Universidad Politécnica de Cartagena

Escuela Técnica Superior de Ingeniería de Telecomunicación

PRÁCTICAS DE REDES DE ORDENADORES

Propuesta del Trabajo de Prácticas 2009

Simulación de algoritmos de scheduling en redes inalámbricas

Profesores:

Esteban Egea López
Juan José Alcaraz Espín
Juan A. Veiga Gontán
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 inalámbrica sencilla en la que un nodo (estación base o punto de acceso) transmite paquetes a una serie de terminales móviles por un canal radio. La transmisión de datos se realiza únicamente en sentido *downlink* (del nodo a los terminales). Los paquetes transmitidos podrán no ser recibidos correctamente en el receptor en función de las condiciones del enlace radio del terminal. Los módulos que implementan la evolución del canal radio se proporcionan a los alumnos. En este entorno, los alumnos programarán y evaluarán el rendimiento de tres sencillas disciplinas de servicio (algoritmos de *scheduling*). Finalmente, se propone un ejercicio adicional que permite obtener la calificación más alta. Además, los alumnos que estén especialmente interesados podrán proponer y evaluar sus propios algoritmos y también investigar en más detalle los factores que influyen en el rendimiento. El impacto de estas contribuciones en la calificación obtenida dependerá de su originalidad y de su calidad (no de la cantidad).

1.2 Qué deben hacer los alumnos

Los alumnos deben implementar tres módulos simples en C++ y NED: Fuente, Nodo y Terminal. De manera breve, la funcionalidad de estos módulos es la siguiente: El módulo fuente genera paquetes de tamaño fijo para cada uno de los usuarios y los envía al Nodo. El módulo Nodo almacena los paquetes de cada usuario en su cola correspondiente a la espera de ser transmitidos. El Nodo transmite por el canal radio un solo paquete en cada Intervalo de Transmisión (IT). La cola de la que se extrae el paquete en cada IT la escoge el algoritmo de *scheduling*. Finalmente, el módulo terminal se encarga de determinar si los paquetes se reciben con o sin errores, en función del estado de su enlace radio, definido por su ganancia en dB. El módulo Terminal está asociado a otros dos módulos, que ya están implementados, y que permiten al Terminal conocer el estado de su enlace radio en cada momento.

En este entorno, el alumno deberá programar tres algoritmos de *scheduling* que se evaluarán tomando una serie de medidas que quedarán recogidas en una memoria final junto con el código comentado. 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. De igual forma, los alumnos que lo deseen podrán ir más allá de lo planteado en esta propuesta y desarrollar y probar sus propias ideas.

1.3 Qué código se proporciona

Como se ha comentado anteriormente, hay dos módulos que se encargan de generar el estado del canal (ganancia en dB) de acuerdo a unos modelos de movimiento, propagación y desvanecimiento tipo *Rayleigh*. El código de estos módulos, denominados “ms” y “channelStateCalc” está disponible íntegramente en la página de la asignatura (Aula Virtual). El primero actualiza la posición del usuario y el segundo calcula la ganancia del canal. Para comunicar el módulo Terminal con estos módulos haremos uso un tipo de mensaje denominado “channelStateMsg”, que también se proporciona.

1.4 Cómo está organizada esta propuesta.

En el siguiente apartado se describe en detalle el simulador a implementar. En el apartado 3 se explican las medidas a realizar. En el apartado 5 se especifica la memoria a entregar y los criterios de evaluación. El apartado 4 se dan algunos consejos y sugerencias para ayudar en el desarrollo del trabajo. Finalmente, en el apartado 5 se proporciona una breve bibliografía.

2 Descripción del simulador

2.1 Estructura general del simulador

Como se ha comentado en el apartado 1, el simulador se compone de un módulo Fuente, que genera los paquetes de cada usuario de acuerdo a su tasa de transmisión, un módulo Nodo, que recibe los paquetes de la Fuente y un número de módulos Terminal, que reciben los paquetes procedentes del Nodo. Cada módulo Terminal está asociado a un módulo *ms* y a un módulo *channelStateCalc*. En la siguiente figura se muestra un ejemplo de un simulador con tres Terminales.

