Universidad Politécnica de Cartagena

Escuela Técnica Superior de Ingeniería de Telecomunicación

PRÁCTICAS DE REDES DE ORDENADORES

Propuesta del Trabajo de Prácticas 2006

Simulación de algoritmos de control de tráfico en ATM

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 un sencillo conmutador (*switch*) de celdas ATM que multiplexará varias conexiones de tasa variable. Los módulos generadores de tráfico (fuentes) se proporcionan al alumno y se facilita la configuración de los mismos. El tráfico generado por cada fuente está caracterizado por unos parámetros conocidos como descriptores de tráfico. La función principal del switch a implementar es el control de tráfico sobre cada conexión. El control de tráfico consiste en comprobar que el tráfico procedente de cada conexión se encuentra dentro de los márgenes especificados por los descriptores de tráfico. En la presente especificación se propone el desarrollo del simulador por etapas, en cada una de las cuales se irán añadiendo funcionalidades al simulador y se irán proponiendo experimentos de los que el alumno deberá ir extrayendo conclusiones.

1.2 Introducción

1.2.1 Qué deben hacer los alumnos

Los alumnos deben implementar por parejas un módulo simple en C++ y NED llamado Switch. Lo que debe hacer este módulo se detalla en esta propuesta. Este módulo deberá ser incorporado en un entorno de simulación del que se proporcionan el resto de los módulos. 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 del módulo se han agrupado de forma que se puedan ir añadiendo de forma incremental al módulo. Para verificar el funcionamiento de dichas funcionalidades se proponen una serie de escenarios de simulación caracterizados por una topología de conexión y por los algoritmos que entran en juego. 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 1.2.3 y 1.2.4 de la introducción se proporciona al alumno una visión general de ATM y de las funcionalidades que se van a implementar, así como una serie de definiciones útiles para la comprensión del resto de la especificación.
En el apartado 2 se explican los algoritmos a implementar.
En el apartado 3 se describe el código proporcionado y cómo emplearlo.
El apartado 4, de gran importancia, 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 5 se dan algunas sugerencias y “pistas” para la realización de este trabajo.
Finalmente en el apartado 6 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 ATM, categorías de servicio y parámetros de tráfico

ATM (Asynchronous Transfer Mode) es un protocolo de comunicaciones de capa 2 desarrollado inicialmente por la ITU-T (Unión Internacional de Telecomunicaciones) para dar soporte a la RDSI-BA (Red Digital de Servicios Integrados de Banda Ancha) y posteriormente ampliado por el ATM Forum. En la actualidad ATM es una de las tecnologías más extendidas ya que se emplea en ADSL y en la telefonía móvil 3G. Como característica fundamental, ATM es un protocolo orientado a conexión, que se basa en la transmisión y conmutación de paquetes de tamaño fijo denominados células ó celdas (*cells*) de 53 bytes de longitud (5 de cabecera y 48 de carga útil). ATM tiene la posibilidad de integrar distintos tipos de tráfico (tasa de bits constante, tráfico tiempo real, tráfico *background*, etc), diferenciándolos y asegurándo a cada uno de ellos unas garantías diferenciadas de calidad de servicio (QoS).

En ATM están definidos los siguientes tipos de tráfico o categorías de servicio: CBR, VBR-rt, VBR-nrt, UBR, ABR, GFR. En el ámbito de este trabajo de prácticas se considera tráfico VBR-rt (*Variable Bit Rate – Real Time*), correspondiente a aplicaciones que requieren un retardo y una variación de retardo restringidos (*Real Time*) pero cuya tasa de transmisión es variable (*Variable Bit Rate*). Cada categoría de servicio está caracterizada por un conjunto de atributos ATM, que en el caso de fuentes VBR-rt son de dos tipos: Descriptores de Tráfico y Parámetros QoS.

Los descriptores de tráfico ó parámetros de tráfico caracterizan el patrón de tráfico de un flujo de celdas procedentes de una fuente en una conexión ATM. En el caso de fuentes VBR-rt los descriptores de fuente son los siguientes:

- PCR (Peak Cell Rate): Régimen máximo de celdas.
- SCR (Sustainable Cell Rate): Régimen sostenible de celdas.
- MBS (Maximum Burst Size): Tamaño máximo de una ráfaga
- CDVT: (Cell Delay Variation Tolerance): Variación permitida del retardo en la entrega de celdas.

En cuanto a los parámetros de QoS tenemos los siguientes:

- CDV (Cell Delay Variation): Es la diferencia entre el retardo fijo experimentado por una celda y el retardo máximo tolerado (maxCTD). El retardo fijo incluye el retardo de propagación a través de los medios físicos, los tiempos de inserción de celdas (tiempo necesario para introducir los 424 bits de una celda en el enlace, considerando la tasa binaria del mismo) y otros retardos debidos al procesamiento de conmutación de los equipos. El retardo variable (CDV) es debido a los buffers y a la planificación de celdas. No confundir con CDVT.
- maxCTD (max Cell Transfer Delay): Retardo máximo requerido por una conexión.
- CLR (Cell Loss Ratio) Tasa de pérdida de celdas.

1.2.4 Control de tráfico

El control de tráfico comprende una serie de funciones cuyo objetivo es mantener la QoS de las conexiones ATM. Estas funciones son básicamente un conjunto de acciones llevadas a cabo por la red para evitar situaciones de congestión o minimizar los efectos de la congestión. Se han definido las funciones siguientes:

1. Administración de recursos empleando rutas virtuales.
2. Control de admisión de conexiones.
3. Control del parámetro de utilización.
4. Descarte selectivo de celdas.
5. Conformación de tráfico.
6. Indicación explícita de congestión progresiva.

