Universidad Politécnica de Cartagena

Escuela Técnica Superior de Ingeniería de Telecomunicación

PRÁCTICAS DE REDES DE ORDENADORES

Propuesta del Trabajo de Prácticas 2005

Simulación de un protocolo de encaminamiento de estado de enlace

Profesores:

Esteban Egea López
Juan José Alcaraz Espín
Joan García Haro

Índice

Índice. 2
1 Consideraciones generales. 3
1.1 Objetivos 3
1.2 Introducción 3
1.2.1 Qué deben hacer los alumnos. 3
1.2.2 Cómo está organizada esta propuesta. 3
1.3 Protocolos de encaminamiento de estado de enlace 4
1.4 Inundación fiable 4
1.5 Cálculo de rutas 4
1.6 Métricas 5
2 Requerimientos del simulador a realizar 6
2.1 Topología de la red a simular 6
2.2 Elementos que componen el simulador 7
2.3 Módulo Red 7
2.4 Definición del la red en NED 8
2.5 Enlaces 8
2.6 Estadísticos 8
3 Sugerencias para la implementación 8
3.1 handleMessage 8
3.2 Consejos para una mayor claridad en el código. 8
3.3 Uso de la librería de plantillas STL 9
3.4 Fases del desarrollo 10
3.5 Utilidades de OMNET++ 10
3.6 Consejos para la depuración, verificación y validación del módulo 11
3.7 Gestión eficiente de la memoria 11
4 Medidas a Realizar 11
4.1 Selección de la semilla 11
4.2 Validación del simulador 12
4.3 Experimento 1. Ajuste de tasas de las fuentes de tráfico 12
4.4 Experimento 2. Variación del tipo de tráfico 12
5 Método y Criterios de Evaluación 13
5.1 Memoria 13
5.2 Criterios de evaluación 13
6 Bibliografía: 14

1 Consideraciones generales.

1.1 Objetivos

En este trabajo de prácticas los alumnos realizarán por parejas un simulador de una red de encaminadores. Cada nodo constará de una fuente de tráfico y un nivel de red que implementa las funciones de encaminamiento y reenvío. Se utilizará un protocolo de encaminamiento por estado de enlace. Se validará mediante la obtención de gráficas que muestren la evolución de diversos parámetros cuyo comportamiento es conocido. En este entorno se comprobará la utilidad de una herramienta de simulación para evaluar las prestaciones de una red y el dimensionado de sus nodos y enlaces.

1.2 Introducción

1.2.1 Qué deben hacer los alumnos.

Los alumnos deben implementar los módulos simples en C++ y NED llamados Red y Aplicacion. Lo que deben hacer estos módulos se detalla en esta propuesta. Los alumnos también deberán declarar la red a simular mediante un módulo compuesto en NED.

Una vez que los módulos estén implementados, integrados en la red y depurados se podrán realizar medidas de ciertos parámetros. Estas medidas también deberán ser programadas por el alumno y vienen especificadas en este documento.

Se deberá entregar una memoria del trabajo realizado, el código fuente y un ejecutable y se evaluará tanto el programa como la memoria. El contenido de la memoria y los criterios de evaluación también están descritos en esta propuesta.

1.2.2 Cómo está organizada esta propuesta.

A lo largo de esta propuesta se detalla exactamente lo que deben hacer los módulos, qué deben implementar, como ya se ha indicado, nodos con un nivel de red que encamina los paquetes mediante un algoritmo de estado de enlace. Los algoritmos están descritos en los apartados 1.3,1.4,1.5 y 1.6.

En el apartado 2 se dan indicaciones que serán requisitos indispensables del simulador a realizar, como son la estructura en módulos del mismo y su implementación en NED, y los requisitos mínimos necesarios de la implementación de los módulos.

En el apartado 3 se dan sugerencias para la implementación. No son indispensables, como las del apartado anterior, pero pueden servir de ayuda para conseguir un código mejor y en menor tiempo.

En el apartado 4 se describen las medidas a realizar y los escenarios de referencia.

En el apartado 5 se describe el formato de la memoria a entregar, cómo será el examen que realizarán los alumnos y los criterios que se tendrán en cuenta a la hora de evaluar.

El apartado 6 es de bibliografía relacionada.

