UNIVERSIDAD POLITECNICA DE VALENCIA

DEPARTAMENTO DE COMUNICACIONES

Máster Universitario en Tecnologías, Sistemas y Redes de Comunicaciones

Caracterización y Simulación de Fuentes de Tráfico en *Smart Cities*

Autor: Miguel Alfredo Coello Ojeda

Director1: Miguel Ángel Rodríguez Hernández

Director2: Víctor Sempere Payá

Fecha de comienzo: 1/01/2014

Lugar de trabajo: E.T.S.I. Telecomunicaciones

Objetivos — Contribuir a la generación de conocimiento que se pueda compartir libremente en el ámbito educativo y científico mediante la construcción de un prototipo para la simulación de flujos de tráfico en *Smart Cities*. Beneficiar con el uso del presente trabajo al laboratorio de redes de la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universidad Politécnica de Valencia y en general a toda la Escuela.
La lectura de esta tesina quisiera encaminar una manera de abordar el uso de una herramienta de software de ámbito científico que da libertad de crear y simular las diferentes fuentes de tráfico que se pueden encontrar agregando flujos en las redes de próxima generación e intentar mostrar cómo la herramienta de software libre llamada OMNet++ es una buena opción que podemos encontrar al día de hoy.
El enfoque de aplicación a *Smart Cities*, es tendiente al estudio de la integración de los flujos de tráfico generados por las *Machine Type Communications* [1] con las fuentes de tráfico "tradicionales": Voz, vídeo y multimedia.
Se pretende que este documento sea una guía para futuros investigadores mediante la cual sea más sencillo elegir, qué módulos pre-existentes en la herramienta de software se pueden utilizar en el diseño de una simulación de fuentes de tráfico para una *model network* en un entorno *Smart City*.

Metodología —En los capítulos I y II se establece el marco teórico de la clasificación de fuentes generadoras de flujos de tráfico de información. Esta primera parte presenta conceptos que lideran las investigaciones en el ámbito científico. En el capítulo III se caracterizan las fuentes de tráfico que se pueden simular con OMNet++. Se escogen únicamente dos proyectos marco con librerías afines al planteamiento de este trabajo de investigación. Finalmente, se hace una simulación de tráfico y se explican detalles de utilización del software en el capítulo IV.

Desarrollos teóricos realizados — Este trabajo se basa en la literatura referida para la propuesta del presente documento, como son: Las recomendaciones UIT-T Q.3925, UIT-T Q.543, especificaciones del grupo 3GPP, artículos de investigación y demás documentación de los manuales OMNeT++ y de los proyectos INET y MiXiM. Se brinda al lector investigador con poca experiencia en la utilización de OMNeT++ una guía que le permita abordar de la mejor manera posible la utilización de esta herramienta de software libre para simular fuentes de tráfico en cualquiera de los tipos de escenarios de las redes modernas.

Desarrollo de prototipos y trabajo de laboratorio — Se construyó un modelo de red con módulos de los proyectos INET y MiXiM para simular flujos de tráfico: *file transfer*, vídeo, voz y de sensores. Luego se configuraron pruebas de ping para garantizar la correcta entrega de "mensajes" entre módulos. Finalmente se crearon nuevos escenarios para simular los flujos de tráfico mencionados de forma independiente y de forma agregada. Todo este proceso conllevó muchas horas de estudio, pruebas y solución de problemas como bugs del código, errores de Windows, errores en los procesos de compilación, etc.
Como parte de este trabajo de investigación, como una contribución didáctica, se han subido vídeos de la simulación a un canal de Youtube del autor y compartido públicamente.

Resultados — El material bibliográfico referenciado propuesto por el Director y el recabado de la Web, permite tener una visión global de la complejidad y el tamaño de lo que es una red de comunicaciones para

Smart City. Con respecto al software simulador, se tuvo acceso sólo a la documentación proveniente del manual de OMNet++. La lectura de los tres primeros capítulos del manual de OMNet++ resultó de gran ayuda para el entendimiento del funcionamiento del simulador, es el comienzo ideal en su aprendizaje, sin embargo una base muy sólida en el conocimiento de los lenguajes de programación C++ y Java es mucho más que preferible. El manual del proyecto INET no es muy formal en algunas secciones de su contenido. Cabe destacar que por las continuas modificaciones en el lanzamiento de nuevas versiones de OMNet++, se necesita dedicar un tiempo de investigación y minucioso trabajo para adaptar los códigos NED antiguos, lo que revela un inconveniente para replicar los experimentos. A futuro se espera que esto sea corregido.

Líneas futuras — En base a las tendencias globales observadas en el campo de las redes de comunicaciones, se espera que el estudio de su crecimiento y la evolución de los protocolos en lo concerniente a las fuentes de tráfico puedan abarcarse más fácilmente como legado del presente trabajo en futuras implementaciones de modelos de simulación. En ese camino, las actualizaciones, mejoras y la creación de nuevos módulos de simulación deberían únicamente requerir modificaciones al código pre-existente.

Una tarea intrínseca al uso de OMNet++ para conseguir simular una *model network* de alta complejidad, es el desafío a la capacidad del ordenador que ejecute la simulación. Pudiera convertirse en otra rama de investigación, el diseño de un ordenador especializado para simular *model networks* de próxima generación y posteriormente ser comercializado en el ámbito investigativo.

Puesto que la ITU-T recomienda pruebas de la QoS utilizando model networks, un trabajo que se pudiera realizar a futuro es la comprobación en el laboratorio del parámetro "Hurst" para diferentes tipos de tráfico USN. Los datos necesarios para calcular el "Hurst", que forman la serie de tiempo fractal del proceso de llegada de tráfico se tomarían del archivo de análisis ".anf" que genera el software. Habría que calcular estadísticos de los procesos aleatorios involucrando ecuaciones como (1) o (2) y los estudios [2] y [3].

En un artículo del año 2005 [4], argumentan que los modelos de Poisson son más adecuados para modelar tráfico en la Internet, puesto que estos procesos estocásticos sólo conservan la auto-similaridad en rango corto. Esas afirmaciones se basan en bases de datos de tráfico reales capturados a principios de los años 90. Aquí también se puede investigar al respecto mediante el uso de simuladores con tráfico moderno.

En cuanto al uso de OMNeT++, sería muy útil contribuir a las mejoras de la guía del usuario para incluir la resolución de errores de configuración y programación, además de reconfigurar los mensajes para hacerlos más precisos y comprensibles, en especial para quienes se inician en la utilización del software.

Abstract — To facilitate the study of networking wired and wireless next generation networks in a Smart City, where the traditional traffic digital media voice, share the stage with those formed by the links M2M (Machine-to-Machine) [5], USN[9], Ad-Hoc networks, and networks with LTE mobile technology, is essential to have the most suitable software simulation tools for research. A set of traffic sources were characterized with OMNeT++ for such task.

Autor: Coello Miguel, email: micoeoj@teleco.upv.es
Director 1: Rodríguez Miguel, email: marodrig@upvnet.upv.es
Fecha de entrega: 7-09-14

ÍNDICE

I. Conceptos introductorios y antecedentes.

La palabra tráfico se suele utilizar como abreviación de lo que en realidad se debería definir como medida de intensidad de tráfico. La intensidad de tráfico instantánea es el número de recursos ocupados en un instante de tiempo dado [6] en el que esos recursos estén disponibles. Los recursos pueden ser: Servidores, líneas, circuitos, canales, ancho de banda, capacidad de canal, paquetes de datos, etc.

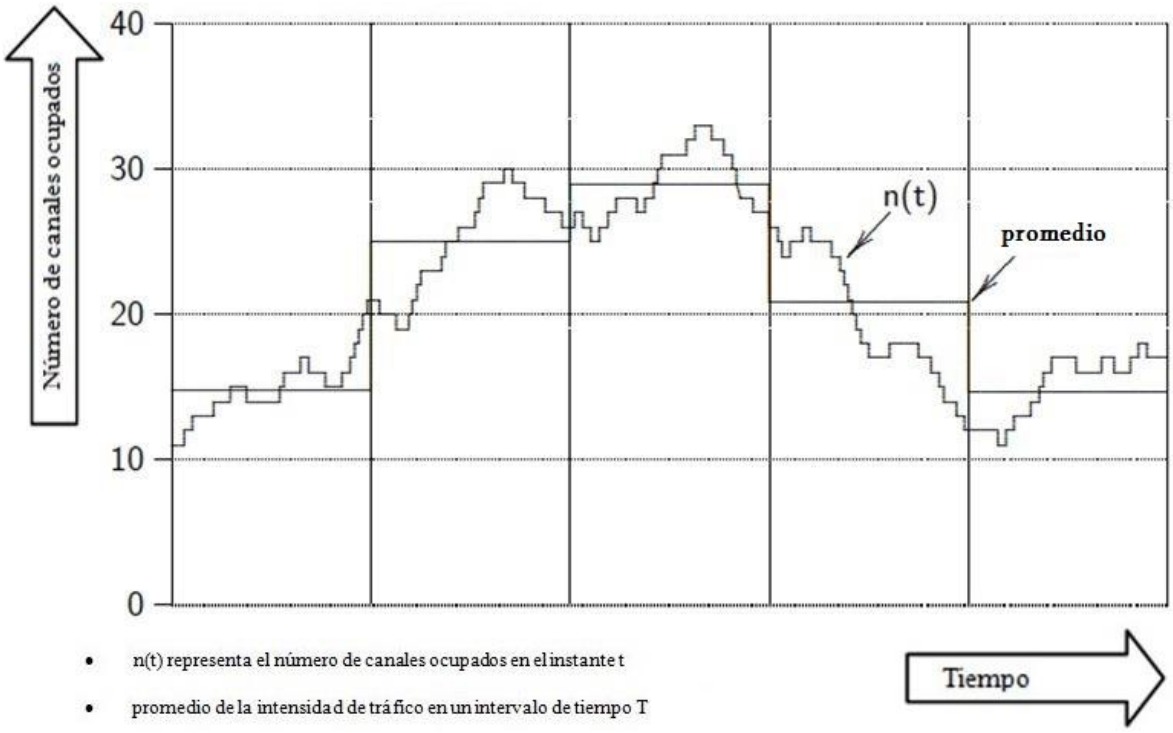


Fig.1. Intensidad de tráfico promediada en intervalos de tiempo. Fuente: "Teletraffic Engineering and Network Planning", Iversen, 2010.

A un proveedor de servicios de red le interesaría entre otras cosas: Saber si la capacidad de red instalada es suficiente para satisfacer los requerimientos de sus servicios con QoS [11], la medida del promedio de intensidad de tráfico en intervalos de segundos o hasta meses, proyectar a futuro el crecimiento de la red, proyectar oportunidades de negocio para desplegar nuevos servicios. La monitorización del flujo de tráfico no sólo permite ver qué está consumiendo la capacidad de un canal de datos: Navegar en Internet, tráfico VoIP, descargas ftp, etc. Sino que además ayuda a tener un control efectivo del tráfico de red para detectar posibles "cuellos de botella", cerrar aplicaciones con alta tasa de *bitrate* cuando se necesitan los recursos para otras aplicaciones, limitar la tasa de trasmisión de datos disponible para cada equipo, etc.

Los requerimientos de servicios se pueden traducir en parámetros de tráfico. Por ejemplo: El sensor de humedad de un refrigerador conectado a una red LTE envía datos de actualización a un servidor local o remoto en forma periódica cada hora; el envío de estos datos requerirá la utilización de recursos de la red por lo que se necesita dimensionarlos, en este caso, una cantidad

de bytes de información ocuparán la capacidad de los canales de datos de la red. La red, mediante protocolos, gestionará cuáles y cuántos de sus recursos utilizará para transportar esos bytes. En un instante de tiempo específico, o en promedio dentro de un intervalo de tiempo, se podrán observar y medir los recursos de la red que se están ocupando, no sólo con los datos generados por uno de los sensores del refrigerador sino por todos los demás tipos de fuentes de tráfico.

Los procesos aleatorios, "Tiempo de Llegada del Requerimiento de Servicio" y "Tiempo de Mantenimiento del Servicio", en una comunicación de datos son suficientes para describir las propiedades del tráfico [6]. En materia de procesos estocásticos son de especial interés aquellos procesos aleatorios que involucran la llegada o arribo de llamadas, paquetes o bytes de información. Se asume que estos procesos son independientes entre si, por ejemplo, la duración del mantenimiento de una llamada es independiente del tiempo de llegada de la llamada.

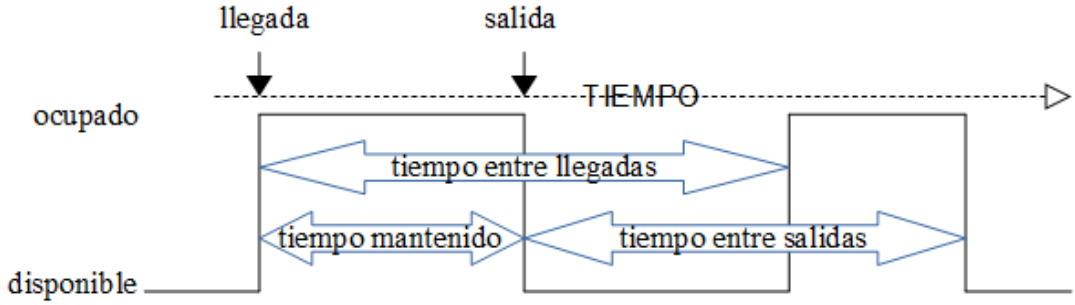


Fig.2. Terminología para los procesos de tráfico. Fuente: "Teletraffic Engineering and Network Planning", Iversen, 2010.

Las características de cada tipo de flujo de tráfico en redes de comunicaciones vienen dadas por el servicio y el contenido que está implementado sobre la red, como pueden ser: Voz, VoIP, navegación Web, transferencia de archivos (*file transfer*), *e-mail*, *peer-to-peer*, vídeo, y el generado por *ubiquitous sensor networks* (USN).

Una correcta apreciación de los efectos de cada tipo de flujo en las Redes de Próxima Generación (*NGN*) sólo es posible a través de un modelo de red que contemple la totalidad de sus características. La *model network* es la red, que simula las capacidades de forma similar a las disponibles en las actuales redes de telecomunicaciones, esto es: Una arquitectura similar, la funcionalidad y los usuarios; en resumen, los mismos medios técnicos.

El concepto *model network* viene a cubrir la necesidad de realizar pruebas de mayor complejidad sobre las redes existentes de servicios digitales, en donde hasta hace poco tiempo predominaba el tráfico de voz, el cual ha sido desplazado por el auge de las nuevas tecnologías en manos de los usuarios y el surgimiento de las *Smart Cities*. Esta complejidad se debe al equipamiento y características necesarias para soportar muchos nuevos servicios, además de los de voz y multimedia, considerando en todo momento la calidad del servicio *QoS*.

En la recomendación UIT-T Q.3925 encontramos las características que deben tener los tipos de flujo de tráfico que se deben generar para probar parámetros de *QoS* en *model networks*,

distinguiéndose aquellos tipos por servicios como la voz, el vídeo y en general, los representativos de la constelación que puede hallarse actualmente en redes móviles, vehiculares, Intranet, Internet, etc. Esta clasificación nos ayudará a relevar la importancia del software de simulación para emular esas fuentes con un alto grado de confianza.

En las redes de próxima generación, los flujos de tráfico de vídeo y datos tienen características auto-similares o *self-similar*, que se observan en el análisis de los resultados estadísticos obtenidos de la medición de la utilización de los recursos de la red.

