Universidad Politécnica de Cartagena

Escuela Técnica Superior de Ingeniería de Telecomunicación

PRÁCTICAS DE REDES DE ORDENADORES

Propuesta del Trabajo de Prácticas 2007

Simulación de algoritmos Automatic Repeat Request (ARQ)

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 algoritmos clásicos ARQ, es decir, *Go-Back-N* y *Selective Repeat*. 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 Qué deben hacer los alumnos

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

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

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

1.2.2 Cómo está organizada esta propuesta.

En los apartados **¡Error! No se encuentra el origen de la referencia.** a 1.2.6 de la introducción se proporciona al alumno una visión general de los mecanismos de ARQ 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.

1.2.3 Mecanismos Automatic Repeat reQuest (ARQ)

Los mecanismos ARQ son algoritmos genéricos de control de flujo y control de errores extremo a extremo [2-3]. Aunque habitualmente se describen en la capa de enlace, se emplean también en capas superiores, con las modificaciones pertinentes. Por ejemplo, las diferentes versiones de TCP emplean variantes de estos algoritmos para el control de errores extremo a extremo y control de la congestión de red.

Estos algoritmos basan su funcionamiento en el uso de detección de errores, temporizadores, paquetes de reconocimiento positivo (**ACK**) y negativo (**NACK**) y retransmisiones de los diferentes tipos de paquetes Para explicar su funcionamiento supondremos que existen dos estaciones (transmisor, tx y receptor, rx), conectadas directamente con un enlace bidireccional. El enlace viene descrito por una velocidad de transmisión binaria (**V**), una probabilidad de error de bit (**BER**) y un retardo de propagación extremo a extremo ($\mathbf{T_{prop}}$). Estos tres parámetros definen el enlace **para cada sentido** independientemente.

De manera general el funcionamiento de estos algoritmos es el siguiente: el transmisor encapsula los datos en tramas de nivel de enlace (**DATA**) y los envía al receptor. Cada trama se numera consecutivamente, mediante un **número de secuencia**, de manera que se pueda garantizar la entrega ordenada al nivel superior. La trama de enlace incluye un campo de verificación de trama (CRC) de manera que el receptor puede comprobar si la trama ha sido dañada en el tránsito. En este caso la trama se descarta. En caso contrario, se envía al transmisor una trama de reconocimiento, que incluye el número de secuencia de la trama recibida correctamente. El transmisor debe retransmitir las tramas que no han sido recibidas correctamente. Para evitar que el protocolo quede bloqueado, es necesario utilizar temporizadores

para la retransmisión de las tramas. Los distintos algoritmos ARQ definen la forma en particular en la que se deben emplear los distintos tipos de tramas y los temporizadores para implementar el comportamiento deseado.
El control de flujo se realiza mediante la técnica de ventana deslizante. En esencia, esta técnica consiste en limitar el número máximo de tramas *en tránsito*. Es decir, el número de tramas que han sido enviadas pero no han sido reconocidas. A este número se le denomina **tamaño máximo de ventana, W**. El transmisor mantiene un buffer con todas las tramas enviadas pendientes de reconocimiento. A este buffer se le denomina **ventana de transmisión**.
Los dos mecanismos ARQ más comunes son *Go-Back-N* y *Retransmisión Selectiva (Selective Repeat)*. Estos algoritmos difieren en la forma de gestionar los errores y tienen distintas variantes. En las siguientes secciones detallaremos su funcionamiento.

1.2.4 Go-Back-N

El mecanismo de *Go-Back-N* es el más comúnmente empleado. Existen diferentes variantes del mismo. Nosotros proponemos aquí la que se rige por las siguientes reglas:

1. Suponemos que el transmisor siempre dispone de **tramas de datos (DATA)** para enviar. El transmisor inicia los números de secuencia por 0. Cada nueva trama generada se numera secuencialmente.
2. El transmisor envía tramas al receptor consecutivamente y las **almacena en la ventana de transmisión** a la espera de ser reconocidas. Por cada trama enviada el transmisor incrementa el tamaño de la ventana de transmisión. El número máximo permitido de tramas en tránsito (enviadas pero no reconocidas) es W, por tanto, este es el tamaño máximo de la ventana de transmisión.
3. El receptor acepta las tramas de manera ordenada. Es decir, descarta las tramas fuera de secuencia. Por tanto, al recibir una trama de datos se comprueba si tiene errores.
    a. Si los tiene, se descarta.
    b. Si no los tiene, se comprueba si su número de secuencia corresponde al de la trama esperada.
        i. Si corresponde, se entrega al nivel superior y **se envía una trama de reconocimiento ACK** indicando el número de la trama recibida correctamente.
        ii. Si no corresponde, se descarta la trama y se **envía una trama NACK** indicando el número de secuencia de la trama esperada. A partir de este momento, **el receptor no envía ningún otro tipo de trama** hasta recibir la trama esperada correctamente.
4. Al recibir una trama ACK, el transmisor elimina la trama reconocida de la ventana de transmisión, decrementa el tamaño de la ventana y comprueba si puede seguir transmitiendo, es decir, si el tamaño de la ventana es menor que W.
5. Al recibir una trama NACK, el transmisor **reenvía toda la ventana de transmisión**. Esta operación tiene prioridad sobre el envío normal de tramas, es decir, incluso aunque no se haya alcanzado W, el transmisor comienza la retransmisión de las tramas.

Es necesario el uso de un temporizador para el correcto funcionamiento del protocolo. Nótese que el protocolo quedaría atascado si:

- Se pierde toda la ventana de transmisión.
- Se pierden tramas de reconocimiento (tanto ACK como NACK). Recuerde que existe una probabilidad de error, BER, en ambos sentidos.

Por tanto, el transmisor dispone de un temporizador (**rtxTO**) de duración **rtxInterval**. Este temporizador se inicia al enviar la primera trama y estará activo siempre que haya alguna trama en tránsito.

- El temporizador se reinicia al recibir un ACK y al recibir un NACK.
- Cuando vence el temporizador **se reenvía toda la ventana**. Esto implica que se vuelve a poner en marcha el temporizador al comenzar las retransmisiones.

La duración del temporizador deber ser ligeramente superior al **retardo de ida y vuelta** (**RTT**, *Round Trip Time*).

La ventaja de *Go-Back-N* es que es muy sencillo de implementar y que el receptor no necesita disponer de un buffer de almacenamiento, puesto que sólo acepta tramas entregadas en orden. Sin embargo, por cada error, se produce la retransmisión de N tramas. Nótese que, en general, **N es distinto de W**, y depende del RTT**.** El número de retransmisiones N corresponde al número de tramas que se pueden enviar durante un RTT, que es el tiempo mínimo necesario para detectar un error. N siempre será menor o igual que W.

1.2.5 Retransmisión Selectiva

El mecanismo *Go-Back-N* destaca por su sencillez y porque no es necesario que el receptor disponga de un buffer de almacenamiento de tramas. Sin embargo, cuando se produce un error, es ineficiente, puesto que la pérdida de una única trama da lugar a múltiples retransmisiones (N). El mecanismo de retransmisión selectiva corrige esta ineficiencia a costa de complicar la implementación del transmisor y receptor.

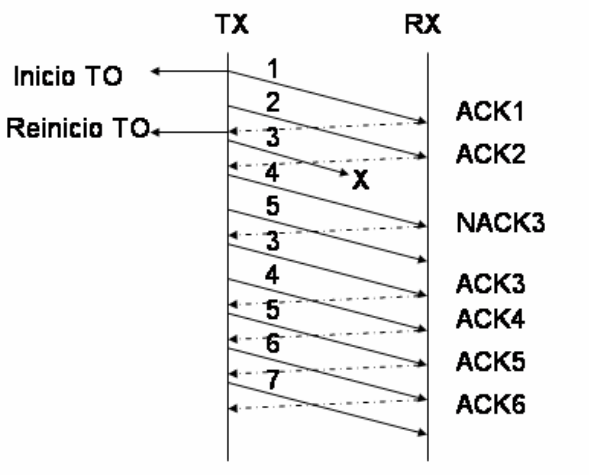

Figura 1. Go-Back-N

En este caso, el receptor acepta todas las tramas, **independientemente de si llegan en secuencia o no**, pero sólo las entrega de manera ordenada al nivel superior. Por su parte, el transmisor sólo retransmite las tramas que no han sido reconocidas correctamente. Por tanto, es necesario un buffer de almacenamiento tanto en transmisor como receptor, es decir, existe una **ventana de recepción y ventana de transmisión**. Ambas ventanas tendrán un tamaño máximo W. El receptor almacenará las tramas recibidas en su ventana de recepción y las irá entregando si llegan secuencialmente. En caso de producirse un error, el receptor sigue aceptando las siguientes tramas, pero **sólo las entrega cuando dispone de una secuencia completa**, es decir, cuando ha recibido de nuevo y correctamente la trama errónea (**entrega ordenada**).
Existen diferentes variantes de este mecanismo. Nosotros proponemos las siguientes:

1.2.5.1 RETRANSMISIÓN SELECTIVA CON UN ÚNICO TEMPORIZADOR

Las reglas de este mecanismo son las siguientes:

1. Suponemos que el transmisor siempre dispone de **tramas de datos (DATA)** para enviar. El transmisor inicia los números de secuencia por 0. Cada nueva trama generada se numera secuencialmente.
2. El transmisor envía tramas al receptor consecutivamente y las **almacena en la ventana de transmisión** a la espera de ser reconocidas. Por cada trama enviada el transmisor incrementa el tamaño de la ventana de transmisión. El número máximo de tramas en tránsito (enviadas pero no reconocidas) es W, por tanto, este es el tamaño máximo de la ventana de transmisión.
3. El receptor acepta todas las tramas libres de error. Por tanto, al recibir una trama de datos se comprueba si tiene errores.
    a. Si los tiene, se descarta.
    b. Si no los tiene, se comprueba si su número de secuencia está dentro del rango permitido, es decir, si está en $]S_{min}, S_{min} + W]$, dónde S_{min} es el número de secuencia de la última trama entregada al nivel superior.
        i. Si está dentro del rango permitido, **se envía una trama de reconocimiento ACK** indicando el número de la trama recibida correctamente. Además, se entregan al nivel superior todas las tramas almacenadas en secuencia correcta.
        ii. En caso contrario, se descarta.
4. Al recibir una trama ACK, el transmisor **marca** como reconocida la trama indicada por el ACK (si se encuentra dentro de su ventana de transmisión). A continuación, comprueba si el reconocimiento corresponde al número de secuencia de la primera trama de la ventana de transmisión.
    a. Si no corresponde, quiere decir que se ha producido un error en la recepción en alguna de las tramas enviadas. El transmisor **marca para reenviar** las tramas sin reconocer entre la primera trama de la ventana y la recién reconocida. A continuación, reenvía las tramas marcadas, si puede enviar en ese momento.
    b. Si corresponde, el transmisor elimina de la ventana todas las tramas reconocidas en secuencia y decrementa el tamaño de la ventana consecuentemente. A continuación, comprueba si puede enviar de nuevo.

De nuevo, es necesario el uso de un temporizador para el correcto funcionamiento del protocolo. Nótese que el protocolo quedaría atascado si:

- Se pierde toda la ventana de transmisión.

- Se pierden tramas de reconocimiento. Recuerde que existe una probabilidad de error, BER, en ambos sentidos.

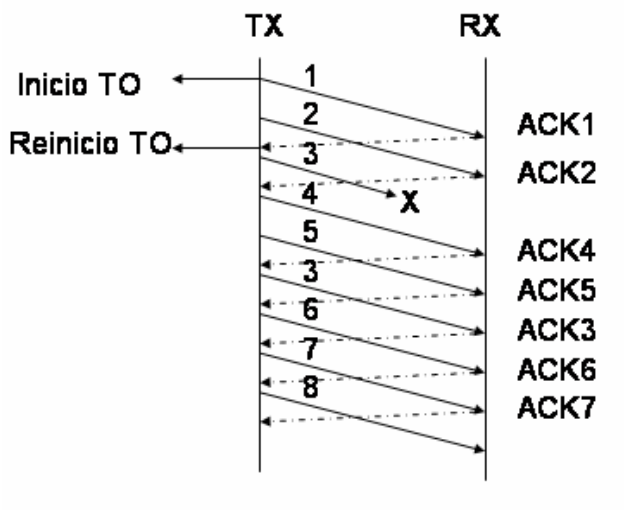

Figura 2. Selective Repeat, 1 temporizador

Por tanto, el transmisor dispone de un temporizador (**rtxTO**) de duración **rtxInterval**. Este temporizador se inicia al enviar la primera trama y estará activo siempre que haya alguna trama en tránsito.

