de OMNeT++.

Antes de continuar se puede verificar la correcta instalación de OMNeT ++ ejecutando cada uno de sus ejemplos; en este caso, se ejecuta el ejemplo Dyna (sugerido por la guía de instalación oficial), desde la consola, como se muestra en las Figuras 14 y 15.

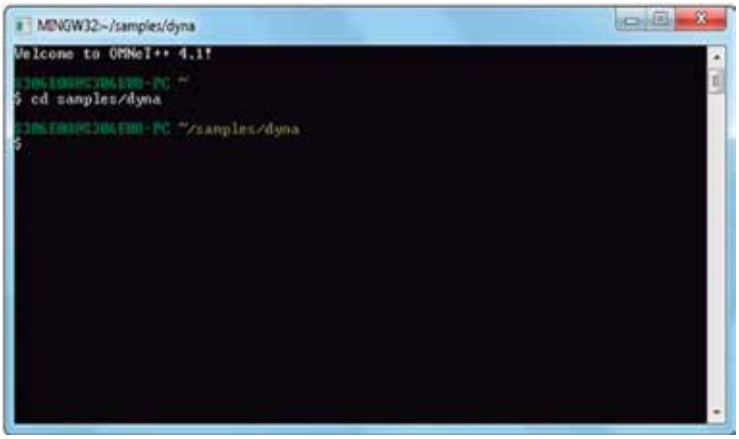


Figura 14. Ventana - Consola de ejecución del ejemplo Dyna (1 de 2)

Figura 15. Ventana - Consola de ejecución del ejemplo Dyna (2x de 2)

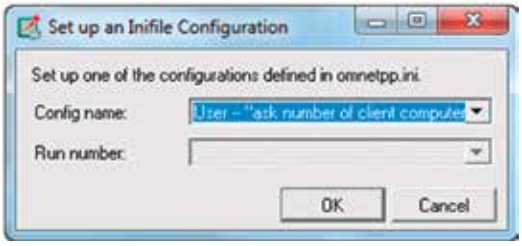


Figura 16. Ventana - *Set up an inifile configuration* (1 de 2)

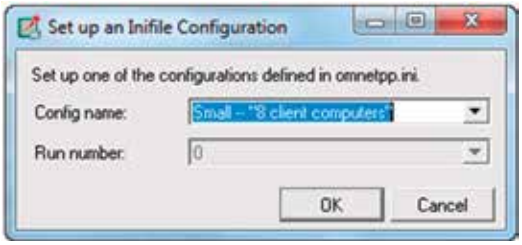


Figura 17. Ventana - *Set up an inifile configuration* (2 de 2)

Una vez se haya digitado el comando anterior se presiona `enter`, lo que desplegará en pantalla una ventana como la que muestra la Figura 16.

Para efectos del ejemplo, en la casilla *Config name* se selecciona la opción Small -"8 client computers", como muestra la Figura 17.

Al hacerlo, se despliegan dos nuevas ventanas en la pantalla como las que presentan las Figuras 18 y 19.

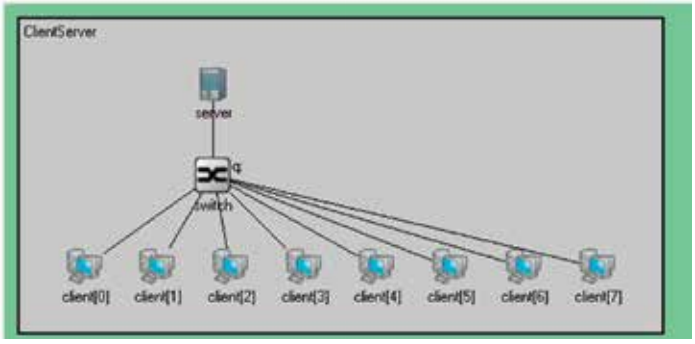


Figura 18. Interfaz gráfica de OMNet++

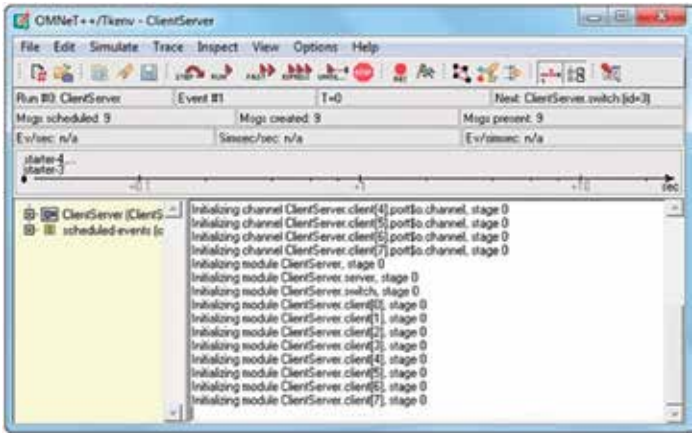


Figura 19. Ventana - Información el proceso de simulación

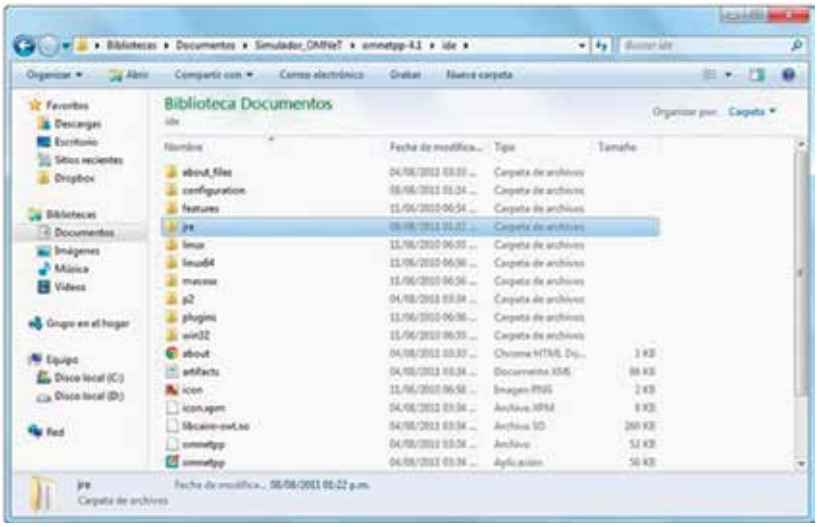


Figura 20. Ubicación de creación de la carpeta *jre*

No se explica en detalle sus componentes, ya que esta parte tiene como único objetivo enseñar la instalación de la herramienta y la verificación de su correcta instalación.

Antes de ver el IDE del OMNeT++, se debe crear manualmente una carpeta en la ubicación que muestra la Figura 20.

En esta carpeta se debe instalar el JRE que se descargó en los pasos anteriores; hecho esto, la carpeta *jre* debe contener los archivos que muestra la Figura 21.

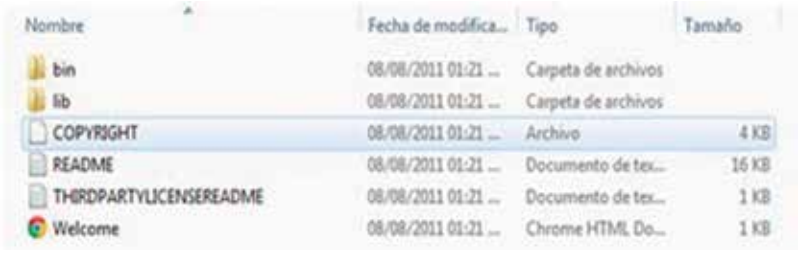


Figura 21. Contenido de la carpeta *jre*

Figura 22. Ventana - Comando de ejecución del IDE de OMNet++

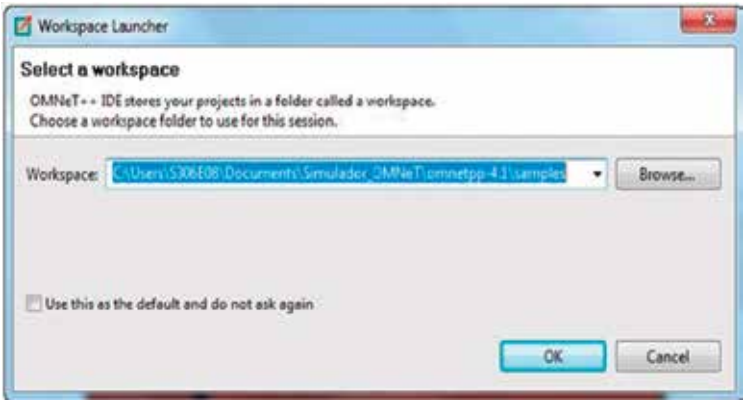


Figura 23. Ventana - Workspace launcher OMNet++

Este comando desplegará en pantalla una ventana como la que muestra la Figura 23, que presenta la ubicación del espacio de trabajo.

Una vez se seleccione el espacio de trabajo, el programa comienza la carga de librerías (Figura 24) y muestra la ventana de bienvenida (Figura 25).

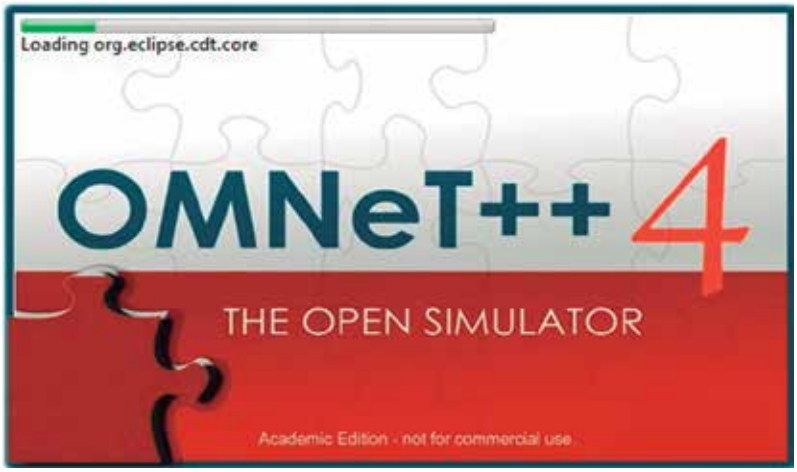


Figura 24. Ventana - Loading

Figura 25. Ventana - Bienvenida al simulador OMNet++

Para evitar tener que iniciar el IDE por consola cada vez que se requiera, se puede crear un acceso directo.

En este punto ya se tiene instalada la aplicación; sin embargo, es necesario instalar unos modelos de simulación que servirán como base para desarrollar las simulaciones introductorias para empezar a trabajar con la herramienta. El modelo que se va usar se denomina INET Framework (su instalación se detalla en la siguiente sección), que cuenta con modelos de IP, TCP, UDP, Ethernet, MPLS y otros protocolos e incluye un soporte para simulaciones móviles e inalámbricas (Piotr, Stankiewicz, Cholda, & Jajszczyk, 2011).

3.2. Instalación de INET Framework

El proceso inicia con la descarga del paquete INET Framework, lo que se puede hacer desde http://inet.omnetpp.org/index.php?n=Main.Download (INET Framework, 2012)

Una vez descargado, el paquete se debe descomprimir, preferiblemente, en la ubicación dónde se ubicó el OMNet++, como muestra la Figura 26.

Hecho esto, se procede a abrir el IDE y a importar el INET Framework, como se ilustra en las Figuras 27, 28, 29, 30, 31 y 32.

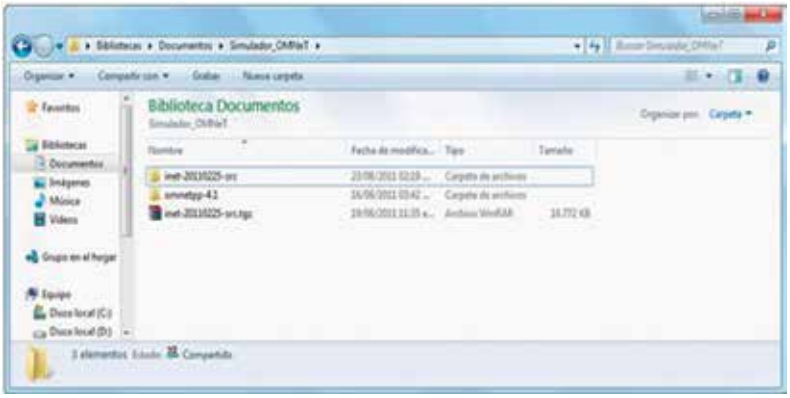


Figura 26. Ventana - Documentos

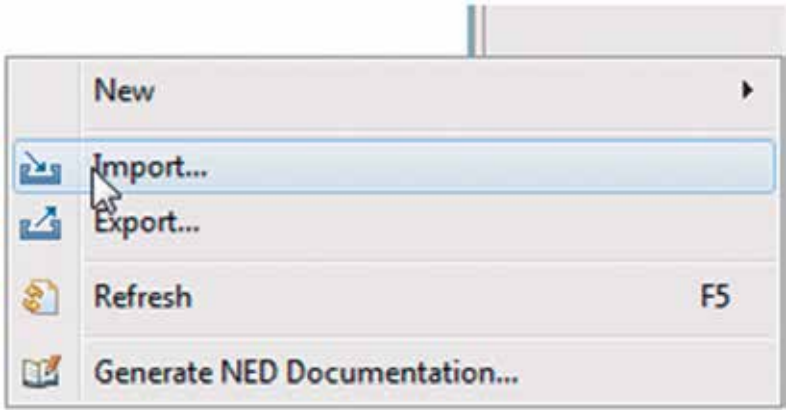


Figura 27. Ventana - Importación de INET Framework (1 de 4)

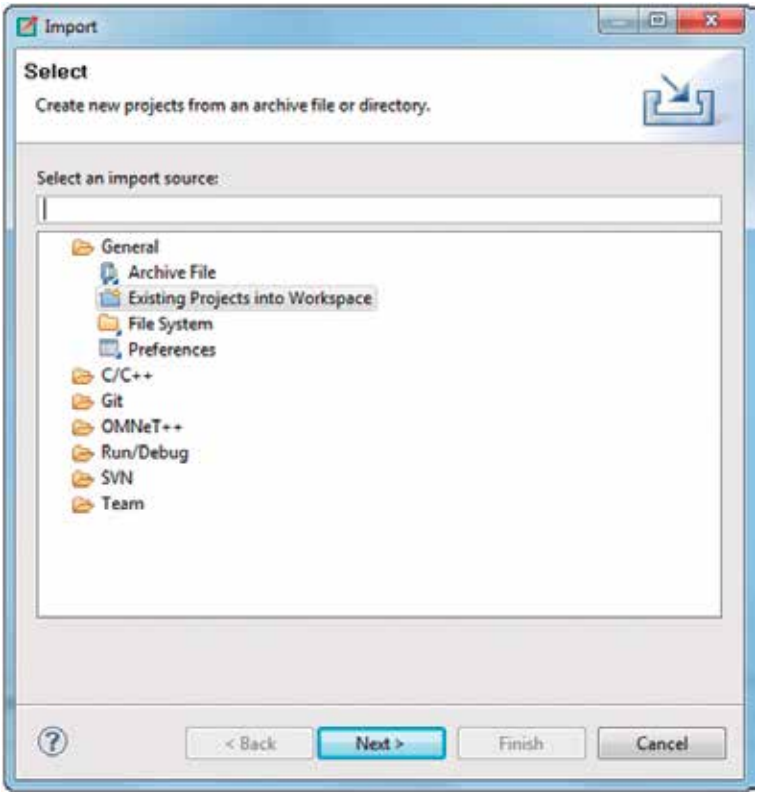


Figura 28. Ventana - Importación de INET Framework (2 de 4)