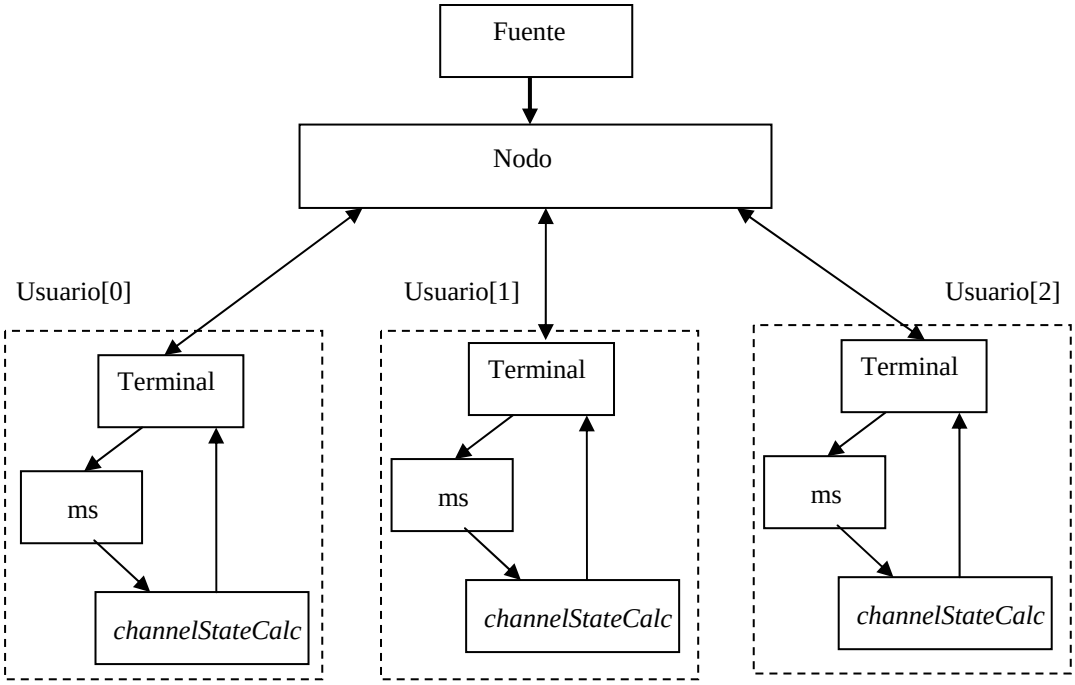

Figura 1. Ejemplo de topología del simulador

El conjunto formado por los módulos Terminal, ms y *channelStateCalc* se ha enmarcado en un rectángulo punteado, sugiriendo que se agrupen en un módulo compuesto, (denominémoslo Usuario), para una mayor claridad en la implementación. En la definición del módulo compuesto deberá generar un *array* de módulos Usuario. Como puede observar, el módulo Nodo deberá disponer de un *array* de puertas de salida y de un *array* de puertas de entrada, y además una puerta de entrada para los paquetes procedentes de la fuente. Si le resulta más sencillo, puede implementar un *array* de módulos Fuente en vez de un solo módulo Fuente.

2.2 Descripción de los módulos a desarrollar

En este apartado se describirán las funcionalidades y los parámetros que deben contener, **como mínimo**, los módulos a desarrollar.

2.2.1 Fuente

El módulo fuente generará, para cada usuario, paquetes de tamaño fijo. Los paquetes deberán indicar el usuario al que pertenecen. El tiempo entre la generación de dos paquetes consecutivos para un usuario estará determinado por una variable aleatoria exponencial. Se deben incluir como parámetros:

- Tamaño del paquete.
- Tiempo medio entre llegadas ó tasa de generación de paquetes[1].

[1] El ejercicio final de este trabajo implica que cada usuario tenga una tasa distinta. Si ha implementado un solo módulo Fuente puede necesitar emular un *array* de parámetros. Consulte el apartado 4.7.1 del manual de OMNET++.

2.2.2 Nodo

El módulo Nodo contendrá un vector de colas, con una cola por cada Terminal en el sistema[2]. Cada vez que llegue un paquete, éste se guardará en la cola correspondiente al Terminal indicado en el paquete. El Nodo debe generar intervalos de transmisión (IT) de duración constante. En cada IT, el Nodo transmitirá un solo paquete. El algoritmo de *scheduling* decide de qué cola se debe extraer el paquete a transmitir. Un paquete extraído de la cola ***k*** se enviará por la puerta de salida ***k***. Antes de transmitirlo, el paquete se marcará con la potencia de transmisión del Nodo. Finalmente, cada vez que se recibe un paquete de reconocimiento (ACK) procedente de un Terminal, se borrará un paquete de la cola correspondiente a dicho Terminal.

