ALGORITMOS (M+1) MMDP Y 2 MMDP PARA EL CÁLCULO DE LA PROBABILIDAD DE PÉRDIDA DE CÉLULA EN MULTIPLEXORES ATM.

Dpto. de Ingeniería de Sistemas y Automática.

Área de Ingeniería Telemática.

Autor: Eusebio Salgado Benítez.

Director: Antonio J. Sierra Collado.

INDICE

1 OBJETIVOS DEL PROYECTO 7

2 EL MODELO MMDP 9

2.1 Fuentes ON-OFF 9

2.2 Modelo MMDP 11

2.3 El sistema de cola MMDP/1/D/K 12

2.4 Cálculo de la probabilidad de pérdida de célula. 14

3 APROXIMACIÓN (M+1) MMDP 19

3.1 Algoritmo de cálculo de la probabilidad de pérdida 21

4 IMPLEMENTACIÓN M+1 MMDP 23

4.1 Lenguaje de implementación y consideraciones generales. 23

4.2 Diagrama de flujo. 24

4.3 Estructura de librerías. 27

4.3.1 F_fichero 27
4.3.2 F_lectura 28
4.3.3 F_memoria 28
4.3.4 F_matemáticas 29
4.3.5 F_calculo_probabilidades 29

4.4 Variables y constantes principales. 30

4.4.1 Parámetros iniciales. 30
4.4.2 Variables de cálculo 32
4.4.3 Variable de control. 34

4.5 Estructura del fichero de entrada. 35

4.6 Estructura del fichero de salida. 35

5 APROXIMACIÓN 2 MMDP 37

5.1 Cálculo de la probabilidad de pérdida de célula. 39

6 IMPLEMENTACIÓN 2 MMDP 43

6.1 Lenguaje de implementación y consideraciones generales. 43

6.2 Diagrama de flujo. 45

6.3 Estructura de librerías. 47
6.3.1 F_fichero 47
6.3.2 F_lectura 47
6.3.3 F_matemáticas 48
6.3.4 F_calculo_probabilidades 49
6.4 Variables y constantes principales. 49
6.4.1 Parámetros iniciales. 50
6.4.2 Variables de cálculo 50
6.4.3 Variable de control. 51
6.5 Estructura del fichero de entrada. 52
6.6 Estructura del fichero de salida. 53
7 RESULTADOS Y ESTUDIO NUMÉRICO 55
7.1 Resultados 56
7.1.1 Sistema grande. Tráfico de voz. Variación en M. 56
7.1.2 Sistema grande. Tráfico de datos. Variación en M. 57
7.1.3 Sistema grande. Tráfico de imágenes. Variación en M. 59
7.1.4 Sistema grande. Tráfico de voz. Variación en K. 61
7.1.5 Sistema grande. Tráfico de datos. Variación en K. 62
7.1.6 Sistema grande. Tráfico de imágenes. Variación en K. 64
7.1.7 Sistema pequeño. Tráfico de voz. Variación en M. 66
7.1.8 Sistema pequeño. Tráfico de datos. Variación en M. 68
7.1.9 Sistema pequeño. Tráfico de imágenes. Variación en M. 69
7.1.10 Sistema pequeño. Tráfico de voz. Variación en K. 71
7.1.11 Sistema pequeño. Tráfico de datos. Variación en K. 72
7.1.12 Sistema pequeño. Tráfico de imágenes. Variación en K. 74
7.2 Estudio numérico. 76
8 CONCLUSIONES 81
8.1 Líneas futuras. 81
9 PLANOS DEL PROYECTO 85
9.1 IMPLEMENTACIÓN M+1 MMDP 85
9.1.1 Código del fichero principal 85
9.1.2 Cabeceras de las librerías de funciones 91
9.1.3 Código de las librerías de funciones 93
9.2 IMPLEMENTACIÓN 2 MMDP 113
9.2.1 Código del fichero principal 113
9.2.2 Cabeceras de las librerías de funciones 116
9.2.3 Código de las librerías de funciones 117

9.3 IMPLEMENTACIÓN EN MATLAB M+1 MMDP 127
9.3.1 Código del fichero principal 127
9.3.2 Código de las funciones auxiliares 129

10 LISTA DE SÍMBOLOS Y ACRÓNIMOS 133

11 FÓRMULAS Y ECUACIONES 139

12 ÍNDICE DE ILUSTRACIONES 143

13 ÍNDICE DE TABLAS 145

14 REFERENCIAS BIBLIOGRÁFICAS 147

1 OBJETIVOS DEL PROYECTO

El objetivo de este proyecto es la implementación de las aproximaciones M+1 MMDP y 2MMDP de cálculo de la probabilidad de pérdida de célula en un sistema ATM y el estudio de sus resultados comparándolos entre sí.

Comenzaremos con el estudio y caracterización del modelo MMDP, así como el desarrollo del método de cálculo de la probabilidad de pérdida usando este modelo. Éste será el capítulo 2

Una vez conocido el modelo MMDP pasaremos a analizar las dos aproximaciones que son objeto de nuestro estudio. Eso lo haremos en los capítulos 3 y 5, describiendo en ellos respectivamente la aproximación M+1 MMDP y 2MMDP junto con el método de cálculo de pérdida de células en cada uno de ellos.

Después de cada uno de los desarrollos teóricos se presentará la implementación que se ha llevado a cabo. Esto se hará en los capítulos 4 y 6.

En el capítulo 7 se hará una presentación de los resultados obtenidos, así como un estudio numérico de los mismo que nos permitirá llegar a las conclusiones que se presentarán en el capítulo 8.

Finalmente adjuntaremos algunos anexos necesarios para completar el proyecto:

- Capítulo 9: Planos del proyecto
- Capítulo 10: Lista de símbolos y acrónimos.
- Capítulo 11: Índice de fórmulas y ecuaciones.

- Capítulo 12: Índice de ilustraciones.
- Capítulo 13: Índice de tablas
- Capítulo 14: Referencias bibliográficas.

2 EL MODELO MMDP

El objetivo de este capítulo es el cálculo *teórico* de la probabilidad de pérdida de célula de un sistema de cola con un servidor de capacidad fija cargado con el tráfico generado por un número finito de fuentes ON-OFF.

Para ello, comenzaremos describiendo las fuentes ON-OFF y el sistema MMDP así como los parámetros que los caracterizan y como se modela el tráfico generado por un conjunto de fuentes ON-OFF con un sistema MMDP. Posteriormente nos centraremos en el estudio del sistema de cola que nos ocupa: MMDP/1/D/K; para finalmente desarrollar el cálculo de la probabilidad de pérdida de célula de dicho sisterma.

2.1 Fuentes ON-OFF

Asumiremos para nuestras fuentes el modelo ON-OFF con tasa determinista. Este modelo consiste en caracterizar una fuente por dos estados posibles:

- ON, el periodo de actividad durante el cual se generarán células con una tasa dada.
- OFF, el periodo de inactividad, durante el cual tendremos otra tasa de generación de células diferente (normalmente inferior en media a la del periodo ON).

La duración de los periodos de actividad e inactividad también serán generalmente variables aleatorias.

En el ámbito de este proyecto consideraremos la siguiente parametrización:

- ON, la tasa de generación de células será constante e igual a Δ células por segundo. La duración de este periodo seguirá una distribución exponencial de parámetro λ.

- OFF, durante este periodo no se generarán células (tasa 0) y la duración del periodo seguirá una distribución exponencial de parámetro μ.

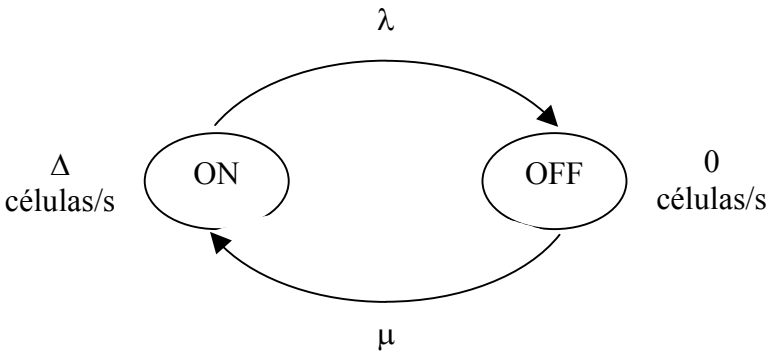


Ilustración 2-1: Diagrama de estados de una fuente ON-OFF

Otra forma alternativa de caracterizar el modelo es con los parámetros:

- Φ, tasa media.
- L, tamaño de ráfaga. Número medio de células generados durante el periodo ON.
- b, parámetro de ráfaga. Relación entre la tasa de pico (Δ) y la tasa media (Φ).

Éstos, se relacionan con los anteriores a través de las siguientes fórmulas:

$$b = 1 + \frac{\lambda}{\mu} \qquad \Phi = \frac{\Delta}{b} \qquad L = \frac{\Delta}{\mu} \qquad \textbf{(E 2-1)}$$

El hecho de que la tasa de generación de células en periodo ON sea determinista implica que estamos asumiendo que las fuentes son a ráfagas con alta correlación. Para fuentes a ráfagas con menor correlación, donde las células llegasen de forma más aleatoria, sería más conveniente usar un modelo donde la tasa de llegadas en el periodo ON siguiese una distribución de Bernoulli en vez de ser determinista.

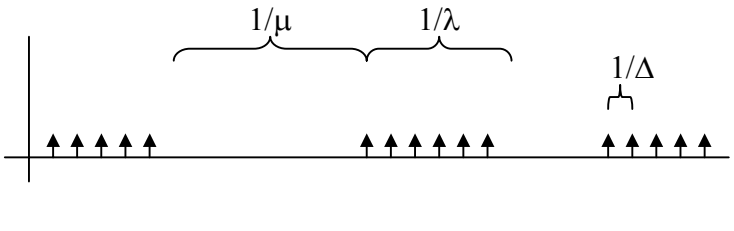


Ilustración 2-2: Ejemplo gráfico del tráfico generado por una fuente ON-OFF con la parametrización escogida.

2.2 Modelo MMDP

Usaremos el modelo MMDP (Markov modulated deterministic process / proceso determinista modulado por cadena de Markov) para modelar la superposicón de fuentes ON-OFF. El MMDP es un modelo de tráfico caracterizado por los siguientes parámetros:

- Matriz de probabilidad de transición de estados: $P = [p_{ij}]$
- Tiempo medio de permanencia en cada uno de los estados: $\Gamma = (\varphi_0, \varphi_1, \ldots, \varphi_{m-1})$
- Tasa de llegada de células en cada uno de los estados: $B = (B_0, B_1, \ldots, B_{m-1})$

Designaremos como $X(t)$ con $t \geq 0$, a la cadena de Markov; que será finita, irreducible y continua en el tiempo con un espacio de estados que designaremos por $S = \{0,1,\ldots,m-1\}$. Esta cadena de Markov tendrá una matriz de transición de estados P, matriz cuadrada de tamaño m-1) y $1/\varphi_i$ será el tiempo medio en el estado i.

El proceso MMDP es un proceso de llegada de células. $X(t)$ controlará el tiempo en cada uno de los posibles estados del proceso y las transiciones entre estos. La tasa de llegada de células en cada uno de los estados estará determinada por cada uno los elementos del vector B.

La principal diferencia de este modelo con el sistema real es que en el sistema real el tiempo entre células no tiene por qué ser constante; es más, en el sistema real podría darse el caso de que varias células llegasen en un periodo de tiempo muy corto, lo que generaría un pico en la tasa de llegada pudiendo llegar a producir una pérdida de células (*congestión a nivel de células o congestión a corto plazo*). En el sistema MMDP al ser una tasa fija B_i esto no se considera, sólo se tiene en cuenta la posible pérdida de células debida al hecho de que durante el periodo que el sistema pasa en un estado i la tasa de generación de células B_i pueda ser superior a la tasa de servicio C y por tanto llegar a desbordar la cola (*congestión a nivel de ráfaga o congestión a largo plazo*).

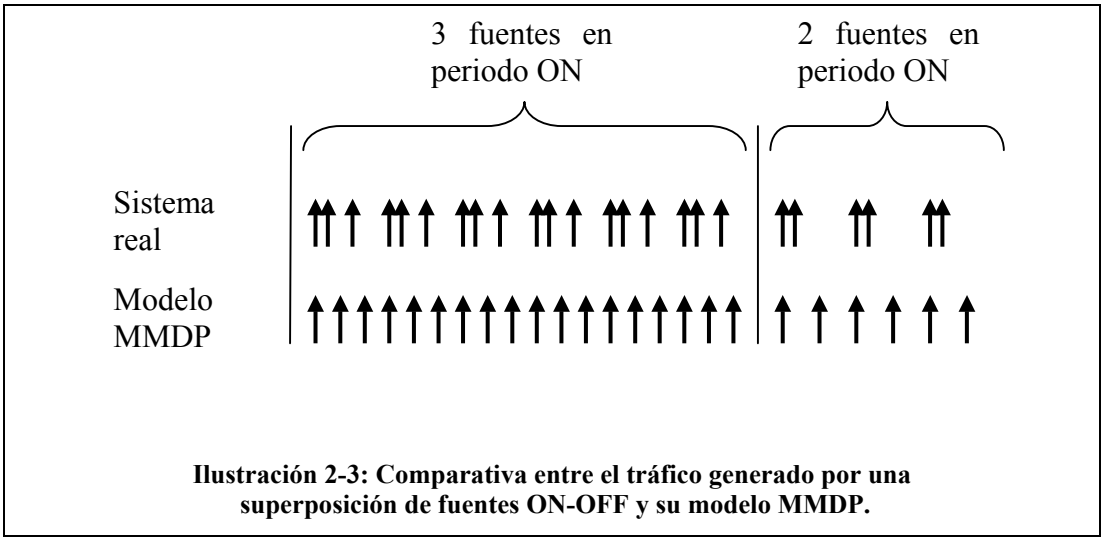


Ilustración 2-3: Comparativa entre el tráfico generado por una superposición de fuentes ON-OFF y su modelo MMDP.

En el modelo de tráfico que nos ocupa estamos considerando fuentes individuales ON-OFF a ráfagas altamente correladas, en una escala temporal lo suficientemente grande. Esto hace que la congestión a nivel de ráfaga sea una buena estimación de la congestión del sistema y que por tanto el uso del modelo MMDP sea adecuado.

2.3 El sistema de cola MMDP/1/D/K

Una vez que hemos descrito las fuentes ON-OFF y la forma en que el MMDP modela la superposición de estas, pasaremos a estudiar específicamente el sistema de cola que nos ocupa.

Consideraremos un sistema de cola con K posiciones ($K \geq 1$) con un solo servidor alimentado por el tráfico generado por una superposición de M fuentes ON-OFF modeladas por un sistema MMDP de m estados de parámetros P, Γ, B. Definiremos como $X(t)$ el estado del sistema MMDP y como $Y(t)$ el número de células en la cola (incluyendo la que estuviese siendo servida) en el momento t.

Con vistas a simplificar la notación haremos un cambio de variables. Definiremos ξ_n con $n = \{1,2,\ldots\}$ y $\xi_0 \equiv 0$, como los diferentes momentos de cambio de estado del sistema MMDP. De forma que $Y_n \equiv Y\left(\zeta_n^+\right)$ y $X_n \equiv X\left(\zeta_n^+\right)$.

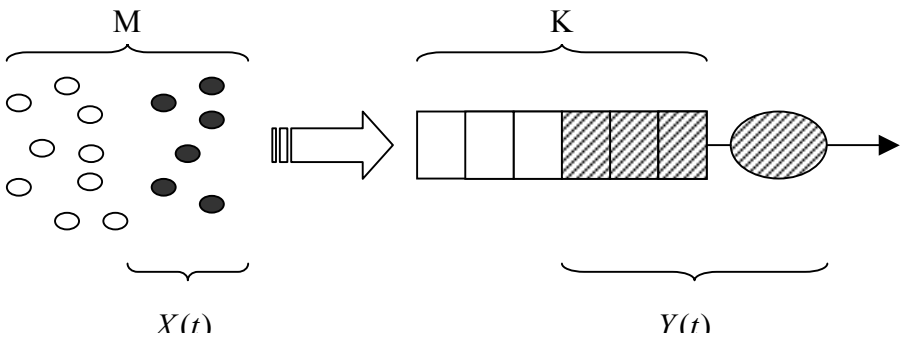


Ilustración 2-4: Sistema de cola MMDP/1/D/K

Visto de esta manera el sistema a analizar sería $\{(X_n, Y_n), n = 0,1,\ldots\}$. Este sistema es no-markoviano y por tanto su análisis es muy complicado. Con vistas a simplificarlo asumiremos dos hipótesis razonables en sistemas ATM de banda ancha:

- *Repetición de servicio:* Si el cambio de estado del sistema se diera durante el tiempo de servicio de una célula, éste se interrumpe y se procede a retransmitir completamente la célula tras el cambio de estado. Puesto que el tiempo de restante de servicio es cómo máximo $1/C$ (donde C es la tasa de servicio) y éste normalmente es despreciable frente a la duración del estado, esta hipótesis no afecta sensiblemente a los resultados. Asumiendo esta hipótesis transformamos el sistema en markoviano.

- *Hipótesis de continuidad:* Consideraremos U_i como el tiempo de permanencia del sistema en el estado i. U_i es por tanto una variable aleatoria de distribución exponencial con parámetro φ_i. Teniendo en cuenta la hipótesis de repetición del servicio y que el tiempo entre llegadas de células y de servicio son despreciables frente a U_i (situación típica en redes de alta velocidad) podemos llegar a la conclusión de que si $B_i > C$ entonces $Y_{n+1} = \min\{K; [Y_n + (B_i - C)U_i]\}$; es decir, si la tasa de llegada de células es mayor que la tasa de servicio el tamaño de la cola se incrementará a velocidad $B_i - C$ pudiendo llegar a desbordar dependiendo del estado inicial (Y_n) y del tiempo de permanencia en este estado (U_i). Si por el contrario $B_i < C$ entonces $Y_{n+1} = \max\{0; [Y_n - (C - B_i)U_i]\}$; es decir, tendremos la situación contraria, el tamaño de la cola irá decrementándose pudiendo llegar a vaciarse por completo.

Aunque el análisis del sistema con sólo la hipótesis de repetición del servicio es posible, la solución es complicada y computacionalmente ineficiente. La hipótesis de continuidad nos permitirá simplificar enormemente el análisis ya que las distribuciones $[(B_i - C)U_i]$ y $[(C - B_i)U_i]$ son de tipo geométrico y fácilmente calculables.

2.4 Cálculo de la probabilidad de pérdida de célula.

Definiremos $q_{(i,k),(j,h)}$ como la probabilidad de transición del sistema del estado (i, k) al estado (j, h). De modo que:

$$q_{(i,k),(j,h)} = P\{Y_{n+1} = h \,|\, X_n = i, Y_n = k\} p_{i,j} = a^i_{k,h} p_{i,j} \qquad \textbf{(E 2-2)}$$

Donde $p_{i,j}$ es un elemento de la matriz P de transición de estados del sistema MMDP y hemos definido $a^i_{k,h} \equiv P\{Y_{n+1} = h \,|\, X_n = i, Y_n = k\}$.

Si definimos las matrices $A_i = \left[a^i_{k,h}\right]$ y $Q = \left[q_{(i,k),(j,h)}\right]$; entonces podríamos particionar la matriz Q en bloques de (K+1) x (K+1) de la forma:

$$Q = \begin{bmatrix} 0 & p_{0,1}A_0 & \cdots & p_{0,m-2}A_0 & p_{0,m-1}A_0 \\ p_{1,0}A_1 & 0 & \cdots & p_{1,m-2}A_1 & p_{1,m-1}A_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ p_{m-2,0}A_{m-2} & p_{m-2,1}A_{m-2} & \cdots & 0 & p_{m-2,m-1}A_{m-2} \\ p_{m-1,0}A_{m-1} & p_{m-1,1}A_{m-1} & \cdots & p_{m-1,m-2}A_{m-1} & 0 \end{bmatrix} \qquad \textbf{(E 2-3)}$$

Sabemos que para el caso $C > B_i$ la variable aleatoria $[(C - B_i)U_i]$ tiene una distribución geométrica sobre enteros positivos de parámetro $\rho_i = e^{-\varphi_i / (C - B_i)}$ (debido a que U_i tiene una distribución exponencial de parámetro φ_i); lo que equivale a:

$$P\{[(C - B_i)U_i] = k\} = \rho_i^k (1 - \rho_i) \qquad \text{para k=0, 1, ...}$$

El caso para $C < B_i$ es análogo teniendo en cuenta que la velocidad de llenado del buffer es $B_i - C$ (donde antes la velocidad de vaciado del buffer era $C - B_i$) de modo que fácilmente puede deducirse:

$$P\{[(B_i - C)U_i] = k\} = \rho_i^k (1 - \rho_i) \qquad \text{para k=0, 1, ... con } \rho_i = e^{-\varphi_i / (B_i - C)}$$

Teniendo en cuenta esto y aplicando la hipótesis de continuidad podemos calcular $a^i_{k,h}$ diferenciando entre los tres posibles casos de relación entre C y B_i:

- $C > B_i$ (lo que lleva a $\rho_i = e^{-\varphi_i / (C - B_i)}$)

$$a^i_{k,h} = \begin{cases} 0 & k < h \\ \rho_i^{k-h}(1-\rho_i) & k \geq h \geq 1 \\ \rho_i^k & k \geq h = 0 \end{cases} \quad \text{para } C > B_i \qquad \textbf{(E 2-4)}$$

Estaríamos en el caso de que la cola fuera vaciándose ya que la tasa de servicio es mayor que la tasa de llegada de células. Esto nos lleva a que no existe la posibilidad de que acabemos con más células de las que teníamos inicialmente (caso 1, h no puede ser mayor que k); y la posibilidad de vaciar completamente la cola es el caso 3 (h llegaría a ser cero).

- $C < B_i$ (lo que lleva a $\rho_i = e^{-\varphi_i / (B_i - C)}$)

$$a^i_{k,h} = \begin{cases} 0 & h < k \\ \rho_i^{h-k}(1-\rho_i) & k \leq h \leq K \\ \rho_i^{K-k} & k \leq h = K \end{cases} \quad \text{para } C < B_i \qquad \textbf{(E 2-5)}$$

Estaríamos en el caso de que la cola fuera llenándose ya que la tasa de servicio es menor que la tasa de llegada de células. Esto nos lleva a que no existe la posibilidad de que acabemos con menos células de las que teníamos inicialmente (caso 1, k no puede ser mayor que h); y la posibilidad de llenar completamente la cola es el caso 3. (K es el tamaño máximo de la cola).

- $C = B_i$

$$a^i_{k,h} = \begin{cases} 1 - e^{-\gamma_i / C} & h = k = 0 \\ e^{-\gamma_i / C} & k = 0, h = 1 \\ 1 & k = h > 0 \\ 0 & \textit{en otro caso} \end{cases} \quad \text{para } C = B_i \qquad \textbf{(E 2-6)}$$

Si $Y_n = 0$; Y_{n+1} puede ser 0, en el caso de que el tiempo en el estado fuera menor que $1/B_i$, con lo que no daría tiempo a que llegase ninguna célula; o 1, en caso contrario. Si por el contrario $Y_n > 0$ entonces $Y_{n+1} = Y_n$ en cualquier caso.

Definiremos la probabilidad límite:

$$\pi_{i,j} = \lim_{n \to \infty} P\{X_n = i, Y_n = j\}$$

que puede ponerse en forma de vector (probabilidad del estado i):

$$\pi_i \equiv (\pi_{i,0}, \pi_{i,1}, \pi_{i,2}, \ldots \pi_{i,K})$$

Puede demostrarse que $\{(X_n, Y_n), n \geq 0\}$ es un proceso ergódico y por tanto π_i con $i = 0,1,\ldots m-1$ está determinado unívocamente por las ecuaciones de balance:

$$\pi_i = \sum_{j \neq i} p_{j,i} \pi_j A_j \qquad i = 0,1,\ldots m-1$$
$$\sum_{i=0}^{m-1} \pi_i e^T = 1 \qquad \textbf{(E 2-7)}$$

(e^T sería el vector unitario traspuesto de la dimensión apropiada)

Éste sería un sistema lineal de ecuaciones susceptible de ser resuelto de forma exacta por alguno de los muchos métodos numéricos de resolución de ecuaciones, no obstante, debemos observar que este cálculo en el caso de que m y K sean relativamente grandes y P sea una matriz densa puede resultar engorroso.

Si partimos del supuesto de que este cálculo está solucionado, podemos centrarnos en el cálculo de la probabilidad de pérdida de célula a partir de las probabilidades de estado.

Partiendo de la más pura simplicidad podemos entender fácilmente que la probabilidad de pérdida de célula tiene la forma:

$$p = \frac{\textit{Número medio de células perdidas en un periodo}}{\textit{Número medio de células que llegan en el mismo periodo}}$$

Definiremos el número de células que llegan como $N_{i,j}$ en un estado del sistema caracterizado por $Y_n = j$ y $X_n = i$. Éste es una variable aleatoria de distribución geométrica de parámetro $e^{-\gamma_i / B_i}$; como se deduce de que $N_{i,j} = [B_i U_i]$, de lo cual llegamos a que:

$$E[N_{i,j}] = \frac{e^{-\gamma_i / B_i}}{1 - e^{-\gamma_i / B_i}} \quad \text{para } B_i > 0 \qquad \textbf{(E 2-8)}$$

En el caso de que γ_i / B_i sea lo suficientemente pequeño podríamos aproximar el valor a $E[N_{i,j}] \approx B_i / \gamma_i$

El segundo término a calcular serían las células perdidas que usando una notación análoga a la anterior denominaremos como $R_{i,j}$. De nuevo tenemos la casuística de la relación entre las tasas de llegada y servicio:

- Si $B_i \leq C$ no hay células perdidas, luego $P\{R_{i,j} = 0\} = 1$.

- Si $B_i > C$; teniendo en cuenta la hipótesis de continuidad podemos ver que $R_{i,j} = \max(0,[(B_i - C)U_i] - K + j)$, de donde, para k=1,2,... $P\{R_{i,j} = k\} = P\{[(B_i - C)U_i] = K - j + k\}$

$$P\{R_{i,j} = k\} = \rho_i^{K-j+k}(1-\rho_i) \qquad \textbf{(E 2-9)}$$

Lo que nos lleva a que el número medio de células perdidas sea:

$$E[R_{i,j}] = \frac{\rho_i^{K-j+1}}{1-\rho_i} \qquad \text{para } B_i > C \text{ con } \rho_i = e^{-\varphi_i/(B_i - C)} \qquad \textbf{(E 2-10)}$$

Recapitulando y teniendo en cuenta los resultados obtenidos llegaríamos a:

$$p = \frac{\textit{Número medio de células perdidas en un periodo}}{\textit{Número medio de células que llegan en el mismo periodo}} = \frac{\sum_{i\in V}\sum_{j=0}^{K} E[R_{i,j}]\pi_{i,j}}{\sum_{i=0}^{m-1}\sum_{j=0}^{K} E[N_{i,j}]\pi_{i,j}}$$

donde V estaría definido como $V = \{i : B_i > C\}$

$$p = \frac{\sum_{i\in V}\sum_{j=0}^{K} \frac{\rho_i^{K-j+1}}{1-\rho_i}\pi_{i,j}}{\sum_{i=0}^{m-1}\sum_{j=0}^{K} B_i/\gamma_i \,\pi_{i,j}} \qquad \textbf{(E 2-11)}$$

Como hemos visto, la única dificultad en el cálculo de la probabilidad de pérdida de célula estribaría en la resolución del sistema de ecuaciones que nos da las probabilidades de estado. Como dijimos, la resolución de éste puede ser engorroso.

Las dos aproximaciones al problema que analizaremos a lo largo de este proyecto se basan en simplificar el sistema haciendo que o bien *P* sea una matriz poco densa (caso de la aproximación M+1 MMDP) o *m* sea pequeño (aproximación 2 MMDP).

3 APROXIMACIÓN (M+1) MMDP

En el apartado anterior hemos visto que el principal problema para el cálculo de la probabilidad de pérdida de célula era la resolución del sistema de ecuaciones:

$$\pi_i = \sum_{j \neq i} p_{j,i} \pi_j A_j \qquad i = 0,1,\ldots m-1$$

$$\sum_{i=0}^{m-1} \pi_i e^T = 1$$

cuyo cálculo en el caso de que m y K fueran relativamente grandes y P una matriz densa podía resultar engorroso.

La primera aproximación al problema que vamos a ver: (M+1) MMDP; se basa en simplificar el problema haciendo que la matriz P sea tridiagonal y reduciendo por tanto la magnitud de las ecuaciones.

Lo que haremos será identificar X(t), estado del proceso modulador del sistema MMDP, con N(t), el número de fuentes activas; para aprovechar el hecho de que N(t) es un proceso de nacimiento y muerte de tasas $\lambda_i = (M-i)\lambda$ y $\mu_i = i\mu$, respectivamente.