1.3 Protocolos de encaminamiento de estado de enlace

Los protocolos de encaminamiento basados en estado de enlace (link-state) son los más usados en la práctica junto con los de vector de distancia. Parten de la asunción de que un nodo conoce el estado de sus enlaces con otros nodos. Si un nodo tiene información suficiente, puede calcular el mejor camino hacia cualquier destino. La idea básica es que los nodos diseminen la información que tienen (el estado de los enlaces con sus vecinos directos). Cuándo esta información esté disponible para cada nodo, podrá construir un mapa de la red y elegir el mejor camino a cada destino. Esta es una condición suficiente para encontrar ese mejor camino. Los protocolos de estado de enlace se basan en dos mecanismos para conseguir su propósito: diseminación fiable de la información de estado de enlace y cálculo de las rutas a partir de la información acumulada.

1.4 Inundación fiable

Este mecanismo se encarga de asegurar que todos los nodos obtienen una copia de la información de estado de enlace. Llamamos LSP (Link-State Packet) a los paquetes de información de estado de enlace. La inundación se realiza reenviando los LSP que un nodo recibe por todos sus enlaces excepto por el que le llegó. Este proceso debe continuar hasta que toda la información ha llegado a todos los nodos de la red.

Cada nodo genera un paquete LSP con la siguiente información:

- El ID del nodo que creó el LSP (mi ID).
- Una lista de nodos directamente conectados a este nodo (a mí mismo), junto con el coste de cada enlace. Cada entrada de esta lista se denomina Vecino en el algoritmo descrito en 1.5.
- Un número de secuencia.
- Un tiempo de vida del paquete (TTL).

Los dos primeros campos se utilizan para calcular rutas. Los dos últimos para hacer fiable el proceso de inundación. Esto es: asegurar que todos los nodos reciben una copia del LSP y que el LSP sea el más reciente.
El proceso es el siguiente:

1. Un nodo genera su LSP y lo envía por todos sus enlaces.
2. Cuando un nodo recibe un LSP comprueba si ya lo tiene, consultando el campo ID del mensaje. Hay dos opciones:
    a. Si no lo tiene, almacena una copia y lo reenvía por todos los enlaces excepto por el que le llegó.
    b. Si lo tiene. Compara el número de secuencia del nuevo con el del LSP que posee.
        i. Si el número de secuencia del nuevo es menor o igual, implica que el LSP que ha llegado es más antiguo. Se descarta el paquete y no se reenvía nada.
        ii. Si es mayor, implica que el nuevo LSP es más reciente que el almacenado. Se almacena el nuevo (reemplazando al anterior) y se reenvía por todos los puestos excepto por el que llegó.
3. Cada vez que se reenvía un paquete se decrementa el TTL. Si el TTL llega a 0 se descarta el paquete y no se reenvía.

Este mecanismo básico en la práctica es más elaborado y sofisticado. El protocolo OSPF (Open Shortest Path First Protocol) es un protocolo de enlace muy usado en Internet. Hace uso de este mecanismo pero con ciertas mejoras que permiten el mantenimiento de las rutas.

1.5 Cálculo de rutas

Una vez que un nodo dado tiene una copia de un LSP de **todos los nodos del resto de la red**, es capaz de calcular una mapa de la topología de la red y, de este mapa, elegir la mejor ruta a un destino cualquiera. ¿Cómo se calcula esta mejor ruta? Hay varias opciones, entre ellas el algoritmo de Dijkstra. Puede encontrar una descripción teórica del algoritmo en los manuales. En esta propuesta describiremos una realización del algoritmo de Dijkstra que es la que deberán implementar los nodos simulados. Esta implementación se llama "búsqueda hacia delante" (forward search). Y consiste en lo siguiente: cada nodo mantiene, además de una lista de mensajes LSPs recibidos, dos listas más: **Confirmado** y **Provisional,** que contendrán un conjunto de entradas del tipo (**Destino, Coste, PróximoSalto**).
El algoritmo es:

1. Inicializar la lista **Confirmado** con una entrada para mí mismo, con coste 0.
2. Al nodo que se ha añadido a **Confirmado** en el paso anterior le denominaremos **Siguiente** y seleccionamos su LSP.

