Universidad Politécnica de Cartagena

Escuela Técnica Superior de Ingeniería de Telecomunicación

PRÁCTICAS DE REDES DE ORDENADORES

Propuesta del Trabajo de Prácticas 2010

Simulación de protocolos anticolisión para un sistema de identificación por radiofrecuencia (RFID)

Profesores:
Esteban Egea López
Juan José Alcaraz Espín
Joan García Haro

Índice.

1 Consideraciones generales.

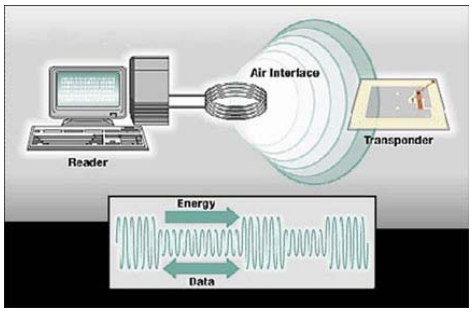
1.1 Objetivos

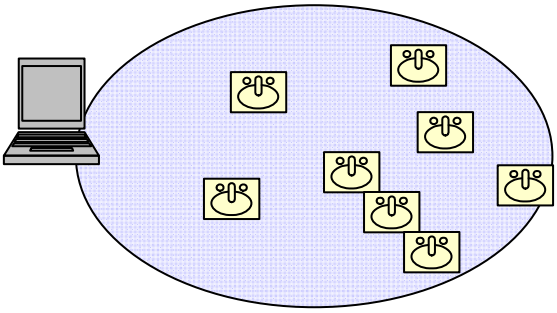
En este trabajo de prácticas los alumnos realizarán por parejas un simulador de protocolos anti-colisión o de control de acceso al medio (MAC) para sistemas de identificación por radiofrecuencia. Los alumnos deben desarrollar y depurar el simulador y extraer y analizar una serie de resultados. Tanto los algoritmos como los resultados a presentar se detallan en esta memoria.

1.2 Introducción

1.2.1 Sistemas de identificación por radiofrecuencia

Los sistemas de identificación por radiofrecuencia consisten en acoplar a los objetos que se desea identificar un dispositivo electrónico que responderá a una señal radio emitida por un dispositivo lector o interrogador.
Las aplicaciones de los sistemas RFID abarcan desde sistemas antirrobo, sustitución de sistemas de códigos de barras hasta la localización/detección de contenedores en grandes vehículos de transporte (buques, trenes). Hay numerosas propuestas de tecnologías RFID con el fin de cubrir tan extensa variedad de aplicaciones. Un sistema RFID básico está compuesto de un antena lectora, llamada *Reader* o *master* y una serie de etiquetas de radiofrecuencia (RFID *tags*), posiblemente muy grande, que responden a las peticiones u órdenes del *Reader*. Los *tags* RFID pueden clasificarse según su fuente de energía: los *tags* pasivos son dispositivos simples, normalmente con una memoria de solo lectura, que carecen de fuente de energía de alimentación propia y son alimentados por parte del *Reader* por inducción electromagnética. El rango de cobertura puede oscilar desde los pocos centímetros hasta un par de metros. La simplicidad de estos dispositivos hace que sean muy económicos. Por el contrario, los *tags* activos son dispositivos de mayor complejidad, incorporan su propia batería y normalmente incluyen un microprocesador y una memoria de lectura/escritura, por lo que pueden realizar operaciones más complejas. El rango de cobertura de estos dispositivos es mucho mayor, llegando a alcanzar distancias de hasta 100 metros [1]. Por supuesto, el coste *hardware* de estos dispositivos es muy superior en comparación con los *tags* pasivos.





En ambos casos surge el problema de las colisiones en el acceso al medio. Es decir, si varios *tags* reciben simultáneamente una petición de identificación de un *Reader*, los mensajes de respuesta de éstos colisionarán, impidiendo la correcta identificación. Por ello es necesario incluir algún tipo de mecanismo anticolisión. Dicho mecanismo debe garantizar la correcta y rápida identificación de *todos* los *tags* que circulen por la zona de cobertura. Asimismo, en el caso de los *tags* activos es necesario implementar un mecanismo de ahorro de energía para maximizar la duración de las baterías. Por lo tanto, el problema de la identificación de un *tag* consiste en identificar múltiples objetos con un retardo y consumo de energía mínimos, con fiabilidad y capacidad de adaptación a incrementos en el número de etiquetas (escalabilidad). A diferencia de los protocolos clásicos de acceso medio, la utilización global de un canal y la justicia (reparto justo de la capacidad entre los usuarios) no se tienen en cuenta en el diseño de los sistemas RFID.

