Universidad Politécnica de Cartagena

Escuela Técnica Superior de Ingeniería de Telecomunicación

PRÁCTICAS DE REDES DE ORDENADORES

Propuesta del Trabajo de Prácticas 2008

Simulación de una red troncal de fibra óptica (EuroSim)

Profesores:

Juan A. Veiga Gontán
Esteban Egea López
Juan José Alcaraz Espín
Joan García Haro

Índice.

1 Consideraciones generales.

1.1 Objetivos

En este trabajo de prácticas los alumnos realizarán por parejas un simulador de una ***red troncal de fibra óptica***. Los alumnos deben desarrollar y depurar el simulador y extraer y analizar una serie de resultados. Tanto los algoritmos como los resultados a presentar se detallan en esta memoria.

1.2 Introducción

1.2.1 Redes troncales (Backbone networks)

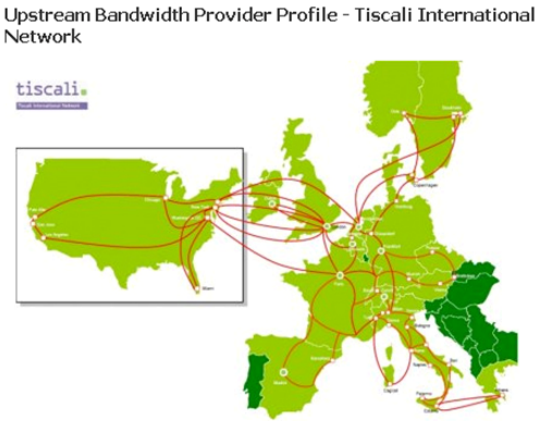

Figura 1. Ejemplo de red troncal

La palabra Backbone se refiere a las principales conexiones troncales de Internet. Está compuesta de un gran número de routers comerciales, gubernamentales, universitarios y otros de gran capacidad interconectados que llevan los datos entre países, continentes y océanos del mundo.

La "columna vertebral" de Internet consiste en muchas redes diferentes. Normalmente, el término se usa para describir grandes redes que se interconectan entre ellas y pueden tener ISPs (Internet Service Provider) individuales como clientes. Por ejemplo, un ISP local puede proporcionar servicio para una única ciudad, y conectar a un proveedor regional que tiene varios ISPs locales como clientes. Este proveedor regional conecta a una de las redes del backbone, que proporciona conexiones a escala nacional o mundial.

Estos proveedores backbone normalmente proporcionan instalaciones de conexión en muchas ciudades para sus clientes, y ellos mismos conectan con otros proveedores backbone en IXPs (Internet Exchange Point) como el CATNIX de Barcelona, el ESPANIX de Madrid o el GALNIX de Santiago de Compostela. El más grande de estos IXP en términos de tasa de transferencia y rutas accesibles es el LINX (London Internet Exchange) en la zona portuaria de Londres.

Las redes de backbones son implementadas normalmente por entes comerciales, educacionales, o gubernamentales, como redes militares. Algunas grandes compañías que proporcionan conectividad backbone incluyen UUnet (ahora una división de Verizon), British Telecom, AT&T, Sprint Nextel, France Télécom, BSNL, Teleglobe, Qwest, y SAVVIS.

Hoy en día los enlaces que unen los nodos en las redes troncales son de fibra óptica, con lo cual disponemos de un enorme ancho de banda de transmisión entre los nodos. El consumo de ancho de banda crece debido a que cada vez Internet se extiende más y ofrece más servicios. El cuello de botella de las redes troncales de cara al futuro lo constituyen los nodos, la razón es que estos nodos son electrónicos, es necesario realizar un proceso de conversión óptico-electrónico y electrónico-óptico. Como respuesta a este problema se está trabajando en nuevos nodos de

conmutación ópticos que sean capaces de conmutar el enorme ancho de banda que ofrecen los enlaces de fibra óptica. La conmutación óptica de paquetes (OPS) permite aprovechar al máximo los recursos de la red.

1.2.2 Qué deben hacer los alumnos

Los alumnos deben implementar por parejas básicamente dos módulos simples en C++ y NED llamados respectivamente **generador de tráfico** y **nodo**. Lo que deben hacer estos módulos se detalla en esta propuesta. Estos módulos deberán ser incorporados al entorno de simulación OMNET++, es decir, se añadirán el resto de módulos que sean necesarios. Con el simulador completo, se deberán tomar una serie de medidas, que quedarán recogidas en una memoria final, junto con el código comentado.

Las funcionalidades de los módulos deben realizarse de forma incremental. Para verificar el funcionamiento de dichas funcionalidades se proponen una serie de escenarios de simulación. Para cada uno de estos escenarios se plantean una serie de experimentos de los que el alumno **deberá obtener unas medidas concretas y extraer conclusiones**. El alumno deberá programar la toma de muestras para la obtención de estas medidas.

Se deberá entregar una memoria del trabajo realizado. El contenido de la memoria y los criterios de evaluación también están descritos en esta propuesta. Conviene destacar por adelantado que para alcanzar el aprobado no es necesario que se implementen en su totalidad los algoritmos y medidas propuestos en esta memoria.

