LABORATORIO DE SIMULACIÓN DE REDES Y TELETRÁFICO

ENUNCIADOS DE LAS PRÁCTICAS
Y
DOCUMENTACIÓN COMPLEMENTARIA
Curso 2013/2014

Autor: J. Vozmediano

Índice

1. Introducción

El Laboratorio de Simulación y Teletráfico tiene como misión introducir al alumno en el uso de herramientas de simulación para el diseño de protocolos y la evaluación de prestaciones de las redes de telecomunicación.

El curso proporcionará unas pocas clases teóricas necesarias para llevar a cabo las sesiones de laboratorio.

Para las sesiones de laboratorio, que se impartirán en el Centro de Cálculo, se propone el desarrollo de una serie de prácticas. Dichas prácticas son individuales, y deberán funcionar en los ordenadores del Centro de Cálculo.

2. Herramientas

Para llevar a cabo las prácticas son necesarias las siguientes herramientas:

1. Simulador de red con intérprete de OTcl `ns` y animador `nam`.
2. Herramienta de representación gráfica `gnuplot`.
3. Compilador de C++ y depurador simbólico `g++/gdb`.
4. Herramientas auxiliares (`awk`, `bash`).

3. Normas de la asignatura

Las Normas Generales del Laboratorio son las siguientes:

Sobre el desarrollo de las Prácticas

1. Las prácticas son individuales.
2. Las prácticas son optativas con puntos asociados, de modo que los alumnos puedan ir sumando puntos según el trabajo que hagan. Las prácticas están diseñadas para que con un esfuerzo razonable el alumno pueda aprobar cómodamente. En cada práctica se plantean una serie de cuestiones que el alumno debe intentar responder.

3. La asistencia al laboratorio se considera obligatoria. La falta de asistencia no justificada imposibilitará la superación de la asignatura.

4. Para facilitar el trabajo a los alumnos, se proporcionará un directorio accesible desde `http://trajano.us.es/docencia/LaboratorioDeSimulacionDeRedesYTeletrafico/`

   En los subdirectorios `pX`, siendo `X` el número de la práctica, se encontrarán las plantillas de los ficheros descritos en los enunciados, así como las herramientas necesarias para la realización de las prácticas.

Sobre la evaluación y calificación

1. Las prácticas se entregarán dejando los programas y resultados en una cuenta al efecto asignada a cada alumno, **obligatoriamente** en los subdirectorios `pX`, siendo `X` el número de la práctica. Los nombres de los ficheros entregados y el contenido de los directorios deberán ajustarse **obligatoriamente** a lo exigido en cada práctica. Los directorios correspondientes a las prácticas que no se deseen entregar deberán dejarse vacios.

2. **En todas aquellas cuestiones individuales planteadas en las prácticas que requieran la ejecución de más de un experimento:**

   - el primero de ellos usará la **semilla = 2014**
   - los siguientes usarán una semilla que termine con su número de laboratorio, que se asignará al comenzar la asignatura.

   IMPORTANTE: Por favor, ajústense a estas sencillas especificaciones. El no hacerlo conlleva el no superar el laboratorio, aunque algunas prácticas funcionasen.

3. La fecha de entrega coincidirá con el último día lectivo de Laboratorio del grupo a que pertenezca el alumno interesado.

4. El mismo día de la recogida de prácticas se entregará una memoria impresa por cada práctica presentada, en la que se respondan a las cuestiones planteadas en los enunciados. Sólo se considerarán presentadas las prácticas que vayan acompañadas de su respectiva memoria.

5. La calificación de cada práctica se obtendrá tras el examen de la memoria y de los ficheros entregados. En la evaluación de cada práctica se tendrán en cuenta los siguientes factores:

   - **que no sea copia de otra (se advierte que se realizarán comprobaciones automáticas de copias).**

- **que los resultados se presentan de forma gráfica, comprobando que los ejes de abcisas y ordenadas mantienen la misma escala y rango de variación.**
- **que los programas y makefiles tienen un estilo correcto y están bien documentado.**
- **que la memoria responde a las cuestiones planteadas de forma correcta y razonada.**

PRÁCTICA 11:
Familiarización con tcl ..... *(0 puntos)*

Familiarícese con Tcl. Para ello realice un programa sencillo denominado `factorial.tcl` que calcule el factorial de un número. Puede utilizar esta plantilla. Realice una versión recursiva y otra no recursiva.

Listado: Factorial de un número

```
 1: proc Factorial {x} {
 2:   set result 1
 3:
 4:
 5:   .... RELLENAR ....
 6:
 7:
 8:   return $result
 9: }
10:
11: set result [Factorial [lindex $argv 0]]
12: puts "$result"
```

El resultado de la ejecución debe ser:

```
$ ns factorial.tcl 10
3628800
```

PRÁCTICA 12:
Familiarizacion con otcl ..... *(1 punto)*

Familiarícese con OTcl. Realice un programa denominado `multi.tcl` que multiplique y divida números reales, enteros y complejos. Para ello rellene esta plantilla.

Listado: Multiplicación y división

```
 1: Class Real
 2:
 3: Real instproc init {x} {
 4:   $self instvar valor_
 5:   set valor_ $x
 6: }
 7:
 8: Real instproc multiplica {x} {
 9:   $self instvar valor_
10:   return [expr $valor_ * [$x set valor_]]
11: }
12:
13: Real instproc divide {x} {
14:   $self instvar valor_
15:   return [expr $valor_ / [$x set valor_]]
16: }
17:
18: Class Entero -superclass Real
19:
20: Entero instproc divide {x} {
21:   $self instvar valor_
22:
23:     .... RELLENAR ....
24: }
25:
26: set realA [new Real 12.3]
27: set realB [new Real 0.5]
28:
29: set realC [new Real [ $realA multiplica $realB ]]
30: puts [ $realC set valor_ ]
31: set realD [new Real [  $realA divide $realB ]]
32: puts [ $realD set valor_ ]
33:
34: set enteroA [new Entero 12]
35: set enteroB [new Entero 5]
36:
37: set enteroC [ new Entero [$enteroA multiplica $enteroB]]
38: puts [$enteroC set valor_]
39: set enteroD [ new Entero [$enteroA divide $enteroB]]
40: puts [$enteroD set valor_]
41:
42: Class Complejo -superclass Real
43:
44: Complejo instproc init {x y} {
45:   $self instvar real_
46:   $self instvar imaginaria_
47:   set real_ $x
```

```
48:   set imaginaria_ $y
49: }
50:
51: Complejo instproc multiplica {x} {
52:
53:   .... RELLENAR ....
54: }
55:
56:
57: set complejoA [new Complejo 0 3]
58: set complejoB [new Complejo 3 3]
59: set complejoC [$complejoA multiplica $complejoB]
60:
61: puts "([$complejoA set real_], [$complejoA set imaginaria_] i)"
62: puts "([$complejoB set real_], [$complejoB set imaginaria_] i)"
63: puts "([$complejoC set real_], [$complejoC set imaginaria_] i)"
```

El resultado de la ejecución debe ser:

```
$ ns multi.tcl
6.1500000000000004
24.600000000000001
60
2
(0, 3 i)
(3, 3 i)
(-9, 9 i)
```

Se pide

Entregue un fichero de nombre `multi.tcl`.

