n°: 2006–no tiene

Proyecto de Grado

Presentado ante la ilustre Universidad de Los Andes como requisito parcial para obtener el Título de Ingeniero de Sistemas

Ajuste automático de las ganancias del PI-AQM de ns-2

Por

Br. Mariemma K. Sosa

Tutor: Dr. Richard Márquez

Cotutor: Prof. Andrés Arcia

Septiembre 2006

Ajuste automático de las ganancias del PI-AQM de ns-2

Br. Mariemma K. Sosa

Proyecto de Grado — Control y Automatización, 66 páginas

Resumen: Carrero (2006), haciendo uso del PI discreto de Hollot et al. (2002), propuso una serie de reglas de ajuste para los parámetros del control PI-AQM para el manejo activo de colas. En este trabajo se diseñó un algoritmo que realiza el ajuste del PI-AQM de *Network Simulator 2* (*ns-2*) de manera automática basado en este conjunto de reglas, usando el distanciamiento de los valores de la cola del enrutador respecto a la cola promedio para modificar la diferencia $(a_c - b_c)$ entre los parámetros a_c y b_c del PI. Ajustando el valor de b_c, se aumenta o disminuye el valor de la ganancia $K = a_c - b_c$, logrando así modificar el tamaño de las oscilaciones de la cola según sea requerido. Además, se añadió una nueva regla de ajuste del PI a las reglas ya existentes para realizar un entonamiento más preciso del controlador. Las pruebas realizadas para ajustar los valores iniciales y probar la robustez del algoritmo ante cambios en la red fueron realizadas en *ns-2* usando diferentes configuraciones de red.

Palabras clave: Control de congestión, manejo activo de colas, PI-AQM, entonamiento automático

Este trabajo fue procesado en LaTeX.

Índice

Índice de Figuras

Introducción

Cuando se transmiten o envían datos a través de Internet, un emisor particular puede perder parte de esos datos (paquetes de datos). Tales pérdidas ocurren en muchos casos debido a un suceso llamado **congestión**.

Desde mediados de los años noventa, se ha hecho necesario el análisis de la transmisión de datos para así poder buscar soluciones cada vez más eficientes al problema de congestión. Este análisis, gracias a la configuración punto a punto puede reducirse al estudio de tres elementos fundamentales: "la fuente", "el destino" y "el enrutador" (Saltzer et al. 1984). Los dos primeros componentes conforman las entidades que intercambian datos, mientras que "el enrutador" representa un elemento fundamental para la existencia de Internet, ya que es el ente encargado de evitar que ocurran un gran número de congestiones y permite la conexión entre diferentes redes.

Los enrutadores tienen una capacidad finita, y valga decir insuficiente, para procesar todas las peticiones de servicio. Debido a esto, surge el problema de la pérdida de datos, generada por una demanda de servicio que sobrepasa la capacidad del enrutador (congestión).

Para lograr dar con nuevas soluciones al problema de congestión, o perfeccionar las ya existentes, es necesario estudiar el protocolo de control de transmisión (TCP, *transmission control protocol*), que ajusta dinámicamente la tasa de envío (o tamaño de la ventana) en las fuentes (Noureddine & Tobagi 2002), y los algoritmos de manejo activo de colas (AQM, *active queue management*), los cuales gestionan el tamaño de la memoria del enrutador para mantenerla en un valor deseado, menor al tamaño físico máximo de la misma.

Planteamiento del problema

Los algoritmos existentes a finales de los años noventa, aunque necesarios y poderosos, sólo existían en Internet y su rendimiento era ineficaz en determinadas situaciones. Jacobson (1988), cuando propuso los mecanismos de control de flujo, ya anticipaba que sólo los enrutadores tenían la información suficiente para controlar la distribución de la capacidad de la red entre todas las conexiones, debido a que en ellos convergen los flujos y pueden observar la red de forma completa.

Estos mecanismos, ejecutados en los enrutadores para complementar el control punto a punto, se denominan algoritmos para el manejo activo de la cola (AQM, *active queue management*) porque gerencian el tamaño de la cola mediante la eliminación de paquetes, veáse (Hassan & Jain 2004) para más detalles.

La ventaja de los algoritmos AQM es que su implementación sólo atañe a un elemento de la red, el "enrutador", no siendo necesario cambiar los demás componentes para su utilización.

Muchos esquemas AQM han sido estudiados en trabajos recientes, ver (Long et al. 2005). En la Tabla 1 se presentan los diferentes esquemas AQM organizados según su método de diseño[1].

Uno de estos algoritmos AQM, específicamente el PI-AQM propuesto por Hollot et al. (2002), fue estudiado en profundidad por Carrero (2006). De este estudio se derivaron una serie de reglas de ajuste del control PI-AQM las cuales llamaremos reglas Carrero-Márquez o simplemente reglas CAM.

La presente investigación tuvo como objetivo extender y evaluar mediante diversas pruebas y bajo diferentes escenarios los resultados obtenidos por Carrero (2006), con el objeto de proponer un algoritmo de entonamiento automático para las ganancias del PI-AQM implementado en *ns-2*. La versión de *ns-2* en la que se ha trabajado es la versión 2.29, instalada en Linux-Ubuntu y ubicada en `/usr/local/src/ns-2.29`. Tanto el algoritmo que se obtuvo de este estudio, como los resultados y conclusiones derivados de él se expondrán en los siguientes capítulos.

[1]Tomado de Long, Zhao, Guan y Yang "*The yellow active queue management algorithm*"(2005).