1.2.3 Cómo está organizada esta propuesta.

En los apartados 1.3 a 1.3.4 se proporciona al alumno una descripción de la red y de las funcionalidades que se van a implementar.
En el apartado 2 se describen uno a uno los escenarios que se deben configurar y se especifican las medidas que se deben tomar en cada uno de ellos.
En el apartado 3 se dan algunas sugerencias y "pistas" para la realización de este trabajo.
Finalmente en el apartado 4 se especifican los criterios de evaluación y en 0 se da una breve reseña bibliográfica que se recomienda consultar para resolver dudas y profundizar en los aspectos descritos.

1.3 Red EuroSim

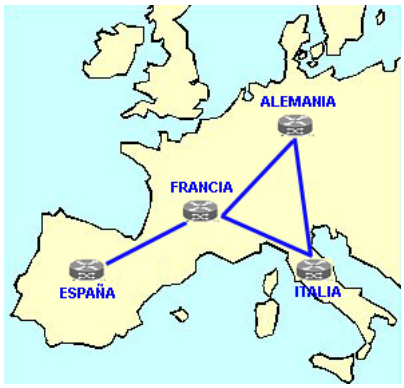

Figura 2. Red EuroSim

La figura 1 representa la topología de la red a simular, que llamaremos EuroSim. Consta de 4 nodos de conmutación y 4 generadores de tráfico (cada uno conectado a un conmutador).

Nodo ***España*** (***E***): link al generador y link bidireccional al nodo ***F***
Nodo ***Francia*** (***F***): link al generador y links bidireccionales a los nodos ***E***, ***I*** y ***A***
Nodo ***Italia*** (***I***): link al generador y links bidireccionales a los nodos ***F*** y ***A***
Nodo ***Alemania*** (***A***): link al generador y links bidireccionales a los nodos ***F*** y ***I***

1.3.1 Descripción general

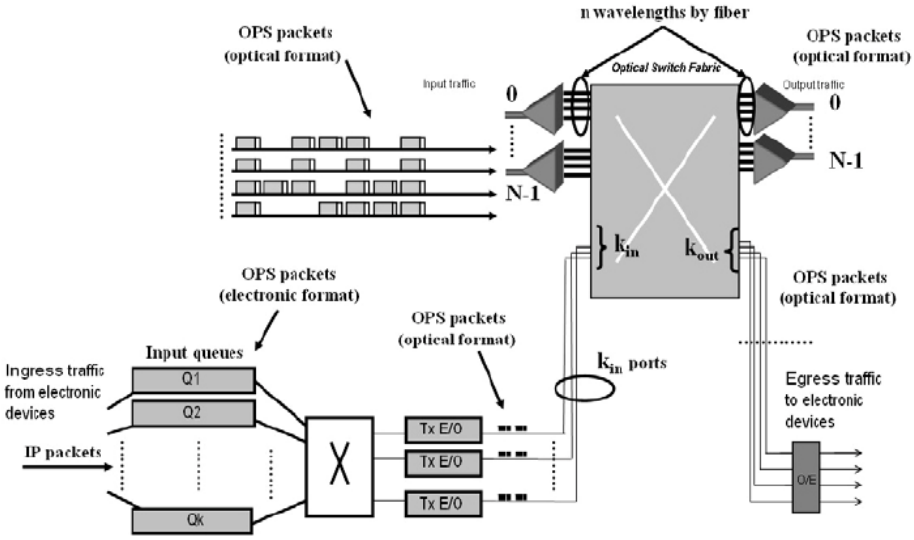

Figura 3. Esquema de un nodo OPS

El trafico llega a la red troncal óptica en formato electrónico, típicamente paquetes IP, como el ancho de banda de estas redes troncales es muy elevado, los paquetes electrónicos se agrupan teniendo en cuenta el nodo troncal destino y la QoS (Quality of Service) demandada, formándose así un paquete mucho mayor. Este nuevo paquete, se convierte a formato óptico. En la simulación trabajaremos directamente con esos paquetes ópticos, olvidándonos del proceso previo de ensamblado.

EuroSim será una red síncrona de conmutación óptica de paquetes, con paquetes de tamaño fijo (1µs). Por lo tanto, tendremos el tiempo ranurado en slots de 1µs.

Los paquetes se transmiten por fibras ópticas, una fibra óptica consta de varias longitudes de onda, cada longitud de onda constituye un canal físicamente independiente de los demás

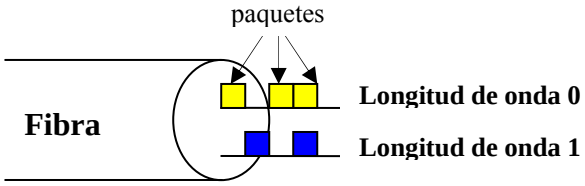

Figura 4. Ejemplo de fibra optica con 2 longitudes de onda

En la red EuroSim todas las fibras tendrán 8 longitudes de onda. Esto es, podremos enviar de forma simultánea 8 paquetes por cada enlace.

Puesto que las redes ópticas son muy rápidas y robustas consideraremos ideales los enlaces en la simulación.

1.3.2 Generador

Cada slot de tiempo el generador deberá enviar paquetes al conmutador, para ello, deberá enviarse a si mismo un mensaje cada slot de tiempo (cada µs). Para simplificar, todos los generadores de *tráfico inyectado a la red* serán iguales y generaran el número de paquetes dado por la distribución binomial de parámetros: *numero de longitudes de onda* y *carga***,** donde carga será uno de los parámetros que el modulo leerá del omnet.ini en cada *run*. De esta forma en cada slot se ejecutara una vez el código del generador y enviara una cantidad (entre 0 y el número de longitudes de onda) de paquetes al conmutador. Nótese que "carga" representa la posibilidad de generar un paquete por cada longitud de onda.