En el ámbito del presente trabajo nos centraremos en las funciones 3, 4 y 5 únicamente.

1.2.4.1 CONTROL DEL PARÁMETRO DE UTILIZACIÓN

Cuando se ha establecido una conexión (extremo a extremo) para una fuente, esta conexión, denominada VCC (*Virtual Circuit Connection*), tiene asociado un contrato de tráfico. Este contrato de tráfico contiene, por un lado, el descriptor de tráfico de la conexión (PCR, SCR, MBS, CDVT) y por otro lado los objetivos de QoS requeridos (CTV, maxCTD, CLR). De esta forma, si la fuente se mantiene dentro de los márgenes delimitados por sus descriptores de tráfico, la red se "compromete" a garantizar los objetivos de QoS del contrato.

El control del parámetro de utilización (UPC: *Usage Parameter Control*) es el algoritmo (o algoritmos) que se encarga de comprobar si el flujo de celdas procedente de una fuente cumple el contrato, es decir, se ajusta a los descriptores de tráfico del VCC. El objetivo principal del UPC es proteger los recursos de red contra una sobrecarga de una conexión que pudiera afectar negativamente a la QoS de otras conexiones. Las celdas que no cumplen el contrato son descartadas o marcadas como descartables en caso de congestión (descarte selectivo). Al UPC también se le denomina política de tráfico (*traffic policing*).

El UPC se implementa generalmente mediante dos algoritmos por separado, el algoritmo de régimen máximo de celdas, que controla el PCR y el CDVT y el algoritmo de régimen sostenible de celdas, que controla el SCR y el MBS.

1.2.4.2 CONFORMACIÓN DE TRÁFICO

La conformación de tráfico (*traffic shaping*) es una técnica que pretende suavizar el flujo de tráfico y reducir el agolpamiento de celdas. Esto puede dar lugar a una asignación más equitativa de recursos y a mejorar la QoS proporcionada.

2 Algoritmos a implementar

2.1 Algoritmo de régimen máximo de celdas

Como se ha comentado anteriormente el objetivo de este algoritmo es controlar dos parámetros: CDVT y PCR. El algoritmo empleado más habitualmente es el denominado Generic Cell Rate Algorithm (Algoritmo Genérico de Tasa de Celdas) que admite dos argumentos, un incremento I y un límite L, y se expresa de la forma GCRA(I, L). Como veremos, para el control de régimen sostenible de celdas también se usa el GCRA. A continuación veremos dos posibles implementaciones equivalentes de este algoritmo: Algoritmo de planificación virtual y algoritmo de cubo con pérdidas (Leaky Bucket).

Veamos primero la versión dada por el algoritmo de planificación virtual. Supongamos que hemos especificado una velocidad de pico R (PCR) y un límite a la variación del retardo de celdas τ. El tiempo entre llegadas de celdas, a la tasa de pico es T=1/R. Dados T y τ, el algoritmo se puede expresar como GCRA(T, τ).

En la Figura 1 puede ver el diagrama de flujo del algoritmo de planificación virtual. El algoritmo se inicializa con la llegada de la primera celda de la conexión en el instante $t_a(1)$. Durante su ejecución actualiza el tiempo de llegada teórico (TAT): la estimación del instante de llegada para la próxima celda. Si la celda llega mas tarde que el TAT, entonces está conforme y el TAT se actualiza con el instante de llegada $t_a(k)$ más T. Si la celda llega antes que el TAT pero después que el (TAT - τ), la celda es aún conforme y el TAT es incrementado en T. En este último caso, la celda que llega antes es conforme porque aún está dentro de la variación del retardo de celda "permitido". Si la celda llega demasiado pronto, antes que el (TAT - τ), entonces está fuera de la variación del retardo de celda "permitido" y es declarada como no conforme. En este caso el TAT no cambia. La Figura 2 ilustra estas tres zonas.