- El temporizador se reinicia al recibir un ACK.
- Cuando vence el temporizador **se reenvía toda la ventana**. Esto implica que se vuelve a poner en marcha el temporizador al comenzar las retransmisiones.

La duración del temporizador deber ser ligeramente superior al **retardo de ida y vuelta** (**RTT**, *Round Trip Time*).

1.2.5.2 RETRASNMISIÓN SELECTIVA CON MÚLTIPLES TEMPORIZADORES

El rendimiento del rechazo selectivo se puede mejorar si se pone en marcha **un temporizador por trama enviada.** De hecho, el rendimiento se aproxima mucho al que se obtendría con un *Selective Repeat* ideal (en el que el receptor dispone de una ventana de tamaño infinito). Esta mejora se obtendría a costa de complicar la lógica del transmisor.

En nuestra implementación, las reglas que seguiremos son las mismas que las del apartado anterior para la transmisión y recepción de tramas, salvo la regla 4.a. En este caso, dicha regla queda como sigue:

a. Si no corresponde, quiere decir que se ha producido un error en la recepción en alguna de las tramas enviadas. El transmisor **no realiza ninguna acción**. La trama se reenviará cuando venza su temporizador. Como explicamos a continuación.

En efecto, la gestión de los temporizadores se realiza de la siguiente forma:
El transmisor dispone de un temporizador (**rtxTO$_x$**) por cada trama *en tránsito*, de duración **rtxInterval**. Es decir, un número de temporizadores igual al tamaño máximo de ventana. Cada temporizador se inicia al enviar la una nueva trama y estará activo mientras dicha trama esté en tránsito.

- El temporizador se para al recibir un ACK para la trama con la que se inició.
- Cuando vence un temporizador **se reenvía únicamente la trama afectada**. Esto implica que se vuelve a poner en marcha el temporizador al comenzar las retransmisión.

La duración del temporizador deber ser ligeramente superior al **retardo de ida y vuelta** (**RTT**, *Round Trip Time*).

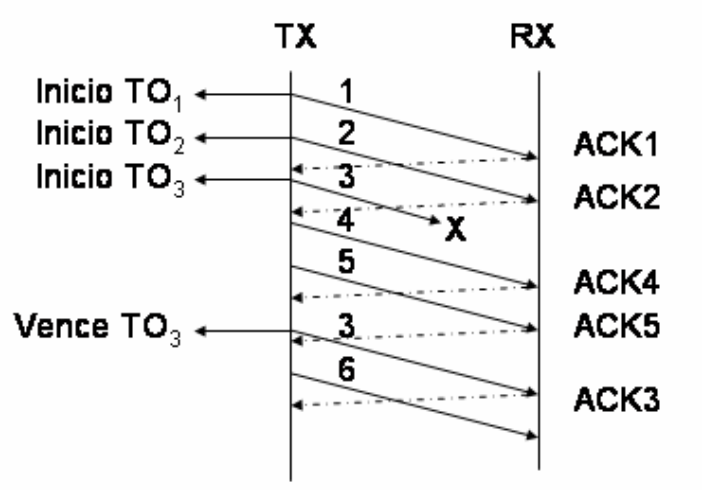

Figura 3. Selective Repeat, múltiples temporizadores

1.2.6 Rendimiento teórico de los mecanismos de ARQ

Las siguientes fórmulas [2] proporcionan el rendimiento teórico de los mecanismos de ARQ. Antes definimos el parámetro $a = \frac{T_{propagacion}}{T_{transmision}}$, donde $T_{propagacion}$ es el retardo del canal y $T_{transmision}$ el tiempo de transmisión de una trama, y la probabilidad de que una trama sea errónea, $P = (1 - BER)^n$, donde n es el tamaño de la trama. Además, necesitamos el tamaño máximo de ventana, W. A partir de aquí, y teniendo en cuenta la velocidad binaria del canal, V, podemos obtener el *throughput* normalizado $S = \frac{Bits\,\mathrm{Re}conocidos}{BitsTransmitidos} \cdot \frac{1}{V}$.