El módulo Nodo debe contener, al menos, los siguientes parámetros:

- tipoScheduler: Permite seleccionar qué scheduler utilizará el nodo.
- iTransmision: Determina el tiempo entre intervalos de transmisión.
- potenciaTransmision: Potencia con la que se transmiten los paquetes, en dBW.

2.2.3 Terminal

El módulo Terminal debe actualizar periódicamente el estado de su enlace radio (su ganancia en dB). Para ello debe generar un mensaje "channelStateMsg" y enviarlo al módulo "ms". Éste lo mandará al módulo "channelStateCalc" que lo devolverá al Terminal con el valor de la ganancia de su enlace radio.

También periódicamente, el Terminal debe enviar un mensaje hacia el Nodo indicando el valor de la ganancia de su enlace radio. Este tipo de información se denomina CQI (Channel Quality Information) y es muy habitual en redes inalámbricas.

En función de la potencia de transmisión, la ganancia del enlace radio, el factor de ruido y el ancho de banda, el Terminal calculará la probabilidad de error de bit (BER) de cada paquete recibido. A partir del BER y de la longitud del paquete, calculará la probabilidad de error del paquete y estimará si el paquete ha llegado con o sin errores. Si el paquete ha llegado sin errores, enviará un paquete de ACK al Nodo.

El módulo Terminal debe contener, al menos, los siguientes parámetros:

- figuraDeRuido: figura de ruido del receptor, en dB.
- anchoDeBanda: ancho de banda, en Hz, del canal radio.
- iCQI: intervalo de envío del CQI.
- iActualizacion: intervalo de actualización del estado del enlace radio.
- miDireccion: número que identifica al Terminal (0, 1, 2…).

2.2.3.1 CÓMO ACTUALIZAR EL ESTADO DEL CANAL RADIO

Cada vez que se produzca un evento correspondiente a un intervalo de actualización del estado del canal, el módulo Terminal deberá enviar un mensaje "channelStateMsg" al módulo ms. Para ello deberá crear un mensaje de este tipo e inicializarlo:

```
channelStateMsg *msgout = new channelStateMsg(…)
msgout->setChannelStateArraySize(1);
```

Cuando llegue un mensaje del tipo "channelStateMsg", el módulo Terminal deberá obtener el valor de la ganancia del canal y actualizar la variable correspondiente:

```
channelStateMsg *channelState = (channelStateMsg*)msg;
gananciaCanal = channelState->getChannelState(0);
```

2.2.3.2 CÓMO CALCULAR LA PROBABILIDAD DE ERROR DE UN PAQUETE

La probabilidad de error se calculará asumiendo que el paquete se ha transmitido por el canal radio mediante una modulación BPSK. En esta modulación la probabilidad de error de bit (BER) se obtiene de la siguiente forma:

```
double BER = 0.5*erfc(sqrt(Eb/No));
```

[2] En varias prácticas encontrará módulos con *arrays* de colas, puertas, etc. Por ejemplo en la práctica 10. De hecho, gran parte del código de dicha práctica se puede reutilizar en este trabajo.

Donde E_b es la energía por bit y N_0 es la densidad espectral de ruido. E_0 se obtiene a partir de la potencia recibida (en Watios) y el ancho de banda (en Hertzios):

```
double Eb = Prx/(2*AB);
```

Donde AB es el ancho de banda del canal de transmisión. N_0 se obtiene a partir de la constante de Botzmann (K_B), del factor de ruido (f) y de la temperatura de referencia (t_0 = 290 ºK):

```
No = f*To*K_B;
```

Una vez obtenido el BER, podemos obtener la probabilidad de error de un paquete (P) a partir de su longitud en bits:

```
double P = 1-pow((1-BER), bits);
```

Para saber si un paquete ha llegado con error o no, generaremos una variable aleatoria distribuida uniformemente entre 0 y 1 y compararemos su valor con P. Si la variable generada es menor que P, el paquete ha llegado con error.

2.3 Descripción de los módulos proporcionados

Los módulos proporcionados son parte de un código libre para OMNeT++, denominado Chsim, desarrollado en la Universidad de Paderborn (Alemania) y ligeramente modificado para su uso en este trabajo de prácticas.