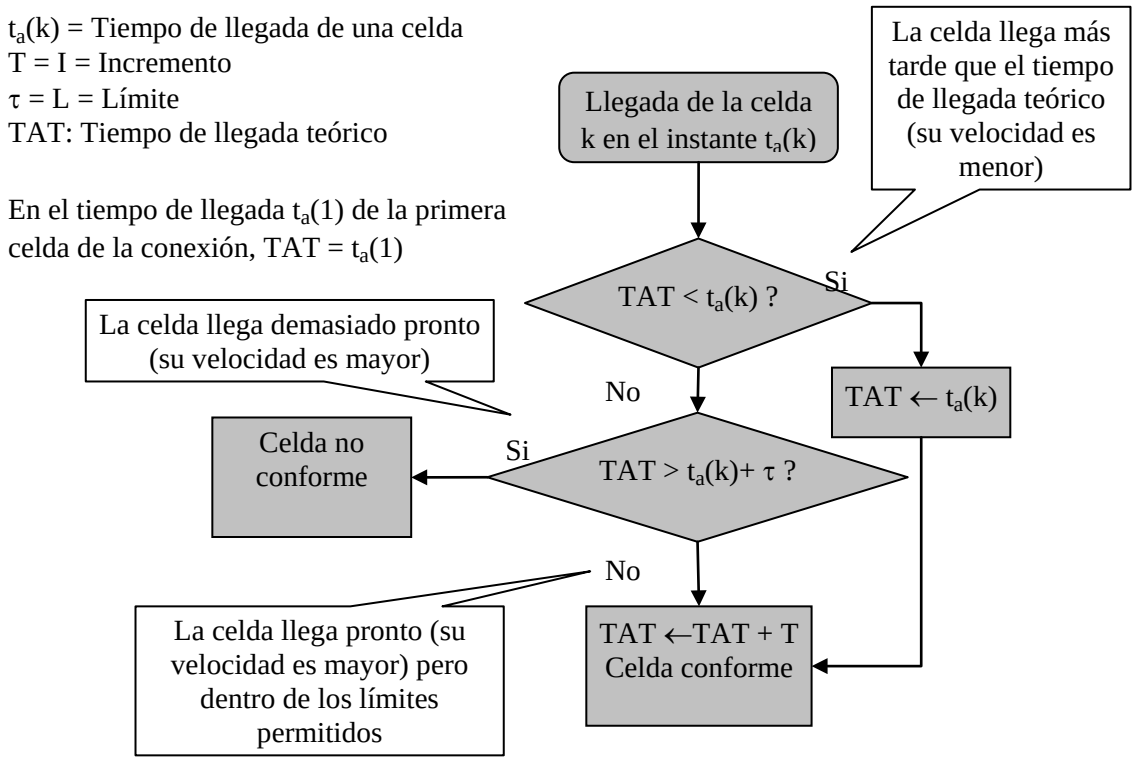

Figura 1. Algoritmo de planificación virtual

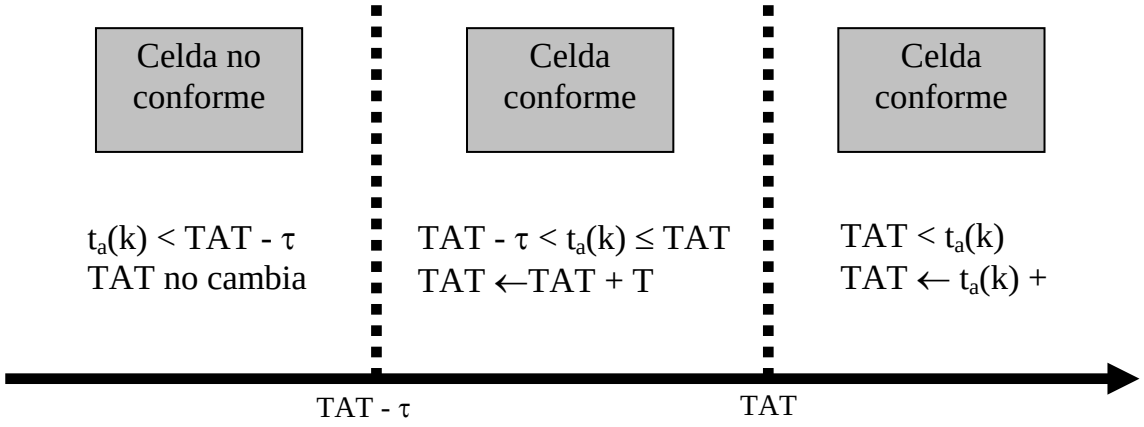

Figura 2. Instantes de llegada de celdas en el algoritmo de planificación virtual

Ahora veamos el algoritmo GCRA(T, τ) como Leaky Bucket. El algoritmo mantiene un contador X con la cantidad de datos enviados. El contador es decrementado a una velocidad constante de una unidad por unidad de tiempo hasta el valor mínimo 0; esto es equivalente a un cubo que se llena con cada celda y se vacía a velocidad constante. El contador es incrementado en I unidades cuando llega una celda, teniendo en cuenta la restricción de que I + L es el máximo valor. Cualquier celda que llega que cause que el contador exceda este valor máximo es declarada como no conforme.

En la Figura 3 se puede apreciar el diagrama de flujo del algoritmo, donde I = T y L = τ. Por tanto, la capacidad total del cubo es T + τ. Después de la llegada de la celda k, $t_a(k)$, el algoritmo mira si el cubo esta desbordado. Si es así, la celda no es conforme. Si no lo está, el cubo se incrementa.