En contraste, el flujo de tráfico de voz se describe mejor por un proceso de Poisson. El tráfico de voz digital está ampliamente cubierto en el estándar UIT-T Q.543.

Estos contrastes tienen como consecuencia la necesidad de la utilización de un método de interoperabilidad que conlleve a analizar esas características estocásticas del tráfico en forma integral. Las pruebas de las *model networks* requieren un método global denominado *global interoperability*, es decir, que considere la interoperabilidad de medios técnicos, de los servicios, y de los parámetros y clases de QoS. La consideración del tráfico moderno es un requisito fundamental para las pruebas. El grupo de estudio 11 de ITU-T se encuentra trabajando en esta actividad. [8]

Históricamente se ha pasado de redes de conmutación de circuitos a redes de conmutación de paquetes. En el devenir de las telecomunicaciones, muy probablemente el protocolo de red IP sea el más utilizado en la mayoría de las redes locales, el tráfico de datos superará al de voz en las SmartCities y se hablará mucho del Internet de las Cosas (IoT). Las NGN traerán la convergencia en servicios e infraestructura.

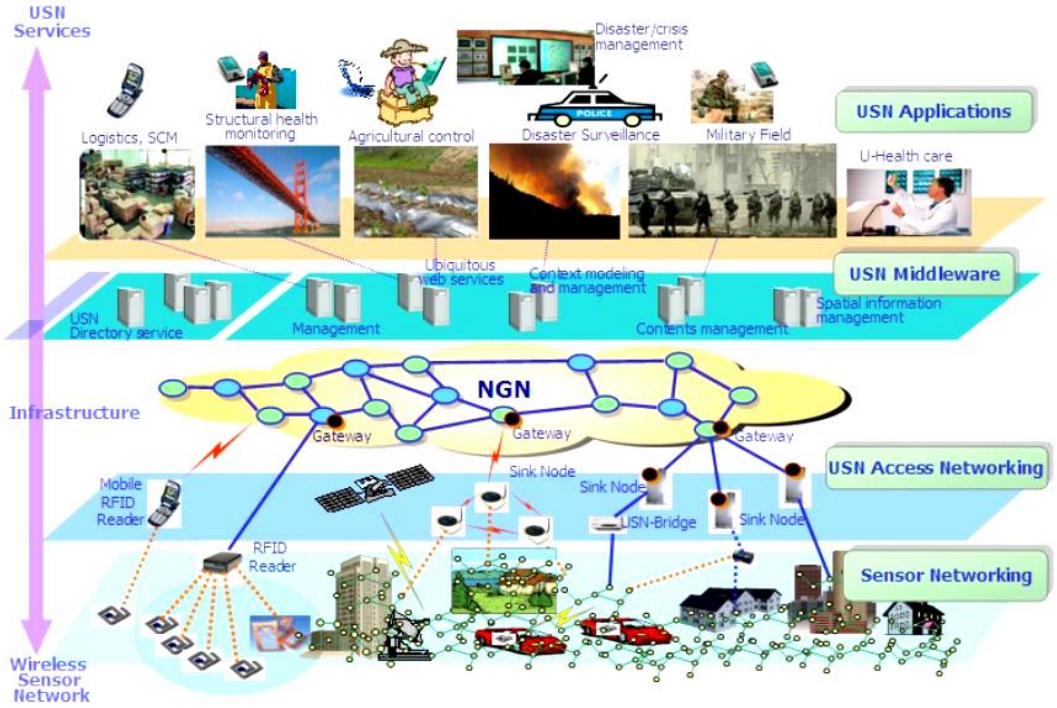


Fig.3. Esquema del avance de las Redes de Sensores. Fuente: ITU-T, 2007, www.itu.int/md/T05-NGN.GSI-DOC-0266/en

II. Clasificación Teórica de Modernas Fuentes de Tráfico.

La siguiente clasificación, entre los subcapítulos I.1 y I.7, ha sido redactada tal cual se ha traducido, en su mayor parte de la recomendación U.I.T.-T.Q.3925 con ciertas observaciones y aclaraciones personales.

I.1. *Tráfico Poisson.*

El proceso de llegada de tráfico de voz en la red de telecomunicaciones pública conmutada es el Proceso de Poisson. Esto se ha verificado por la observación a largo plazo.

Las propiedades fundamentales del proceso de Poisson son estacionarias, independientes en todos los instantes de tiempo, y simples. Este tipo de proceso aleatorio tiene dos propiedades importantes [6]:

- El número de eventos dentro de un intervalo de tiempo de longitud fija se distribuye como Poisson. Por ejemplo, el número de intentos de llamada en 1 hora.

- El tiempo entre eventos consecutivos se distribuye de manera exponencial. Por ejemplo, el tiempo entre la llegada de llamadas.

El flujo Poisson está determinado totalmente por su propio parámetro. Tanto el valor esperado (primer momento) como la varianza de la variable aleatoria con distribución de Poisson son iguales a λ. El parámetro λ es un número positivo que representa el número de veces que se espera que ocurra el fenómeno durante un intervalo dado.

El proceso de llegada de Poisson es adecuado para la representación de otros procesos de llegada, por ejemplo, el proceso de llegada de tráfico de Internet dial-up [6].

I.2. *Tráfico Self-similar.*

El tráfico *self-similar* o auto-similar, también denominado como fractal, se identifica por procesos aleatorios que presentan una característica peculiar la cual es que sin importar la escala de tiempo que se analice, los diferentes momentos de la variable aleatoria se mantienen casi invariantes. El tráfico original y el tráfico agregado mantienen lo que se conoce como memoria a largo plazo que los caracteriza.

I.2.1 *Coeficiente de Hurst*.

El coeficiente o exponente de Hurst es una medida de independencia de las series de tiempo que fue estudiada inicialmente por el científico británico Harold Edwin Hurst (1880-1978), como elemento para distinguir series fractales. Hurst descubrió que muchos fenómenos naturales exhiben un comportamiento que puede ser caracterizado por un proceso aleatorio sesgado, en el cual existe "memoria de largo plazo" entre las observaciones, es decir, que los eventos de un periodo influyen en todos los siguientes.

Posteriormente, Benoit B. Mandelbrot generalizó su trabajo y lo llamó análisis de rango reescalado (R/S), definido como un método estadístico utilizado para evaluar la ocurrencia de eventos poco comunes, dando origen a una herramienta ideal para procesos físicos, financieros, y demás, por lo que puede ser usado en cualquier serie de tiempo.[10]

El Análisis de Rango Reescalado R/S permite encontrar el parámetro o exponente de Hurst, el cual es un valor numérico que hace posible determinar la auto-correlación en una serie de datos.

Tanto el tráfico de los datos como el de vídeo, que crean los nuevos tipos de tráfico de llegada que circulan en las NGN, son flujos de tráfico auto-similar. [10] El proceso de llegada del flujo de tráfico es considerado como auto-similar, estadísticamente hablando, porque la función de distribución del proceso original y los procesos agregados son iguales. El proceso de llegada agregado es creado por el promediado del proceso original en bloques de tamaño *m* (el bloque debe ser parte del proceso original). [2]

La característica más importante del proceso de llegada auto-similar es el parámetro Hurst. [11]

El conocimiento del valor del parámetro Hurst posibilita generar flujo de tráfico auto-similar con las características especificadas.

El parámetro Hurst, H, podría determinarse de la siguiente manera:

$$\ln\left(\frac{D\left(X^{m}\right)}{D(X)}\right)=(2H-2)\ln(m) \tag{1}$$

Donde $D(X^{(m)})$ es la varianza del flujo agregado y $D(X)$ es la varianza del flujo original. La expresión ($2H$-2) es el coeficiente de la pendiente de la línea.

En [3], se determina el parámetro Hurst a partir de la pendiente de la gráfica que se obtiene relacionando la función distribución acumulada complementaria con el tiempo entre llegadas, esto es: Función vs Variable Aleatoria, ambas llevadas a escala logarítmica.

$$H=1-\frac{pendiente}{2} \tag{2}$$

I.2.2 *Tráfico LAN Ethernet*.

Las redes LAN con sistemas Ethernet en su capa física y de enlace de datos del modelo OSI, son de amplísima utilización en la actualidad. Esto se debe a la facilidad para modificar su topología y al control de acceso al medio no centralizado que se traduce en una alta tolerancia a fallos.

Investigadores de Bellcore [11], en 1996, llegaron a la conclusión de que mientras más cargada esté la red Ethernet, más alto será su grado de auto-similaridad, es decir, que el coeficiente de Hurst será cada vez más cercano al valor 1 y no menor que 0,5.

II.3. *Características del tráfico de voz de acuerdo con la Recomendación UIT-T Q.543.*

El flujo de tráfico de origen para los abonados RDSI no se han considerado ya que hay un número significativamente menor de abonados RDSI que de los abonados PSTN. La intensidad del flujo de tráfico en Erlangs y los intentos de llamadas en hora punta (BHCA) se muestran en la Tabla 1 para

el tipo normal de carga de nivel A. Para probar los parámetros de QoS de la *model network* para condiciones de sobrecarga, los valores de la Tabla 1 se deben aumentar hasta un 25% en Erlangs y el 35% en los intentos de llamadas en hora punta (tipo de carga de nivel B), de conformidad con la cláusula 2.1.2.2 de UIT-T Q.543.

Clase de tráfico	**Tipo de flujo**	**Promedio de intensidad de tráfico (Erlang)**	**Promedio de intentos de llamada en la hora ocupada (BHCA)**
1	Poisson	0.03	1.2
2	Poisson	0.06	2.4
3	Poisson	0.10	4
4	Poisson	0.17	6.8

Tabla 1: Características de origen del tráfico de voz. Fuente: Recomendación UIT-T Q.543, tabla 1a.

La superposición de las clases de flujo de tráfico se debe utilizar durante las pruebas en la *model network*. La superposición de N procesos de Poisson independientes será el proceso de Poisson de acuerdo con el teorema de superposición Poisson [6]. Las contribuciones de las clases de flujo de tráfico en el flujo de superposición están determinadas por las características concretas de la red o parte de la red que deba ser probada. Este flujo de tráfico debe ser generado con características tomadas del tráfico de origen.

II.4. *Características del tráfico de voz en NGNs.*

Las características de flujo de tráfico de voz en las NGN son determinadas por la transferencia de tráfico de voz sobre protocolos IP (VoIP). La contribución del usuario VoIP se incrementa constantemente, pero las características de tráfico de voz de la Tabla 1 permanecen muy estables.

La observación estadística de los flujos de tráfico de VoIP en las redes reales demostró que la llegada del flujo de tráfico podría ser descrita por el proceso de Poisson, incluso para redes todo IP [10].

Los métodos de supresión de silencio podrían utilizarse durante el procesamiento del tráfico VoIP. Los métodos utilizados en UIT-T P.59 pueden ser considerados para la generación de tráfico VoIP para la *model network*. La duración de conversación ininterrumpida (*talk-spurt*) y la duración de la pausa presentan la distribución exponencial de conformidad con UIT-T P.59. En la conversación, la duración de la pausa es de 61,47% y la duración del silencio mutuo es 22.48%.

II.5. *Características del tráfico de datos.*

II.5.1. *Características del tráfico "www".*

El tráfico "www" es auto-similar con parámetro Hurst *[2] H=0.7-0.9 [10].

II.5.2. *Características del tráfico file transfer.*

El tráfico *file transfer* es auto-similar con parámetro Hurst H=0.85-0.95 [10].

II.5.3. *Características del tráfico e-mail.*

El tráfico *e-mail* es auto-similar con parámetro Hurst H=0.75 [10].

II.5.4. *Características del tráfico peer-to-peer.*

El tráfico *peer-to-peer* es auto-similar. Como un ejemplo del tráfico P2P, el tráfico de Skype es auto-similar con H=0.6 para el tiempo entre llegadas y H=0.7 para la longitud de los paquetes. [10]

II.6. *Características del tráfico de vídeo.*

La observación estadística del flujo de tráfico IPTV probó que es auto-similar tanto para el tráfico *multicast* como el *unicast* [10]. Los valores del parámetro Hurst son H=0.55-0.6 para el tráfico *multicast* y H=0.75-0.8 para el tráfico *unicast*. Las características del flujo del tráfico para la generación de tráfico de vídeo en la *model network* se muestran en la Tabla 2.

Tipos de tráfico	**Tipos de flujos**	**Valores de Hurst**
Multicast	Auto-similar	0.55 – 0.6
Unicast	Auto-similar	0.75 – 0.8

Tabla 2: Características del flujo de tráfico IPTV. Fuente: Rec. UIT-T Q.3925.

Las características de los flujos de tráfico *unicast* auto-similares podrían ser recomendadas para el flujo de tráfico IPTV agregado [10]. El método ON / OFF [2] podría ser recomendado para la generación de tráfico IPTV en la *model network*.

II.7. *Características del tráfico en USN.*

En un futuro próximo, la red de sensores ubicuos (USN) será un gran generador de tráfico para las redes públicas. La investigación sobre el flujo de tráfico USN demostró que el flujo de tráfico en las redes USN puede ser auto-similar. Los flujos de tráfico son auto-similar con H = 0,67-0,69 para la USN, tanto con nodos sensores estacionarios de telemetría, así como una mezcla de nodos sensores estacionarios y con movilidad [10]. Se propone una clasificación de las clases de tráfico para la USN en la Tabla 3.

Clases de tráfico USN	**Tipo de flujo**	**Valores de Hurst**
Voz	Para estudio futuro	Para estudio futuro
Señalización	Auto-similar	0.83
Sensores estacionarios de telemetría	Auto-similar	0.67
Sensores de telemetría combinados	Auto-similar	0.69
Fotografías	Para estudio futuro	Para estudio futuro
Reconfiguración	Auto-similar	0.83
Posicionamiento local	Para estudio futuro	Para estudio futuro

Tabla 3: Características del flujo de tráfico de una USN. Fuente: Rec. UIT-T Q.3925.

III. Clasificación de fuentes de tráfico en OMNET++.

La descripción de cada uno de los siguientes módulos ha sido recabada principalmente en base a diversas fuentes como son: La documentación de cada proyecto incluida en los archivos ".pdf ", ".html", etc.; los propios sitios web de descarga del simulador y de los proyectos, y el código de los archivos ".ned" que conforman la parte fundamental del entorno de la herramienta OMNet++.

Los futuros usuarios de OMNet++ pueden encontrar en la Internet, varios tutoriales en vídeo y texto con ejemplos de proyectos que han sido compartidos por sus creadores en varios idiomas. OMNet++ es software libre; esto tiene una ventaja, que es la colaboración que existe entre los usuarios desarrolladores para mejorar partes o todo un proyecto, lo cual constituye un valor agregado de OMNet++, a pesar de que la información no esté suficientemente detallada ni organizada. El investigador que se inicie en el uso de OMNet++ necesitará de todas estas múltiples fuentes colaborativas para lograr el conocimiento que de esta herramienta se requiere.

Existen muchos proyectos para OMNet++, basta visitar el sitio web "omnetpp.org" para darse una idea de la amplitud de ramas de simulación que existen actualmente y que constantemente prosperan. Cabe comentar que existen varios proyectos creados por universidades u otros centros de investigación como por ejemplo el proyecto Core de la Universidad de Hamburgo para simulación de redes de tiempo real [12].

Para este trabajo, en el que se ha utilizado la versión 4.2.2 de OMNet++, nos centraremos en el proyecto INET que es muy popular para redes de comunicaciones y en el proyecto MiXiM que propone algunos módulos wireless para redes de nueva generación, sobre todo redes de sensores. Todo el software puede ser descargado por medio de los hipervínculos del sitio web mencionado.