Para Go-Back-N:

$$S = \begin{cases} \dfrac{1-P}{1+2aP} & W \geq 2a+1 \\ \dfrac{W(1-P)}{(2a+1)(1-P+WP)} & W < 2a+1 \end{cases}$$

Para *Selective Repeat*:

$$S = \begin{cases} 1-P & W \geq 2a+1 \\ \dfrac{W(1-P)}{(2a+1)} & W < 2a+1 \end{cases}$$

2 Fases del desarrollo del simulador y medidas a realizar

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

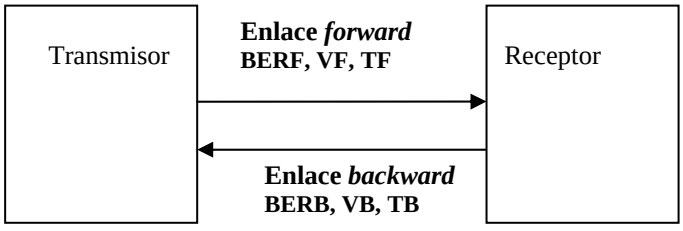

Figura 4 Topología del simulador

La topología del simulador, reflejada en la Figura 4, será la misma para todos los escenarios. Se compone de los módulos transmisor y receptor, unidos mediante dos enlaces, uno transmisor-receptor (enlace *forward*) y otro de retorno, es decir, receptor-transmisor (enlace *backward*). En enlace *forward* tiene asociados una tasa de error de bit (BERF), una velocidad binaria (VF) y un retardo de propagación (TF). El enlace *backward* también tiene asociados una tasa de error de bit (BERB), una velocidad binaria (VB) y un retardo de propagación (TB). Todos estos parámetros son configurables de manera independiente.

Los paquetes de control que se utilizarán (ACK y NACK) tendrán un tamaño siempre de 32 bits. Los paquetes de datos (DATA) tendrán un tamaño de 1992 bits.

Observe que es necesario un temporizador de transmisión. Al utilizar una velocidad de transmisión y un tamaño de paquete, el canal está ocupado durante el tiempo que dura la transmisión de un paquete. **Tenga en cuenta este hecho** en la implementación. No se puede enviar un paquete si se está enviando uno previo (aunque la función *send()* de OMNET++ sí que lo admite, encolando internamente las sucesivas transmisiones).

2.1 Escenario 1. Algoritmo Go-Back-N

En este escenario debe implementar el algoritmo Go-Back-N, tal y como se describe en el apartado 1.2.4. Para poder obtener estadísticas del funcionamiento del sistema, los módulos (el que corresponda) deben guardar los siguientes datos:

- Número de tramas enviadas.
- Número de tramas reconocidas.
- Número de tramas reenviadas.
- Número de tramas recibidas con error.

Las medidas que deben realizar para este escenario son las siguientes:

1. **Influencia del tamaño de ventana en el throughput.** El objetivo es obtener una **representación del throughput normalizado medio del sistema**, comparado con el valor teórico**.** El valor de los parámetros del sistema es el siguiente: VB=VF=1.5 Mbps, TF=TP=0.03 y BERB=10^{-10}. Los valores de W serán 16 y 128. Y los valores de BERF serán: 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3} y 10^{-2}. Por tanto, debe representar una familia de gráficas con el throughput normalizado medido para cada tamaño de ventana y el valor teórico correspondiente, variando la BERF en el eje de abcisas. El eje de las abcisas debe estar en escala logarítmica.
2. **Influencia del retardo en el throughput.** El objetivo es obtener una **representación del throughput normalizado medio del sistema**, comparado con el valor teórico**.** El valor de los parámetros del sistema es el siguiente: VB=VF=1.5 Mbps, W=64 y BERB=10^{-10}. Los valores de TF=TB serán 30 ms y 300 ms. Y los valores de BERF serán: 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3} y 10^{-2}. Por tanto, debe representar una familia de gráficas con el throughput normalizado medido para cada retardo y el valor teórico correspondiente, variando la BERF en el eje de abcisas. El eje de las abcisas debe estar en escala logarítmica.

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

3. Represente en una tabla el número medio de tramas reenviadas por segundo para VB=VF=1.5 Mbps, TF=TP=0.03, BERB=10^{-10}, BERF=10^{-5}, W=16 y W=128, junto con sus intervalos de confianza.

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

2.2 Escenario 2: Selective Repeat. Un temporizador

En este escenario debe implementar el algoritmo *Selective Repeat* con un solo temporizador tal y como se describe en el apartado 1.2.5.1. Para poder obtener estadísticas del funcionamiento del sistema, los módulos (el que corresponda) deben guardar los siguientes datos:

- Número de tramas enviadas.
- Número de tramas reconocidas.
- Número de tramas reenviadas.
- Número de tramas recibidas con error.