PRÁCTICA 13:
ns elemental ..... *(1 puntos)*

Defina una red muy simple sobre la que realizar algunos experimentos. Esta red servirá también para familiarizarse con el `ns`.

La red constará de sólo dos nodos separados por un enlace. Dicho enlace presenta las siguientes características:

Régimen Binario	36	Kb/s
Distancia	5.000	Km
c	0.67	c_0

El tráfico procede de una única fuente poissoniana conectada a uno de los nodos. Los paquetes en cuestión tienen una longitud constante e igual a 2000 octetos. Para enviarlos al destino ubicado en el otro nodo se encapsulan en datagramas UDP lo suficientemente grandes como para que no haya fragmentación.

`ns` no proporciona fuentes de Poisson, pero sí *on/off* exponenciales. En ellas hay periodos alternos de actividad y silencio, cuya duración sigue una distribución exponencial negativa. En los periodos activos se transmiten paquetes de un cierto tamaño a la tasa que se indique.

```
set e [new Application/Traffic/Exponential]
$e set packetSize_ W
$e set burst_time_ Xms
$e set idle_time_  Yms
$e set rate_       Zk
```

En [2] se indica que para que dichas fuentes se comporten como poissonianas, basta hacer la tasa muy elevada y la duración media de ráfaga nula. El tercer parámetro, `idle_time_`, será igual al tiempo medio entre llegadas.

```
set e [new Application/Traffic/Exponential]
$e set packetSize_ 210
$e set burst_time_ 0
```

```
$e set rate_ 999999999.9k
$e set idle_time_ 500ms
```

IMPORTANTE: *Se ha comprobado experimentalmente que las fuentes de tráfico on/off modelan adecuadamente el comportamiento de fuentes de voz. Sin embargo, también se ha comprobado que las fuentes poissonianas* NO MODELAN CORRECTAMENTE *el tráfico generado por fuentes de datos. En estas prácticas, por simplicidad y para comparar con los resultados analíticos, se utilizará a pesar de todo dicho modelo. Pero* NO LO USE *si necesita resultados fiables. En estos casos es preferible usar fuentes de Pareto superpuestas.*

Para modelar el hecho de que las colas de salida de los nodos tienen capacidad finita, limitaremos a Q el tamaño de las mismas. Los datagramas que sobrepasen dicho valor se perderán.

```
Q = 20
```

Se realizarán simulaciones de 60s de tiempo simulado.

1. Escriba el fichero `link.tcl` que define el escenario descrito. Para ello puede usar esta plantilla:

Listado: Plantilla mínima para ns

```
 1: #Inicio de link.tcl
 2: # ns link.tcl SEED RBIN SIMTIME
 3: puts "ns link.tcl SEED RBIN SIMTIME"
 4: puts "ns link.tcl $argv"
 5: #Set seed
 6: ns-random [lindex $argv 0]
 7:
 8: # Create a new simulator object.
 9: set ns [new Simulator]
10:
11: # Create a nam trace datafile.
12: set namfile      [open "[lindex $argv 0].nam" w]
13: $ns namtrace-all       $namfile
14: $ns set-animation-rate  0.8ms
15:
16: # Create nodes.
```

```
17: set node(1) [$ns node]
18: set node(2) [$ns node]
19:
20: # Create links between nodes.
21: # Light speed Km/s
22: set c    0.67
23: set c0   270000
24: #Distance Km
25: set d    5000
26: set rbin [lindex $argv 1]
27: set retr [ expr $d / ( $c0 * $c ) ]
28: set q    20
29:
30: $ns duplex-link $node(1) $node(2) $rbin $retr DropTail
31: # Set Queue Properties for link 1<->2
32: [[$ns link $node(1) $node(2)] queue] set limit_ $q
33: #Monitor the queue for the link
34: $ns duplex-link-op $node(1) $node(2) queuePos 0.5
35:
36: # Create agents, with given maximum packet size
37: # and flow indicator
38: set agent(1) [new Agent/UDP]
39: $ns attach-agent $node(1) $agent(1)
40: $agent(1) set fid_ 1
41: $agent(1) set packetSize_ 99999
42:
43: # Create traffic sources and add them to the agent.
44: set traffic_source(1) [new Application/Traffic/Exponential]
45: $traffic_source(1) attach-agent $agent(1)
46: $traffic_source(1) set burst_time_ 0ms
47: $traffic_source(1) set idle_time_  0.5
48: $traffic_source(1) set rate_       999999999.9k
49: $traffic_source(1) set packetSize_ 2000
50:
51: set agent(2) [new Agent/Null]
52: $ns attach-agent $node(2) $agent(2)
53:
54: # Connect agents.
55: $ns connect $agent(1) $agent(2)
56:
57: # Run the simulation
58: proc finish {} {
59:   global ns namfile
60:   $ns flush-trace
61:   close $namfile
62:   exit 0
63: }
64:
65: $ns at 0.0 "$traffic_source(1) start"
66: $ns at [lindex $argv 2] "finish"
67: $ns run
68: #Fin de link.tcl
```

2. Realice una simulación de 60.0 s para un régimen binario de 36.0 Kb/s con una semilla igual a su número de laboratorio.

   Observe el resultado en el fichero de trazas `.nam` generado. Observe la animación con `nam` ¿Qué representan los eventos marcados con `+`, `-`, `h` y `r` (sea preciso y concreto)?