En el caso de los *tags* pasivos, y debido sobre todo a las limitaciones de estos dispositivos, los protocolos suelen ser muy simples y la mayoría de ellos se agrupan dentro de una de estas dos categorías:

- Algoritmos deterministas de segmentación (*Splitting*). El conjunto de *tags* a identificar se descompone en pequeños subgrupos mediante técnicas de *splitting* hasta que el número de *tags* por subgrupo sea de uno. Para llevar a cabo este algoritmo, los *tags* seleccionan un número aleatorio, o bien el *Reader* envía un número de serie correspondiente a un único subgrupo de *tags* (ID). Estos algoritmos se conocen también como algoritmos de búsqueda en árbol o búsqueda binaria.
- Algoritmos probabilísticos. La otra gran familia de protocolos se basa en el protocolo *Frame Slotted ALOHA* (*Frame Slotted Aloha*, FSA) [6]. En este caso, tras recibir una señal del *Reader*, los *tags* arbitrariamente seleccionan una ranura de entre *K* (longitud de la trama) y envían su identificador *ID* en la ranura elegida. Este mecanismo es muy simple, pero cuando el número de *tags* aumenta, es necesario algún mecanismo de adaptación de trama, para decidir el valor de *K*. [3].

1.2.2 Qué deben hacer los alumnos

Los alumnos deben implementar por parejas un simulador de un sistema de identificación por radiofrecuencia en el entorno de simulación OMNET++. Con el simulador completo, se deberán tomar una serie de medidas, que quedarán recogidas en una memoria final, junto con el código comentado (tanto de desarrollo del simulador como de automatización del proceso de simulación y cálculo de resultados).

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 el apartado 1.3 se proporciona al alumno una descripción teórica del sistema 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 Protocolo Framed Slotted ALOHA (FSA)

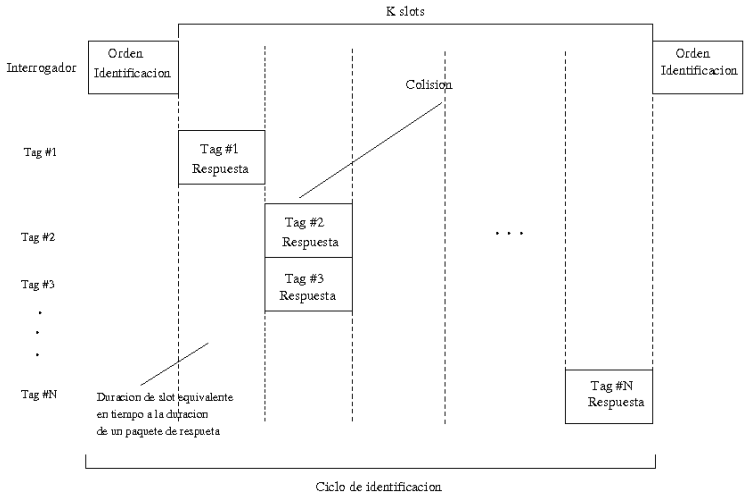


Fig 1. Procedimiento de anticolisión ISO 18000-7 ([5])

Tanto los estándares ISO 18000-7 como EPC ``Gen 2" [3] utilizan un protocolo del tipo *Frame Slotted ALOHA* (FSA). En ambos casos, una población de *N tags* comienza el proceso de identificación tras recibir una orden de inicio

de identificación por parte del *Reader*. A partir de este punto, como muestra la Fig. 1, los *tags* seleccionan de forma aleatoria y siguiendo una distribución uniforme, una ranura temporal (*slot*) y transmiten su número identificativos, *ID*, en la ranura seleccionada. Los *tags* deben elegir una ranura de entre *K* posibles ranuras (tamaño de la trama). Si dos o más *tags* seleccionan la misma ranura temporal, se produce una colisión.
Por tanto, un ciclo de identificación abarca los comandos de orden de identificación enviados por el *Reader* y las *K* ranuras temporales, como se muestra en la Fig. 1. Tras un ciclo de identificación, el lector envía una secuencia de mensajes de reconocimiento (*ACK*) que incluye el identificador de cada uno de los *tags* que se han identificado correctamente. Al recibir dicho mensaje, los *tags* se retiran del proceso de identificación (no envían más mensajes de identificación). Después de tres rondas de identificación sin respuesta, el *Reader* asume que todos los *tags* han sido identificados. Para la simulación, el lector enviará los mensajes de reconocimiento justo antes de enviar la siguiente orden de identificación.