2.3.1 Modulo de movilidad

Su función es recibir el mensaje "`channelStateMsg`" procedente del Terminal y enviarlo al módulo "channelStateCal", indicando en dicho mensaje la distancia del enlace de transmisión. Como ya se ha comentado, este módulo emula el movimiento del terminal, actualizando su posición dentro de un área rectangular definida por el usuario. En esta cuadrícula se emplea un sencillo sistema de coordenadas:

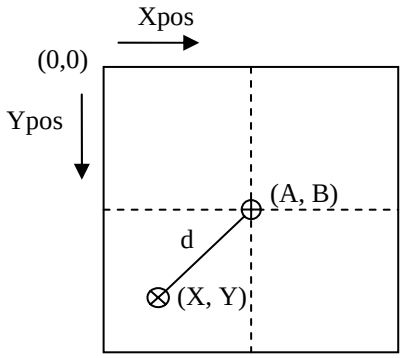

Figura 2. Sistema de coordenadas del módulo de movilidad

Como vemos, el origen de coordenadas se encuentra en la esquina superior izquierda. Se asume que el transmisor (Nodo) se encuentra en el punto central (A, B), por lo que la longitud del enlace radio (d) entre el Nodo y el Terminal es igual a la distancia entre las coordenadas del terminal (X, Y) y el punto central (A, B).

Existen tres patrones de movimiento (`mobilityModel`), del cual seleccionaremos el primero (0), que corresponde a un movimiento circular respecto al punto central, con lo que la distancia respecto al punto central permanecerá constante. Recuerde que la distancia determina las pérdidas por propagación, y por tanto éstas serán constantes. La variabilidad de las pérdidas estarán determinadas por el modelo de *fading* tipo *Rayleigh*.

A continuación se explican algunos de los parámetros del módulo "ms":

- playgroundSizeX: anchura del area rectangular.
- playgroundSizeY: altura del area rectangual.
- initialXpos: coordenada X inicial del terminal.
- initialYpos: coordenada X inicial del terminal.
- mobilityModel: modelo de movilidad.
- speed: velocidad del terminal.

El módulo "ms" tiene una puerta de entrada llamada "in" y una puerta de salida llamada "out".

2.3.2 Modulo de estado del canal

Su función es recibir el mensaje "`channelStateMsg`" procedente del módulo "ms", calcular la ganancia del canal radio a partir de la distancia indicada y otros parámetros, y devolver el mensaje con la ganancia. Este módulo también tiene una puerta de entrada llamada "in" y una puerta de salida llamada "out".

2.4 Descripción de los algoritmos de *scheduling*

Se deben implementar tres sencillos algoritmos de *scheduling*:

- **Round Robin**: Las colas se van sirviendo siempre en el mismo orden (1,2,3,4,1,2,3,4…). Si en un IT, la cola con el turno de transmisión está vacía, se pasa a la siguiente, y así hasta encontrar una cola no vacía.
- **Mayor número de paquetes**: Se sirve la cola con un mayor número de paquetes.
- **Mejor enlace radio**: Se sirve la cola del usuario con una mayor ganancia en el enlace radio. Si la cola está vacía, se elige el mejor de los restantes enlaces radio, y así hasta encontrar una cola no vacía. Para implementar este algoritmo, el Nodo debe contener un vector o *array* con los valores de la ganancia de cada enlace. Este vector se actualiza cada vez que llegue un CQI o un ACK.

3 Medidas a realizar

3.1.1 Configuración de parámetros

En todos los ejercicios se emplearán los siguientes valores de configuración en los parámetros de cada módulo:

Módulo channelStateCalc:
calculatePathLoss = 1
calculateShadowing = 0
calculateFading = 1
correlated_subbands = 0
use_linear_scale = 0
lightspeed = 299792458
alpha = 2.4
tenlogk = -46.7
mean = 0
std_dev = 5.8
fadingPaths = 20
center_frequency = 5200000000
freq_spacing = 500000
subbands = 1
delay_rms = 0.00000015

Módulo ms:
playgroundSizeX = 1000
playgroundSizeY = 1000
gridSpacing = 10
speed = 1
mobilityModel = 0
waitTime = 0

Módulo Fuente:
Longitud: 1000 (paquetes de 1000 bits)

Módulo Nodo:
potenciaTransmision = -10
iTransmision = 2ms

Módulo Terminal:
anchoDeBanda = 500000
figuraDeRuido = 10
iCQI = 4ms
iActualizacion = 2ms

3.1.2 Ejercicio 1: Medidas de validación.

El escenario de validación consiste en un sistema con **un único usuario** (el *scheduling* no es necesario). Para comprobar el funcionamiento de algunas funcionalidades básicas del simulador se generarán las siguientes **trazas** de funcionamiento en este escenario:

- Traza del **estado del canal** frente al tiempo.
- Probabilidad de error de bit (**BER**) frente al tiempo.
- Número de **paquetes reconocidos** frente al tiempo.