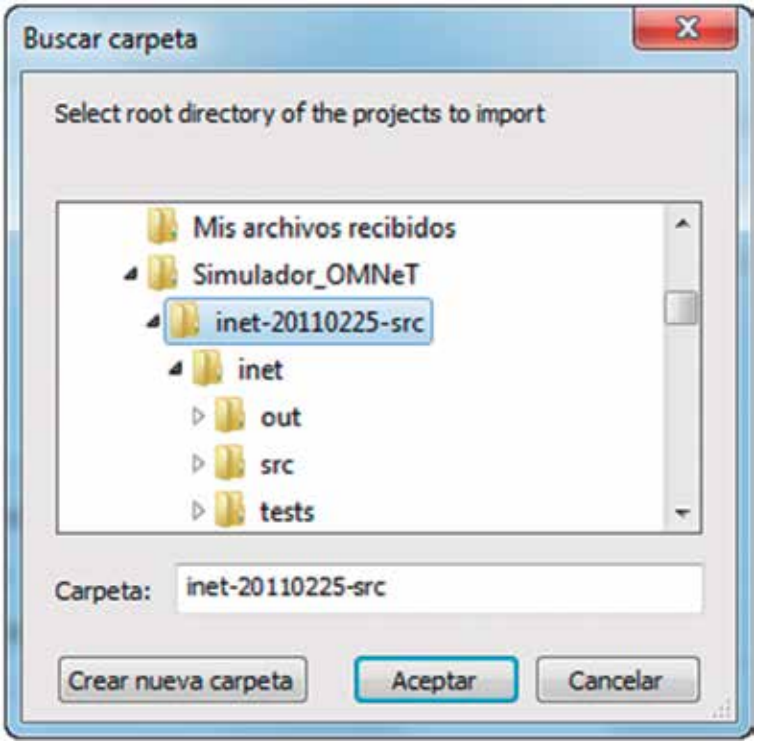


Figura 29. Ventana - Importación de INET Framework (3 de 4)

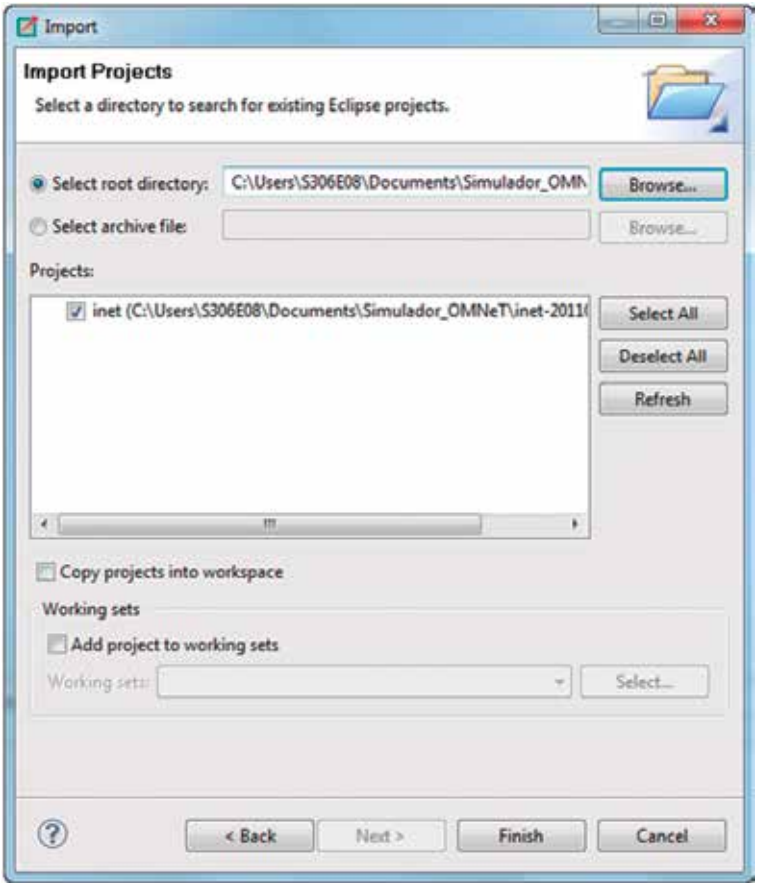


Figura 30. Ventana - Importación de INET Framework (4 de 4)

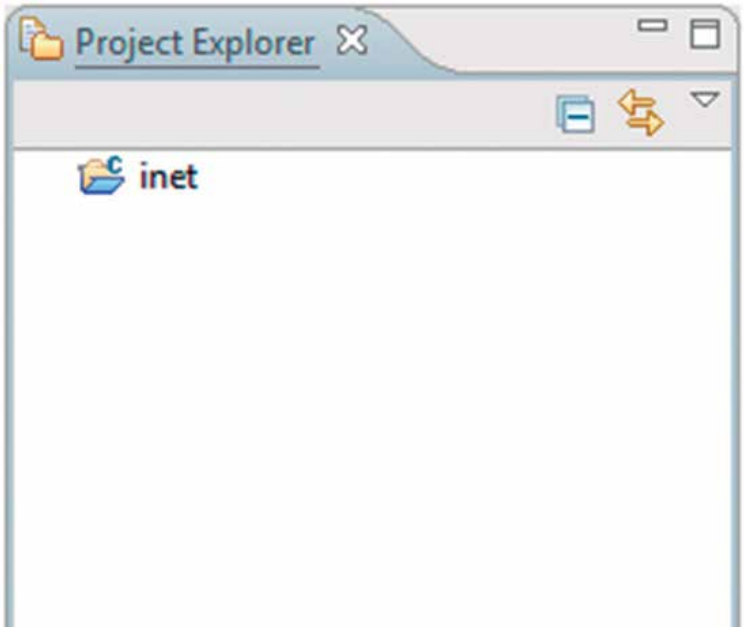


Figura 31. Project Explorer con el paquete INET OK

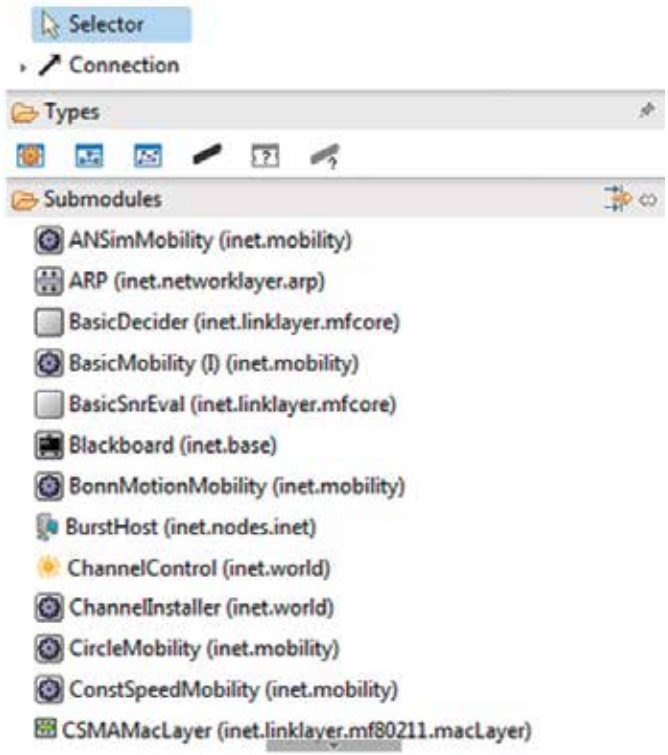


Figura 32. Ventana - Utilidades con los módulos INET cargados

Capítulo 4

Lenguajes de OMNET++

Los programas de simulación son desarrollados en diferentes lenguajes, los cuales le permiten al programador definir los comportamientos de las simulaciones. En OMNeT++ se utilizan dos tipos de lenguaje: el primero de ellos, es desarrollado para implementar la parte gráfica de OMNeT++, su nombre es NED; el segundo, es utilizado para desarrollar la parte lógica del proyecto, C++. En este capítulo, se explican ambos lenguajes y algunas características necesarias para lograr una implementación en OMNeT++.

El lenguaje NED es una de las principales características de OMNeT++, ya que es quien le permite al usuario describir la estructura del modelo de simulación; en otras palabras, el lenguaje NED se utiliza para la descripción de las redes. Con este grupo de reglas sintácticas y semánticas es posible declarar módulos simples, los cuales representan elementos de la red, y módulos compuestos, que son grupos de módulos simples que trabajan de manera conjunta. También es posible referirse a la red como un módulo compuesto.

A su vez, los canales no son módulos, sino otro tipo de componente, y pueden ser utilizados para conectar módulos simples dentro de módulos compuestos.

4.1. Características del lenguaje NED

El lenguaje NED permite la escalabilidad de las redes simuladas, gracias a algunas de sus características (OMNeT++ Community, 2011a), como son:

Jerarquías. OMNeT++ es una herramienta que se centra en la simplicidad para crear los componentes; por esto, cada componente que es demasiado complejo puede ser dividido en varios módulos simples y ser representado como un módulo compuesto, lo que le permite al programador, editar con más facilidad cada uno de los elementos a utilizar y hacer modificaciones de una forma más sencilla.

Basado en componentes. Tanto los módulos simples como los compuestos son, de manera innata, reusables, lo que permite utilizar módulos desarrollados en cualquier otra parte de los proyectos, sin la necesidad de tener que copiar códigos. Gracias a esto, todos los programas desarrollados se convierten en librerías que pueden ser utilizadas por otros.

Interfaces. En cada módulo simple es posible definir las conexiones que este puede tener; a su vez, cada módulo compuesto tiene las conexiones que define el módulo simple que hace la función de interfaz. Como ejemplo de esto, se hace la suposición de un *router* (módulo compuesto) definido por dos módulos simples, una cola y un módulo de enrutamiento. El módulo de enrutamiento sería quien tiene la función de interfaz de salida, para comunicarse con los nodos hacia donde se puede encaminar la información. De modo análogo, la cola funcionaría como interfaz, pero esta vez de los paquetes entrantes; por ende, el módulo *router* (compuesto) tiene: las interfaces de entrada, que provee la cola (un módulo simple), y las interfaces de salida, que posee el módulo enrutador (otro módulo simple).

Herencia. Los módulos y los canales pueden tener subclases. Por ser una herramienta de simulación basada en componentes, como se dijo, tiene la posibilidad de implementar proyectos ya realizados, y de que a su vez, estos puedan ser reimplementados, dando la posibilidad de agregar parámetros, puertos y, en caso de ser módulos compuestos, de cambiar los módulos simples por otros.

Paquetes. El lenguaje NED presenta una estructura de paquetes, como la de *java*, para minimizar los riesgos de conflictos entre los diferentes paquetes. A su vez, utiliza, análogo a *Classpath* de *java, Nedpath*, que se generó para especificar la dependencia de los módulos de simulación, de una manera más sencilla.

4.2. Módulos simples

Cuando se hace referencia a un módulo simple, de manera implícita se puntualiza sobre los componentes activos del modelo. Un módulo simple se define por una palabra clave y es representado como un objeto para ser llamado por otras clases.

4.2.1. Simulaciones de eventos discretos

Un sistema de eventos discretos es un sistema donde los estados cambian; para el caso de OMNeT++, los eventos cambian y esto sucede en un instante de tiempo. Además, cada evento sucede en un tiempo cero (0), asumiendo que nada interesante sucede entre dos eventos consecutivos, lo que conlleva a representar la simulación de sistemas continuos.

Esto es ideal para un simulador de red, ya que generalmente estas tienen comportamientos de eventos discretos, como son: el inicio en la transmisión de un paquete, la llegada del paquete, el *timeout* de una retransmisión, etc.

El simulador utiliza una estructura de datos para hacer toda la simulación; al iniciar, todos los datos a simular se cargan en una estructura de datos llamada FES (*Future Event Set*), de la cual se extraen todos los eventos hasta que esta se encuentra vacía. Su pseudocódigo se presenta en la Figura 33.

```
while (FES not empty and simulation not yet complete)
{
    retrieve first event from FES
    t:= timestamp of this event
    ...
}
```

Figura 33. Comportamiento FES

4.2.2. Implementación de la FES

Esta implementación es de gran importancia para el rendimiento de las simulaciones de eventos discretos. Su funcionamiento en OMNeT++ es implementado como una pila binaria (*binary heap*). Su definición se encuentra en la clase *cMessageHeap*, pero generalmente es irrelevante para el programador de las simulaciones.

4.3. Estructura de los componentes y las conexiones

Los módulos desarrollados en OMNeT++, como se ha mencionado, se centran en módulos simples o compuestos y en conexiones que permiten el paso de mensajes entre ellos, de manera general (OMNeT++ Community, 2011a); las conexiones vienen con ancho de banda infinito, o canales ideales, lo que debe ser editado por el programador para generar el canal deseado.

Ya que los módulos simples son los componentes activos de la red, a su vez los canales también son entendidos como componentes, y ambos obedecen a códigos programados en lenguaje C++.

Teniendo en cuenta lo anterior, OMNeT++ diseñó todos los componentes heredados de la clase *cComponent*, un objeto que contiene todos los comportamientos de la clase *cObject*, ya que hereda de ella, como se presenta en la Figura 34.

También, se puede apreciar, que del cComponent se heredan dos clases: *cChannel* y *cModule*, que representan los canales y los módulos simples, respectivamente.

La clase *cChannel*, como se aprecia en la Figura 35, permite crear tres tipos de canales: *cIdealChannel*, *cDelayChannel* y *cDatarateChannel*. Aun así, el usuario tiene la posibilidad de crear su propia clase de canal, la cual, debe heredar de *cChannel* o de sus hijos directos.

Con la clase *cModule* ocurre algo similar; hereda dos clases principales, de las que ya se ha hablado: *cSimpleModule*, de la cual los usuarios generan aplicaciones, enrutadores

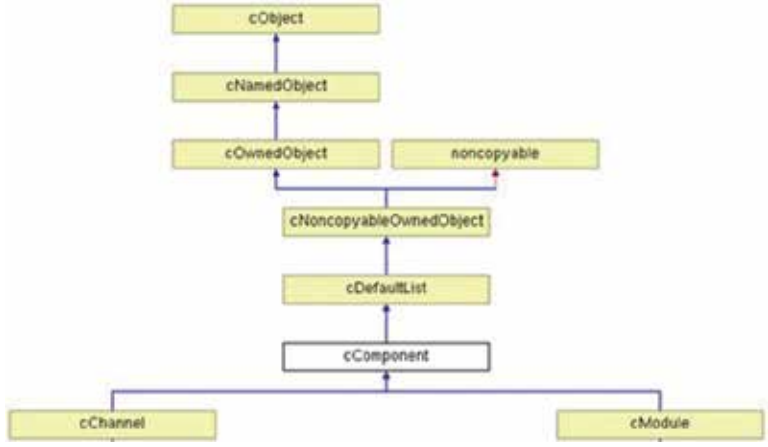


Figura 34. Representación *cComponent* (OMNeT Community, 2011b)

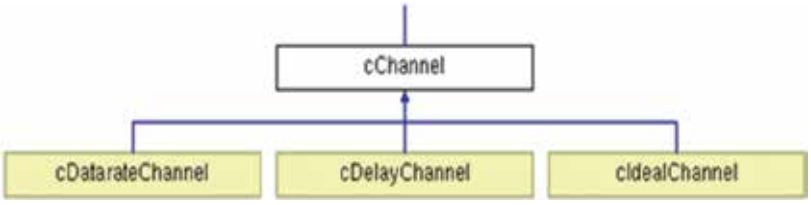


Figura 35. Herencia *cChannel* (OMNeT Community, 2011b)

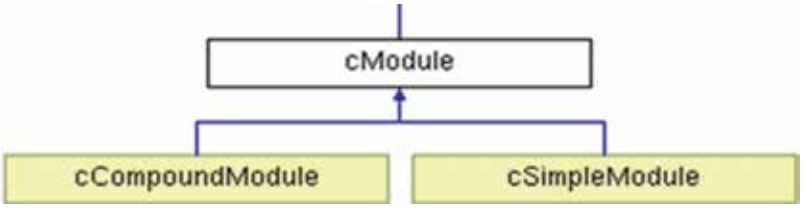


Figura 35. Herencia *cModule* (OMNeT Community, 2011b)

y nodos activos de la red; y *cCompoundModule*, que representan la unión de varios *cSimpleModule* para que funcionen en conjunto. Esto se puede ver en la representación de herencias que presenta la Figura 36.