Para elegir el destino de cada paquete seguiremos la siguiente distribución de probabilidades:

Destino	Probabilidad
España (E)	0,265
Italia (I)	0,245
Francia (F)	0,245
Alemania (A)	0,245

Tabla 1. Distribución de probabilidades para elección de destino

Teniendo en cuenta que un *generador de trafico inyectado* no enviará paquetes con destino al propio nodo. Por tanto $P(\text{destino } j) = P_j/\Sigma(p_i)$. Por ejemplo, para el nodo E, P(destino I) = 0.245 / (0.245 + 0.245 + 0.245) = 0.33

Para conseguir estas distribuciones de probabilidades usaremos la Uniforme (0,1). Siguiendo el ejemplo del nodo E, si el resultado de la U(0,1) es menor que 0.33 el destino será el nodo I, si esta entre 0,33 y 0,67 el destino será el nodo F y si es mayor que 0,67 el destino será el nodo A.

1.3.3 Nodos de conmutación

Debido a que la red es síncrona, los paquetes llegan al nodo de conmutación todos al mismo tiempo gracias a unos dispositivos que se encargan de sincronizar los paquetes, compensando los desajustes causados por las fibras. Por tanto, al inicio de cada slot de tiempo, el conmutador debe leer la cabecera de cada paquete que tiene en las fibras de entrada, sacar de la red los que ya hayan llegado a su destino, y para el resto consultar en una tabla la fibra de salida que les corresponde y transmitirlos si la fibra correspondiente tiene longitudes de onda disponibles, si no, los almacenará en **FDLs (Fiber Delay Line)** y si todas las FDLs están ocupadas tirará el paquete.

Puesto que todos los paquetes llegan en el mismo instante, haremos que cada vez que llegue un paquete el conmutador lo borre si ya ha llegado al destino, o lo procese y envíe con *sendDelayed()* si tiene éxito o lo elimine en caso contrario.

Como los paquetes están en formato óptico hoy en día la única forma viable de poder retardar un paquete es enviándolo por un **lazo de fibra (FDL)**, de forma que al cabo de un tiempo cuando el paquete recorra la fibra volverá a estar disponible para ser transmitido. Las FDL tendrán una duración múltiplo de un slot de tiempo. Es decir, si un nodo tiene 4 FDLs, las duraciones de las mismas serán 1 slot, 2 slots, 3 slots y 4 slots respectivamente.

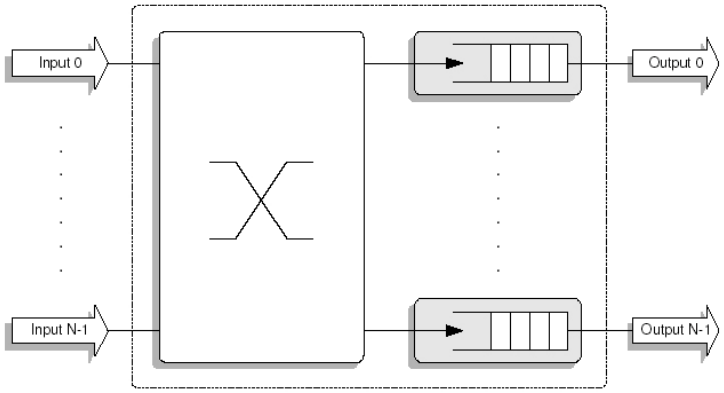

Figura 5. Conmutador de colas a la salida

La arquitectura de los conmutadores será **colas a la salida**, que como sabéis son los que mejores prestaciones ofrecen (aunque también son los más costosos de fabricar).

Para controlar la **contención a la salida** (en un slot de tiempo dado no puede salir más de un paquete por cada longitud de onda), el conmutador necesita tener un vector por cada longitud de onda de salida de cada fibra de B+1 posiciones, siendo B, el numero de FDLs. En cada posición almacenara un **1** si un paquete va salir en ese slot de tiempo por esa longitud de onda y un **0** en caso contrario.

Ejemplo: Fibra 2, Longitud de onda 1, vector: (1,1,0,0,0)

A la vista de ese vector sabemos que están ocupados el slot actual y el siguiente, por tanto para enviar un paquete por esta longitud de onda debemos asignarle la FDL de duración 2 slots, para que no colisione y actualizar el vector a (1,1,1,0,0).

Como el conmutador ejecuta sus tareas una vez cada slot de tiempo lo primero que ha de hacer es desplazar una posición hacia la izquierda el contenido de todos los vectores, para reflejar el paso del tiempo.

Cada microsegundo los generadores y los otros nodos envían todos paquetes al mismo tiempo, ¿Cómo puede hacer el conmutador para asegurarse de desplazar los vectores antes de procesar ningún paquete? La solución adoptada es la siguiente: el conmutador (al igual que el generador) va a enviarse a si mismo un mensaje cada microsegundo, y al recibir este mensaje lo único que hará será desplazar los vectores. Tanto este mensaje como los nuevos paquetes llegaran todos al conmutador en el mismo instante de tiempo. Para asegurarnos de que el mensaje de actualización de vectores se ejecute en primer lugar le asignaremos prioridad 0 (msg->setPriority (0)), que es la prioridad mas alta.