Hecha esta identificación las probabilidades de transición de estado nos quedan como:

$$p_{i,j} = \begin{cases} (M-i)\lambda / \gamma_i & i = 0,1,...,M-1; j = i+1 \\ i\mu / \gamma_i & i = 1,2,...,M; j = i-1 \\ 0 & \textit{en otro caso} \end{cases} \qquad \text{donde } \gamma_i = (M-i)\lambda + i\mu$$

Esto nos simplifica enormemente la matriz de transición de estado Q:

$$Q = \begin{bmatrix} 0 & \alpha_0 A_0 & 0 & \cdots & 0 \\ \beta_1 A_1 & 0 & \alpha_1 A_1 & & \vdots \\ 0 & \beta_2 A_2 & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & \alpha_{M-1} A_{M-1} \\ 0 & \cdots & 0 & \beta_M A_M & 0 \end{bmatrix}$$

Las ecuaciones de balance **(E 2-7)** que nos permitirían calcular las probabilidades de estado π_i quedarían pues reducidas de la forma:

$$\pi_i = \alpha_{i-1}\pi_{i-1}A_{i-1} + \beta_{i+1}\pi_{i+1}A_{i+1} \qquad i = 0,1,\ldots M \qquad \textbf{(E 3-1)}$$

con $\alpha_{-1} \equiv 0$; $\beta_{M+1} \equiv 0$; $\alpha_i = (M-i)\lambda/\gamma_i$ y $\beta_i = i\mu/\gamma_i$ para i=0,1,...,M

Estas probabilidades de estado podrían calcularse ahora con los métodos numéricos clásicos para resolución de sistemas de ecuaciones lineales. No obstante, vamos a intentar simplificar aún más este cálculo.

1. La cadena de Markov $\{X_n, n \geq 0\}$ es un proceso de nacimiento y muerte discreto en el tiempo con probabilidades de nacimiento y muerte α_i y β_i respectivamente. Si definimos $p_i = \lim_{n\to\infty} P\{X_n = i\}$ con i=0...M; tenemos que p_i es igual a la probabilidad marginal $\pi_i e^T$ y viene dado por:

$$p_i = \frac{\alpha_{i-1}\alpha_{i-2}\ldots\alpha_0}{\beta_i\beta_{i-1}\ldots\beta_1}p_0 \qquad i = 1,2,\ldots M \qquad \textbf{(E 3-2)}$$

$$p_0 = \frac{1}{1 + \sum_{i=1}^{M} \frac{\alpha_{i-1}\alpha_{i-2}\ldots\alpha_0}{\beta_i\beta_{i-1}\ldots\beta_1}}$$

El hecho de que la probabilidad marginal $\pi_i e^T$ pueda ser calculada sin conocer π_i será la piedra angular del algoritmo que desarrollaremos más adelante.

2. Definiremos la transformación lineal:

$$y_i \equiv \pi_i A_i \qquad i > v \qquad v \equiv \max\{i : i\Delta \leq C\}$$

Recordando el valor de $a^i_{k,h}$ cuando $B_i > C$ **(E 2-5)** llegaríamos a:

$$y_{i,K} = \sum_{j=0}^{K} \pi_{i,j}\rho_i^{K-j}$$

Con esta transformación **(E 2-11)** quedaría con la forma:

$$p = \frac{\sum_{i=v+1}^{M} \frac{\rho_i y_{i,K}}{1-\rho_i}}{\sum_{i=1}^{M} p_i \frac{e^{-\gamma_i / B_i}}{1-e^{-\gamma_i / B_i}}} \qquad \textbf{(E 3-3)}$$

La importancia de esta transformación radica en que ya no es necesario calcular π_i sino y_i. Esto es más eficiente y nos acelerará el algoritmo. Como podemos ver comparando:

$$\pi_i = \alpha_{i-1}\pi_{i-1}A_{i-1} + \beta_{i+1}\pi_{i+1}A_{i+1} \qquad i = 0,1,\ldots M$$

$$y_i = (\alpha_{i-1}y_{i-1} + \beta_{i+1}y_{i+1})A_i \qquad i = 0,1,\ldots M \qquad \textbf{(E 3-4)}$$

Para el cálculo de π_i eran necesarios dos productos vectoriales, mientras que para y_i sólo es necesario uno.

Haciendo uso del conocimiento de que A_i es una matriz estocástica se puede extraer que: $y_i e^T = \pi_i A_i e^T = \pi_i e^T = p_i$ y por tanto $\sum_{i=0}^{M} y_i e^T = 1$. Lo que implica que $\{y_i\}$ tiene forma de distribución de probabilidad.

3.1 Algoritmo de cálculo de la probabilidad de pérdida

Teniendo en cuenta los dos puntos descritos anteriormente, desarrollaremos un algoritmo iterativo basado en el algoritmo de agregación-desagregación para resolución de cadenas de Markov grandes[1].

Puesto que las probabilidades $y_i e^T = p_i$ son calculables sin la necesidad de conocer y_i, no es necesario que llevemos a cabo el paso de agregación; realizaremos sólo el paso de desagregación: iremos actualizando y_i desde i=0 hasta M, usando **(E 3-4)** como en el método iterativo Gauss-Seidel, realizando además una normalización de modo que se cumpla $y_i e^T = p_i$ (previamente calculados **(E 3-2);** de este modo conseguimos que las probabilidades que deberían calcularse en el paso de agregación ($y_i e^T$) siempre estén ajustadas.

[1] P.J. Schweitzer, "A survey of aggregation-disaggregation in large Markov chains", in *Numerical Solution of Markov Chains*, W. J. Stewart, Ed New York: Marcel Dekker, 1991.

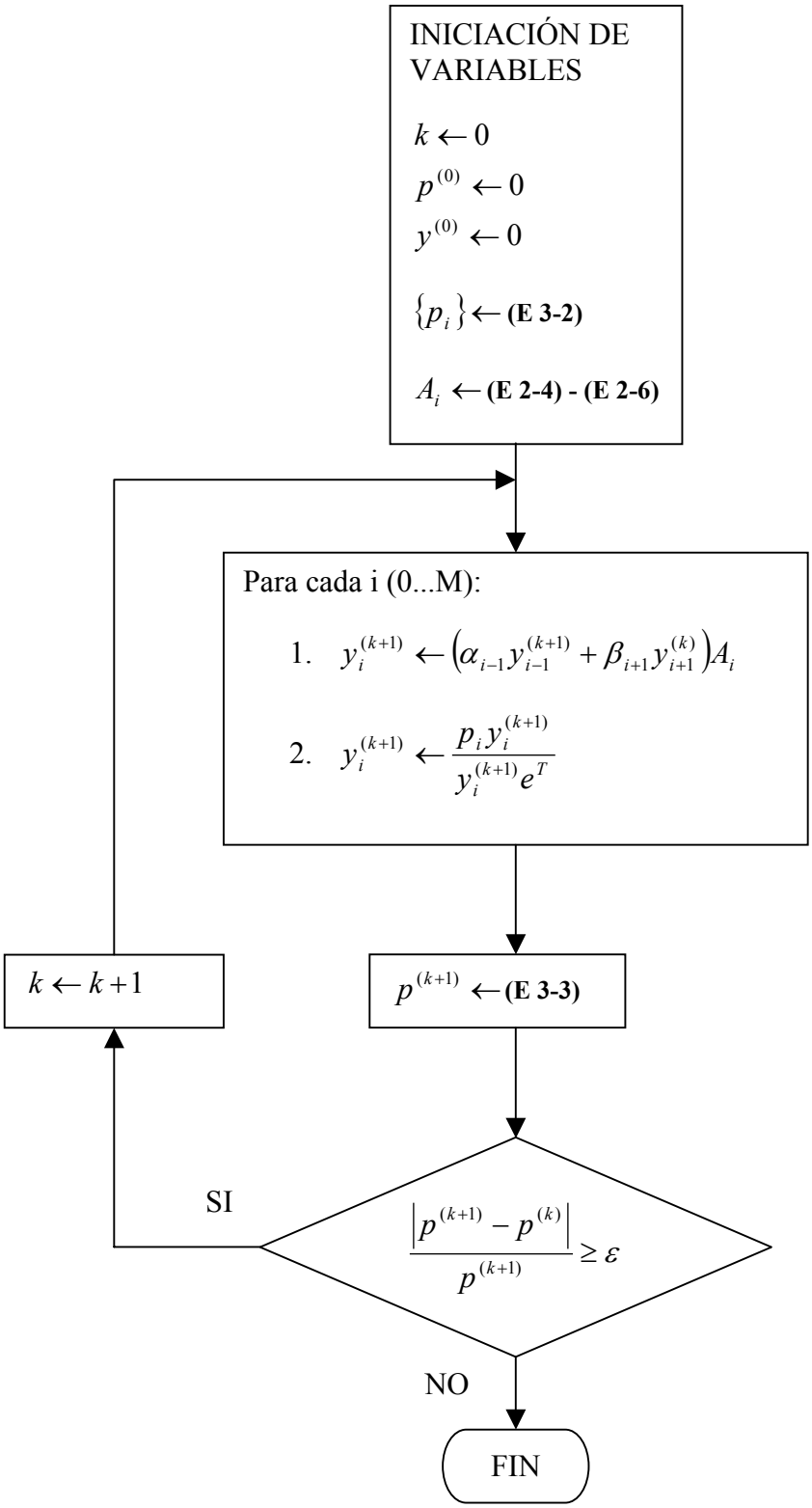


Ilustración 3-1: Diagrama de estados del algoritmo iterativo de cálculo de la probabilidad de pérdida de célula con la aproximación M+1 MMDP

4 IMPLEMENTACIÓN M+1 MMDP

A lo largo del presente capítulo vamos a ver los pasos principales que se han llevado a cabo para la implementación del cálculo de la probabilidad de pérdida de célula siguiendo la aproximación M+1 MMDP que se ha realizado en este proyecto para la obtención de resultados numéricos. El código completo puede encontrarse en el capítulo 9. PLANOS DEL PROYECTO.

4.1 Lenguaje de implementación y consideraciones generales.

La implementación del algoritmo se ha llevado a cabo en lenguaje ANSI C pensando siempre en la maximizar la portabilidad del código, para que la compilación fuera posible y sencilla en cualquier plataforma.

La ejecución del programa se realiza desde la línea de comando con el formato:

```
PROMPT> ALG_M1MMDP <Entrada> <Salida>
```

Tabla 4-1: Código, ejecución del programa.

Los parámetros <entrada> y <salida> son opcionales y corresponden con los nombres que hayamos decidido para los ficheros de entrada y salida respectivamente. En el primero de ellos es el fichero a través del cual le pasaremos al programa los parámetros que definen el sistema. El segundo es el fichero dónde se almacenarán los resultados.

Una descripción más pormenorizada de ellos y de sus estructuras se hará en los apartados 4.5 y 4.6

4.2 Diagrama de flujo.

Para la concepción de la implementación en forma de código de programa se ha seguido de forma *casi literal* el desarrollo teórico descrito en el capítulo anterior. Es por ello que el siguiente diagrama de flujo se ha creado tomando como base el de la ilustración 3-1, detallándolo.

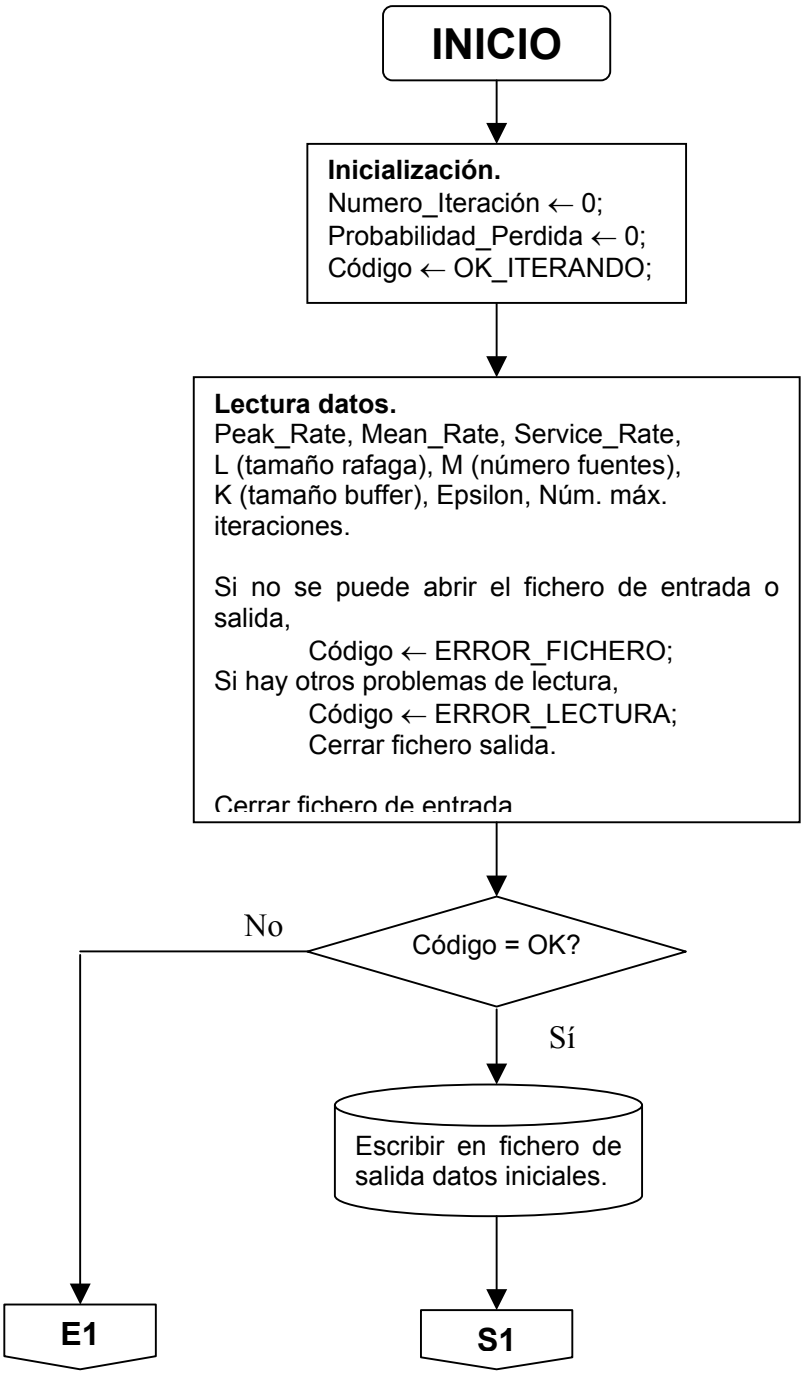


Ilustración 4-1: Diagrama de flujo de la implementación M+1 MMDP (1 de 3)

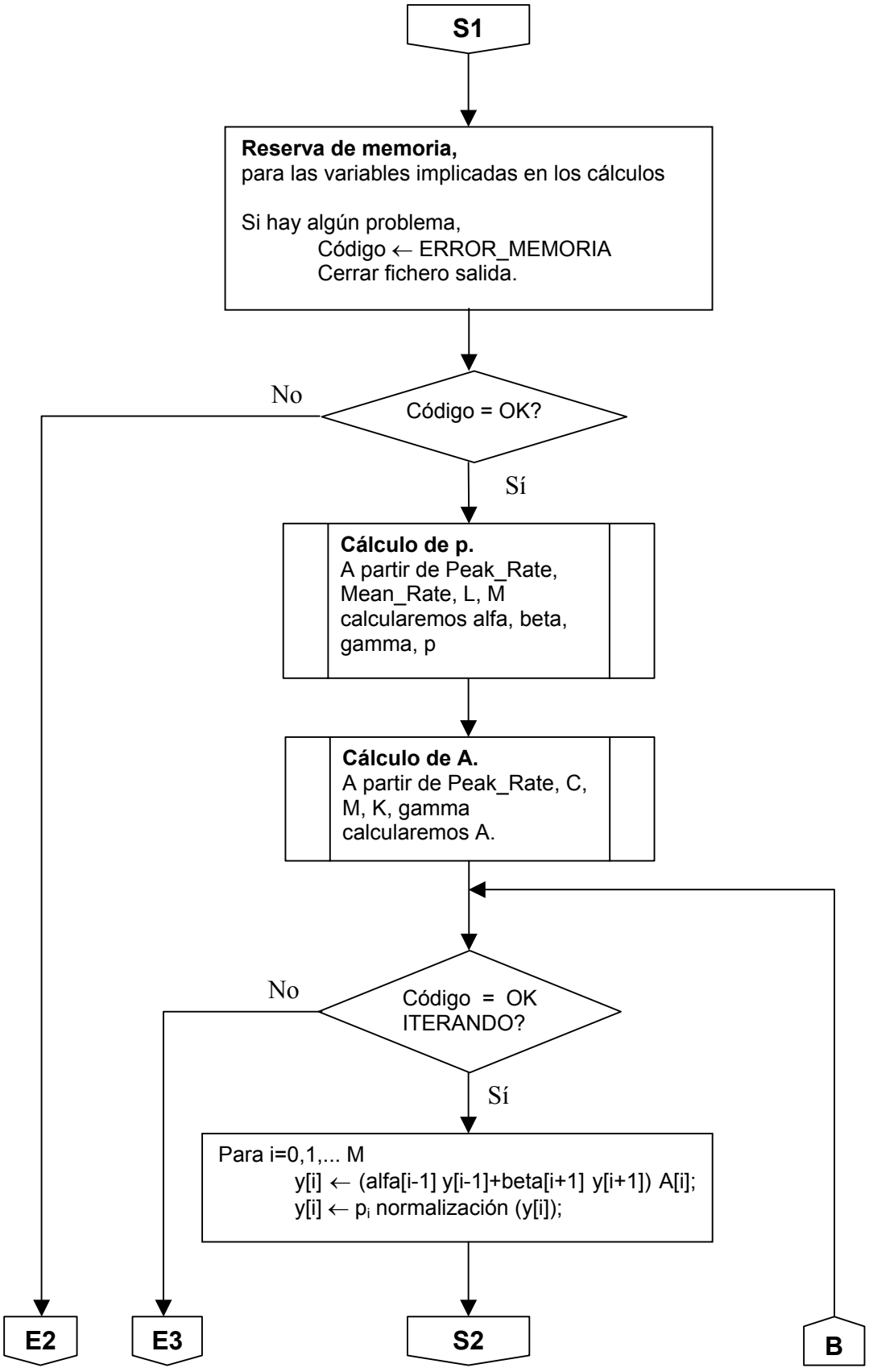


Ilustración 4-2: Diagrama de flujo de la implementación M+1 MMDP (2 de 3)

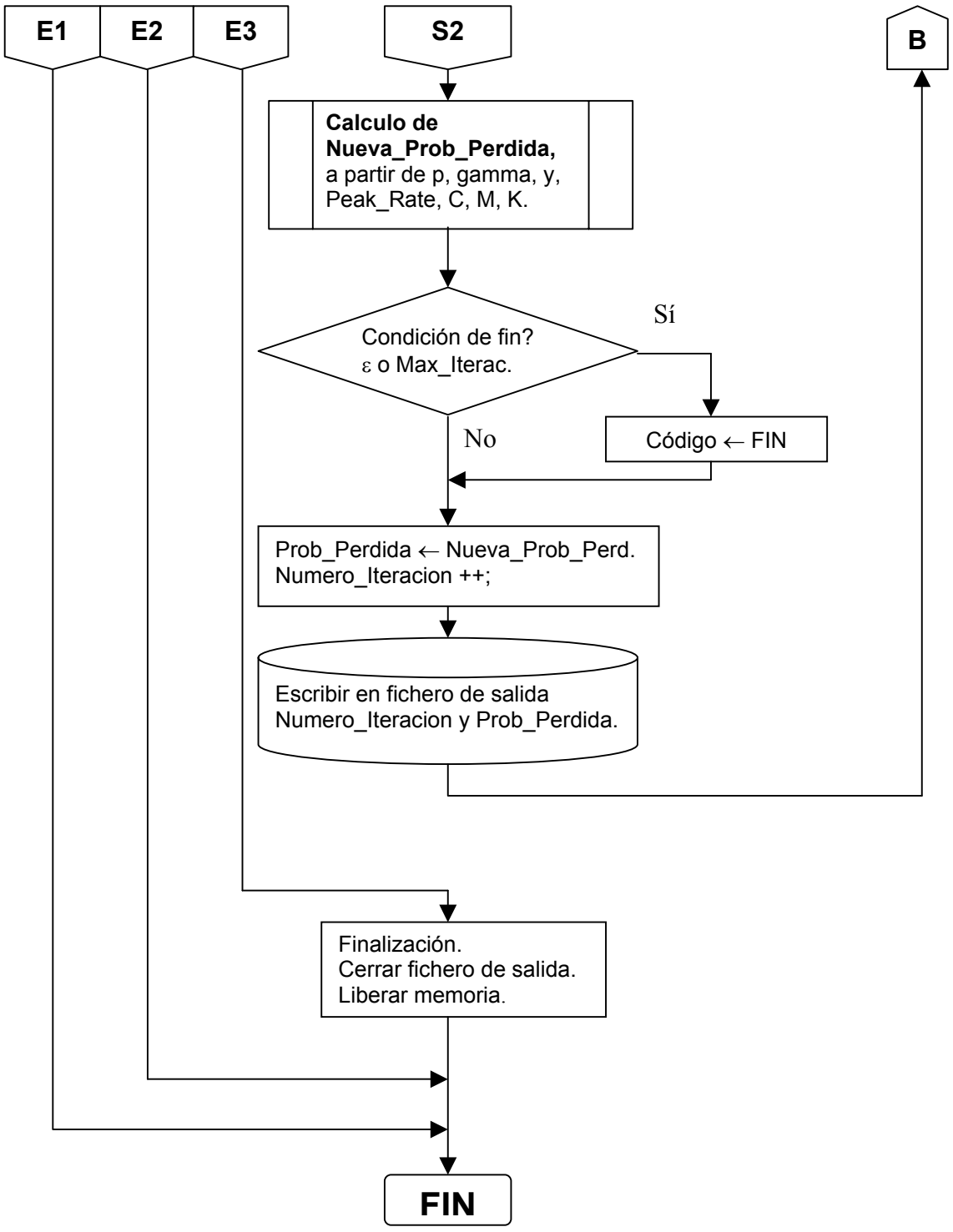


Ilustración 4-3: Diagrama de flujo de la implementación M+1 MMDP (3 de 3)

Por motivos prácticos, hemos añadido la posibilidad de fijar un número máximo de iteraciones, para evitar que la ejecución del programa pudiera prolongarse en exceso. Así, ahora, tenemos dos condiciones que pueden sacarnos del bucle de cálculo: el hecho de haber convergido a una solución (dentro de un cierto margen definido por épsilon) y/o

haber llegado al número máximo permitido de iteraciones. Estas dos condiciones pueden usarse conjuntamente, con lo que la primera que se cumpliese nos sacaría del bucle o podemos anular alguna de ellas y quedarnos sólo con la otra (creo que no hace falta destacar el hecho de que anular ambas condiciones a un tiempo no tiene sentido). Para ello en la entrada de datos del programa no hay más que darle un valor negativo a la condición (ya sea épsilon o el número máximo de iteraciones) para que ésta no sea usada.

El flujo del programa es determinado por el valor de la variable *código*. La variable *código* es un flag que nos determinará en cada momento el estado en que se encuentra el algoritmo: iterando, finalizado correctamente al encontrar una solución (o haber superado el número máximo de iteraciones) o finalizado debido a algún error.

4.3 Estructura de librerías.

Tomando como base el diagrama de flujo ilustrado en la sección anterior podemos identificar una serie de tareas susceptibles de ser agrupadas funcionalmente. Ésta es la base que se ha tomado para la concepción de las librerías de funciones que acompañan al código principal.

Áreas funcionales identificadas:

- Tareas relacionadas con los ficheros de entrada y salida: apertura, cierre, etc. (**F_fichero.c**)
- Lectura de datos: tratamiento de los comentarios, etc. (**F_lectura.c**)
- Reserva de memoria. (**F_memoria.h**)
- Funciones matemáticas: producto vectorial, productorios, normalizaciones, etc. (**F_matemáticas.c**)
- Cálculo de probabilidades. (**F_calculo_probabilidades.c**)

A continuación se describirán someramente la funciones contenidas en cada una de las librerías y su misión. Tanto los ficheros de cabeceras como el código completo puede encontrarse en el capítulo 9: PLANOS DEL PROYECTO.

4.3.1 F_fichero

Esta librería está constituida por una única función:

```
char Apertura_Fichero(int argc, char *argv[],
                      FILE **Fichero_Entrada, FILE **Fichero_Salida);
```

La misión de la cual es abrir los ficheros de entrada y salida a partir de los nombres que se le suministren en la línea de comando. Caso de que sólo se le suministre uno, considerará que se trata del nombre del fichero de entrada, también puede darse el caso de que no se le

proporcione ninguno de los dos. En caso de omisión se usarían los nombres de fichero por defecto: ENTRADA.TXT y SALIDA.TXT

Se consideró la posibilidad de implementar una función de cierre de ficheros, pero puesto que esta función no añadía nada a la ya suministrada por las librerías estándar se desechó la idea.

4.3.2 F_lectura

En esta librería dispondremos de dos funciones:

```
char leer_sin_comentarios (float * parametro, FILE * fichero);

char Lectura_Datos(float *Peak_Rate, float *Mean_Rate, float *C,

            int *L, int *M, int *K, float *epsilon,

            int *Numero_Max_Iteraciones, FILE * fichero);
```

La función principal es `Lectura_Datos` encargada de leer sucesivamente y **en orden** los parámetros de entrada al sistema del fichero de entrada (ya abierto con el uso de la función descrita en el apartado anterior).

La función `leer_sin_comentarios` es usada desde la anterior con el fin de filtrar el contenido del fichero de entrada y extraer los datos desechando los comentarios que puedan encontrarse.

4.3.3 F_memoria

```
char Reserva_Memoria(double **alfa, double **beta, double **gamma,

                    double **p, double ***y, double ****A, int M, int K,

                    double **vector1, double **vector2,double **vector3);
```

La única función contenida en esta librería es la encargada de la reserva de memoria necesaria para todas las variables que serán usadas en los cálculos: vectores, matrices, variables auxiliares, etc.

Su resultado es global, es decir, si se reserva la memoria debe ser para todas las variables. Si por el contrario en algún momento del proceso se encuentra un problema debe liberarse toda la memoria reservada hasta ese instante y devolver el código de error correspondiente.

Al igual que sucedía en la librería f_ficheros se consideró la posibilidad de implementar una función de liberación de memoria, pero puesto que esta función no añadía nada a la ya suministrada por las librerías estándar finalmente no se hizo.

4.3.4 F_matemáticas

Las siguientes son funciones de cálculo vectorial puramente matemáticas de las que se hace uso y que no se encontraban implementadas en las librerías estándar[2].

- `double Productorio (double * variable, int indice_inicial,`

  `int indice_final);`

  Nos devuelve el productorio de los elementos contenidos entre los índices inicial y final de un vector (introducido mediante la variable *variable*).

- `void Vector_x_Matriz (double *vector, double **matriz,int K,`

  `double *vector_resultado);`

  Producto vectorial entre un vector y una matriz. Se entiende que las dimensiones son: (1xK) x (KxK). Con lo que el resultado es otro vector de dimensión (1xK).

- `void Factor_x_Vector (double factor, double *vector, int K,`

  `double *vector_resultado);`

  Escalado de un vector por el factor proporcionado. Se entiende que el vector es de dimensión (1xK).

- `void Vector_mas_Vector (double *vector1, double *vector2, int K,`

  `double *vector_resultado);`

  Suma vectorial. Se entiende que los vectores son de dimensión (1xK)

- `void Normalizacion_Vector (double * vector, int K);`

  Normalización de un vector por su módulo. Se entiende que el vector es de dimensión (1xK).

- `double Valor_Absoluto(double x);`

  Devuelve el valor absoluto de un número.

4.3.5 F_calculo_probabilidades

En esta librería dispondremos de cuatro funciones, las cuales se encargarán de los cálculos de las probabilidades específicas de esta aproximación.

---

[2] Como ya se comentó en el apartado 4.1 la idea era maximizar la portabilidad del código. Es por ello por lo que las funciones matemáticas necesarias se han implementado en lugar de usar alguna de las muchas librerías de cálculo vectorial disponibles.

```
double Calculo_p0(double *alfa, double *beta, int M);

void Calculo_p(float Peak_Rate, float Mean_Rate, int L, int M,

            double *alfa, double *beta, double *gamma, double *p);

void Calculo_A(float Peak_Rate, float C, int M, int K,

      double * gamma, double ***A);

double Calculo_Probabilidad_Perdida(double *p, double *gamma, double **y,

            float Peak_Rate, float C, int M, int K);
```

La función `Calculo_p` tiene dos misiones, por un lado el cálculo de las variables auxiliares *alfa*, *beta* y *gamma* a partir de los parámetros iniciales, éstas serán usadas por el resto de las funciones de esta librería, por lo que es importante que esta sea la primera función de esta librería en ejecutarse; por el otro, el cálculo específico de *p* (distribución de probabilidad de estado de la cadena de Markov), para ello debe apoyarse en la función `Calculo_p0`, que nos calcula la probabilidad del primer estado (0).

La función `Calculo_A`, nos devolverá la matriz de probabilidades de transición de estado del buffer de la cola condicionado al estado de la cadena de Markov.

Por último la función `Calculo_Probabilidad_Perdida` implementará la función **(E 3-3)** dándonos en cada iteración un valor para la probabilidad de pérdida de célula hasta que el algoritmo converja a una solución.