Una vez entendidas las clases, se puede empezar a indagar sobre los métodos básicos que se utilizan en las simulaciones. Los cuatro métodos principales son:

» *Initialize()*.Se encarga de dar los valores iniciales a la simulación y de iniciar el resto del programa.

» *Finish()*.Se encarga de terminar el código correctamente. El método es llamado automáticamente cuando la FES termina pero, si se detiene la simulación, es recomendable llamar este método para minimizar los posibles errores de compilación.

» *Handle Message (cMessage *msg)*. Es invocado con el mensaje como parámetro, siempre que el módulo recibe un mensaje. Este método es el encargado de procesar el mensaje y de retornarlo a la simulación; el tiempo transcurrido en el método no se asume como tiempo de simulación.

» *Activity()*. El método empieza como un hilo al iniciar la simulación y corre hasta que termina o hasta que se utilice *return*. Los mensajes son obtenidos con el método *receive()* y el tiempo de simulación se toma cuando actúa el método *receive()*.

4.4. Definir módulos simples en OMNet++

Como se ha mencionado, un módulo simple es un programa en C++ que, básicamente, hereda de la clase *cSimpleModule*. Para conectar la programación en lenguajes C++ y NED, es necesario hacerlo por medio del método *Define_Module()*, el cual se encarga de registrar la clase C++ con OMNeT++.

» Un módulo simple debe tener definidas como prioritarios los siguientes elementos:

» un *include files*, que incluya la cabecera de *omnetpp.h*;

» un modelo que virtualiza las clases y define el tipo de módulo a utilizar, como se presenta en la Figura 37;

» el registro de la clase .cc a OMNeT++, por medio de *define_module()*; y

» la definición de los módulos incluidos y virtualizados en la cabecera.

```
classNombre_Modulo : publiccSimpleModule
{
protected:

      virtualvoidinitialize();
      virtualvoidhandleMessage(cMessage *msg);
      virtualvoidactivity();
   };
```

Figura 37. Estructura inicial módulo simple

4.5. Constructor

Los módulos simples nunca son instanciados por los usuarios que crean las clases, ya que todos los módulos son heredados directamente de la clase *cSimpleModule*. Lo anterior implica que los constructores no se pueden definir arbitrariamente. Para solucionar esto, OMNeT++ define para su implementación los constructores públicos. Los módulos simples no deben tener argumentos.

Por otro lado, la clase *cSimpleModule* presenta dos constructores: el primero de ellos, sin ningún tipo de argumentos, como se presenta en la Figura 38**;** el segundo, con la posibilidad de cambiar el tamaño de la pila, como se presenta en la Figura 39**.** Este último constructor es importante cuando se va a utilizar el método *activity()*, ya que el tamaño por defecto de la pila se agota al usar este método.

```
classCol : publiccSimpleModule
{
public:
      Col() : cSimpleModule(){}
      virtualvoidinitialize();
      virtualvoidactivity();
      virtualvoidhandleMessage(cMessage *msg);
};
```

Figura 38. Constructor sin argumentos

```
classCol : publiccSimpleModule
{
public:
      Col() : cSimpleModule(16384){}
      virtualvoidinitialize();
      virtualvoidactivity();
      virtualvoidhandleMessage(cMessage *msg);
};
```

Figura 39. Constructor con argumentos

Anexo a lo anterior, cabe decir que el lenguaje NED presta numerosos recursos adicionales, lo que ayuda a desarrollar más herramientas para la comprensión de la telemática y el funcionamiento de los sistemas en red. Lo que se ha presentado en este capítulo, responde a las necesidades básicas para lograr la implementación de una red, encaminado a entender la influencia de diferentes colas en la calidad de servicio (QoS).

Capítulo 5

Ejemplo Tic-Toc

Para comprender el funcionamiento de la herramienta y el proceso inicial de simulación, el primer ejemplo que se documentará será el ejemplo clásico de Tic-Toc, el cual es un ejemplo sencillo a la hora de adentrarse con esta herramienta; consiste en la disposición de dos módulos simples que van a transmitir, de un lado al otro, un mensaje. A continuación se presentan los pasos necesarios para efectuar la correcta instalación y ejecución del ejemplo.

Primero, se crea un nuevo proyecto tipo OMNeT++ project, como se muestra en la Figuras 40, 41 y 42.

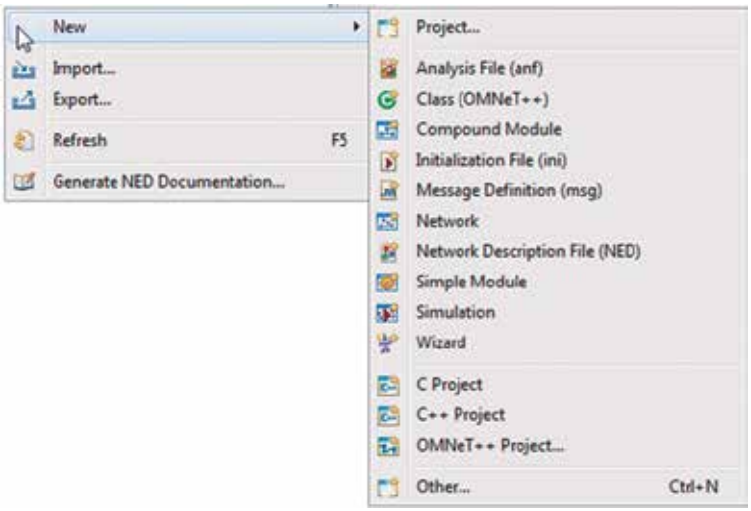


Figura 40. Creación de un proyecto tipo OMNeT++

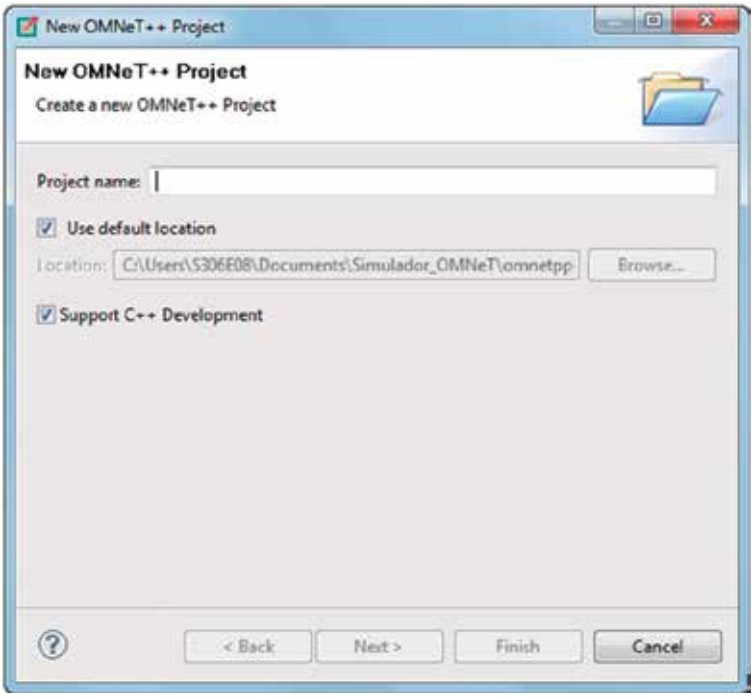


Figura 41. Ventana de creación de un proyecto tipo OMNeT++ (1 de 2)

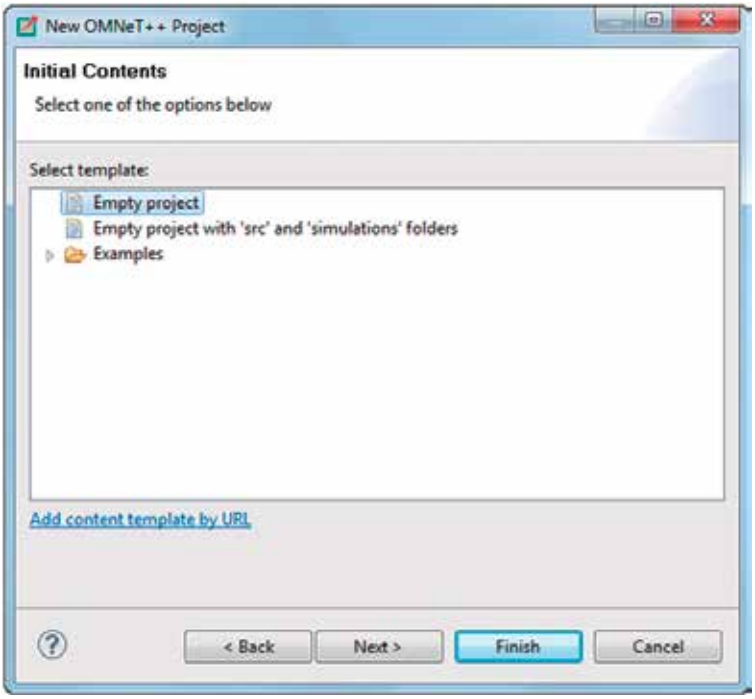


Figura 42. Ventana de creación de un proyecto tipo OMNeT++ (2 de 2)

Una vez finalizada la creación del proyecto se debe observar en el explorador del proyecto lo que se muestra la Figura 43.

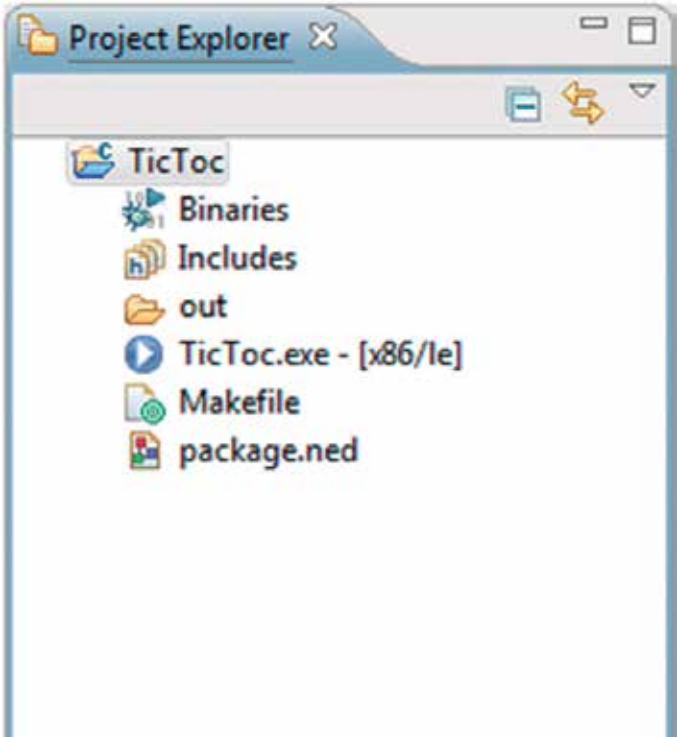


Figura 43. *Project Explorer* del proyecto TicToc

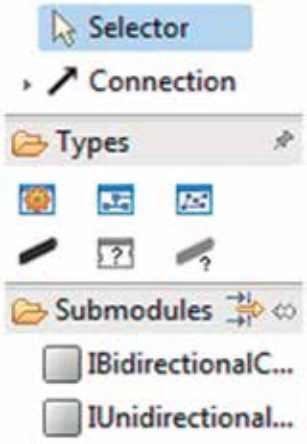


Figura 44. Ventana de utilidades del proyecto

Luego se debe abrir el archivo con extensión *.ned* (*package.ned*). En la parte de Utilidades, cuando se abra el archivo con extensión *.ned* se debe apreciar lo que muestra la Figura 44.

Para la creación de un proyecto OMNeT++ se deben tener en cuenta los siguientes componentes:

» un archivo con extensión *.ned*;

» un archivo con extensión *.cc*; y

» un archivo con extensión *.ini*

Se inicia la creación del archivo con extensión *.ned*, donde va la simulación (parte grafica); en este espacio es donde se definen los módulos que se van a usar en la simulación. Las Figuras 45 y 46 muestran cómo proceder para la creación del archivo *.ned*

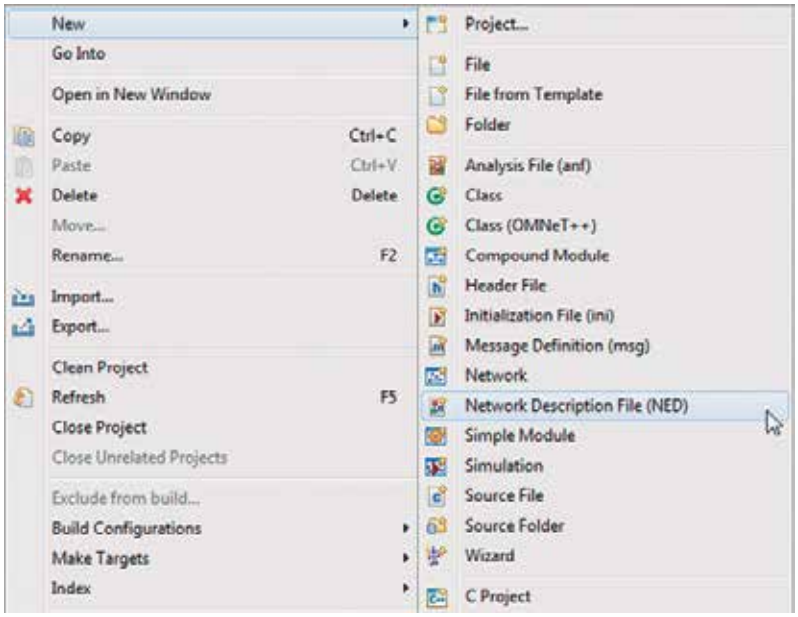


Figura 45. Creación del archivo *.ned*

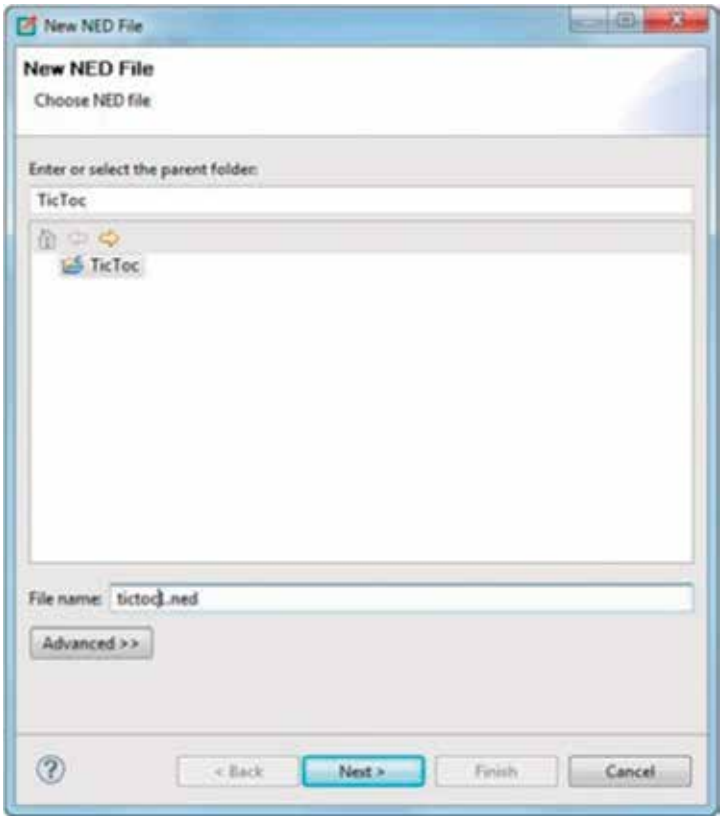


Figura 46. Ventana de creación del archivo *.ned*

Una vez creado el archivo se observa lo que se muestra la Figura 47. Como se puede observar en ella, hay dos pestañas en la parte inferior, una, *Design*, donde se encuentra la parte gráfica, el diseño (que es la que se observa en la figura), y otra, *Source* que hace referencia al código fuente de ese diseño.