3. Para cada vecino (**Vecino**) en el LSP seleccionado calcular el coste (**Coste**) para llegar a él como la suma del coste desde mí mismo hasta **Siguiente** y desde **Siguiente** a **Vecino**. $C=C_{yo\text{-}Siguiente} + C_{Siguiente\text{-}Vecino}$.
    a. Si **Vecino** no está en **Confirmado** ni **Provisional**, añadir (**Vecino, Coste, PróximoSalto**) a la lista **Provisional**, siendo **PróximoSalto** la dirección (interfaz o enlace) por dónde tengo que ir para llegar a **Siguiente**.
    b. Si **Vecino** está en la lista **Provisional** pero el **Coste** recién calculado es menor que el coste para **Vecino** que se indica en la lista, sustituir la entrada de la lista por (**Vecino**, **Coste**, **PróximoSalto**), siendo **PróximoSalto** la dirección (interfaz o enlace) por dónde tengo que ir para llegar a **Siguiente**.
4. Si la lista **Provisional** está vacía, para. En caso contrario, mover la entrada de la lista **Provisional** con menor coste a la lista **Confirmado** y volver al paso 2.

Este algoritmo se entiende mejor con un ejemplo. Desarrolle el algoritmo con lápiz y papel con la topología que se simulará (Fig. 2) para entenderlo mejor y comprobar que se obtiene el menor coste para cada ruta. Este algoritmo es una implementación del que han visto en teoría, repase sus apuntes.

Los algoritmos de estado de enlace tienen varias propiedades deseables: se estabilizan rápidamente, no generan mucho tráfico y se adaptan rápidamente a cambios en la topología. Sin embargo, la cantidad de información que se tiene que almacenar en los nodos puede llegar a ser muy grande. Este es un problema general del encaminamiento y un problema específico del problema genérico de la *escalabilidad*.

Como se dijo anteriormente OSPF es una implementación de un protocolo de estado de enlace con características adicionales como autenticación de paquetes, niveles adicionales de jerarquía y equilibrio de carga. En este trabajo lo que tendrá que realizar es un implementación sencilla de un protocolo de estado de enlace.

1.6 Métricas

En los apartados anteriores se asumió que el coste de los enlaces, o métricas, es conocido por los nodos cuando ejecutan el algoritmo. Hay muchas formas de asignar un coste a un enlace. En esta sección veremos algunas que se han demostrado efectivas en la práctica.

1. Asignar coste 1 a todos los enlaces. Supondrá que los paquetes viajan por el camino que llega al destino con un menor número de saltos. Problemas: no distingue a los enlaces ni por capacidad ni por latencia, es decir, asigna el mismo coste a un enlace de 45 Mbps que a uno de 50 Kbps, o a uno con 1 ms de retardo con uno de 500 ms.
2. Asignar mayor coste a los enlaces con mayor número de paquetes en cola. Es muy inestable ya que es una medida artificial de la carga. El resultado es que los paquetes van hacia la cola más corta y no al destino.
3. Ancho de banda y latencia. Parte de la siguiente fórmula para el retardo: Retardo = (Salida-Llegada) + TiempoTransmision + Latencia. Donde TiempoTransmision y Latencia son estáticos para un canal (BW y retardo de canal) y Salida y Llegada son los instantes de tiempo desde que el paquete llega a la cola hasta que se recibe un reconocimiento de llegada al otro extremo. El coste se asigna en función del retardo medio experimentado por paquetes enviados por ese enlace. Este mecanismo también provoca inestabilidad a altas cargas, haciendo que haya enlaces muy ocupados y otros vacíos.
4. "Revised ARPANET routing metric". Intenta resolver los problemas anteriores. Sin entrar en detalles, lo que hace es comprimir el rango dinámico de las métricas y suavizar su variación a lo largo de tiempo.

Nótese que la mayoría de las métricas son dinámicas y cambian a lo largo del tiempo. Los nodos deben enviar paquetes de actualización cuando el cambio en las métricas supera cierto límite. En la Fig.1 se muestra una gráfica de la métrica 4.

En el trabajo a realizar no utilizaremos métricas dinámicas y por tanto, tampoco se realizará actualización de rutas. La métrica a emplear se explica en los apartados 2.1 y 2.3.