3. Tomando el fichero de trazas, extraiga la longitud media de la cola de salida del enlace y el retardo de envío (transmisión más propagación) medio sufrido por los datagramas. Para ello pueden hacer uso de sendos filtros `awk` que se les proporcionan:

```
awk  -f cola.awk    'FROM_NODE=0' 'TO_NODE=1' 'OUTFILE=cola' \
          'TRANSIT=0.0' <file.nam>
awk  -f retardo.awk 'FID=1'  'OUTFILE=retardo' 'TRANSIT=0.0' <file.nam>
```

(También puede monitorizar la cola desde la ejecución de `ns`*, con algo como*

```
set qmon [$ns monitor-queue $node(1) $node(2) stdout]

proc qhist {accum nval} {
  global ns qmon
  set time 0.01
  set len [$qmon set pkts_ ]
  set accum [expr $accum+$len]
  incr nval
  puts "[expr 1.0*$accum/$nval]"
  $ns at [expr [$ns now] + $time] "qhist $accum $nval"
}

$ns at 0 "qhist 0 0"
```

Esta solución se ejecuta durante la simulación, pero evita escribir en disco o pantalla información menos relevante. Escójala cuando no vaya a mostrar los resultados con `nam`*, o cuando haya refinado el experimento y sepa exactamente qué datos le interesan.)*

Inspeccione los ficheros del filtro ¿Qué significa la información de salida de cada uno de ellos? La representación gráfica puede realizarla con cualquier programa, pero le resultará cómodo `gnuplot`, ya que basta ejecutar:

```
gnuplot> plot 'retardo.avg', 'longcola.avg'
```

¿Observa algún transitorio? ¿De qué duración? Reserve los ficheros de trazas para uso futuro.

4. Suba a 30000 s el tiempo de simulación y repita el paso anterior. ¿Qué opina ahora del transitorio?

5. Vuelva a los valores iniciales, suba a 36 Mb/s el régimen binario realice una simulación de 60.0 s ¿Cómo evoluciona ahora la cola? Continúe con la misma semilla.

6. Suponga ahora un régimen binario de 3.6 Kb/s y, con una duración de 60 s de simulación, indique qué ocurre en los ficheros de trazas.

   IMPORTANTE: *Los ficheros de trazas pueden crecer sin control en situaciones de alta carga en la red – que son precisamente las situaciones problemáticas y que por tanto interesa conocer. Es conveniente seleccionar con cuidado los estadísticos a trazar* antes *de realizar las simulaciones, para ahorrar* espacio.

   IMPORTANTE: *En una simulación, gran parte del tiempo se ocupa en la recogida de estadísticas y en el guardado de trazas. Estas últimas son especialmente lentas, porque implican accesos a disco. De nuevo, es conveniente seleccionar con cuidado los estadísticos a trazar* antes *de realizar las simulaciones, para ahorrar* tiempo. *Por el mismo motivo, una vez depurado el modelo debe evitar todo tipo de trazas en pantalla o disco.*

7. Tome un un régimen de 36 Kb/s y una duración de los experimentos de 30000 s. Realice 10 experimentos modificando la semilla. ¿Cuál es el retraso medio sufrido por los paquetes en el estado estacionario? ¿Cuál es el tamaño del intervalo de confianza del 90 %? Responda a las mismas preguntas considerando un transitorio de 0 s y otro de 200 s. Para truncar el transitorio puede usar los filtros anteriores, indicando el periodo transitorio en una variable adicional:

```
awk  -f retardo.awk 'FID=1' 'OUTFILE=retardo'   'TRANSIT=200.0' <file.nam>
awk  -f cola.awk    'FROM_NODE=0' 'TO_NODE=1' 'OUTFILE=longcola' \
          'TRANSIT=200.0' <file.nam>
```

8. Compare el resultado obtenido con el teórico. Discuta los resultados.

Se pide

Entregue una memoria impresa de esta práctica en la que conteste razonadamente a los puntos anteriores. Explique paso a paso cómo ha llegado a la respuesta.

Entregue también un directorio con los programas (.cc, .h o .tcl) y makefiles necesarios, pero no incluya ficheros de trazas, objetos ni ejecutables.

PRÁCTICA 14:
Simuladores. Simulación de eventos ..... *(1 punto)*

Con objeto de que se familiarice con la construcción de modelos orientados a eventos, se ha desarrollado una pequeña biblioteca para la simulación orientada a eventos discretos. Esta biblioteca está disponible en la página del laboratorio, y se compone de:

- **distrib.h**: Generador de números aleatorios (algoritmo de Wichmann-Hill) integrado con algunas de las distribuciones más frecuentes.
- **lstat.h**: Recolector de estadísticos temporales y no temporales.
- **ev.h**: Evento genérico implementado como clase abstracta del que derivar tipos de evento concretos, planificador básico con funciones de encolado y ejecución de la simulación y la definición del único ejemplar de objeto simulador que debe existir (`Simulation`).

La construcción de modelos de simulación puede simplificarse si se aprovechan las técnicas de la orientación a objetos. Con idea de enriquecer los conocimientos del alumno, se ha escogido C++ [5] como lenguaje para algunas de las prácticas.

La utilización de las funciones proporcionada por la biblioteca es muy sencilla, como puede comprobarse de los ficheros de definición. La dificultad suele estar en la construcción del modelo a simular. A modo de ejemplo se facilita un programa que modela un sistema de transmisión como una cola M/M/1. Dicho sistema se compone de una fuente poissoniana que emite tramas de longitud variable y de un servidor que resulta ocupado durante el tiempo de transmisión de la trama. Cuando el servidor está ocupado, las tramas que llegan esperan su turno en una cola de longitud infinita.