Categoría	**AQMs**	**Método de diseño**	**Ventajas/Desventajas**
Uso del tamaño de la cola	RED	Heurístico; tamaño promedio de la cola; controlador pseudo proporcional.	Elimina problemas de sincronización global; respuesta lenta; rango de entrada de referencia fluctuante en la salida de la planta.
	ARED	Modifica p_{max} adaptativamente basándose en la cola promedio.	Soluciona uno de los problemas de configuración de parámetros;´´avg" resulta en un efecto dominó de retraso (*lag domino effect*).
	SRED	Estima el número de fuentes activas sin los estados de cada una de las fuentes.	Mitiga la pesada fluctuación de la cola instantánea; poca confiabilidad en la exactitud de la estimación.
	DRED	Heurístico; EWMA del error de la cola instantánea; control integral.	Estabiliza cola instantánea en la cola deseada; problemas en escogencia de la ganancia del control.
	PI	Método clásico de teoría de control; error de la cola instantáneo.	Mejora sensibilidad; elimina error en estado estable; desempeño local debido a linealización.
Uso de la cola y de la carga	REM	Error de la cola instantáneo y proporción desigual; método de optimización.	Desacopla índice de congestión e índice de rendimiento; estabiliza la proporción de entrada alrededor de la capacidad del enlace como la cola alrededor de un objetivo pequeño; calcula peso; parámetro de sensibilidad.
	VRC	Controlador PID; método de análisis de estabilidad.	Mejora desempeño de la sensibilidad; desempeño local debido a linealización.
Uso de la carga	Blue	Heurístico; control on-off modificado.	Respuesta rápida; retardos largos y fluctuantes en la formación de colas bajo tráfico dinámico.
	AVQ	Método de optimización; modelo *token bucket* modificado.	Respuesta rápida; retardos pequeños en colas; los retardos de las colas se incrementan con el crecimiento de la congestión.
	Yellow	Utilidades principales/secundarias; análisis de la teoría de control	Respuesta rápida; estabilidad en la cola; pequeñas fluctuaciones del retardo en las colas.

Tabla 1: Clasificación de los algoritmos AQM en términos de su diseño

Organización

Este trabajo está organizado de la siguiente manera. En el Capítulo 1 se presentan las reglas de ajuste propuestas por Carrero (2006) para el algoritmo del PI-AQM de Hollot et al. (2002) implementados actualmente en *ns-2*. Además, se presenta una nueva regla de ajuste que surgió durante la realización del trabajo y la que fue añadida a las actuales reglas de ajuste CAM.

En el Capítulo 2 se presenta el algoritmo diseñado para realizar el entonamiento del PI-AQM de forma automática y, además, su implementación en *ns-2*. Los resultados obtenidos del algoritmo propuesto usando *ns-2* se exponen en el Capítulo 3. Finalmente, se presentan las conclusiones y recomendaciones derivadas de este trabajo.

Capítulo 1

Reglas de ajuste del control PI-AQM para el manejo activo de colas

El algoritmo del controlador PI en *ns-2* está basado en el esquema AQM propuesto por Hollot et al. (2002), el cual está implementado de manera discreta.

El cálculo de la la señal de control p dentro de *ns-2* se realiza dentro de una función llamada `calculate_p` que se encuentra en el archivo `pi.cc` y se realiza por medio de la siguiente expresión:

$$p = a_c \cdot (qlen - qref) - b_c \cdot (qold - qref) + v\_prob \tag{1.1}$$

donde p es la probabilidad de pérdida de paquetes o señal de control; a_c y b_c son los parámetros del controlador PI; *qlen* es el tamaño de la cola en el instante actual; *qold* es el tamaño de la cola en el instante anterior; $qref$ es el tamaño de cola deseado; *v_prob* es la probabilidad de pérdida de paquetes en el instante anterior.

Uno de los problemas que presentaba el algoritmo PI-AQM existente es que debía conocerse la configuración de la red para poder ajustar las ganancias del controlador. Ver, por ejemplo, el archivo `picoeff.dat.gz` del directorio `/tcl/ex/pi` de *ns-2*, que en nuestro caso se encuentra específicamente en `/usr/local/src/ns-2.29/tcl/ex/pi`, donde se presenta una lista de valores propuestos para los parámetros del PI-AQM que sólo pueden ser usados si se conocen las condiciones de la red.

Como ya se ha dicho, Niliana Carrero en su proyecto de grado (2006), propuso una serie de reglas de ajuste de los parámetros del PI-AQM (reglas CAM). Estas reglas ajustan las ganancias de los parámetros del controlador para mantener el tamaño de la cola alrededor de un valor deseado y aminorar las oscilaciones sin necesitar tener conocimiento previo de la configuración de la red.

Al finalizar su estudio, Carrero (2006) planteó tres reglas de ajuste que permiten estimar las ganancias adecuadas de los coeficientes a_c y b_c del control PI-AQM basadas en las tendencias del comportamiento cualitativo de la cola del enrutador.

Las reglas de ajuste CAM son las siguientes:

- Para que el sistema sea estable, es necesario:
  Regla 1: b_c menor que a_c, $b_c < a_c$.

- Para atenuar las oscilaciones de la cola del enrutador, se recomienda:
  Regla 2: Acercar b_c a a_c, $b_c \rightarrow a_c$.

- Se mostró que no es necesario emplear altas frecuencias de muestreo. De hecho, emplear frecuencias de muestreo bajas ($<$ 1 KHz) puede permitir, bajo ciertas condiciones, obtener mayores rangos de operación de a_c y b_c. Por ello, se recomienda:
  Regla 3: Valor de frecuencia w pequeño

Para llegar a plantear estas reglas, Carrero (2006) realizó una serie de estudios analíticos para comprobar la controlabilidad y estabilidad del controlador y así proseguir a realizar sus pruebas de entonación. A continuación se hace una breve revisión de sus resultados. Veáse (Carrero 2006) para más detalles.