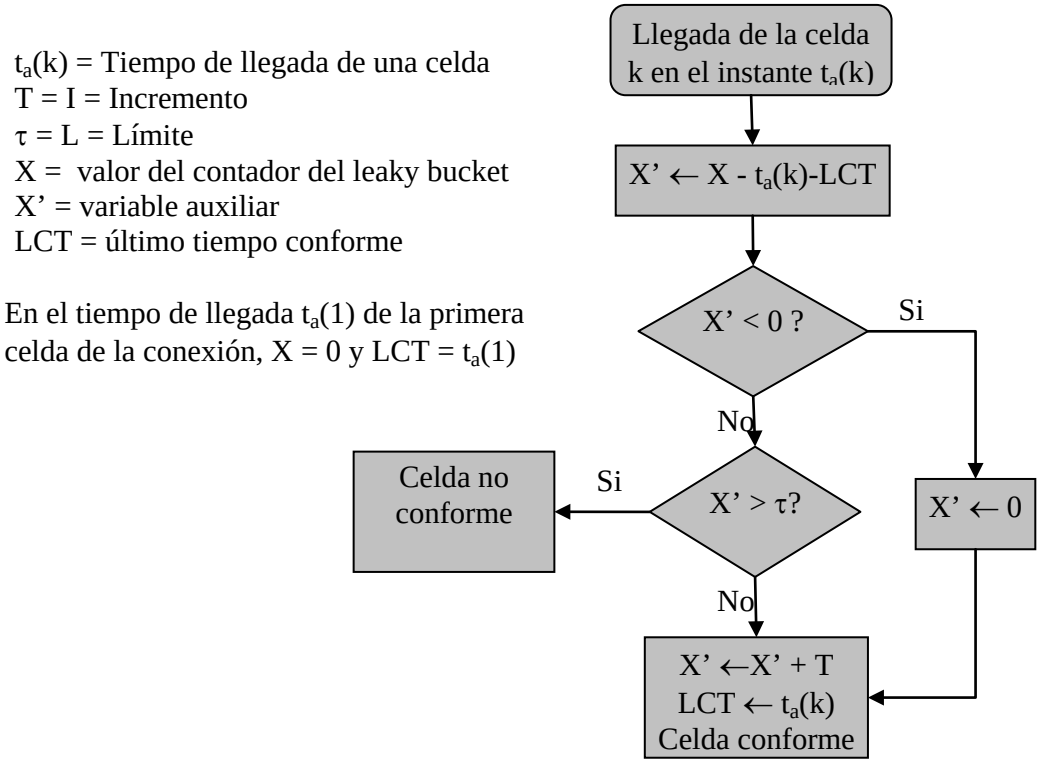

Figura 3. Algoritmo de cubo de pérdidas en tiempo contínuo

La Figura 4 ilustra la interpretación del estado del cubo: la parte de la izquierda muestra el estado del cubo después de que una celda "conforme" ha sido procesada y la parte derecha muestra el estado del cubo después de

que una nueva celda llegue. Finalmente, en la figura 5 puede observar un ejemplo en el que T = 10 y τ = 4. La celda 2 cumple sin problemas, pero la celda 3 incumple el contrato.

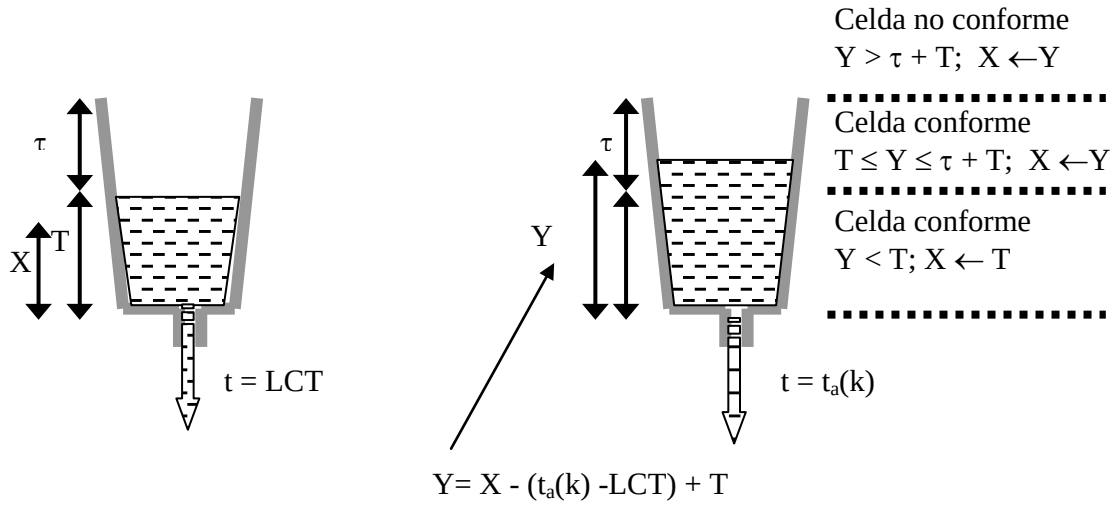

Figura 4. Representación de GCRA(T,τ) como Leaky Bucket

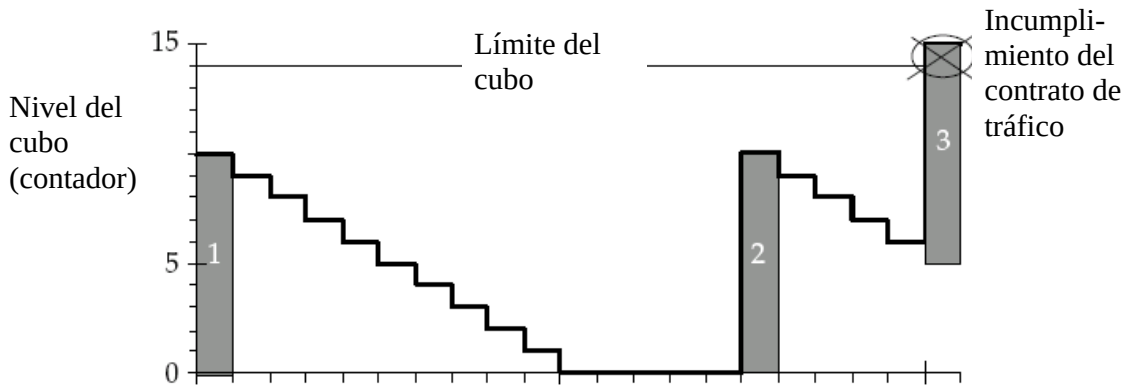

Figura 5. Ejemplo de Leaky Bucket

Es interesante señalar que, especificando un PCR (1/T) para una fuente inferior a la tasa del enlace físico, el parámetro CDVT (τ) puede permitir que un cierto número de celdas se transmitan de forma consecutiva, es decir, que se transmitan a la tasa del enlace. Esta condición ocurre cuando τ excede T - δ, siendo δ el tiempo de inserción de una celda en el enlace físico. Especificamente, para τ > T - δ, el máximo número N de celdas juntas conformes que se puede transmitir es igual a

$$N = \lfloor 1 + (\tau / (T - \delta)) \rfloor \quad (1)$$

donde $\lfloor x \rfloor$ es la parte entera de x. N es por tanto un parámetro implícito del algoritmo.