Listado: Modelo de cola M/M/1

```
 1: //Inicio de mm1.cc
 2: #include <stdio.h>
 3: #include <stdlib.h>
 4:
 5: //*******************************************************
 6: // Includes for distributions, statistics and
 7: // event simulation
 8:
 9: #include "distrib.h"
10: #include "lstat.h"
11: #include "ev.h"
12:
```

```
13: //*****************************************************
14: // Declare Events
15: class Arrival;    //Models the arrival of a frame to the server
16: class Departure;  //Models the departure of a frame from the server
17:
18: //*****************************************************
19: //State-related objects
20:
21: class Source {
22:   // Source-related state variables and statistics:
23: private:
24:   Exponential* tlleg_;      //Burst Inter-arrival time
25:
26:   //Statistics
27:   LStat        S_interArrival;
28:
29: public:
30:   Source(timeunits tll):
31:     tlleg_(new Exponential (tll)),
32:     S_interArrival("Inter Arrival Time")
33:   {
34:   };
35:
36:   ~Source()
37:   {
38:     delete tlleg_;
39:   }
40:
41:   timeunits NextArrival(timeunits now);
42: };
43:
44: //Gets the time of next arrival
45: timeunits Source::NextArrival (timeunits now)
46: {
47:
48:   timeunits retval = now + tlleg_->GetVal();
49:
50:   //Accumulate this interarrival time
51:   S_interArrival.accum((double) (retval-now));
52:   return retval;
53: }
54:
55: //*****************************************************
56: class Server {
57: private:
58:   bool          isEmpty_;      //Server status
59:   timeunits     departure_;    //Next departure time
60:   Exponential*  servtime_;     //Service time distribution
61:   int           queuedFrames_; //Current input queue size
62:
63: public:
64:
65:   //Statistics
66:   LStat         S_serviceTime; //Service Time
67:   LStat         S_obsNumFrames;//Observed num of frames in
68:                                //the server (from arriving frames)
69:   TStat         T_numFrames;   //Expected num of frames in the server
70:   TStat         T_load;        //Server load
71:
72:   Server(timeunits meanserv, int qsize=0):
73:     isEmpty_(true),
74:     departure_(0),
```

```
 75:     servtime_(new Exponential(meanserv)),
 76:     queuedFrames_(0),
 77:     S_serviceTime ("Service Time"),
 78:     S_obsNumFrames("Obsvd. Num. Frames"),
 79:     T_numFrames   ("Avg.   Num. Frames", &Simulation),
 80:     T_load        ("Server Load (<100%!)", &Simulation) {};
 81:
 82:   ~Server(){
 83:     delete servtime_;
 84:   };
 85:
 86:   //Data manipulation functions
 87:   inline bool IsEmpty() const {return isEmpty_;}
 88:   inline int  GetQLen() const {return queuedFrames_;}
 89:
 90:   //Increment and decrement queue length, and
 91:   //gather proper statistics
 92:   inline void incr (timeunits now) {
 93:
 94:     if (isEmpty_) {
 95:       T_load.accum(1);
 96:     }
 97:     T_numFrames.accum(++queuedFrames_);
 98:     isEmpty_=false;
 99:   }
100:
101:   inline void decr (timeunits now) {
102:
103:     T_numFrames.accum(--queuedFrames_);
104:     isEmpty_ = (0 == queuedFrames_);
105:     if (isEmpty_) {
106:       T_load.accum(0);
107:     }
108:   }
109:
110:   timeunits GetDeparture(timeunits now);
111: };
112:
113:
114: timeunits Server::GetDeparture(timeunits now) {
115:
116:   timeunits retval = now + servtime_->GetVal();
117:
118:   S_serviceTime.accum((double) (retval - now));
119:   return retval;
120: }
121:
122: //*******************************************************
123: class Departure : public Event {
124: private:
125:   Server  *from_;
126: public:
127:   Departure(timeunits t, Server* from):
128:     Event(t), from_(from){  };
129:   ~Departure(){  };
130:   void Handle();
131: };
132:
133:
134: void Departure::Handle (){
135:
136:   from_->decr(Time());
```

```
137:   if (!from_->IsEmpty()){
138:     timeunits t = from_->GetDeparture(Time());
139:     Departure* d = new Departure (t, from_);
140:     Simulation.Schedule(d);
141:   }
142:   else {
143:     //Idle server
144:     ;
145:   }
146: }
147: //*******************************************************
148: class Arrival : public Event {
149: private:
150:   Source*  from_;
151:   Server*  to_;
152:
153: public:
154:   Arrival(timeunits t, Source* from, Server* to):
155:     Event(t), from_(from),
156:     to_(to) {};
157:   ~Arrival(){};
158:   void Handle();
159: };
160:
161: void Arrival::Handle (){
162:   to_->S_obsNumFrames.accum(to_->GetQLen());
163:
164:   if (to_->IsEmpty()){
165:     to_->incr(Time());
166:     timeunits t = to_->GetDeparture(Time());
167:     Departure* d = new Departure (t, to_);
168:     Simulation.Schedule(d);
169:   }
170:   else {
171:     to_->incr(Time());
172:   }
173:   //Enqueue next arrival
174:   timeunits t = from_->NextArrival(Time());
175:   Arrival* r = new Arrival(t, from_, to_);
176:   Simulation.Schedule(r);
177: }
178:
179:
180: //*************************************************
181: //Main program
182: int  main (int argc, char* argv[])
183: {
184:
185:
186:   if (5 != argc){
187:     printf(\
188:       "Usage: %s <seed> <arrival_rate> <mean_serv_time> <time>\n",\
189:            argv[0]);
190:     return -1;
191:   }
192:   for (int k=0; k<argc; k++)
193:     printf("%s ",argv[k]);
194:   printf("\n");
195:
196:   EvSim::AssertVersion(1.20);
197:   Distrib::InitRng(atoi(argv[1]));
198:
```

```
199:   timeunits tlleg    = (timeunits) (1/atof(argv[2]));
200:   timeunits meanserv = (timeunits) atof(argv[3]);
201:
202:   Source S1(tlleg);
203:   Server C1(meanserv);
204:
205:   Arrival* r = new Arrival(0, &S1, &C1);
206:   Simulation.Schedule(r);
207:
208:   Simulation.Run(0, (timeunits)atof(argv[4]));
209: }
210: //Fin de mm1.cc
```

Al inicio del fichero se encuentran los `#include` necesarios.

Se han diferenciado dos tipos de evento[1]:

- Arrival: Modela una llegada de una trama a la cola del servidor del sistema M/M/1.
- Departure: Modela el fin de la transmisión de la trama y su salida del servidor.

Note que los eventos son instantáneos, esto es *no tienen un estado que pueda evolucionar con el tiempo*, y durante su tratamiento, en `Event::Handle()`, el tiempo simulado se congela. Así, los eventos:

- ocurren en un instante de tiempo,
- se procesan,
- pueden crear otros eventos nuevos y encolarlos para su ocurrencia en un instante futuro,
- y a continuación se destruyen (en `Simulation.Run`).

Identifique dichas fases en las funciones miembro adecuadas. Toda la información de estado se encuentra en los objetos `Source` y `Server`, a los que la rutina de tratamiento del evento puede acceder mediante los punteros adecuados. Los estadísticos se guardan también en ellos, si bien su actualización se realiza en las funciones de tratamiento de los eventos.

Así, el evento de tipo `Arrival` llevará un puntero a `Server` y otro a `Source`, mientras que el evento de tipo `Departure` llevará sólo uno a `Server`.

[1] Este NO ES sel único modelado posible.

El objeto `Source` sólo contiene una variable miembro: la que apunta al generador de la distribución del tiempo entre llegadas (exponencial, por ser la fuente poissoniana). Esto es razonable, puesto que dichas fuentes son sin memoria, lo que implica sin estado[2]. Se incluye también un recolector de estadísticas para mostrar su uso.