4.4 Variables y constantes principales.

Siempre que ha sido posible se ha mantenido la nomenclatura usada durante el desarrollo teórico para que el código fuera lo más claro posible. Una lista exhaustiva de las variables y símbolos usados a lo largo de este documento puede encontrarse en el capítulo 10.

4.4.1 Parámetros iniciales.

`Float Peak_Rate`	Tasa de pico de una fuente ON-OFF individual (dimensión: bits/segundo).
`Float Mean_Rate`	Tasa media de una fuente ON-OFF individual (dimensión: bits/segundo).
`int L`	Longitud media de ráfaga de una fuente ON-OFF individual. Número medio de células generadas durante el periodo ON (dimensión: células).

`Flota C`	Tasa de servicio de células del sistema. Velocidad a la que las células van saliendo de la cola (dimensión: bits/segundo).
`int K`	Tamaño máximo de la cola (dimensión: células).
`Float epsilon`	Límite, que en la comparación entre dos resultados consecutivos, nos decidirá si continuar o salir del bucle de cálculo.
`int Numero_Max_Iteraciones`	Número máximo de iteraciones del algoritmo que permitiremos (parar acotar superiormente el tiempo de procesado).
`int M`	M+1 sería el número de estados del sistema MMDP. En esta aproximación en concreto podemos identificarlo también con el número de fuente ON-OFF presentes en el sistema.

Tabla 4-2: Parámetros iniciales.

En la tabla 4-2 podemos encontrar los parámetros iniciales que nos definen unívocamente el sistema a simular.

Los tres primeros (*Peak_Rate*, *Mean_Rate* y *L*) nos dan todos los parámetros para definir una fuente individual ON-OFF. *M* nos da el número de fuentes ON-OFF en el sistema; y por último *K* y *C* nos darían los parámetros que nos definirían la cola (el tamaño de la cola y la velocidad de servicio).

Epsilon y *Num_Max_Iteraciones* son los dos parámetros comparativos cuya misión es sacarnos del bucle que constituye el algoritmo iterativo. Son dos condiciones que pueden usarse de forma conjunta, o individual, para lo cual deberemos anular una de ellas. Esto se hace dándole un valor negativo.

- *Epsilon* es la máxima tolerancia que admitimos entre dos resultados consecutivos para considerar que el algoritmo ha convergido, es decir, si la diferencia entre p_i y p_{i-1} es menor o igual que *epsilon* consideraremos que hemos encontrado la solución y esta era p_i.

- *Num_Max_Iteraciones* es simplemente el máximo número de iteraciones que consideraremos, es decir, cuando el algoritmo haya realizada dicho número de iteraciones saldremos del algoritmo y daremos por bueno el último resultado obtenido.

Los parámetros iniciales se introducen en el programa a través del fichero de entrada de datos cuya estructura se detalla en el apartado 4.5.

4.4.2 Variables de cálculo.

`Double Probabilidad_Perdida = 0`	Variable en la que iremos guardando el valor de la probabilidad de pérdida calculado en la iteración anterior. Su misión es servir de patrón de comparación para determinar si el algoritmo ha convergido.
`Double N_Probabilidad_Perdida`	Variable en la que podremos encontrar el valor de la probabilidad de pérdida que calculemos en la iteración en curso.
`double **y`	Matriz cuyos elementos (vectores) se corresponden con el cambio de variable: $y_i \equiv \pi_i A_i$. Dimensión: (M+1) x (K+1).
`double *alfa`	Vector, cuyos elementos se corresponden con el cambio de variable: $\alpha_i = (M-i)\lambda / \gamma_i$ con $\alpha_{-1} \equiv 0$ para i=0,1,...,M. Dimensión: M+1.
`double *beta`	Vector, cuyos elementos se corresponden con el cambio de variable: $\beta_i = i\mu / \gamma_i$ con $\beta_{M+1} \equiv 0$ para i=0,1,...,M. Dimensión: M+1.
`double *gamma`	Vector, cuyos elementos se corresponden con el cambio de variable: $\gamma_i = (M-i)\lambda + i\mu$. Dimensión: M+1.
`double *p`	Vector, cuyos elementos se corresponden con la distribución de la probabilidad de estado de la cadena de Markov. Dimensión: M+1.

`double ***A`	Matriz de probabilidades de transición de estado del buffer de la cola condicionado al estado i, $i \in S$ de la cadena de Markov. Dimensión (M+1) x (K+1) x (K+1). Elemento de la matriz A_i $a_{k,h}^{i} = P\{Y_{n+1} = h \mid X_n = i, Y_n = k\}$ Probabilidad de transición del estado del buffer de la cola de *k* a *h,* en el estado de la cadena de markov i, $i \in S$.

Tabla 4-3: Variables de cálculo.

Las dos primeras variables nos sirven para ir comparando los distintos resultados que se van obteniendo en cada una de las iteraciones (siempre que el valor de *epsilon* sea positivo ya que como vimos antes esta condición puede anularse dándole un valor negativo). Cómo se vio en la sección 3.1, se haría una comparación no de la distancia absoluta entre ambas soluciones, sino del cambio porcentual de la calculada respecto a la anterior: Entendemos pues, implícitamente, que el valor de *epsilon,* es el porcentaje (unitario) máximo de cambio para aceptar la convergencia del algoritmo. No obstante, hemos considerado la posibilidad de que la variable `N_Probabilidad_Perdida` pudiera llegar a tener un valor nulo, con lo que esto nos provocaría un error al tratar de calcular el cambio porcentual (al dividir por cero). En caso de que esto sucediera se compararía la distancia absoluta entre ambas soluciones y se entendería que el valor de *epsilon* es la distancia máxima para aceptar la convergencia del algoritmo. Esto se ilustra en el siguiente extracto del código:

```
     if (epsilon>0)
           {
           if (N_Probabilidad_Perdida == 0)
                 {
                 if (epsilon > Valor_Absoluto
                         (N_Probabilidad_Perdida - Probabilidad_Perdida))
                       {
                       fprintf(Fichero_Salida,"\n...");
                       codigo = 1;
```

```
                        }

                  }

            else

                  {

                  if (epsilon > Valor_Absoluto
(N_Probabilidad_Perdida-Probabilidad_Perdida/N_Probabilidad_Perdida))

                        {

                        fprintf(Fichero_Salida,"\n...");

                        codigo=1;

                        }

                  }

            }
```

Ilustración 4-4: Código, comparación de dos resultados consecutivos (uso de épsilon).

El resto de las variables enunciadas en la tabla 4-3 no son más que meros calcos de las variables usadas durante el desarrollo teórico por lo que no se considera necesario comentario alguno al respecto.

4.4.3 Variable de control.

```
  char codigo = 0;     /* Esto es un flag que nos dice el estado del
                        * programa:
                        * 0 .- Continuar iterando.
                        * 1 .- Se ha encontrado la solución.
                        * 2 .- Problemas al abrir los ficheros.
                        * 3 .- Problemas al leer los datos.
                        * 4 .- Problemas al reservar la memoria. */
```

Ilustración 4-5: Código, variable de control.

La variable *codigo* es un flag a través del cual controlaremos el flujo del programa. En cada momento reflejará el estado en que se haya el algoritmo: iterando, finalizado correctamente o con algún error.

4.5 Estructura del fichero de entrada.

El fichero de entrada es un fichero de texto plano en el que el programa leerá los parámetros que definen el sistema a simular. Por defecto el fichero de entrada se llamará "ENTRADA.TXT" y estará en el mismo directorio que el ejecutable del programa. No obstante esto puede cambiarse en la llamada al programa como se explicó en el apartado 4.1

En este fichero se permiten comentarios que deberán estar limitados por los símbolos: /* (al principio del comentario) y */ (al final del comentario). Estos símbolos deberán aparecer aislados, no siendo válido un principio de comentario como, por ejemplo, *'/*a'*

El tamaño máximo de palabra será de 50 caracteres (tamaño máximo del buffer de lectura), no garantizándose la estabilidad del sistema si existen palabras de tamaño mayor[3].

El orden en que los parámetros son leídos es fijo, por lo que es muy importante no variarlo. Dicho orden puede verse en el siguiente ejemplo.

```
2.5e6       /* Peak Rate en b/s */

0.5e6       /* Mean Rate */

150e6       /* C en b/s */

1000        /* L en celdas */

200         /* M nº de fuentes */

50          /* K en celdas */
```

Ilustración 4-6: Código, ejemplo de fichero de entrada.

4.6 Estructura del fichero de salida.

El fichero de salida será un fichero de texto plano donde el programa irá escribiendo los resultados del programa. Por defecto el fichero de salida se llamará "SALIDA.TXT" y

[3] La única posibilidad que se me ocurre para introducir una palabra de mayor tamaño es introducir algún tipo de comentario-separador como una línea de asteríscos o similar para dar una mayor claridad al texto.

estará en el mismo directorio que el ejecutable del programa. No obstante esto puede cambiarse en la llamada al programa como se explicó en el apartado 4.1

Primeramente, en él, nos encontraremos una relación de los parámetros del sistema (que se obtuvieron del fichero de entrada) con los que se han calculado los resultados. El objetivo de esto es evitar confusiones en la asociación entre parámetros y resultados. Cada fichero de resultados lleva implícito los parámetros con los que han sido calculados.

Seguidamente encontramos una lista de todos los resultados parciales que se han ido obteniendo junto con el correspondiente número de iteración. Si la salida del bucle del algoritmo se ha producido como consecuencia de haber convergido a una solución (diferencia entre dos soluciones consecutivas inferior a *epsilon*), entonces la última solución calculada se haya separada de la anterior por tres puntos consecutivos.

```
Datos iniciales

Peak_Rate   2500000.000000

Mean_Rate   500000.000000

C     150000000.000000

L     1000

M     300

K     50

Epsilon     10e-10

Num_max_iteraciones     200


Iteración   Resultado

1     0.045979445841537812

2     0.045994391958725303

3     0.045994415908612703

4     0.045994415986348548

...

5     0.045994415986622773
```

Ilustración 4-7: Código, ejemplo de fichero de salida.

5 APROXIMACIÓN 2 MMDP

Ya vimos que el principal problema para el cálculo de la probabilidad de pérdida de célula era la resolución del sistema de ecuaciones:

$$\pi_i = \sum_{j \neq i} p_{j,i} \pi_j A_j \qquad i = 0,1,\ldots m-1$$

$$\sum_{i=0}^{m-1} \pi_i e^T = 1$$

cuyo cálculo en el caso de que m y K fueran relativamente grandes y P una matriz densa podía resultar engorroso.

La aproximación (M+1) MMDP; se basaba en simplificar el problema haciendo que la matriz P fuera poco densa, tridiagonal. La aproximación en la que nos vamos a centrar ahora, 2 MMDP, se basa en simplificar el problema reduciendo m a sólo dos posibles estados.

Asumiremos que $M\Delta > C$; ya que si esto no se cumple, no tendríamos congestión a largo plazo y nuestro análisis no tendría sentido (ya que no tendríamos células perdidas). Podemos entonces establecer una clara división en el espacio de estados de N(t):

1. *Región de vaciado del buffer:* aquellos estados $\{i : i = 0,1\ldots v\}$ en los que se cumple que $i\Delta \leq C$. La tasa de llegada de células es inferior o igual a la tasa de servicio con lo que el número de células en la cola va decrementándose o como mucho se mantiene.

2. *Región de llenado del buffer:* aquellos estados $\{i : i = v+1,\ldots M\}$ en los que se cumple que $i\Delta > C$. La tasa de llegada de células es superior a la tasa de servicio

con lo que el número de células en la cola va incrementándose pudiendo llegar a desbordar.

La aproximación 2 MMDP se basa en identificar los dos posibles estados de X(t) con estas dos regiones del espacio de estados de N(t). La idea será pues, calcular $\{B_o, B_1, \gamma_0, \gamma_1\}$ que son los parámetros que nos determinan unívocamente un sistema 2 MMDP a partir de los datos $\{\Delta, \lambda, \mu, M\}$ que tenemos del conjunto de fuentes ON-OFF.

Identificaremos B_o con la tasa media de llegada de células en la región de vaciado y $1/\gamma_o = E(T_0)$ con el tiempo medio en la misma región. Análogamente, B_1 y $1/\gamma_1 = E(T_1)$ serán la tasa media de llegada de células y el tiempo medio en la región de llenado.

Recordando que N(t) es un proceso de nacimiento y muerte de tasas $\lambda_i = (M-i)\lambda$ y $\mu_i = i\mu$; entonces puede demostrarse[4] que la distribución de probabilidad de estado de N(t) puede deducirse a partir de la distribución de probabilidad de estado de X(t) (calculables con **(E 3-2)**) de la forma:

$$q_i = \frac{p_i/\gamma_i}{\sum_{j=0}^{M} p_j/\gamma_j} \qquad \text{i=0,1...M} \qquad \textbf{(E 5-1)}$$

La probabilidad de cada una de las regiones será la suma de las probabilidades individuales de cada uno de los estados que las conforman:

$$p_{vaciado} = \sum_{k=0}^{v} q_k \qquad v = \max\{i : i\Delta \le C\}$$
$$p_{llenado} = 1 - p_{vaciado} \qquad \textbf{(E 5-2)}$$

Las tasas medias de llegada de células en cada una de las regiones quedarían de la forma:

$$B_0 = \sum_{k=1}^{v} \frac{k\Delta q_k}{p_{vaciado}}$$
$$B_1 = \sum_{k=v+1}^{M} \frac{k\Delta q_k}{(1-p_{vaciado})} \qquad \textbf{(E 5-3)}$$

Sólo nos quedaría calcular los valores de γ_0 y γ_1, o lo que es lo mismo, $E(T_0)$ y $E(T_1)$. Para ello definiremos la variable W_i con i= 0,1 , ... M-1; como el tiempo que tarda N(t) en

[4] S. M. Ross,*Introduction to Probability Models (4th ed.) New York: Academic, 1989. (p 276)*

llegar al estado i+1 partiendo del estado i.. $E(W_i)$ es calculable usando la siguiente fórmula recursiva[5]:

$$E(W_i) = \frac{1 + i\mu E(W_{i-1})}{(M-i)\lambda} \qquad i = 1,2,\ldots M-1$$
$$E(W_0) = 1/M\lambda \qquad \textbf{(E 5-4)}$$

Teniendo en cuenta lo anterior, es fácil calcular $E(T_0)$ ya que T_0 y W_v tienen la misma distribución, y por tanto $E(T_0)$ puede calcularse a través de la fórmula recursiva**.**

Una vez calculado $E(T_0)$, es fácil calcular $E(T_1)$ a partir de éste; ya que es fácil demostrar que se cumpliría:

$$p_{vaciado} = \frac{E(T_o)}{E(T_o) + E(T_1)}$$

$$E(T_1) = E(T_0)\frac{P_{vaciado}}{1 - P_{vaciado}}$$

5.1 Cálculo de la probabilidad de pérdida de célula.

Una vez calculados los parámetros $\{B_o, B_1, \gamma_0, \gamma_1\}$ que determinan el sistema 2 MMDP, todo lo visto en el apartado [2.4] sería aplicable.

De modo que la resolución del sistema de ecuaciones **(E 2-7)** de las probabilidades de estado quedaría de la forma:

$$\pi_0 = \pi_1 A_1 \qquad \pi_1 = \pi_0 A_0 \qquad \pi_0 e^T + \pi_1 e^T = 1$$
$$\pi_0 = \pi_0 A_0 A_1 \qquad \pi_0 e^T = 1/2$$

(A_0 es una matriz estocástica, lo que implica $\pi_i A_i e^T = \pi_i e^T$)

En la región *de vaciado* o región 0, sólo serían no nulos los $a^0_{k,h}$ donde $h \le k$, lo que conlleva que A_0 será una matriz triangular inferior. Lo propio sucede en la región de *llenado* o zona 1, sólo que esta vez los no nulos serán los $a^1_{k,h}$ donde $h > k$, por lo que la

[5] S. M. Ross,*Introduction to Probability Models (4th ed.) New York: Academic, 1989. (p 257)*

matriz A_1 será una matriz triangular superior. Por tanto el producto $A_0 A_1$ será una matriz plena. Podemos simplificar el sistema transformándolo de la forma:

$$\pi_0 A_1^{-1} A_0^{-1} = \pi_0 \qquad \pi_0 e^T = 1/2$$

Para aprovecharnos de la forma de las matrices inversas, ya que ahora el resultado del producto $A_1^{-1} A_0^{-1}$ es tridiagonal.

$$A_0^{-1} = \frac{1}{1-\rho_0} \begin{bmatrix} (1-\rho_0) & & & \cdots & \\ -\rho_0 & 1 & & & \vdots \\ & -\rho_0 & 1 & & \\ \vdots & & \ddots & \ddots & \\ & \cdots & & -\rho_0 & 1 \end{bmatrix}$$

$$A_1^{-1} = \frac{1}{1-\rho_1} \begin{bmatrix} 1 & -\rho_1 & & \cdots & \\ & 1 & -\rho_1 & & \vdots \\ & & 1 & \ddots & \\ \vdots & & & \ddots & -\rho_1 \\ & \cdots & & & (1-\rho_1) \end{bmatrix}$$

Resolviendo finalmente el sistema de ecuaciones llegamos a:

- $\pi_{0,k}$
  - $\rho_0 \neq \rho_1$

$$\pi_{0,k} = \begin{cases} \left(\rho_1 / \rho_0\right)^k \pi_{0,0} & 0 < k < K \\ \left(\rho_1 / \rho_0\right)^K \dfrac{\pi_{0,0}}{1-\rho_1} & k = K \end{cases}$$

$$\pi_{0,0} = \frac{1/2}{\dfrac{\rho_0}{\rho_0 - \rho_1} - \left(\rho_1 / \rho_0\right)^K \left[\dfrac{\rho_1 (1-\rho_0)}{(1-\rho_1)(\rho_0 - \rho_1)}\right]}$$

  - $\rho_0 = \rho_1$

$$\pi_{0,k} = \begin{cases} \frac{1}{2}\frac{1-\rho_0}{K(1-\rho_0)+1} & 0 \le k < K \\ \frac{1}{2}\frac{1}{K(1-\rho_0)+1} & k = K \end{cases}$$

- $\pi_{1,k}$
  - $\rho_0 \neq \rho_1$

$$\pi_{1,k} = \begin{cases} \left(\rho_0 / \rho_1\right)^{K-k} \pi_{1,K} & 0 < k < K \\ \left(\rho_0 / \rho_1\right)^{K} \frac{\pi_{1,K}}{1-\rho_0} & k = K \end{cases}$$

$$\pi_{1,K} = \frac{1/2}{\frac{\rho_1}{\rho_1-\rho_0} - \left(\rho_0/\rho_1\right)^K \left[\frac{\rho_0(1-\rho_1)}{(1-\rho_0)(\rho_1-\rho_0)}\right]}$$

  - $\rho_0 = \rho_1$

$$\pi_{0,k} = \begin{cases} \frac{1}{2}\frac{1-\rho_1}{K(1-\rho_1)+1} & 0 < k \le K \\ \frac{1}{2}\frac{1}{K(1-\rho_1)+1} & k = 0 \end{cases}$$

Aplicando estos resultados a **(E 2-11)** llegamos a una expresión cerrada para el cálculo de la probabilidad de pérdida de célula:

$$p = \begin{cases} \frac{1}{1-\rho_1}\frac{\rho_0-\rho_1}{\left(\rho_0/\rho_1\right)^{K+1}\left((1-\rho_1)-(1-\rho_0)\right)}\frac{(1-\eta_0)(1-\eta_1)}{\eta_0-2\eta_0\eta_1+\eta_1} & \rho_0 \neq \rho_1 \\ \frac{1}{1-\rho_1}\frac{\rho_1}{K(1-\rho_1)+1}\frac{(1-\eta_0)(1-\eta_1)}{\eta_0-2\eta_0\eta_1+\eta_1} & \rho_0 = \rho_1 \end{cases} \qquad \textbf{(E 5-5)}$$

donde $\eta_i = e^{-\gamma_i / B_i}$, $i = 0,1$.

Un hecho a tener en cuenta que se desprende de la ecuación precedente, es que si bien p es inversamente proporcional a K en cualquier caso (intuitivamente es un hecho muy claro: cuanto mayor sea el tamaño de la cola, menor será la probabilidad de que ésta desborde), en

el caso de que $\rho_0 \geq \rho_1$, p tiende a cero cuando K tiende a infinito; mientras que cuando $\rho_0 < \rho_1$ y K tiende a infinito, p no tiende a cero, sino a un valor p'.

$$\left.\begin{array}{l}\rho_0 \geq \rho_1 \\ K \to \infty\end{array}\right\} p \to 0 \qquad \left.\begin{array}{l}\rho_0 < \rho_1 \\ K \to \infty\end{array}\right\} p \to p'$$

Para entender este fenómeno tenemos que ver las implicaciones que tiene el hecho de que $\rho_0 < \rho_1$. En concreto, esto significa que se cumple: $\frac{C - B_0}{\gamma_0} < \frac{B_1 - C}{\gamma_1}$, lo que desarrollado implica: $C < \frac{B_1\gamma_0 + B_0\gamma_1}{\gamma_0 + \gamma_1}$. Es decir, que la tasa de servicio del sistema es menor que la tasa media de recepción de células.

6 IMPLEMENTACIÓN 2 MMDP

A lo largo del presente capítulo vamos a ver los pasos principales que se han llevado a cabo para la implementación del cálculo de la probabilidad de pérdida de célula siguiendo la aproximación 2 MMDP que se ha realizado en este proyecto para la obtención de resultados numéricos. El código completo puede encontrarse en el capítulo 9. PLANOS DEL PROYECTO.

Esta implementación se ha llevado a posteriori de la M+1 MMDP, es por ello por lo que podríamos decir que está fuertemente basada en la anterior, esto es observable en el hecho de que la estructura principal es muy parecida y hay una reutilización de funciones creadas para la implementación anterior.

6.1 Lenguaje de implementación y consideraciones generales.

La implementación del algoritmo se ha llevado a cabo en lenguaje ANSI C pensando siempre en maximizar la portabilidad del código, para que la compilación fuera posible y sencilla en cualquier plataforma.

La ejecución del programa se realiza desde la línea de comando con el formato:

```
PROMPT> ALG_2MMDP <Entrada> <Salida>
```

Tabla 6-1: Código, ejecución del programa.

Los parámetros <entrada> y <salida> son opcionales y corresponden con los nombres que hayamos decidido para los ficheros de entrada y salida respectivamente. En el primero de

ellos es el fichero a través del cual le pasaremos al programa los parámetros que definen el sistema. El segundo es el fichero dónde se almacenarán los resultados.

Una descripción más pormenorizada de ellos y de sus estructuras se hará en los apartados 6.5 y 6.6.

6.2 Diagrama de flujo.

Para la concepción de la implementación en forma de código de programa se ha seguido de forma *casi literal* el desarrollo teórico descrito en el capítulo 6. Existen muchas similitudes entre la aproximación M+1 MMDP y 2 MMDP, por lo que la implementación descrita en el capítulo 5 se ha tomado como base para esta implementación.

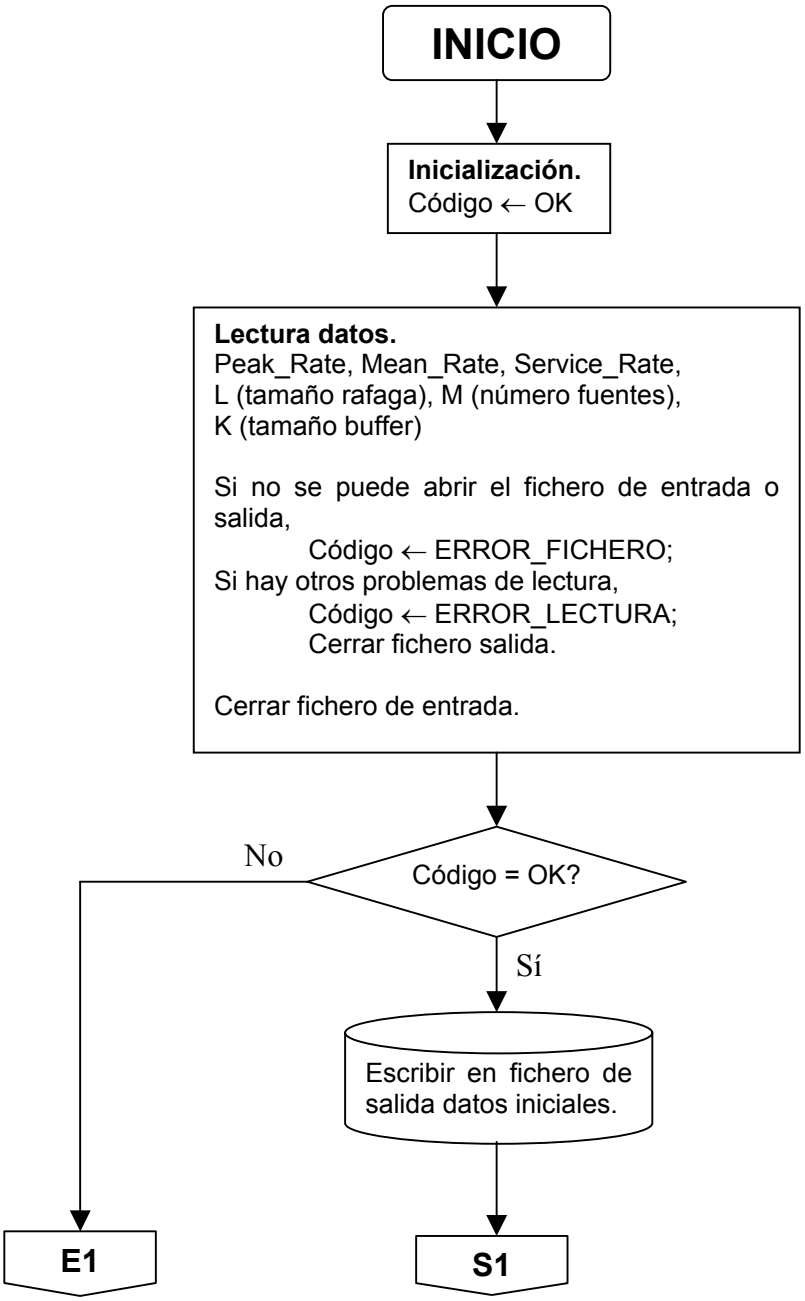


Ilustración 6-1: Diagrama de flujo de la implementación 2 MMDP (1 de 2)

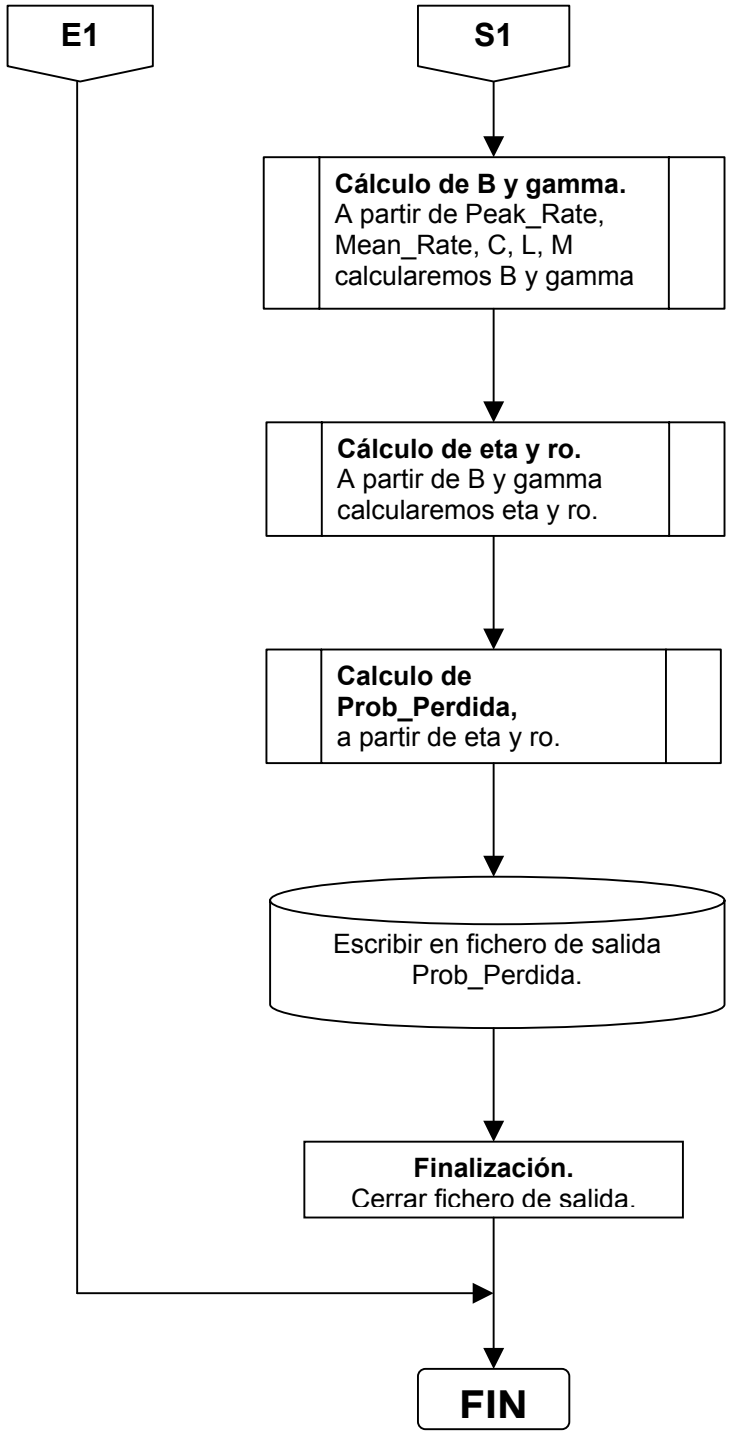


Ilustración 6-2: Diagrama de flujo de la implementación 2 MMDP (2 de 2)

El flujo del programa es determinado por el valor de la variable *código.* La variable *código* es un flag que nos determinará en cada momento el estado en que se encuentra el algoritmo:

ejecutándose correctamente, finalizado al encontrarse una solución o abortado debido a algún error.