2.2 Algoritmo de régimen sostenible de celdas

El mismo algoritmo que se ha usado para definir la monitorización de la velocidad de pico también es usado para definir la monitorización de la velocidad sostenible. En este caso, dada una velocidad sostenible Rs (SCR), Ts=1/Rs es el tiempo entre llegadas de celdas a esta velocidad si no hay ráfagas. La tolerancia a las ráfagas es representada por τs. Por lo tanto, el algoritmo de la velocidad sostenible es expresado como GCRA(Ts, τs).
Para determinar τs, debemos considerar que el tamaño máximo de la ráfaga (MBS) que puede ser transmitida a la velocidad de pico es dada por:

$$MBS = \lfloor 1 + (\tau s / (T s - T)) \rfloor \quad (2)$$

donde T es el tiempo entre llegada de celdas a la velocidad de pico. Esta fórmula es análoga a (1).
Dados MBS, Ts y T , entonces τs puede ser cualquier valor en el intervalo:

$$[(MBS-1)(Ts-T), MBS\ (Ts-T)] \quad (3)$$

Por uniformidad, se emplea el valor mínimo:

$$\tau s = (MBS-1)(Ts-T) \quad (4)$$

2.3 Descarte selectivo de celdas

El algoritmo GCRA se emplea para asegurar el cumplimiento del contrato de tráfico que se ha negociado. La estrategia más sencilla consiste en que las células que cumplen el contrato pasan y las que no cumplen el contrato no pasan, es decir son descartadas. Otra opción es usar un descarte selectivo, que consistiría en que las celdas que no cumplen el contrato no son descartadas, sino sólo "marcadas" como celdas de menor prioridad. Para marcar una celda como "descartable" se cambia un bit de la cabecera de la celda ATM, el bit CLP (*Cell Loss Priority*). Cuando una celda cumple el contrato, se mantiene su bit CLP = 0. Si la celda no lo cumple, se cambia CLP = 1. Por tanto, cuando la red detecte congestión, comenzará a descartar primero las celdas con bit CLP = 1. Por ejemplo, en el buffer de salida se puede configurar un umbral de ocupación a partir del cual empiecen a descartarse celdas con CLP = 1.

2.4 Conformación de tráfico

La conformación de tráfico o *traffic shaping* se utiliza para suavizar el flujo de tráfico y reducir el agolpamiento de celdas. Un planteamiento sencillo para la conformación de tráfico consiste en emplear el cubo de fichas o *token bucket*. La Figura 6 muestra el principio básico del token bucket. Las celdas que llegan desde la fuente son colocadas en un buffer que tiene una capacidad máxima de k celdas. Los "tokens" son generados a una velocidad ρ por segundo y son colocados en el buffer que tiene una capacidad máxima de β tokens. Para cada celda transmitida a través del servidor, se debe borrar un token. Si el buffer está vacío, la celda debe esperar hasta que haya un nuevo token. Tenga en cuenta que se debe realizar el traffic shaping por cada VCC.

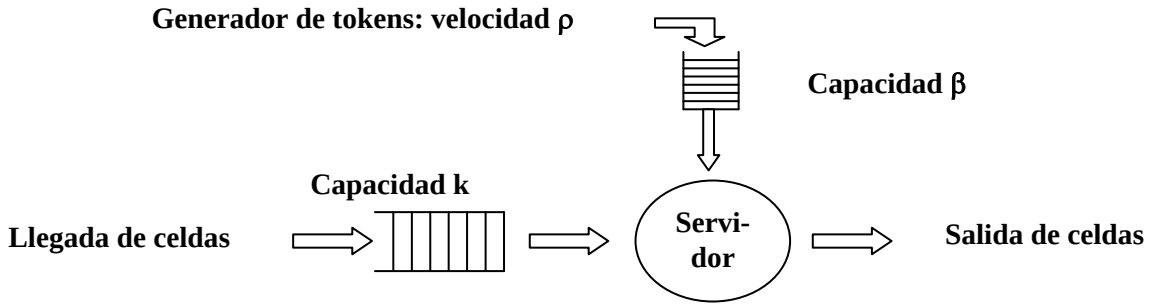

Figura 6. Ejemplo de Token Bucket

3 Código proporcionado

3.1 Fuentes

Los módulos *Fuente* son los encargados de generar tráfico de tasa de bits variable (VBR). Para la generación de este tipo tráfico la opción escogida ha sido la de emular tráfico de voz agregado. El tráfico generado por cada fuente está compuesto por la suma del tráfico generado por N conversaciones de voz. Cada conversación de voz tiene un comportamiento ON-OFF, es decir, de dos estados. En el estado ON se generan paquetes de datos de 30 bytes cada 10 ms (TTI = *Transmission Time Interval*) y en el estado OFF no se genera nada de tráfico. La duración de los periodos ON y OFF es exponencial, y la duración total de las llamadas también.