Usando el modelo promedio no lineal para la ventana de congestión (Marquez et al. 2004), y el modelo de la cola del enrutador, (Hollot et al. 2002) se obtuvo el sistema no lineal TCP-enrutador en lazo abierto, dado por las siguientes ecuaciones diferenciales:

$$\begin{aligned} \frac{d\bar{w}(t)}{dt} &= \frac{1}{RTT}\left[a - \left(a + \frac{2b}{2-b}\bar{w}\right)\bar{w}p\right] \\ \frac{dq(t)}{dt} &= \frac{N}{RTT}\bar{w} - C \\ RTT &= \frac{q}{C} + T_p \end{aligned} \tag{1.2}$$

donde $\bar{w}$ y q representan, respectivamente, la ventana de congestión y la cola del enrutador; RTT (*round trip time*) es el tiempo de ida y vuelta en segundos; N es el número de fuentes TCP; C es la capacidad del enlace del enrutador en paquetes/segundos; T_p es el retardo que poseen las fuentes medido en segundos; q_{ref} es el valor deseado de la cola del enrutador; a y b son los parámetros de incremento y decremento para el TCP respectivamente; p la probabilidad de pérdida de paquetes. La variable p corresponde a la señal de control.

Luego de linealizar el sistema alrededor del punto de equilibrio y estudiar su controlabilidad, se halla el polinomio característico en lazo cerrado del sistema lineal, el cual queda:

$$P_c(s) = \det[sI - \bar{A}] = s^3 + \alpha_2 s^2 + \alpha_1 s + \alpha_0 \tag{1.3}$$

donde

$$\alpha_2 = \left(\frac{C}{(Q+T_pC)} - \frac{aCN(-2aN+abN-4bQ-4bCT_p)}{Q+T_pC)^2(2aN-abN+2bQ+2bCT_p)} \right)$$

$$\alpha_1 = \left(-\frac{aC^2N(-2aN+abN-4bQ-4bCT_p)}{(Q+T_pC)^3(2aN-abN+2bQ+2bCT_p)} - \frac{a_cC^2N^3(2aN-abN+2bQ+2bCT_p)}{(b-2)(Q+T_pC)} \right)$$

$$\alpha_0 = -\frac{(a_c-b_c)C^2(2aN-abN+2bQ+2bCT_p)}{(b-2)N\Delta T(Q+T_pC)}$$

Para la realización de las pruebas, se tomaron 5 casos:

- Caso 1: $b_c > a_c$
- Caso 2: $b_c = a_c$
- Caso 3: b_c muy alejado de a_c ($a_c > b_c$)
- Caso 4: b_c cerca a_c ($a_c > b_c$)
- Caso 5: b_c muy cerca a_c ($a_c > b_c$)

Cuando el valor de b_c es ligeramente mayor al valor del parámetro a_c (caso 1), la cola no se estabiliza en el valor de referencia deseado (igual ocurre en el caso 2, cuando a_c y b_c son iguales). De los casos anteriores surge la regla 1.

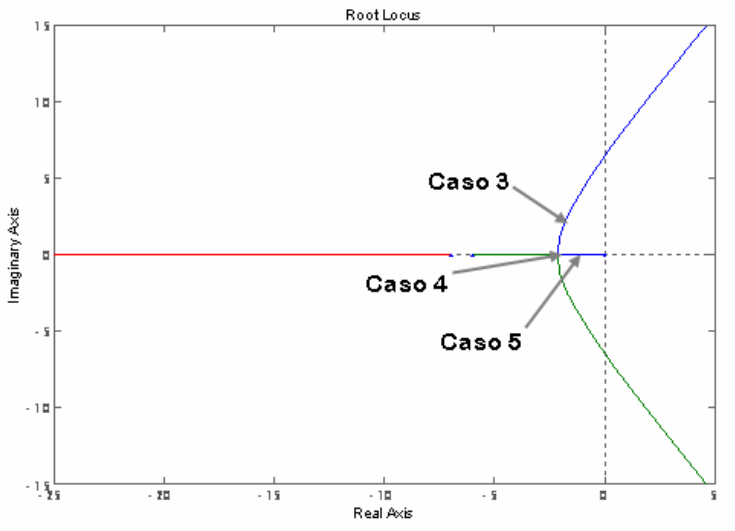

Figura 1.1: Lugar geométrico de las raíces para $K = a_c - b_c$

En los tres casos restantes, la cola se estabiliza en el valor deseado. En el caso 3, la cola presenta oscilaciones y la respuesta del sistema es más rápida que para los casos 4 y 5. En estos tres casos se basa la regla 2.

El lugar de las raíces de la Figura 1.1, cuando el parámetro $K = a_c - b_c$ varía, muestra estos tres últimos casos y los compara con los resultados que se obtuvieron en *ns-2*.

Se observó que al variar el valor b_c de manera que el valor de K fuese pequeño (caso 5), el sistema presentaba una raíz real dominante, lo que hacía que la respuesta del sistema fuese lenta. Al aumentar la diferencia entre a_c y b_c, la raíz se aleja del eje imaginario, haciendo que la respuesta del sistema sea más rápida (caso 4), luego se hace oscilatorio (caso 3).