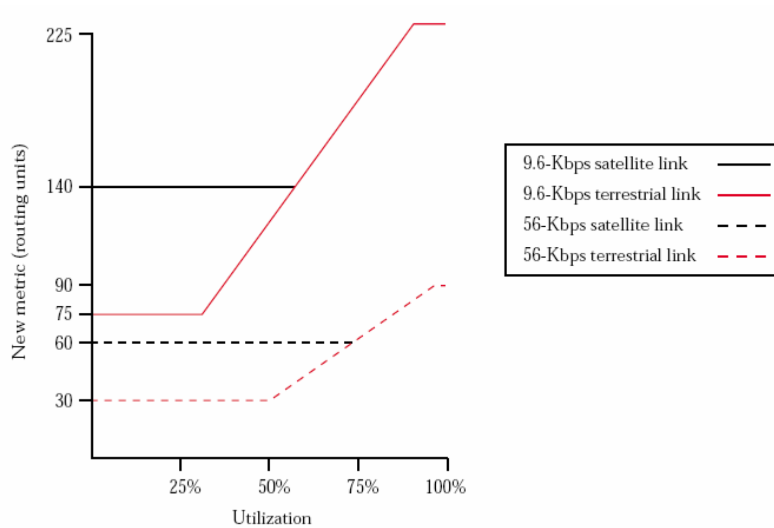

Figura 1

2 Requerimientos del simulador a realizar

2.1 Topología de la red a simular

Se simulará la topología de la Figura 2:

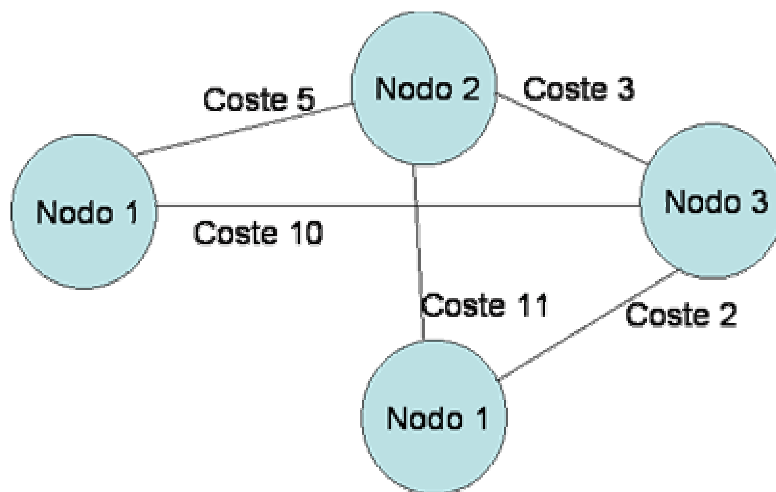

Figura 2

Cada uno de los nodos está compuesto de una fuente de tráfico y un nivel de red que realiza las funciones de encaminamiento y reenvío. Los enlaces son simétricos. Aunque los enlaces se representan en la figura con un coste, en el simulador cada enlace es un canal con una determinada velocidad (bps) y retardo (s). El coste, utilizado para calcular las rutas en el trabajo a realizar, será la función de la velocidad. En el apartado 2.3 se explicará la relación entre Coste y Ancho de Banda.

2.2 Elementos que componen el simulador

El simulador estará compuesto por los siguientes módulos:

Nodo: Módulo compuesto. Formado por una fuente de tráfico y un módulo de red. Los nodos están conectados entre sí según la figura 2. Habrá, por tanto, nodos con 2 y 3 enlaces. El número de enlaces es un parámetro de este módulo. Cada enlace es simétrico y tiene una velocidad (bps) y un retardo (s). Los parámetros de este módulo son:

- id: un número único que identifica al módulo.
- enlaces: el número de enlaces del nodo.

Aplicacion: Módulo simple que actúa como fuente de tráfico. Este módulo genera Paquetes que envía por la red a un determinado destino. El tiempo entre llegadas de Paquetes sigue una distribución exponencial. La longitud de los paquetes es variable. Admite los siguientes parámetros:

- media: la media de la distribución del tiempo entre llegadas de los Paquetes.
- transitorio: un tiempo fijo antes de la generación del primer paquete. Se utiliza para dar tiempo al módulo red para calcular la tabla de rutas.
- id: identificador del módulo, coincide con el identificador de su correspondiente **Nodo**.
- longitud: tamaño de los paquetes (en bits).
- destino: el id del nodo al que van dirigidos los paquetes que genera esta fuente.

Red: Módulo simple que realiza las funciones de encaminamiento y reenvío. Este módulo recibe Paquetes tanto del módulo Aplicación como del resto de la red a través de los enlaces del **Nodo**. Admite los siguientes parámetros:

- refBW: Ancho de banda de referencia. Es un valor numérico que se utiliza para calcular el coste de los enlaces.