Además se obtendrán las siguientes **medidas**:

- Tasa de **paquetes perdidos**.
- **Throughput**: paquetes reconocidos/tiempo de simulación.
- **Máxima ocupación** de la cola.

Estas trazas se obtendrán para un tiempo entre llegadas de 3 ms y para dos posiciones del Terminal:

1) Posición alejada del Nodo: `initialXpos = 500, initialYpos = 1`
2) Posición cercana al Nodo: `initialXpos = 500, initialYpos = 450`

3.1.3 Ejercicio 2: Medidas de análisis de rendimiento.

El escenario para el ejercicio 2 consiste en un Nodo conectado a cuatro usuarios con distintas condiciones de propagación. Las posiciones iniciales de cada terminal son las siguientes:

- Terminal 1: `initialXpos = 500, initialYpos = 1`
- Terminal 2: `initialXpos = 500, initialYpos = 100`
- Terminal 3: `initialXpos = 500, initialYpos = 300`
- Terminal 4: `initialXpos = 500, initialYpos = 450`

Todos los terminales tendrán la misma tasa de llegadas de paquetes.

Para cada algoritmo de *scheduling* implementado se tomarán una serie de medidas variando progresivamente la intensidad de tráfico, desde valores de baja intensidad (20 ms entre llegadas) a valores de gran intensidad (4 ms entre llegadas). Con estas medidas se realizarán las siguientes **gráficas de análisis de rendimiento**:

- **Throughput** de la estación base (paquetes reconocidos/tiempo de simulación) frente a intensidad de tráfico.
- **Número medio de paquetes almacenados** en la estación base frente a intensidad de tráfico.
- Longitud de la cola que haya alcanzado la máxima ocupación (**máxima cola**) frente a intensidad de tráfico.
- **Factor de justicia** (*Fairness Index*) frente a intensidad de tráfico.

Es conveniente que en cada gráfica se representen los resultados de cada *scheduler* implementado.

3.1.3.1 DEFINICIÓN DEL ÍNDICE DE JUSTICIA (FAIRNESS INDEX)

Existen muchas definiciones y por tanto muchos posibles índices. Nosotros vamos a emplear el que se define como la **máxima diferencia entre los *throughputs* normalizados (T_n) de dos usuarios del sistema**. El *throughput* normalizado es el cociente entre el *throughput* ofrecido a un usuario y el *throughput* que el sistema se compromete a ofrecer a dicho usuario. En nuestro caso, asumimos que el *throughput* comprometido es igual a la tasa de llegadas de paquetes del usuario.

3.1.4 Ejercicio 3 (Extra): Cálculo de la región de capacidad.

La región de capacidad de una red se define como el conjunto de tasas de entrada de datos para las cuales la red se mantiene estable. Se dice que la red se mantiene estable siempre que la longitud de ninguna de sus colas crezca indefinidamente. En nuestro caso, definiremos como criterio de inestabilidad que alguna cola supere un umbral máximo durante el tiempo de simulación. Determine este umbral a partir de los resultados del Ejercicio 1.

El ejercicio propuesto en este apartado consiste en determinar, para varios escenarios, la región de capacidad de la red mediante simulación. En todos los escenarios el número de terminales será 2. Por tanto, el resultado de este ejercicio será una serie de gráficas de este tipo:

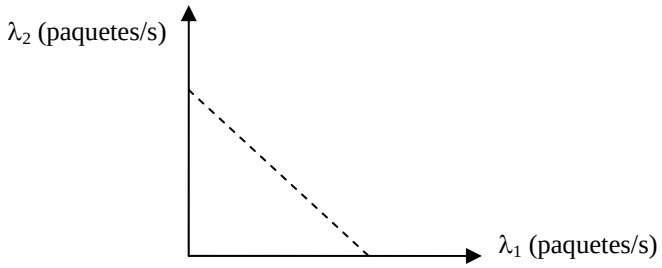

Figura 3. Ejemplo de representación de la región de capacidad

Donde cada eje corresponde a la intensidad de tráfico de cada usuario. La región de capacidad viene dada por el área delimitada por los ejes de coordenadas y la línea punteada. Los puntos de la línea punteada también son parte de la región de capacidad.