Figura 47. Ventana de diseño del archivo *.ned*

Como lo que se quiere en este ejemplo es tener un módulo simple que tenga un puerto de entrada y otro de salida, para realizar el traspaso de mensajes –que es en lo que consiste este ejemplo–, en la pestaña *Source* se va a disponer el código que se presenta a continuación:

```
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as publish
ed by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; Without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public Licen
se
// along with this program.  If not, see http://www.gnu.org/licenses/.
//

package tictoc;

import ned.DelayChannel;
import ned.DatarateChannel;


simple Txc1
{
    gates:
        input in;
        output out;
}

//
// Two instances (tic and toc) of Txc1 connected both ways.
// Tic and toc will pass messages to one another.
//
network Tictoc1
{
    submodules:
        tic: Txc1 {
            @display("p=205, 70");
        }
        toc: Txc1;
    connections:
        tic.out >DelayChannel > toc.in;
        toc.out >DelayChannel > tic.in;
}
```

Este código, primero define un módulo simple, con un puerto de entrada y otro de salida. Luego, dentro de las llaves donde se define la red, determina las instancias que van a ser usadas en la simulación, una se llama *Tic* y la otra *Toc*. También determina que los enlaces que conectan a *Tic* con *Toc*, son de tipo *DelayChannel*.

Figura 48. Tic-Toc archivo *.ned*

Al pasar a la pestaña de diseño se va a observar que se crearon unos módulos y una topología como se muestra en la Figura 48.

En la Figura 49, por su parte, se puede observar que aparece el nuevo tipo de módulo que fue definido en el archivo *.ned*

A continuación se procede a crear la clase *.cc*, que va a ser la encargada de tener las funcionalidades de la simulación, la que, cómo se explicó, lo que va a realizar es el paso de un mensaje de un módulo a otro. Por esta razón, se debe redefinir el método *handleMessage(cMessage *msg)* y adicionalmente el método *Initialize()*, porque es tanto el que se llama al inicio de la simulación, como donde se define si va a ser el Tic o el Toc, quien inicie la comunicación. En las Figuras 50 y 51 se ilustra el proceso de creación de la clase *.cc*

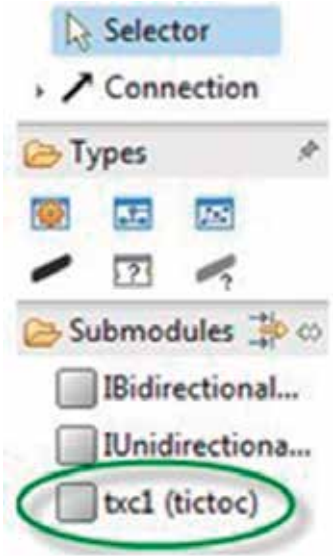


Figura 49. Ventana de utilidades del proyecto con el nuevo módulo

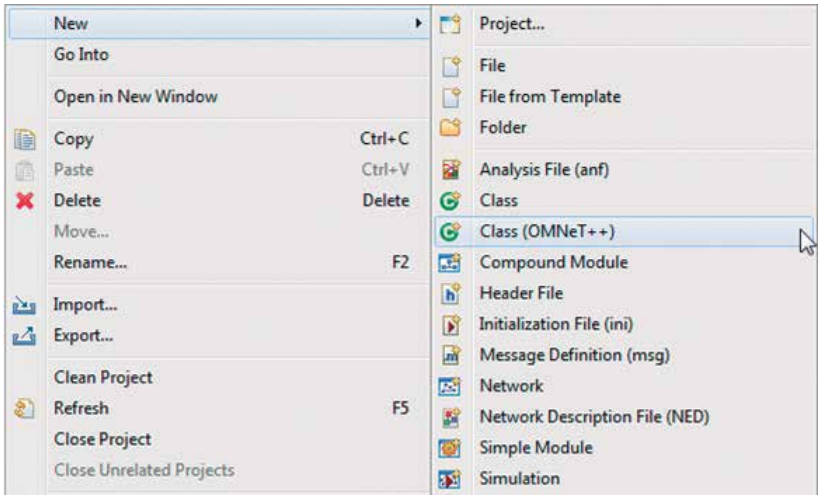


Figura 50. Creación de la clase *.cc*

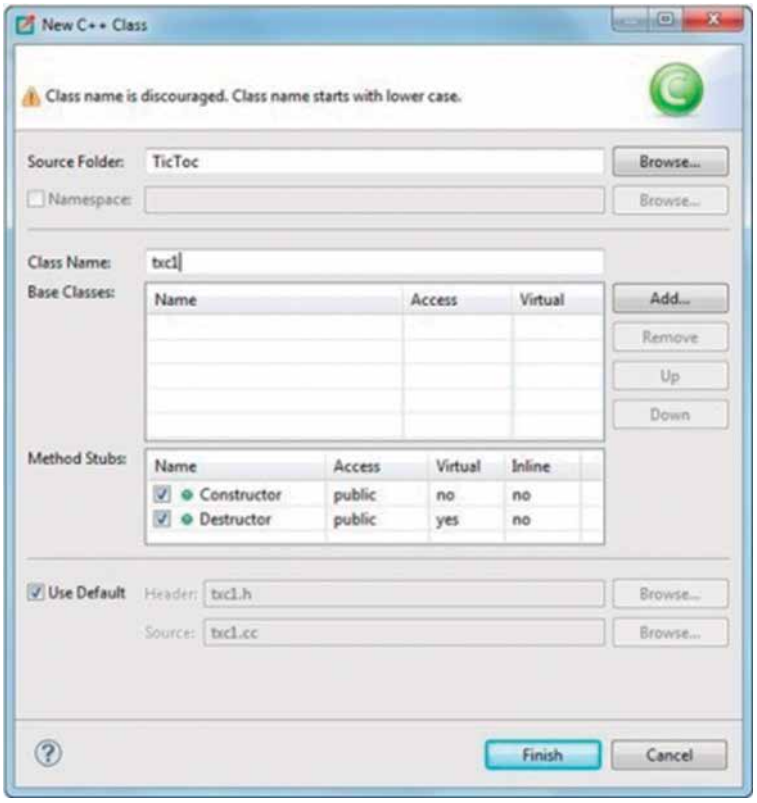


Figura 51. Ventana de creación de la clase *.cc*

Una vez realizado lo anterior en el Explorador del proyecto se debe observar lo que presenta la Figura 52.

Al hacer clic en la clase *.cc* (*txc1.cc*) para comenzar a editarla, se dispondrá el siguiente código, en el cual se encuentra la redefinición de los métodos que se mencionaron, el *handleMessage* y el *Initialize*:

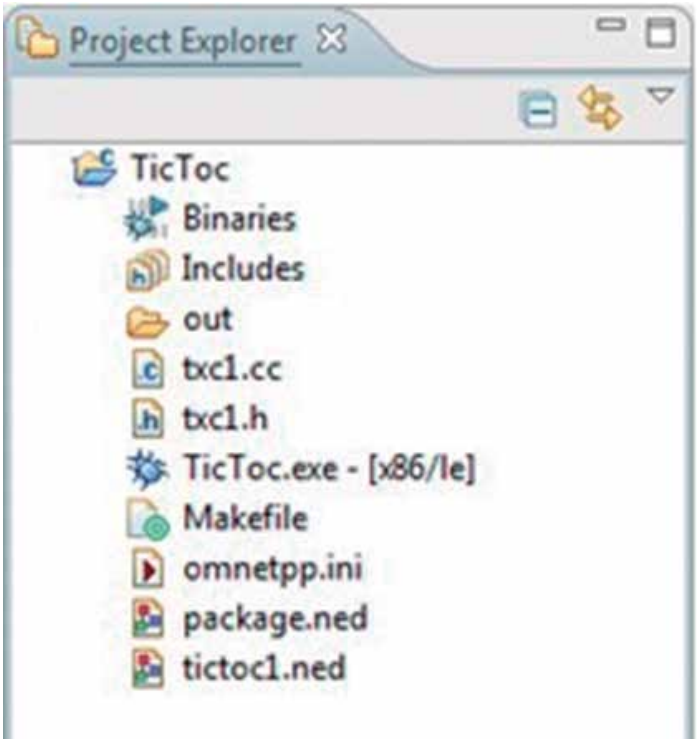


Figura 52. Ventana *Project Explorer* con la clase *.cc*

Hecho esto, solo resta editar un último archivo para poder dar inicio a la simulación, el archivo *.ini*, el archivo de configuración de la simulación, que es usado por la herramienta para realizar la compilación y ejecución de la simulación. En las Figuras 53 y 54 se observa la creación de dicho archivo.

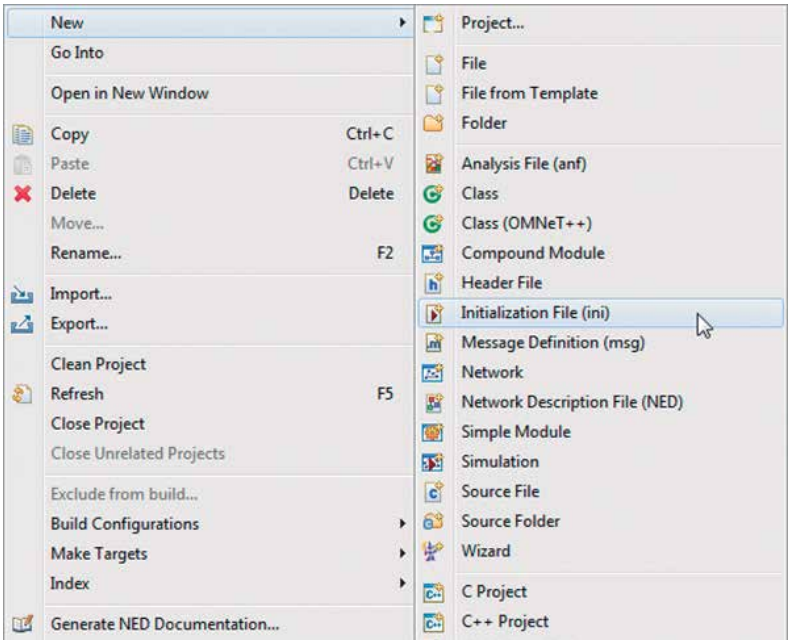


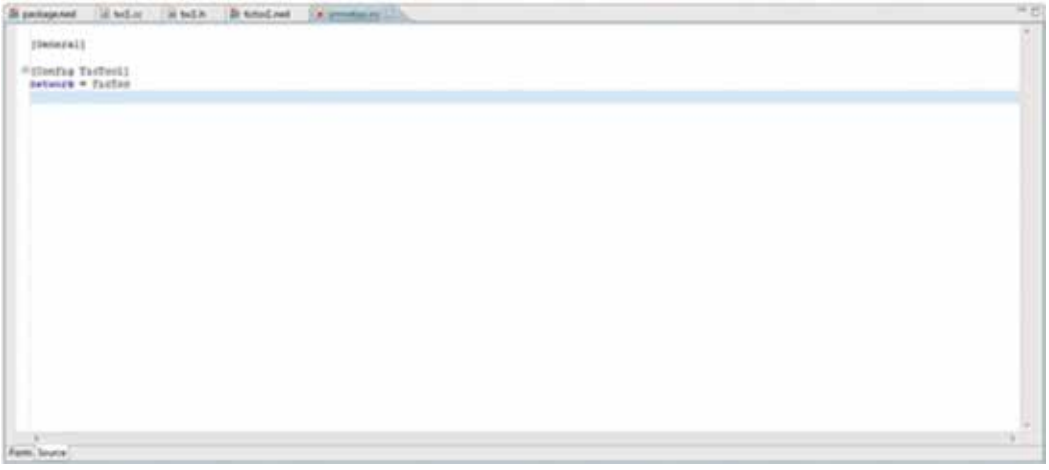
Figura 53. Creación del archivo *.ini*

Figura 54. Ventana de la creación del archivo *.ini*

Una vez se abre el archivo *.ini* se observa una interfaz como la que muestra la Figura 55**.** Aquí, al igual que en el archivo *.ned*, existen dos pestañas, una denominada *Form* y otra denominada *Source;* en esta última se va a disponer el código de la red a simular.

Hecho esto, ubicándose en el proyecto, se presiona el clic derecho y se selecciona la opción de *Run as*, como se muestra en la Figura 56.

Figura 55. Código fuente del archivo *.ini*



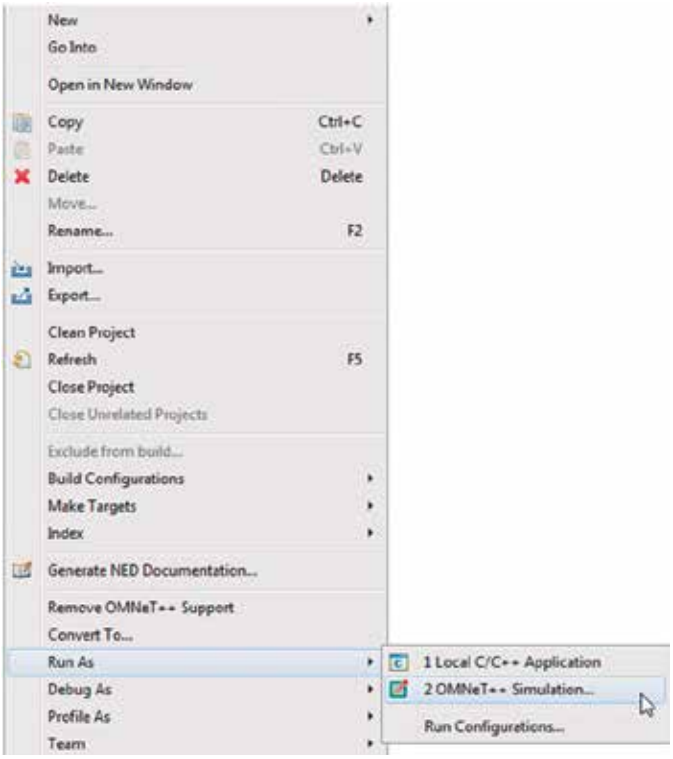


Figura 56. Inicio de la simulación

Cuando se realiza esta acción un mensaje de advertencia se despliega en la pantalla, indicando que se ha creado una nueva configuración, como muestra la Figura 57.

Al aceptar el mensaje aparece otra ventana emergente que indica la selección del archivo .ini, como muestra la Figura 58.