Las medidas que deben realizar para este escenario son las siguientes:

1. **Influencia del retardo en el throughput.** El objetivo es obtener una **representación del throughput normalizado medio del sistema**, comparado con el valor teórico**.** El valor de los parámetros del sistema es el siguiente: VB=VF=1.5 Mbps, W=64 y BERB=10^{-10}. Los valores de TF=TB serán 30 ms y 300 ms. Y los valores de BERF serán: 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3} y 10^{-2}. Por

tanto, debe representar una familia de gráficas con el throughput normalizado medido para cada retardo y el valor teórico correspondiente, variando la BERF en el eje de abcisas. El eje de las abcisas debe estar en escala logarítmica.

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

2. Muestre una gráfica con la evolución temporal de la ventana de recepción para VB=VF=1.5 Mbps, TF=TP=0.03, BERB=10^{-10}, BERF=10^{-5}, W=16.
3. Represente en una tabla el número medio de tramas reenviadas por segundo para VB=VF=1.5 Mbps, TF=TP=0.03 y TF=TP=300ms, BERB=10^{-10}, BERF=10^{-5}, W=16, junto con sus intervalos de confianza.

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

2.3 Escenario 3: Selective Repeat. Múltiples temporizadores.

En este escenario debe implementar el algoritmo *Selective Repeat* con un solo temporizado tal y como se describe en el apartado 1.2.5.2. Para poder obtener estadísticas del funcionamiento del sistema, los módulos (el que corresponda) deben guardar los siguientes datos:

- Número de tramas enviadas.
- Número de tramas reconocidas.
- Número de tramas reenviadas.
- Número de tramas recibidas con error.

Las medidas que deben realizar para este escenario son las siguientes:

1. **Comparativa con *Selective Repeat* con un temporizador**. **Repita las medidas 1 y 3** del Escenario 2 y compare los resultados entre ambos Escenarios.

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

3 Sugerencias para la implementación

3.1 Consejos para la depuración y verificación del módulo

Para las tareas de depuración, aquí tiene unos consejos:

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

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

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

Al principio del fichero debe incluir:

```
#include <fstream>
```

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

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

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

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

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

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

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

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

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

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

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

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

3.2 Consejos para una mayor claridad en el código.

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

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

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

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

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

3.3 Gestión eficiente de la memoria

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

3.4 Toma de muestras

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

cStdDev
cWeightedStdDev
cOutVector

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

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

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

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

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

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

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

Puede consultar todo esto en el manual y en API.

3.5 Selección de semillas

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

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

3.6 Ejecuciones más rápidas

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

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

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

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

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

```
opp_makemake –f –u Cmdenv
```

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

```
[Cmdenv]
```

```
express-run = true
express-mode = yes
event-banners = no
```

Nuevamente le animamos a que consulte el manual.

3.7 Realización de réplicas

La obtención rigurosa de resultados por simulación exige cuantificar el error cometido en la estimación. Para ello es necesario o bien tomar medidas por bloques o bien replicar la simulación. Normalmente, la simulación finaliza cuando se ha obtenido un nivel de calidad determinado en la simulación. Esto exigiría, por ejemplo, realizar réplicas hasta que el intervalo de confianza se encuentra dentro de unos márgenes predeterminados (la calidad deseada).

La realización de simulaciones y el procesado de los resultados debe automatizarse: la puesta en marcha de las réplicas y la configuración de los parámetros para cada una de ellas, así como el procesado de los resultados se automatiza mediante el uso de un programa específicamente desarrollado a tal fin. En la práctica, se suelen utilizan lenguajes de *script* para realizar esta tarea, ya que simplifican el proceso. El proceso es el siguiente: una vez obtenido una versión depurada de la simulación, se compila sin interfaz gráfica (ver apartado anterior). De esta manera se puede lanzar el proceso y poner en marcha la simulación sin intervención del usuario. Por ejemplo, el siguiente comando ejecutaría el *Run* 4 de un simulador (consulte el capítulo 8.9 del Manual):

```
./simulador –r 4
```

Mediante un *script*, por tanto, podemos crear los ficheros de configuración correspondiente y ejecutar múltiples simulaciones. El capítulo 8.9 del Manual explica este proceso.