6.3 Estructura de librerías.

Tomando como base el diagrama de flujo ilustrado en la sección anterior podemos identificar una serie de tareas susceptibles de ser agrupadas funcionalmente. Ésta es la base que se ha tomado para la concepción de las librerías de funciones que acompañan al código principal.

Áreas funcionales identificadas:

- Tareas relacionadas con los ficheros de entrada y salida: apertura, cierre, etc. (**F_fichero.c**)
- Lectura de datos: tratamiento de los comentarios, etc. (**F_lectura.c**)
- Funciones matemáticas: producto vectorial, productorios, normalizaciones, etc. (**F_matemáticas.c**)
- Cálculo de probabilidades. (**F_calculo_probabilidades.c**)

A continuación se describirán someramente la funciones contenidas en cada una de las librerías y su misión. Tanto los ficheros de cabeceras como el código completo puede encontrarse en el capítulo 9: PLANOS DEL PROYECTO.

6.3.1 F_fichero

Esta librería es una reutilización de la librería descrita en la sección 4.3.1. Está constituida por una única función:

```
char Apertura_Fichero(int argc, char *argv[],
                      FILE **Fichero_Entrada, FILE **Fichero_Salida);
```

La misión de la cual es abrir los ficheros de entrada y salida a partir de los nombres que se le suministren en la línea de comando. Caso de que sólo se le suministre uno, considerará que se trata del nombre del fichero de entrada, también puede darse el caso de que no se le proporcione ninguno de los dos. En caso de omisión se usarían los nombres de fichero por defecto: ENTRADA.TXT y SALIDA.TXT

Se consideró la posibilidad de implementar una función de cierre de ficheros, pero puesto que esta función no añadía nada a la ya suministrada por las librerías estándar se desechó la idea.

6.3.2 F_lectura

En esta librería dispondremos de dos funciones:

```
char leer_sin_comentarios (float * parametro, FILE * fichero);

char Lectura_Datos(float *Peak_Rate, float *Mean_Rate, float *C,

            int *L, int *M, int *K, FILE * fichero);
```

La función principal es `Lectura_Datos` encargada de leer sucesivamente y **en orden** los parámetros de entrada al sistema del fichero de entrada (ya abierto con el uso de la función descrita en el apartado anterior). Esta función es idéntica a la descrita en la sección 4.3.2; pero se ha variado el número de parámetros ya que en la aproximación 2 MMDP, al no tratarse de una algoritmo iterativo, no hacen falta los parámetros *epsilon* ni *Num_Max_Iteraciones.*

La función `leer_sin_comentarios` es usada desde la anterior con el fin de filtrar el contenido del fichero de entrada y extraer los datos desechando los comentarios que puedan encontrarse.

6.3.3 F_matemáticas

Esta librería es una reutilización de la descrita en la sección 4.3.4. Su contenido son funciones de cálculo vectorial puramente matemáticas de las que se hace uso y que no se encontraban implementadas en las librerías estándar[6].

- ```
  double Productorio (double * variable, int indice_inicial,

                      int indice_final);
  ```

  Nos devuelve el productorio de los elementos contenidos entre los índices inicial y final de un vector (introducido mediante la variable *variable*).

- ```
  void Vector_x_Matriz (double *vector, double **matriz,int K,

                      double *vector_resultado);
  ```

  Producto vectorial entre un vector y una matriz. Se entiende que las dimensiones son: (1xK) x (KxK). Con lo que el resultado es otro vector de dimensión (1xK).

- ```
  void Factor_x_Vector (double factor, double *vector, int K,

                      double *vector_resultado);
  ```

  Escalado de un vector por el factor proporcionado. Se entiende que el vector es de dimensión (1xK).

- ```
  void Vector_mas_Vector (double *vector1, double *vector2, int K,
  ```

[6] Como ya se comentó en el apartado 6.1 la idea era maximizar la portabilidad del código. Es por ello por lo que las funciones matemáticas necesarias se han implementado en lugar de usar alguna de las muchas librerías de cálculo vectorial disponibles.

```
                    double *vector_resultado);
```

Suma vectorial. Se entiende que los vectores son de dimensión (1xK)

- `void Normalizacion_Vector (double * vector, int K);`

Normalización de un vector por su módulo. Se entiende que el vector es de dimensión (1xK).

- `double Valor_Absoluto(double x);`

Devuelve el valor absoluto de un número.

6.3.4 F_calculo_probabilidades

En esta librería dispondremos de cuatro funciones, las cuales se encargarán de los cálculos de las probabilidades específicas de esta aproximación.

```
double Calculo_p0(double *alfa, double *beta, int M);

void Calculo_q(double nu, double lambda, int M, double *q);

void Calculo_B_gamma(double *B, double *gamma, float Peak_Rate,

                  float Mean_Rate, float C, int M, int L);

double Calculo_Probabilidad_Perdida(double *ro, double *eta, int K);
```

La función `Calculo_q` calcula $\{q_i\}$, la distribución de probabilidad del número de fuentes ON-OFF activas en un momento dado; para ello, como paso previo, deben calcularse $\{p_i\}$ la distribución de probabilidad de los estados de la cadena de Markov $\{X_n\}$ (para lo que se usa la función `Calculo_p0)`, y las variables auxiliares *alfa*, *beta* y *gamma*

La función `Calculo_B_gamma`, nos calcula los vectores *B* y *gamma* (a través de las ecuación **E 5-3** y **E 5-4**) que nos determinan de forma unívoca el sistema 2 MMDP.

Por último la función `Calculo_Probabilidad_Perdida` implementará la función **(E 5-5)** dándonos el valor para la probabilidad de pérdida de célula.

6.4 Variables y constantes principales.

Siempre que ha sido posible se ha mantenido la nomenclatura usada durante el desarrollo teórico para que el código fuera lo más claro posible. Una lista exhaustiva de las variables y símbolos usados a lo largo de este documento puede encontrarse en el capítulo 10.

6.4.1 Parámetros iniciales.

`Float Peak_Rate`	Tasa de pico de una fuente ON-OFF individual (dimensión: bits/segundo).
`Float Mean_Rate`	Tasa media de una fuente ON-OFF individual (dimensión: bits/segundo).
`int L`	Longitud media de ráfaga de una fuente ON-OFF individual. Número medio de células generadas durante el periodo ON (dimensión: células).
`Flota C`	Tasa de servicio de células del sistema. Velocidad a la que las células van saliendo de la cola (dimensión: bits/segundo).
`int K`	Tamaño máximo de la cola (dimensión: células).
`int M`	Número de fuente ON-OFF presentes en el sistema.

Tabla 6-2: Parámetros iniciales.

En la tabla 6-2 podemos encontrar los parámetros iniciales que nos definen unívocamente el sistema a simular.

Los tres primeros (*Peak_Rate*, *Mean_Rate* y *L*) nos dan todos los parámetros para definir una fuente individual ON-OFF. *M* nos da el número de fuentes ON-OFF en el sistema; y por último *K* y *C* nos darían los parámetros que nos definirían la cola (el tamaño de la cola y la velocidad de servicio).

Los parámetros iniciales se introducen en el programa a través del fichero de entrada de datos cuya estructura se detalla en el apartado 6.5.

6.4.2 Variables de cálculo.

`Double Probabilidad_Perdida = 0`	Variable en la que guardaremos el valor de la probabilidad de pérdida calculado.

`double gamma[2]`	Inversa del tiempo medio de estancia cada uno de los estados del sistema 2 MMDP.
`double B[2]`	Tasa media de llegadas de células en cada uno de los estados del sistema 2 MMDP.
`double eta[2]`	$= e^{-\gamma_i / B_i}$ Cambio de variable destinado a hacer más sencillo el cálculo de la probabilidad de pérdida.
`double ro[2]`	$\rho_0 = e^{-\gamma_0 / C - B_0}$ $\quad$ $\rho_1 = e^{-\gamma_1 / B_1 - C}$ Cambio de variable destinado a hacer más sencillo el cálculo de la probabilidad de pérdida.

Tabla 6-3: Variables de cálculo.

6.4.3 Variable de control.

```
char codigo = 0;     /* Esto es un flag que nos dice el estado del
                      * programa:
                      * 0 .- Continuar iterando.
                      * 1 .- Se ha encontrado la solución.
                      * 2 .- Problemas al abrir los ficheros.
                      * 3 .- Problemas al leer los datos.
                      * 4 .- Problemas al reservar la memoria. */
```

Ilustración 6-3: Código, variable de control.

La variable *codigo* es un calco de la que podemos encontrar en la aproximación M+1 MMDP. Se ha conservado tal cual para mostrar el fuerte paralelismo entre ambas implementaciones. No obstante es fácil observar que si bien la función principal de la

misma permanece idéntica (en cada momento reflejará el estado en que se haya el algoritmo) hay ciertas diferencias:

- El estado cero, es el estado OK del algoritmo, pero su significado ya no es: continuar iterando. Ya que en esta aproximación no hay iteraciones.
- Puesto que no hay funciones relacionadas con la reserva de memoria, el estado 4 es prescindible.

6.5 Estructura del fichero de entrada.

El fichero de entrada de datos tiene la misma estructura que el utilizado en la aproximación M+1 MMDP. La única diferencia radicaría en los parámetros, ya que ahora no son necesarios los relacionados con el control de la iteración del algoritmo: *epsilon* y *Num_Max_Iteraciones.*

Es un fichero de texto plano en el que el programa leerá los parámetros que definen el sistema a simular. Por defecto el fichero de entrada se llamará "ENTRADA.TXT" y estará en el mismo directorio que el ejecutable del programa. No obstante esto puede cambiarse en la llamada al programa como se explicó en el apartado 6.1.

En este fichero se permiten comentarios que deberán estar limitados por los símbolos: /* (al principio del comentario) y */ (al final del comentario). Estos símbolos deberán aparecer aislados, no siendo válido un principio de comentario como, por ejemplo*, '/*a'*

El tamaño máximo de palabra será de 50 caracteres (tamaño máximo del buffer de lectura), no garantizándose la estabilidad del sistema si existen palabras de tamaño mayor[7].

El orden en que los parámetros son leídos es fijo, por lo que es muy importante no variarlo. Dicho orden puede verse en el siguiente ejemplo.

```
2.5e6       /* Peak Rate en b/s */

0.5e6       /* Mean Rate */

150e6       /* C en b/s */

1000        /* L en celdas */

200         /* M nº de fuentes */

50          /* K en celdas */
```

[7] La única posibilidad que se me ocurre para introducir una palabra de mayor tamaño es introducir algún tipo de comentario-separador como una línea de asteriscos o similar para dar una mayor claridad al texto.

Ilustración 6-4: Código, ejemplo de fichero de entrada.

6.6 Estructura del fichero de salida.

El fichero de salida será un fichero de texto plano donde el programa irá escribiendo los resultados del programa. Por defecto el fichero de salida se llamará "SALIDA.TXT" y estará en el mismo directorio que el ejecutable del programa. No obstante esto puede cambiarse en la llamada al programa como se explicó en el apartado 6.1.

En él, nos encontraremos una relación de los parámetros del sistema (que se obtuvieron del fichero de entrada) con los que se ha calculado el resultado. El objetivo de esto es evitar confusiones en la asociación entre parámetros y resultado. Cada fichero de resultados lleva implícito los parámetros con los que han sido calculados.

```
Datos iniciales

Peak_Rate   2500000.000000

Mean_Rate   500000.000000

C           150000000.000000

L           1000

M           200

K           50


Resultado   0.000015
```

lustración 6-5: Código, ejemplo de fichero de salida.

7 RESULTADOS Y ESTUDIO NUMÉRICO

A lo largo del este capítulo vamos a presentar los resultados que se han obtenido con las distintas aproximaciones.

Se plantearán tres escenarios en función del tráfico[8] con el que trabajaremos en cada caso. En concreto usaremos los tres tipos típicos de fuentes:

	Δ, tasa de pico (b/s)	Φ, tasa media (b/s)	L, tamaño de ráfaga (células)
Voz	64K	22K	58
Datos	10M	1M	339
Imágenes	2M	87K	2604

Tabla 7-1: Características del tráfico generado por las fuentes ON-OFF.

Dentro de estos escenarios estudiaremos la variación de la probabilidad de pérdida de célula con respecto al número de fuentes presentes en el sistema (M) y con respecto al tamaño del buffer (K).

[8] Las características para definir cada uno de los tipos de tráfico se han obtenido de: P. Castelli, E. Cavallero, A. Tonetti, "Policing and call call admission problems in ATM networks", in Proc. ITC-13, 1991 pp 847-852.

Además para cada tipo de tráfico y variable veremos los resultados para dos posibles tipos de sistemas: grandes y pequeños.

De modo que finalmente tendremos 12 combinaciones.

Posteriormente haremos un pequeño estudio comparando los resultados obtenidos con cada una de las aproximaciones.

7.1 Resultados.

7.1.1 Sistema grande. Tráfico de voz. Variación en M.

DATOS TRAFICO		DATOS COLA		DATOS ITERACION	
Δ, tasa de pico (b/s)	64000	C, tasa de servicio (b/s)	7000000	Epsilon	0,0001
Φ, tasa media (b/s)	22000	K, tamaño de la cola (células)	50	Número máximo de iteraciones	200
L, tamaño de ráfaga (células)	58				

Tabla 7-2: Características generales del cálculo. (Sistema grande. Tráfico de voz. Variación en M).

M	2MMDP	M+1 MMDP
250	2,6226E-05	2,6413E-05
255	6,1889E-05	6,2279E-05
260	1,3639E-04	1,3715E-04
265	2,8193E-04	2,8328E-04
270	5,4881E-04	5,5106E-04
275	1,0099E-03	1,0134E-03
280	1,7630E-03	1,7682E-03
285	2,9301E-03	2,9372E-03
290	4,6515E-03	4,6606E-03
295	7,0752E-03	7,0862E-03

300	1,0343E-02	1,0356E-02

Tabla 7-3: Datos (Sistema Grande. Tráfico Voz. Variación en M)

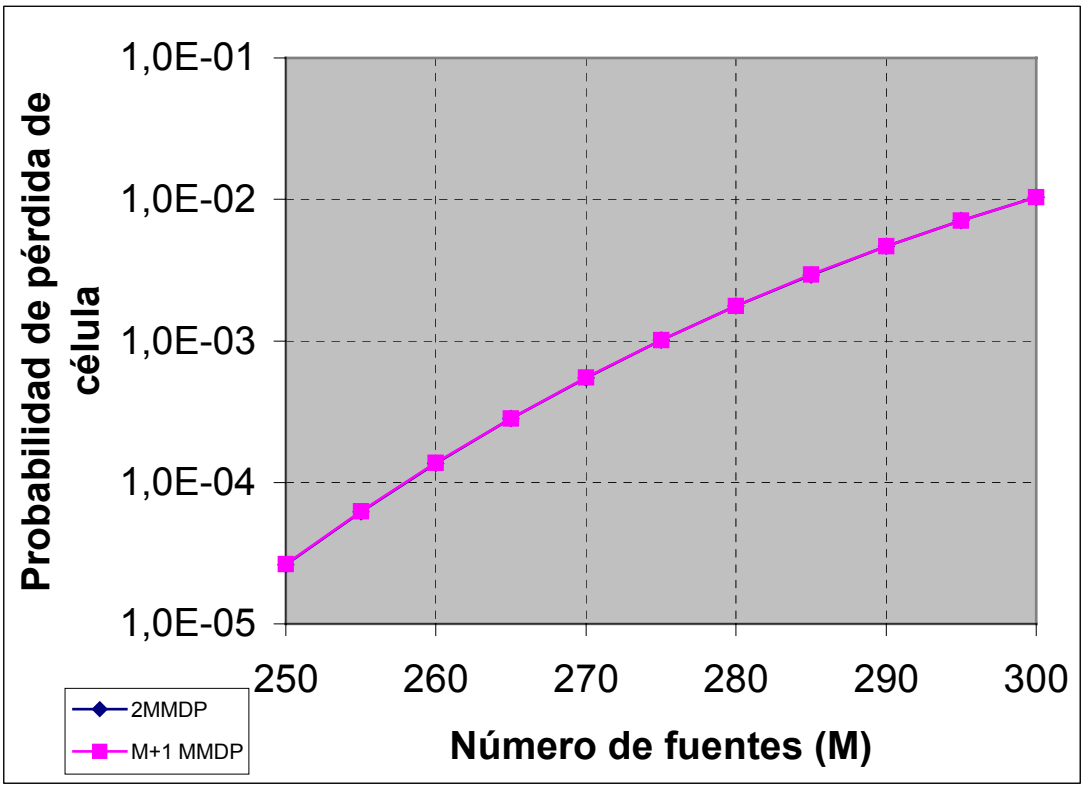


Ilustración 7-1: Gráfica de variación de la probabilidad de pérdida de célula (p). Sistema grande. Tráfico de voz. Variación en M.

7.1.2 Sistema grande. Tráfico de datos. Variación en M.

DATOS TRAFICO		DATOS COLA		DATOS ITERACION	
Δ, tasa de pico (b/s)	10000000	C, tasa de servicio (b/s)	350000000	Epsilon	0,0001
Φ, tasa media (b/s)	1000000	K, tamaño de la cola (células)	50	Número máximo de iteraciones	200
L, tamaño de ráfaga (células)	339				

Tabla 7-4:Características generales del cálculo. (Sistema grande. Tráfico de datos. Variación en M).

M	2MMDP	M+1 MMDP
150	5,4818E-08	5,4902E-08
165	5,6855E-07	5,6937E-07
180	4,0967E-06	4,1022E-06
195	2,1939E-05	2,1966E-05
210	9,1885E-05	9,1986E-05
225	3,1313E-04	3,1343E-04
240	8,9599E-04	8,9674E-04
255	2,2084E-03	2,2100E-03
270	4,7883E-03	4,7911E-03
285	9,2959E-03	9,3005E-03
300	1,6404E-02	1,6411E-02

Tabla 7-5: Datos (Sistema Grande. Tráfico Datos. Variación en M)

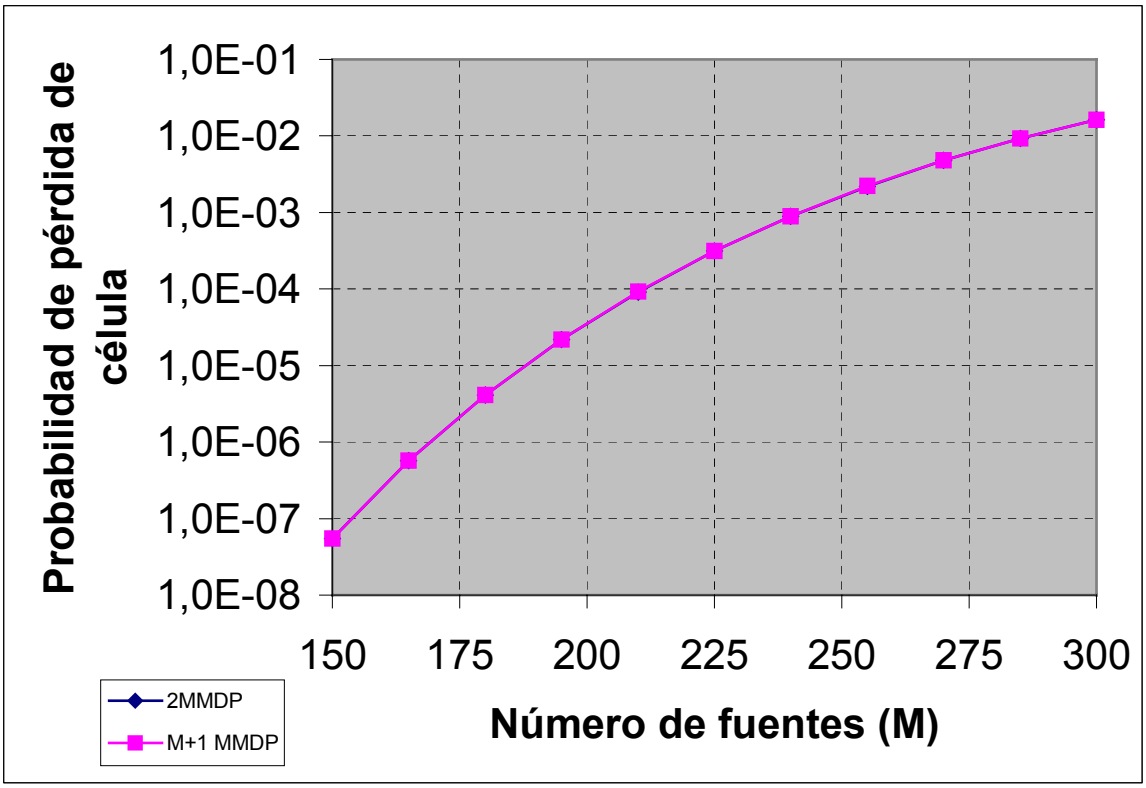


Ilustración 7-2: Gráfica de variación de la probabilidad de pérdida de célula (p). Sistema grande. Tráfico de datos. Variación en M.

7.1.3 Sistema grande. Tráfico de imágenes. Variación en M.

DATOS TRAFICO		DATOS COLA		DATOS ITERACION	
Δ, tasa de pico (b/s)	2000000	C, tasa de servicio (b/s)	30000000	Epsilon	0,0001
Φ, tasa media (b/s)	87000	K, tamaño de la cola (células)	50	Número máximo de iteraciones	200
L, tamaño de ráfaga (células)	2604				

Tabla 7-6:Características generales del cálculo. (Sistema grande. Tráfico de imágenes. Variación en M).

M	2MMDP	M+1 MMDP
75	4,3706E-08	4,3715E-08
100	1,9867E-06	1,9869E-06

125	2,8619E-05	2,8620E-05
150	2,0615E-04	2,0616E-04
175	9,3579E-04	9,3587E-04
200	3,0613E-03	3,0615E-03
225	7,8568E-03	7,8573E-03
250	1,6755E-02	1,6756E-02
275	3,0924E-02	3,0925E-02
300	5,0909E-02	5,0911E-02

Tabla 7-7: Datos (Sistema Grande. Tráfico Imágenes. Variación en M)

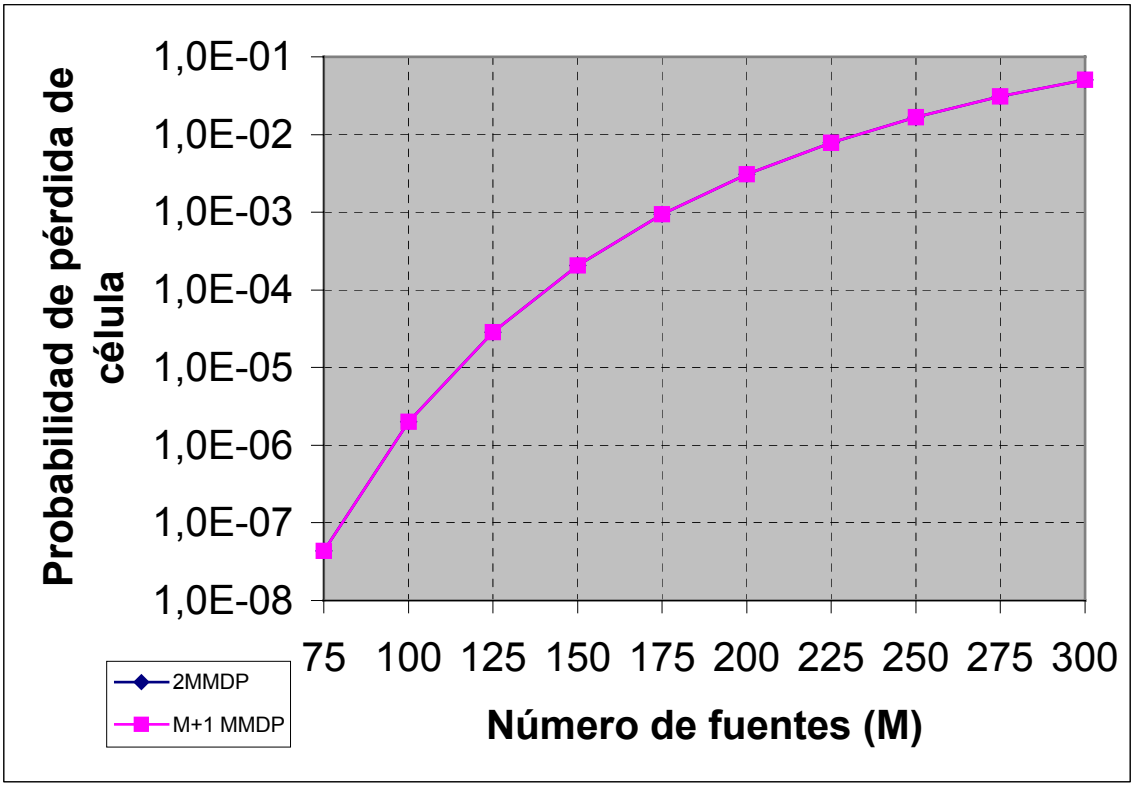


Ilustración 7-3: Gráfica de variación de la probabilidad de pérdida de célula (p). Sistema grande. Tráfico de imágenes. Variación en M.

7.1.4 Sistema grande. Tráfico de voz. Variación en K.

DATOS TRAFICO		DATOS COLA		DATOS ITERACION	
Δ, tasa de pico (b/s)	64000	C, tasa de servicio (b/s)	7000000	Epsilon	0,0001
Φ, tasa media (b/s)	22000	M, número de fuentes ON-OFF presentes en el sistema	250	Número máximo de iteraciones	200
L, tamaño de ráfaga (células)	58				

Tabla 7-8: Características generales del cálculo. (Sistema grande. Tráfico de voz. Variación en K).

K	2MMDP	M+1 MMDP
2	2,7109E-05	2,7076E-05
4	2,7071E-05	2,7040E-05
6	2,7034E-05	2,7005E-05
8	2,6997E-05	2,6971E-05
10	2,6960E-05	2,6938E-05
20	2,6774E-05	2,6787E-05
30	2,6590E-05	2,6651E-05
40	2,6408E-05	2,6527E-05
50	2,6226E-05	2,6413E-05
60	2,6046E-05	2,6308E-05
70	2,5867E-05	2,6208E-05
80	2,5689E-05	2,6114E-05
90	2,5513E-05	2,6028E-05
100	2,5338E-05	2,5944E-05

Tabla 7-9: Datos (Sistema Grande. Tráfico Voz. Variación en K)

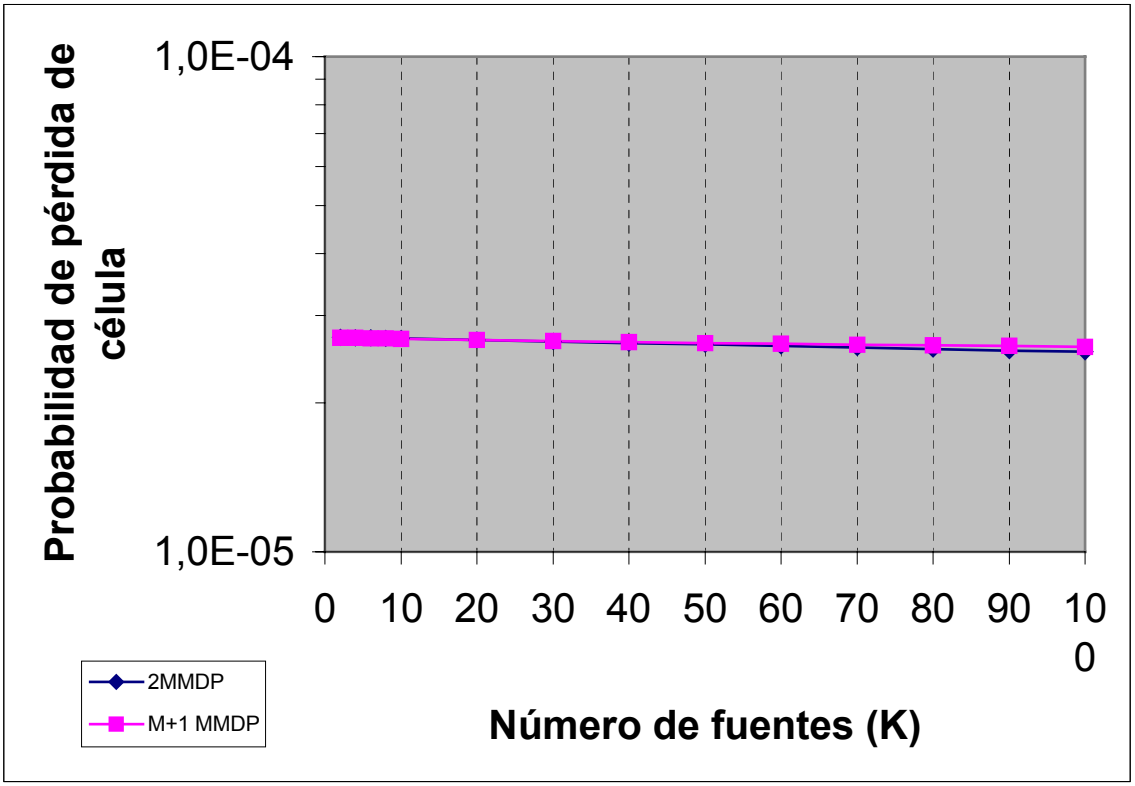


Ilustración 7-4: Gráfica de variación de la probabilidad de pérdida de célula (p). Sistema grande. Tráfico de voz. Variación en K.