Los parámetros del módulo fuente son los siguientes:

```
PCR: numeric,
CDVT: numeric,
SCR: numeric,
```

```
      MBS: numeric,
      clientes: numeric,
      tasa_cliente: numeric,
      tiempo_llamada: numeric,
      tiempo_medio_on: numeric,
      tiempo_medio_off: numeric,
      TTI: numeric,
      bytes_TTI: numeric,
```

Los cuatro primeros corresponden al contrato de tráfico y por tanto su valor depende de la configuración propuesta en cada escenario. Con el valor de estos cuatro parámetros la fuente genera un tipo de mensaje, llamado contrato, que se envía al switch al principio de la ejecución. El resto de parámetros definen el tráfico de la fuente, su configuración será la misma en todos los escenarios:

```
*.fuente[*].clientes = 6
*.fuente[*].tasa_cliente = 0.2
*.fuente[*].tiempo_llamada = 60
*.fuente[*].tiempo_medio_on = 3
*.fuente[*].tiempo_medio_off = 3
*.fuente[*].TTI = 0.01
*.fuente[*].bytes_TTI = 30
```

3.2 Celdas

Las celdas derivan de cMessage añadiendo dos atributos tipo entero: VCI, que identifica el VCC al que pertenece la celda (y que deberá asignarle el switch) y CLP, que indica la prioridad en el descarte, que se determinará mediante los algoritmos de UPC. La definición de las celdas se ha realizado mediante el siguiente fichero denominado *celda.msg*:

```
message Celda
{
fields:
      int VCI;
      int CLP;
};
```

Al compilar con OMNET++, este fichero da lugar a los ficheros celda_m.h y celda_m.cpp, que desarrollan la clase **Celda**, aportando métodos del tipo getVCI( ), setVCI( ) etc… (consulte el manual de OMNET++ para saber más sobre definición de mensajes).

3.3 Contrato de tráfico

El contrato de tráfico es otro tipo de mensaje derivado de cMessage, denominado Contrato. Su definición se ha realizado mediante el siguiente fichero *contrato.msg*:

```
message Contrato
{
fields:
      int PCR;
      double CDVT;
      int SCR;
      int MBS;
};
```

3.4 Sumidero

El módulo sumidero se limita únicamente a ir borrando las celdas que le van llegando. Sin embargo, en el transcurso del desarrollo del simulador tendrá que ir añadiendo código en este módulo para tomar estadísticos y presentar resultados.

4 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. Como verá, al avanzar en los escenarios, son necesarias más funcionalidades implementadas en el simulador. Esto le permitirá ir completando la memoria a medida que va incluyendo estas funcionalidades en su código.

4.1 Escenario 0. Requerimientos mínimos del simulador

Como ya se anunciaba en el apartado 1.2.1 la tarea principal en cuanto a programación consiste en la creación de un módulo simple denominado Switch. Las funcionalidades básicas del switch son las siguientes:

- El switch tendrá las siguientes puertas:
  - Vector de N puertas de entrada
  - Vector de N puertas de salida
  - Enlace troncal de entrada
  - Enlace troncal de salida
- El switch recibirá celdas de N fuentes a través del vector de N puertas de entrada. Las celdas porcedentes de una puerta de entrada se considerarán pertenecientes a un VCC (*Virtual Circuit Connection*), y por tanto deberá identificarlas con su VCI (*Virtual Circuit Identifier*). Cuando el switch recibe celdas de una fuente su objetivo es multiplexarlas a través del enlace troncal de salida, tras haber aplicado los algoritmos de control de tráfico en cada flujo de celdas. El switch dispone de una cola de salida para este enlace que, en principio será de capacidad infinita.
- Al principio de la simulación, las fuentes conectadas al switch le enviarán un mensaje tipo "contrato", en el que comunican al switch sus descriptores de tráfico: PCR, SCR, MBS y CDVT. El switch debe almacenar dichos descriptores para realizar las tareas de control de tráfico.
- Cuando un switch recibe celdas a través del enlace troncal de entrada, deberá reenviar las celdas a través de la puerta de salida correspondiente (del vector de N puertas de salida) en función del VCI de la celda.
- Una característica muy importante de ATM es que los enlaces físicos están constantemente enviando celdas, lleven éstas información o no. Si en un instante de envío no hay información que enviar, el equipo ATM inserta en el canal una celda vacía denominada Empty Cell Slot, como puede apreciarse en la figura 8. La consecuencia de esto es que las celdas no pueden enviarse en cualquier momento, sino que deben enviarse en instantes concretos de tiempo. Esto es aplicable tanto al troncal como al vector de puertas de salida. En el código del módulo fuente tiene un ejemplo de cómo implementar el envío de celdas.

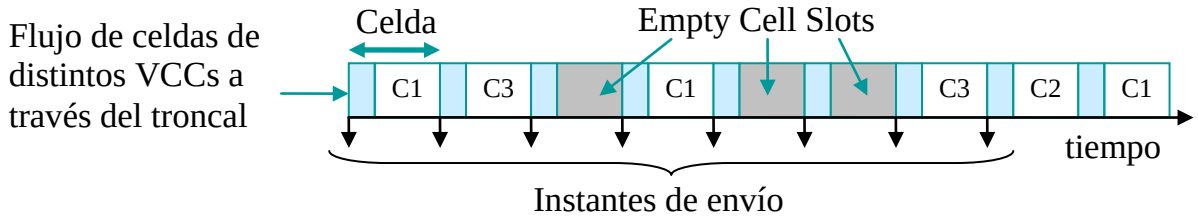

Figura 8. Envío de celdas por un interfaz

- Los escenarios que se plantearán tienen todos básicamente la misma topología, que se ilustra en la figura 9. Por un lado tendremos N fuentes conectadas a un switch que multiplexa los flujos de celdas a través de un troncal (switch multiplexor) y los envía a un switch que demultiplexa los flujos (demultiplexor) para entregarlos a los módulos sumidero. El switch multiplexor es el que debe aplicar los algoritmos de control de tráfico sobre cada uno de los VCCs.