2.3 Módulo Red

La implementación del módulo Red es el objetivo principal del presente trabajo de prácticas. En este apartado se describirán los requisitos mínimos necesarios que deberá cumplir su implementación. En el apartado 3 se dan sugerencias que pueden servir de referencia.

1. El primer requisito es que el módulo implemente las funciones de encaminamiento y reenvío. La función de encaminamiento consiste en decidir el enlace de salida para cada paquete que llega al **Nodo**. La función de reenvío consiste, como su propio nombre indica, en enviar los paquetes entrantes no dirigidos al módulo por el enlace correspondiente para que lleguen a su destino. Nótese que la función de reenvío necesita de la función encaminamiento para su correcto funcionamiento.
2. La función de encaminamiento requiere un cálculo de las rutas. Este cálculo se debe realizar mediante el algoritmo de estado de enlace descrito en la introducción (apartados 1.3, 1.4, 1.5). Es decir, los nodos calcularán la ruta óptima para cada destino según el algoritmo de Dijkstra, en función de los costes de los enlaces. Para poder ejecutar el algoritmo de Dijkstra es necesario tener conocimiento del coste de los enlaces del resto de los nodos de la red, por tanto, cada módulo de red debe enviar paquetes con la información del estado de sus enlaces (LSP). En resumen:
   a. El módulo debe comunicar el estado de sus enlaces mediante paquetes de estado (LSP). Estos paquetes se diseminan por la red con el mecanismo de inundación fiable descrito en 1.4.
   b. Una vez que el nodo disponga de los LSP del resto de los nodos, está en condiciones de calcular las rutas óptimas a todos los destinos, mediante el algoritmo descrito en 1.6.
3. La métrica que deberá utilizar para el coste de los enlaces es la siguiente: Coste = refBW/anchoBandoEnlace. El valor concreto de los enlaces y de refBW se da en el apartado 4.
4. Para no complicar la simulación se supondrá que la red es estática y los enlaces no fallan. Con estas condiciones no es necesario el mantenimiento de las tablas. Es decir, las rutas se calculan al comienzo y se utilizan durante toda la simulación. A diferencia de los protocolos reales, no es necesario refrescar la información inyectando nuevos LSP a la red cada cierto tiempo. Por tanto, tampoco es necesario recalcular rutas.
5. Los módulos Red deben interoperar con su Aplicación. Tanto enviando a la red los Paquetes que la Aplicación genera, como enviando a la Aplicación los paquetes dirigidos al **Nodo**. Para ello se utilizará la clase Paquete. La clase Paquete debe derivar de mensaje y contiene al menos los campos origen y destino. El módulo red encaminará los Paquetes en función de su destino, lógicamente.

6. Para homogenizar el código es conveniente que todos los trabajos utilicen la misma nomenclatura. La regla que se utilizará es la siguiente: el valor de los parámetros se almacenará en una variable con el mismo nombre en la clase correspondiente. Por ejemplo, el módulo Red tendrá un atributo llamado refBW que almacenará el valor del parámetro refBW .
7. Es un requisito del simulador que el código sea claro y esté bien documentado. Durante la implementación el alumno decidirá qué variables auxiliares pueden ser necesarias, y deberá describirlas claramente en la documentación y en el código. En el apartado 3 se dan sugerencias al respecto.

2.4 Definición del la red en NED

La red deberá ser definida en lenguaje NED por el alumno. Como es sabido, la red debe constar de un módulo compuesto en el que se especifican todos los elementos que la componen y cómo se interconectan (módulos, conexiones, parámetros, etc). Utilice un solo fichero llamado *simulador.ned* para describir toda la estructura del simulador. Este fichero debe contener, por tanto, las declaraciones NED de los módulos Aplicación, Red, Nodo y el módulo compuesto que define la red.

2.5 Enlaces

La capacidad en bps, así como el retardo de propagación de cada enlace debería ser un parámetro del módulo compuesto y por tanto configurable en omnetpp.ini. La asignación de estas capacidades a los enlaces se llevará a cabo en la definición en NED

2.6 Estadísticos

Otro de los requisitos fundamentales es que el simulador genere estadísticos sobre ciertos parámetros de interés. El simulador deberá calcular al menos los siguientes estadísticos:

- Número medio de paquetes en la cola de salida de cada enlace, para cada nodo.
- Retardo medio extremo a extremo de los paquetes enviados.

En el apartado 4 se describirán en detalle las medidas que debe realizar e incluir en la memoria.