El objeto `Server` contiene las variables de estado del servidor y su cola de entrada, así como cuantos contadores y recolectores de estadísticas se ha estimado conveniente. Note que las estadísticas dependientes del tiempo necesitan un puntero al objeto simulador para acceder al reloj de la simulación. El objeto se completa con las funciones de tratamiento de las variables miembro, de entre las que se destacan las que gestionan los cambios de estado cuando llega o parte una trama, es decir, cuando se incrementa o decrementa el número de usuarios en el sistema. Dichas funciones se encargan, además, de actualizar los estadísticos de interés.

En cuanto a las funciones de tratamiento de eventos, el evento `Departure` se limita a decrementar la cola del servidor y, si no queda vacía, encolar (con `Simulation.Schedule`) la proxima salida para cuando corresponda al tiempo de servicio que el servidor indique. Nótese que los eventos se crean con `new`, pero nunca se borran: ésa es la tarea de la función `EvSim::Run()`, puesto que va extrayendo los eventos de la cola de eventos, avanzando el reloj de simulación y destruyéndolos tras invocar a su función de tratamiento.

Por su lado, el evento `Arrival` se trata de la siguiente forma: si cuando se activa el servidor está vacío, se incrementa el número de tramas en el sistema y se encola la siguiente salida del servidor para cuando corresponda; y si el servidor no está vacío, simplemente se incrementa el número de tramas en el sistema. Antes de finalizar, se encola también un nuevo evento `Arrival` tanto tiempo después como indique la fuente.

Los eventos no llevan información asociada, aparte de los punteros a `Source` o `Server`. Si se necesitara información adicional (por ejemplo datos tales como la dirección IP del paquete que llega o parte) pueden declararse variables miembro nuevas y pasarse del evento `Arrival` al `Departure` correspondiente por medio de los constructores. Si el tipo de variable lo permite (y para evitar engordar innecesariamente el tipo de evento, no se olvide que implica gestión de memoria dinámica y por tanto lentitud) también podría recogerse dicha información en otro objeto, a una instancia del cual apuntasen todos los tipos de evento afectados.

El programa principal es muy corto, en parte gracias a la orientación a objetos y en parte gracias a que la impresión de estadísticos se efectúa automáticamente en los destructores. En efecto, y aparte de una sencilla comprobación de los parámetros de entrada, se limita a:

[2]Vea que esto plantea la posibilidad de modelos más simples, donde se elimine el objeto `Source` y sus funciones se incluyan en `Server`, pero de momento, por claridad, es preferible mantener un nivel de detalle suficiente en el modelado

- Mostrar la línea de invocación (imprescindible para facilitar el procesado automático por lotes),
- comprobar la versión de la biblioteca de simulación e iniciar el generador de números aleatorios con la semilla correspondiente,
- crear un objeto `Source` y otro `Server` con los parámetros apropiados,
- cebar el simulador encolando el primer evento de tipo `Arrival` en el instante $t = 0$ y
- ejecutar la simulación hasta que termina.

Se pide

Para validar el modelo, y puesto que se conoce su solución analítica, se propone:

1. Compile el programa. El fichero resultante, obtenido mediante el `makefile` correspondiente, debe llamarse `mm1.exe`:

   ```
   bash.exe $ make mm1.exe
   g++ -g -Wall -I$(DIR) -c  mm1.cc
   g++ -g -o mm1.exe  mm1.o  -L$(DIR) -laitsim
   ```

   donde `$(DIR)` indica la ruta donde se encuentran los ficheros con los prototipos y la biblioteca `libaitsim.o`.

2. Ejecute una batería de simulaciones de 500.000 unidades de tiempo de duración cada una, de forma que se realicen 20 ejecuciones para cada nivel de carga del servidor entre 0.05 y 0.95 (en intervalos de 0.15). Fije el tiempo de servicio a 1 unidad de tiempo y varíe el tiempo entre llegadas según convenga. Recuerde, en cuanto a semillas, las normas generales de la asignatura.

   Deposite el fichero que permita repetir la batería completa de simulaciones en `bateria`. Este fichero podrá invocarse así:

   ```
   bash.exe $ make bateria > resultado.txt
   ```

Nadie (en su sano juicio) escribirá las 140 invocaciones distintas en el fichero `makefile`. Es preferible automatizar las simulaciones usando un `shell-script` con varios bucles `for`.

Listado: Ejemplo de `shell-script`

```
 1: #Inicio de bateria
 2: modelo=mm1.exe
 3: for seed in 2005 *RELLENAR*
 4: do
 5:   loadstep=0.15
 6:   simtime=500000
 7:   arrivrate=0.05
 8:   while [ "$arrivrate" != "1.10" ]
 9:   do
10:     servtime=1
11:     $modelo $seed $arrivrate $servtime $simtime
12:     arrivrate=‘echo $loadstep $arrivrate | awk ’{printf("%02.2f\n",$1+$2)}’‘
13:   done
14: done
15: #Fin de bateria
```

No olvide redireccionar la salida a un fichero para su procesado posterior (pero no entregue ese fichero).

3. Con el fichero de trazas resultantes, determine las estimaciones de los tiempos medios entre llegadas y de servicio, y la estimación del número medio de tramas en el sistema, con sus correspondientes intervalos de confianza del 90 %. No olvide que al acercar la carga al 100 % la cola de entrada crecerá sin control (observe la varianza).

   Si lo desea, en lugar de hacerlo a mano, le resultará de mucha utilidad crear un programa en `awk` (o cualquier otro lenguaje interpretado) que procese el fichero de trazas y obtenga los datos en cuestión. El filtro `tstudent.awk` le resultará de utilidad para calcular los intervalos de confianza sin necesidad de buscar en tablas.

```
echo <num> <alfa> --intervalo <media> <varianza> \
                                              | awk -f tstudent.awk
```

4. ¿Se corresponden, en líneas generales, los valores estimados de los tiempos de servicio y entre llegadas con las distribuciones correspondientes? ¿Por qué?

5. Compare los valores citados con el resultado teórico para los valores que escogió en el punto inicial. ¿Están dentro del intervalo de confianza?

6. Por último, modifique el programa para que modele un sistema de cola finita M/M/1/k y repita los pasos anteriores con `k=10`. El fichero fuente debe llamarse `mm1k.cc`, y el

generador de los experimentos `bateria2.sh`. Entregue también el `makefile` correspondiente.

Entregue una memoria impresa de esta práctica en la que conteste razonadamente a los puntos anteriores. Explique paso a paso cómo ha llegado a la respuesta.

Entregue también un directorio con los programas (.cc, .h o .tcl) y makefiles necesarios, pero no incluya ficheros de trazas, objetos ni ejecutables.