Distinguiremos dos formas de funcionamiento del protocolo FSA:

- Básico: en este caso el tamaño de la trama, *K*, es decir, el número de slots que se emplea es fijo. Esta versión es muy simple y permite su implementación de manera sencilla en tags muy limitados en recursos. Sin embargo, a partir de un cierto número de tags, el protocolo es incapaz de identificar a los tags, debido al elevado número de colisiones. De hecho, se puede demostrar que el protocolo funciona de manera óptima cuando el tamaño de la trama coincide exactamente con el número de tags que intentan identificarse (*K*=*N*). En ese caso, el *throughput* o rendimiento de la identificación, definido como $\frac{Tags\_identificados\_por\_ciclo}{slots\_empleados\_por\_ciclo}$ tiende a e^{-1}. Como se puede comprobar el protocolo no es especialmente eficiente, ni siquiera en el caso óptimo. El problema además es que el lector no conoce a priori el número de tags presente.
- Adaptativo: en este caso, el lector varía el valor de *K* en cada ciclo de identificación. La idea es adaptar el rendimiento del protocolo a la carga: si hay pocos tags y utilizamos una longitud de trama alta, muchos slots estarán vacíos, con lo que desperdiciamos capacidad del canal y se alarga el tiempo de identificación. Si la longitud de trama es insuficiente, se producirán muchas colisiones. El lector decidirá mediante un algoritmo determinada la longitud de la trama.

1.3.1 Análisis mediante procesos de Markov de FSA básico.

El proceso de identificación puede ser modelado cuando un proceso (homogéneo) Markov $\{X_s\}$, donde $\{X_s\}$ denota el número de *tags* no identificados en el ciclo *s*. Así pues, dados N tags para identificar, el espacio de estados del proceso de Markov se denota como {*N*, *N*-1, ..., 0}. Es decir, el estado inicial corresponde a N tags sin identificar, y así sucesivamente hasta llegar al estado final con 0 tags sin identificar. Las probabilidades de transición p_{ij} coinciden con la probabilidad de identificar *j* tags cuando hay *i* tags compitiendo. Estas probabilidades dependerán del número de slots *K* que tenga una trama FSA. Dichas probabilidades se pueden expresar como:

$$P(N,K,i)=\frac{N!}{K^N}\binom{K}{i}\sum_{c=0}^{N-i}\left((-1)^c\binom{K-i}{c}\frac{(K-(i+c))^{N-i-c}}{(N-i-c)!}\right)$$

Donde *N* es el número de tags, *K* el tamaño de trama e *i* el número de tags identificados. Por ejemplo, la probabilidad de identificar 3 tags con una trama de 8 slots y 10 tags compitiendo sería P(10,8,3). Observe que la cadena de Markov correspondiente es absorbente, es decir, hay estados de los cuales el sistema de no sale nunca, es decir p_{ii}=1. En este caso el único estado absorbente es el correspondiente a N=0 (todos los tags identificados). Puesto que el estado absorbente es el correspondiente a la identificación de todos los tags, el número medio de ciclos de identificación es igual al número medio de pasos hasta la absorción, que se denota como:

$$\mathbf{t} = F\mathbf{c} \qquad (4)$$

donde ***t*** es un vector columna (*N+1x1*) y cada uno de sus elementos t_s es el número esperado de pasos (ciclos, en este caso) hasta que la cadena sea absorbida para cada uno de los estados. *F* es la matriz fundamental correspondiente a la matriz de transición *P*, es decir $F=(I-Q)^{-1}$, dónde *Q* es la matriz resultante de eliminar la última fila y columna de *P*. ***c*** es un vector columna (*N+1x1*) cuyos valores son todos 1 (ver [1]). Por tanto, podemos saber el número medio de ciclos hasta identificación resolviendo la ecuación anterior y observando el valor del primer elemento de **t**.
La tabla 1 muestra el número medio de ciclos frente al número de *tags N* para diferentes longitudes de trama (en número de *slot*s). La tabla muestra que con una longitud de trama fija, el número de ciclos aumenta exponencialmente con el número de *tags*. Por lo tanto, un mecanismo simple como un *Frame Slotted ALOHA* no escala bien, y requiere un mecanismo de adaptación de trama si el número de *tags* aumenta de forma considerable.