7.1.5 Sistema grande. Tráfico de datos. Variación en K.

DATOS TRAFICO		DATOS COLA		DATOS ITERACION	
Δ, tasa de pico (b/s)	10000000	C, tasa de servicio (b/s)	350000000	Epsilon	0,0001
Φ, tasa media (b/s)	1000000	M, número de fuentes ON-OFF presentes en el sistema	250	Número máximo de iteraciones	200
L, tamaño de ráfaga (células)	339				

Tabla 7-10:Características generales del cálculo. (Sistema grande. Tráfico de datos. Variación en K).

K	2MMDP	M+1 MMDP

2	4,1134E-06	4,1136E-06
4	4,1127E-06	4,1131E-06
6	4,1120E-06	4,1126E-06
8	4,1113E-06	4,1122E-06
10	4,1106E-06	4,1117E-06
20	4,1071E-06	4,1093E-06
30	4,1036E-06	4,1069E-06
40	4,1001E-06	4,1045E-06
50	4,0967E-06	4,1022E-06
60	4,0932E-06	4,0998E-06
70	4,0897E-06	4,0974E-06
80	4,0863E-06	4,0951E-06
90	4,0828E-06	4,0927E-06
100	4,0793E-06	4,0904E-06

Tabla 7-11: Datos (Sistema Grande. Tráfico Datos. Variación en K)

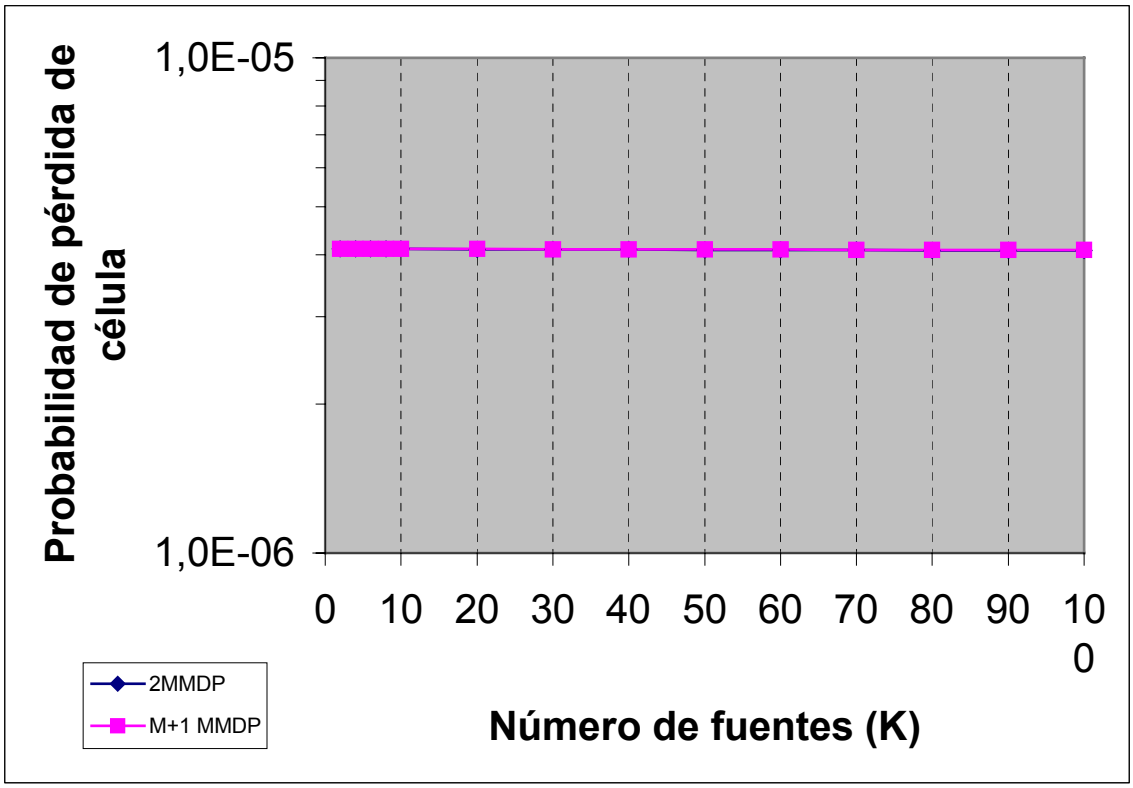


Ilustración 7-5: Gráfica de variación de la probabilidad de pérdida de célula (p). Sistema grande. Tráfico de datos. Variación en K.

7.1.6 Sistema grande. Tráfico de imágenes. Variación en K.

DATOS TRAFICO		DATOS COLA		DATOS ITERACION	
Δ, tasa de pico (b/s)	2000000	C, tasa de servicio (b/s)	30000000	Epsilon	0,0001
Φ, tasa media (b/s)	87000	M, número de fuentes ON-OFF presentes en el sistema	250	Número máximo de iteraciones	200
L, tamaño de ráfaga (células)	2604				

Tabla 7-12:Características generales del cálculo. (Sistema grande. Tráfico de imágenes. Variación en K).

K	2MMDP	M+1 MMDP
2	1,9875E-06	1,9875E-06

4	1,9875E-06	1,9875E-06
6	1,9875E-06	1,9875E-06
8	1,9874E-06	1,9874E-06
10	1,9874E-06	1,9874E-06
20	1,9872E-06	1,9873E-06
30	1,9870E-06	1,9871E-06
40	1,9869E-06	1,9870E-06
50	1,9867E-06	1,9869E-06
60	1,9865E-06	1,9867E-06
70	1,9863E-06	1,9866E-06
80	1,9862E-06	1,9864E-06
90	1,9860E-06	1,9863E-06
100	1,9858E-06	1,9861E-06

Tabla 7-13: Datos (Sistema Grande. Tráfico Imágenes. Variación en K)

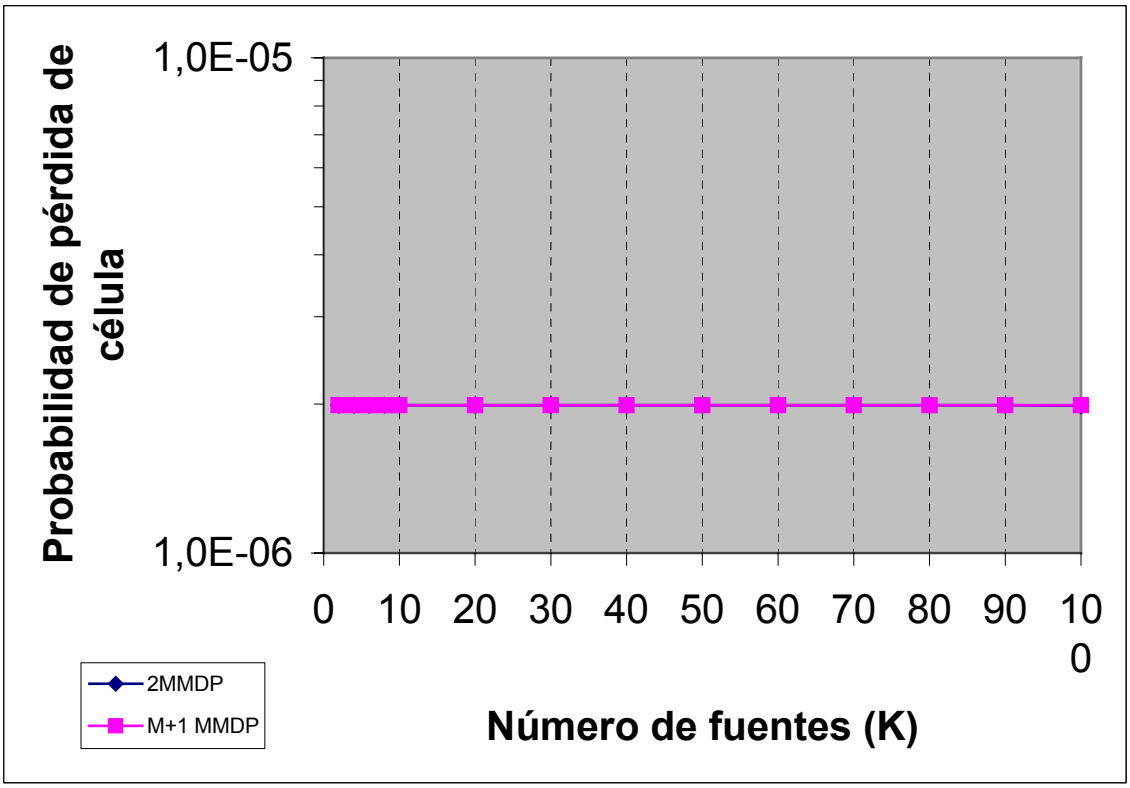


Ilustración 7-6: Gráfica de variación de la probabilidad de pérdida de célula (p). Sistema grande. Tráfico de imágenes. Variación en K.

7.1.7 Sistema pequeño. Tráfico de voz. Variación en M.

DATOS TRAFICO		DATOS COLA		DATOS ITERACION	
Δ, tasa de pico (b/s)	64000	C, tasa de servicio (b/s)	700000	Epsilon	0,0001
Φ, tasa media (b/s)	22000	K, tamaño de la cola (células)	50	Número máximo de iteraciones	200
L, tamaño de ráfaga (células)	58				

Tabla 7-14: Características generales del cálculo. (Sistema pequeño. Tráfico de voz. Variación en M).

M	2MMDP	M+1 MMDP
15	1,0687E-04	1,0790E-04
16	2,7425E-04	2,7640E-04

17	6,1001E-04	6,1388E-04
18	1,2138E-03	1,2201E-03
19	2,2089E-03	2,2183E-03
20	3,7355E-03	3,7484E-03
21	5,9417E-03	5,9587E-03
22	8,9732E-03	8,9942E-03
23	1,2962E-02	1,2987E-02
24	1,8019E-02	1,8047E-02
25	2,4223E-02	2,4254E-02

Tabla 7-15: Datos (Sistema pequeño. Tráfico Voz. Variación en M)

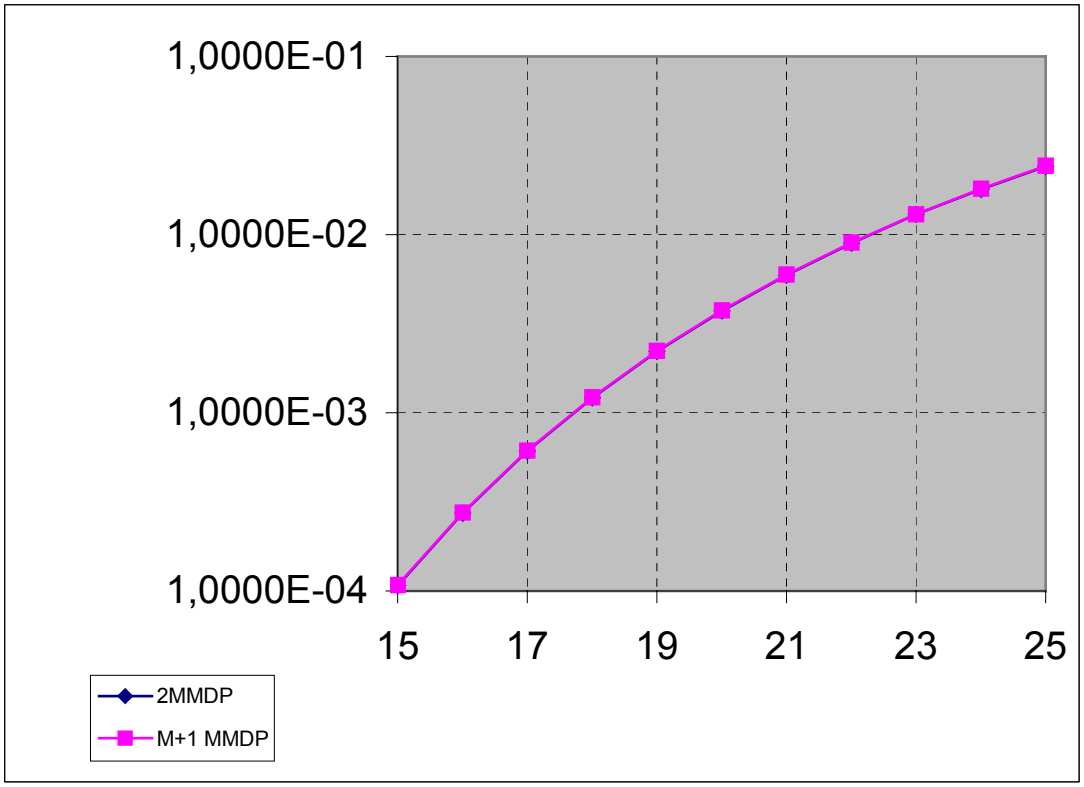


Ilustración 7-7: Gráfica de variación de la probabilidad de pérdida de célula (p). Sistema pequeño. Tráfico de voz. Variación en M.

7.1.8 Sistema pequeño. Tráfico de datos. Variación en M.

DATOS TRAFICO		DATOS COLA		DATOS ITERACION	
Δ, tasa de pico (b/s)	10000000	C, tasa de servicio (b/s)	52000000	Epsilon	0,0001
Φ, tasa media (b/s)	1000000	K, tamaño de la cola (células)	50	Número máximo de iteraciones	200
L, tamaño de ráfaga (células)	339				

Tabla 7-16:Características generales del cálculo. (Sistema pequeño. Tráfico de datos. Variación en M).

M	2MMDP	M+1 MMDP
7	7,4382E-06	7,4382E-06
9	6,0377E-05	6,0377E-05
11	2,3653E-04	2,3653E-04
13	6,4784E-04	6,4784E-04
15	1,4286E-03	1,4286E-03
17	2,7222E-03	2,7222E-03
19	4,6685E-03	4,6685E-03
21	7,3940E-03	7,3937E-03
23	1,1005E-02	1,1005E-02
25	1,5583E-02	1,5583E-02

Tabla 7-17: Datos (Sistema Pequeño. Tráfico Datos. Variación en M)

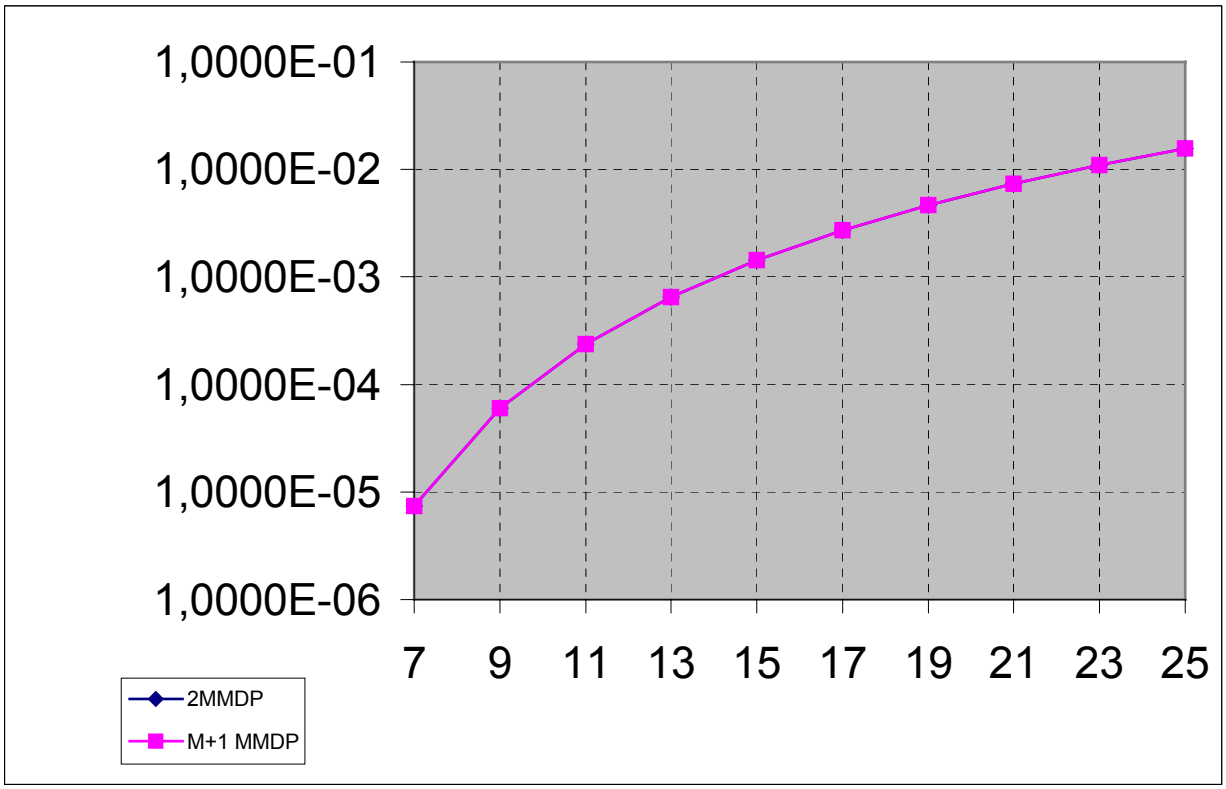


Ilustración 7-8: Gráfica de variación de la probabilidad de pérdida de célula (p). Sistema pequeño. Tráfico de datos. Variación en M.

7.1.9 Sistema pequeño. Tráfico de imágenes. Variación en M.

DATOS TRAFICO		DATOS COLA		DATOS ITERACION	
Δ, tasa de pico (b/s)	2000000	C, tasa de servicio (b/s)	7000000	Epsilon	0,0001
Φ, tasa media (b/s)	87000	K, tamaño de la cola (células)	50	Número máximo de iteraciones	200
L, tamaño de ráfaga (células)	2604				

Tabla 7-18:Características generales del cálculo. (Sistema pequeño. Tráfico de imágenes. Variación en M).

M	2MMDP	M+1 MMDP
5	4,0426E-05	4,0426E-05

7	1,9513E-04	1,9513E-04
9	5,2735E-04	5,2735E-04
11	1,0905E-03	1,0905E-03
13	1,9291E-03	1,9292E-03
15	3,0792E-03	3,0795E-03
17	4,5698E-03	4,5701E-03
19	6,4229E-03	6,4231E-03
21	8,6544E-03	8,6546E-03
23	1,1275E-02	1,1275E-02
25	1,4291E-02	1,4290E-02

Tabla 7-19: Datos (Sistema Pequeño. Tráfico Imágenes. Variación en M)

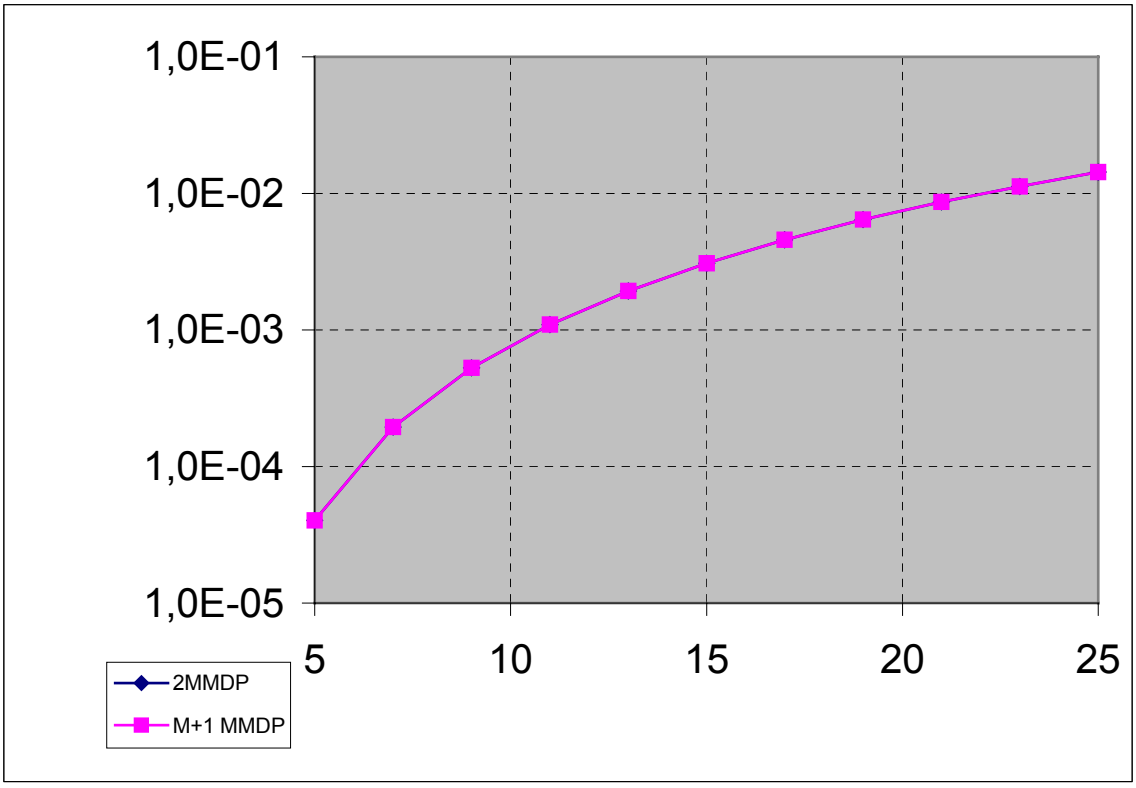


Ilustración 7-9: Gráfica de variación de la probabilidad de pérdida de célula (p). Sistema pequeño. Tráfico de imágenes. Variación en M.

7.1.10 Sistema pequeño. Tráfico de voz. Variación en K.

DATOS TRAFICO		DATOS COLA		DATOS ITERACION	
Δ, tasa de pico (b/s)	64000	C, tasa de servicio (b/s)	700000	Epsilon	0,0001
Φ, tasa media (b/s)	22000	M, número de fuentes ON-OFF presentes en el sistema	17	Número máximo de iteraciones	200
L, tamaño de ráfaga (células)	58				

Tabla 7-20: Características generales del cálculo. (Sistema pequeño. Tráfico de voz. Variación en K).

K	2MMDP	M+1 MMDP
2	6,3807E-04	6,3789E-04
4	6,3687E-04	6,3672E-04
6	6,3568E-04	6,3557E-04
8	6,3449E-04	6,3443E-04
10	6,3330E-04	6,3331E-04
20	6,2740E-04	6,2795E-04
30	6,2155E-04	6,2294E-04
40	6,1575E-04	6,1826E-04
50	6,1001E-04	6,1388E-04
60	6,0432E-04	6,0977E-04
70	5,9869E-04	6,0590E-04
80	5,9310E-04	6,0227E-04
90	5,8757E-04	5,9884E-04

100	5,8209E-04	5,9559E-04

Tabla 7-21: Datos (Sistema pequeño. Tráfico Voz. Variación en K)

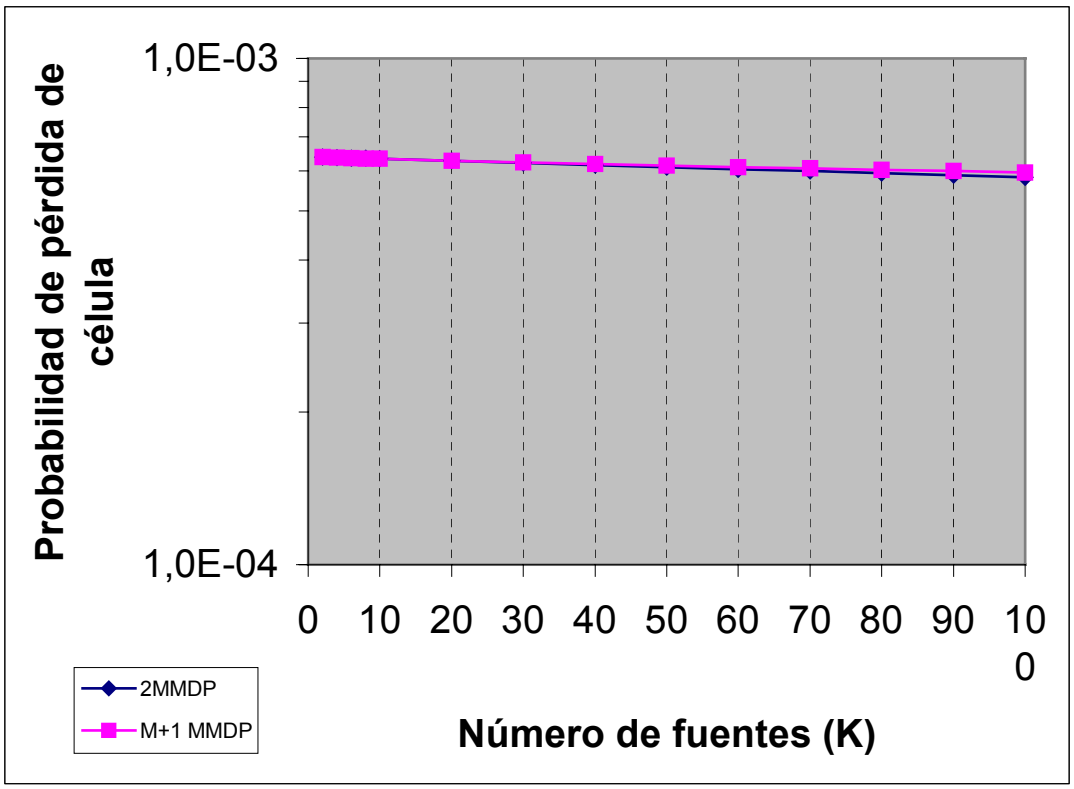


Ilustración 7-10: Gráfica de variación de la probabilidad de pérdida de célula (p). Sistema grande. Tráfico de voz. Variación en K.

7.1.11 Sistema pequeño. Tráfico de datos. Variación en K.

DATOS TRAFICO		DATOS COLA		DATOS ITERACION	
Δ, tasa de pico (b/s)	10000000	C, tasa de servicio (b/s)	52000000	Epsilon	0,0001
Φ, tasa media (b/s)	1000000	M, número de fuentes ON-OFF presentes en el sistema	9	Número máximo de iteraciones	200
L, tamaño de ráfaga (células)	339				

Tabla 7-22:Características generales del cálculo. (Sistema pequeño. Tráfico de datos. Variación en K).

K	2MMDP	M+1 MMDP
2	6,0513E-05	6,0516E-05
4	6,0507E-05	6,0513E-05
6	6,0502E-05	6,0502E-05
8	6,0496E-05	6,0496E-05
10	6,0490E-05	6,0491E-05
20	6,0462E-05	6,0462E-05
30	6,0434E-05	6,0434E-05
40	6,0405E-05	6,0406E-05
50	6,0377E-05	6,0377E-05
60	6,0349E-05	6,0349E-05
70	6,0320E-05	6,0321E-05
80	6,0292E-05	6,0292E-05
90	6,0264E-05	6,0264E-05
100	6,0235E-05	6,0236E-05

Tabla 7-23: Datos (Sistema pequeño. Tráfico Datos. Variación en K)

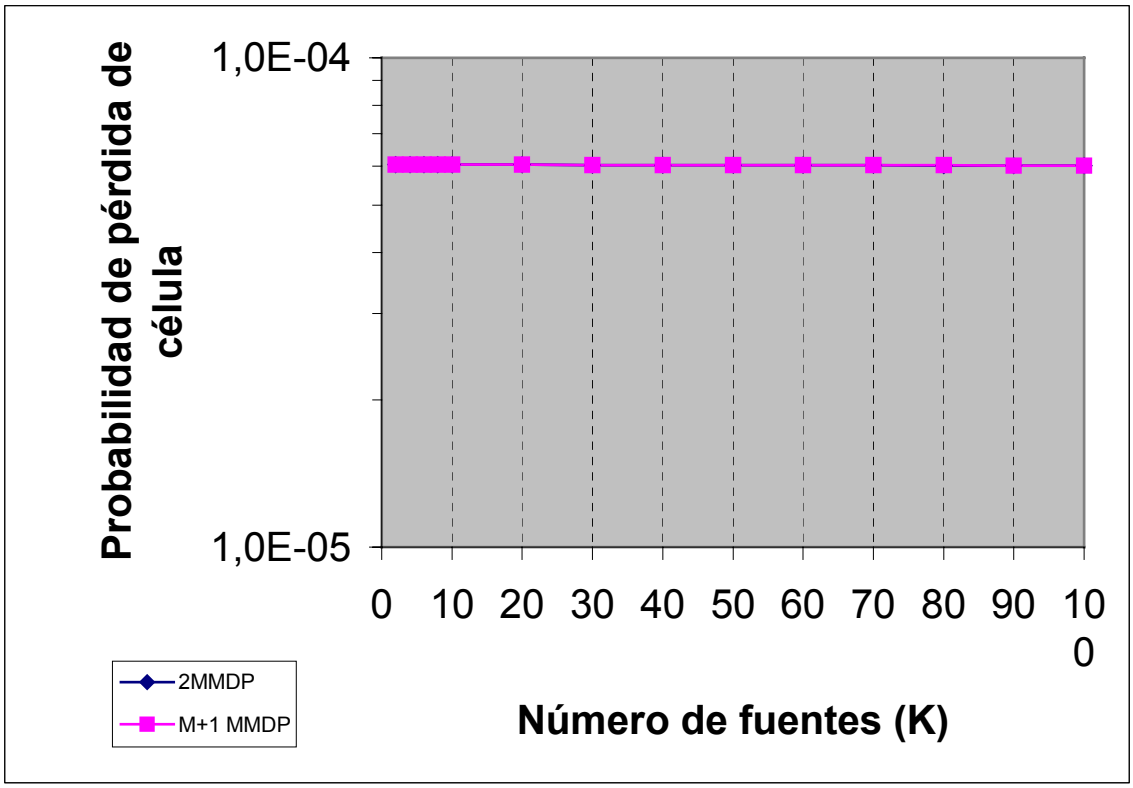


Ilustración 7-11: Gráfica de variación de la probabilidad de pérdida de célula (p). Sistema pequeño. Tráfico de datos. Variación en K.