Para el lector interesado, se recomienda revisar, tanto para el proyecto INET como para el MiXiM, el hipervínculo *Networks* que reseña todas las redes creadas para su simulación y que se encuentra en el archivo index.html de cada directorio: En "c:\omnetpp-4.2.2\samples\inet\doc\" y en "c:\omnetpp-4.2.2\samples\mixim\doc\". En cuanto a la documentación general, se debe recurrir al directorio "c:\omnepp-4.2.2\doc".

Como se puede leer en el manual de OMNet++ [13], existen módulos simples y compuestos. Un módulo compuesto puede contener, módulos simples pero también otros módulos compuestos, como sus submódulos; todos ellos necesarios para convertirse en una entidad de simulación que pasa a llamarse *Network*. Los módulos necesitan intercambiar "mensajes" a través de conexiones dentro de un módulo compuesto. Estos mensajes sirven para la comunicación entre protocolos de distintas capas. Por ejemplo, un módulo compuesto de Red incluirá submódulos y la comunicación de control tanto con la capa MAC como con la capa de Transporte. La transferencia de información entre objetos módulos, ya sea hacia el interior de un módulo compuesto o entre ellos se realiza por la "señales": *sentPk* y *rcvdPk* del tipo *cPacket*.

Para diseñar redes con módulos que cumplan necesidades específicas hará falta más que el lenguaje NED. El investigador debe programar las clases y cabeceras en lenguaje C++. Saber esto es muy importante para entender los alcances de la investigación con esta herramienta de software.

Las características de los módulos que implementan fuentes de tráfico se presentan más adelante. Una descripción detallada de su interconectividad modular sólo sería posible con un esfuerzo que sobrepasaría el tiempo propuesto para este trabajo. Hubiese sido deseable que cada autor hiciera esa descripción. Sin embargo, es posible indagar en el código de los archivos cabecera ".h" y archivos fuente ".cc", programados en C++, que se encuentran entre los archivos de cada proyecto marco.

Para la lectura de la siguiente caracterización, se debe tener en cuenta terminología de redes y lenguaje de programación C++; podría suceder, por ejemplo, que la palabra paquete se refiera a una clase C++, a la unidad de transmisión de datos o a todo un proyecto creado para OMNet. La salvedad a este respecto está en la contextualización.

En un módulo *network*, los módulos compuestos en donde están implementadas las fuentes de tráfico, que veremos más adelante, forman parte del sistema jerárquico de la programación en lenguaje NED. Hay que considerar que un módulo simple, generador de tráfico, estará necesariamente "incluido" en un módulo compuesto como puede ser un *host* o un *router*, los cuales intervienen en la construcción de la topología de la red que se visualiza en forma comprensiva en la interface gráfica del simulador. Cada módulo, sin importar su naturaleza tiene en su código punteros y llamadas a funciones y librerías del árbol de carpetas del proyecto asociado (llámese este INET, MIXIM o cualquier otro). Es decir, los módulos generadores de tráfico no se encuentran actuando en un modelo de red de manera independiente sino que siempre formarán parte de un módulo complejo como los que hemos mencionado. Al tener esto en cuenta, la siguiente clasificación expondrá como unidades básicas a aquellos módulos simples y mencionará la implicación que tienen dentro de un modelo de red. Por ejemplo, un generador de tráfico para la aplicación FTP, se definirá como la configuración de una serie de parámetros ligados al módulo interface *ITCPApp* dentro del módulo compuesto *StandardHost*.

III.1. *Paquete marco INET para OMNEST/OMNET++.*

El marco de trabajo INET es un paquete de código abierto que se utiliza para la simulación de redes de comunicación, escrito para el sistema de simulación OMNEST/OMNeT++. El marco INET contiene modelos para varios protocolos de Internet como IP, ICMP, IGMP, ARP, TCP, UDP, Ethernet, PPP y MPLS con LDP o RSVP-TE. [14]

Dentro de INET también encontramos módulos compuestos prediseñados como son *router, switch, host, channel, hub*, que están compuestos de uno o varios de los módulos de protocolo mencionados en el párrafo anterior según el caso.

Con un conocimiento experto en el lenguaje C++ y Java, todas estas clases de módulo sirven como base para la programación de nuevos módulos compuestos para la experimentación con nuevos protocolos de comunicaciones.

Al ser OMNet++ un software de código abierto, está sujeto a mejoras y modificaciones constantes. Son varios los creadores que han contribuido con INET. Estas contribuciones están contenidas como proyectos en las carpetas de su directorio.

ethernet/	Ejemplos de redes Ethernet
inet/	Ejemplos de redes basadas en TCP/IP
ipv6/	Ejemplos de redes basadas en IPv6
sctp/	Ejemplos con protocolo SCTP
rtp/	Ejemplos con protocolo RTP (not yet integrated)
emulation/	Ejemplos de redes que utilizan interfaces para conectarse con redes reales en el exterior
adhoc/	Redes móviles y ad-hoc (incomplete)
wireless/	Ejemplos con protocolo IEEE 802.11
mpls/	Ejemplo de redes con protocolos de transporte MPLS/LDP/RSVP-TE
ospfv2/	Ejemplos de enrutamiento OSPF

Tabla 4: Proyectos del marco INET.

Las fuentes de tráfico que se encuentran en este paquete generalmente las encontramos formando parte de módulos compuestos, como un host. Los módulos fuente se encuentran jerárquicamente implementados en las capas superiores del modelo OSI, como la capa de aplicación.

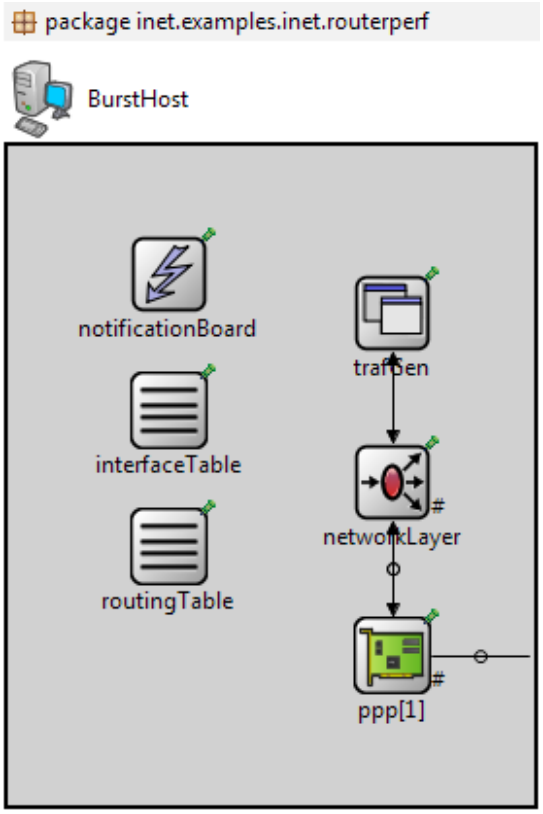


Fig.4. Esquema de un módulo compuesto en donde está contenido jerárquicamente el módulo IIPvXTrafficGenerator bajo el nombre trafGen. Fuente: Framework INET.

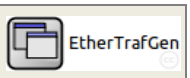
III.1.1. *Módulos generadores de tráfico Ethernet (capa física y capa de enlace de datos).*

III.1.1.1. *Módulo simple EtherTrafGen.*

Este módulo es un generador de tráfico simple para modelar el protocolo Ethernet y el IEEE802.11, y en general para cualquier modelo de capa 2 que acepte información de control en los paquetes. Debe ser conectado directamente al módulo *EtherEncap* o a un módulo *Ieee80211Nic*. [15]

Utilizado en el módulo compuesto *EtherHost2*.

Parámetros:

Nombre	Tipo	Valor por defecto	Descripción 
destAddress	*string*	" "	dirección MAC de destino, o ruta del nombre del módulo de la estación de destino; vacío significa off
etherType	*int*	0	etherType se establece en los paquetes salientes
startTime	*double*	this.sendInterval	momento del envío del primer paquete
stopTime	*double*	-1s	momento en que termina el envío, valores negativos significan para siempre
sendInterval	*double*		intervalo entre el envío de ráfagas
numPacketsPerBurst	*int*	1	número de paquetes para enviar por ráfaga (paquetes dentro de una ráfaga son enviados al mismo tiempo de simulación)
packetLength	*int*		longitud de los paquetes a enviar

Propiedades:

Nombre	Valor	Descripción
display	*i=block/app*	Indica la imagen que utiliza en el entorno gráfico Tkenv

Gates:

Nombre	Dirección	Descripción
in	*input*	punto de conexión hacia el interior del módulo
out	*output*	punto de conexión hacia el exterior del módulo

Estadísticas:

Título	Fuente	Registro	Unidad
packets sent	sentPk	count, sum(packetBytes), vector(packetBytes)	
end-to-end delay	messageAge(rcvdPk)	histogram, vector	s
packets received	rcvdPk	count, sum(packetBytes), vector(packetBytes)	

Simulaciones donde interviene:

Espacio de trabajo	Descripción
inet/tests/misc/etherfixes/	inet/tests/misc/etherfixes/README
inet/tests/networks/ethernet/	

III.1.1.2. *Módulo simple EtherAppCli.*

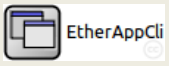
Otro módulo generador de tráfico simple, similar al anterior, para implementar los protocolos Ethernet y IEEE802.11, y en general para cualquier implementación de protocolo de capa 2 que

acepte información de control en los paquetes. Genera paquetes de solicitud *EtherAppReq* (clase de valor C++). [15]

Se conecta directamente al módulo *EtherLLC* o a un módulo *Ieee80211Nic*.

Utilizado en los módulos compuestos *EtherHost* y *ThroughputClient*.

Parámetros:

Nombre	Tipo	Valor por defecto	Descripción 
destAddress	*string*	" "	dirección MAC de destino, o ruta del nombre del módulo de la estación de destino; vacío significa off
startTime	*double*	this.sendInterval	momento del envío de la primera solicitud
stopTime	*double*	-1s	momento en que termina el envío, valores negativos significan para siempre
localSAP	*int*	0xf0	parámetro implícito de NED.
remoteSAP	*int*	0xf1	parámetro implícito de NED
sendInterval	*double*	uniform(0s,1s)	intervalo entre el envío de solicitudes
reqLength	*int*	100B	longitud de los paquetes de petición
respLength	*int*	1KiB (kibibyte)	longitud de los paquetes de respuesta
registerSAP	*bool*	false	si desea enviar el IEEE802CTRL_REGISTER_DSAP en el arranque

Propiedades:

Nombre	Valor	Descripción
display	*i=block/app*	indica la imagen que utiliza en el entorno gráfico Tkenv

Gates:

Nombre	Dirección	Descripción
in	*input*	punto de conexión hacia el interior del módulo
out	*output*	punto de conexión hacia el exterior del módulo

Estadísticas:

Título	Fuente	Registro	Unidad
packets sent	sentPk	count, sum(packetBytes), vector(packetBytes)	
end-to-end delay	messageAge(rcvdPk)	histogram, vector	s
packets received	rcvdPk	count, sum(packetBytes), vector(packetBytes)	

Simulaciones donde interviene:

Espacio de trabajo	Descripción
inet/examples/ethernet/lans/	inet/examples/ethernet/lans/LargeNet-doc.ned
inet/examples/ieee8021d/	inet/examples/ieee8021d/ README
inet/examples/voipstream/VoIPStreamLargeNet/	Consultar el archivo README
inet/examples/wireless/hosttohost/	inet/examples/wireless/hosttohost/ README
inet/examples/wireless/throughput/	inet/examples/wireless/throughput/ README

III.1.1.3. *Módulo simple EtherAppSrv.*

Este módulo constituye el lado del servidor del módulo *EtherAppCli*. Genera paquetes *EtherAppResp* con el número de bytes solicitados por el cliente en *EtherAppReq*. Debe ser

conectado directamente al módulo *EtherLLC* o un módulo de *Ieee80211Nic*. [15]

Utilizado en el módulo compuesto *EtherHost*.

Parámetros:

Nombre	Tipo	Valor por defecto	Descripción
registerSAP	*bool*	false	si desea enviar el IEEE802CTRL_REGISTER_DSAP en el arranque
localSAP	*int*	0xf1	Parámetro implícito de NED

Propiedades:

Nombre	Valor	Descripción
display	*i=block/app*	Indica la imagen que utiliza en el entorno gráfico Tkenv

Gates:

Nombre	Dirección	Descripción
in	*input*	punto de conexión hacia el interior del módulo
out	*output*	punto de conexión hacia el exterior del módulo

Estadísticas:

Título	Fuente	Registro	Unidad
packets sent	sentPk	count, sum(packetBytes), vector(packetBytes)	
end-to-end delay	messageAge(rcvdPk)	histogram, vector	s
packets received	rcvdPk	count, sum(packetBytes), vector(packetBytes)	

Simulaciones donde interviene:

Espacio de trabajo	Descripción
inet/examples/ethernet/lans/	inet/examples/ethernet/lans/LargeNet-doc.ned
inet/examples/ieee8021d/	inet/examples/ieee8021d/ README
inet/examples/voipstream/VoIPStreamLargeNet/	inet/examples/voipstream/VoIPStreamLargeNet/ README

III.1.2. *Módulos generadores de tráfico IP (capa de red).*

III.1.2.1. *Módulo simple IPvXTrafGen.*

Este módulo envía datagramas IPv4 o IPv6 a la dirección indicada en el período de tiempo *sendinterval*. *sendinterval* puede ser una constante o un valor aleatorio (por ejemplo, *exponential(1)*). [15]

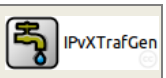
Si el parámetro *destAddresses* contiene más de una dirección, una de ellas es escogida de forma aleatoria para cada paquete. Una dirección se puede dar en la notación decimal con puntos o, para IPv6, en la notación habitual con dos puntos, o con el nombre del módulo. La clase (archivo cabecera, código C++) *IPvXAddressResolver* se utiliza para resolver la dirección. Por defecto la resolución de direcciones está desactivada, *destAddresses* = " ". [15]

Este módulo se puede emparejar con el módulo *IPvXTrafSink* u otro *IPvXTrafGen*.

Cuando se utiliza este módulo dentro un módulo *network* se requiere agregar los módulos *interfaceTable*, *routingTable* y *notificationBoard*.

Se puede encontrar como submódulo del módulo compuesto *BurstHost*.

Parámetros:

Nombre	Tipo	Valor por defecto	Descripción 
startTime	*double*	1s	momento de envío del primer paquete
stopTime	*double*	-1s	momento en que termina el envío, valores negativos significan para siempre
sendInterval	*double*	10ms	puede ser un valor aleatorio, por ejemplo, exponential(1)
numPackets	*int*	-1	máximo número de paquetes a generar, -1 significa sin cesar
protocol	*int*		lista de protocolos en IPProtocolId.msg . Valor para el campo protocolo de IPv4ControlInfo ó IPv6ControlInfo.
packetLength	*int*		longitud del paquete en bytes
destAddresses	*string*	" "	lista de direcciones de destino, separadas por espacios

Propiedades:

Nombre	Valor	Descripción
display	*i=block/source*	indica la imagen que utiliza en el entorno gráfico Tkenv

Gates:

Nombre	Dirección	Descripción
ipIn	*input*	punto de conexión IPv4 hacia el interior del módulo
ipv6In	*input*	punto de conexión IPv6 hacia el interior del módulo
ipOut	*output*	punto de conexión IPv4 hacia el exterior del módulo
ipv6Out	*output*	punto de conexión IPv6 hacia el exterior del módulo

Estadísticas:

Título	Fuente	Registro	Unidad
packets sent	sentPk	count, sum(packetBytes), vector(packetBytes)	
end-to-end delay	messageAge(rcvdPk)	histogram, vector	s
packets received	rcvdPk	count, sum(packetBytes), vector(packetBytes)	

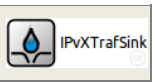
Simulaciones donde interviene:

Espacio de trabajo	Descripción
inet/examples/inet/routerperf/	inet/examples/inet/routerperf/README

III.1.2.2. *Módulo simple IPvXTrafSink.*

Este módulo consume los paquetes recibidos tanto desde el módulo *IPv4* como desde el *IPv6*. También guarda información estadística de paquetes recibidos y retardo extremo a extremo. Sólo acepta señales *rcvdPk*. Generalmente se encontrará funcionando dentro de una misma *network* como complemento de los módulos *IPvXTrafGen* y *IIPvXTrafficGenerator*. [15]

Parámetros:

Nombre	Tipo	Valor por defecto	Descripción 
protocol	*int*		lista de protocolos en IPProtocolId.msg . Valor del campo protocolo de IPv4ControlInfo ó IPv6ControlInfo.