En el presente trabajo, al analizar detenidamente estos tres últimos casos se propone una regla complementaria a la regla 2 la cual dice que si en el sistema no se presentan oscilaciones, b_c *se puede alejar de* a_c, haciendo el sistema más rápido sin que llegue a ser oscilatorio.

Esta regla fue denotada como regla 2b, quedando las reglas CAM de la siguiente forma:

- Para que el sistema sea estable, es necesario:

Regla 1: b_c menor que a_c, $b_c < a_c$.

- Para atenuar las oscilaciones de la cola del enrutador, se recomienda:
  Regla 2a: Acercar b_c a a_c, $b_c \rightarrow a_c$

- Si en el sistema no se presentan oscilaciones, se sugiere:
  Regla 2b: Alejar b_c de a_c, $b_c \rightarrow 0$.

- Se mostró que no es necesario emplear altas frecuencias de muestreo. De hecho, emplear frecuencias de muestreo bajas ($<$ 1 KHz) puede permitir, bajo ciertas condiciones, obtener mayores rangos de operación de a_c y b_c. Por ello, se recomienda:
  Regla 3: Valor de frecuencia w pequeño

En el trabajo de Carrero (2006) las reglas 2a y 2b no están escritas explícitamente en la forma en que se presentan aquí.

Capítulo 2

Algoritmo PI auto-ajustable

El algoritmo PI auto-ajustable (o ATPI, tal como fue nombrado) se diseñó de manera heurística, teniendo como fundamento las reglas de ajuste CAM del PI-AQM; Además, se empleó wl valor de la cola promedio basado en el filtro pasa bajo del RED (Floyd & Jacobson 1993).

Partiendo de las reglas CAM se expuso la necesidad de buscar una manera de medir las oscilaciones de la cola del enrutador y de proponer un rango límite de oscilaciones, de manera que si las oscilaciones sobrepasan dicho rango el algoritmo modifique los valores de las ganancias del controlador según las reglas establecidas.

La solución que se planteó fue establecer un valor de referencia, y hacer muestreo durante cierto tiempo de la diferencia entre la cola del enrutador y la referencia escogida. El valor obtenido se compara con el rango de oscilaciones establecido y, según el resultado obtenido, se aplica la estrategia necesaria para entonar el controlador adecuadamente.

En nuestro caso se tomó la cola promedio como valor de referencia, ya que así se puede calcular el grado de distanciamiento de los valores de la cola del enrutador respecto a su valor medio, obteniendo como resultado el grado de dispersión de dicha cola.

Algoritmo

1. Se inicializan los valores de las variables y los parámetros.

2. Se calcula el valor de la cola promedio qp.

3. Se calcula la cantidad de intervalos de muestreo Δt que va a contener la muestra (num_d).

4. Se actualiza el contador $pcount$ que lleva la cuenta de el número de intervalos Δt que han sido muestreados.

5. Se calcula sum (suma acumulativa de los valores absolutos de la diferencia entre la cola actual ($qlen$) y la cola promedio (qp)).

6. Cuando $pcount = num\_d$ se calcula el porcentaje de dispersión de la cola del enrutador respecto a su valor medio ($disp$).

7. Se compara el porcentaje de dispersión con los límites externos ($extlimit$) e internos ($intlimit$ y $lowlimit$) establecidos en el algoritmo.
   Si $disp > extlimit$, entonces $a_c \leftarrow b_c$.
   Si $disp < intlimit$ y $disp > lowlimit$, entonces $b_c \rightarrow 0$.

8. $pcount = 0$ y $sum = 0$.

9. Se calcula la probabilidad de pérdida de paquetes p.

10. Se regresa al paso 2.

El cálculo de la cola promedio se hizo con un filtro pasa bajo que fue propuesto por Floyd & Jacobson (1993) en su algoritmo RED. Este filtro es un promedio móvil exponencial ponderado (EWMA, *exponential weighted moving average*):

$$qp = (1 - w_q) \cdot qpold + w_q \cdot qlen \tag{2.1}$$

donde el peso w_q determina la constante de tiempo del filtro pasa bajo, $qpold$ es el valor de la cola promedio en el instante anterior y qlen el valor de la cola actual.

El valor de dispersión se calcula haciendo una suma acumulativa del valor absoluto de las diferencias entre la cola actual y la cola promedio cada Δt (`sum=abs(qlen-qp)+sum`)) durante el tiempo que dura la muestra (`edp_.ssize`).

Por otra parte, se realiza el cálculo del número de intervalos de muestreo Δt en los que está dividida la muestra (`num_d=floor(edp_.ssize*edp_.w)`) de manera que, cuando el contador iguala el número de intervalos de muestreo que ésta contiene, se procede a calcular la dispersión tomando la suma acumulativa y dividiéndola entre el valor del contador. El resultado obtenido se multiplica por 100 y se divide entre el *qlim* para colocar la dispersión en términos de porcentaje de cola máxima (`disp=((sum/edv_.pcount)*100.0)/qlim`).

Durante el proceso de diseño del algoritmo surgió una nueva regla para la entonación, la regla 2b: si el sistema presenta oscilaciones muy pequeñas se recomienda alejar b_c de a_c para hacer el sistema más rápido sin que sea oscilatorio. Debido a esta nueva regla el rango de oscilaciones esta divido en dos límites principales: el límite externo o `edp_.extlimit` que representa el 5% de la cola máxima y el límite interno o `edp_.intlimit` que es el 1% de la cola máxima.

Si el valor de dispersión está por encima de `edp_.extlimit`, entonces b_c se acerca a a_c para disminuir las oscilaciones; si la dispersión está por debajo de `edp_.intlimit`, b_c se aleja de a_c y si el valor de dispersión se encuentra entre los dos límites, los valores de los parámetros se mantienen.