7.1.12 Sistema pequeño. Tráfico de imágenes. Variación en K.

DATOS TRAFICO		DATOS COLA		DATOS ITERACION	
Δ, tasa de pico (b/s)	2000000	C, tasa de servicio (b/s)	7000000	Epsilon	0,0001
Φ, tasa media (b/s)	87000	M, número de fuentes ON-OFF presentes en el sistema	5	Número máximo de iteraciones	200
L, tamaño de ráfaga (células)	2604				

Tabla 7-24:Características generales del cálculo. (Sistema pequeño. Tráfico de imágenes. Variación en K).

K	2MMDP	M+1 MMDP
2	4,0440E-05	4,0440E-05

4	4,0439E-05	4,0440E-05
6	4,0438E-05	4,0439E-05
8	4,0438E-05	4,0439E-05
10	4,0437E-05	4,0439E-05
20	4,0434E-05	4,0437E-05
30	4,0432E-05	4,0432E-05
40	4,0429E-05	4,0429E-05
50	4,0426E-05	4,0426E-05
60	4,0423E-05	4,0423E-05
70	4,0420E-05	4,0420E-05
80	4,0417E-05	4,0417E-05
90	4,0415E-05	4,0415E-05
100	4,0412E-05	4,0412E-05

Tabla 7-25: Datos (Sistema pequeño. Tráfico Imágenes. Variación en K)

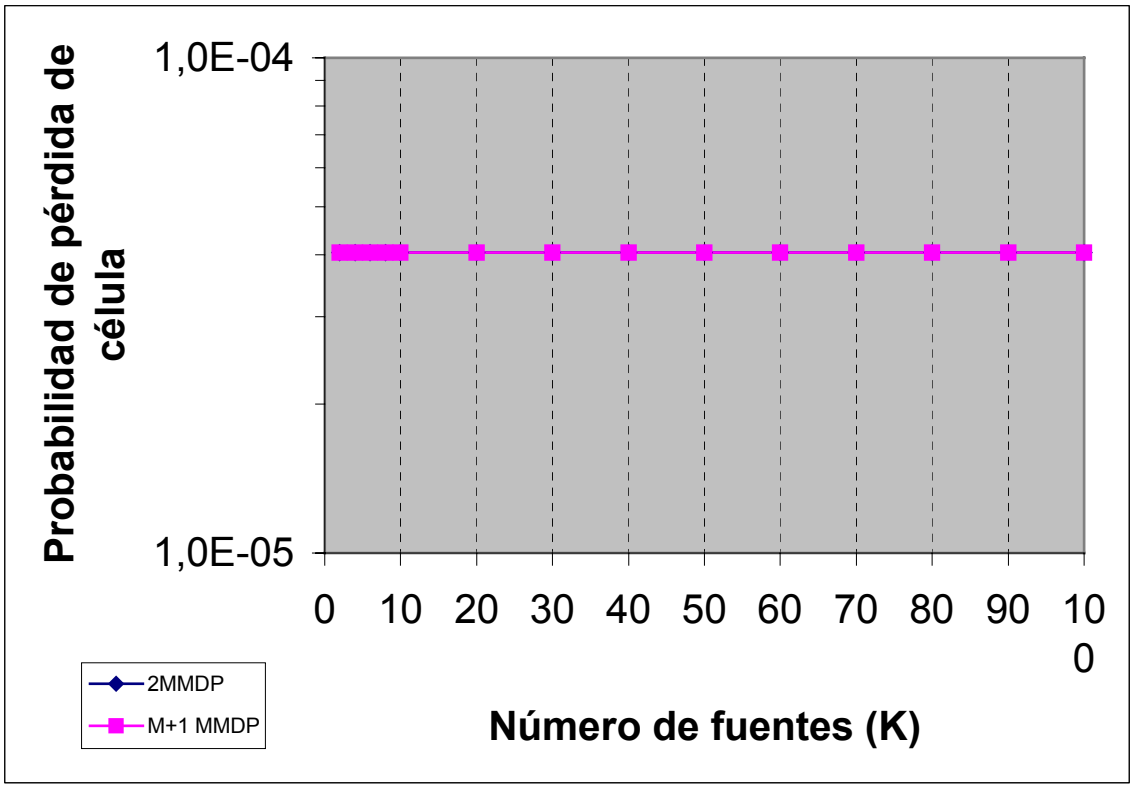
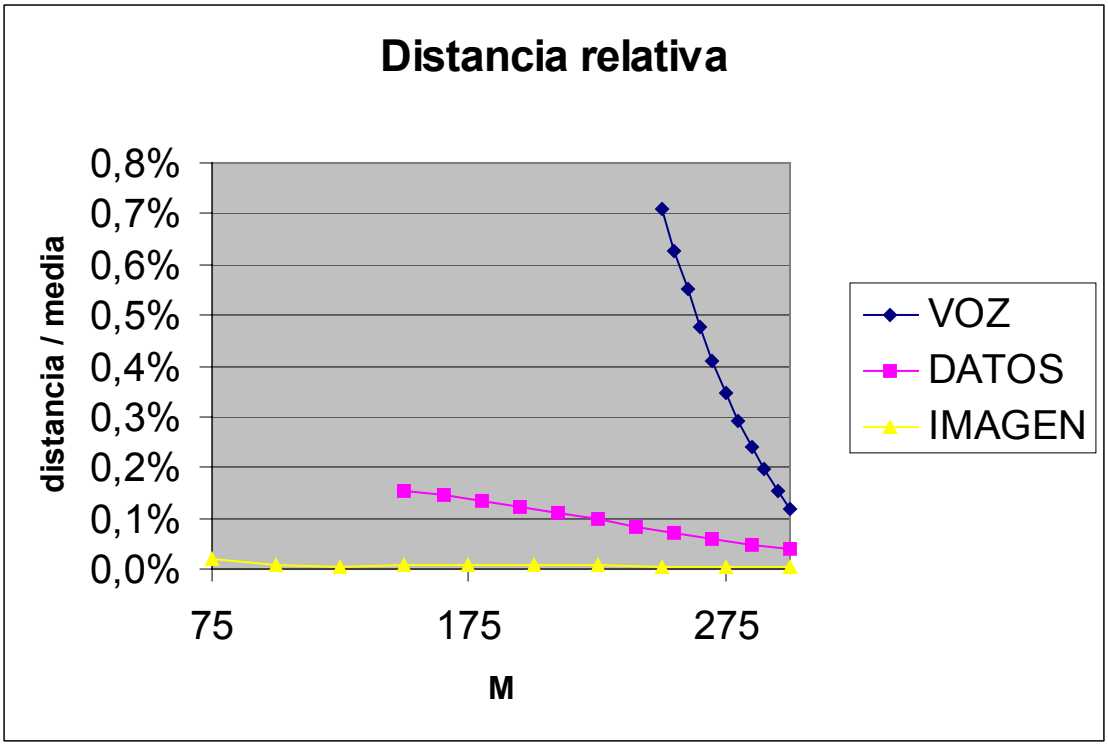


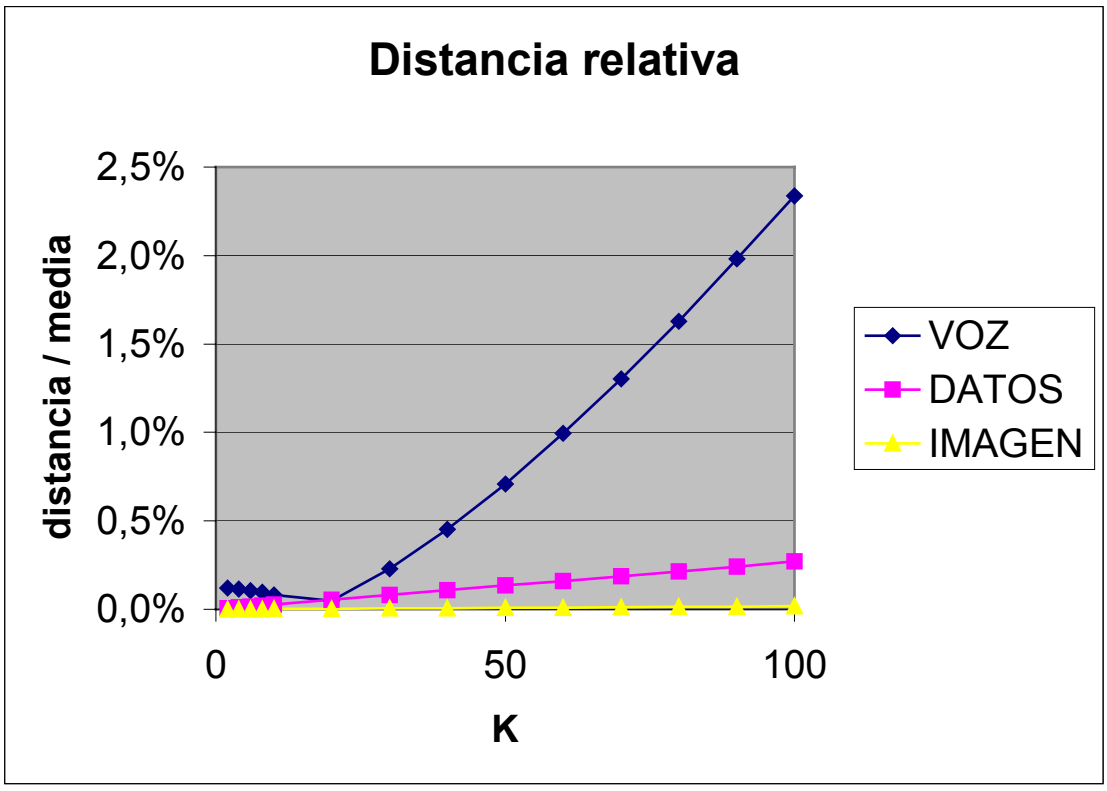
Ilustración 7-12: Gráfica de variación de la probabilidad de pérdida de célula (p). Sistema pequeño. Tráfico de imágenes. Variación en K.

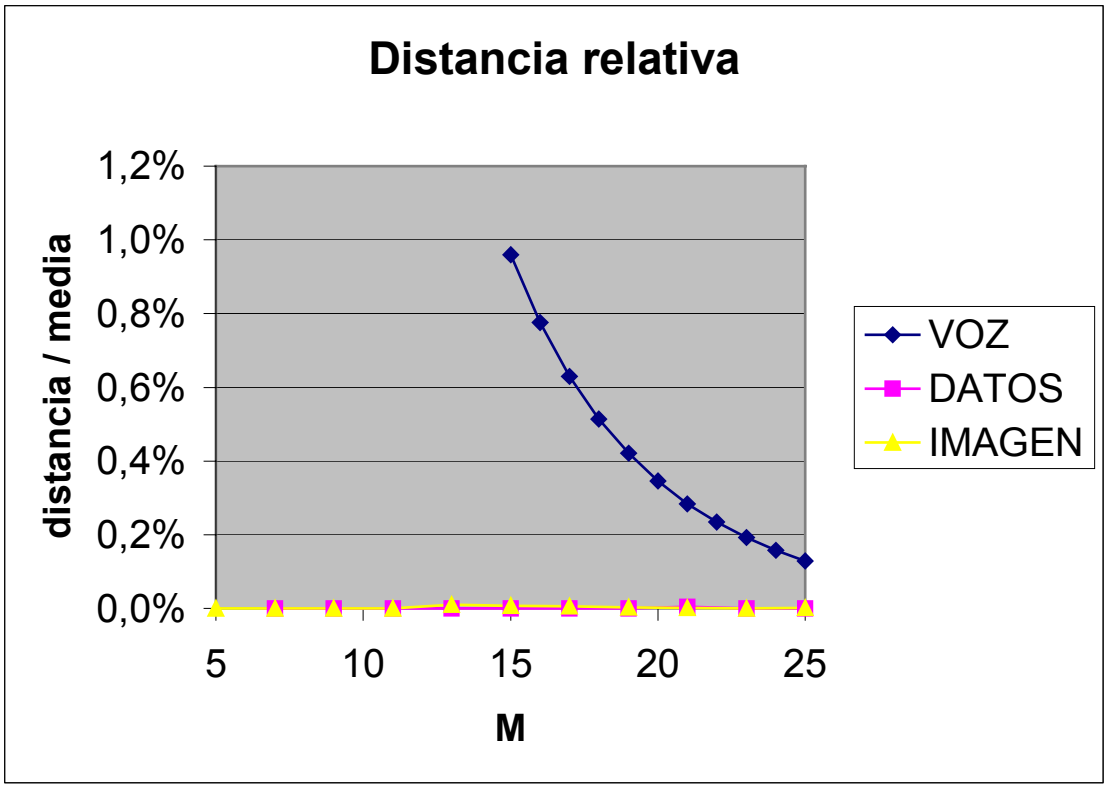
7.2 Estudio numérico.

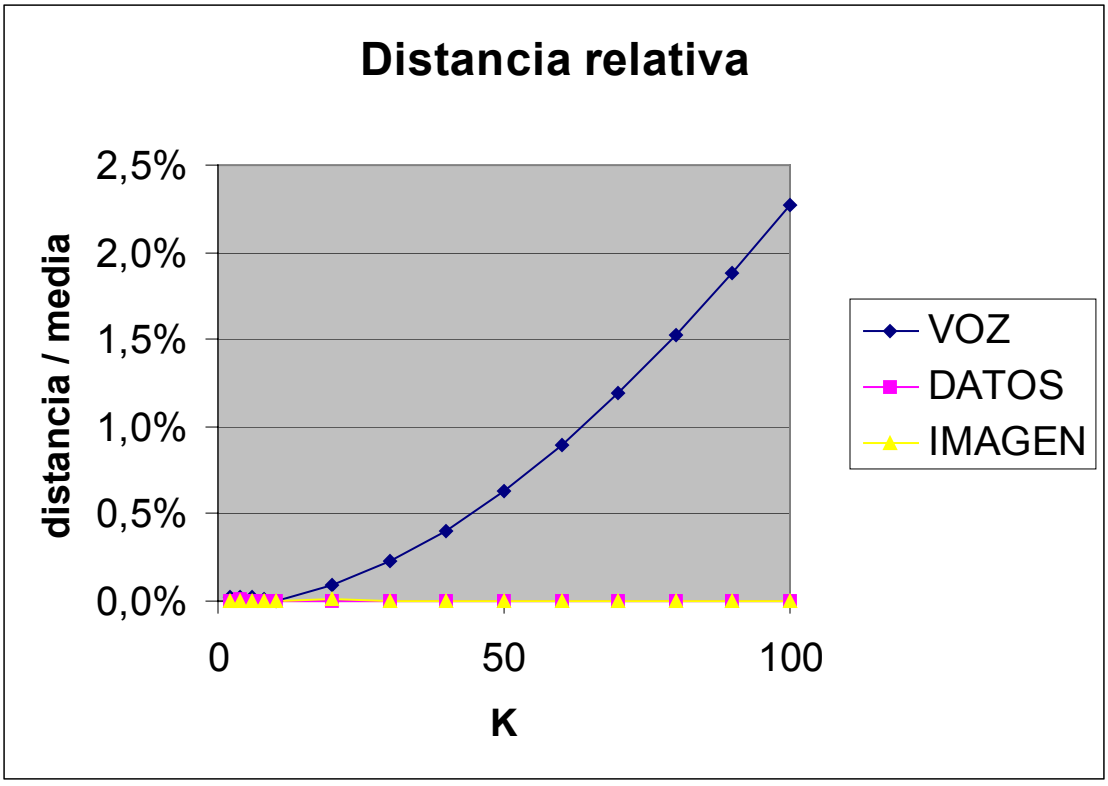
Observando las anteriores gráficas se puede observar que los resultados obtenidos a partir de cada una de las aproximaciones son, en la mayor parte de los casos, casi idénticos. Lo que hace que a nivel gráfico resulte complicado el distinguirlas. Para ayudarnos a comparar los resultados de una y otra aproximación vamos a representar de forma gráfica la distancia entre los dos resultados ponderado por la media de los resultados. Sería algo así como el porcentaje de diferencia entre ambas aproximaciones.

Lo haremos para todos los casos, quedando la información reducida a cuatro gráficas posibles: variación con M, en casos de M grande y pequeña; y variación con K, con Ks grandes y pequeñas.


Distancia relativa
distancia / media
0,8%
0,7%
0,6%
0,5%
0,4%
0,3%
0,2%
0,1%
0,0%
75
175
275
M
VOZ
DATOS
IMAGEN


Distancia relativa
distancia / media
2,5%
2,0%
1,5%
1,0%
0,5%
0,0%
0
50
100
K
VOZ
DATOS
IMAGEN


Distancia relativa
1,2%
1,0%
0,8%
0,6%
0,4%
0,2%
0,0%
distancia / media
5
10
15
20
25
M
VOZ
DATOS
IMAGEN


Distancia relativa
2,5%
2,0%
1,5%
1,0%
0,5%
0,0%
distancia / media
0
50
100
K
VOZ
DATOS
IMAGEN

Lo primero que me gustaría destacar es que debemos tener en cuenta que para los casos en que K es pequeña los resultados de las aproximaciones MMDP no son demasiado fiables. Esto es debido a que cuando nuestro buffer no es demasiado grande podemos tener una componente de pérdida de células debido a la congestión a nivel de célula que, debemos recordar, no está contemplada en nuestras aproximaciones. Esto se traduce en que en las ilustraciones 8.4-8.6 no son fiables los resultados con K menor que 25 y en las ilustraciones 8.10-8.12 no son fiables los resultados con K menor que 10.

Entre ambas aproximaciones es intuitiva y fácilmente demostrable que la aproximación M+1 MMDP es más ajustada a los resultados reales que pudieran obtenerse. Teniendo esto en mente y viendo las gráficas que nos muestran la diferencia relativa entre los resultados podemos observar que la aproximación 2 MMDP no se separa mucho de la M+1 MMDP en los casos de tráfico de datos o imágenes. Es decir, podríamos establecer que los resultados de la aproximación 2 MMDP son de una exactitud comparable a los de la aproximación M+1 MMDP en los casos en los que tenemos un tamaño de ráfaga suficiente (que a nivel empírico podríamos establecer en mayor que 300).

Cuando L es más pequeño (que en los casos que hemos estudiado se cumple con el tráfico de voz) es muy significativo observar que con la variación de M, los resultados van acercándose (de forma **relativa**) a medida que se incrementa el número de fuentes en el sistema. Mientras que con la variación de K, los resultados se separan (de forma **relativa**) a medida que K aumenta.

A nivel de eficiencia computacional debemos tener en cuenta que ésta, no sólo depende de la naturaleza del cálculo, sino también de la implementación del mismo que se ha llevado a cabo. En nuestro caso no nos hemos centrado en la eficiencia al implementarlo, sino en la claridad del código (siguiendo paso por paso el desarrollo teórico en cada caso) y su portabilidad. Es por ello, por lo que una comparativa exacta del tiempo de CPU empleado en el cálculo no nos parece adecuada. Ahora bien ambas aproximaciones requieren la resolución de sistemas lineales de ecuaciones, una forma de comparación sería a través del tamaño de la matriz de coeficientes de dicho sistema. En el caso de la aproximación M+1 MMDP éste es, aproximadamente (MK x MK), mientras que en el caso de la aproximación 2 MMDP la matriz tiene un tamaño de (2K x 2K). Aunque debemos tener en cuenta que la mayor parte de los elementos de la matriz, en el caso de M+1 MMDP, son nulos, simplificando mucho el cálculo, tampoco hay que olvidar que la aproximación 2MMDP tiene una solución directa, mientras que la aproximación M+1 MMDP es un algoritmo iterativo. Podemos pues concluir que, **cualitativamente**, el cálculo de la probabilidad de pérdida de célula es más rápido con la aproximación 2MMDP que con la aproximación M+1 MMDP.

8 CONCLUSIONES

En este proyecto se ha desarrollado una aproximación para el cálculo de la probabilidad de pérdida de célula en sistemas ATM alimentados con el tráfico generado por un número determinado de fuentes ON-OFF: la aproximación MMDP; cuyas principales ventajas frente a otros métodos de cálculo son su simplicidad, eficiencia computacional y estabilidad numérica.

El cálculo en sí de la probabilidad de pérdida de célula se ha resulto a través de dos posibles aproximaciones: M+1 MMDP y 2MMDP.

La aproximación M+1 MMDP nos da una exactitud en el cálculo suficiente en los casos en los que la congestión a nivel de ráfaga sea la razón de la pérdida de las células. Esta aproximación sería útil para aplicaciones como la planificación de redes, el desarrollo de esquemas fijos de reserva de ancho de banda o políticas de admisión de llamadas.

La aproximación 2MMDP nos da una exactitud similar a la anterior si, como antes, la congestión a nivel de ráfaga es la dominante y el tamaño medio de ráfaga generado por las fuentes ON-OFF es suficientemente grande (mayor que 300 células). La gran ventaja de esta aproximación respecto a la anterior es su eficiencia computacional, lo que la hace útil para los casos en que sea necesario un cálculo de la probabilidad de pérdida de célula en tiempo real.

8.1 Líneas futuras.

Vamos a proponer dos líneas en las que cuales podrían desarrollarse nuevos proyectos basados en el presente.

- Teniendo en cuenta la filosofía en la que se basa la aproximación 2MMDP podría desarrollarse una nueva aproximación para el cálculo de la probabilidad de pérdida

de célula extendiendo el número de estados a cualquiera entre 2 y M+1. En la aproximación 2 MMDP los M+1 posibles estados en virtud del número de fuentes activas en cada momento se agrupaban en sólo dos: una agrupación en la que estaban incluidos todos los estados cuyo tráfico era inferior a la tasa de servicio de la cosa y otra, en la que estaban los estados cuyo tráfico era igual o superior a la tasa de servicio. Esto nos simplificaba enormemente el cálculo. Lo que ahora se propone es no simplificar tanto el cálculo para obtener una mayor exactitud. La idea sería obtener un híbrido entre las aproximaciones presentadas en este proyecto que nos permitiría llegar a un compromiso entre la exactitud del cálculo y el tiempo empleado para ello, ajustando a voluntad el número de estados posibles de la cadena de Markov moduladora del tráfico.

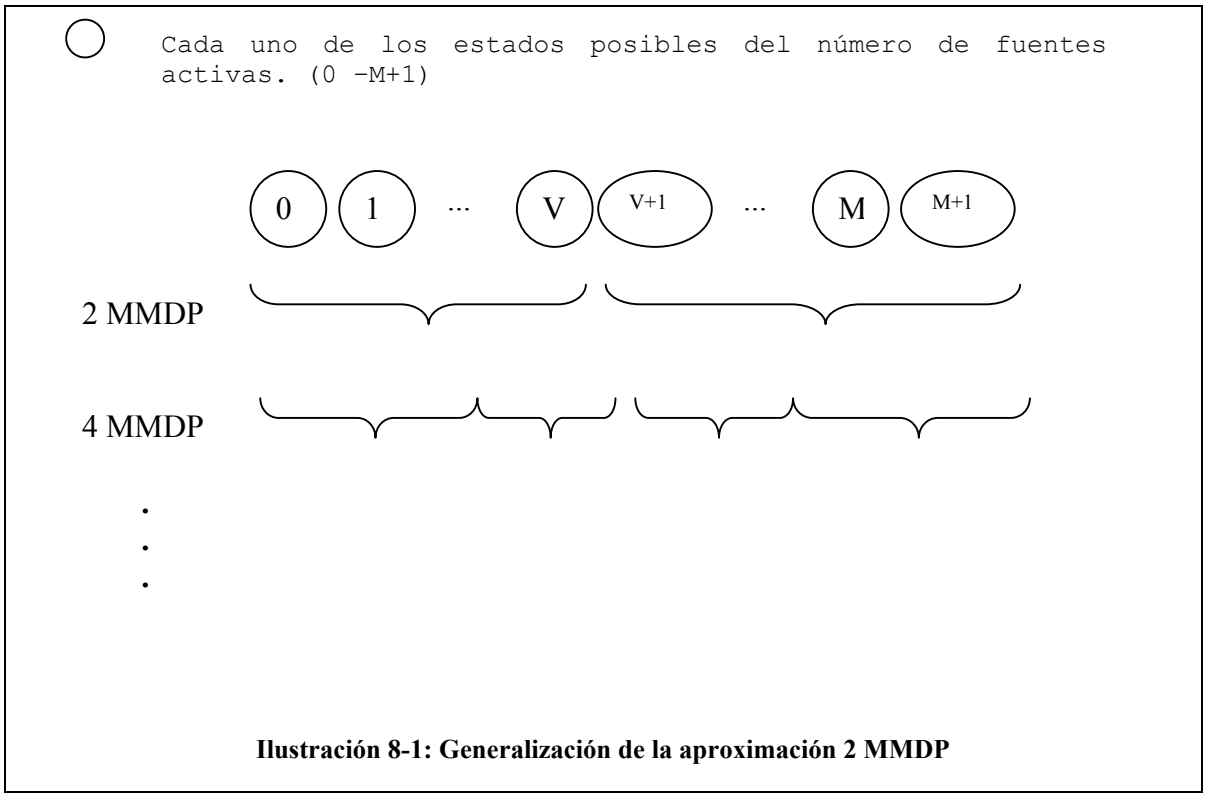


Ilustración 8-1: Generalización de la aproximación 2 MMDP

- La aproximación M+1 MMDP podría generalizarse para poder calcular la probabilidad de pérdida de célula en el caso de que tuviéramos diferentes tipos de tráfico a un tiempo en el sistema. Ello nos llevaría a que el proceso de Markov modulador fuera n-dimensional, donde n sería el número de diferentes tipos de fuentes, ya que ahora no sólo tendríamos la posibilidad de pasar del estado i al estado i+1 o i-1 (nacimiento o muerte respectivamente) sino que tendríamos 2*n posiblidades (por ejemplo en un caso bidimensional, es decir con sólo dos tipos de tráfico tendríamos que desde el estado {i,j} podríamos ir a los estados {i+1,j}; {i-1,j}; {i,j+1} y {i,j-1}) El algoritmo presentado en la sección 3 debería ser modificado para ello. Esto nos permitiría abordar escenarios de cálculo más realistas.