Tabla 1. Numero medio de ciclos para identificación

	Tags									
Slots	10	20	30	40	50	60	70	80	90	100
4	8.2	60	630	8159	$1.1\ 10^5$	$1.6\ 10^6$	$2.5\ 10^7$	$3.8\ 10^8$	$6.0\ 10^9$	$9.6\ 10^{10}$
8	3.67	8.56	19.6	49.4	138.0	413.9	1304.2	4244.6	14127	47797
16	2.44	4.11	6.15	8.93	13.03	19.3	29.41	46.0	73.81	121.3
32	1.89	2.76	3.60	4.47	5.424	6.50	7.76	9.26	11.0	13.2
64	1.54	2.15	2.61	3.06	3.465	3.90	4.32	4.77	5.23	5.72

2 Descripción del simulador

Es tarea del alumno decidir la estructura del simulador, es decir, los módulos que se usarán y como se relacionan entre sí, además de los parámetros que admiten, etc.. Se tendrá en cuenta las siguientes especificaciones:

- La velocidad del canal entre el lector y los tags es de 40kbps. El canal se supone libre de errores.
- Se utilizarán tres tipos de paquetes: orden de identificación (QUERY) que emite el lector, con el que comienza un ciclo de identificación, cuyo tamaño es 25 bytes. El paquete de identificación de los tags (ID) cuyo tamaño es 40 bytes. El paquete de reconocimiento (ACK) cuyo tamaño es de 10 bytes.

2.1 Escenario 1: FSA básico y validación

En este escenario se simulará el algoritmo FSA básico, es decir, en cada simulación se utiliza un **tamaño de trama fijo, K.** El objetivo de este escenario es validar el simulador comparando los resultados de la simulación con los resultados teóricos que se muestran en la tabla. Se mostrarán los siguientes resultados:

- Genere una **única gráfica** en la que se muestre **Número medio de ciclos para identificación** frente a número de tags, para distintos tamaños de tramas. Es decir, debe mostrar una familia de curvas, una por cada tamaño de trama, para K={4,8,16,32,64}, y para un número de tags N={10,20,30,40}. **Es obligatorio mostrar los intervalos de confianza** en cada una de las curvas. Compare los resultados con los resultados teóricos de la Tabla 1.
- Genere una **única tabla** en la que se muestre **el Throughput medio** y **tiempo medio para identificar a todos los tags** (observe que, debido a que el lector envía un número paquetes ACK variable, según número de identificados, el tiempo entre ciclos de identificación será variable, con lo que no basta con conocer el número medio de ciclos) para un conjunto de N=10 tags y tamaños de tramas K={4,8,16,32,64}. **Es obligatorio mostrar los intervalos de confianza** en cada una de los resultados. Interprete los resultados.

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

2.2 Escenario 2: FSA adaptativo óptimo

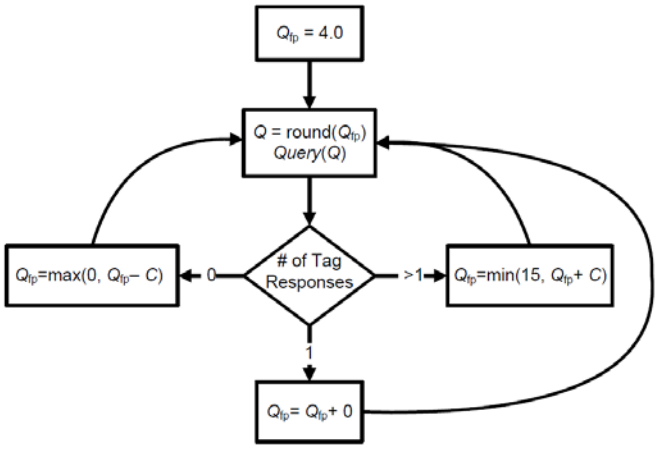
Puesto que FSA básico o bien no es capaz de adaptarse a un gran número de tags, o bien es muy ineficiente, es necesario emplear un algoritmo de adaptación del tamaña de trama. En este escenario se simulará un algoritmo adaptativo óptimo, es decir, el lector conoce en todo momento el número de tags que quedan por identificar y en cada ciclo de identificación utiliza una longitud de trama K igual al número de tags que quedan por identificar. Se mostrarán los siguientes resultados:

- Genere una **única gráfica** en la que se muestre **Número medio de ciclos para identificación** frente a número de tags, para un número de tags N={10,20,30,40,50,60}. **Es obligatorio mostrar los intervalos de confianza**. Interprete y compare los resultados con los resultados teóricos.
- Genere una **única tabla** en la que se muestre **el Throughput medio** y **tiempo medio para identificar a todos los tags** para un conjunto de tags N={10,20,30,40,50,60}. **Es obligatorio mostrar los intervalos de confianza** en cada una de los resultados. Interprete los resultados.