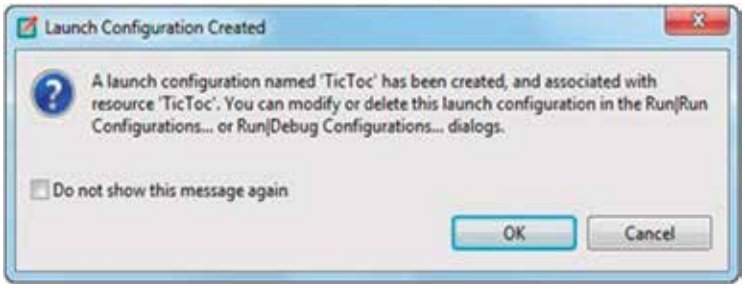


Figura 57. Mensaje de advertencia

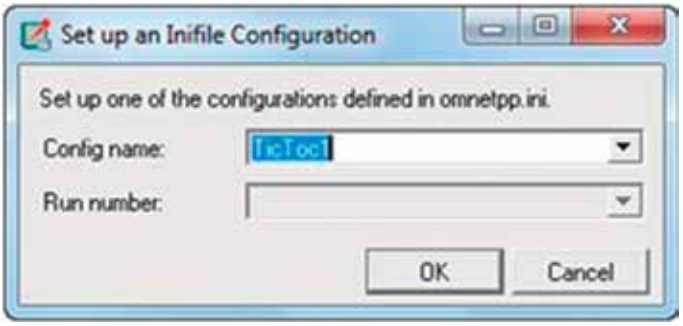


Figura 58. Ventana - *Set up an Inifile Configuration*

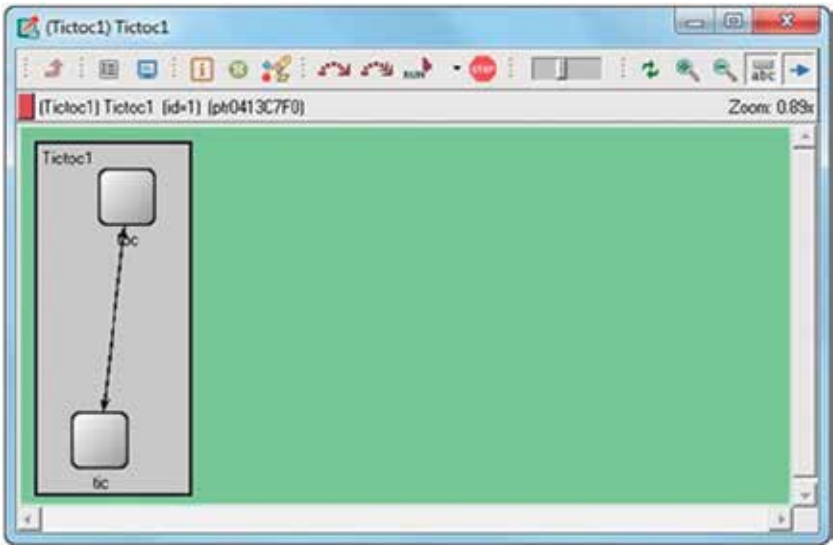


Figura 59. Ventana - Interfaz gráfica de OMNeT++ (1 de 3)

Finalizada la acción anterior se desplegará en pantalla una interfaz como la que muestra la Figura 59.

Para iniciar la simulación se presiona el botón de *Run* que se encuentra la parte central del menú superior de la Figura 59. Al hacerlo se observa lo que muestra la Figura 60 y 61, en la que el recuadro rojo indica que módulo está actuando y el círculo rojo que se mueve a través del enlace, hace referencia al mensaje que está siendo transmitido.

Además de la ventana de interfaz gráfica de OMNeT++, el simulador abre otra ventana que arroja toda la información de la simulación, como se puede observar en la Figura 62.

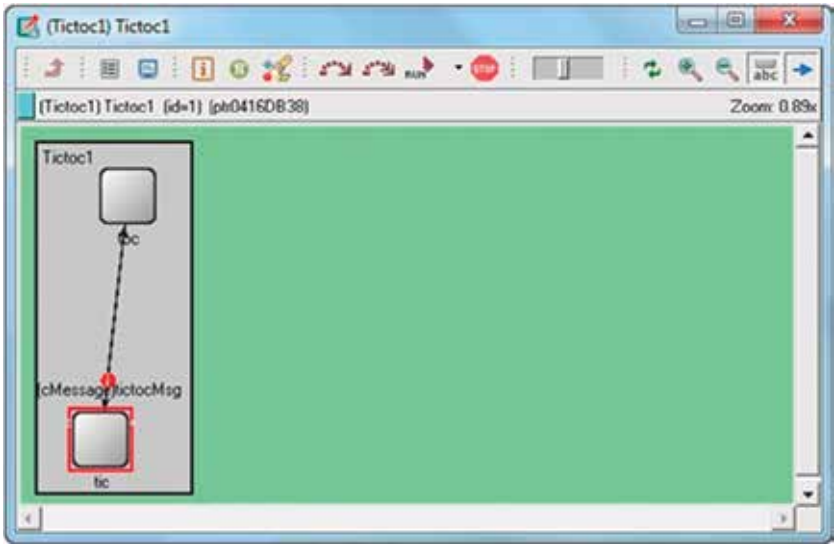


Figura 60. Ventana - Interfaz gráfica de OMNeT++ (2 de 3)

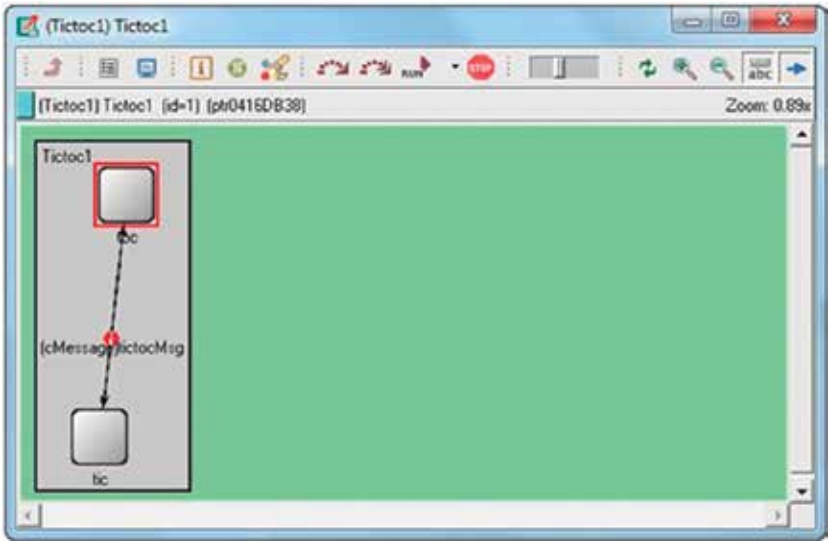


Figura 61. Ventana - Interfaz gráfica de OMNeT++ (3 de 3)

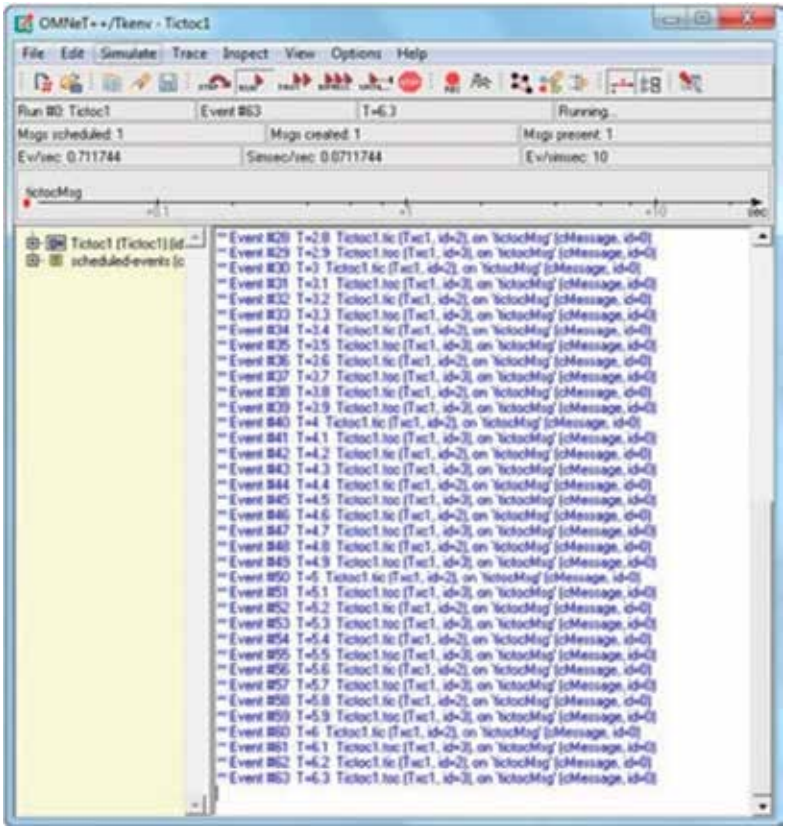


Figura 62. Ventana de información del proceso de simulación del TicToc

Para terminar la simulación se presiona el botón *Stop*, que se encuentra en la parte central del menú superior de la ventana de la interfaz (ver Figura 61).

Capítulo 6

Simulación de esquemas de encolamiento

En este capítulo se presenta el proceso realizado para simular diferentes esquemas de encolamiento (FIFO, PQ y LLQ) de manera básica con el fin de ofrecer QoS en una red. También se presentan gráficas que permiten realizar el análisis posterior de los datos obtenidos en la simulación.

6.1. Entorno de desarrollo utilizado para la implementación

Como presenta Varga (2011), OMNeT++ es una herramienta que se puede ejecutar en diferentes sistemas operativos. Ha sido probada en los siguientes ambientes:

- Windows 7, Vista y XP;
- Mac OS Leopard (10.5) y Snow Leopard (10.6);
- Ubuntu 10.04 LTS y 11.04;
- Fedora Core 15 y Core 16;
- Red Hat Enterprise Linux Desktop Workstation 5.5; y
- Open SUSE 11.4.

OMNeT++ se ha probado en arquitecturas Intel de 32 y 64 bits, aunque *Red Hat* es el único sistema operativo en el que se ha probado con la arquitectura Intel-32 bits.

Para la implementación realizada en este capítulo se probó el simulador OMNeT++ 4.1 en Mac OS X Lion (10.7), con resultados exitosos. Para su implementación fue necesario instalar, *XCode 4.3.2* y *Java Runtime*, el cual no viene por defecto en la última versión de Mac OS X; en Windows 7, se adicionó el JDK en la carpeta pertinente del simulador, para su correcto funcionamiento.

Las simulaciones de OMNeT++, pueden correr prácticamente en cualquier entorno Unix que contenga un compilador C++ actualizado, ya que algunas características (i.e., *Tkenv*, *parallel simulation*, *XML support*, etc.) necesitan librerías externas (e.g., *Tcl/Tk, MPI, LibXML* y *Expat*)

El entorno de desarrollo integrado de OMNeT 4.x se basa en la plataforma *Eclipse* (Varga, 2011); utiliza además nuevos editores, asistentes y funcionalidades adicionales. Para lograr la plataforma visual y facilitar un entorno agradable para la herramienta, OMNeT++ integra nuevas funcionalidades que permiten crear y configurar modelos; estos archivos utilizan extensiones *.ned y .ini*. Incluye también otro tipo de funcionalidades para realizar ejecuciones por lotes y analizar resultados. *Eclipse*, por su parte, permite editar el funcionamiento de las partes adjuntas del código C++, para el correcto funcionamiento de la simulación.

Los archivos *.ned*, permiten al usuario generar módulos simples con su propio lenguaje (*Ned*), los cuales son componentes activos de los modelos (i.e., *routers*, generadores de tráfico, *switches*, etc.), y describir la estructura del modelo de simulación que se desea

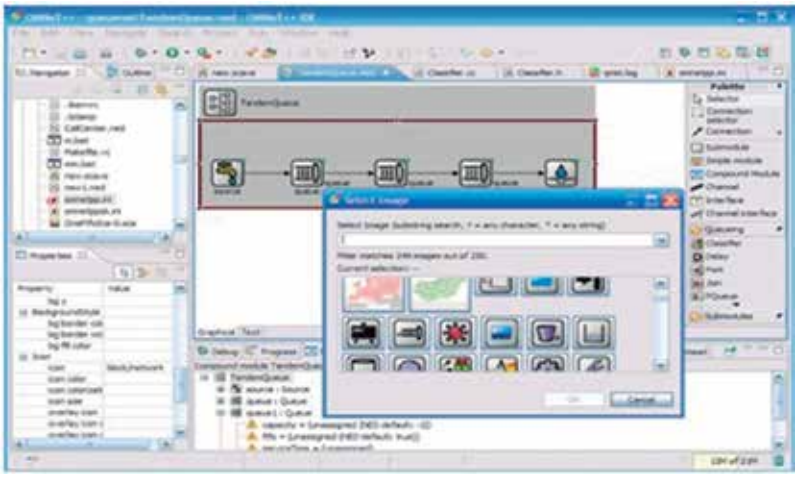


Figura 63. Presentación archivo *.ned* (OMNeT++ Community, 2012)

implementar. A su vez un archivo *.ned* puede utilizar módulos desarrollados en otros archivos *.ned* para generar una red con diferentes elementos. En la Figura 63 se puede ver una red generada por múltiples módulos, en el que la red es, a su vez, un módulo.

6.2. Escenario de simulación

En términos de calidad de servicio (QoS), son muchos los obstáculos que hay que superar para lograr entregar los datos dentro de los parámetros ideales para las aplicaciones sensibles al retardo. Para evaluar el tipo de esquema en el cuál el funcionamiento es óptimo, es necesario diseñar una red que contenga los elementos primordiales para la evaluación de cada uno de los componentes que afectan la calidad de servicio.

Teniendo en cuenta lo anterior, cabe aclarar que los componentes que influyen en el retardo de extremo a extremo, como indica Tanenbaum (2003) son muchos, incluyendo el retardo del algoritmo de codificación, el tiempo de empaquetado, los tiempos de propagación (despreciable excepto cuando la información viaja grandes distancias), el tiempo de transmisión, los tiempos de espera en las colas de red (lo cual depende de la cantidad de paquetes que cada cola maneje), el tiempo de descompresión, etc.

Por ende, para lograr efectuar una simulación que sea lo más ajustada posible a la realidad, es necesario tener un escenario donde se encuentren todos estos componentes básicos y se pueda aislar el componente con el cual se desea trabajar. Obligatoriamente tienen que existir cuatro módulos principales: un equipo generador de tráfico; un módulo que direccione y maneje los paquetes, simulando el paso por otro u otros nodos de la red; un elemento que consuma los mensajes para terminar su recorrido y; los enlaces que se encargan de transportar los paquetes.

Para el escenario de simulación elegido, se cuenta con los cuatro componentes primordiales explicados. El primero, un generador de tráfico con la posibilidad

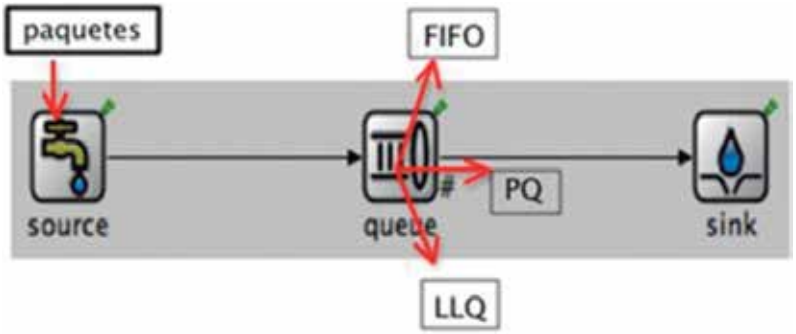


Figura 64. Escenario de simulación adaptado de simulador OMNeT++

de etiquetar los paquetes creados y emitirlos por el canal. Este elemento tiene configuradas, a su vez, las características necesarias para congestionar el siguiente componente, una cola, la que, en este caso en particular, simulará los elementos por los que tienen que pasar a través de la red, donde existe la posibilidad de congestión, seguido por un consumidor, quien se encargará de representar el elemento al cual llega el mensaje. La implementación realizada se presenta en la Figura 64.