3 Sugerencias para la implementación

3.1 handleMessage

Aunque OMNET++ permite utilizar las funciones activity() y handleMessage() para implementar los módulos simples, es conveniente utilizar la segunda opción. Las ventajas de emplear una u otra solución están claramente desarrolladas en el apartado 4.4 del manual de omnet++ versión 3.0. En caso de que se decida emplear handleMessage es necesario recordar los siguientes detalles:

- Las variables del módulo deben definirse necesariamente como atributos del módulo, a no ser que su valor se reinicie en cada evento (por ejemplo iteradores de bucles).
- Con handleMessage no se puede hacer uso de las funciones receive( ) y wait( ). Los timeouts deberán implementarse como automensajes, con ScheduleAt( ).
- Cuando se hace uso de handleMessage es necesario inicializar las variables en initialize( ), donde además se deberá enviar el primer mensaje.
- Con handleMesage es habitual tener como atributos del módulo punteros a mensajes que se inicializan una sola vez y que se van reutilizando, generalmente automensajes de timeout.

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, por ejemplo *encaminar()*, *procesarLSP()* o *procesarPaquete()*. Si determinadas líneas de código se repiten en distintas partes del código es posible que puedan implementarse como una sola función. Por ejemplo una función del tipo *buscarIdEnListaLSP()*.

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. Comentarios del tipo: "*leemos parámetros*" que pueden estar asociados a más de una línea de código son suficientes. También se pueden identificar con comentarios determinados momentos en el algoritmo, por ejemplo "*No está en ninguna lista, inserto en Provisional*". En cualquier caso el criterio para comentar el código debe definirlo el programador, pensando en qué indicaciones le harían falta si necesitase revisar ese código mucho tiempo después de haberlo escrito. Recuerde que un código excesivamente comentado puede producir el efecto contrario, es decir, dificultar su lectura y comprensión

3.3 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. Recuerde que la documentación de las STL se encuentra en la siguiente dirección:

http://www.sgi.com/tech/stl/

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)
                formas.erase(it); //Sacamos el elemento de la lista.
        }
        it++;
}
```

Los contenedores asociativos se utilizan para almacenar pares valor-clave. Por ejemplo:

```
#include <map.h>
#include <string.h>
map<string,int> meses;
meses.insert(make_pair("enero",1);
meses.insert(make_pair("febrero",2);
cout <<"febrero se corresponde con el mes "<<meses["febrero"]<<endl;
```

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). Observe los siguientes ejemplos:

```
vector<queue<Objeto*> > colasObjetos; //Atención al espacio entre ">". ¡¡¡Es obligatorio!!!!
std::queue<Objeto*> c1;
std::queue<Objeto*> c2;