Al realizar las simulaciones de prueba se observó que cuando la cantidad de paquetes enviada por el enlace no produce oscilaciones en la cola, el parámetro b_c se alejaba hasta casi hacerse cero sin lograr ningún resultado efectivo. Debido a esto surgió un tercer límite llamado `edp_.lowlimit` que representa el 0.5% de la cola máxima y que tiene como función detener el alejamiento de b_c si las dispersiones están por debajo de su valor.

La Figura 2.1 muestra como están ubicados los límites con respecto a la cola del enrutador.

En la Figura 2.2 se puede observar la ubicación de la ganancia $K = a_c - b_c$ según la variación del valor de b_c. Si la diferencia entre a_c y b_c es muy grande, el sistema es oscilatorio **1** por lo que se aplica la regla CAM 2a disminuyendo la diferencia entre a_c y b_c para aminorar las oscilaciones. Si, por el contrario, el valor de K es muy pequeño, lo que hace la respuesta muy lenta **3**, se aplica la regla CAM 2b que aleja b_c de a_c, **3** para aumentar la diferencia y hacer que el sistema responda más rápidamente.

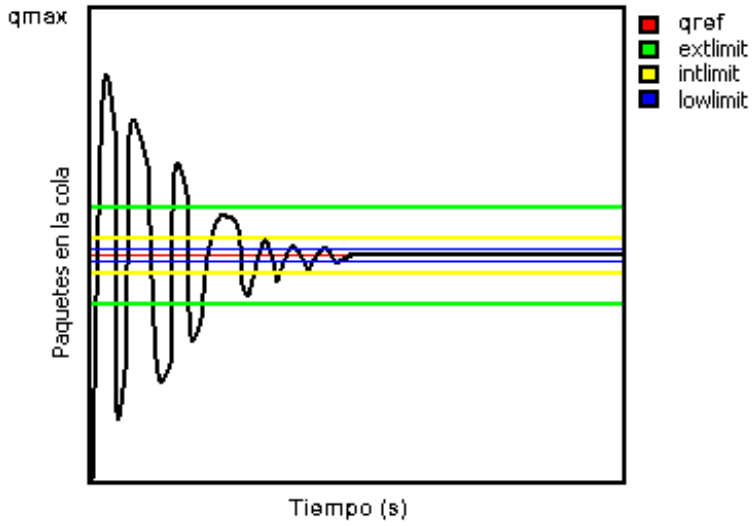

Figura 2.1: Límites del rango de oscilaciones

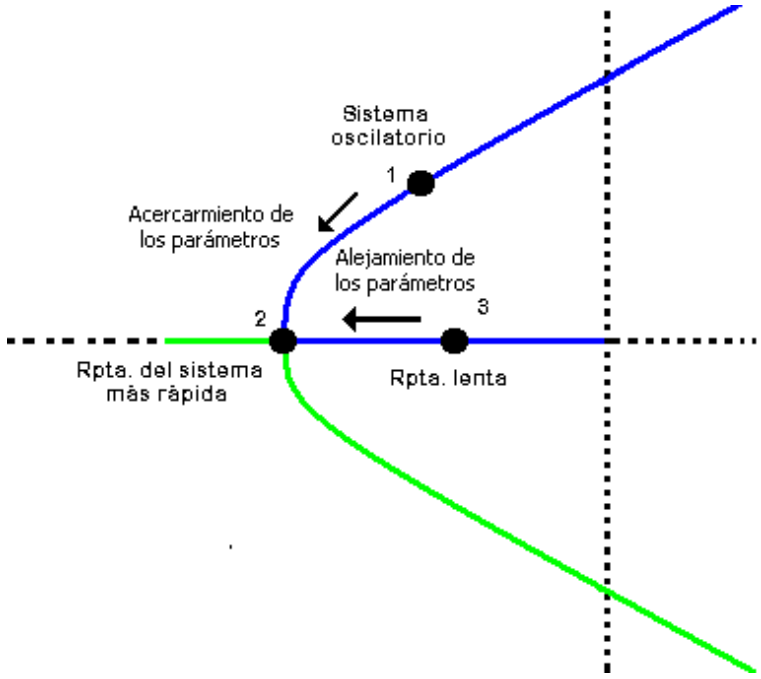

Figura 2.2: Ubicación de la ganancia K en el lugar geométrico de las raíces

El mecanismo que se ideó para realizar la variación de b_c fue usar una constante, a la cual se le nombro α, con un valor menor a 1. Cuando se desea acercar los parámetros

del controlador, el nuevo valor de b_c está definido por la siguiente expresión:

$$b_c = (1 - \alpha) \cdot b_c + \alpha \cdot a_c \tag{2.2}$$

cuyo código está dado por

$$bc = ((1.0 - edp\_alpha) * bc + edp\_alpha * ac) \tag{2.3}$$

En el caso de querer alejar b_c de a_c, el nuevo valor de b_c será el resultado que se obtenga de la expresión:

$$b_c = \alpha \cdot b_c \tag{2.4}$$

codificado de la siguiente forma:

$$bc = edp\_alpha * bc \tag{2.5}$$

La razón por la cual se usa este método para variar b_c es que de esta manera aseguramos que en el caso de acercar los valores, b_c no tome un valor mayor que a_c, respetando así la regla CAM 1, y en el caso de alejarlos, que b_c no se torne negativo. Ver la Figura 2.3.