Cada uno de estos elementos representa, como se ha mencionado, la estructura para que la información viaje, por lo que es importante explicar, cada uno de ellos.

6.2.1. Paquete

OMNeT++ es un simulador orientado al paso de mensajes para la comunicación de sus módulos. Se utilizan dos tipos de objetos para la transmisión de mensajes, los cuales permiten comunicar módulos simples y compuestos, los *cMessages* y los *cPackets*.

cPacket es una subclase de *cMessage* (OMNeT++ Community, 2012). La primera es utilizada para simulaciones que necesitan paquetes de red (i.e., tramas, datagramas, etc.); la segunda, para el resto de paso de mensajes que no son específicos.

En el contexto de la simulación a presentar, se seleccionaron, para el paso de información, mensajes de tipo *cPacket*, ya que presentan un constructor similar al de *cMessage*, por ser una subclase, pero aceptan el argumento de tamaño del mensaje (*bit length)*; este archivo es guardado en bits, pero puede ser almacenado, a su vez, en bytes. En caso que el campo guardado no sea múltiplo de ocho, automáticamente lo completará con una función techo para agregar los bits que hagan falta.

Para la creación del paquete utilizado en este trabajo, fue necesario generar, en lenguaje *Ned*, un paquete que tiene un entero (*int*) llamado *priority*. Una vez se compiló el proyecto, gracias al paquete de extensión *.msg*, se generaron otras dos clases, una representando la clase desarrollada en C++ (*MyPacket_m.cc*) y otra en representación de la cabecera (*MyPacket_m.h*).

En la clase *MyPacket_m.cc* quedaron plasmados todos los métodos que contiene la clase *cPacket* (*get* y *set*) incluyendo los de la variable *priority* adicionada por los desarrolladores del proyecto.

Los paquetes, aunque poseen la propiedad de definir un tamaño variable, se tomarán con un tamaño constante, ya que, para los contextos en los que se evaluará la práctica, no se tendrá en cuenta ese tamaño.

6.2.2. Generador de tráfico

La Figura 65 muestra la imagen con la que se identificará el generador en la red. Se plantea el generador de tráfico como la representación de un nodo en la red que espera transmitir información, lo que puede ser visto desde diferentes perspectivas. Por esta razón, no se especifica como un nodo en particular, ya que puede estar representando, tanto a un equipo que genera paquetes, como a la transmisión de paquetes de varios equipos que tienen llegada a la cola a presentar.

Figura 65. Generador de tráfico (OMNeT++ Community, 2011a)

El generador de tráfico tiene un puerto de salida, por donde encuentra la posibilidad de colocar los paquetes en el canal. A su vez, posee una variable *volatile double interArrival Time*, expresada en unidades de segundos, que representa el tiempo de generación de los paquetes. Como la idea se centra en iniciar la simulación con una cola congestionada para analizar el comportamiento de los algoritmos de encolamiento, se están enviando todos los paquetes con un tiempo de llegada a la cola de cero (0) segundos.

Las clases de OMNeT++ implementadas por el generador desarrollado son las siguientes:

Initialize(). Encargada de asignar el tiempo de inicio a la variable *startTime* y auto-enviar un mensaje al módulo, para que este sea manejado por el método *handleMessage()*; el método que utiliza para realizar lo explicado tiene como nombre *scheduleAt* y recibe como parámetros, el tiempo de simulación dado por la variable *startTime* y el mensaje a enviar.

textbfhandleMessage(). Este método recibe como parámetro un *cMessage*, el que, para este caso, proviene del método *Initialize()*. Es necesario aclarar que para la práctica presentada, se recibe un paquete de tipo *MyPacket*, desarrollado por los autores del proyecto, por lo que es necesario hacer un *cast* para lograr editar el paquete y sus prioridades.

6.2.3. Colas

Para entender la importancia de la implementación de las colas en las redes de datos es necesario primero entender que las aplicaciones son de gran importancia para

la implementación de etiquetas en los paquetes, ya que de acuerdo con la etiqueta, se define el ancho de banda que necesitan y la prioridad que puedan requerir. Para entender esto más a fondo, se puede reunir en dos grupos las diferentes aplicaciones: sensibles y no sensibles al retardo.

Las aplicaciones sensibles al retardo se centran en los diferentes programas ejecutados en la red, que requieren llegar con el mínimo retraso posible, desde su fuente hasta su destino final. Generalmente, este tipo de aplicaciones suelen representarse con comunicaciones de VoIP y video, pero no son necesariamente estos los ejemplos que pueden definir este tipo de información de una forma totalmente objetiva.

Por lo anterior, se debe tener en cuenta el desarrollo exponencial de las aplicaciones que generan las grandes compañías y las aplicaciones de salud, las cuales tienden a tener el comportamiento de una aplicación sensible al retardo, ya que en ellas se encuentra información valiosa que debe llegar con tiempos efectivos a su destino final. En ambientes industriales, las medidas de ciertos sensores son críticas, como medidas de químicos, operaciones de precisión, detección de bombas, entre otros aspectos que afectan, incluso, la vida de las personas involucradas.

Por otro lado, existen las aplicaciones que no son sensibles al retardo, aquellas cuyo momento de llegada no es crítico, como es el caso del correo electrónico, los mensajes de texto, los chats, las actualizaciones en páginas Web, la descarga de archivos y los elementos de las redes sociales, entre otros.

El tráfico de estas aplicaciones podría en determinado momento esperar por el uso del canal; pero el hecho de que espere no indica que se quede sin utilizarlo, ya que en determinado momento –y según las características del algoritmo de encolamiento–, este tráfico debe utilizar el canal.

Otra característica de la mayoría de aplicaciones no sensibles al retardo (Tanenbaum, 2003) es el alto nivel de confiabilidad que deben tener, ya que, aunque son aplicaciones que permiten tener tiempo de retardo extenso, es necesario que sus paquetes lleguen completos y sin errores, lo que implica, hacer cálculos de suma, para comprobar la llegada correcta de todos los bits y retransmisiones, en caso de que los paquetes lleguen con errores.

Para efectos del esquema de simulación a realizar en este capítulo, se definieron dos tipos de prioridad de tráfico, alta y baja. Prioridad alta para tráfico sensible a retardo, como VoIP o video, y prioridad baja para tráfico transaccional, tipo ftp o correo electrónico.

En la comparación de los comportamientos de las colas, además de definir las prioridades, es necesario seleccionar las colas a evaluar; para este caso, se valoraron tres tipos de cola, para analizar, en efectos prácticos, como OMNeT++ permite apreciar la diferencia entre los comportamientos de estas colas y lo eficientes que ellas pueden ser. Las colas elegidas para evaluar la diferencia en su comportamiento son FIFO, PQ y LLQ.

FIFO (*First In First Out*) es un tipo de encolamiento que se utiliza generalmente por defecto. Es implementado también cuando no se maneja tráfico sensible al retardo, ya que al no tener ningún algoritmo de encolamiento complejo para el almacenamiento y la extracción en la cola, minimiza el tiempo de procesamiento. Su comportamiento, como se muestra en la Figura 66**,** define que el primer paquete en entrar a la cola es el primero en salir y así sucesivamente.

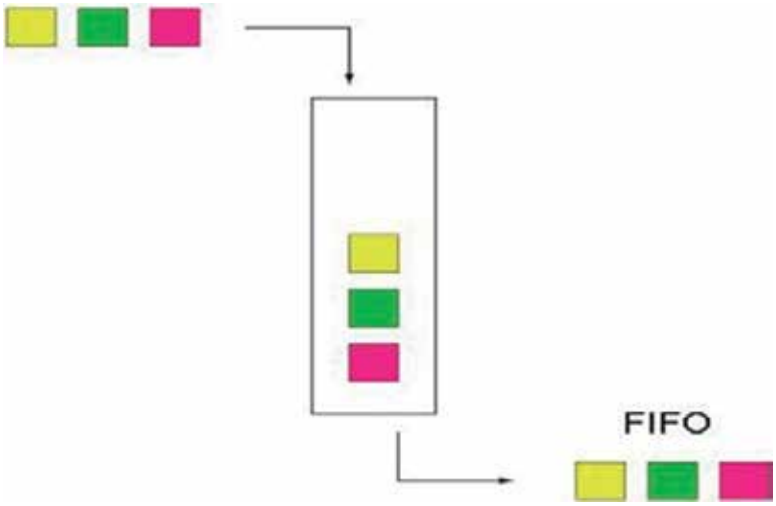


Figura 66. Encolamiento FIFO (Frabs, 2008)

PQ (*Priority Queuing*) es un tipo de encolamiento orientado a la prioridad, que organiza cada paquete en proceso de inserción a la cola, según la prioridad que cada paquete tenga, como se presenta en la Figura 67.

Priority Queuing conserva varias colas, las cuales comprenden, a su vez, diferentes tipos de prioridades. Cada cola es una cola FIFO en sí misma, y guarda los elementos según su etiqueta lo defina. Su implementación es ideal para atender aplicaciones de alto nivel de sensibilidad, cuando el tráfico que ellas generan no es demasiado. Es necesario aclarar que la cola no se debe implementar cuando el tráfico de alta prioridad generado es constante, porque el tráfico de baja prioridad se demoraría en

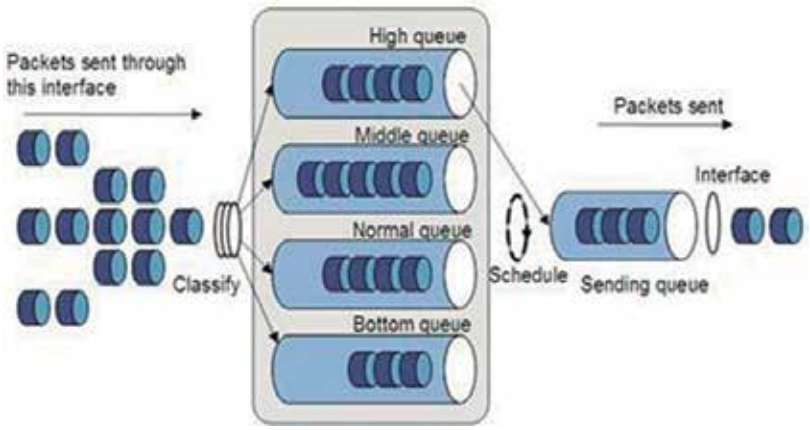


Figura 67. Encolamiento PQ (Moorey, 2008)

ser atendido o incluso, en el peor de los casos, podría no llegar a ser despachado, ya que el algoritmo está diseñado para atender primero los paquetes de mayor prioridad, siempre.

LLQ (*Low Latency Queuing*) es un algoritmo que presenta aspectos importantes en el manejo de tráfico, ya que le da mayor prioridad a las aplicaciones sensibles al retardo, sin olvidar la cola de baja prioridad, como se muestra en la Figura 68. Para lograrlo, conmuta entre las diferentes colas, asignando más ancho de banda a las colas con mayor prioridad, logrando que estas lleguen en tiempo ideal a su destino final.

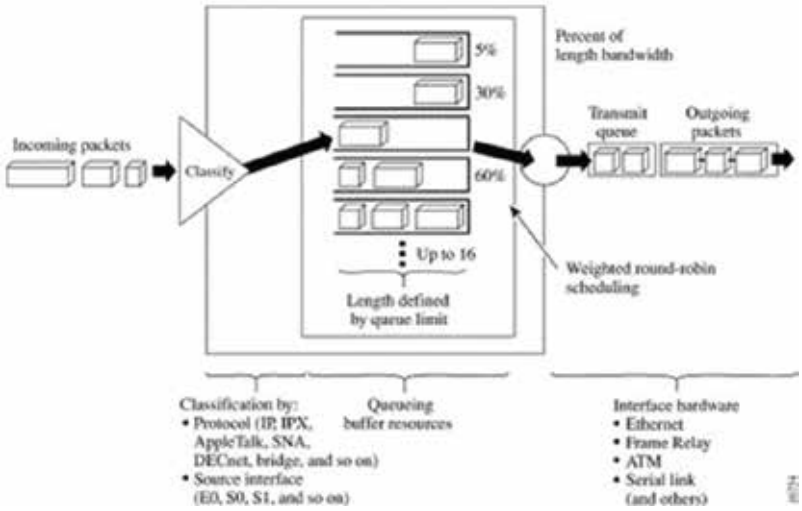


Figura 68. Encolamiento LLQ (http://www.ccietalk.com/)

Según las colas utilizadas, se tendrán diferentes tiempos de llegada; se espera analizar qué tipo de encolamiento presenta mejor desempeño frente a los demás.

La cola, al igual que los otros módulos desarrollados, cuenta con tres métodos principales que permite utilizar OMNeT++ y con una variable *Volatile Double Service Time* que permite medir los tiempos de llegada de los mensajes a este módulo.

Initialize(). Este método se encarga de asignar el nombre a la cola y generar un paquete *end-Service*, para su uso posterior.

handleMessage().Recibe el mensaje, lee la prioridad según la información obtenida y la cola implementada, guarda el paquete en la ubicación que le corresponda y, en el momento indicado, saca los paquetes de la cola para su posterior envío.

startService(). Define el tiempo en el cual llega el mensaje.

6.2.4. Consumidor

El consumidor es un módulo desarrollado para simular el usuario final de la aplicación. En este punto, el paquete recibido será eliminado, permitiendo que el módulo reciba otros paquetes.

Ya que este módulo no debe hacer ningún tipo de algoritmo de procesamiento alto, solamente se utilizará un método para manejar todo el tráfico, para que reciba y haga

el procesamiento indicado, *handleMessage()*, el cual como se mencionó, se encargará de tomar el mensaje y eliminarlo. Es un módulo que queda diseñado para poder ser editado y posteriormente implementado, para la recepción y manejo del diverso tráfico que puede recibir.

6.2.5. Canal

El canal que se implementó en la simulación es el predeterminado por OMNeT++, el cual se asume como infinito, por lo que no influirá en el tiempo manejado en la implementación, lo que permite asegurar que el tiempo expuesto es total y únicamente el resultado de lo sucedido en la cola.

6.3. Resultados experimentales

6.3.1. Datos obtenidos en las simulaciones

A continuación se presenta la interfaz que utiliza OMNeT++ para evidenciar la ocurrencia de cada evento en el simulador. Gracias a este mecanismo de simulación, se puede realizar un análisis comparativo de la situación ocurrida en la simulación. En la Figura 69, se puede apreciar, en color azul, la descripción de los eventos arrojados por OMNeT++, y en verde, la descripción de los eventos realizada por los programadores de la simulación.

En la Figura 69 es posible notar como al inicio de la simulación, se empiezan a describir los eventos por un número consecutivo, el tiempo de simulación, el nombre de la red y el módulo que genera el evento separados por un punto, la información del mensaje y el ID del mensaje.