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

2.3 Escenario 3: FSA adaptativo con algoritmo no óptimo

El protocolo FSA adaptativo óptimo en general no se puede utilizar en la práctica porque el lector no conoce el número de tags que están compitiendo, para ajustar el tamaño de trama adecuadamente. La solución normalmente consiste en utilizar algún mecanismo para estimar el número de tags compitiendo. En este escenario se simulará el heurístico utilizado por el protocolo EPC Gen 2 [3]. Este algoritmo utiliza tres parámetros: Q, Q_{fp} y c. El tamaño de trama en cada ciclo se establece como $K=2^Q$, por tanto sólo se utilizan tamaños potencia de dos. El algoritmo se muestra en la siguiente figura:



Es decir, por cada slot con colisión se incrementa el parámetro Q_{fp} por el valor de c. Por cada slot vacío, se decrementa y si hay respuesta correcta se deja igual. Se parte de un valor inicial de Q=4 y Q_{fp} =0 y se sugiere un valor de c entre 0.1 y 0.5.

Se mostrarán los siguientes resultados:

- Genere una **única gráfica** en la que se muestre **Número medio de ciclos para identificación** frente a número de tags, para un número de tags N={20,40,60,80}, para el **heurístico de EPC y el óptimo del apartado anterior**. Interprete los resultados. **Es obligatorio mostrar los intervalos de confianza**.
- Genere una **única tabla** en la que se muestre **el Throughput medio** y **tiempo medio para identificar a todos los tags** para un conjunto de tags N={10,20,30,40,50,60}, para el heurístico de EPC. **Es obligatorio mostrar los intervalos de confianza** en cada una de los resultados. Interprete los resultados.

Se valorarán los comentarios argumentados a los resultados que ha obtenido. **Se valorará especialmente que aporte, describa y justifique un algoritmo que pudiera mejorar los resultados.**

2.4 Estudio teórico

En este apartado se propone un ejercicio teórico de análisis mediante cadenas de Markov del proceso de identificación. A partir de las indicaciones del apartado 1.3.1, realice lo siguiente:

a) Proporcione un diagrama de transición de estados para el proceso de identificación.
b) Proporcione la expresión de la matriz de transición P en función de las probabilidades de transición.
c) Realice e incluya en la memoria un script en Matlab que calcule el número medio de ciclos hasta identificación a partir de una N y K cualquiera. Genere una tabla con el script como la Tabla 1 pero para valores de K={7,14,21,28,35}.
d) Se puede comprobar que el número medio de tags identificados por ciclo es igual a $E[i] = N\left(\frac{K-1}{K}\right)^{N-1}$.

   Dibuje una gráfica en la que se muestre para un tamaño de trama K=16 el número medio de identificaciones para N={1...80}. Compare los resultados con las gráficas clásicas del throughput para Aloha ranurado.

3 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.**

3.1 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.

3.2 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	3
Escenario 2	3
Escenario 3	3
Estudio teórico	1

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. Como se dijo en la introducción, s**e valorarán propuestas de otros algoritmos y el estudio de la influencia de distintos parámetros 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.

3.3 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	4
Escenario 2	3
Escenario 3	2
Estudio teórico	1

Grupos de tres personas:

Apartado	Puntuación máxima
Escenario 1	2
Escenario 2	2
Escenario 3	4
Estudio teórico	2

4 Sugerencias para la implementación

4.1 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
```

4.2 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.

4.3 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.

4.4 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.

4.5 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 líneas de depuración en ningún fichero, lo cual ralentiza enormemente la ejecución. Deberá comentar o desactivar la escritura en ficheros (que no sean necesarios para obtener los resultados), 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 crear una instancia nueva). 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.

4.6 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.

5 Bibliografía

[1] András Varga, “OMNET++ discrete Event Simulation System, Versión 3.2. User Manual.”
[2] Grinstead, C. M., Snell, J. L., Introduction to Probability, 2nd Edition, Capítulo 11, American Mathematical Society, 2003. Disponible online en:
http://www.dartmouth.edu/~chance/teaching_aids/books_articles/probability_book/pdf.html
[3] Class 1 Generation 2 UHF Air Interface Protocol Standard Version 1.0.9: ``Gen 2".
Disponible online en: http://www.epcglobalinc.org/standards