Propiedades:

Nombre	Valor	Descripción
display	*i=block/sink*	indica la imagen que utiliza en el entorno gráfico Tkenv

Gates:

Nombre	Dirección	Descripción
ipIn	*input*	punto de conexión IPv4 hacia el interior del módulo
ipv6In	*input*	punto de conexión IPv6 hacia el interior del módulo
ipOut	*output*	punto de conexión IPv4 hacia el exterior del módulo
ipv6Out	*output*	punto de conexión IPv6 hacia el exterior del módulo

Estadísticas:

Título	Fuente	Registro	Unidad
end-to-end delay	messageAge(rcvdPk)	histogram, vector	s
packets received	rcvdPk	count, sum(packetBytes), vector(packetBytes)	

Simulaciones donde interviene:

Espacio de trabajo	Descripción
inet/examples/inet/routerperf/	inet/examples/routerperf/README

III.1.2.3. *Módulo simple PingApp.*

El módulo genera solicitudes de ping y calcula los parámetros pérdida de paquetes y *round-trip time* de las respuestas. El tiempo mínimo entre el envío de un segmento de datos y la llegada de su correspondiente reconocimiento ACK es exactamente un round-trip time. [16]

La clase *IPvXAddressResolver* (código C++), se utiliza para resolver la dirección. Para desactivar el envío, se especifica vacío el parámetro *destAddr*.

Cada solicitud de ping se envía con un número de secuencia, y se espera que las respuestas lleguen en el mismo orden. Cada vez que hay un salto en el número de secuencia de las respuestas *ping* recibidas, los pings que faltan se cuentan como perdidos. Luego, si todavía llega, pero más tarde (es decir, una respuesta con un número de secuencia más pequeño que el más grande recibido hasta ahora), será contado como llegada fuera de secuencia. Así que el número de *pings* realmente perdidos será "perdidos" menos "fuera de orden" (suponiendo que no hay duplicados o respuesta falsa). [15]

Parámetros:

Nombre	Tipo	Valor por defecto	Descripción
destAddr	*string*	" "	dirección de destino, escrita de la forma habitual
srcAddr	*string*	" "	dirección de la fuente (muy útil con multi-homing)
packetSize	*double*	56B	tamaño de paquete de ping de carga útil, en bytes
sendInterval	*double*	1s	tiempo de espera entre pings (puede ser aleatorio)
hopLimit	*double*	32	TTL (tiempo de vida) o límite de saltos para los paquetes IP
count	*double*	0	solicitudes de ping antes de detenerse la cuenta, 0

			significa continuamente
startTime	*double*	uniform(0s,this.sendInterval)	envía el primer ping en el instante startTime
stopTime	*double*	0	instante de fin del envío, 0 significa para siempre
printPing	*bool*	false	escribe el log de sucesos a la salida estándar stdout

Propiedades:

Nombre	Valor	Descripción 
display	*i=block/app*	indica la imagen que utiliza en el entorno gráfico Tkenv

Gates:

Nombre	Dirección	Descripción
pingIn	*input*	punto de conexión IPv4 hacia el interior del módulo
pingOut	*output*	punto de conexión IPv4 hacia el exterior del módulo
pingv6In	*input*	punto de conexión IPv6 hacia el interior del módulo
pingv6Out	*output*	punto de conexión IPv6 hacia el exterior del módulo

Estadísticas:

Título	Fuente	Registro	Unidad
ping rx seq		count, vector	
ping out-of-order arrivals		last, vector	
pings lost		last, vector	
ping tx seq		count, vector	
ping round-trip time		histogram, vector	s

Simulaciones donde interviene:

Espacio de trabajo	Descripción
inet/examples/	Ver los proyectos: adhoc, dhcp, inet, manetrouting, mobileipv6, wireless
MiXiM/examples-inet/	Ver los proyectos: adhoc, hybrid

III.1.2.4. *Módulo simple PingTestApp.*

Este módulo genera solicitudes de ping a varios hosts (o más bien, interfaces de red), y calcula la pérdida de paquetes y los tiempos de ida y vuelta (*round-trip time*) de las respuestas. Funciona exactamente igual que el módulo *PingApp* excepto que se puede especificar varias direcciones de destino como una lista separada por espacios de direcciones IP o en su defecto los nombres de los módulos. Especificando '*' en lugar de las direcciones, hace ping a todas las interfaces de red configuradas en la simulación. Esto es útil para comprobar si un host puede llegar a todos los otros hosts en la red (es decir, las tablas de enrutamiento se establecieron correctamente). [15]

Para especificar el número de solicitudes de ping enviadas a una única dirección de destino, utiliza el parámetro '*count*'. Después de que el número especificado de solicitudes de ping fue enviado a una dirección de destino, la aplicación se va a dormir por el tiempo '*sleepDuration*'. Una vez que el temporizador de *sleep* ha expirado, la aplicación cambia al siguiente destino y comienza a hacer ping de nuevo. La aplicación deja de hacer ping una vez que todas las direcciones de

destino se hayan probado o el tiempo de simulación alcanza '*stopTime*'. [15]

III.1.3. *Módulos generadores de tráfico con protocolos de la capa de transporte.*

Los módulos que entran en esta clasificación se encuentran suficientemente explicados en el capítulo 11 del manual de INET. [15]

Estos módulos se denominan: *UDPBasicApp, UDPBasicBurst, UDPEchoApp, UDPSink, UDPVideoStreamCli, UDPVideoStreamSvr*.

Se los puede encontrar en proyectos de simulaciones referenciadas a INET, donde se utilicen los módulos StandardHost, StandardHost6, MobileHost, MobileHost6, WirelessHost, WirelessHost6 y versiones extendidas de estos.

III.1.4. *Interfaces y condicionadores de tráfico.*

III.1.4.1. *Módulo interface IIPvXTrafficGenerator.*

Este es el módulo prototipo de interfaz que utilizan los módulos *IPvXTrafGen* y *IPvXTrafSink* que permite generar tráfico directamente sobre IP. Compatible con los módulos *IPv4* y *IPv6*. [15]

Se puede encontrar como un submódulo del módulo compuesto *BurstHost*.

Propiedades:

Nombre	**Valor**	**Descripción** IIPvXTrafficGenerator (interface)
display	*i=block/app*	indica la imagen que utiliza en el entorno gráfico Tkenv

Simulaciones donde interviene:

Espacio de trabajo	**Descripción**
inet/examples/inet/routerperf/	inet/examples/routerperf/README

III.1.4.2. *Módulo interface ITrafficConditioner.*

Este es el módulo interfaz de los condicionadores de tráfico.

Los condicionadores de tráfico implementan la vigilancia y dan forma a las acciones de un enrutador "*Diffserv"* (enrutador de servicios diferenciados). Estos se añaden a la ruta de entrada o de salida de los módulos interfaz de los enrutadores que están en el borde del dominio "*Diffserv"*.

Se puede encontrar formando parte de módulos compuestos como *EthernetInterface, IdealWirelessNic, LoopbackInterface* o *PPPInterface.*[15]

Propiedades:

Nombre	**Valor**	**Descripción** ITrafficConditioner (interface)
display	*i=block/classifier*	Indica la imagen que utiliza en el entorno gráfico Tkenv

Simulaciones donde interviene:

Espacio de trabajo	**Descripción**
inet/examples/diffserv/onedomain/	inet/examples/diffserv/onedomain/README
inet/examples/diffserv/simple_/	inet/examples/diffserv/simple_/README

III.1.4.3. *Módulo compuesto TrafficConditioner.*

Módulo compuesto que ejecuta las funciones del enrutamiento *DiffServ*.

TrafficConditioner utiliza los siguientes módulos: *IdealChannel, DSCPMarker, SingleRateThreeColorMeter, TokeBucketMeter, Sink, BehaviourAggregateClassifier, Join* y *MultiFieldClassifier*. [15]

El condicionador de tráfico realiza las siguientes acciones: Clasificar los paquetes entrantes, monitorizar el tráfico en cada clase de paquete, marcar o soltar paquetes en función del resultado de la monitorización (comparación con un umbral), moldear el tráfico para ajustarlo al perfil de tráfico deseado. Todo ello puede implementarse como un módulo compuesto, utilizando los submódulos clasificador, monitorizador, y marcador.

Se puede consultar RFC 3290 para una descripción detallada de la arquitectura de los condicionadores de tráfico. [17]

Propiedades:

Nombre	**Valor**	**Descripción** 
display	*i=block/classifier*	Indica la imagen que utiliza en el entorno gráfico Tkenv

Gates:

Nombre	**Dirección**	**Descripción**
in	*input*	punto de conexión hacia el interior del módulo
out	*output*	punto de conexión hacia el exterior del módulo

Parámetros sin asignar de los submódulos:

Nombre	**Tipo**	**Valor por defecto**	**Descripción**
efMeter.cir	string		tasa de información comprometida, como bitrate absoluto (ej. "100kbps"), o relative a la tasa de datos del enlace (ej. "20%")
efMeter.cbs	int		tamaño de ráfaga comprometido
efMeter.colorAwareMode	bool	false	habilita el modo tomar en cuenta los colores
defaultMeter.cir	string		tasa de información comprometida, como bitrate absoluto (ej. "100kbps"), o relative a la tasa de datos del enlace (ej. "20%")
defaultMeter.cbs	int		tamaño de ráfaga comprometido
defaultMeter.ebs	int		exceso del tamaño de ráfaga
defaultMeter.colorAwareMode	bool	false	habilita el modo tomar en cuenta los colores

Simulaciones donde interviene:

Espacio de trabajo	**Descripción**
inet/examples/diffserv/onedomain/	inet/examples/diffserv/onedomain/README
inet/examples/diffserv/simple_/	inet/examples/diffserv/simple_/README

III.2. *Paquete marco MiXiM para OMNEST/OMNET++.*

MiXiM es un marco de modelado OMNeT creado para redes móviles y fijas inalámbricas: Redes

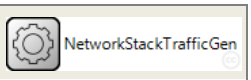
de sensores inalámbricos, redes de área corporal, las redes ad-hoc, redes vehiculares, etc. Ofrece modelos detallados de la propagación de ondas de radio, la estimación de la interferencia, el consumo de energía y protocolos MAC inalámbricos como Zigbee que se basa en IEEE 802.15.4 [5].

III.2.1. *Módulos generadores de tráfico multiprotocolo para enlaces de red inalámbicos.*

III.2.1.1. *Módulo simple NetworkStackTrafficGen.*

Módulo simple que permite configurar parámetros para la generación de tráfico en pequeñas ráfagas para el protocolo IEEE802.15.4. [18]

Parámetros:

Nombre	**Tipo**	**Valor por defecto**	**Descripción** 
notAffectedByHostState	*bool*	false	
coreDebug	*bool*	false	activa la depuración para el marco básico
stats	*bool*	false	activa la recopilación de estadísticas
headerLength	*int*		Tamaño de la cabecera del paquete de red (bits)
packetTime	*double*		intervalo de tiempo del grupo de paquetes (s)
packetsPerPacketTime	*double*		cantidad de paquetes en el intervalo de tiempo del paquete
debug	*bool*	false	activa la depuración
burstSize	*int*	1	tamaño de la ráfaga
packetLength	*int*		tamaño del paquete generado (bits)
destination	*int*	-1	sin destino especificado

Propiedades:

Nombre	**Valor**	**Descripción**
classs	*NetworkStackTrafficGen*	como el valor “tipo de red” del módulo *PhyMacHost*

Gates:

Nombre	**Dirección**	**Descripción**
upperLayerIn	*input*	conexión de entrada desde la capa superior
upperLayerOut	*output*	conexión de salida hacia la capa superior
upperControlIn	*input*	entrada de control de la capa superior
upperControlOut	*output*	salida de control hacia la capa superior
lowerLayerIn	*input*	conexión de entrada desde la capa superior
lowerLayerOut	*output*	conexión de salida hacia la capa superior
lowerControlIn	*input*	entrada de control de la capa inferior
lowerControlOut	*output*	salida de control hacia la capa inferior

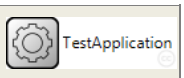
Simulaciones donde interviene:

Espacio de trabajo	**Descripción**
MiXiM/examples/ieee802154Narrow/	MiXiM/examples/ieee802154Narrow/README

III.2.1.2. *Módulo simple TestApplication.*

Este módulo es utilizado en el módulo compuesto *host802154A* como clase de valor del parámetro *applicationType*, para generar tráfico de una aplicación. [18]

Parámetros:

Nombre	**Tipo**	**Valor por defecto**	**Descripción** 
debug	*bool*	false	activa la depuración del código
stats	*bool*	false	activa la recolección de estadísticas
trace	*bool*	false	activa la traza
trafficParam	*double*		media del tiempo entre paquetes(velocidad de arribo tipo Poisson)
nodeAddr	*double*		dirección del nodo
dstAddr	*double*		dirección del nodo destino de los paquetes
flood	*bool*		enviar paquetes en forma contínua
payloadSize	*double*		número de bytes por paquete
nbPackets	*double*		número de paquetes que se generarán
headerLength	*int*		tamaño de la cabecera del mensaje de la aplicación (bits)

Gates:

Nombre	**Dirección**	**Descripción**
lowerLayerIn	*input*	recibe datos de la pila de comunicaciones
lowerLayerOut	*output*	envía datos a la pila de comunicaciones
lowerControlIn	*input*	recibe mensajes de control de la pila de comunicaciones
lowerControlOut	*output*	envía mensajes de control hacia la pila de comunicaciones

Simulaciones donde interviene:

Espacio de trabajo	**Descripción**
MiXiM/examples/ieee802154a/	MiXiM/examples/ieee802154a/README

III.2.1.3. *Módulo simple TrafficGen.*

Módulo diseñado para la capa de aplicación que genera una cierta cantidad casi constante de tráfico. [18]

Puede ser empleado de la siguiente manera en el archivo ".ini" de la simulación:

**.node[*].applicationType = "TrafficGen"

**.node[*].appl.debug = false

**.node[*].appl.headerLength = 512bit

**.node[*].appl.burstSize = 1

**.node[*].appl.packetTime = (600/15000) * 1s

**.node[*].appl.packetsPerPacketTime = 1/5

Donde (600/15000) debe ser el tamaño máximo del paquete dividido por la tasa de bits mínima y **.node[*] representa cualesquiera de los módulos de la carpeta "MiXiM/src/modules/node/".

En MiXiM no se encontró ninguna simulación donde se haya utilizado.