En la red *EuroSim* todos los conmutadores tendrán el mismo numero de FDLs (B = 8)

Cada paquete ha de ser enviado por una fibra de salida determinada, pero la longitud de onda la elige el conmutador (SCWP). Hay distintos criterios a la hora de asignar longitud de onda, veremos 3: (a) Arbitrario (usando una función aleatoria) se escoge una longitud de onda de forma arbitraria y se le asigna el menor retardo posible según el contenido del vector, si no hay sitio libre se tira el paquete. (b) Round-Robin. Tendremos un puntero que apunta a la longitud de onda por la que enviaremos el paquete que iremos incrementando en una posición tras cada envió. Se le asigna el menor retardo posible según el contenido del vector, si no hay sitio libre se tira el paquete. (c) Mejor caso, Se comprueban los vectores de todas las longitudes de onda de la fibra y se envía por la que ofrezca un menor retardo, si están todas ocupadas se tira el paquete.

1.3.3.1 ALGORITMO

El conmutador se *"despertará"* cada vez que le llegue un evento y ejecutará ***handleMessage.*** El algoritmo que ha de implementar es el siguiente:

1 - ***Comprobar si es evento actualización de vectores o evento paquete***

1.a - ***Actualización de vectores***: Desplaza una posición el contenido de cada uno de los vectores de las longitudes de onda de todas las fibras de salida.

1.b - ***Nuevo paquete:*** Comprueba destino del paquete

1.b.a - ***El destino del paquete es el propio nodo: (ESCENARIO 2)*** Actualiza variables estadísticas de retardo total sufrido por el paquete y borra el paquete

1.b.a - ***El destino del paquete es el propio nodo: (ESCENARIO 1)*** Borra el paquete.

1.b.b - ***El destino del paquete es otro nodo.***

1.b.b.1 - ***Obtiene la fibra de salida del paquete,*** usando la información de la tabla de encaminamiento.

1.b.b.2 - ***Busca la longitud de onda de salida del paquete,*** determinada por el criterio correspondiente (Arbitrario, Round-Robin o Mejor caso).

1.b.b.3 - ***Comprueba si la longitud de onda esta libre.*** Consultando los vectores de ocupación.

1.b.b.3.a - ***Vector lleno*** (unos en todas las posiciones): Actualiza variables estadísticas de paquetes perdidos y borra el paquete.

1.b.b.3.b - ***Vector con posiciones disponibles.***

1.b.b.3.b.1 – *Asigna retardo:* Selecciona la posición libre de menor retardo y pone a uno esa posición del vector.

1.b.b.3.b.2 – (ESCENARIO 2) *Envía el paquete* empleando la función *sendDelayed()* y el valor de retardo obtenido en el paso anterior.

1.b.b.3.b.2 – (ESCENARIO 1) Actualiza variables estadísticas de retardo asignado y borra el paquete.

Nota: En el *Escenario 1*, el conmutador debido al hecho de ser solamente uno, tiene un funcionamiento ligeramente diferente al *Escenario 2***.** Para las medidas estadísticas en el *Escenario 1*, solamente manejamos paquetes enviados y paquetes perdidos, en cambio en el *Escenario 2*, tenemos paquetes enviados para cada destino y paquetes perdidos por destino.

1.3.4 Plano de control

Para realizar el control de red en redes ópticas tales como OPS todavía no hay estándares, pero el mecanismo mejor posicionado es GMPLS. MPLS significa Conmutación Multi-Protocolo mediante Etiquetas. La mayor ventaja que ofrece MPLS a las redes OPS es que permite un control de tráfico muy eficiente.

Nosotros para simplificar el trabajo en vez de implementar MPLS vamos a utilizar direccionamiento absoluto basado en destino, es decir, cuando el nodo A quiera enviar un paquete al nodo E, pondrá "E" en la cabecera de ese paquete. Cada nodo tendrá una tabla que le indique la fibra de salida para cada paquete dependiendo de su cabecera.

Nodo España (E)	
Destino	**Fibra de salida**
Francia (F)	0
Alemania (A)	0
Italia (I)	0

Nodo Francia (F)	
Destino	**Fibra de salida**
España (E)	0
Alemania (A)	1
Italia (I)	2

Nodo Italia (I)	
Destino	**Fibra de salida**
Alemania (A)	0
Francia (F)	1
España (E)	1

Nodo Alemania (A)	
Destino	**Fibra de salida**
Francia (F)	0
Italia (I)	1
España (E)	0

Tabla 2. Tablas de encaminamiento de los 4 nodos troncales

2 Fases del desarrollo del simulador y medidas a realizar

A continuación se describen los escenarios de simulación que deben implementarse, su configuración y las medidas a realizar.