Se deben obtener las siguientes gráficas:

- Gráfica 1: Regiones de capacidad de cada algoritmo para dos terminales situados a 499 m del Nodo.
- Gráfica 2: Regiones de capacidad de cada algoritmo para dos terminales situados a 50 m del Nodo.
- Grafica 3: Regiones de capacidad de un algoritmo cualquiera para terminales a 50 m y a 499 m del Nodo.
- Gráfica 4: Regiones de capacidad de cada algoritmo para un terminal a 499 m y otro a 50 m.

El cálculo de las regiones de capacidad mediante simulación requiere un coste computacional elevado, sobre todo teniendo en cuenta que deberá realizar varias réplicas para determinar si cada punto es o no es estable. En la realización de este ejercicio se valorará la eficiencia y la precisión de la estrategia empleada para obtener los puntos de la gráfica.

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 cada convocatoria de exámenes, hasta Febrero de 2010, que será la última convocatoria en la que se podrá entregar este trabajo. Se avisará con suficiente antelación de la fecha límite de entrega, que será una semana o dos 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 de contenidos
- Código implementado comentado. Deberá contener, al menos, un subapartado por cada algoritmo implementado
- Por cada escenario del que se obtengan resultados, se especificará el número de réplicas o la precisión con la que se han obtenido los valores (excepto en el caso de trazas de validación), el grado de confianza, las gráficas (con sus intervalos de confianza, si procede) y una interpretación de los resultados.

El código se entregará en un CD o diskette, 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 (cuando lo realizan dos personas) se ha repartido en los distintos ejercicios propuestos. La puntuación total de cada ejercicio se muestra en la siguiente tabla:

Apartado	Puntuación máxima
Ejercicio 1	2
Ejercicio 2	6 (*)
Ejercicio 3	2

(*) Cada algoritmo puntúa por igual en este apartado.

En general 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 y el número de réplicas o bien la precisión con la que se han obtenido los intervalos.
- La interpretación dada de los resultados y se apreciará especialmente que los alumnos demuestren haber consultado bibliografía adicional, Internet, etc, para mejorar sus conclusiones.
- 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 y la velocidad de ejecución.
- La claridad, rigor y concisión en las explicaciones aportadas en la memoria, así como la claridad en la presentación de gráficas y resultados.

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

Como se dijo en la introducción, s**e valorarán propuestas de otros algoritmos y el estudio de la influencia de distintos factores del entorno en el rendimiento**. Las aportaciones se valorarán por su originalidad, su calidad, y su argumentación, **no por su cantidad ni su extensión**. Estas contribuciones pueden ser útiles para mejorar la **nota final de prácticas** de la asignatura.

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 o 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
Ejercicio 1	3
Ejercicio 2	6 (*)
Ejercicio 3	1

(*) Cada algoritmo puntúa por igual en este apartado.

Grupos de tres personas:

Apartado	Puntuación máxima
Ejercicio 1	1
Ejercicio 2	6 (*)
Ejercicio 3	3

(*) Cada algoritmo puntúa por igual en este apartado.

5 Sugerencias para la implementación

5.1 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 (puede ser un parámetro del módulo).

A lo largo del código se insertan líneas del tipo:

if (debug) ev << "El BER obtenido es = "<<BER<<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 empleé un fichero debe cerrarlo.

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

5.2 Gestión eficiente de la memoria

Es conveniente que se eliminen los mensajes recibidos en los módulos cuando los mensajes no vayan a emplearse más, para evitar que 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, si borra un mensaje que posteriormente se planifica ó ya está planificado.

5.3 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 "intervalo", vuelva a reenviárselo, 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.

6 Bibliografía

[1] Andras Varga, "OMNET++ discrete Event Simulation System, Versión 3.2. User Manual."
[2] Chsim web page: http://wwwcs.upb.de/cs/chsim
[3] D. Tse y P. Viswanath, "Fundamentals of Wireless Communications" Cambridge University Press, 2005.
[4] John Proakis, "Digital Communications, 4th Edition", Mc Graw Hill, 2001.
[5] H. Fattah y Cyril Leung, "An Overview of Scheduling Algorithms in Wireless Multimedia Networks", IEEE Wireless Communications, Octubre 2002.