Parámetros:

Nombre	Tipo	Valor por defecto	Descripción
packetTime	*double*		intervalo de tiempo que necesita un paquete para ser transmitido
packetsPerPacketTime	*double*		paquetes que se deben generar por unidad de tiempo de paquete (regula el tráfico)
debug	*bool*	false	activar depuración
burstSize	*int*	1	
headerLength	*int*		

Gates:

Nombre	Dirección	Descripción
lowerLayerIn	*input*	conexión de entrada desde la capa inferior
lowerLayerOut	*output*	conexión de salida hacia la capa inferior
lowerControlIn	*input*	recibe mensajes de control desde la capa inferior
lowerControlOut	*output*	envía mensajes de control a la capa inferior

IV. Proyectos de simulación en OMNeT++.

Antes de discernir la "naturaleza" del numeroso conjunto de módulos de OMNet++, para intentar implementarlos en cualquier aplicación que utilizase los protocolos TCP, UDP, SCTP, o cualquier otro de la pila de protocolos de capas inferiores, no existía una guía sobre cuáles de esos módulos se necesitarían, ni qué parámetros debían utilizarse, ni como configurarlos. El método que se empleó para lograrlo, en la mayor parte del trabajo de la simulación fue el de "prueba y error", tratando de interpretar los mensajes que presentaba el software en cada intento de "ejecución" de la simulación. De esta forma se fue investigando con apoyo en los manuales y probando cambios en los archivos ".ned" y ".ini" hasta conseguir una respuesta que luego se pudo analizar. En virtud de haber entendido las características comunes entre los módulos compuestos, aquellos que en la interface gráfica se pueden concebir como "dispositivos" de la red: router, switch, host, etc., finalmente fue posible visualizar similitudes y clasificar los submódulos "generadores de tráfico" como "fuentes".

Una vez conocida la clasificación del tráfico moderno, según la ITU-T, en el capítulo ÍI y que se han caracterizado los módulos del simulador concernientes al tráfico en el capítulo III, se presenta a continuación una simulación en la que se combinarán diferentes tipos de tráfico sobre una misma red, tratando de emular a pequeñísima escala lo que sucedería en una red real. Es de suponer que en un ambiente de laboratorio a gran escala, la información estadística que se pueda obtener de la simulación, debería servir para cotejar la autosimilaridad mediante el cálculo del coeficiente de Hurst y además estudiar la distribución de Poisson para el tráfico de VoIP e Internet dial-up. Adicionalmente se comentan algunos de los errores y cómo se salvó la situación.

IV.1. *Simulación, Proyecto "demodemo".*

El proyecto "*demodemo*" es el prototipo definitivo luego de varios precursores que se construyeron durante el aprendizaje de la herramienta. Para su implementación se utilizó la versión 4.2.2 del simulador en un ordenador portátil *acer* con procesador Intel Core i3, 4GB de RAM y S.O. Windows 7 – 64bits. Se encuentra referenciado a los proyectos marco INET y MiXiM. Consta de un módulo *network* que gráficamente se visualiza como una superficie cuadrilátera en cuyo interior se ubican el resto de módulos. Como se ve en la figura 5, cuatro módulos compuestos tipo "*host*" son los originadores de los flujos de tráfico: VoIP, vídeo, transferencia de archivo y datos de sensor. El módulo "*hostEth_C"* es el receptor de todo el tráfico de carga útil.

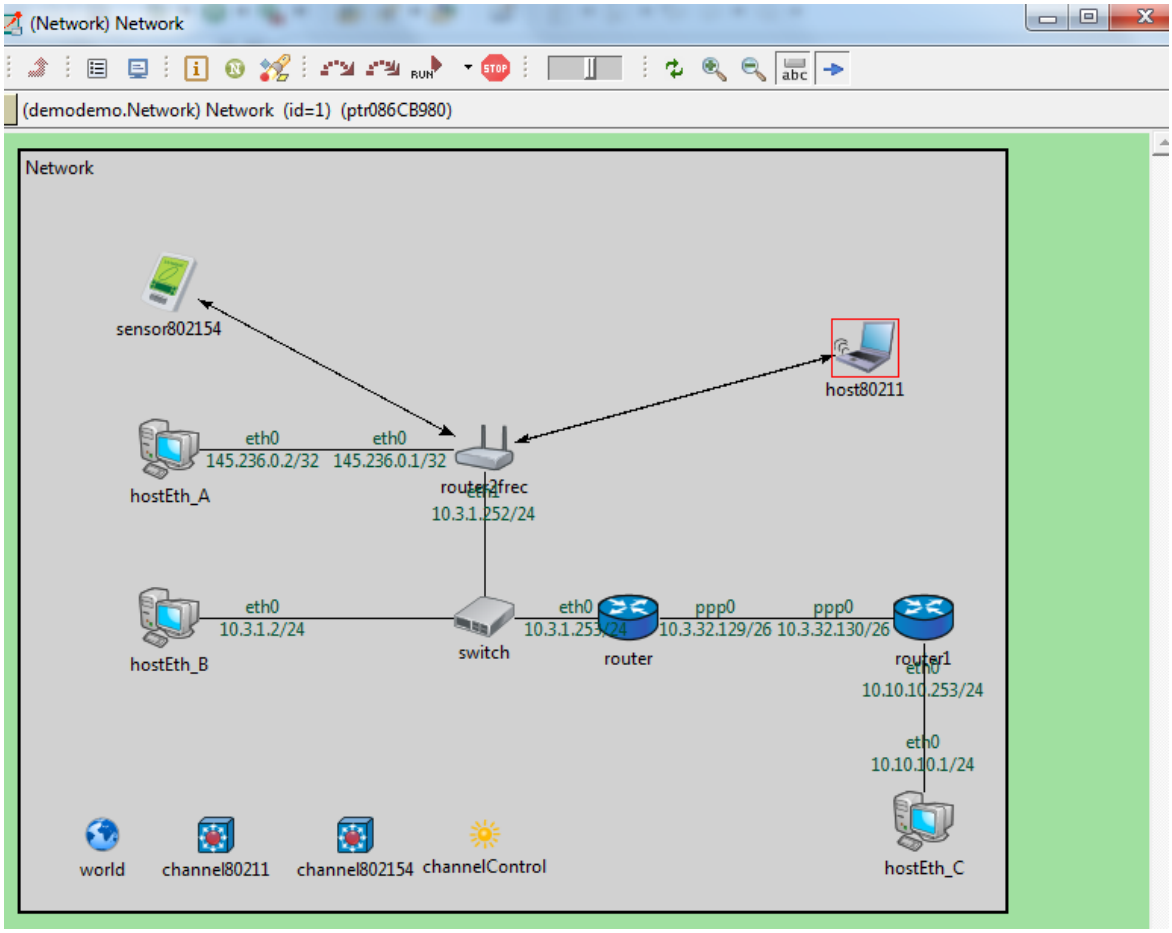


Fig. 5. Visualización del archivo *demodemo.ned* donde se aprecia la topología de red.

El funcionamiento de los módulos "existentes" en OMNeT++ no sólo está determinado por el código NED sino que está enlazado al código de los archivos fuente y cabecera, escritos en lenguaje C++. Estos archivos están contenidos en la sub-carpeta "Includes" del proyecto y en los proyectos marco INET y MiXiM, a los cuales está referenciado el proyecto de simulación. Debido a ello, se suele hablar de clases y aplicaciones, que son funciones u "objetos" programados en esos archivos de código complementarios al lenguaje NED.

La topología de red, número de puertos de los módulos y las conexiones entre ellos, se programa en el archivo "demodemo.ned" que se puede visualizar gráficamente o en líneas de código NED.

La configuración de algunos parámetros se puede hacer en el miso archivo ".ned", pero no se recomienda porque es más complicado hacer cambios; esta tarea se la hace sobre el archivo ".ini". En el archivo "demodemo.ini" se configuran, entre otras cosas, los escenarios y el tiempo de la simulación. En este caso el tiempo de simulación para todos los escenarios es 4s.

Durante la simulación, los mensajes clase *cPacket*, transportan la información como paquetes de una red IP. Las rutas IP que siguen estos paquetes se configuran en los archivos ".mrt" y ".route". Estos archivos se ubican bajo el mismo árbol de directorios del proyecto.

El módulo *world* (*BaseWorldUtility*) define las distancias en tres dimensiones (x, y, z), como las dimensiones del módulo *network*, para simular los efectos de las distancias sobre los parámetros de capa física de los tres *hosts* inalámbricos, por ejemplo: Interferencia, potencia de la señal, sensibilidad, retardo de propagación, *bit error rate*, etc.

Se ha exagerado el parámetro *speed* del módulo *IMobility* en el módulo compuesto "*host80211"* con una velocidad de 500 m/s para hacer visible su desplazamiento en la interface gráfica Tkenv. Este módulo representa un dispositivo móvil que se desplaza con movimiento circular emitiendo una señal de vídeo, bajo requerimiento de hostEth_C y sin él, dependiendo del escenario.

La configuración de los parámetros relacionados con las interfaces inalámbricas se realiza a través de los módulos *ConnectionManager* y *ChannelControl* en conjunto con los archivos: *config80211.xml, config802154.xml, Nic802154_TI_CC2420_Decider.xml*. Estos parámetros se dividen principalmente en parámetros de la capa MAC y parámetros de la capa PHY (física). Por ejemplo, las frecuencias para IEEE802.11 son 2.4GHz y para IEEE802.15.4 son 868MHz. [5]

Los módulos inalámbricos tienen configurados parámetros de uso de batería para su autonomía energética durante el tiempo de la simulación, pero esa característica no se pone a prueba en este trabajo.

IV.1.1. *Escenarios de simulación.*

El proyecto tiene diseñados varios escenarios de simulación, los cuales están agrupados y descritos brevemente en el código del archivo "demodemo.ini". Se acompañan comentarios en el código. Ver apéndices I y II.

Los escenarios permiten la "ejecución" independiente de una sección del archivo "demodemo.ini". Para acceder a la selección del escenario, se puede hacer lo siguiente: "*clic* derecho" sobre el archivo ".ini" en la columna "*Project Explorer*", se escoge la opción *Run as*, *Run Configurations…*, luego sobre "*OMNeT++ Simulation*", clic derecho, *New*, y grabar un nombre como se ve en la figura 6. Escoger el escenario en la caja *Config name* y hacer *clic* sobre *Run* en la parte inferior de la ventana.

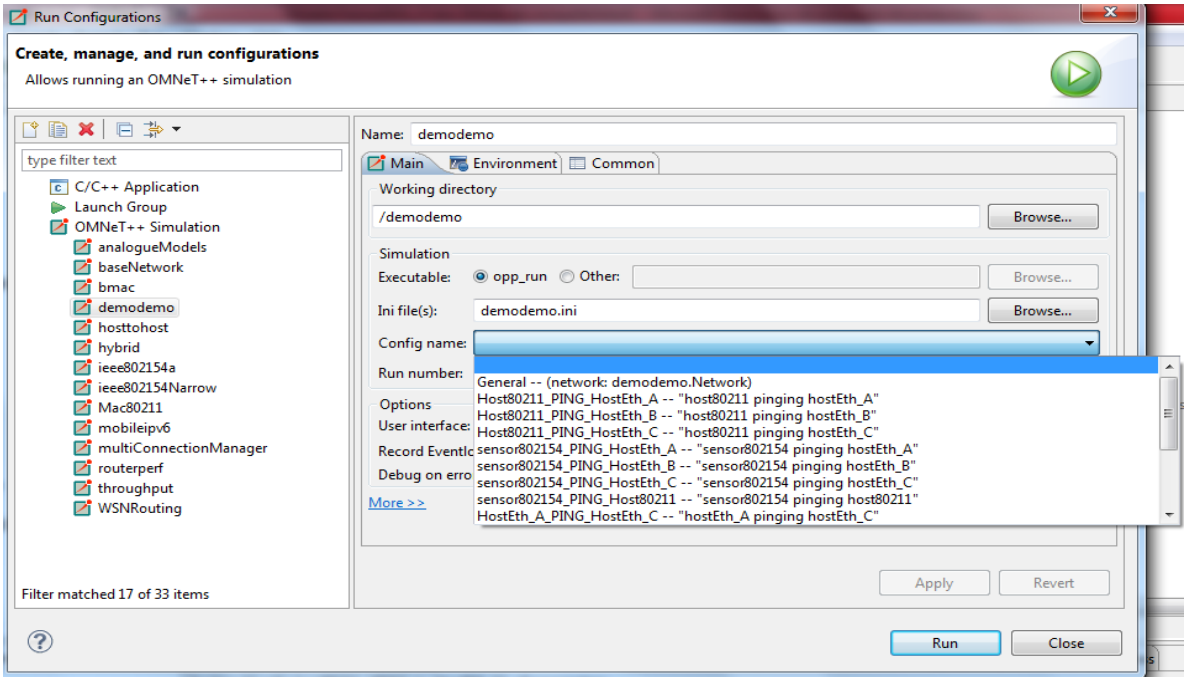


Fig.6. Configuración para la "ejecución" de varios escenarios de simulación.

Con respecto a la configuración de los escenarios, destacamos en las siguientes líneas dos aspectos con respecto a los flujos de tráfico simulados.

Los flujos de tráfico UDP y TCP tienen varios modos de transferencia, que se configuran como un valor de la información de control de sus respectivas aplicaciones. Esto se hace a través de clases que se asocian a los módulos tipo *host*. Los parámetros de las aplicaciones UDP y TCP se observan en las líneas del archivo ".ini" precedidos de: *udp*, *tcp*, *udpApp* y *tcpApp*.

El flujo de tráfico de un sensor real es muy bajo, del orden de decenas de MB por año [5]. Así que sus efectos como tráfico agregado se tratan de simular en una situación especial como ocurre al dispararse una alarma [19], mediante la generación aleatoria de ráfagas de datos con protocolo de transporte UDP equiparables en tamaño a las de los otros dispositivos en la red.

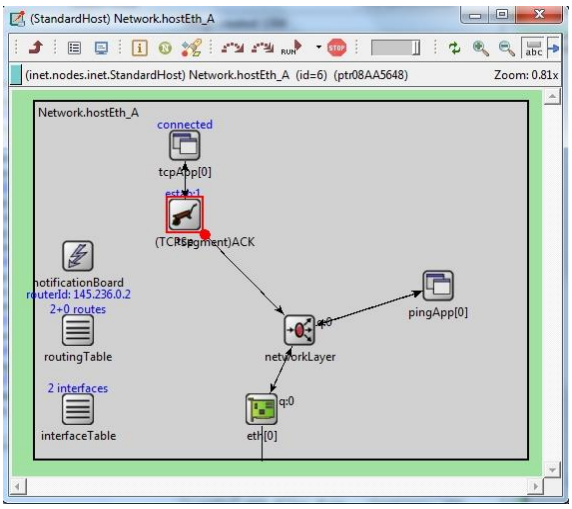


Fig.7. Visualización del módulo compuesto *standardHost* etiquetado "hostEth_A" durante la simulación. Se aprecia el módulo fuente de tráfico *ITCPApp* etiquetado "tcpApp[0]".

Las estadísticas de la simulación se recogen en dos tipos de archivos, para los datos de forma escalar y vectorial. Estos archivos se encuentran bajo el directorio de la carpeta "results" en el árbol de carpetas del proyecto "demodemo". A partir de los datos estadísticos se crea el archivo de análisis de resultados "demodemo.anf".