PRÁCTICA 15:
Simuladores. Modelado de alto nivel ..... *(1 puntos)*

La biblioteca de simulación presentada en la práctica anterior contiene también un conjunto de elementos básicos predefinidos derivados de `NetElement`:

- **PacketSource**: Fuente generadora de paquetes. Acepta como parámetros dos distribuciones cualesquiera. La primera modela el tiempo entre generaciones sucesivas, en unidades de tiempo. La segunda, el tamaño de los paquetes generados, en bits.
- **Server**: Servidor de paquetes, con una capacidad de `rBin` bits por unidad de tiempo y con una cola de entrada de `qsize` bits (-1 implica posiciones ilimitadas).
- **PacketSink**: Sumidero de paquetes.

Para gobernar el funcionamiento combinado de estos elementos se ha optado por un modelado alternativo al de la práctica anterior, con un único tipo de evento, `Timer`, que se dirige a un `NetElement` concreto. Las definiciones de todos estos elementos se encuentran en `netelem.h`.

Comparado con el modelo anterior, su uso es de una simplicidad casi ofensiva:

Listado: Modelo de cola M/M/1 usando NetElements

```
 1: //Inicio de net1.cc
 2: #include <stdio.h>
 3: #include <stdlib.h>
 4: //*****************************************************
 5: // Includes for distributions, statistics and
 6: // event simulation
 7: #include "netelem.h"
 8:
 9: //*****************************************************
10: //Main program
11: int  main (int argc, char* argv[])
12: {
13:
14:   if (7 != argc){
15:     printf(\
16:    "Usage: %s <seed> <arrival> <psize> <capacity> <q> <time>\n",\
17:            argv[0]);
18:     printf("<arrival>  rate in packets per timeunit\n");
19:     printf("<psize>    in bits per packet\n");
20:     printf("<capacity> server capacity in bits per timeunit\n");
21:     printf("<q>        input queue, in bits. -1 if unlimited\n");
22:     return -1;
```

```
23:   }
24:   for (int k=0; k<argc; k++)
25:     printf("%s ",argv[k]);
26:   printf("\n");
27:
28:   EvSim::AssertVersion(1.20);
29:   Distrib::InitRng(atoi(argv[1]));
30:
31:   timeunits tarriv   = (timeunits) (1/atof(argv[2]));
32:   timeunits psiz     = (timeunits) atof(argv[3]);
33:
34:   PacketSource S1(new Exponential(tarriv), new Exponential(psiz));
35:
36:   Server       C1("C1", (int) atof(argv[4]), (int) atoi(argv[5]));
37:   PacketSink   K1;
38:
39:   S1.ConnectTo(&C1);
40:   C1.ConnectTo(&K1);
41:
42:   S1.Start(0);
43:
44:   Simulation.Run(0, (timeunits)atof(argv[6]));
45: }
46: //Fin de net1.cc
```

Se pide

1. Compile el programa `net1.cc`. Se le facilitan los ficheros de cabecera y la biblioteca compilada para UNIX (Linux) o *CygWin*, no olvide escoger la adecuada. El fichero resultante, obtenido mediante el `makefile` correspondiente, debe llamarse `net1.exe` independientemente del sistema operativo.

2. Ejecute una batería de simulaciones de 100.000 unidades de tiempo de duración cada una, de forma que se realicen 10 ejecuciones para un nivel de carga del servidor de 0.85. Escoja los tiempos de servicio y entre llegadas que desee (ojo a las unidades), pero recuerde, en cuanto a semillas, las normas generales de la asignatura.

   Modifique `bateria.sh` para que se adapte a este nuevo escenario. El fichero resultante se denominará también `bateria.sh`.

   ```
   bash.exe $ sh bateria.sh > resultado.txt
   ```

   No olvide redireccionar la salida a un fichero para su procesado posterior (pero no entregue ese fichero).

3. Con el fichero o ficheros de trazas resultantes, determine las estimaciones de los tiempos medios entre llegadas y de servicio, y la estimación del número medio de tramas en el sistema, con sus correspondientes intervalos de confianza del 90 %.

Si lo desea, en lugar de hacerlo a mano, le resultará de mucha utilidad crear un programa en `awk` (o cualquier otro lenguaje de script) que procese el fichero de trazas y obtenga los datos en cuestión. El filtro `tstudent.awk` le resultará de utilidad para calcular los intervalos de confianza sin necesidad de buscar en tablas.

```
echo <num> <alfa> --intervalo <media> <varianza> \
                                          | awk -f tstudent.awk
```

4. Por último, modifique el programa para que modele un sistema compuesto por tres conmutadores y tres fuentes de células.

   Todas las fuentes generan ráfagas separadas según una distribución exponencial de media 1 segundo. La longitud de las ráfagas sigue una distribución de tipo Pareto de media 3.000 células ATM y factor de forma $\alpha = 1.6$.

   La distribución de Pareto se caracteriza por la siguiente función de distribución complementaria:

$$P(x > X) = \begin{cases} \left(\frac{x}{\delta}\right)^{-\alpha} & \text{si } x \geq \delta \\ 1 & \text{en otro caso} \end{cases} \quad (1)$$

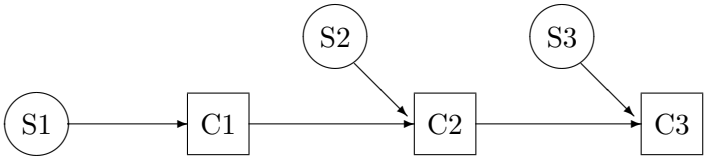
   con $1 < \alpha < 2$. La varianza de la distribución de Pareto es infinita. Necesitará calcular su media. A δ se la denomina también factor de escala, y a α, factor de forma.

   IMPORTANTE: *La multiplexión de un número elevado de fuentes de* on/off *en las que la duración del periodo* on *sigue una distribución de Pareto parece modelar adecuadamente el tráfico en redes de paquetes.*

   Cada una de las fuentes está conectada a un conmutador. Cada conmutador posee una cola de k=64 posiciones. Los commutadores están conectados en serie con enlaces ATM que soportan un régimen binario de 149.76 Mb/s y retardo de propagación despreciable.

   El fichero fuente debe llamarse `3md1k.cc`, y el generador de los experimentos `bat3md1k.sh`. Entregue también el `makefile`.

   Estime la probabilidad de pérdida de paquetes por desbordamiento en el tercer conmutador con una precisión tal que la anchura del intervalo de confianza sea inferior a 10^{-2} para un nivel de confianza del 95 %.