colasObjetos.push_back(c1);
colasObjetos.push_back(c2);
for (int i=0;i<colasObjetos.size();i++){
        colasObjetos[i].push(new Objeto()); //Observe que con queue se utiliza push, no push_back
}
colasObjetos[0]->ejecutaFuncion();
```

Recuerde que está implementando nodos con varios enlaces, así que deberá utilizar una cola de salida para cada enlace, donde almacenará los paquetes en espera de ser enviados. Las STL son una buena opción para implementar este comportamiento, combinando vectores y colas, por ejemplo.

3.4 Fases del desarrollo

El desarrollo de software es un proceso incremental e iterativo, es decir, se comienza el programa implementando cierta funcionalidad básica, se verifica y se continúa añadiendo funcionalidad. Intentar resolver un problema complejo de golpe suele ser una mala idea que provoca muchas dificultades en la depuración. Nuestro consejo a la hora de implementar el simulador es que siga los siguientes pasos:

1. Defina y declare la estructura del simulador en NED.
2. Implemente el módulo Aplicación.
3. Implemente el módulo Red con encaminamiento estático. El objetivo es asegurarse que el módulo Red encola y encamina correctamente los paquetes sin mezclarlo con el cálculo de las rutas. Pruebe el módulo con tráfico determinista. El módulo red debe utilizar una cola infinita para almacenar los paquetes en espera de ser enviados
4. Implemente la creación e inundación de la red con LSP. Compruebe que la información de los LSP es correcta.
5. Implemente el cálculo de rutas mediante el algoritmo "forward search" una vez que tiene los LSP. Muestre por pantalla los pasos del algoritmo para depurarlo.
6. Compruebe que los paquetes siguen la ruta adecuada, es decir, funciona el encaminamiento y reenvío.

3.5 Utilidades de OMNET++

En cualquier lenguaje de programación se trabaja con librerías. El programador no tiene que conocer todas las funciones, pero debe ser capaz de encontrar las que necesite en cada momento. OMNET++ no es distinto: debe consultar el API y el manual para poder implementar el simulador. Por ejemplo, si necesita saber cuándo termina la transmisión de un mensaje, en el manual encontrará que la función *transmissionFinishes()* de cGate le proporciona esa información.

Si tiene que crear mensajes derivados de cMessage o clases nuevas, aproveche las ventajas de opp_msgc. Este script tiene dos funciones:

1. Proporcionar el código de la interfaz de las clases derivadas, es decir, métodos get y set para cada campo.

2. Generar código adicional que se utiliza para explorar los campos de la clase o mensaje mediante la interfaz gráfica de OMNET++ en tiempo de ejecución. Esto es muy útil a la hora de depurar su programa.

3.6 Consejos para la depuración, verificación y validación del módulo

La corrección de un simulador se comprueba en dos fases: verificación y validación. La primera consiste en asegurarse que el modelo está bien implementado, es decir, que no hay errores de programación. La segunda consiste en asegurar que el simulador reproduce el comportamiento del sistema real (con las simplificaciones que se haya asumido previamente). La primera fase se suele llamar también depuración y es una tarea ardua en muchos casos, ya que la única forma que hay de hacerla es seguir el código línea a línea y comprobar que es correcto. Para la validación se comparan los resultados del simulador con resultados conocidos o esperados. En el apartado 4 se retomará la cuestión de la validación del simulador, es parte de la memoria que debe entregar.

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, o cambiarse en cada compilación. 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 módulo Red “<<miID <<” ha recibido un paquete”<<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.

3.7 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.

4 Medidas a Realizar

En este apartado se describen las medidas que deberá realizar e incluir en la memoria. Dedique una sección de la memoria a los resultados de cada uno de los siguientes apartados, excepto el 4.1. Incluya los comentarios e interpretaciones que considere oportunos.

4.1 Selección de la semilla

En prácticas anteriores se estudió que la elección incorrecta de semillas, da lugar a correlación en los resultados. Se debe asegurar que las secuencias de números aleatorios utilizadas para generar variables aleatorias independientes no se solapan.
OMNET++ utiliza por defecto un generador llamado Mersenne Twister RNG. Este generador tiene la propiedad de que su ciclo es de 2^{19937}-1. Esto es un ciclo infinito, en la práctica. Esta propiedad da lugar a otra: la probabilidad de elegir dos semillas cualesquiera que den lugar a secuencias solapadas es ínfima. No son necesarias precauciones para elegir la semilla como con otros generadores de ciclo más corto, en definitiva. Además, OMNET++ permite asignar mediante el fichero de configuración diferentes generadores a diferentes módulos. Esto se llama “asignación de RNG” (RNG mapping, sección 8.6.3 del manual).
Para todas las simulaciones que realice:

1. Establezca Mersenne Twister como RNG. En omnetpp.ini, rng-class=”cMersenneTwister”.
2. Utilice 4 generadores (0..3) cuyas respectivas semillas serán: 5454,78823,83839,29299.
3. Asigne a cada nodo un generador (nodo 0-generador 0, etc.)

4.2 Validación del simulador

Es imprescindible al realizar una simulación asegurar que se está representando correctamente el modelo simulado. Para ello se deben comparar los resultados obtenidos con un modelo analítico o con resultados experimentales.