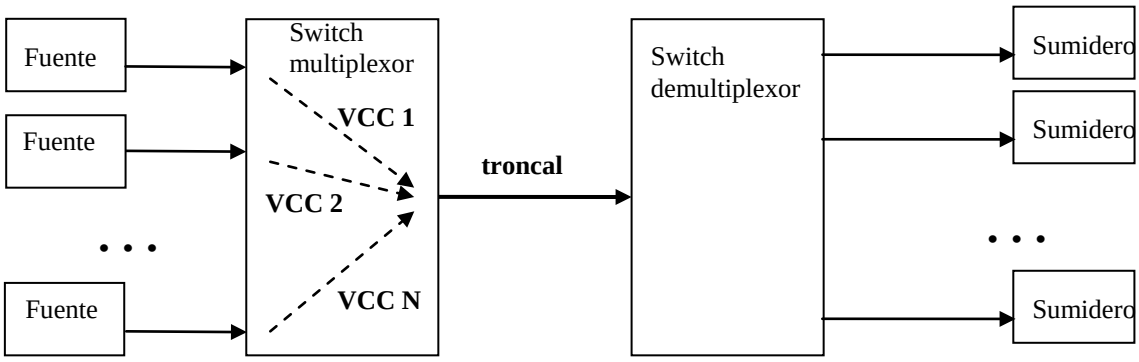

Figura 9. Topología típica

Observará que siempre habrá un switch multiplexor y un switch demultiplexor. Si cree que le puede resultar más sencillo, puede definir dos tipos de módulo switch: multiplexor y de multiplexor.

4.2 Escenario 1: Doble Leaky-Bucket. Una sola conexión

En este primer escenario se contabilizará el número medio de celdas, en porcentaje, que no cumplen los parámetros del contrato en cada uno de los algoritmos de control de tráfico. La topología del simulador es la de la Figura 10, donde se han representado las tasas, en bit/s de cada enlace.

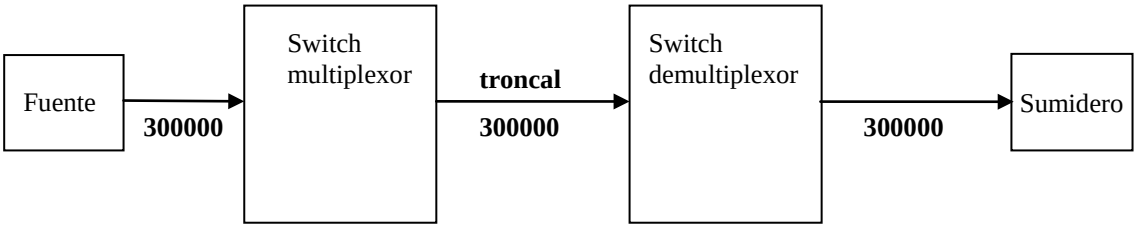

Figura 10. Topología escenario 1

La expresión doble Leaky-Bucket se refiere a la implementación conjunta de los algoritmos UPC. Al flujo de celdas entrante se le hace pasar primero por el algoritmo de régimen máximo de células y, posteriormente, a las células no descartadas (o con CLP = 0) se les aplica el algoritmo de régimen sostenible de celdas. Puede verse, por tanto como dos *leaky bukets* en cadena, a veces también denominado *Dual Leaky Bucket* ó *The Leaky Cup and Saucer*.

Las medidas a realizar en este escenario son las siguientes:

a.) Mida el porcentaje de celdas descartadas (o marcadas con CLP = 1) en el algoritmo de régimen máximo configurando los siguientes parámetros:

SCR = 300 (celdas/s), PCR = 600 (celdas/s), MBS = 2000 celdas
El número de celdas consecutivas permitidas N = 2, 4 y 6

b.) Mida el porcentaje de celdas descartadas (o marcadas con CLP = 1) en el algoritmo de régimen sostenible con las siguientes configuraciones:

SCR = 300 (celdas/s), PCR = 600 (celdas/s), número de celdas consecutivas permitidas N = 4
MBS = 50, 500, 1000 y 2000 celdas

4.3 Escenario 2: Doble Leaky-Bucket. Varias conexiones

En adelante la topología será la mostrada en la figura 11, con cinco fuentes conectadas al switch multiplexor por sus respectivos enlaces de 300000 bits/s cada uno.

En este escenario no tendrá que programar funcionalidades adicionales a las del apartado anterior. Tan sólo tendrá que introducir el código necesario en el módulo *sumidero* para tomar las muestras solicitadas. Asegúrese de que las celdas que no cumplen (o con CLP = 1) deben ser descartadas antes de ser enviadas al buffer del troncal.

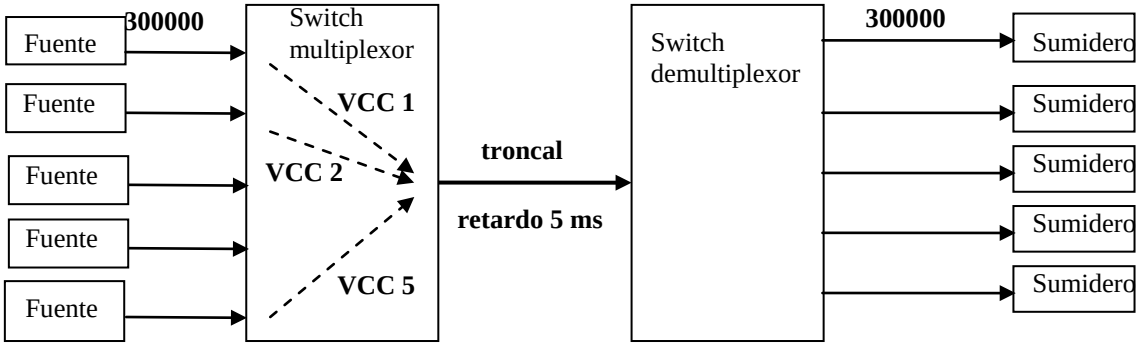

Figura 11. Topología con 5 fuentes

Debe tomar las siguientes medidas:

- Retardo medio experimentado por las celdas (desde que entran al switch multiplexor hasta que llegan al destino).
- Porcentaje de celdas descartadas
- Ocupación media de la cola de del switch multiplexor.
- Ocupación máxima de la cola del switch multiplexor.
- Porcentaje de celdas que superan 15 ms de retardo
- Histograma del retardo de uno de los VCCs

La configuración será la siguiente:

SCR = 300 (celdas/s), PCR = 600 (celdas/s), celdas consecutivas N = 4, MBS = 2000 celdas.
Hará las medidas para las siguientes tasas del troncal (al que debe asignar un retardo de 5 ms):

- 650000 bit/s
- 800000 bit/s
- 1000000 bit/s
- 1500000 bit/s

4.4 Escenario 3: Doble Leaky-Bucket y Descarte Selectivo. Varias conexiones

Con la misma topología de la Figura 11, se implementará un descarte selectivo que consistirá en determinar un umbral de descarte en la cola de salida del switch multiplexor. Cuando se alcance dicho umbral, se buscará una celda en el buffer con CLP = 1 y se descartará (preferiblemente la celda más antigua). Lógicamente las celdas que no cumplan no deben ser descartadas en el doble leaky bucket, tan sólo marcadas con CLP = 1.

Medidas a realizar: las mismas que en el escenario 2.
Configuración: la misma que en el escenario 2, considerando un umbral de descarte de 20 celdas.

4.5 Escenario 4: Doble Leaky-Bucket, Descarte Selectivo y Traffic Shaping. Varias conexiones

Finalmente incorporará la funcionalidad de token bucket para realizar traffic shaping en cada flujo VCC antes de enviar las celdas al buffer del troncal. El token bucket sólo se aplica a las celdas con CLP = 0, es decir sólo estas celdas consumen tokens. Las colas de celdas del traffic shaping se supondrán infinitas.

La configuración será la misma del escenario 3, fijando la tasa del troncal a 800000 bit/s.
Tasa de tokens (ρ) = 350 tokens/s, la capacidad del "cubo" de tokens (β) = 50.

Las medidas a realizar serán:

- Porcentaje de celdas descartadas
- Porcentaje de celdas que superan 15 ms de retardo
- Ocupación máxima de la cola del switch multiplexor.

Finalmente representará una gráfica con la evolución, a lo largo de una simulación, del throughput (en bits/s) de un VCC a la entrada del switch y a la salida del *traffic shaper*.

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.
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 empleé un fichero debe cerrarlo.

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

5.2 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.

5.3 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.

5.4 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 switch 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.

5.5 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 y a cada entidad. Debe saber que las fuentes emplean tantos números aleatorios como indique su parámetro clientes. Como en todos los escenarios tenemos 6 clientes por fuente, y en varios hay 5 fuentes en total, debe hacer un “mapeo” de generadores como el que se muestra a continuación:

```
num-rngs= 30
Red.fuente[0].rng-0 = 0
Red.fuente[0].rng-1 = 1
Red.fuente[0].rng-2 = 2
Red.fuente[0].rng-3 = 3
Red.fuente[0].rng-4 = 4
```

```
Red.fuente[0].rng-5 = 5
Red.fuente[1].rng-0 = 6
Red.fuente[1].rng-1 = 7
Red.fuente[1].rng-2 = 8
Red.fuente[1].rng-3 = 9
Red.fuente[1].rng-4 = 10
Red.fuente[1].rng-5 = 11
Red.fuente[2].rng-0 = 12
Red.fuente[2].rng-1 = 13
```
etc …
Hasta completar los 30 generadores requeridos. En cada ejecución OMNET++ asigna las semillas a cada generador. Consulte cómo y compruebe cómo puede cambiar estas semillas.

5.6 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.

5.7 Uso de la librería de plantillas STL

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

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

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

```
#include <vector.h>

vector<Figura*> formas;
formas.push_back(new Cuadrado());
formas.push_back(new Triángulo());
```

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

También puede utilizar los iteradores:

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

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

6 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.**

6.1 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.
- Indice
- 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 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.

6.2 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 0	1
Escenario 1.	3
Escenario 2.	2
Escenario 3.	2
Escenario 4.	2

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.

6.3 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 0	1
Escenario 1.	4
Escenario 2.	3
Escenario 3.	1,5
Escenario 4.	0,5

Grupos de tres personas:

Apartado	Puntuación máxima
Escenario 0	0
Escenario 1.	3
Escenario 2.	2
Escenario 3.	2
Escenario 4.	2

Para conseguir el punto que falta, los grupos de 3 personas tendrán que realizar un apartado extra que consiste en evaluar el efecto de los parámetros β y ρ en el rendimiento de *leaky bucket*. Los alumnos decidirán qué valores probarán y qué estadísticos se mostrarán. Llamaremos a este apartado Escenario 4 extendido.

7 Bibliografía

[1] András Varga, “OMNET++ discrete Event Simulation System, Versión 3.0. User Manual.”
[2] William Stallings, “Redes e Internet de Alta Velocidad. Rendimiento y Calidad de Servicio”, 2º Edición, Prentice Hall, 2002.
[3] ATM Forum, AF-TM-0121.000, “Traffic Management Specification, Version 4.1”, 1999.
[4] ITU-T Recommendation I.371, “Traffic control and congestion control in B-ISDN”, 1997
[5] H. Deitel, “C++, Como programar”, Prentice Hall, 1999.