Entregue una memoria impresa de esta práctica en la que conteste razonadamente a los puntos anteriores. Explique paso a paso cómo ha llegado a la respuesta.

Entregue también un directorio con los programas (.cc, .h o .tcl) y makefiles necesarios, pero no incluya ficheros de trazas, objetos ni ejecutables.

PRÁCTICA 16:
Estudio comparativo con resultados analíticos .. *(2 puntos)*

En esta práctica se pretende que diseñe un experimento que pueda validar el cálculo teórico del rendimiento de una conexión TCP en una situación con probabilidad de pérdida de paquetes no nula.

Considere el estudio publicado en *Modeling TCP Throughput: A Simple Model and its Empirical Validation* (Jitendra Padhye, Victor Firoiu, Don Towsley, Jim Kurose), de los *Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication.* En él aparece una aproximación analítica del rendimiento:

$$S(P)[b/s] \approx min\left(\frac{W}{RTT}, \frac{L}{RTT * \sqrt{\frac{4P}{3}} + T_0 min\left(1, 3\sqrt{\frac{6P}{8}}\right) P(1+32P^2)}\right) \quad (2)$$

donde W es el tamaño máximo de la ventana de transmisión en bits, L es el tamaño del paquete TCP con las cabeceras IP incluidas, RTT es el valor medio del retardo de ida y vuelta y T_0 es el tiempo que se espera un asentimiento antes de proceder a la retransmisión y que, en la implementación de TCP del `ns`, vale 0.2s.

`ns` dispone de varias implementaciones de TCP. La expresión anterior es válida cuando el receptor TCP usa la estrategia de asentimientos retrasados, en la que, antes de asentir un paquete, espera un cierto tiempo. Si un segundo paquete llega, entonces se asienten ambos inmediatamente. En `ns` este comportamiento se consigue usando un sumidero del tipo `[Agent/TCPSink/DelAck]`.

Para modelar la probabilidad de pérdida, el manual del `ns` proporciona la descripción del elemento `ErrorModel`

```
# Elemento
set em [new ErrorModel]
$em unit pkt
# Probabilidad de pérdida de paquete
$em set rate_   P
$em ranvar      [new RandomVariable/Uniform]
$em drop-target [new Agent/Null]
# Adhesión al enlace afectado
```

```
$ns link-lossmodel $em $n(0) $n(1)
```

Se pide

Diseñe el experimento que permita validar la expresión citada. Dicho experimento deberá incluir:

1. La definición del modelo a simular, en el fichero `rendtcp.tcl`
2. El fichero que permita la ejecución de la batería de simulaciones `rendtcp.sh`
3. La comparación del resultado teórico con el simulado para los valores de P `0.4`, `0.3`, `0.2`, `0.1`, `0.05`, `0.01`, `0.005`, `0.001`, `0.00001` y `0.0000001` y `0`.

PRÁCTICA 17:
Simuladores. Modelado de bajo nivel ..... *(2 puntos)*

En esta práctica se le pide que construya un nuevo elemento para la biblioteca de simulación.

El elemento en cuestión modela un bus con mecanismo de acceso CSMA, derivado de `NetElement` y de nombre `CsmaBus`

Listado: Definición de CsmaBus

```
 1: //Cabecera
 2: #ifndef CSMA
 3: #define CSMA
 4:
 5: #include "netelem.h"
 6:
 7: class CsmaBus : public NetElement {
 8: public:
 9:   CsmaBus  (char* name,          //Identifies the bus
10: double rBin,          //Bus bitrate in bits/timeunit
11: timeunits delay = 0);  //Propagation delay
12:
13:   virtual ~CsmaBus();
14:
15:   void          Send (timeunits now, Packet *p);
16:   void          Recv (timeunits now, Packet *p);
17:   void          Reset (bool dumpStats = false);
18:
19: private:
20:   char*        name_;
21:   double       rBin_;  //In bits per timeunit
22:   timeunits    delay_; //In timeunits
23:
24:   bool         colFlag_;
25:   unsigned int inSystemPackets_;
26:
27:   //Counters
28:   unsigned long int numCollisions_;
29:
30:   //Statistics
31:   LStat         S_serviceTime;  //Service Time
32:   TStat         T_occupancy;    //Server occupancy
33:
34:   void  namConnectTrace ();
35: };
36: #endif
37: //Fin cabecera
```

Listado: Esqueleto de implementación

```
1: //Inicio esqueleto
2: void CsmaBus::Send (timeunits now, Packet *p)
3: {
4:   packetTrace(now,'r', p);
5:   /*
6:     ...
7:     ...
8:   */
9: }
10:
11: void CsmaBus::Recv  (timeunits now, Packet *p)
12: {
13:   packetTrace(now, '+', p);
14:   /*
15:     ...
16:     ...
17:   */
18: }
19:
20:
21: CsmaBus::~CsmaBus  ()
22: {
23:   /*
24:     ...
25: Dump statistics
26:     ...
27:   */
28: }
29:
30: CsmaBus::CsmaBus  (char* name, double rBin,    timeunits delay)
31:   :  name_(new char[1+sizeof(name)]),
32:      rBin_(rBin),
33:      delay_(delay),
34:      ...
35: {
36:   strcpy(name_,name);
37:
38:   if (IsTraceing()) {
39:     if (0!=traceFile())
40:       fprintf(traceFile(), "n -t * -a %i -s %i -S UP -v circle -c black -i black \n",
41:               GetId(), GetId());
42:   }
43: }
44:
45:
46: void  CsmaBus::namConnectTrace ()
47: {
48:   if (IsTraceing()) {
49:     unsigned int index = 0; // This parameter is used to trace up to N outputs
50:     while (index < GetSizeTo()){
51:       unsigned int me   = GetId();
52:       unsigned int you  = GetTo(index)->GetId();
53:       if (0 != traceFile()){
54:         fprintf(traceFile(), "l -t * -s %u -d %u -S UP -r %f -D %f -c black -o\n",
55:                 me,
56:                 you,
57:                 rBin_,
58:                 delay_);
59:         fprintf(traceFile(), "q -t * -s %u -d %u -a 0.5\n", me, you);
60:         fprintf(traceFile(), "q -t * -s %u -d %u -a 0.5\n", you, me);
```

```
61:       }
62:       ++index;
63:     }
64:   }
65: }
66: //Fin esqueleto
```

Se pide

1. Complete `csma.cc`. Use, pero no modifique, los fragmentos de código para generación de trazas. No son necesarios, pero pueden ayudarle a depurar su programa. Puede (necesitará) definir atributos y funciones miembro adicionales.