En este apartado debe realizar las siguientes tareas:

1. Seleccione un modelo analítico con el que pueda comparar su simulador. Justifíquelo.
2. Defina un experimento que valide su simulador.
3. Realice el experimento y valide su simulador. Justifique los resultados.

Como consejo: para validar se utilizan modelos analíticos sencillos, de los que se conoce su análisis matemático, como colas M/M/1. Piense en qué experimentos su simulador se comportaría como uno de estos sistemas y reprodúzcalo.

4.3 Experimento 1. Ajuste de tasas de las fuentes de tráfico

Configure su simulador con los siguientes parámetros:

Enlace	Velocidad de transmisión (Kbps)
0-1	20
0-2	10
1-2	33
1-3	9
2-3	50

Nodo	Media tiempo entre llegadas	Nodo destino
0	0.5	3
1	0.33	3
2	0.5	1
3	0.33	0

Para todos los nodos:

- Ancho de banda de referencia: 100000
- Tamaño de paquete fijo: 8000 bits

Realice las siguientes tareas.

1. Ejecute la simulación durante 500 minutos. Rellene una tabla con el número medio de paquetes en cada cola para cada nodo y el retardo medio extremo a extremo para cada destino. Comprobará que el sistema está congestionado. Con el modelo utilizado para la validación del apartado 4.2 intente explicar este comportamiento.
2. Busque tiempos medios entre llegadas para las fuentes que hagan que el sistema sea estable y haya una utilización razonable de los recursos, es decir, no estén infrautilizados ni saturados. Rellene una tabla para 3 conjuntos de valores para las tasas, con el número medio de paquetes en cada cola para cada nodo y el retardo medio extremo a extremo para cada destino.

4.4 Experimento 2. Variación del tipo de tráfico

Configure su simulador con los siguientes parámetros:

Enlace	Velocidad de transmisión (Kbps)
0-1	20
0-2	10
1-2	33
1-3	9
2-3	50

Nodo	Media tiempo entre llegadas	Nodo destino
0	0.5	3
1	0.5	3
2	0.5	1
3	0.5	0

Para todos los nodos:

- Ancho de banda de referencia: 100000
- Tamaño de paquete fijo: 8000 bits

Realice las siguientes tareas.

1. Ejecute la simulación durante 500 minutos. Rellene una tabla con el número medio de paquetes en cada cola para cada nodo y el retardo medio extremo a extremo para cada destino. Incluya una gráfica para cada nodo del histograma del retardo de los paquetes.
2. Configure la simulación para que el tamaño de los paquetes siga una función exponencial de media 8000 bits. Ejecute la simulación durante 500 minutos. Rellene una tabla con el número medio de paquetes en cada cola para cada nodo y el retardo medio extremo a extremo para cada destino. Incluya una gráfica para cada nodo del histograma del retardo de los paquetes.
3. Interprete los resultados. Indique si es apropiada la métrica que está utilizando para cualquier tipo de tráfico. Justifique por qué es necesario utilizar métricas dinámicas y actualizar las rutas incluso aunque no cambie la topología.

5 Método 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. El código y el ejecutable se dejará en un directorio llamado *evaluacionTrabajo* dentro de la cuenta de un componente del grupo. Se especificará claramente en la memoria el número de cuenta en el que se deposita el código. La entrega y evaluación se realizará en Junio. Se avisará con suficiente antelación de la fecha límite de entrega. **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.**

5.1 Memoria

La memoria constará de los siguientes apartados:

1) Introducción: En este apartado describirá la estructura general del programa realizado, y podrá apoyarse en un diagrama de bloques o una implementación en pseudocódigo muy general. Se indicarán también las funciones de que consta, y qué variables se han definido.
2) Código de los módulos. Deberá presentarlo función a función, *eliminando los comentarios entre las líneas de código así como las líneas de código que haya insertado para su depuración*. La explicación de cada función deberá darla al inicio ó al final del código de la misma, y si desea comentar una línea de código en concreto deberá referenciarla con una nota de este tipo: **[1] [2] [3]**, etc.
3) Código NED implementado.
4) Resultados de las medidas, tablas y gráficas obtenidas. Interpretación de los resultados.

5.2 Criterios de evaluación

En este apartado se enumeran los aspectos que se tendrán en cuenta en la evaluación del trabajo.

Se valorará que el simulador compile y funcione correctamente con las distintas configuraciones que se planteen.

Se valorará que se haya implementado y que funcione correctamente la toma de medidas. También se valorará que los alumnos sepan interpretar los resultados.

La memoria se evaluará de manera general en cuanto a su claridad y rigor. Si se han implementado medidas, están implementadas correctamente, y funcionan, se valorarán los resultados de las mismas presentados en la memoria y su interpretación.

Se valorará la corrección en la implementación de los distintos apartados:

- Algoritmos que se describen en esta propuesta.
- Implementación en NED del simulador.
- Medidas.

Se valorará la claridad del código. En general cuantas menos líneas de código se empleen (sin quitar funcionalidades) la implementación será mejor y más clara.

Se valorará que los módulos no almacenen paquetes innecesariamente (gestión óptima de memoria).

6 Bibliografía:

[1] L. Peterson, B. Davie, “Computer networks. A systems approach”, Morgan Kaufmann, 2003
[2] H. Deitel, “C++, Como programar”, Prentice Hall, 1999.