La estructura de los paquetes será la siguiente, tendremos un fichero llamado **OpticalPacket.msg** con el siguiente contenido:

```
message OpticalPacket
{
   fields:
      char idDestination;              // identifica el nodo destino del paquete
      double generationTime;           //el tiempo de generación del paquete óptico
};
```

2.1 Escenario 1. Red Mono-Nodo Alemania

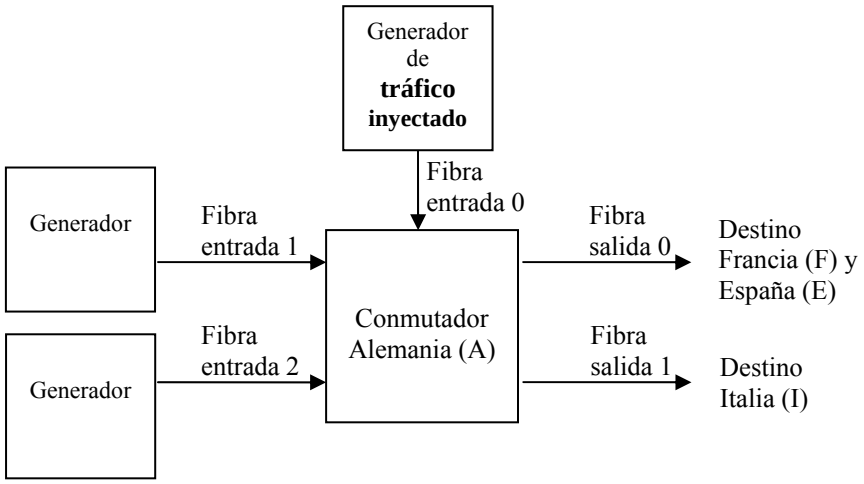

Figura 6. Esquema red Mono-Nodo Alemania

En este escenario debe implementar la red de la figura 6. El generador de *tráfico inyectado a la red* no genera paquetes destinados al propio nodo. Los otros dos generadores, **generan paquetes para los 4 destinos** (incluido el propio nodo) según la ***tabla 1.***
Para poder obtener estadísticas del funcionamiento del sistema, los módulos *generador* (tanto el de *tráfico inyectado a la red* como los otros) deben guardar el fichero (uno por cada modulo):

Resultados_generador_X.txt

Carga	paquetes_enviados	intervalo_confianza
0.1		
0.2		
0.3		
...		

Nota: El generador de *tráfico inyectado a la red* genera un fichero llamado: *Resultados_generador_A.txt* y los otros dos generadores: *Resultados_generador_auxiliar_fibra_1.txt* y *Resultados_generador_auxiliar_fibra_2.txt.*

El módulo *conmutador* debe guardar el siguiente fichero:

Resultados_conmutador_A.txt

Carga	paquetes_perdidos	intervalo_confianza	retardo_medio	intervalo_confianza
0.1				
0.2				
0.3				
…				

Cada vez que enviamos un paquete, según el retardo asignado incrementamos en una unidad la posición correspondiente de un vector de tamaño $B + 1$. El contenido de este vector es lo que usaremos para calcular el retardo medio.

Nota: Antes de generar las graficas es preciso unir resultados tanto de generadores (numero de paquetes creados) como del conmutador (numero de paquetes perdidos), para poder calcular la probabilidad de perdida de paquete y el retardo.

Las medidas que deben realizar para este escenario son las siguientes y se **repetirán para cada uno de los 3 criterios** de asignación de longitudes de onda propuestos para el conmutador:

1. **Influencia de la carga de tráfico de entrada en la PLP.** El parámetro carga tomará los siguientes valores: 0.1, 0.2, 0.3,.. 0.8. El objetivo es obtener una **representación de la probabilidad de perdida de paquete frente a la carga de entrada**. Debe representar el valor medio de la VA *paquetes perdidos / paquetes generados* en el eje de ordenadas y la carga (0.1, 0.2, 0.3,.. 0.8) en el eje de abcisas. El eje de ordenadas debe estar en escala logarítmica.
2. **Influencia de la carga de tráfico de entrada en el retardo medio.** El parámetro carga tomará los siguientes valores: 0.1, 0.2, 0.3,.. 0.8. El objetivo es obtener una **representación del retardo medio frente a la carga de entrada**. Debe representar el valor medio de la VA *retardo medio* en el eje de ordenadas y la carga (0.1, 0.2, 0.3,.. 0.8) en el eje de abcisas.

Se valorará especialmente el cálculo de los intervalos de confianza de las medidas, que representará en la gráfica en forma de barras de error asociadas a cada punto.

Se valorarán los comentarios argumentados y justificación de los resultados que ha obtenido.

2.2 Escenario 2: Red EuroSim

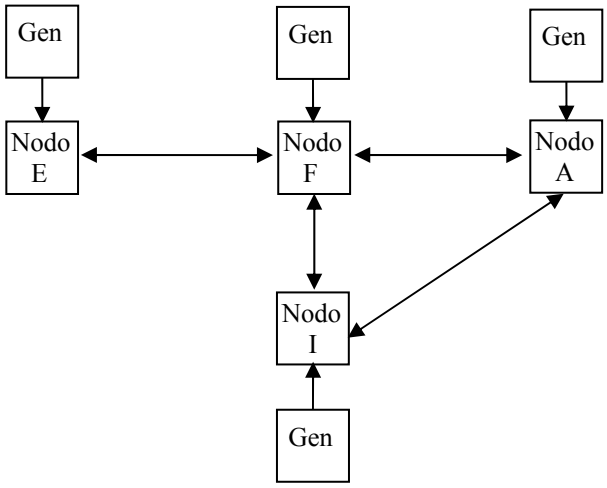

Figura 7. Esquema red EuroSim

En este escenario debe implementar la red de 4 nodos de la figura 7. Todos los generadores de la figura son generadores de *tráfico inyectado a la red* y por tanto no generan paquetes con destino al propio nodo.