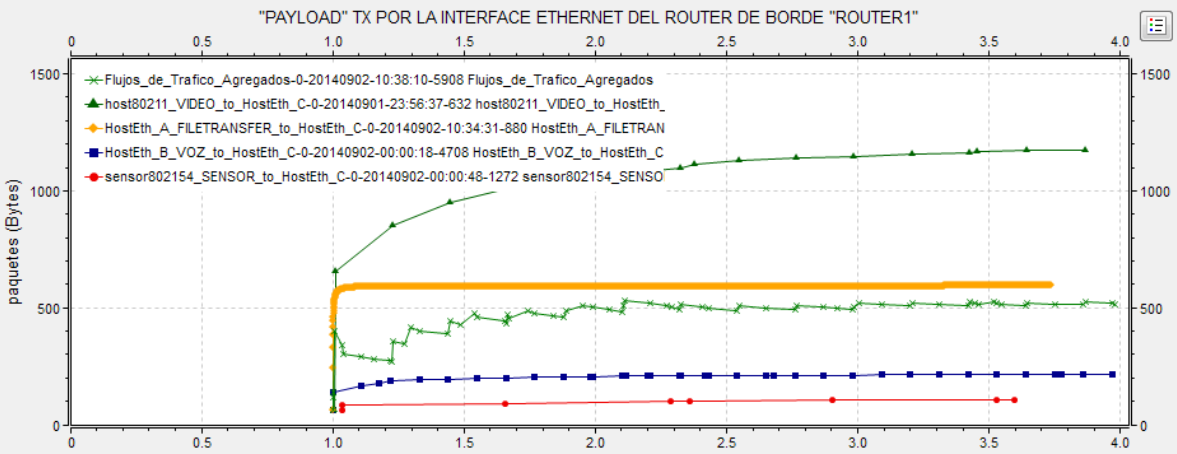


Fig.8. Gráfico linealizado de la evolucións de diferentes flujos de tráfico simulados en forma independiente.

La capacidad del canal (bits/segundo) Ethernet en la conexión "*router1*" hacia "*hostEth_C*" se definió en 100Mbps. En la siguiente figura se aprecia la tasa de bit promedio que calcula el submódulo *EtherMACBase* del módulo "*router1*", durante el intervalo de tiempo que dura la simulación completa (ver código fuente *EtherMACBase.cc*). Cada par de columnas rojo-azul corresponde a una "ejecución" diferente de la simulación con diferente tipo de carga de tráfico. El flujo de tráfico agregado se obtiene simulando todas las fuentes en la misma "ejecución". El *throughput* está constituido por los datos útiles y la señalización.

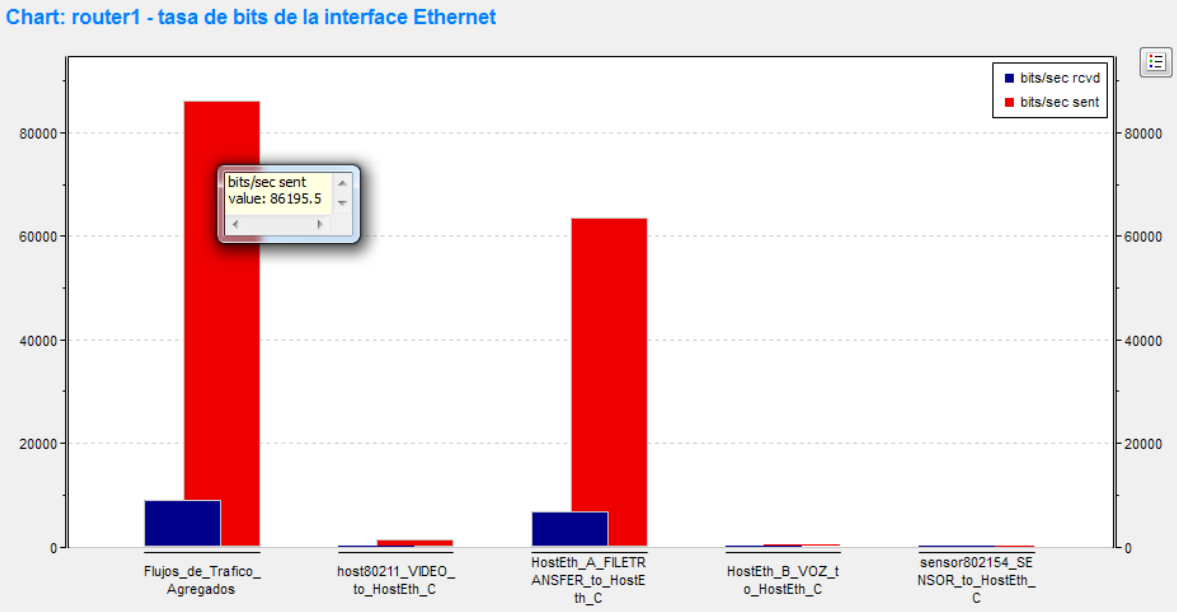


Fig.9. Gráfico del "throughput" que atraviesa la interface Ethernet del "router1".

Existen muchas combinaciones y operaciones matemáticas que se pueden realizar para crear los gráficos y guardarlos en Datasets como los mostrados en las dos figuras anteriores. Los datos de los

archivos escalares y vectoriales pueden ser exportados para su manejo con otro software como Wireshark y Matlab.

Otra parte importante de OMNeT++ es el visor del LOG de eventos que se genera como parte de la simulación. En la figura 10 se observa la interface que permite repasar con detenimiento el paso de mensajes entre los módulos durante la última simulación. Esta herramienta es muy útil puesto que facilita el análisis, las correcciones y mejoras así como encontrar errores. El LOG se guarda en el archivo ".elog" dentro del directorio "results" en la carpeta del proyecto.

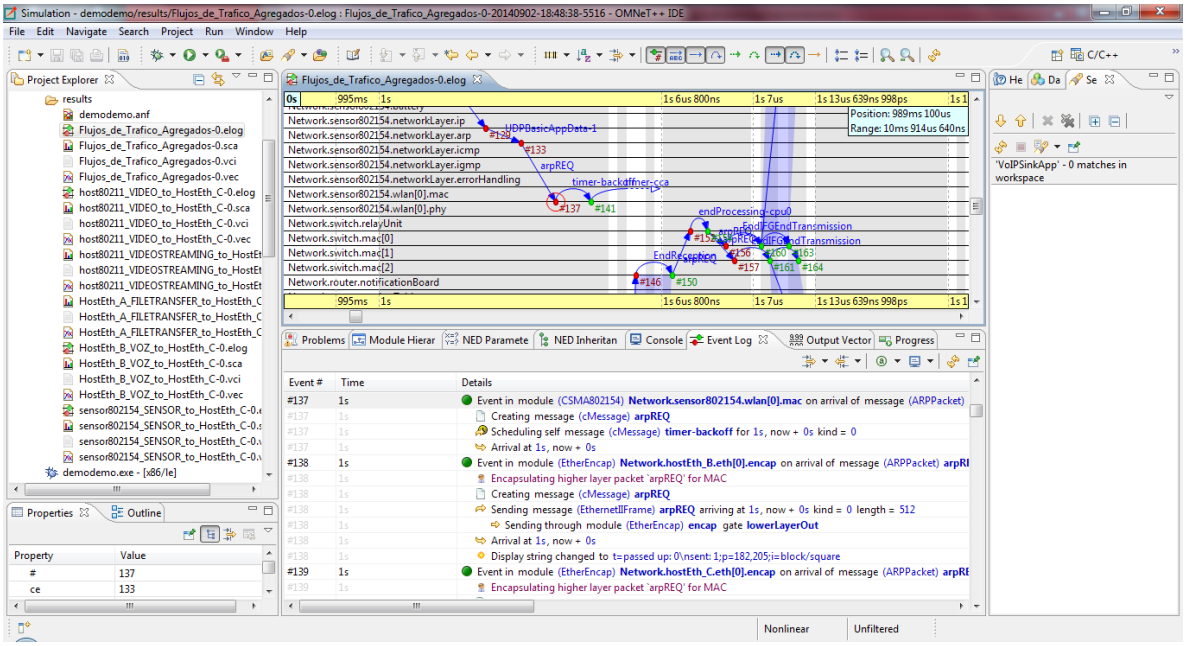


Fig.10. Log de eventos del intercambio de mensajes *cMessage* entre los módulos. Se actualiza automáticamente luego de cada "ejecución".

IV.2. *Errores en la ejecución de una simulación.*

En esta sección se describen algunos de los errores más comunes del uso de OMNeT++. Esta sección tiene su importancia en el intento de cumplir uno de los objetivos de esta investigación, que es la de ser una guía.

Pese a que en los archivos INSTALL.txt e InstallGuide.pdf incluidos en las carpetas del instalador, se detalla muy bien el proceso de instalación, no se ha hecho mención a un problema básico como es el relacionado con el software antivirus cargado comúnmente en ordenadores con el sistema Windows. El antivirus suele configurarse para sensar constantemente la creación de archivos y los que se crean en el proceso de instalación de OMNeT++ son del tipo ejecutable y librerías ".dll" que son "interceptados" y eliminados a veces sin darnos cuenta. La solución a esto es deshabilitar esa función del antivirus durante el proceso de instalación y más adelante también cuando el software ejecuta el proceso "*build*" de la compilación.

La copia de carpetas o archivos desde el espacio gestionado por Project Explorer en la interface IDE conlleva la pérdida de referencias en el código NED. Se recomienda no hacerlo a menos que

se tenga cierta experiencia. En su lugar es mejor transcribir el código que se desea a un nuevo proyecto "*empty*". Los archivos ".xml", ".route", ".ini" si se pueden copiar limpiamente.

A presar de ser posible el cambio de los nombres de archivos y carpetas, resulta casi tan negativo como Cambio nombre de archivos. Por tanto, tampoco se recomienda.

Hablando concretamente de la interface IDE, un error muy común es el siguiente, el cual indica que los módulos que se ha tratado de unir no son compatibles por alguna razón o se ha obviado la conexión. En este caso se debe recurrir a verificar el código. Una forma de corregirlo es cambiar la sección *connections* por *connections allowunconected* en el archivo ".ned" del módulo compuesto *network*.

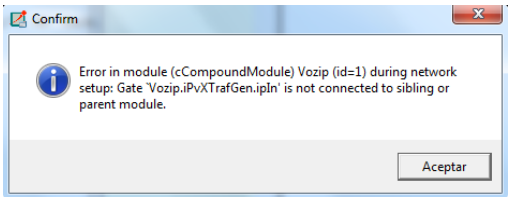


Fig.11. Ventana con mensaje de error en la sección *connections* del código del archivo ".ned".

Los "*Warnings*" e "*Info*" en la ventana *Problems* del entorno IDE no significan que la simulación tendrá falencias. Muchas veces indican variables de otros proyectos que no tienen asignado su valor. Cuando hay mensajes *error* la simulación no se ejecuta.

Al estar referenciados los proyectos, a veces un fallo de compilación de la simulación puede afectar a los proyectos marco INET y MiXiM con el borrado automático de archivos fuente y cabecera (".c" y ".h"), lo cual se corrige cerrando IDE y volviendo a copiar los originales desde las carpetas respaldadas de la instalación. El archivo que no se encuentra porque ha sido borrado se muestra entre los mensajes que aparecen en la ventana *Console*.

El modo *debug* sólo funciona si se inicia omnetpp.exe desde la consola MINGW.

Cuando uno de los módulos requiere la presencia de otro, al iniciarse la simulación sin él, el siguiente mensaje recuerda que debe ser incluido. Esto se corrige visualizando en el modo gráfico el archivo ".ned" para luego seleccionar el módulo faltante de la paleta de íconos (*Palette*).

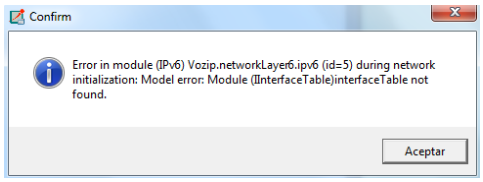


Fig.12. Ventana con mensaje de error de módulo faltante.

En el ejercicio de simulación "demodemo" se notó que conforme el alchivo ".ini" se vuelve más extenso mientras introducimos la configuración de parámetros de los módulos, la interface IDE actualiza constantemente los cambios en el código y pueden ocurrir "congelamientos". Nótese que se trabajó con un ordenador con 4Gb de RAM y un procesador doble núcleo Intel corei3 1.4GHz x 2 y sistema Windows 7 de 64 bit. Para solucionarlo fue necesario terminar con el proceso desde el

"Administrador de tareas" de "Windows" y volver a ejecutar el comando *omnetpp* desde la consola MINGW para volver a cargar IDE.

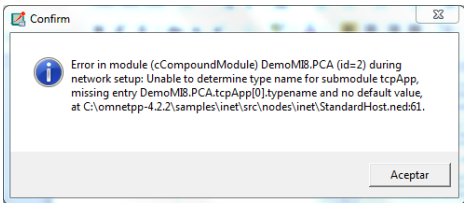


Fig.13. Ventana con mensaje de falta de un valor de la aplicación TCP.

En el establecimiento del transporte TCP, un *host* "A" envía la señal SYN al *host* "B" pero este, en lugar de contestar con *ACK*, le envía *RST+ACK*, en consecuencia "A" contesta con *RST* rechazando la conexión. Se soluciona reasignando los valores para los parámetros sendInterval o packetLenght.

Para la configuración de los parámetros de los módulos del proyecto MiXiM en el proyecto prueba de simulación #8, primero se copiaron a la carpeta del proyecto los archivos ".xml" que están bajo la carpeta "MiXiM/examples-inet/hybrid", una vez asignados los valores y ejecutar la simulación ocurría constantemente el "congelamiento" de los procesos lanzados por opp_run.exe. La solución a esto fue volver a configurar desde el inicio un nuevo proyecto, el número #9.

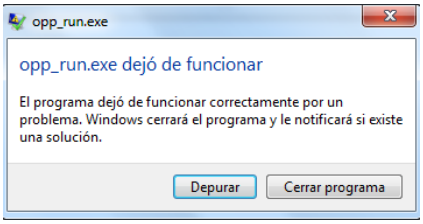


Fig.14. Mensaje Windows por el "congelamiento" del motor de ejecución de la simulación (opp_run.exe).

Por otra parte, hablando del entorno de programación: Los mensajes del tipo "*pop up*" (al combinar teclas Ctrl+space) que aparecen como ayuda en la introducción del código no expresan información suficiente para su rápido entendimiento. Además suele ser necesario hacer varios intentos de la combinación de teclas para que se muestren todas las opciones. En algunos casos se puede "congelar" toda la interface perdiéndose los cambios realizados al archivo. Se recomienda grabar constantemente los cambios en el código.

V. Resultados y observaciones.

Durante la búsqueda en la Web (páginas web oficiales, blogs, YouTube, etc.), no se pudo hallar suficiente información enmarcada en la didáctica con respecto al uso de OMNet++. De entre los tutoriales y blogs relacionados al simulador que fueron consultados, no fue posible encontrar una clasificación o caracterización de módulos por el tipo de uso como la que se ha desarrollado en esta investigación.

El material bibliográfico que está referenciado en este trabajo, permite enmarcar la necesidad de la caracterización de las fuentes de un sistema de comunicaciones para *Smart City* y sirve como

pauta para la caracterización en OMNet++.

Con respecto al manejo del software simulador OMNet++, casi exclusivamente se contó con documentación proveniente del manual, la guía del usuario y el estudio de los ejemplos auto-contenidos, para el desarrollo de la caracterización de fuentes de tráfico y la construcción del modelo de red que se presenta. Se concluye que el manual de OMNet++ es una pieza clave para el entendimiento del funcionamiento del simulador, es el comienzo ideal en su aprendizaje. El manual del proyecto INET se considera poco formal en su redacción; en general, se percibe que son pocos los contribuyentes del código abierto que aportan mayor detalle a sus propios esfuerzos.