Hay múltiples lenguajes de *script* que pueden ser utilizados para realizar estas tareas. Entre ellos Perl, shell scripts, Python, TCL e incluso Matlab (o su versión libre en Linux, Octave).

El siguiente código muestra un *script* de ejemplo en Perl:

```
#!/usr/bin/perl
$exec = @ARGV[0];
$rep = @ARGV[1];
print ("Ejecutable de la simulacion: $exec\n");
print ("Numero de replicas: $rep\n");
@ber =(1e-8,1e-7,1e-6,1e-5,1e-4,1e-3,1e-2);
foreach $p (@ber){
      print(STDERR "BER: $p\n");
      &run($p);
}


sub printRuns {
      ($number,$b,$r,$seq) = @_;
      open(RUNS,">>runs.ini");
      print(RUNS "[Run ".$number."]\n");
      print(RUNS "ppplink.berForward = ".$b."\n");
      print(RUNS "ppplink.retForward = ".$r."\n");
      print(RUNS "ppplink.tx.datalinkLayer.rtxInterval = ".(2*$r+2e-3)."\n");
      close(RUNS);
}


sub run {
      ($bber)=@_;


      system("rm runs.ini");
      system("rm sca.ini");


      $numb=0;
```

```
	open(SCA,">>sca.ini");
	print(SCA "[General]\n");
	print(SCA "output-scalar-file = $bber.sca\n");
	for ($i=1; $i<=$rep; $i++){

			&printRuns($numb,$bber,0.03,127);
			system("./$exec -r $numb > err$bber-$numb");
			print (STDERR "Finalizado Run $numb\n");
			$numb++;

	}
	close(SCA);


}
```

En dicho ejemplo, se realizan varias réplicas. La configuración de cada réplica se guarda en un fichero *runs.ini* y el nombre del fichero de resultados se guarda en un fichero *sca.ini*. Recuerde que es sólo un ejemplo, se omiten una serie de cuestiones como la configuración de las semillas para cada réplica o se mantiene fijo el retardo y el tamaño de ventana, aunque puede servir de base para su propio *script*.

El procesado de los resultados requiere también la ayuda de un programa que los procese. Una solución habitual es guardar los resultados en un formato adecuado para el programa que los procesará. Por ejemplo, si quiere procesar los resultados y realizar gráficas con Matlab, basta con que guarde los resultados en un fichero de texto (ASCII). Si los resultados se guardan en forma de columnas (campos separados por un tabulador), Matlab los importará como cualquier otro vector.

3.8 Uso de la librería de plantillas STL

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

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

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

```
#include <vector.h>

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

También puede utilizar los iteradores:

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

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

4 Material a entregar y criterios de evaluación

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

4.1 Memoria y Código

La memoria debe contener, al menos los siguientes apartados:

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

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

4.2 Criterios de evaluación

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

Apartado	Puntuación máxima
Escenario 1.	4
Escenario 2.	3,5
Escenario 3.	2,5

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

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

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

4.3 Grupos de una ó tres personas

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

Grupos de una persona:

Apartado	Puntuación máxima
Escenario 1.	5
Escenario 2.	4
Escenario 3.	1

Grupos de tres personas:

Apartado	Puntuación máxima
Escenario 1.	3
Escenario 2.	4
Escenario 3.	2

Para conseguir el punto que falta, los grupos de 3 personas tendrán que realizar el apartado extra que se explica a continuación. Llamaremos a este apartado Escenario 3 extendido.
En una implementación real, los números de secuencia no pueden crecer indefinidamente. Se utiliza un número máximo de secuencia M. Al llegar a este número, los números de secuencia vuelven a 0. A esto se denomina implementación *módulo-M*. Este tipo de implementación además impone una serie de restricciones al tamaño máximo de ventana de los algoritmos, W.

1. Indique dichas restricciones.
2. Proporcione las modificaciones al código necesarias para que su simulador funcione con módulo M.

5 Bibliografía

[1] András Varga, "OMNET++ discrete Event Simulation System, Versión 3.2. User Manual."
[2] William Stallings, "Redes e Internet de Alta Velocidad. Rendimiento y Calidad de Servicio", 2º Edición, Prentice Hall, 2002.
[3] Andrew Tanembaun, "Computer Networks", 4º Edición, Pearson Education, 2003.
[4] H. Deitel, "C++, Como programar", Prentice Hall, 1999.