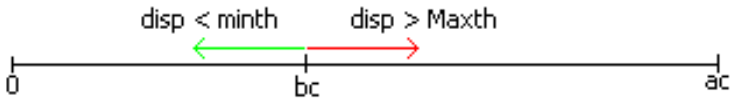

Figura 2.3: Ubicación de b_c en el eje real

2.1 Implantación del algoritmo en *ns-2*

Network Simulator 2 (*ns-2*) proporciona gran cantidad de soporte para la simulación de múltiples protocolos, tanto para redes cableadas como no cableadas, locales o vía satélite, y con topologías complejas que poseen gran número de generadores de tráfico. También contempla mecanismos referentes a la capa de enlace de datos en redes y diversos algoritmos para la planificación de colas, véase (Miguel 2003) para más detalles. Por todas las razones anteriormente expuestas, *ns-2* es el simulador de redes más usado actualmente y es el escogido para implementar el algoritmo diseñado y realizar sus pruebas.

El *ns-2* soporta una jerarquía compilada escrita en C++ y otra jerarquía similar a la anterior pero dentro del intérprete que se presenta al programador en OTcl conocida como jerarquía interpretada. Estas dos jerarquías están estrechamente relacionadas entre sí, de modo que desde la perspectiva del usuario hay una correspondencia uno a uno entre ellas. Cuando el usuario crea un objeto simulador desde el intérprete, con lo que generalmente se comienza un script, este objeto es instanciado dentro del intérprete y se crea una estrecha relación con otro objeto idéntico, pero dentro de la jerarquía compilada (Fall & Varadhan 2002).

Para realizar la implantación del algoritmo dentro de *ns-2* se modificó el archivo `ns-default.tcl`, que se encuentra en `/usr/local/src/ns-2.29/tcl/lib`, el cual es el archivo donde se encuentran los valores por defecto de *ns-2*. Además, se agregaron dos nuevos archivos al simulador, el `atpi.cc` (archivo fuente) y el `atpi.h` (archivo de encabezamiento) ubicados en `.../ns2.29/queue`. Estos archivos se encuentran escritos en C++, el lenguaje de programación en el que está estructurada la jerarquía compilada de *ns-2*; para su creación se uso como base copias de los archivos `pi.cc` y `pi.h` ya existentes en *ns-2*.

En el archivo de encabezamiento (`atpi.h`) se crearon los parámetros de pérdidas tempranas (edp, *early drop parameters*) y las variables de pérdidas tempranas (edv, *early drop variables* necesarias para el funcionamiento del algoritmo dentro del simulador.

Los *edp* son los parámetros que pueden ser administrados por el usuario mientras que las *edv* son las variables provistas por el controlador. Uno de los primeros cambios que se le hizo al PI implementado en ns-2 fue pasar las ganancias del controlador de parámetros edp a variables edv, de manera que estos valores sean manejados por el PI y no por el usuario y puedan ser graficados.

En `atpi.cc` se realiza el enlace entre la jerarquía compilada y la jerarquía interpretada de *ns-2*. Es donde se codifica todo lo referente a la implementación de los métodos de se declararon en el `atpi.h`. La función más importante para nosotros dentro de este archivo es la llamada `calculate_p`, ya que es dentro de esta función donde se encuentra codificado el algoritmo ATPI.

Luego de copiar estos dos archivos dentro de la carpeta correspondiente en *ns-2*,

se realizó su compilación para así obtener el archivo ejecutable del ATPI (`atpi.o`) y poder usar el algoritmo dentro de *ns-2*.

Función calculate_p: a continuación, se muestra la función `calculate_p`, codificada en C++, donde se encuentra implementado el algoritmo de ajuste automático de las ganancias del PI para su funcionamiento en *ns-2*.

```
double ATPIQueue::calculate_p() {
    double disp = edv_.disp;
    double ac = edv_.ac;
    double bc = edv_.bc;
    double sum = edv_.sum;
    double p, qp, num_d;
    int c;
    int qlen = qib_ ? q_->byteLength() : q_->length();
    int qlim = qlim_;       if (qib_) {
        qlen = qlen * 1.0 / edp_.mean_pktsize;}

    qp = ((1.0 - edp_.wq) * edv_.qpold + edp_.wq * qlen);

    num_d = floor(edp_.ssize * edp_.w);
    edv_.pcount = ++edv_.pcount;
    sum = abs(qlen - qp) + sum;
    if (edv_.pcount == num_d) {
        disp = ((sum/edv_.pcount)*100.0)/qlim;
        if (disp > edp_.extlimit)
                bc = ((1.0 - edp_.alpha) * bc + edp_.alpha * ac);
        if (disp < edp_.intlimit && disp > edp_.lowlimit)
            bc = edp_.alpha * bc;
        edv_.pcount = 0;
        sum = 0;}
```

```
p = ac * (qlen - edp_.qref) - bc * (edv_.qold - edp_.qref) + edv_.v_prob;

if (p < 0) p = 0;
if (p > 1) p = 1;

edv_.v_prob = p;
edv_.qold = qlen;
edv_.qpold = qp;
edv_.qprom = qp;
edv_.disp = disp;
edv_.sum = sum;
edv_.ac = ac;
edv_.bc = bc;

CalcTimer.resched(1.0/edp_.w);
return p;}
```

Capítulo 3

Simulación en *ns-2*

3.1 Escenario de simulación en *ns-2*

Para la realización de las pruebas del algoritmo se usaron como valores base los obtenidos por Hollot et al. (2002) para el PI-AQM. Se hace uso de fuentes TCP *NewReno*. Se usan enlaces dobles para la comunicación debido a que TCP requiere del envío de un ACK cuando los paquetes llegan al destino. Los enlaces de las fuentes tienen un retardo de 1 ms, una capacidad de 1000 Mb/s (250000 paquetes/s)) y el mecanismo de manejo de cola usado es *DropTail*. El enlace enrutador-destino también es doble, posee retardo de 10 ms, capacidad de 30 Mb/s (7500 paquetes/s) y el algoritmo AQM que rige la cola es el ATPI. Una representación gráfica de la topología de red usada se muestra en la Figura 3.1

Según lo expuesto anteriormente, el enlace enrutador-destino tiene menor capacidad que los enlaces de las fuentes. Esto se debe a que se desea provocar un "cuello de botella" en el enlace de salida para probar la eficiencia del algoritmo haciendo la entonación del PI.