De la información publicada en la Web, en tesinas realizadas por investigadores de varias universidades que comparten libremente sus experiencias en el uso de OMNeT++. Se destaca que por las continuas modificaciones en las versiones de OMNet++, se necesita dedicar un tiempo de investigación y minucioso trabajo para adaptar los códigos antiguos de las simulaciones, lo que revela un inconveniente para replicar los experimentos que a futuro se espera que lo corrijan.

Como parte de este trabajo, adicionalmente se subieron vídeos en Youtube con distintos escenarios de simulación como una contribución didáctica. Se accede libremente a ellos en el canal: https://www.youtube.com/user/mcoello73/videos?view_as=public

VI. Conclusiones.

Se construyó un modelo de red con módulos de los proyectos INET y MiXiM para simular flujos de tráfico: *file transfer,* vídeo, voz y de sensores. Luego se configuraron pruebas de ping y los escenarios para simular los flujos de tráfico mencionados, de forma independiente y de forma agregada. Todo este proceso conllevó muchas horas de estudio, pruebas y solución de problemas como bugs del código, errores de Windows, errores en los procesos de compilación, etc. Es así que se concluye que OMNeT++ no es una herramienta para el ejercicio de la ingeniería, sino de investigación, que puede desarrollarse mucho más.

En general es muy importante conocer los fundamentos de la programación orientada a objetos y particularmente los lenguajes de programación C++ y Java, si se quiere explotar el uso de este simulador en la investigación del entorno Smart City.

Si se trabaja bajo el sistema operativo Windows hay que contar con un ordenador moderno de recursos altos. La versión 4.2.2 de OMNeT++ que se utilizó sufrió “congelamientos” en determinados momentos durante su uso. Preferiblemente debería trabajarse bajo un sistema operativo que disponga mejor de los recursos de procesamiento y memoria.

Se brinda al lector investigador con poca experiencia en la utilización de OMNeT++ una guía que le permita abordar de la mejor manera posible la utilización de esta herramienta de software libre para simular fuentes de tráfico en cualquiera de los tipos de escenarios de las redes modernas.

AGRADECIMIENTOS

A los creadores de los proyectos INET y MiXiM.

REFERENCIAS

[1] 3GPP TS 22.368 V1.2.2, Especificacion Técnica, 2010.

[2] Willinger, W., Taqqu, M., Sherman, R., and Wilson, D. (1997), *Self-similarity through high-variability*. IEEE/ACM Transaction on Networking, vol. 5, No. 1, 1997, pp. 71-86.

[3] Rabinovich Daniel, *Desarrollo de un modelo generador de tráfico autosimilar para la evaluación de los algoritmos de asignación de ancho de banda en WiMAX*, tesis, Universidad Nacional de La Plata, 2010.

[4] Alarcon-Aquino, V.; Guerrero-Ojeda, L.G.; Rodriguez-Asomoza, J. y Rosas-Romero, R.. *Análisis de Tráfico Auto-similar en Redes de Comunicaciones Usando Onditas (Wavelets)*. *Inf. tecnol.* [online]. 2005, vol.16, n.2 [citado 2014-07-13], pp. 61-66. http://dx.doi.org/10.4067/S0718-07642005000200010.

[5] Orrevad Andrés, *M2M Traffic Characteristics*, tesina, Suecia, 2009.

[6] Iversen, *Teletraffic Engineering And Network Planning, 2010*, http://www.fotonik.dtu.dk.

[7] Aymen Hafsaoui, Navid Nikaein, Christian Bonnet, *Analysis and Experimentation with a Realistic Traffic Generation Tool for Emerging Application Scenarios*, EURECON, Francia, 2013.

[8] A. Koucheryavy, *Seminar on ITU-T hot topics for Standardization*, Mar del Plata, Argentina, 2009.

[9] Quintero O., Ruiz J., "*Estimación del exponente de Hurst y la dimensión fractal de una superficie topográfica a través de la extracción de perfiles*", Colombia, 2011.

[10] Recomendación UIT-T Q.3925, *Traffic flow types for testing quality of service parameters on model networks*, 2012.

[11] Sheluin, Smolskiy,Osin, *Self-Similar Processes in Telecommunications*, 2007.

[12] T. Steinbach, H. Dieumo Kenfack, F. Korf, and T. C. Schmidt, *An Extension of the OMNeT++ INET Framework for Simulating Real-time Ethernet with High Accuracy*. In Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques, SIMUTools '11, pages 375-382, ICST, Brussels, Belgium, Belgium, 2011. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[13] Varga Andras, *OMNet++ User Manual, 2011*, incluido en la carpeta "doc" de la instalación.

[14] Documentación de INET, archivo local de vínculos index.html.

[15] *INET Framework for OMNeT++ Manual*, 2012, incluido en la carpeta "doc" de la instalación.

[16] http://www.cs.unc.edu/~fhernand/diss-html/node25.html, *Parámetros de la capa de red.*

[17] http://tools.ietf.org/html/rfc3290#ref-DSARCH, *An Informal Management Model for Diffserv Routers*, RFC 3290.

[18] Proyecto MiXiM, Comentarios adjuntos al código de los archivos ".ned" y ".ini".

[19] 3GPP TR 21.905, *Vocabulary for 3GPP Specifications*, Reporte Técnico, 2013.

[20] http://www.webopedia.com/TERM/T/traffic_meter.html. Definición de traffic meter en Webopedia.

[21] W.E. Leland, M.S. Taqqu, W. Willinger, D.V. Wilson, *On the Self-Similar Nature of Ethernet Traffic*, SIGCOMM, 1993.

[22] López Guillermo, *Simulacion de Redes de Computadores con OMNet++ 4*, proyecto de fin de carrera, Pamplona, 2010.

GLOSARIO

“.c” y en general el resto de la nomenclatura: “.ned” , “.ini” , “.h” , “.elog” ,etc., se ha utilizado para denominar los archivos por su tipo. Por ejemplo, a veces se hace referencia a “demodemo.ini” como el archivo “.ini” dependiendo del contexto empleado.

LDP es un protocolo de aplicación de distribución de etiquetas MPLS.

Monitorizador (Traffic Meter) es una característica del enrutador (puede ser enrutador Wi-Fi) que monitoreará el consumo de datos (carga y descarga) de todos los dispositivos de la red y le notificará cuando se acerque a un umbral predefinido. Un traffic meter es útil para el ISP que está haciendo cumplir una asignación mensual máxima de transferencia de datos (normalmente se trata de 250 GB). [20]

NED es el lenguaje de OMNeT++. Acrónimo de Network Description.

RSVP-TE (RFC 3209) es un protocolo de aplicación que permite la reserva de recursos en una red IP. Indica a los otros nodos la naturaleza del flujo de paquetes que quiere recibir, como son ancho de banda, jitter, burst máximo, etc.

SAP es un protocolo de anuncio de sesiones de conferencia multicast en Internet. Una llamada de conferencia es anunciada por la multidifusión periódica de un paquete de anuncio UDP a una dirección y puerto multicast. SAP no es adecuado para llamadas telefónicas IP de uno-a-uno. El Grupo de Trabajo MMUSIC IETF ha desarrollado SAP, que se define en el RFC 2974 (Protocolo de Anuncio de Sesión, octubre de 2000).

SCTP, *Stream Control Transmission Protocol*, es un protocolo de transporte alternativo a TCP y UDP.

APÉNDICE I

Código NED del archivo "demodemo.ned".

Este código contiene la estructura básica de la topología de red que se va a simular.

```
//

package demodemo;
import inet.nodes.ethernet.EtherSwitch;
import inet.nodes.inet.Router;
import inet.nodes.inet.StandardHost;
import inet.world.radio.ChannelControl;
import ned.DatarateChannel;
import org.mixim.base.connectionManager.ConnectionManager;
import org.mixim.base.modules.BaseWorldUtility;
import org.mixim.examples.inet.hybrid.RouterWithBattery;
import org.mixim.examples.inet.hybrid.WirelessHostWithBattery;


//
// Esta red se utiliza para simular por separado diferentes fuentes de tráfico.
// En uno de los escenarios se involucran todas las fuentes dentro de la misma simulación.
// Utiliza módulos del proyecto INET y del proyecto MiXiM. Es muy versátil su configuración.
// Se pueden modificar parámetros relacionados con los protocolos: TCP, UDP, IP, IEEE802.11, IEEE802.15.4,
// Ethernet, PPP, etc.
// Se pueden modificar parámetros relacionados con: Mobilidad de hosts, baterías, etc.
//
network Network
{
    @display("bgb=650,500");
    submodules:
        world: BaseWorldUtility {
            @display("p=54,450;is=s");
        }
        channel80211: ConnectionManager {
            @display("p=129,450;is=s");
        }
        channel802154: ConnectionManager {
            @display("p=221,450;is=s");
        }
        channelControl: ChannelControl {
            @display("p=306,450;is=s");
        }
        hostEth_A: StandardHost {
            @display("p=98,196");
        }
        router2frec: RouterWithBattery {
            @display("p=306,195;i=device/accesspoint");
            gates:
                ethg[2];
        }
        host80211: WirelessHostWithBattery {
            @display("p=508,57");
        }
        sensor802154: WirelessHostWithBattery {
            @display("p=98,86;i=device/palm2");
        }
        switch: EtherSwitch {
            @display("p=306,306");
            gates:
                ethg[3];
        }
        router: Router {
            @display("p=401,307");
        }
        router1: Router {
            @display("p=596,307");
        }
        hostEth_B: StandardHost {
            @display("p=98,307");
        }
        hostEth_C: StandardHost {
            @display("p=596,441");
        }
    connections:
        hostEth_A.ethg++ <--> DatarateChannel {  delay = 0.1us; datarate = 100Mbps; } <--> router2frec.ethg[0];
        router2frec.ethg[1] <--> DatarateChannel {  delay = 0.1us; datarate = 100Mbps; } <--> switch.ethg[0];
        switch.ethg[1] <--> DatarateChannel {  delay = 0.1us; datarate = 100Mbps; } <--> router.ethg++;
        router.pppg++ <--> DatarateChannel {  delay = 1us; datarate = 10Mbps; } <--> router1.pppg++;
        switch.ethg[2] <--> DatarateChannel {  delay = 0.1us; datarate = 100Mbps; } <--> hostEth_B.ethg++;
        router1.ethg++ <--> DatarateChannel {  delay = 0.1us; datarate = 100Mbps; } <--> hostEth_C.ethg++;

}
```

APÉNDICE II

Código del archivo "demodemo.ini".