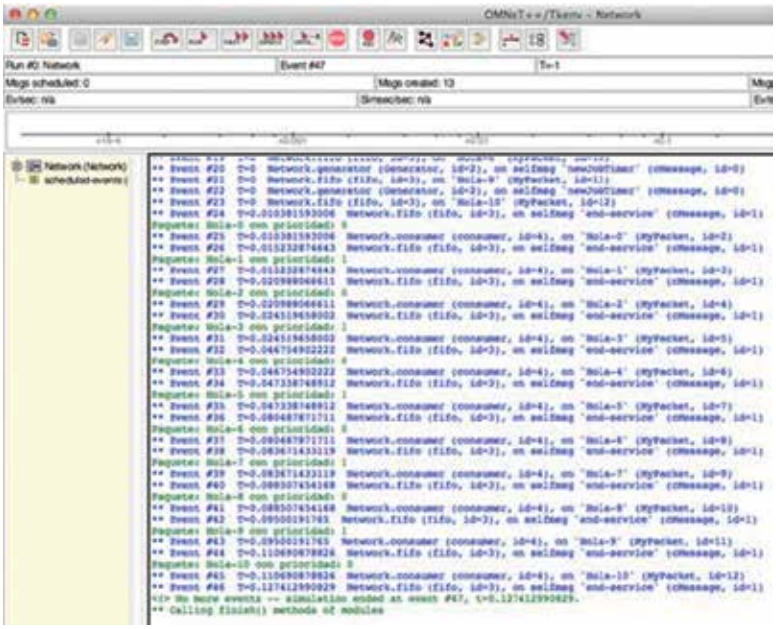


Figura 69. Resultados de la simulación FIFO

Adicional a los eventos, se muestran las prioridades de los paquetes en color verde, para facilitar la visualización del orden de llegada de los paquetes.

Para las siguientes dos simulaciones se presenta la Tabla 5, como resumen, detallando los datos más relevantes de la simulación para el desarrollo del estudio realizado. La Tabla 5 presenta los valores principales para los tres tipos de cola evaluados en las simulaciones, FIFO, PQ y LLQ. De cada tipo de cola, se evaluó el tiempo de salida del primer paquete de alta prioridad y el último paquete de alta prioridad, para poder comprobar, en comparación con las otras colas, si tiene mucho retardo en salir el primer y el último paquete. Por otra parte, se espera poder comprobar si los paquetes de baja prioridad están saliendo en tiempos pertinentes y si está quedando por tiempo indefinido en la cola, lo cual implica la necesidad de evaluar el tiempo del primero y el último de los paquetes de baja prioridad.

Tabla 5. Resumen de los resultados de simulación

COLAS	Salida primer paquete con prioridad alta	Salida último paquete con prioridad alta	Salida primer paquete con prioridad baja	Salida último paquete con prioridad baja
FIFO	0.01523	0.09500	0.01038	0.11069
PQ	0.01038	0.04675	0.04733	0.11069
LLQ	0.01038	0.04733	0.02451	0.11069

De lo anterior, se puede asumir la necesidad de implementación de colas para el manejo de las redes teniendo en cuenta los siguientes enunciados:

La cola que permitió a la aplicación con más prioridad finalizar primero, fue *Priority Queuing*.

La cola en la que ambas aplicaciones llegaron en tiempos similares fue la cola *FIFO*.

La cola que presentó más eficiencia respecto a la llegada de los paquetes tanto prioritarios como no prioritarios fue la *LLQ*.

La cola de prioridades (*Priority queuing*), no permitió brindar un servicio óptimo, ya que deja pasar solo paquetes de alta prioridad antes que los de baja prioridad, lo que deja los paquetes de baja prioridad, esperando durante mucho tiempo. Como se puede apreciar en la Tabla 5, el primer paquete de baja prioridad en FIFO, es retirado de la cola para su envío a los 0.01038 s, en LLQ a los 0.02451 s, y en PQ 0.04734s; este último presenta tiempo ineficiente para los paquetes de baja prioridad, con aproximadamente el doble de la retirada de paquetes de baja prioridad de LLQ y cuatro veces la de FIFO.

El tiempo de procesamiento que toman los módulos en las tres diferentes colas es despreciable, por lo que es importante recalcar, que siendo esta una característica a favor de la cola FIFO, se puede objetar que es la menos ideal para implementar.

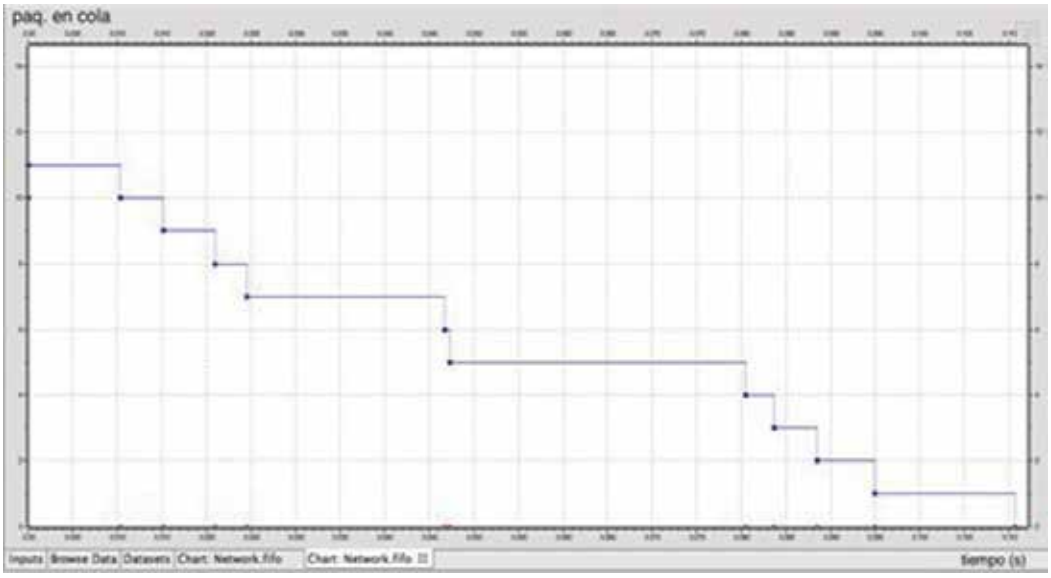
En la cola LLQ el tiempo de llegada del último paquete es 0.047338 s, y en PQ es 0.046754 s, la diferencia es relativamente pequeña, teniendo en cuenta que la cola LLQ, permitió dar paso a otro tipo de paquetes.

6.4. Gráficas del comportamiento de las colas

6.4.1. Gráfica FIFO

Además de lo presentado, OMNeT++ tiene una forma fácil para entender el comportamiento de los eventos y los paquetes. En la Figura 70 se pueden apreciar los puntos rojos inferiores, que resaltan los tiempos en que los paquetes son retirados de la cola, y la línea azul, que muestra el comportamiento de la cola; se puede ver claramente que la cola se encuentra congestionada al inicio de la simulación, lo que ocurre porque se establece que todos los paquetes llegan en el segundo cero (0), para generar una congestión; se puede ver cómo se va liberando la cola hasta quedar totalmente vacía.

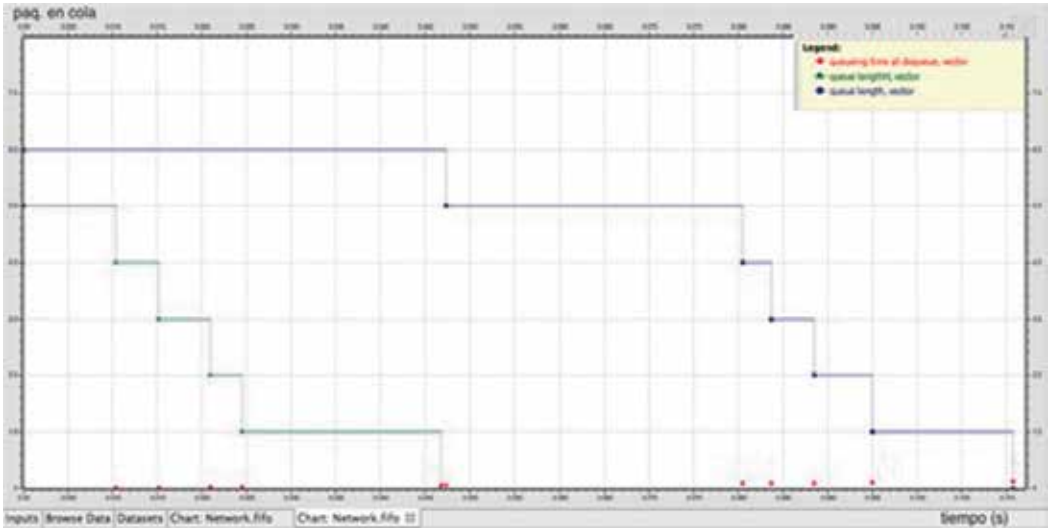
Figura 70. Comportamiento de la cola FIFO



6.4.2. Gráfica PQ

En el encolamiento PQ se puede apreciar que la cola de baja prioridad permanece cogestionada, hasta que la cola de alta prioridad queda completamente vacía; se aprecia como el primer paquete de baja prioridad, tiene mucho retardo al salir (ver Figura 71).

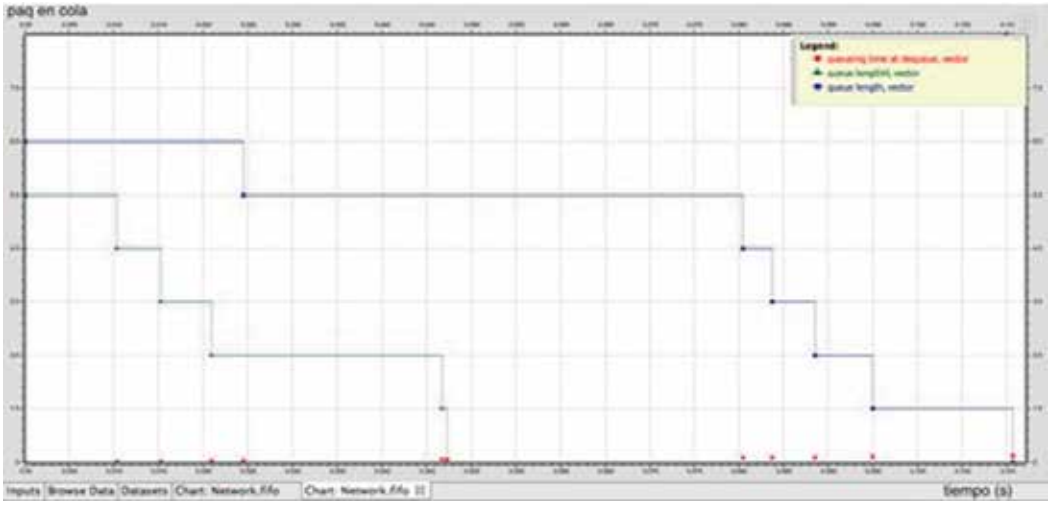
Figura 71. Comportamiento del encolamiento PQ



6.4.3. Gráfica LLQ

Para el algoritmo de encolamiento LLQ, es visible que hay extracción de paquetes en paralelo para ambas colas, aunque una tiene mayor prioridad. La Figura 72 presenta un solo paquete que hace esto, ya que para mayor comprensión de la simulación, esta se realizó con pocos paquetes.

Figura 72. Comportamiento del encolamiento LLQ



6.5. Comentarios finales

OMNeT++ ostenta ser un simulador de gran potencial. Al ser libre, permite que los usuarios desarrollen nuevas ideas y nuevos servicios. Aunque el manual presenta información bastante favorable, se tiene poca información sobre desarrollo, lo

cual dificulta sustantivamente el desarrollo eficiente de las simulaciones, ya que es necesario buscar mucha información para satisfacer las demandas que requiere una implementación como tal.

Con las diferentes simulaciones realizadas se puede comprender tan solo una parte de los servicios que OMNeT++ presta para la academia; por tanto, es necesario presentar proyectos a largo plazo que logren maximizar el uso ideal de este simulador.

También es claro que entre los simuladores de licencia libre, este es uno de los más potentes, no solo porque permite al usuario desarrollar sus propios códigos, sino también por su interfaz gráfica, definida por módulos, que permite definir iconos personalizados para una comprensión más ágil.

En comparación con otros simuladores, se puede decir que OMNeT++ se desarrolló pensando en el futuro de las redes, que es un simulador actual y que tiene todas las herramientas necesarias para hacer trabajos tanto complejos y densos, como pequeños, para el fácil entendimiento de las personas que están iniciando la obtención de conocimientos orientados a redes de datos. No obstante, OMNeT++ presenta una dificultad extensa para su aprendizaje, porque es necesario programar en C++ y porque adicionalmente plantea un nuevo lenguaje para el desarrollador, el lenguaje *Ned*, que aunque facilita la visualización final y facilita métodos, afecta directamente el desarrollo rápido de las implementaciones.

Por otra parte, es interesante ver a OMNeT++ como un simulador que tiene la posibilidad de movilizarse por todas las capas de OSI, por lo que tanto desarrolladores de software como ingenieros en telecomunicaciones, entre otros profesionales, tienen la posibilidad de utilizar la herramienta, con logros positivos para el diseño y modelamiento de nuevos recursos.

Con respecto al análisis de las simulaciones realizadas con los esquemas de encolamiento y el manejo de la herramienta se llegó a las siguientes conclusiones:

- Se encontró que en la mayor cantidad de los casos, cuando se implementan aplicaciones de diferentes tipos, el encolamiento LLQ (*LowLatencyQueuing*) es más propicio de aplicar, ya que permite a todas las aplicaciones llegar a su destino final, permitiendo a la vez, a las más sensibles, tener la oportunidad de llegar con tiempos efectivos.
- Se puede apreciar que aunque los algoritmos de encolamiento LLQ son, en la mayor cantidad de los casos, más propicios para su implementación, cuando hay aplicaciones críticas, que necesitan mayor priorización y adicional a ello, las aplicaciones ocurren en ráfagas de tiempo relativamente pequeñas, el algoritmo PQ (*PriorityQueuing*), presenta mejores resultados, ya que provee tiempos más rápidos de respuesta que LLQ, para las aplicaciones sensibles al retardo.
- Se resaltan las ventajas de tener simuladores de eventos discretos para la implementación de eventos de la red, ya que gracias a ser esta una característica de

OMNeT++, le permite a los usuarios entender cada movimiento y comportamiento de los paquetes en la red.

» El lenguaje *Ned* provee facilidad en el diseño de la red, ya que gracias a sus funciones permite utilizar todas las simulaciones desarrolladas, como librerías, sin la necesidad de copiar código, lo que convierte a OMNeT++ en una herramienta de simulación, no solo potente si no también eficiente.

» Al necesitar el conocimiento previo de dos lenguajes para la edición o creación de proyectos, OMNeT++ se convierte en una herramienta compleja de manipular para personas que no poseen conocimientos en C++.

Referencias

Abusubaih, M. (2010). On Performance Anomaly in 802.11 Wireless LANs: Problem and Solution Approaches. Fourth International Conference on Next Generation Mobile Applications, Services and Technologies, NGMAST 2010, (pp.208-212). Piscataway, NJ: IEEE

Aranud, J.P. (2012). Academic Research and teaching with OPNET Technologies. [Online]. Recuperado de http://deptinfo.cnam.fr/Enseignement/CycleSpecialisation/IRE/University%20Programs.htm

Bateman, M., & Bhatti, S. (2010). TCP Testing: how well does ns2 match reality? En 24th IEEE International Conference on Advanced Information Networking and Applications AINA, (pp.276-284). Piscataway, NJ: IEEE

Belzarena, P., & González-Barbone, V. (2006). Incorporación de un simulador grafico de redes en un objetivo de aprendizaje reutilizable [ponencia en Tecnologías aplicadas a la enseñanza de la electrónica (TAEE), Madrid, España, 2006]. Recuperado de http://e-spacio.uned.es:8080/fedora/get/taee:congreso-2006-1124/SD104.pdf

Blandon, D. (2010). Marco de Referencia. En *Medición de la calidad de servicio en redes de proxima generación*, (pp. 15-48). Cali, Colombia: CINTEL

Bragg, A. (2000). Which network design tool is right for you? IT Professional 2(5), 23-32

Breslau, L. (2000). Advances in Network Simulation. Computer 33(5). 59-66

Brugge, J., Paquereau, L., Heegaard, P.E. (2010). Experience report on implementing and simulating a routing protocol in ns-2 and ns-3. En Advances in System Simulation (SIMUL), 2010 Second International Conference on, (pp. 88-93). Piscataway, NJ: IEEE

Chang, X. (1999). Network simulations with Opnet. En P.A. Farrington, H.B. Nembhard, D.T. Sturrock, y G.W. Evans [Eds.] Proceeding of the 1999 Winter Simulation conference, (pp. 307-314). New York, NY: Association for Computing Machinery

Cruz, E., Camara, D., & Guardia, H. (2009). Providing billing support in wimax mesh networks. Wireless and Mobile Computing, Networking and Communications, 2009. WIMOB 2009. IEEE International Conference on (pp. 161-166). Piscataway, NJ: IEEE.