3.2 Pruebas

A fin de ajustar los valores bases del algoritmo y probar su resistencia a los cambios en la red se realizaron dos tipos de pruebas del ATPI: pruebas de entonación y pruebas de

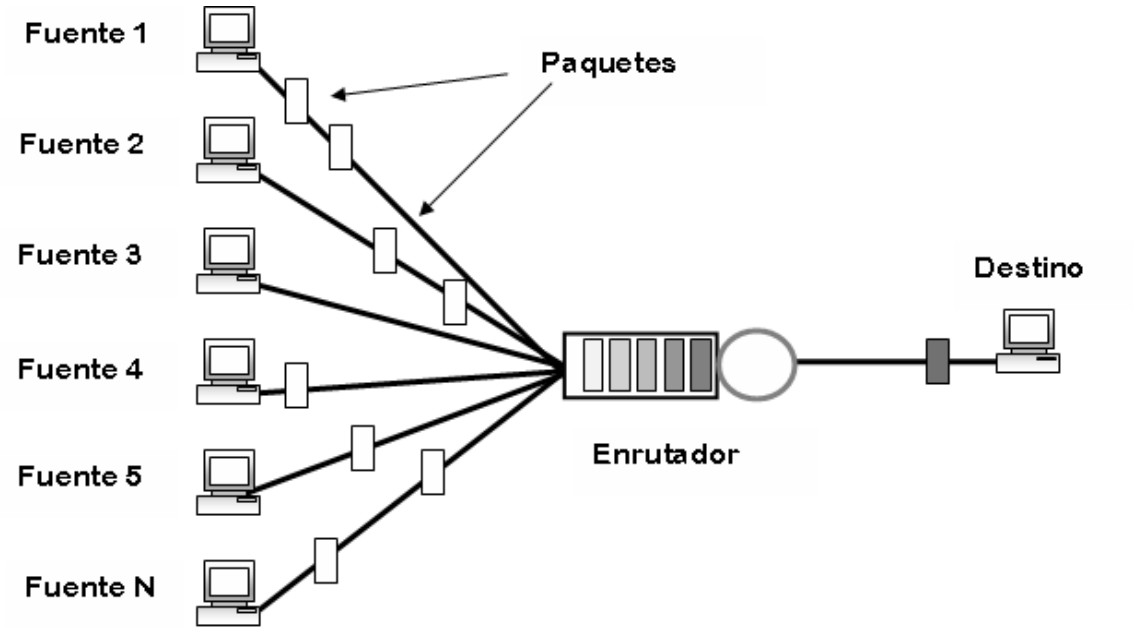

Figura 3.1: Topología de red tipo estrella: N fuentes, 1 enrutador y 1 destino

robustez. Las pruebas de entonación corresponden a los ensayos realizados haciendo cambios en la ubicación inicial de b_c y en el orden de magnitud de los parámetros iniciales del controlador. En el caso de las pruebas de robustez, se efectuaron simulaciones con diferentes RTT, tráfico UDP, cambios en en el número de fuentes (un ejemplo propuesto en (Carrero 2006)) y en la capacidad C del enrutador. Por último, se realizó una prueba con una topología tipo estrella conformada por 6 enrutadores y con tráfico cruzado UDP.

En las siguientes simulaciones sólo se muestran las variables que, a nuestra consideración, son de importancia para observar el desempeño del algoritmo ATPI, dichas variables son: tamaño de la cola del enrutador, variación de los parámetros del controlador PI y la probabilidad de pérdida de paquetes. Si desea mayor información acerca de como se realizaron las simulaciones que se presentan a continuación o de los scripts de simulación puede revisar la versión digitalizada del proyecto de grado en el directorio *ns-2*.

3.2.1 Pruebas de entonación

Las pruebas de entonación se llevaron a cabo con el fin de buscar los valores por defecto más adecuados para lograr un buen desempeño del algoritmo. Se realizaron pruebas para ajustar cada una de las variables y parámetros de manera adecuada pero acá sólo se presentarán las más sobresalientes.

Simulación del experimento base: como se dijo anteriormente, los valores del experimento base fueron los usados por Hollot et al. (2002) para el PI-AQM y que además fueron los mismos que usó Carrero (2006) en el desarrollo de su proyecto de grado.

Valores nominales del experimento base: $N = 200$ fuentes; $a_c = 1.822 \cdot 10^{-5}$ y $b_c = 1.816 \cdot 10^{-5}$; $C = 30$ Mb/s (7500 paquetes); $T_p = 0.022$ s (retardo en los enlaces de las fuentes (1 ms) más retardo del enlace de salida (10 ms) multiplicado por 2 (ida y vuelta)); $q_{ref} = 400$ paquetes; $q_{max} = 1300$ paquetes; $w = 160$ Hz. En lo sucesivo, al hacer referencia al "experimento base" se estará haciendo alusión a los valores anteriormente expuestos.