Para poder obtener estadísticas del funcionamiento del sistema, los módulos *generador* deben guardar el fichero:

Resultados_generador_X.txt

Carga	paquetes_enviados_destino_Y	…	intervalo_confianza
0.1			
0.2			
0.3			
...			

Nota: Los generadores de *tráfico inyectado a la red* generan un fichero llamado: *Resultados_generador_X.txt* donde la *X* es el identificador del nodo al que están conectados.

Los módulos *conmutador* deben guardar el siguiente fichero:

Resultados_conmutador_X.txt

Carga	paquetes_perdidos_destino_Y …	intervalo_confianza	retardo_medio	intervalo_confianza
0.1				
0.2				
0.3				
…				

Cuando un paquete tiene como destino final este nodo, calculamos el retardo total que ha sufrido e incrementamos en una unidad la posición correspondiente de un vector de tamaño *B* * (Número de nodos - 1) + 1. El contenido de este vector es lo que utilizaremos para calcular el retardo medio. En la posición cero de este vector se guardará el número de paquetes que ha sufrido retardo total cero slots, en la posición uno los paquetes que hayan recibido retardo total un slot, y así sucesivamente.

Nota: Antes de generar las gráficas es preciso unir resultados tanto de generadores (numero de paquetes creados por cada destino) como de conmutadores (numero de paquetes perdidos por cada destino), para poder calcular la probabilidad de perdida de paquete por destino y el retardo medio por destino.

Las medidas que deben realizar para este escenario son las siguientes y se **realizarán empleando únicamente el criterio de asignación de lambdas Round-Robin**:

1. **Influencia de la carga de tráfico de entrada en la PLP.** El parámetro carga tomará los siguientes valores: 0.1, 0.2, 0.3,.. 0.8. El objetivo es obtener una **representación de la probabilidad de perdida de paquete por destino frente a la carga de entrada**. Debe representar el valor medio de la VA *paquetes perdidos / paquetes generados* para cada uno de los 4 posibles destinos (E, A, I y F) en el eje de ordenadas y la carga (0.1, 0.2, 0.3,.. 0.8) en el eje de abcisas. El **eje de ordenadas debe estar en escala logarítmica**.
2. **Influencia de la carga de tráfico de entrada en el retardo medio.** El parámetro carga tomará los siguientes valores: 0.1, 0.2, 0.3,.. 0.8. El objetivo es obtener una **representación del retardo medio por destino frente a la carga de entrada**. Debe representar el valor medio de la VA *retardo medio* para cada uno de los 4 posibles destinos (E, A, I y F) en el eje de ordenadas y la carga (0.1, 0.2, 0.3,.. 0.8) en el eje de abcisas.

Se valorará especialmente el cálculo de los intervalos de confianza de las medidas, que representará en la gráfica en forma de barras de error asociadas a cada punto.

Se valorarán los comentarios argumentados a los resultados que ha obtenido.

3 Sugerencias para la implementación

3.1 Consejos para la depuración y verificación del módulo

Para las tareas de depuración, aquí tiene unos consejos:

Uso de variables de depuración: Consiste en definir ciertas variables booleanas en el módulo y cuyo valor se puede ser definido por el usuario (por ejemplo como parámetros de un módulo). Estas variables servirán de condición inicial para que se ejecuten líneas de código destinadas únicamente a la depuración. Por ejemplo:
Se define la variable (o flag) debug = true, en la inicialización del módulo.
A lo largo del código se insertan líneas del tipo:
if (debug) ev << "El Switch marca una celda CLP = 1"<<endl;
de esta forma se pueden activar y desactivar los mensajes que interesen en función de lo que se esté depurando, sin tener que borrar líneas de código cada vez y sin que se sature la interfaz gráfica con mensajes que ya no interesan.

Presentar en pantalla y en tiempo de ejecución la evolución de determinadas variables del sistema. Hay dos formas: puede utilizar la macro WATCH() de OMNET (consulte el manual) o definir objetos cOutVector (uno por cada variable que se desee monitorizar) y cada vez que se actualice el valor de la variable en cuestión, se llamará al método record(…) de dichos objetos. Consulte el manual y el API para más información. El entorno gráfico puede presentar en pantalla la evolución de la gráfica generada por cada objeto cOutVector definido en el módulo.

En ocasiones la depuración mediante mensajes en pantalla resulta muy tediosa. Puede serle de mayor utilidad generar ficheros de trazas en los que los módulos vayan escribiendo datos de utilidad. A continuación tiene un ejemplo:

Al principio del fichero debe incluir:

```
#include <fstream>
```

Para abrir un fichero para escritura (borrando su contenido anterior) se emplea la siguiente instrucción:

```
ofstream FicheroControl("Traza.dat",ios::out);
```

Para abrir un fichero para escritura (escribiendo a continuación del último carácter) se emplea la siguiente instrucción:

```
ofstream FicheroControl("Traza.dat",ios::app);
```

Para escribir en el fichero, se emplea el operador <<.

```
FicheroControl<<simTime()<<" Se descarta una celda " <<endl;
```