En estas líneas se configuran los módulos sin alterar sus parámetros por defecto.

```
[General]
network = demodemo.Network

#**************************** configuración del área de trabajo de la red

record-eventlog = false

sim-time-limit = 4s
**.world.playgroundSizeX = 650m
**.world.playgroundSizeY = 500m
**.world.playgroundSizeZ = 0m
**.coreDebug = true

#****************************configuración de mobilidad para el host80211
#******************* Esta configuración le da mobilidad gráfica al módulo host80211 para simular un dispositivo
#********** (ordenador, tablet, etc.) portátil que reproduce un vídeo mientras tiene un desplazamiento circular

**.host80211.mobilityType = "CircleMobility"
**.host80211.mobility.cx = 550m
**.host80211.mobility.cy = 90m
**.host80211.mobility.r = 40m
**.host80211.mobility.constraintAreaMaxX = 650m/1
**.host80211.mobility.constraintAreaMaxY = 500m/1
**.host80211.mobility.constraintAreaMaxZ = 0m/0
**.host80211.mobility.constraintAreaMinX = 0m/1
**.host80211.mobility.constraintAreaMinY = 0m/1
**.host80211.mobility.constraintAreaMinZ = 0m/0
**.host80211.mobility.updateInterval = 10ms
**.host80211.mobility.speed = 500mps

#**************************** archivos de rutas para todos los host

*.hostEth_A.routingFile = "hostEth_A.route"
*.host80211.routingFile = "host80211.route"
*.sensor802154.routingFile = "sensor802154.route"
*.router2frec.routingFile = "router2frec.route"
*.hostEth_B.routingFile = "hostEth_B.mrt"
*.hostEth_C.routingFile = "hostEth_C.mrt"
*.router.routingFile = "router.route"
*.router1.routingFile = "router1.route"

#**************************** configuración de las interfaces inalámbricas MiXiM

**.router2frec.numRadios = 2
**.router2frec.wlan[0].typename = "org.mixim.modules.nic.Nic80211"
**.router2frec.wlan[1].typename = "org.mixim.modules.nic.Nic802154_TI_CC2420"
**.sensor802154.wlan[*].typename = "org.mixim.modules.nic.Nic802154_TI_CC2420"
**.host80211.wlan[*].typename = "org.mixim.modules.nic.Nic80211"

#**************************** parámetros para el gestor de conexión channel80211

**.host80211.wlan[0].connectionManagerName = "channel80211"
**.router2frec.wlan[0].connectionManagerName = "channel80211"

**.channel80211.sendDirect = false
**.channel80211.pMax = 110.11mW  # [mW]
**.channel80211.sat = -110dBm  # [dBm]
**.channel80211.alpha = 3
**.channel80211.carrierFrequency = 2.4e+9Hz  # [Hz]

#**************************** parámetros de la capa MAC 802.11
**.host80211.wlan[0].mac.headerLength = 272 bit
**.host80211.wlan[0].mac.queueLength = 14


#********* valores como máximo 1% de tasa de error de paquete, si no se modela desvanecimiento (fading)

**.host80211.wlan[0].mac.rtsCtsThreshold = 400
**.router2frec.wlan[0].mac.rtsCtsThreshold = 400
**.router2frec.wlan[0].mac.headerLength = 272 bit #longitud de la cabecera MAC 802.11
**.router2frec.wlan[0].mac.queueLength = 14 #longitud de la cola MAC 802.11

#************************* parámetros para la capa física (PHY) del #protocolo 802.11

**.host80211.wlan[0].phy.usePropagationDelay = false
**.host80211.wlan[0].phy.thermalNoise = -110dBm  # [dBm]
**.host80211.wlan[0].phy.analogueModels = xmldoc("config80211.xml")
**.host80211.wlan[0].phy.decider = xmldoc("config80211.xml")
**.host80211.wlan[0].phy.maxTXPower = 110.11mW
**.host80211.wlan[0].phy.useThermalNoise = true

**.router2frec.wlan[0].phy.usePropagationDelay = false
**.router2frec.wlan[0].phy.thermalNoise = -110dBm  # [dBm]
**.router2frec.wlan[0].phy.analogueModels = xmldoc("config80211.xml")
**.router2frec.wlan[0].phy.decider = xmldoc("config80211.xml")
**.router2frec.wlan[0].phy.maxTXPower = 110.11mW
**.router2frec.wlan[0].phy.useThermalNoise = true

#****************************  parámetros para el gestor de conexión 802.15.4
```

```
**.sensor802154.wlan[0].connectionManagerName = "channel802154"
**.router2frec.wlan[1].connectionManagerName = "channel802154"

**.channel802154.sendDirect = false
**.channel802154.pMax = 1.1mW
**.channel802154.sat = -100dBm
**.channel802154.alpha = 2.5
**.channel802154.carrierFrequency = 868E+6Hz ##original 2.4E+9Hz

#**************************** parámetros para la capa MAC del protocolo 802.15.4

**.sensor802154.wlan[0].mac.txPower = 1mW  # [mW]
**.sensor802154.wlan[0].mac.notAffectedByHostState = true
**.sensor802154.wlan[0].mac.macMinBE = 1
**.sensor802154.wlan[0].mac.macMaxBE = 6
**.sensor802154.wlan[0].mac.macMaxCSMABackoffs = 20
**.sensor802154.wlan[0].mac.macAckWaitDuration = 0.000864s
**.sensor802154.wlan[0].mac.aUnitBackoffPeriod = 0.02s

**.router2frec.wlan[1].mac.txPower = 1mW  # [mW]
**.router2frec.wlan[1].mac.notAffectedByHostState = true

#*********************** parámetros para la capa física del protocolo 802.15.4

**.sensor802154.wlan[0].phy.usePropagationDelay = false
**.sensor802154.wlan[0].phy.analogueModels = xmldoc("config802154.xml")
**.sensor802154.wlan[0].phy.sensitivity = -100dBm
**.sensor802154.wlan[0].phy.maxTXPower = 1.1mW
**.sensor802154.wlan[0].phy.useThermalNoise = true

**.router2frec.wlan[1].phy.usePropagationDelay = false
**.router2frec.wlan[1].phy.analogueModels = xmldoc("config802154.xml")
**.router2frec.wlan[1].phy.sensitivity = -100dBm
**.router2frec.wlan[1].phy.maxTXPower = 1.1mW
**.router2frec.wlan[1].phy.useThermalNoise = true

#********** parámetros relacionados con la energía consumida por el sensor y el router 802.15.4

**.battery.nominal = 99999mAh
**.battery.capacity = 99999mAh
**.battery.voltage = 3.3V
**.battery.resolution = 1.1s
**.battery.publishDelta = 1
**.battery.publishTime = 10s
**.battery.numDevices = 1
**.batteryStats.detail = false


#******************* configuración de parámetros generales para las simulaciones con flujos de tráfico
#************************************************************** FILETRANSFER, VOZ, VÍDEO Y SENSOR.
# destino de paquetes UDP
**.hostEth_C.numUdpApps = 4
**.hostEth_C.udpApp[*].typename = "UDPSink"
**.hostEth_C.udpApp[0].localPort = 1000
**.hostEth_C.udpApp[1].localPort = 1001
**.hostEth_C.udpApp[2].localPort = 1002


#******************* configuración de parámetros de ejecución de simulaciones con mensajes ICMP PING
#******** El propósito es coprobar el correcto intercambio y enrutamiento de los mensajes y paquetes

**.numPingApps = 1
*.sensor802154.pingApp[0].sendInterval = 500ms
**.pingApp[0].sendInterval = 100ms

[Config Host80211_PING_HostEth_A]
description = "host80211 pinging hostEth_A"
*.host80211.pingApp[0].destAddr = "hostEth_A"

[Config Host80211_PING_HostEth_B]
description = "host80211 pinging hostEth_B"
*.host80211.pingApp[0].destAddr = "hostEth_B"

[Config Host80211_PING_HostEth_C]
description = "host80211 pinging hostEth_C"
*.host80211.pingApp[0].destAddr = "hostEth_C"

[Config sensor802154_PING_HostEth_A]
description = "sensor802154 pinging hostEth_A"
*.sensor802154.pingApp[0].destAddr = "hostEth_A"

[Config sensor802154_PING_HostEth_B]
description = "sensor802154 pinging hostEth_B"
*.sensor802154.pingApp[0].destAddr = "hostEth_B"

[Config sensor802154_PING_HostEth_C]
description = "sensor802154 pinging hostEth_C"
*.sensor802154.pingApp[0].destAddr = "hostEth_C"

[Config sensor802154_PING_Host80211]
description = "sensor802154 pinging host80211"
*.sensor802154.pingApp[0].destAddr = "host80211"

[Config HostEth_A_PING_HostEth_C]
description = "hostEth_A pinging hostEth_C"
*.hostEth_A.pingApp[0].destAddr = "hostEth_C"

[Config HostEth_B_PING_HostEth_C]
description = "hostEth_B pinging hostEth_C"
*.hostEth_B.pingApp[0].destAddr = "hostEth_C"

#******************* configuración de parámetros de ejecución de simulaciones con mensajes UDP y TCP
#******************************** El propósito es simular flujos de tráfico de vídeo, datos y sensor.
```

```
# El tráfico de un sensor real es muy bajo, del orden de decenas de MB por año. Así que sus efectos
# como tráfico agregado se trata de simular con la generación aleatoria de ráfagas de gran cantidad
# datos con protocolo de transporte UDP.

[Config HostEth_A_FILETRANSFER_to_HostEth_C]
description = "hostEth_A TCP fileTransfer hostEth_C"

#activar captura de estados y algoritmo TCP
**.hostEth_A.tcpType = "TCP"
**.hostEth_C.tcpType = "TCP"

#hostEth_C hace requerimiento de tráfico file-transfer TCP a hostEth_A
**.hostEth_C.numTcpApps = 1   #número de aplicaciones que se transportarán con TCP
**.hostEth_C.tcp.tcpAlgorithmClass = "TCPReno"#uno de los varios algoritmos que se pueden elegir
**.hostEth_C.tcpApp[*].typename  = "TCPSessionApp" # este tipo habilita los parámetros para las siguientes líneas
**.hostEth_C.tcpApp[*].active = true # "true" la aplicación actuará como cliente, "false" como servidor
**.hostEth_C.tcpApp[*].connectAddress = "145.236.0.2"
**.hostEth_C.tcpApp[*].connectPort = 21
**.hostEth_C.tcpApp[*].localAddress = "10.10.10.1"
**.hostEth_C.tcpApp[*].localPort = 21
**.hostEth_C.tcpApp[*].dataTransferMode = "bytestream" #valores posibles “bytecount”, “object”, “bytestream”
**.hostEth_C.tcpApp[*].sendBytes = 0MiB #por ser cliente, sólo recibe
**.hostEth_C.tcpApp[*].tOpen = 1s #apertura del puerto

#envío de tráfico file-transfer TCP desde hostEth_A
**.hostEth_A.numTcpApps = 1
**.hostEth_A.tcpApp[*].typename  = "TCPSessionApp" #sólo con este tipo se habilitan los parámetros para las siguientes
líneas
**.hostEth_A.tcpApp[*].active = false
**.hostEth_A.tcpApp[*].connectAddress = "10.10.10.1"
**.hostEth_A.tcpApp[*].localPort = 21
**.hostEth_A.tcpApp[*].dataTransferMode = "bytestream"
**.hostEth_A.tcpApp[*].sendBytes = 0.003MiB      #0.003*1024*1024 bytes
**.hostEth_A.tcpApp[*].connectPort = 21
**.hostEth_A.tcpApp[*].localAddress = "145.236.0.2"
**.hostEth_A.tcpApp[*].tOpen = 1s # apertura del puerto
**.hostEth_A.tcpApp[*].tSend = 1s
**.hostEth_A.tcpApp[*].tClose = 2s

[Config HostEth_B_VOZ_to_HostEth_C]
description = "hostEth_B UDP VoIP hostEth_C"

# Tráfico de voz
**.hostEth_B.numUdpApps = 1
**.hostEth_B.udpApp[*].typename = "UDPBasicBurst"
**.hostEth_B.udpApp[*].destAddresses = "10.10.10.1"
**.hostEth_B.udpApp[*].chooseDestAddrMode = "once"
**.hostEth_B.udpApp[*].startTime = 1s
**.hostEth_B.udpApp[*].stopTime = 4s
**.hostEth_B.udpApp[*].destPort = 1000
**.hostEth_B.udpApp[*].messageLength = 172B
**.hostEth_B.udpApp[*].sendInterval = exponential(200ms)
**.hostEth_B.udpApp[*].burstDuration = 0.11s
**.hostEth_B.udpApp[*].sleepDuration = 0s

[Config host80211_VIDEO_to_HostEth_C]
description = "host80211 UDP video hostEth_C"

# Tráfico de vídeo
**.host80211.numUdpApps = 1
**.host80211.udpApp[*].typename = "UDPBasicBurst"
**.host80211.udpApp[*].destAddresses = "10.10.10.1"
**.host80211.udpApp[*].chooseDestAddrMode = "once"
**.host80211.udpApp[*].startTime = 1s
**.host80211.udpApp[*].stopTime = 4s
**.host80211.udpApp[*].destPort = 1001
**.host80211.udpApp[*].messageLength = 1200B
**.host80211.udpApp[*].sendInterval = exponential(500ms)
**.host80211.udpApp[*].burstDuration = 0.12s
**.host80211.udpApp[*].sleepDuration = 0.1s

[Config sensor802154_SENSOR_to_HostEth_C]
description = "sensor802154 ráfagas UDP hostEth_C"

# Tráfico sensor IEEE 802.15.4. Ráfagas en el intervalo startTime-stopTime
**.sensor802154.numUdpApps = 1
**.sensor802154.udpApp[*].typename = "UDPBasicBurst"
**.sensor802154.udpApp[*].destAddresses = "10.10.10.1"
**.sensor802154.udpApp[*].chooseDestAddrMode = "once"
**.sensor802154.udpApp[*].startTime = 1s
**.sensor802154.udpApp[*].stopTime = 4s
**.sensor802154.udpApp[*].destPort = 1002
**.sensor802154.udpApp[*].messageLength = uniform(55B,80B) #(trigger ó frame overhead) + payload
**.sensor802154.udpApp[*].sendInterval = exponential(100ms)
**.sensor802154.udpApp[*].burstDuration = 0.13s
**.sensor802154.udpApp[*].sleepDuration = 0.5s

[Config Flujos_de_Trafico_Agregados]
description = "Flujos de Tráfico Agregados"

#****************** configuración de parámetros de ejecución de simulaciones con mensajes UDP y TCP
#****************************** El propósito es simular flujos de tráfico de vídeo, datos y sensor.
# El tráfico de un sensor real es muy bajo, del orden de decenas de MB por año. Así que sus efectos
# como tráfico agregado se trata de simular con la generación aleatoria de ráfagas de gran cantidad
# datos con protocolo de transporte UDP.

#activar captura de estados y algoritmo TCP
**.hostEth_A.tcpType = "TCP"
**.hostEth_C.tcpType = "TCP"

#hostEth_C hace requerimiento de tráfico file-transfer TCP a hostEth_A
**.hostEth_C.numTcpApps = 1   #número de aplicaciones que se transportarán con TCP
**.hostEth_C.tcp.tcpAlgorithmClass = "TCPReno"#uno de los varios algoritmos que se pueden elegir
```

```
**.hostEth_C.tcpApp[*].typename  = "TCPSessionApp" #sólo con este tipo se habilitan los parámetros para las siguientes
líneas
**.hostEth_C.tcpApp[*].active = true # "true" la aplicación actuará como cliente, "false" como servidor
**.hostEth_C.tcpApp[*].connectAddress = "145.236.0.2"
**.hostEth_C.tcpApp[*].connectPort = 21
**.hostEth_C.tcpApp[*].localAddress = "10.10.10.1"
**.hostEth_C.tcpApp[*].localPort = 21
**.hostEth_C.tcpApp[*].dataTransferMode = "bytestream" #valores posibles “bytecount”, “object”, “bytestream”
**.hostEth_C.tcpApp[*].sendBytes = 0MiB #por ser cliente, sólo recibe
**.hostEth_C.tcpApp[*].tOpen = 1s #apertura del puerto

#envío de tráfico file-transfer TCP desde hostEth_A
**.hostEth_A.numTcpApps = 1
**.hostEth_A.tcpApp[*].typename  = "TCPSessionApp" #sólo con este tipo se habilitan los parámetros para las siguientes
líneas
**.hostEth_A.tcpApp[*].active = false
**.hostEth_A.tcpApp[*].connectAddress = "10.10.10.1"
**.hostEth_A.tcpApp[*].localPort = 80
**.hostEth_A.tcpApp[*].dataTransferMode = "bytestream"
**.hostEth_A.tcpApp[*].sendBytes = 0.003MiB      #0.003*1024*1024 bytes
**.hostEth_A.tcpApp[*].connectPort = 80
**.hostEth_A.tcpApp[*].localAddress = "145.236.0.2"
**.hostEth_A.tcpApp[*].tOpen = 1s # apertura del puerto
**.hostEth_A.tcpApp[*].tSend = 1s
**.hostEth_A.tcpApp[*].tClose = 2s


# Tráfico de voz
**.hostEth_B.numUdpApps = 1
**.hostEth_B.udpApp[*].typename = "UDPBasicBurst"
**.hostEth_B.udpApp[*].destAddresses = "10.10.10.1"
**.hostEth_B.udpApp[*].chooseDestAddrMode = "once"
**.hostEth_B.udpApp[*].startTime = 1s
**.hostEth_B.udpApp[*].stopTime = 4s
**.hostEth_B.udpApp[*].destPort = 1000
**.hostEth_B.udpApp[*].messageLength = 172B
**.hostEth_B.udpApp[*].sendInterval = exponential(200ms)
**.hostEth_B.udpApp[*].burstDuration = 0.11s
**.hostEth_B.udpApp[*].sleepDuration = 0s


# Tráfico de vídeo
**.host80211.numUdpApps = 1
**.host80211.udpApp[*].typename = "UDPBasicBurst"
**.host80211.udpApp[*].destAddresses = "10.10.10.1"
**.host80211.udpApp[*].chooseDestAddrMode = "once"
**.host80211.udpApp[*].startTime = 1s
**.host80211.udpApp[*].stopTime = 4s
**.host80211.udpApp[*].destPort = 1001
**.host80211.udpApp[*].messageLength = 1200B
**.host80211.udpApp[*].sendInterval = exponential(500ms)
**.host80211.udpApp[*].burstDuration = 0.12s
**.host80211.udpApp[*].sleepDuration = 0.1s


# Tráfico sensor IEEE 802.15.4. Ráfagas en el intervalo startTime-stopTime
**.sensor802154.numUdpApps = 1
**.sensor802154.udpApp[*].typename = "UDPBasicBurst"
**.sensor802154.udpApp[*].destAddresses = "10.10.10.1"
**.sensor802154.udpApp[*].chooseDestAddrMode = "once"
**.sensor802154.udpApp[*].startTime = 1s
**.sensor802154.udpApp[*].stopTime = 4s
**.sensor802154.udpApp[*].destPort = 1002
**.sensor802154.udpApp[*].messageLength = uniform(55B,80B) #(trigger ó frame overhead) + payload
**.sensor802154.udpApp[*].sendInterval = exponential(100ms)
**.sensor802154.udpApp[*].burstDuration = 0.13s
**.sensor802154.udpApp[*].sleepDuration = 0.5s

[Config host80211_VIDEOSTREAMING_to_HostEth_C]
description = "sensor802111 VIDEO STREAM UDP hostEth_C"

#implementación de la aplicación UDP Video Stream con un nodo servidor y un cliente. Es posible más de un
#cliente.

**.hostEth_C.numUdpApps = 1
**.hostEth_C.udpApp[*].typename = "UDPVideoStreamCli"
**.hostEth_C.udpApp[*].localPort = 5005
**.hostEth_C.udpApp[*].serverAddress = "145.236.0.3"
**.hostEth_C.udpApp[*].serverPort = 5004
**.hostEth_C.udpApp[*].startTime = 1s

**.host80211.numUdpApps = 1
**.host80211.udpApp[*].typename = "UDPVideoStreamSvr"
**.host80211.udpApp[*].videoSize = 1.2MiB
**.host80211.udpApp[*].packetLen = 72B #cabecera+payload = 12B + 60B (como un paquete RTP)
**.host80211.udpApp[*].localPort = 5004
**.host80211.udpApp[*].sendInterval = uniform(1e-6s,1.01e-6s)
```