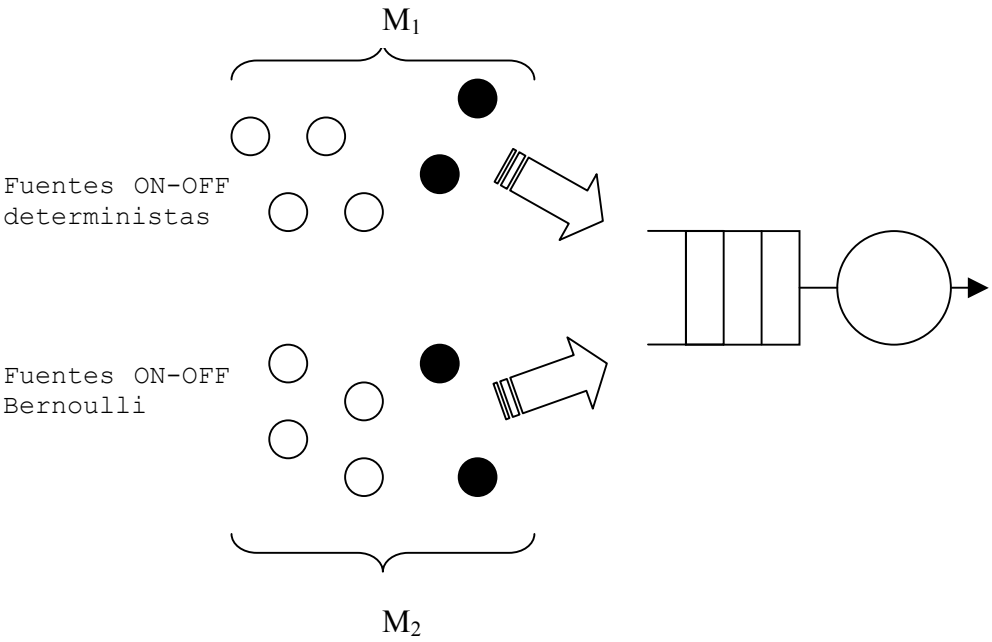


Ilustración 8-2: Ejemplo de generalización del algoritmo M+1 MMDP

9 PLANOS DEL PROYECTO

9.1 IMPLEMENTACIÓN M+1 MMDP

9.1.1 Código del fichero principal

```
#include <stdio.h>

#include <stdlib.h>


#include "F_Fichero.h"

#include "F_Lectura.h"

#include "F_Memoria.h"

#include "F_Matematicas.h"

#include "F_Calculo_Probabilidades.h"



void main (int argc, char *argv[])

{


   /* Declaración de variables */
```

```
FILE *Fichero_Entrada;
FILE *Fichero_Salida;

float Peak_Rate;     /* Este grupo son las condiciones iniciales */
float Mean_Rate;
int L;
float C;
int K;
float epsilon;
int Numero_Max_Iteraciones;
int M;

int Numero_Iteracion = 0;
double Probabilidad_Perdida = 0;
double N_Probabilidad_Perdida;
double **y;
double *alfa;
double *beta;
double *gamma;
double *p;
double ***A;

char codigo = 0;     /* Esto es un flag que nos dice el estado del
                      * programa:
                      * 0 .- Continuar iterando.
                      * 1 .- Se ha encontrado la solución.
                      * 2 .- Problemas al abrir los ficheros.
                      * 3 .- Problemas al leer los datos.
```

```
                        * 4 .- Problemas al reservar la memoria. */


int i,j;              /* Esto son variables auxiliares */

double *vector1;

double *vector2;

double *vector3;



/* Inicialización */



codigo=Apertura_Fichero (argc,argv,&Fichero_Entrada,&Fichero_Salida);


   if (codigo==0)

         {

         codigo=Lectura_Datos(&Peak_Rate, &Mean_Rate, &C, &L, &M, &K,

                     &epsilon,&Numero_Max_Iteraciones,
                     Fichero_Entrada);

         fclose(Fichero_Entrada);

         if (codigo!=0)

               fclose(Fichero_Salida);

         }

   if (codigo==0)

         {

         fprintf(Fichero_Salida,"\nDatos iniciales");

         fprintf(Fichero_Salida,"\nDatos iniciales");

         fprintf(Fichero_Salida,"\nPeak_Rate\t%f",Peak_Rate);
```

```
            fprintf(Fichero_Salida,"\nMean_Rate\t%f",Mean_Rate);

            fprintf(Fichero_Salida,"\nC\t%f",C);

            fprintf(Fichero_Salida,"\nL\t%d",L);

            fprintf(Fichero_Salida,"\nM\t%d",M);

            fprintf(Fichero_Salida,"\nK\t%d",K);

            fprintf(Fichero_Salida,"\nEpsilon\t%e",epsilon);

            fprintf(Fichero_Salida,"\nNúmero máximo de
iteraciones\t%d",Numero_Max_Iteraciones);

            fprintf(Fichero_Salida,"\n\nIteración\tResultado");


            codigo=Reserva_Memoria (&alfa, &beta, &gamma,

                        &p, &y, &A, M, K, &vector1, &vector2, &vector3);

      if (codigo!=0)

                  fclose(Fichero_Salida);

            }


      if (codigo==0)

            {

            Calculo_p (Peak_Rate, Mean_Rate, L, M, alfa, beta, gamma, p);

            Calculo_A (Peak_Rate, C, M, K, gamma, A);

            }


  /* Ahora empezamos lo que es el bucle en sí */


  while (codigo==0)

      {

      for (i = 0; i < M + 1; i++)

            {
```

```
      if (i == 0)
            Factor_x_Vector (beta[1], y[1], K, vector3);
      else if (i == M)
            Factor_x_Vector (alfa[M - 1], y[M - 1], K, vector3);
      else
            {
            Factor_x_Vector (alfa[i - 1], y[i - 1], K, vector1);
            Factor_x_Vector (beta[i + 1], y[i + 1], K, vector2);
            Vector_mas_Vector (vector1, vector2, K, vector3);
            }
Vector_x_Matriz (vector3, A[i], K, y[i]);
      Normalizacion_Vector (y[i], K);
      Factor_x_Vector (p[i], y[i], K, y[i]);
      }

N_Probabilidad_Perdida = Calculo_Probabilidad_Perdida (p, gamma, y,
                                          Peak_Rate, C, M, K);

if (epsilon>0)
      {
      if (N_Probabilidad_Perdida == 0)
            {
            if (epsilon > Valor_Absoluto
                    (N_Probabilidad_Perdida - Probabilidad_Perdida))
                  {
                  fprintf(Fichero_Salida,"\n...");
                  codigo = 1;
                  }
```

```
                }
            else
                {
                if (epsilon > Valor_Absoluto
(N_Probabilidad_Perdida-Probabilidad_Perdida/N_Probabilidad_Perdida))
                    {
                    fprintf(Fichero_Salida,"\n...");
                    codigo=1;
                    }
                }
            }

      if ((Numero_Max_Iteraciones>0) &&
                (Numero_Max_Iteraciones==Numero_Iteracion))
            codigo=1;

      Probabilidad_Perdida = N_Probabilidad_Perdida;
      Numero_Iteracion++;

      fprintf(Fichero_Salida,"\n%d\t%.25g",
                    Numero_Iteracion,Probabilidad_Perdida);
      /* printf("\n%d\t%.25g",Numero_Iteracion,Probabilidad_Perdida); */

      }

      /* Finalizamos el proceso cerrando
                        los ficheros y liberando memoria */
if (codigo==1)
```

```
	{
	fclose(Fichero_Salida);
	free(vector1);
	free(vector2);
	free(vector3);
	free(alfa);
	free(beta);
	free(gamma);
	free(p);
	for (i=0;i<M+1;i++)
	{
		for (j=0;j<K+1;j++)
			free(A[i][j]);
		free(A[i]);
		free(y[i]);
	}
	free(A);
	free(y);
	}
}
```

9.1.2 Cabeceras de las librerías de funciones

9.1.2.1 F_fichero.h

```
char Apertura_Fichero(int argc, char *argv[],
                      FILE **Fichero_Entrada, FILE **Fichero_Salida);
```

9.1.2.2 F_lectura.h

```
char leer_sin_comentarios (float * parametro, FILE * fichero);
char Lectura_Datos(float *Peak_Rate, float *Mean_Rate, float *C,
```

```
            int *L, int *M, int *K, float *epsilon,
            int *Numero_Max_Iteraciones, FILE * fichero);
```

9.1.2.3 F_memoria.h

```
char Reserva_Memoria(double **alfa, double **beta, double **gamma,
                     double **p, double ***y, double ****A, int M, int K,
                     double **vector1, double **vector2,double **vector3);
```

9.1.2.4 F_matematicas.h

```
double Productorio (double * variable, int indice_inicial,
                        int indice_final);
void Vector_x_Matriz (double *vector, double **matriz,int K,
                        double *vector_resultado);
void Factor_x_Vector (double factor, double *vector, int K,
                        double *vector_resultado);
void Vector_mas_Vector (double *vector1, double *vector2, int K,
                        double *vector_resultado);
void Normalizacion_Vector (double * vector, int K);
double Valor_Absoluto(double x);
```

9.1.2.5 F_calculo_probabilidades.h

```
double Calculo_p0(double *alfa, double *beta, int M);
      /* Esta función calcula p[0]
            Es usada desde Calculo_p ya que este cálculo es previo
            al de los p[i] con i>0 */
void Calculo_p(float Peak_Rate, float Mean_Rate, int L, int M,
            double *alfa, double *beta, double *gamma, double *p);
void Calculo_A(float Peak_Rate, float C, int M, int K,
      double * gamma, double ***A);
double Calculo_Probabilidad_Perdida(double *p, double *gamma, double **y,
```

```
            float Peak_Rate, float C, int M, int K);
```

9.1.3 Código de las librerías de funciones

9.1.3.1 F_fichero.c

```
#include <stdlib.h>

#include <string.h>

#include <stdio.h>


#define FICHERO_IN "entrada.txt"

#define FICHERO_OUT "salida.txt"
```

char Apertura_Fichero(int argc, char *argv[],
FILE **Fichero_Entrada, FILE **Fichero_Salida)

```
{
   char codigo_retorno=0;

   char Nombre_Fichero_In[255];

   char Nombre_Fichero_Out[255];


   if (argc==1)
     {
      strcpy(Nombre_Fichero_In,FICHERO_IN);
      strcpy(Nombre_Fichero_Out,FICHERO_OUT);
     }
   else if (argc==2)
     {
      strcpy(Nombre_Fichero_In,argv[1]);
      strcpy(Nombre_Fichero_Out,FICHERO_OUT);
     }
   else if (argc==3)
```

```
     {
      strcpy(Nombre_Fichero_In,argv[1]);
      strcpy(Nombre_Fichero_Out,argv[2]);
     }
   else
     {
      printf("\nEl uso de este programa es:");
      printf("\nalgoritmo [fichero_entrada [fichero_salida]]");
      printf("\nPor defecto el fichero de entrada se llamará
'entrada.txt'");
      printf("y el de salida 'salida.txt'");
      codigo_retorno=2;
     }

   if ((*Fichero_Entrada=fopen(Nombre_Fichero_In,"r"))==NULL)
     {
      printf("\nError al abrir el fichero de entrada de datos.");
      codigo_retorno=2;
     }
   else if ((*Fichero_Salida=fopen(Nombre_Fichero_Out,"w+"))==NULL)
        {
        fclose(*Fichero_Entrada);
      printf("\nError al abrir el fichero de salida de datos.");
      codigo_retorno=2;
     }

   return(codigo_retorno);
}
```

9.1.3.2 F_lectura.c

```
#include <stdlib.h>

#include <string.h>

#include <stdio.h>
```

char leer_sin_comentarios (float * parametro,
FILE * fichero)

```
{

   char codigo_retorno=0;

   int flag;

   char cadena[50];

   do

     {

      flag=fscanf(fichero,"%s",cadena);

      if(flag==EOF)

        {

           printf("\nEl fichero de entrada es erroneo. EOF encontrado y
no esperado.");

           codigo_retorno=3;

        }

      else if (strcmp(cadena,"/*")==0)

        {

           for
(;(strcmp(cadena,"*/")!=0)&&(flag==1);flag=fscanf(fichero,"%s",cadena));

           if(flag==EOF)

             {

              printf("\nEl fichero de entrada es erroneo. EOF encontrado
y no esperado.");

              codigo_retorno=3;

             }
```

```
            else
              flag=0;
         }
       else
         {
            *parametro=(float)atof(cadena);
            flag=1;
         }
      }
   while (flag==0);
   return(codigo_retorno);
}
```

char Lectura_Datos(float *Peak_Rate, float *Mean_Rate, float *C,
int *L, int *M, int *K,
float *epsilon,
int *Numero_Max_Iteraciones,
FILE * fichero)

```
{
   char codigo_retorno=0;
   float aux;

   codigo_retorno=leer_sin_comentarios(Peak_Rate,fichero);

   if (codigo_retorno==0)
     leer_sin_comentarios(Mean_Rate,fichero);

   if (codigo_retorno==0)
     leer_sin_comentarios(C,fichero);
```

```
if (codigo_retorno==0)
  {
   leer_sin_comentarios(&aux,fichero);
   *L=(int)aux;
  }

if (codigo_retorno==0)
  {
   leer_sin_comentarios(&aux,fichero);
   *M=(int)aux;
  }

if (codigo_retorno==0)
  {
   leer_sin_comentarios(&aux,fichero);
   *K=(int)aux;
  }

if (codigo_retorno==0)
  leer_sin_comentarios(epsilon,fichero);

if (codigo_retorno==0)
  {
   leer_sin_comentarios(&aux,fichero);
   *Numero_Max_Iteraciones=(int)aux;
  }
```

```
   return(codigo_retorno);
}
```

9.1.3.3 F_memoria.c

```
#include <stdlib.h>
```

char Reserva_Memoria(double **alfa, double **beta, double **gamma,
double **p, double ***y, double ****A, int M, int K,
double **vector1, double **vector2, double **vector3)

```
{
   char codigo_retorno=0;
   int i,j,x,z; /* Variables auxiliares.
           * Índices para las matrices */

   if (((*vector1)=calloc(K+1,sizeof(double)))==NULL)
     {
      printf("\nError al reservar memoria para vector1.\n");
      codigo_retorno=4;
     }
   else if (((*vector2)=calloc(K+1,sizeof(double)))==NULL)
     {
      printf("\nError al reservar memoria para vector2.\n");
      free(*vector1);
      codigo_retorno=4;
     }
   else if (((*vector3)=calloc(M+1,sizeof(double)))==NULL)
     {
      printf("\nError al reservar memoria para vector3.\n");
```

```
    free(*vector1);
    free(*vector2);
    codigo_retorno=4;
   }
 else if (((*alfa)=calloc(M+1,sizeof(double)))==NULL)
   {
    printf("\nError al reservar memoria para alfa.\n");
    free(*vector1);
    free(*vector2);
    free(*vector3);
    codigo_retorno=4;
  }
 else if (((*beta)=calloc(M+1,sizeof(double)))==NULL)
   {
    printf("\nError al reservar memoria para beta.\n");
    free(*vector1);
    free(*vector2);
    free(*vector3);
    free(*alfa);
    codigo_retorno=4;
   }
 else if (((*gamma)=calloc(M+1,sizeof(double)))==NULL)
   {
    printf("\nError al reservar memoria para gamma.\n");
    free(*vector1);
    free(*vector2);
    free(*vector3);
    free(*alfa);
```

```
      free(*beta);
      codigo_retorno=4;
     }
  else if (((*p)=calloc(M+1,sizeof(double)))==NULL)
     {
      printf("\nError al reservar memoria para p.\n");
      free(*vector1);
      free(*vector2);
      free(*vector3);
      free(*alfa);
      free(*beta);
      free(*gamma);
      codigo_retorno=4;
     }

  else if (((*A)=calloc(M+1,sizeof(double **)))==NULL)
     {
      printf("\nError al reservar memoria para A.\n");
      free(*vector1);
      free(*vector2);
      free(*vector3);
      free(*alfa);
      free(*beta);
      free(*gamma);
      free(*p);
      codigo_retorno=4;
     }
  else if (((*y)=calloc(M+1,sizeof(double *)))==NULL)
```

```
  {
   printf("\nError al reservar memoria para y.\n");
   free(*vector1);
   free(*vector2);
   free(*vector3);
   free(*alfa);
   free(*beta);
   free(*gamma);
   free(*p);
   free(*A);
   codigo_retorno=4;
  }



for (i=0;((i<M+1)&&(codigo_retorno==0));i++)
  {
   if (((*A)[i]=calloc(K+1,sizeof(double *)))==NULL)
     {
        printf("\nError al reservar memoria para A[%d].\n",i);
        free(*vector1);
        free(*vector2);
        free(*vector3);
        free(*alfa);
        free(*beta);
        free(*gamma);
        free(*p);
        for (j=0;j<i;j++)
          {
```

```
            free((*A)[j]);
            free((*y)[j]);
           }
         free(*A);
         free(*y);
         codigo_retorno=4;
      }
   else if (((*y)[i]=calloc(K+1,sizeof(double)))==NULL)
      {
         printf("\nError al reservar memoria para y[%n].\n",i);
         free(*vector1);
         free(*vector2);
         free(*vector3);
         free(*alfa);
         free(*beta);
         free(*gamma);
         free(*p);
         for (j=0;j<i;j++)
           {
            free((*A)[j]);
            free((*y)[j]);
           }
         free((*A)[i]);
         free(*A);
         free(*y);
         codigo_retorno=4;
      }
```

```
     }
   for (i=0;((i<M+1)&&(codigo_retorno==0));i++)
     for (j=0;((j<K+1)&&(codigo_retorno==0));j++)
        {
           if (((*A)[i][j]=calloc(K+1,sizeof(double)))==NULL)
             {
              printf("\nError      al      reservar      memoria      para
A[%d][%d].\n",i,j);
              free(*vector1);
              free(*vector2);
              free(*vector3);
              free(*alfa);
              free(*beta);
              free(*gamma);
              free(*p);
              for (x=0;x<i;x++)
                {
                   for (z=0;z<K+1;z++)
                   free((*A)[x][z]);
                   free((*A)[x]);
                   free((*y)[x]);
                }
              for (z=0;z<j;z++)
                free((*A)[i][z]);
              free((*A)[i]);
              free((*y)[i]);
              free(*A);
              free(*y);
```

```
              codigo_retorno=4;
             }

           if (codigo_retorno==0)
             (*y)[i][j]=1/(double)((M+1)*(K+1));
                                  /* Inicializamos y homogéneamente */
        }

   return(codigo_retorno);
}
```

9.1.3.4 F_matemáticas.c

double Productorio (double * variable, int indice_inicial, int indice_final)

```
{
   /* Esta función evalua el productorio de los elementos de un vector
    * desde indice_inicial a indice_final.
    * Esto se usa para evaluar los productorios de beta y alfa en
    * el algoritmo:
    * beta[1]*beta[2]*...*beta[i]
    * alfa[0]*alfa[1]*...*alfa[i-1]
    * */

   double resultado;

   if (indice_inicial!=indice_final)

resultado=variable[indice_inicial]*Productorio(variable,indice_inicial+1,
indice_final);
   else
     resultado=variable[indice_final];
```

```
   return(resultado);

}
```

void Vector_x_Matriz (double *vector, double **matriz,int K, double *vector_resultado)

```
{

   /* Esta función evalua el producto escalar entre un vector y una
matriz.

    * El vector es de 1xK+1 y la matriz K+1xK+1.

    * En el algoritmo se usa para evaluar los productos y[i]*A[i] */


   int i,j;


   for (i=0;i<K+1;i++)

     {

      vector_resultado[i]=0;

      for (j=0;j<K+1;j++)

        vector_resultado[i]+=vector[j]*matriz[j][i];

     }

}
```

void Factor_x_Vector (double factor, double *vector, int K, double *vector_resultado)

```
{

   int i;

   for (i=0;i<K+1;i++)

     vector_resultado[i]=factor*vector[i];

}
```

void Vector_mas_Vector (double *vector1, double *vector2, int K, double *vector_resultado)

```
{
   int i;
   for (i=0;i<K+1;i++)
     vector_resultado[i]=vector1[i]+vector2[i];
}



void Normalizacion_Vector (double * vector, int K)
{
   int i;
   double modulo=0;
   for (i=0;i<K+1;i++)
     modulo+=vector[i];

   if(modulo==0)
     {
      /* printf("\nAtención. Normalizando un vector de módulo 0 a 0.\n");
       exit(EXIT_FAILURE); */
     }
   else
     for (i=0;i<K+1;i++)
     {
      vector[i]=vector[i]/modulo;
     }
}
```

double Valor_Absoluto(double x)

```
{
   if (x>=0)
     return(x);
   else
     return (-x);
}
```

9.1.3.5 F_calculo_probabilidades.c

```
#include <math.h>
#include <stdio.h>
#include "F_Matematicas.h"
```

double Calculo_p0(double *alfa, double *beta, int M)

```
                  /* Esta función calcula p[0]
                * Es usada desde Calculo_p ya que este cálculo es previo
                * al de los p[i] con i>0 */
{
   double suma=0;
   int i;

   for (i=1;i<M+1;i++)
     suma+=(Productorio(alfa,0,i-1)/Productorio(beta,1,i));

   suma+=1;

   if (suma==0)
```

```
     {
      printf("\nError al calcular p[0]\n");
      return(-1);
     }
   else return(1/suma);
}
```

void Calculo_p(float Peak_Rate, float Mean_Rate, int L, int M,
double *alfa, double *beta, double *gamma, double *p)

```
{
   int i; /* Variable auxiliar.
         * Índice. */

   double nu=Peak_Rate/(L*424);           /* 424 es el número de bits
                                  * que contiene una célula. */
   double lambda=nu/((Peak_Rate/Mean_Rate)-1);

   for (i=0;i<M+1;i++)
     {
      gamma[i]=(M-i)*lambda+i*nu;
      alfa[i]=(M-i)*lambda/gamma[i];
      beta[i]=i*nu/gamma[i];
     }

   p[0]=Calculo_p0(alfa, beta, M);

   for (i=1;i<M+1;i++)
```

```
     p[i]=(Productorio(alfa,0,i-1)/Productorio(beta,1,i))*p[0];


}
```

void Calculo_A(float Peak_Rate, float C, int M, int K, double * gamma, double ***A)

```
{
   int i,k,h; /* Variables auxiliares */
   double ro;


   for (i=0;i<M+1;i++)
     {
      if (C==i*Peak_Rate)
        {
           for (k=0;k<K+1;k++)
             for (h=0;h<K+1;h++)
             {
                if ((k==h)&&(h==0))
                  A[i][k][h]=1-exp(-gamma[i]/(double)C);
                else if ((k==0)&&(h==1))
                    A[i][k][h]=exp(-gamma[i]/(double)C);
                else if ((k==h)&&(h>0))
                  A[i][k][h]=1;
                else
                  A[i][k][h]=0;
             }
        }
      else if (C>(i*Peak_Rate))
```

```
    {
       ro=exp(-gamma[i]/(double)(C-i*Peak_Rate));

       for (k=0;k<K+1;k++)
         for (h=0;h<K+1;h++)
         {
          if (k<h)
            A[i][k][h]=0;
          else if ((k>=h)&&(h>=1))
            A[i][k][h]=pow(ro,k-h)*(1-ro);
          else
            A[i][k][h]=pow(ro,k);
         }
    }
  else
    {
       ro=exp(-gamma[i]/(double)(i*Peak_Rate-C));

       for (k=0;k<K+1;k++)
         for (h=0;h<K+1;h++)
         {
          if ((k<=h)&&(h==K))
            A[i][k][h]=pow(ro,K-k);
          else if ((k<=h)&&(h<K))
            A[i][k][h]=pow(ro,h-k)*(1-ro);
          else
            A[i][k][h]=0;
         }
```

```
        }
      }
}
```

double Calculo_Probabilidad_Perdida(double *p, double *gamma, double **y, float Peak_Rate, float C, int M, int K)

```
{
   double e, ro;
   double denominador=0;
   double numerador=0;
   int i;
      double v=C/Peak_Rate;
      double den_aux;
      double num_aux;

      for (i=1;i<M+1;i++)
        {
      den_aux=0;
      e=exp(-gamma[i]/(double)(i*Peak_Rate));
      den_aux=p[i]*e/(1-e);
      denominador+=den_aux;
      if (i>v)
        {
              ro=exp(-gamma[i]/(double)(i*Peak_Rate-C));
              num_aux=ro*y[i][K]/(1-ro);
              numerador+=num_aux;
        }
```

```
     }


   if (numerador==0)

     printf("\nAtención. El numerador en el cálculo de la probabilidad de
pérdida es nulo.\n");

   if (denominador==0)

     {

      printf("\nError. División por cero al calcular la probabilidad de
pérdida.\n");

      return(-1);

     }


   return(numerador/denominador);

}
```

9.2 IMPLEMENTACIÓN 2 MMDP

9.2.1 Código del fichero principal

```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>


#include "F_Fichero.h"

#include "F_Lectura.h"

#include "F_Calculo_Probabilidades.h"


void main (int argc, char *argv[])

{


   /* Declaración de variables */


   FILE *Fichero_Entrada;

   FILE *Fichero_Salida;


                        /* Este grupo son las condiciones iniciales */


   float Peak_Rate;           // Tasa de pico de entrada de células (bps)

   float Mean_Rate;           // Tasa media de entrada de células (bps)

   int L;                     // Tamaño de ráfaga (células)

   float C;                   // Tasa de servicio de células (bps)

   int K;                     // Tamaño máximo del buffer (células)

   int M;                     // Número de fuentes máximo
```

```
                            // Parámetros intermedios

double gamma[2];            // Inversa del tiempo medio de estancia
                            // en el estado i
double B[2];                // Tasa media de llegadas de
                            //células condicional (al estado)

double eta[2];        // Variables intermedias usadas en el
                      //cálculo final de
double ro[2];         // la probabilidad de pérdida de célula.


double Probabilidad_Perdida = 0; // Resultado

   char codigo = 0;         /* Esto es un flag que nos dice el
                             * estado del programa:
                      * 0 .- Continuar iterando.
                      * 1 .- Se ha encontrado la solución.
                      * 2 .- Problemas al abrir los ficheros.
                      * 3 .- Problemas al leer los datos.
                      * 4 .- Problemas al reservar la memoria. */

/* Inicialización */


codigo=Apertura_Fichero (argc,argv,&Fichero_Entrada,&Fichero_Salida);

   if (codigo==0)
```

```
        {
        codigo=Lectura_Datos(&Peak_Rate, &Mean_Rate, &C, &L, &M, &K,
                        Fichero_Entrada);
        fclose(Fichero_Entrada);
        if (codigo!=0)
          fclose(Fichero_Salida);
        }
 if (codigo==0)
   {
 fprintf(Fichero_Salida,"\nDatos iniciales");
 fprintf(Fichero_Salida,"\nPeak_Rate\t%f",Peak_Rate);
 fprintf(Fichero_Salida,"\nMean_Rate\t%f",Mean_Rate);
 fprintf(Fichero_Salida,"\nC\t%f",C);
 fprintf(Fichero_Salida,"\nL\t%d",L);
 fprintf(Fichero_Salida,"\nM\t%d",M);
 fprintf(Fichero_Salida,"\nK\t%d",K);
}



 // Cálculo de variable intermedias y resultado final.


 if (codigo==0)
        {
   Calculo_B_gamma (B, gamma, Peak_Rate, Mean_Rate, C, M, L);



   eta[0]=exp(-gamma[0]/B[0]);
   eta[1]=exp(-gamma[1]/B[1]);
```

```
        ro[0]=exp(-gamma[0]/(C-B[0]));
        ro[1]=exp(-gamma[1]/(B[1]-C));

        Probabilidad_Perdida=Calculo_Probabilidad_Perdida(ro,eta,K);

        fprintf(Fichero_Salida,"\n%f",Probabilidad_Perdida);
        codigo=1;

            /* Finalizamos el proceso cerrando el fichero de salida */
      fclose(Fichero_Salida);
       }
}
```

9.2.2 Cabeceras de las librerías de funciones

9.2.2.1 F_fichero.h

```
char Apertura_Fichero(int argc, char *argv[],
            FILE **Fichero_Entrada, FILE **Fichero_Salida);
```

9.2.2.2 F_lectura.h

```
char leer_sin_comentarios (float * parametro, FILE * fichero);
char Lectura_Datos(float *Peak_Rate, float *Mean_Rate, float *C,
              int *L, int *M, int *K, FILE * fichero);
```

9.2.2.3 F_calculo_probabilidades.h

```
double Calculo_p0(double *alfa, double *beta, int M);
            /* Esta función calcula p[0]
            * Es usada desde Calculo_p ya que este cálculo es previo
            * al de los p[i] con i>0 */
void Calculo_q(double nu, double lambda, int M,
```

```
             double *q);
void Calculo_B_gamma(double *B, double *gamma, float Peak_Rate,
                  float Mean_Rate, float C, int M, int L);
double Calculo_Probabilidad_Perdida(double *ro, double *eta, int K);
```

9.2.3 Código de las librerías de funciones

9.2.3.1 F_fichero.c

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

#define FICHERO_IN "entrada.txt"
#define FICHERO_OUT "salida.txt"
```

char Apertura_Fichero(int argc, char *argv[],
FILE **Fichero_Entrada, FILE **Fichero_Salida)

```
{
   char codigo_retorno=0;
   char Nombre_Fichero_In[255];
   char Nombre_Fichero_Out[255];

   if (argc==1)
     {
      strcpy(Nombre_Fichero_In,FICHERO_IN);
      strcpy(Nombre_Fichero_Out,FICHERO_OUT);
     }
   else if (argc==2)
     {
```

```
      strcpy(Nombre_Fichero_In,argv[1]);
      strcpy(Nombre_Fichero_Out,FICHERO_OUT);
     }
   else if (argc==3)
     {
      strcpy(Nombre_Fichero_In,argv[1]);
      strcpy(Nombre_Fichero_Out,argv[2]);
     }
   else
     {
      printf("\nEl uso de este programa es:");
      printf("\nalgoritmo [fichero_entrada [fichero_salida]]");
      printf("\nPor defecto el fichero de entrada se llamará
'entrada.txt'");
      printf("y el de salida 'salida.txt'");
      codigo_retorno=2;
     }

   if ((*Fichero_Entrada=fopen(Nombre_Fichero_In,"r"))==NULL)
     {
      printf("\nError al abrir el fichero de entrada de datos.");
      codigo_retorno=2;
     }
   else if ((*Fichero_Salida=fopen(Nombre_Fichero_Out,"w+"))==NULL)
        {
        fclose(*Fichero_Entrada);
      printf("\nError al abrir el fichero de salida de datos.");
      codigo_retorno=2;
```