Cuellar, J.C. (2011). Analysis of quality of service parameters using a simulation tool. Entre Ciencia e Ingeniería 5(9), 120-131

David, C.J., Sanmartín, M.P., & Marquez, D.J. (2010). Model of QoS on NGN: An analysis of performance. EN Electronics, Robotics and Automotive Mechanics Conference (CERMA), 2010 (pp. 271-276). Piscataway, NJ: IEEE

Djenane, N., Benaouda, A., Harous, S. (2009). Simulation of a VPN Implementation based on MPLS Protocol, a case study: VPN-MPLS for msn-at. En MoMM

'09 Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia (pp. 589-593). New York, NY: ACM

Education Technology and Computer Science (ETCS), 2010 Second International Workshop on. (pp.15-18). Piscataway, NY: IEEE

El-Darieby, M., Petriu, D., Rolia, J. (2002). A hierarchical distributed protocol for MPLS path creation. En Computers and Communications, 2002. Proceedings. ISCC 2002. Seventh International Symposium on (pp. 920-926). Piscataway, NJ: IEEE

Fan, L. & Taoshen, L. (2009). Implementation and performance analyses of anycast QoS routing algorithm based on genetic algorithm in NS2, En Second International Conference on Information and Computing Science, ICIC 2009 [V.3], (pp. 368-371). Piscataway, NJ: IEEE

Frabs. (2008, octubre 23). *FIFO Queue* [en línea] Recuperado de http://dev.emcelettronica.com/fifo-queue

Free Software Foundation (2012, septiembre). La definición de software libre. L. Arteaga (trad.) [en línea]. Recuperado de http://www.gnu.org/philosophy/free-sw.html

García, A., Escobar, L., Navarro, A., & Vásquez, A. (2011). Método de evaluación y selección de herramientas de simulación de redes. *Sistemas & Telemática, 9*(16), 55-71. Disponible en http://www.icesi.edu.co/revistas/index.php/sistemas_telematica/article/view/1029

GNS3. (2012a). Screenshots [Online]. Recuperado de http://www.gns3.net/screenshots/

GNS3. (2012b). What is GNS3 ? [Online]. Recuperado de http://www.gns3.net

Goldstein, C., Leisten, S., Stark, K., & Tickle, A. (2005). Using a Network Simulation Tool to engage students in Active Learning enhances their understanding of complex data communications concepts. En Proceedings of the 7th Australasian Conference on Computing Education (pp.223-228). Darlinghurst, Australia: Australian Computer Society

Guo, Z., & Zeng, H. (2009). Simulation and analysis of weighted fair queueing algorithms in opnet. En Computer Modeling and Simulation, 2009. ICCMS '09. International Conference on. (pp. 114-118). Piscataway, NJ: IEEE.

Hammoodi, I.S., Stewart, B.G., Kocian. A., & McMeekin, S.G. (2009). A comprehensive performance study of Opnet modeler for zigbee wireless sensor networks. En Next Generation Mobile Applications, Services and Technologies, 2009. NGMAST '09. Third International Conference on (pp.357-362). Piscataway, NJ: IEEE

Henderson, T., Roy, S., Floyd, S., & Riley, G. (2006), ns-3 project goals. En WNS2 '06 Proceeding from the 2006 workshop on ns-2: the IP network simulator (Art. 13). New York, NY: ACM

Hlupic, V. (2000). Simulation software: an operational research society survey of academic and industrial users. En Proceedings of the 2000 Winter Simulation Conference, (pp. 1676-1683). San Diego, CA: Society for Computer Simulation International

In Search of the cert (2012, abril 30). [blog]. Recuperado de http://insearchofthecert.blogspot.com/2012_04_01_archive.html

INET Framework (2012, agosto 7). Download [en línea]. Recuperado de http://inet.omnetpp.org/index.php?n=Main.Download

Information Sciences Institute - University of Southern California [ISI]. (1981). *Internet protocol Darpa Internet program protocol specification [RFC 791]*. Arlington, CA: Darpa. Disponible en http://www.ietf.org/rfc/rfc791.txt

Information Sciences Institute - University of Southern California [ISI] (2011). The Network Simulator: Building Ns [en línea]. Recuperado de http://isi.edu/nsnam/ns/ns-build.html

Khan, K., Zaman, R.U, & Reddy, A.V (2008). Performance comparison of on-demand and table driven ad hoc routing protocols using Nctuns. En Computer Modeling and Simulation, 2008. UKSIM 2008. Tenth International Conference on (pp.336-341). Piscataway, NJ: IEEE

Khan, K., Zaman, R.U., Reddy A.V., Reddy, K.A., & Harsha, T.S. (2008). An efficient destination sequenced distance vector routing protocol for mobile ad hoc networks. En Computer Science and Information Technology, 2008. ICCSIT '08. International Conference on (pp.467-471). Piscataway, NJ: IEEE

Kulgachev, V., & Jasani, H. (2010). 802.11 networks performance evaluation using Opnet. En SIGITE '10 Proceedings of the 2010 ACM conference on Information technology education (pp.149-152). New York, NY: ACM.

LAN/MAN Standards Committee (2006). *IEEE Standard for Local and metropolitan area networks - Virtual Bridged Local Area Network*. New York, NY: IEEE Computer Society

Lessmann, J., Janacik, P., Lachev, L., & Orfanus, D. (2008), Comparative study of wireless network simulators. En ICN '08 Proceedings of the Seventh International Conference on Networking (pp.517-523). Washington, DC: IEEE Computer Society

Mahasweta, R. (2008). An algorithm to enhance QoS for streaming video over WLans. Advances in Electrical and Electronics Engineering - IAENG [Special Edition of the World Conferences on Engineering and Computer Science], 1-10.

Márquez, E., Placido, R., & Sampaio. P. (2009). Visual Network Simulator (VNS): A GUI to QoS simulation for the ns-2 simulator. En Computer Systems and Applications, 2009. AICCSA 2009. IEEE/ACS International Conference on (pp. 342-349). Piscataway, NJ: IEEE

Mehta, S., Ullah, N., Kabir, H., Sultana, N., & Sup, K. (2009). A case study of networks simulation tools for wireless networks. En AMS '09 Proceedings of the

2009 Third Asia International Conference on Modelling & Simulation (pp. 661-666). Washington, DC: IEEE Computer Society

Moorey, P. (2008, marzo 30). Congestion Management and Queuing - LLQ [en línea]. En *Cisco CCNP ont exam study notes* [blog]. Recuperado de http://ccnpont.blogspot.com/2008/03/congestion-management-and-queuing-llq.htmlClaros V., José Ignacio [Editor] (2012). Diseño + Tecnología - 5o Encuentro Internacional de Investigación en Diseño (Oct.25 - 27, 2012). Resúmenes de las Ponencias. Cali, Colombia: Universidad Icesi. 57p., 21x21cm. ISBN xxx - xxxx

Diseño + Tecnología - 5o Encuentro Internacional de Investigación en Diseño (Oct.25 - 27, 2012). Resúmenes de las Ponencias. 2012. Universidad Icesi, Cali-Colombia. ISBN XXXX-XXXX Impreso en Colombia

Las ponencias que corresponden a los resúmenes presentados en esta publicación pueden ser consultados en http://www.icesi.edu.co/revistas/index.php/sistemas_telematica/issue/view/160 . Los artículo ahí publicados pueden ser descargados y utilizados siempre y cuando se incluya el título del artículo y los nombres de sus autores y se cite como fuente a la revista Sistemas & Telemática, Vol.10 No.22. ISSN 1692-5238

Rector Francisco Piedrahita Plata Secretraria General María Cristina Navia Klemperer DIrector Académico José Hernando Bahamón Lozano Editor José Ignacio Claros V. Diseño y diagramación Iván Abadía Quintero Ilustración de la portada Javier Aguirre Impresión Litocencoa SAS

Universidad Icesi - Facultad de Ingeniería Calle 18 No.122-135, Cali (Colombia) editorSyT@icesi.edu.coNational Chiao Tung University [NSTU]. (2010, noviembre 9). NCTUns 6.0 Network Simulator and Emulator [En línea]. recuperado de http://nsl.csie.nctu.edu.tw/nctuns.html

Nichols, K., Blake, S., Baker, F., & Black, D. (1998). *Definition of the differentiated services field (DS Field) in the IPv4 and IPv6 Headers*. US: RFC

ns-3 (2010, noviembre 16). ns-3 generator [Online]. recuperado de http://www.nsnam.org/wiki/index.php/Ns3Generator

ns-3 (2012). ns-3 [Online]. recuperado de http://www.nsnam.org/

OMNeT++ Community. (2011a), *OMNET++ User Manual*, version 4.2 [en línea]. Recuperado de http://www.omnetpp.org/doc/omnetpp/manual/usman.html

OMNeT++ Community (2011b, marzo 27) . *cComponent Class Reference* [en línea]. Recuperado de http://www.omnetpp.org/doc/omnetpp/api/classcComponent.html

OMNeT++ Community. (2012). Graphical Ned editor [en línea]. Recuperado de http://www.omnetpp.org/images/screenshots/img1.png

Pan, J. (2008, noviembre 24). A Survey of Network Simulation Tools: Current Status and Future Developments. [Online]. recuperado de http://www.cse.wustl.edu/~jain/cse567-08/ftp/simtools/index.html

Perez-Hardy, S. (2003). The Use of Network Simulation to Enhance Network Curriculum. En CITC4 '03 Proceedings of the 4th conference on Information technology curriculum (pp. 93-95). New York, NY: ACM

Phillips, C. (2007). A review of High Performance Simulation Tools and Modeling Concepts. En Recent Advances in Modeling and Simulation Tools for Communication Networks and Services (pp. 29-47). New York, NY: Springer

Piotr, A. Stankiewicz, R., Cholda, P., & Jajszczyk, A. (2011). QoX: What is it Really? *IEEE Communications Magazine, 49*(4), 148-158

Qun, Z. & Wang-Jun. (2008). Application of ns2 in education of computer networks. En Advanced Computer Theory and Engineering, 2008. ICACTE '08. International Conference on (pp.368-372). Piscataway, NJ: IEEE

Rhee, B., Cho. S., Xianshu, J., & Han, S. (2009). QoS-aware router combining features of conventional routing and flow-aware. En Advanced Information Networking and Applications Workshops, 2009. WAINA '09. International Conference on (pp. 413-418). Piscataway, NJ: IEEE

Rong, B., Lebeau, J., Bennani, M., Kadoch, M., Elhakeem, A.K. (2005). Modeling and simulation of traffic aggregation based sip over MPLS network architecture. En Simulation Symposium, 2005. Proceedings. 38th Annual, (pp. 305-311). Piscataway, NJ: IEEE

Sameh, A., Wagih, S., & Salama, Q. (2010). Ahmed Dealing with quality of service in hybrid wired-wireless networks. En NETAPPS '10 Proceedings of the 2010 Second International Conference on Network Applications, Protocols and Services, (pp. 105-109). Washington DC: IEEE Computer Society

Shin, J-Y., Jang, J-W., & Kim, J,M. (2009). Result base on ns2, Simulation and Emulation verification. En New Trends in Information and Service Science, 2009. NISS '09. International Conference on (pp. 807-811). Piscataway, NJ: IEEE

Tanenbaum, A. (2003). G, Trujano [Ed.]. *Redes de computadoras* [4a ed.], México DF. México: Pearson

Union Internacional de Telecomunicaciones - Oficina de normalización [ITU-T]. (2011a). *Recomendación Y.1541*. [Online]. Recuperado de http://www.itu.int/rec/T-REC-Y.1541-201112-I/en

Union Internacional de Telecomunicaciones - Oficina de normalización [ITU-T]. (2011b). *Recomendación Y.1540*. [Online]. Recuperado de http://www.itu.int/rec/T-REC-Y.1540/en

Varga, A. (2011). OMNeT++. Installation Guide. Version 4.2.2 [en línea]. Recuperad de http://omnetpp.org/doc/omnetpp/InstallGuide.pdf

Varga, A., & Hornig, R. (2008). The Omnet++ discrete event simulation system. En Simutools '08, Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops (art. 60). Bruselas, Bélgica: ICST.

vom Lehn, H., Weingartner, E., & Wehrle, K. (2008, julio). Comparing recent network simulators: A performance evaluation study (technical report). Aachen, Alemania: RWTH Aachen University. Disponible en https://www.comsys.rwth-aachen.de/fileadmin/papers/2008/AIB-2008-16.pdf

Wang, A. & Liu, Z. (2010). Analysis and utilizing of the error models in network education with ns3. En

Wang, J. & Bao, L. (2005). Layer-2 mobility management in hybrid wired/wireless systems. En *Quality of Service in Heterogeneous Wired/Wireless Networks, 2005. Second International Conference on* (p.40). Piscataway, NJ: IEEE

Wang, S-Y., & Chou, C-L. (2009). Nctuns 5.0 network simulator for advanced wireless vehicular network researches. Mobile Data Management: Systems, Services and Middleware, 2009. MDM '09. Tenth International Conference on (pp.375-376). Piscataway, NJ: IEEE.

Wang, S-Y., Chou, C-L., Lin, C-C., & Huang, C-H. (2010, enero 15). The Protocol Developer Manual for the NCTUns 6.0 Network Simulator and Emulator. Hsinchu, Taiwan: National Chiao Tung University. Disponible en http://nsl10.csie.nctu.edu.tw/support/documentation/DeveloperManual.pdf

Wang,S-Y., Wang, P-F., Li, Y-W., & Lau, L-C. (2011). Design and Implementation of a more realistic radio Propagation Model for Wireless vehicular networks over the Nctuns network simulator. Wireless Communications and Networking Conference (WCNC), 2011 IEEE (pp. 1937-1942). Piscataway, NJ: IEEE

Xia, J-b., Li, M-h., Wan, L-j. (2008). Research on MPLS VPN networking applications based on Opnet. En Information Science and Engineering, 2008. ISISE '08. International Symposium on (pp.404-408). Piscataway, NJ: IEEE

Yaw-Chung Chen, Y-C., Li, S-L., & Chen K-T (2010). An efficient source allocation approach for QoS support in p2ptv systems. En Computer Software and Applications Conference (COMPSAC), 2010 IEEE 34th Annual, (pp. 509-514).

Zhu, W., Dreibholz, T., Rathgeb, E.P., Zhou, X. (2008). A scalable QoS device for broadband access to multimedia Services. En Future Generation Communication and Networking, 2008. FGCN '08. Second International Conference on, [Vol.3] (pp.343-348). Piscataway, NJ: IEEE

ISBN: 978-958-8357-74-4

UNIVERSIDAD ICESI