Al final de las funciones donde emplee un fichero debe cerrarlo.

```
FicheroControl.close();
```

Otra opción posible es añadir un macro a los ficheros de cabecera:

```
#define O(x) if (debug) cerr<<"transmisor:\t"<<x<<"\t"<<simTime()<<endl
```

Así podemos utilizar la función O(x) en cualquier parte del código y se imprimirá la cadena x si se define y establece la variable debug. La cadena se imprime a la salida de error estándar. Se puede redireccionar dicha salida para guardar la depuración en un fichero:

```
./simulador 2> ficheroError
```

3.2 Consejos para una mayor claridad en el código.

Cuanto más claro sea su código, más sencillo le será depurarlo y añadirle funcionalidades posteriormente. En general el desarrollo será más rápido si sigue una serie de normas básicas:

1) Dedique el tiempo suficiente a comprender bien el funcionamiento de los algoritmos y a hacerse un idea general del funcionamiento del módulo que debe implementar. Antes de empezar a teclear código es conveniente que sepa perfectamente qué es lo que va a programar, para ello escriba en papel la estructura general de su programa y para cada una de sus secciones ó funciones escriba el pseudocódigo (o diagrama de flujo) antes de comenzar a programar.

2) Desglose el código en funciones, no lo desarrolle todo en una sola función handleMessage( ). Identifique qué actividades pueden ser agrupadas en una función, y asigne a dicha función un nombre que indique su utilidad. Si determinadas líneas de código se repiten en distintas partes del código es posible que puedan implementarse como una sola función.

3) No espere a tener muchas funcionalidades implementadas para empezar a compilar su código. Cuanto antes lo empiece a depurar, mejor. De la misma forma empiece a ejecutarlo lo antes posible.

4) Comente su código, sobre todo en las partes esenciales de los algoritmos es aconsejable indicar los pasos que realiza, no necesariamente línea a línea.

3.3 Gestión eficiente de la memoria

Es conveniente que se eliminen los mensajes recibidos en los módulos una vez que éstos no van a emplearse más, para evitar que estos se almacenen en la memoria innecesariamente. Sin embargo esta tarea es delicada ya que puede dar lugar a errores en la ejecución no detectables en compilación. Es aconsejable que antes de proceder a la inclusión de instrucciones para la eliminación de mensajes haya implementado y depurado completamente su código. Deje esa tarea para el final.

3.4 Toma de muestras

En el trabajo que se propone deberá tomar una gran cantidad de muestras. Son varias las clases de OMNET++ que tendrá que usar, y deberá apoyarse en el manual y en el API.
cDoubleHistogram
cStdDev
cWeightedStdDev
cOutVector

Además, si va a realizar varias réplicas, le será de utilidad la siguiente funcionalidad de OMNET++. Existe una función de de cModule que le permite escribir resultados numéricos en un fichero de salida. Por ejemplo, supongamos que en su módulo incluye una línea como esta en el método finish:

```
recordScalar("descartados", n_celdas_descartadas);
```

En el fichero de resultados esto daría lugar a una línea como la siguiente:

```
scalar "Red.switch" "descartados "             1456
```

cada vez que se ejecuta el simulador se escriben nuevos resultados en el fichero de resultados. Es decir, no se borran los resultados anteriores. Esto le permitirá ejecutar varias réplicas consecutivas de una experimento, y tener en el fichero los resultados de cada réplica.

El fichero de resultados se especifica en el fichero de configuración omnetpp.ini (igual que el fichero de vectores para objetod cOutVector):

```
output-scalar-file = omnetpp.sca
```

Puede consultar todo esto en el manual y en API.

3.5 Selección de semillas

Si va a ejecutar varias réplicas para aplicar el método de las réplicas independientes debe recordar que debe asignar distintas semillas a cada réplica.

```
num-rngs= 1
```
seed-0-mt = 10677634
Esta línea de omnetpp.ini establece el uso de un generador y asigna la semilla correspondiente a dicho generador.
Tenga en cuenta que OMNET++ utiliza el generador 0 para las variables aleatoria que controla los errores en el canal.
Consulte y compruebe cómo puede cambiar estas semillas.

3.6 Ejecuciones más rápidas

Finalmente, debe saber que el simulador y las medidas que se le han planteado pueden requerir la ejecución de un número muy elevado de eventos. La ejecución puede ser muy lenta si no tiene en cuenta ciertos consejos:

Cuando realice simulaciones para obtener estadísticos, le convendrá que en esas ejecuciones el programa no escriba en ningún fichero, lo cual relentiza enormemente la ejecución. Deberá comentar o desactivar la escritura en ficheros, incluyendo la generación de gráficas con objetos cOutVector. Esto último lo conseguirá sencillamente con una instrucción en omnetpp.ini:

```
[OutVectors]
Red.**.enabled = no
```

De nuevo le insistimos en la importancia de la gestión eficiente de la memoria, es decir, que borre todos los mensajes que no le sean útiles o que los reutilice en la medida de lo posible (por ejemplo, cuando le llegue un automensaje tipo “timer”, vuelva a reenviarselo, en vez de borrarlo y generar uno nuevo). Con esto acelerará las ejecuciones.