```
     }


   return(codigo_retorno);
}
```

9.2.3.2 F_lectura.c

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
```

char leer_sin_comentarios (float * parametro, FILE * fichero)

```
{
   char codigo_retorno=0;
   int flag;
   char cadena[50];
   do
     {
      flag=fscanf(fichero,"%s",cadena);
      if(flag==EOF)
        {
           printf("\nEl fichero de entrada es erroneo. EOF encontrado y
no esperado.");
           codigo_retorno=3;
        }
      else if (strcmp(cadena,"/*")==0)
        {
           for
(;(strcmp(cadena,"*/")!=0)&&(flag==1);flag=fscanf(fichero,"%s",cadena));
           if(flag==EOF)
```

```
             {
              printf("\nEl fichero de entrada es erroneo. EOF encontrado
y no esperado.");
              codigo_retorno=3;
             }
           else
             flag=0;
        }
      else
        {
           *parametro=(float)atof(cadena);
           flag=1;
        }
     }
   while (flag==0);
   return(codigo_retorno);
}
```

char Lectura_Datos(float *Peak_Rate, float *Mean_Rate, float *C,

int *L, int *M, int *K,

FILE * fichero)

```
{
   char codigo_retorno=0;
   float aux;

   codigo_retorno=leer_sin_comentarios(Peak_Rate,fichero);
```

```
   if (codigo_retorno==0)
     leer_sin_comentarios(Mean_Rate,fichero);

   if (codigo_retorno==0)
     leer_sin_comentarios(C,fichero);

   if (codigo_retorno==0)
     {
      leer_sin_comentarios(&aux,fichero);
      *L=(int)aux;
     }

   if (codigo_retorno==0)
     {
      leer_sin_comentarios(&aux,fichero);
      *M=(int)aux;
     }

   if (codigo_retorno==0)
     {
      leer_sin_comentarios(&aux,fichero);
      *K=(int)aux;
     }

   return(codigo_retorno);
}
```

9.2.3.3 F_calculo_probabilidades.c

```
#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include "F_Matematicas.h"
```

double Calculo_p0(double *alfa, double *beta, int M)

```
                  /* Esta función calcula p[0]

                * Es usada desde Calculo_p ya que este cálculo es previo

                * al de los p[i] con i>0 */

{

   double suma=0;

   int i;


   for (i=1;i<M+1;i++)

     suma+=(Productorio(alfa,0,i-1)/Productorio(beta,1,i));


   suma+=1;


   if (suma==0)

     {

      printf("\nError al calcular p[0]\n");

      return(-1);

     }

   else return(1/suma);

}
```

void Calculo_q(double nu, double lambda, int M, double *q)

```
{
   int i; // Variable auxiliar. Indice


   double *alfa;

   double *beta;

   double *gamma;

   double *p;


   double denominador;



   alfa=calloc(M+1,sizeof(double));

   beta=calloc(M+1,sizeof(double));

   gamma=calloc(M+1,sizeof(double));

   p=calloc(M+1,sizeof(double));



   for (i=0;i<M+1;i++)

     {

      gamma[i]=(M-i)*lambda+i*nu;

      alfa[i]=(M-i)*lambda/gamma[i];

      beta[i]=i*nu/gamma[i];

     }


   p[0]=Calculo_p0(alfa, beta, M);


   for (i=1;i<M+1;i++)

     p[i]=(Productorio(alfa,0,i-1)/Productorio(beta,1,i))*p[0];
```

```
   denominador=0;
   for (i=0;i<M+1;i++)
         denominador=denominador+(p[i]/gamma[i]);

   for (i=0;i<M+1;i++)
         q[i]=p[i]/(gamma[i]*denominador);


}
```

void Calculo_B_gamma(double *B, double *gamma, float Peak_Rate,

float Mean_Rate, float C, int M, int L)

```
{
      double *q;
      double pul;
      int v;
      int i,j;

   double nu=Peak_Rate/(L*424);           /* 424 es el número de bits
                                * que contiene una célula. */
   double lambda=nu/((Peak_Rate/Mean_Rate)-1);

      q=calloc(M+1,sizeof (double));


      Calculo_q(nu, lambda, M, q);
```

```
      pul=0;
      for (i=0; i<v+1;i++)
            pul=pul+q[i];



      B[0]=0; B[1]=0;
      for (i=1; i<M+1;i++)
      {
            if(i<v+1)
                  j=0;
            else
                  j=1;
            B[j]=B[j]+i*Peak_Rate*q[i];
      }
      B[0]=B[0]/pul;
      B[1]=B[1]/(1-pul);


   gamma[0]=1/(M*lambda);
   for (i=1;i<v+1;i++)
         gamma[0]=(1+i*nu*gamma[0])/((M-i)*lambda);


   gamma[1]=gamma[0]*(1-pul)/pul;


   gamma[0]=1/gamma[0];
   gamma[1]=1/gamma[1];


}
```

double Calculo_Probabilidad_Perdida(double *ro, double *eta, int K)

```
{

      double resultado;



      if (ro[0]!=ro[1])

            resultado=(ro[0]-ro[1])*(1-eta[0])*(1-eta[1])/((1-
ro[1])*((pow(ro[0]/ro[1],K+1)*(1-ro[1]))-(1-ro[0]))*(eta[0]-
2*eta[0]*eta[1]+eta[1]));

      else

            resultado=ro[1]*(1-eta[0])*(1-eta[1])/((1-ro[1])*(K*(1-
ro[1])+1)*(eta[0]-2*eta[0]*eta[1]+eta[1]));



      return(resultado);



}
```

9.3 IMPLEMENTACIÓN EN MATLAB M+1 MMDP

La presente implementación se llevó a cabo como una forma diferente de cálculo que nos permitiera comprobar la bondad de los resultados obtenidos con la implementación en C.

Se ha añadido al presente proyecto como un valor añadido al mismo.

9.3.1 Código del fichero principal

```
function [Probabilidad_Perdida, Iteracion]=algorit (Peak_Rate, Mean_Rate,
L, C, K, epsilon, M)


% Las tildes han sido suprimidas conscientemente en este programa

% para evitar posibles problemas con el interprete de comandos.


Iteracion=0;

Probabilidad_Perdida=0;

Continuar=1;


y=ones(M+1, K+1);

y=y*1/((M+1)*(K+1));




[alfa beta gamma p ro]= Calc_p (Peak_Rate, Mean_Rate, C, L, M);


while Continuar == 1

      fprintf (1, '\nEmpezamos la iteración %d\n', Iteracion)

      for i = 0:M

            fprintf (1,' %d',i)

            if rem (i, 30) == 0

                  fprintf(1,'\n')

            end
```

```
            if i==0

                  y(1,:)=beta(2)*y(2,:)*Calc_A(Peak_Rate,C,K,gamma,ro,i);

            elseif i == M


      y(M+1,:)=alfa(M)*y(M,:)*Calc_A(Peak_Rate,C,K,gamma,ro,i);

            else

y(i+1,:)=(alfa(i)*y(1,:)+beta(i+2)*y(i+2,:))*Calc_A(Peak_Rate,C,K,gamma,r
o,i);

            end



            y(i+1,:)=p(i+1)*y(i+1,:)/sum(y(i+1,:));

      end



      numerador=zeros(M,1);

      denominador=zeros(M,1);




      for i=1:M

denominador(i)=p(i+1)*exp(-gamma(i+1)/(i*Peak_Rate))/(1-exp

(-gamma(i+1)/i*Peak_Rate)));


if i > C/Peak_Rate

            numerador(i)=ro(i+1)/(1-ro(i+1))*y(i+1,K+1);

end


%     if rem(i,30) ==0

%     sprintf('\nEv. num / ro y den\n%d %e / %e %e', i, numerador(i),
ro(i+1), denominador(i))
```

```
%     pause;

%     end;


      N_Probabilidad_Perdida = sum(numerador)/sum(denominador)


if                        (abs(Probabilidad_Perdida                        -
N_Probabilidad_Perdida)/N_Probabilidad_Perdida >= epsilon)

            Iteracion = Iteracion+1;

            Probabilidad_Perdida=N_Probabilidad_Perdida;

            fprintf(1,'\n%d\t%e',Iteracion,N_Probabilidad_Perdida)

      else

            Continuar=0;

      end

end
```

9.3.2 Código de las funciones auxiliares

9.3.2.1 Calc_p

```
function [alfa, beta, gamma, p, ro] = Calc_p (Peak_Rate, Mean_Rate, C, L,
M)


p=zeros(M+1,1);


nu=Peak_Rate/(L*424);

lambda=nu/((Peak_Rate/Mean_Rate)-1);


alfa=zeros(M+1,1);

beta=zeros(M+1,1);

gamma=zeros(M+1,1);

ro=zeros(M+1,1);
```

```
for i=0:M
      gamma(i+1)=(M-i)*lambda+i*nu;
      alfa(i+1)=(M-i)*lambda/gamma(i+1);
      beta(i+1)=i*nu/gamma(i+1);
      if i*Peak_Rate < C
            ro(i+1) = exp(-gamma(i+1) / (C-i*Peak_Rate));
      elseif i*Peak_Rate > C
            ro(i+1) = exp(-gamma(i+1) / (i*Peak_Rate-C));
      end
end

% Empezaremos calculando p(0) (de 1 en este caso debido a la numeración
% de los vectores en Matlab

p_alfa=zeros(M,1);
p_beta=zeros(M,1);

for i=1:M
      p_alfa(i) = prod(alfa(1:i));
      p_beta(i)=prod(beta(2:i+1));
end

suma=1;
for i=1:M
      suma=suma+p_alfa(i)/p_beta(i);
end

p(1)=1/suma;
```

```
for i=2:M+1
      p(i)=(p_alfa(i-1)/p_beta(i-1))*p(1);
end
```

9.3.2.2 Calc_A

```
function A=Calc_A(Peak_Rate, C, K, gamma, ro, i)

A=zeros(K+1,K+1);

if i*Peak_Rate == C
      for k=0:K
            for h=0:K
                  if k==0
                        if h==0
                              A(k+1,h+1)=1-exp(-gamma(i+1)/C);
                        elseif h==1
                              A(k+1,h+1)=exp(-gamma(i+1)/C);
                        end
                  elseif k ==h
                        A(k+1,h+1)=1;
                  else
                        A(k+1,h+1)=0;
                  end
            end
      end
elseif i*Peak_Rate < C
      for k=0:K
            for h=0:K
                  if k <h
```

```
                        A(k+1,h+1)=0;
                  elseif k >=h
                        if h >=1
                              A(k+1,h+1)=(ro(i+1)^(k-h))*(1-ro(i+1));
                        elseif h==0
                              A(k+1,h+1)=ro(i+1)^k;
                        end
                  end
            end
      end
elseif i*Peak_Rate > C
      for k=0:K
            for h=0:K
                  if k <=h
                        if h==K
                              A(k+1,h+1)=ro(i+1)^(K-k);
                        elseif h<K
                              A(k+1,h+1)=(ro(i+1)^(h-k))*(1-ro(i+1));
                        end
                  elseif h<k
                        A(k+1,h+1)=0;
                  end
            end
      end
end
```

10 LISTA DE SÍMBOLOS Y ACRÓNIMOS

Para facilitar la búsqueda de los símbolos, estos se encuentran en orden alfabético:

A_i	Matriz de probabilidades de transición de estado del buffer de la cola condicionado al estado i, $i \in S$ de la cadena de Markov. Dimensión (K+1) x (K+1).
$a^i_{k,h}$	Elemento de la matriz A_i $= P\{Y_{n+1} = h \mid X_n = i, Y_n = k\}$ Probabilidad de transición del estado del buffer de la cola de *k* a *h,* en el estado de la cadena de markov i, $i \in S$.
B	Vector de tasas de llegada de células en cada uno de los estados i, $i \in S$.
B_i	Elemento del vector B. Tasa de llegada de células en el estado i, $i \in S$.
b	parámetro de ráfaga. Relación entre la tasa de pico (Δ) y la tasa media (Φ) $= \Delta / \Phi$
C	Tasa de servicio de la cola.

K	Tamaño máximo de la cola.
L	Tamaño medio de ráfaga. Número medio de células generados durante el periodo ON.
M	Número de fuentes ON-OFF en el sistema.
m	Número de estados posibles de la cadena de markov . $\{X(t); t \geq 0\}$
MMDP	*Markov Modulated Deterministic Process.* Método por el cual se aproxima el tráfico generado por un número M de fuentes ON-OFF independientes, por un tráfico determinista cuya tasa está modulada por una cadena de Markov.
N(t)	Número de fuentes activas en el momento *t.*
$N_{i,j}$	Número de células generadas durante un periodo caracterizado por $\{X_n = i; Y_n = j\}$
ON	Estado de la fuente ON-OFF caracterizado por un tiempo de permanencia aleatorio de tipo exponencial de parámetro $1/\lambda$ y tasa de generación de tráfico Δ.
OFF	Estado de la fuente ON-OFF caracterizado por un tiempo de permanencia aleatorio de tipo exponencial de parámetro $1/\mu$ y tasa de generación de tráfico nula.
P	Matriz de probabilidad de transición de estado de la cadena de markov $\{X(t); t \geq 0\}$. Dimensión m x m.

$p_{i,j}$	Elemento de la matriz P. $= P[X_{n+} = j \mid X_n = i]$ Probabilidad de pasar del estado i al j.
p_i	Probabilidad del estado i de la cadena de Markov. $P\{X_n = i\} \quad (= \pi_i e^T)$
p	Probabilidad de pérdida de célula del sistema.
Q	Matriz de probabilidad de transición de estado de la cadena de markov $\{(X_n, Y_n); n \geq 0\}$.
$q_{(i,K),(j,h)}$	Elemento de la matriz Q. Probabilidad de pasar del estado i $\{X_n = i, Y_n = k\}$ al estado $\{X_{n+1} = j, Y_{n+1} = h\}$
q_i	Elemento del vector q, que es la distribución de probabilidad del número de fuentes activas. Así q_i sería la probabilidad de tener i fuentes ON-OFF activas en un momento dado.
$R_{i,j}$	Número de células perdidas en un periodo caracterizado por $X_n = i$ y $Y_n = j$
S	Espacio de estados de la cadena de Markov $\{X(t); t \geq 0\}$
U_i	Tiempo que el sistema pasa en el estado i.
v	$= \max\{i : i\Delta \leq C\}$ Es el número máximo de fuentes activas cuyo tráfico combinado es menor o igual que la tasa de servicio.
X(t)	Cadena de Markov moduladora del tráfico del sistema.

X_n	$X_n \equiv X\left(\zeta_n^+\right)$
	Es la cadena de Markov reducida a los instantes en los que se produce un cambio de estado.
Y(t)	Número de células en la cola en cada instante t.
Y_n	$Y_n \equiv Y\left(\zeta_n^+\right)$
$y_{i,j}$	$P\left\{X(\xi_n^-) = i, Y(\xi_n^-) = j\right\}$
y_i	$= \pi_i A_i$
	Vector de valores de $y_{i,j}$ con j = 0... K
α_i	$\alpha_i = (M - i)\lambda / \gamma_i$
	para i=0,1,...,M
β_i	$\beta_i = i\mu / \gamma_i$ $\quad (\beta_i = 1 - \alpha_i)$
	para i=0,1,...,M
γ_i	$\gamma_i = (M - i)\lambda + i\mu$
	para i=0,1,...,M
φ_i	Inversa del tiempo medio de permanencia de la cadena de Markov en cada uno de sus estados.
Γ	$\Gamma = (\varphi_0, \varphi_1, \ldots, \varphi_{m-1})$
Δ	Tasa de pico de una fuente ON-OFF individual

η_i	$= e^{-\gamma_i / B_i}$ para cada uno de los estados de la cadena de Markov.
Φ	Tasa media de una fuente ON-OFF individual
λ	Inversa del tiempo medio de permanencia de la fuente ON-OFF en el estado ON.
μ	Inversa del tiempo medio de permanencia de la fuente ON-OFF en el estado OFF
ξ_n	Punto temporal enésimo de transición del estado de la cadena de Markov.
$\pi_{i,j}$	$P\{X(\xi_n^+) = i, Y(\xi_n^+) = j\}$
ρ_i	$= \begin{cases} e^{-\gamma_i / B_i - C} & B_i > C \\ e^{-\gamma_i / C - B_i} & B_i < C \end{cases}$

11 FÓRMULAS Y ECUACIONES

$$b = 1 + \frac{\lambda}{\mu} \qquad \Phi = \frac{\Delta}{b} \quad L = \frac{\Delta}{\mu} \quad \textbf{(E 2-1)} \ldots\ldots 10$$

$$q_{(i,k),(j,h)} = P\{Y_{n+1} = h \mid X_n = i, Y_n = k\} p_{i,j} = a_{k,h}^{i} p_{i,j} \qquad \text{(E 2-2)} \ldots\ldots 14$$

$$Q = \begin{bmatrix} 0 & p_{0,1}A_0 & \cdots & p_{0,m-2}A_0 & p_{0,m-1}A_0 \\ p_{1,0}A_1 & 0 & \cdots & p_{1,m-2}A_1 & p_{1,m-1}A_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ p_{m-2,0}A_{m-2} & p_{m-2,1}A_{m-2} & \cdots & 0 & p_{m-2,m-1}A_{m-2} \\ p_{m-1,0}A_{m-1} & p_{m-1,1}A_{m-1} & \cdots & p_{m-1,m-2}A_{m-1} & 0 \end{bmatrix} \quad \text{(E 2-3)} \ldots\ldots 14$$

$$a_{k,h}^{i} = \begin{cases} 0 & k < h \\ \rho_i^{k-h}(1-\rho_i) & k \geq h \geq 1 \ \ para \ \ C > B_i \\ \rho_i^{k} & k \geq h = 0 \end{cases} \quad \textbf{(E 2-4)} \ldots\ldots 15$$

$$a_{k,h}^{i} = \begin{cases} 0 & h < k \\ \rho_i^{h-k}(1-\rho_i) & k \leq h \leq K \ \ para \ \ C < B_i \\ \rho_i^{K-k} & k \leq h = K \end{cases} \quad \textbf{(E 2-5)} \ldots\ldots 15$$

$$a_{k,h}^{i} = \begin{cases} 1 - e^{-\gamma_i / C} & h = k = 0 \\ e^{-\gamma_i / C} & k = 0, h = 1 \\ 1 & k = h > 0 \\ 0 & \text{en otro caso} \end{cases} \quad \text{para } C = B_i \quad \textbf{(E 2-6)} \quad \text{..... 15}$$

$$\pi_i = \sum_{j \neq i} p_{j,i} \pi_j A_j \qquad i = 0,1,\ldots m-1$$

$$\sum_{i=0}^{m-1} \pi_i e^T = 1 \qquad \textit{(E 2-7)} \quad \text{..... 16}$$

$$E\left[N_{i,j}\right] = \frac{e^{-\gamma_i / B_i}}{1 - e^{-\gamma_i / B_i}} \quad \text{para } B_i > 0 \qquad \textit{(E 2-8)} \quad \text{..... 16}$$

$$P\left\{R_{i,j} = k\right\} = \rho_i^{K-j+k}(1 - \rho_i) \qquad \textbf{(E 2-9)} \quad \text{..... 17}$$

$$E\left[R_{i,j}\right] = \frac{\rho_i^{K-j+1}}{1 - \rho_i} \quad \text{para } B_i > C \text{ con } \rho_i = e^{-\varphi_i / (B_i - C)} \quad \textit{(E 2-10)} \quad \text{..... 17}$$

$$p = \frac{\sum_{i \in V} \sum_{j=0}^{K} \frac{\rho_i^{K-j+1}}{1 - \rho_i} \pi_{i,j}}{\sum_{i=0}^{m-1} \sum_{j=0}^{K} B_i / \gamma_i \, \pi_{i,j}} \qquad \textit{(E 2-11)} \quad \text{..... 17}$$

$$\pi_i = \alpha_{i-1} \pi_{i-1} A_{i-1} + \beta_{i+1} \pi_{i+1} A_{i+1} \qquad i = 0,1,\ldots M \qquad \textit{(E 3-1)} \quad \text{..... 20}$$

$$p_i = \frac{\alpha_{i-1} \alpha_{i-2} \ldots \alpha_0}{\beta_i \beta_{i-1} \ldots \beta_1} p_0 \qquad i = 1,2,\ldots M \qquad \textbf{(E 3-2)} \quad \text{..... 20}$$

$$p = \frac{\sum_{i=v+1}^{M} \frac{\rho_i y_{i,K}}{1 - \rho_i}}{\sum_{i=1}^{M} p_i \frac{e^{-\gamma_i / B_i}}{1 - e^{-\gamma_i / B_i}}} \qquad \textbf{(E 3-3)} \quad \text{..... 21}$$

$$y_i = \left(\alpha_{i-1} y_{i-1} + \beta_{i+1} y_{i+1}\right) A_i \qquad i = 0,1,\ldots M \qquad \textbf{(E 3-4)} \quad \text{..... 21}$$

$$q_i = \frac{p_i / \gamma_i}{\sum_{j=0}^{M} p_j / \gamma_j} \qquad i=0,1...M \qquad \textit{(E 5-1)} \quad \text{..... 38}$$

$$p_{vaciado} = \sum_{k=0}^{v} q_k \qquad v = \max\{i : i\Delta \le C\}$$
$$p_{llenado} = 1 - p_{vaciado}$$

(E 5-2) *38*

$$B_0 = \sum_{k=1}^{v} \frac{k\Delta q_k}{p_{vaciado}}$$
$$B_1 = \sum_{k=v+1}^{M} \frac{k\Delta q_k}{(1 - p_{vaciado})}$$

(E 5-3) *38*

$$E(W_i) = \frac{1 + i\mu E(W_{i-1})}{(M - i)\lambda} \qquad i = 1,2,\ldots M-1$$
$$E(W_0) = 1/M\lambda$$

(E 5-4)..... *39*

$$p = \begin{cases} \dfrac{1}{1-\rho_1} \dfrac{\rho_0 - \rho_1}{(\rho_0/\rho_1)^{K+1}((1-\rho_1)-(1-\rho_0))} \dfrac{(1-\eta_0)(1-\eta_1)}{\eta_0 - 2\eta_0\eta_1 + \eta_1} & \rho_0 \neq \rho_1 \\ \dfrac{1}{1-\rho_1} \dfrac{\rho_1}{K(1-\rho_1)+1} \dfrac{(1-\eta_0)(1-\eta_1)}{\eta_0 - 2\eta_0\eta_1 + \eta_1} & \rho_0 = \rho_1 \end{cases}$$

(E 5-5) *41*

12 ÍNDICE DE ILUSTRACIONES

Ilustración 2-1: Diagrama de estados de una fuente ON-OFF..... 10

Ilustración 2-2: Ejemplo gráfico del tráfico generado por una fuente ON-OFF con la parametrización escogida. 11

Ilustración 2-3: Comparativa entre el tráfico generado por una superposición de fuentes ON-OFF y su modelo MMDP. 12

Ilustración 2-4: Sistema de cola MMDP/1/D/K..... 13

Ilustración 3-1: Diagrama de estados del algoritmo iterativo de cálculo de la probabilidad de pérdida de célula con la aproximación M+1 MMDP 22

Ilustración 4-1: Diagrama de flujo de la implementación M+1 MMDP (1 de 3) 24

Ilustración 4-2: Diagrama de flujo de la implementación M+1 MMDP (2 de 3) 25

Ilustración 4-3: Diagrama de flujo de la implementación M+1 MMDP (3 de 3) 26

Ilustración 4-4: Código, comparación de dos resultados consecutivos (uso de épsilon). ... 34

Ilustración 4-5: Código, variable de control. 34

Ilustración 4-6: Código, ejemplo de fichero de entrada. 35

Ilustración 4-7: Código, ejemplo de fichero de salida. 36

Ilustración 6-1: Diagrama de flujo de la implementación 2 MMDP (1 de 2)..... 45

Ilustración 6-2: Diagrama de flujo de la implementación 2 MMDP (2 de 2)..... 46

Ilustración 6-3: Código, variable de control. 51

Ilustración 6-4: Código, ejemplo de fichero de entrada. 53

lustración 6-5: Código, ejemplo de fichero de salida. 53

Ilustración 7-1: Gráfica de variación de la probabilidad de pérdida de célula (p). Sistema grande. Tráfico de voz. Variación en M. 57

Ilustración 7-2: Gráfica de variación de la probabilidad de pérdida de célula (p). Sistema grande. Tráfico de datos. Variación en M. 59

Ilustración 7-3: Gráfica de variación de la probabilidad de pérdida de célula (p). Sistema grande. Tráfico de imágenes. Variación en M. 60

Ilustración 7-4: Gráfica de variación de la probabilidad de pérdida de célula (p). Sistema grande. Tráfico de voz. Variación en K. 62

Ilustración 7-5: Gráfica de variación de la probabilidad de pérdida de célula (p). Sistema grande. Tráfico de datos. Variación en K. 64

Ilustración 7-6: Gráfica de variación de la probabilidad de pérdida de célula (p). Sistema grande. Tráfico de imágenes. Variación en K. 66

Ilustración 7-7: Gráfica de variación de la probabilidad de pérdida de célula (p). Sistema pequeño. Tráfico de voz. Variación en M. 67

Ilustración 7-8: Gráfica de variación de la probabilidad de pérdida de célula (p). Sistema pequeño. Tráfico de datos. Variación en M. 69

Ilustración 7-9: Gráfica de variación de la probabilidad de pérdida de célula (p). Sistema pequeño. Tráfico de imágenes. Variación en M. 71

Ilustración 7-10: Gráfica de variación de la probabilidad de pérdida de célula (p). Sistema grande. Tráfico de voz. Variación en K. 72

Ilustración 7-11: Gráfica de variación de la probabilidad de pérdida de célula (p). Sistema pequeño. Tráfico de datos. Variación en K. 74

Ilustración 7-12: Gráfica de variación de la probabilidad de pérdida de célula (p). Sistema pequeño. Tráfico de imágenes. Variación en K. 76

Ilustración 8-1: Generalización de la aproximación 2 MMDP 82

Ilustración 8-2: Ejemplo de generalización del algoritmo M+1 MMDP 83

13 ÍNDICE DE TABLAS

Tabla 4-1: Código, ejecución del programa. *23*

Tabla 4-2: Parámetros iniciales. *31*

Tabla 4-3: Variables de cálculo. *33*

Tabla 6-1: Código, ejecución del programa. *43*

Tabla 6-2: Parámetros iniciales. *50*

Tabla 6-3: Variables de cálculo. *51*

Tabla 7-1: Características del tráfico generado por las fuentes ON-OFF *55*

Tabla 7-2: Características generales del cálculo. (Sistema grande. Tráfico de voz. Variación en M). *56*

Tabla 7-3: Datos (Sistema Grande. Tráfico Voz. Variación en M) *57*

Tabla 7-4:Características generales del cálculo. (Sistema grande. Tráfico de datos. Variación en M). *57*

Tabla 7-5: Datos (Sistema Grande. Tráfico Datos. Variación en M) *58*

Tabla 7-6:Características generales del cálculo. (Sistema grande. Tráfico de imágenes. Variación en M). *59*

Tabla 7-7: Datos (Sistema Grande. Tráfico Imágenes. Variación en M) *60*

Tabla 7-8: Características generales del cálculo. (Sistema grande. Tráfico de voz. Variación en K). *61*

Tabla 7-9: Datos (Sistema Grande. Tráfico Voz. Variación en K) *61*

Tabla 7-10:Características generales del cálculo. (Sistema grande. Tráfico de datos. Variación en K). *62*

Tabla 7-11: Datos (Sistema Grande. Tráfico Datos. Variación en K) *63*

Tabla 7-12:Características generales del cálculo. (Sistema grande. Tráfico de imágenes. Variación en K). *64*

Tabla 7-13: Datos (Sistema Grande. Tráfico Imágenes. Variación en K) *65*

Tabla 7-14: Características generales del cálculo. (Sistema pequeño. Tráfico de voz. Variación en M). *66*

Tabla 7-15: Datos (Sistema pequeño. Tráfico Voz. Variación en M) *67*

Tabla 7-16:Características generales del cálculo. (Sistema pequeño. Tráfico de datos. Variación en M). *68*

Tabla 7-17: Datos (Sistema Pequeño. Tráfico Datos. Variación en M) *68*

Tabla 7-18:Características generales del cálculo. (Sistema pequeño. Tráfico de imágenes. Variación en M). *69*

Tabla 7-19: Datos (Sistema Pequeño. Tráfico Imágenes. Variación en M) *70*

Tabla 7-20: Características generales del cálculo. (Sistema pequeño. Tráfico de voz. Variación en K). *71*

Tabla 7-21: Datos (Sistema pequeño. Tráfico Voz. Variación en K) *72*

Tabla 7-22:Características generales del cálculo. (Sistema pequeño. Tráfico de datos. Variación en K). *72*

Tabla 7-23: Datos (Sistema pequeño. Tráfico Datos. Variación en K) *73*

Tabla 7-24:Características generales del cálculo. (Sistema pequeño. Tráfico de imágenes. Variación en K). *74*

Tabla 7-25: Datos (Sistema pequeño. Tráfico Imágenes. Variación en K) *75*

14 REFERENCIAS BIBLIOGRÁFICAS

[1]	P.J. Schweitzer, *"A survey of aggregation-disaggregation in large Markov chains*", in *Numerical Solution of Markov Chains*, W. J. Stewart, Ed New York: Marcel Dekker, 1991.
[2]	S. M. Ross,*Introduction to Probability Models (4th ed.) New York: Academic, 1989*
[3]	P. Castelli, E. Cavallero, A. Tonetti, *"Policing and call call admission problems in ATM networks"*, in Proc. ITC-13, 1991
[4]	Tao Yang, Danny H. K. Tsang, *"A novel approach to estimating the cell loss probability in an ATM Multiplexer loaded with homogeneous ON-OFF sources"*, IEEE 1996