2. Realice un módulo simulador. Dicho módulo simulará un escenario en el que N fuentes Poissonianas acceden a un bus CDMA. Dichas fuentes generan tramas de longitud fija. Puede conectar el bus a un sumidero de tipo `SrcSink`, que se encargará de recoger datos de los paquetes ilesos. Su invocación por línea de comandos responderá al modelo:

   ```
   $ csmanet <1seed> <2time> <3bus.rbin.in.bps> <4bus.tprop.in.s> \
   <5nsources> <6lambda> <7framesize.in.bits>
   ```

3. Utilice el modelo desarrollado para hallar la curva de rendimiento de un bus a 10 Mb/s con un tiempo de propagación de 0.005 ms. A dicho bus acceden 500 fuentes, que generan tramas de longitud fija e igual a 256 octetos. ¿Coincide el tiempo normalizado en que el bus está ocupado con el rendimiento? Represente gráficamente la curva pedida con su correspondiente anchura del intervalo de confianza para un nivel de confianza del 90 %. Por cierto, si todavía es de los *aficionados* a las interfaces gráficas, pruebe a activar las trazas (invocando a `NetElement::traceOn()` al principio del programa) y disfrute arrancando el `nam`.

4. Repita el experimento para un tiempo de propagación de 1, 0.5 y 0.05 ms. Represente todas las curvas de rendimiento en la misma gráfica. Discuta los resultados.

Entregue una memoria impresa de esta práctica en la que conteste razonadamente a los puntos anteriores. Explique paso a paso cómo ha llegado a la respuesta.

Entregue también un directorio con los programas (.cc, .h o .tcl) y makefiles necesarios, pero no incluya ficheros de trazas, objetos ni ejecutables.

PRÁCTICA 18:
Simuladores. Modelado de bajo nivel ..... *(2 puntos)*

En esta práctica se le pide que estudie la probabilidad de pérdida de células en la cola de multiplexor ATM al que acceden un conjunto de fuentes ON-OFF. Este tipo de fuente VBR modela adecuadamente el tráfico de voz codificada con supresión de silencios.

Las fuentes de tráfico ON-OFF son un caso particular de las fuentes de dos estados. Cada estado se caracteriza por una duración o tiempo de permanencia (estocástico), durante el que se generan, con una separación que sigue una cierta distribución, tramas o paquetes de longitud constante.

En el caso de las fuentes ON-OFF que modelan tráfico de voz el tiempo de permanencia en ambos estados tiene una duración exponencial (geométrica en tiempo discreto), pero sólo en el estado activo se transmite tráfico (ráfaga de voz). Los paquetes de longitud constante se corresponden con las células ATM. Todas las células de una ráfaga están separadas un tiempo constante, e igual a la inversa de la tasa de pico.

Para la construcción de modelos puede utilizar el siguiente elemento de red:

Listado: Definición de BiStateSource

```
 1: class SourceState {
 2: public:
 3:   // The distribution objects will be deleted by the destructor
 4:   SourceState(Distrib* duration, Distrib* interarrival, int samplesize);
 5:   virtual ~SourceState();
 6:   int GetSampleSize();
 7:   timeunits GetDuration();
 8:   virtual timeunits GetInterarrival();
 9: };
10:
11: class BiStateSource : public PacketSource {
12: public:
13:   // Parameters: Pointers to each state. The objects will
14:   // be deleted by the destructor
15:   BiStateSource (SourceState* on, SourceState* off);
16:   ~BiStateSource();
17:   void Start(timeunits now);
18: };
```

Se pide

1. Considere un multiplexor ATM con una cola de salida de k posiciones cuya probabilidad de desbordamiento interesa conocer. El multiplexor está conectado a un enlace STM1, con capacidad para 353207.547 cél/s disponibles para la capa ATM. Sólo el 2 % de esa capacidad está disponible para las fuentes ON-OFF.

   Las fuentes de voz codificadas con supresión de silencios muestran duraciones medias de los silencios de 650 ms, y duraciones medias de las ráfagas de 350 ms. Estas duraciones dependen del tipo de codec usado. El codec de interés genera células ATM a una tasa de 32 Kb/s.

   Construya un modelo del sistema `atmux.cc` cuya línea de invocación sea:

```
$ atmux   <1seed> <2time> <3mux.k> <4mux.rbin> <5ton> <6toff>
          <7on.rate> <8on.size> <9numsources>
```

2. Utilice dicho modelo para determinar la probabilidad de pérdida de célula en la cola del multiplexor. Represente las curvas correspondientes, junto con la curva teórica. Puede encontrar ésta en [10]. Compárelas cualitativamente.

Entregue una memoria impresa de esta práctica en la que conteste razonadamente a los puntos anteriores. Explique paso a paso cómo ha llegado a la respuesta.

Entregue también un directorio con los programas (.cc, .h o .tcl) y makefiles necesarios, pero no incluya ficheros de trazas, objetos ni ejecutables.

Referencias y Lecturas Recomendadas

[1] Kleinrock, L. *Queuing Systems. Volume II, Computer Applications* John Wiley & Sons, 1976.

[2] Fall, K., Varadhan, K. *The ns Manual (formerly ns Notes and Documentation).* 2002. Disponible en `http://www.isi.edu/nsnam/ns/ns-documentation.html`

[3] Jain, R. *The Art of Computer Systems Performance Analysis*, John Wiley & Sons, 1991.

[4] Law, A.M., Kelton, W.D. *Simulation Modeling and Analysis*, McGrawHill, 2 Ed. 1991.

[5] Stroustrup , B. *El Lenguaje de Programación C++*. Edición Especial. Addiso-Wesley. ISBN :84-7829-046-X

[6] Leland, T., Willinger, W. *On the Self-Similar Nature of Ethernet Traffic (Extended Version)*, IEEE Transactions on Networking v2n1, Feb 95, p.1-15.

[7] Paxson, V., Floyd. S., *Wide Area Traffic: The Failure of Poisson Modeling*, IEEE Transactions on Networking, v3n3, Jun 95, p.226-244.

[8] Pawlikowski, K., Hae-Duck J.J., Jong-Suk R.L. *On credibility of simulation studies of telecommunication networks.* IEEE Communications Magazine, Ene 2002.

[9] Leon-García, A., Widjaja, I., *Redes de Comunicación.* McGraw-Hill, 2002. ISBN:84-481-3197-5

[10] Pitts, J.M.; Schormans, J.A., *Introduction to ATM design and performance* John Wiley and Sons, 2a. Ed.. ISBN:0-471-49187-X.