Finalmente, la ejecución en el entorno gráfico de OMNET++ es más lenta que en el entorno del interfaz de línea de comandos (sin ventanas). Para compilar el código en ese entorno (denominado Cmdenv) debe hacer lo siguiente:

```
opp_makemake –f –u Cmdenv
```

En omnetpp.ini podrá incluir las siguientes líneas que le proporcionarán mayor velocidad de ejecución:

```
[Cmdenv]
express-run = true
express-mode = yes
event-banners = no
```

Nuevamente le animamos a que consulte el manual.

3.7 Uso de la librería de plantillas STL

La librería de plantillas estándar STL (Standard Template Library) es una de las capacidades más potentes de C++. No debe desaprovechar sus capacidades. Las STL le proporcionan entre otros, los siguientes contenedores:

- strings
- vectores
- listas simple y doblemente enlazadas
- colas, pilas
- contenedores asociativos

Recordemos su uso con algunos ejemplos. Suponga que desea un vector de Figuras, para almacenar Cuadrados y Triángulos (hará uso del polimorfismo). Entonces, algunos ejemplos de uso de las STL son:

```
#include <vector.h>

vector<Figura*> formas;
formas.push_back(new Cuadrado());
formas.push_back(new Triángulo());
for (int i=0; i<formas.size();i++) {formas[0]->dibujar();} //Llama al método dibujar de
la clase Figura
```

También puede utilizar los iteradores:

```
#include <list.h>
list<Figura*> formas;
list<Figura*>::iterator it;
while(it!=formas.end()){
	if ((*it)->getArea()<10){ //Un iterador es un puntero, debe dereferenciarlo antes
de usarlo (*it)
	Figura * f_aux = *it;
	formas.erase(it);
	delete f_aux;
}
it++;
}
```

Si utiliza las STL evitará errores y seguramente conseguirá mejor rendimiento, ya que están diseñadas buscando la máxima eficiencia. Además los contenedores de las STL se pueden combinar (de hecho, se utilizan habitualmente combinados). Puede obtener más ejemplos de uso en el código de prácticas anteriores y en libros especializados.

4 Material a entregar y criterios de evaluación

Los alumnos deberán entregar una memoria en papel del trabajo, junto con el código fuente y un ejecutable compilado del simulador. La entrega y evaluación se realizará en Junio. Se avisará con suficiente antelación de la fecha límite de entrega, que será aproximadamente 5 ó 6 días antes de la fecha de entrega de actas. **No se admitirá ningún trabajo en fechas posteriores a la fecha límite**. Es posible que en algún caso los profesores necesiten reunirse con los alumnos de un grupo para evaluar su trabajo. Para esos casos **se publicará una lista de los grupos que deben entrevistarse con los profesores para explicar su trabajo.**

4.1 Memoria y Código

La memoria debe contener, al menos los siguientes apartados:

- Nombre completo de los componentes del grupo, correo electrónico y, al menos, un teléfono de contacto.
- Índice
- Código implementado comentado. Deberá contener, al menos, un subapartado por cada algoritmo implementado
- Por cada escenario resuelto, se incluirá un apartado en el que se darán los resultados numéricos obtenidos para cada configuración, las gráficas (en su caso) y una interpretación de los resultados.

El código se entregará en un CD, que deberá contener, como ya se ha indicado: el código fuente y un ejecutable compilado del simulador. El código fuente entregado debe compilar y ejecutarse sin errores.

4.2 Criterios de evaluación

La puntuación total del trabajo se ha repartido en los distintos escenarios propuestos. La puntuación total de cada escenario se muestra en la siguiente tabla:

Apartado	Puntuación máxima
Escenario 1.	6
Escenario 2.	4

En cada escenario se tendrá en cuenta (en orden de importancia):

- Que se hayan implementado correctamente las funcionalidades necesarias.
- Que se hayan programado y ejecutado correctamente las tomas de muestras y la obtención de estadísticos y, en su caso, gráficas.
- Que para cada estadístico se proporcione el intervalo de confianza de las medidas, el tiempo de simulación, el número de réplicas, las semillas empleadas, etc.
- Se valorará la interpretación dada de los resultados y se apreciará especialmente que los alumnos demuestren haber consultado bibliografía, Internet, etc, para mejorar sus conclusiones.
- Se valorará también la claridad, rigor y concisión en las explicaciones aportadas en la memoria, así como la claridad en la presentación de gráficas y resultados.
- Se valorará también la claridad y orden en el código. En general cuantas menos líneas de código se empleen (sin quitar funcionalidades) la implementación será mejor y más clara. También se tendrá en cuenta la gestión eficiente de memoria.

Puede comprobar que **no es necesario completar todos los apartados para obtener el aprobado** en el trabajo.

4.3 Grupos de una ó tres personas

Al inicio de esta propuesta se indica que el trabajo debe realizarse en parejas, pero en ocasiones las circunstancias impiden formar ó mantener un grupo y se forman grupos de tres personas o de una sola persona. Como no es justo aplicar los mismos criterios de evaluación en estos grupos, en estos casos se aplicará el siguiente reparto de puntos:

Grupos de una persona:

Apartado	Puntuación máxima
Escenario 1.	7
Escenario 2.	3

Grupos de tres personas:

Apartado	Puntuación máxima
Escenario 1.	4
Escenario 2.	6

5 Bibliografía

[1] András Varga, “OMNET++ discrete Event Simulation System, Versión 3.2. User Manual.”
[2] H. Deitel, “C++, Como programar”, Prentice Hall, 1999.