En la Figura 3.2 se puede observar la forma en que el algoritmo va ajustando el valor de b_c según el comportamiento de las oscilaciones en la cola del enrutador a medida que va transcurriendo la simulación (gráfica central). El ajuste de los parámetros permite a la cola estabilizarse en el valor de referencia ($qref = 400$ paquetes) con un porcentaje de oscilaciones que se encuentra dentro de los límites establecidos (entre el 1% y el 5% de la cola máxima). Se debe recordar que el algoritmo ATPI ajusta el valor del parámetro b_c de acuerdo al tamaño de oscilaciones que presente la cola del enrutador. En la gráfica inferior de la figura se muestra el comportamiento de p (probabilidad de pérdida) a medida que se va modificando el valor de b_c.

Respuesta del sistema ante variaciones del parámetro b_c: en las simulaciones que se presentan en la Figura 3.3 se muestran los resultados correspondientes a la variación del valor inicial del parámetro b_c del controlador PI.

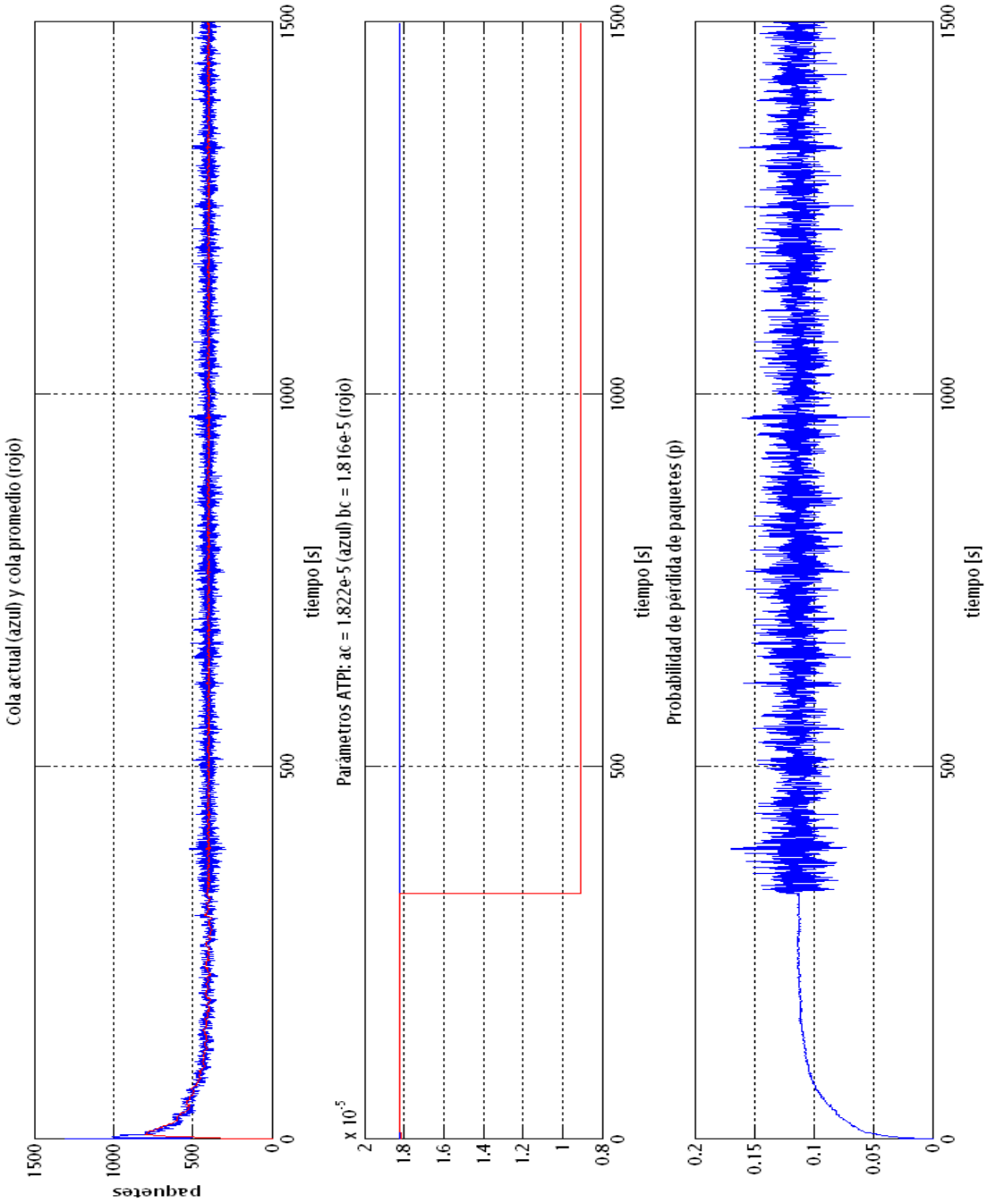

Figura 3.2: Simulación del experimento base: $N = 200$; C=30 Mb/s; $a_c = 1.822 \cdot 10^{-5}$ y $b_c = 1.816 \cdot 10^{-5}$; $w = 160Hz$.

Se puede notar como el cambio del valor inicial de b_c afecta ligeramente el comportamiento de la cola y la probabilidad p. Cuando a_c y b_c se encuentran muy cerca, la cola sufre un cambio brusco para encontrar la estabilización mientras que cuando se alejó el valor de b_c del valor de a_c, la cola se estabiliza rápidamente pero la probabilidad p comienza a oscilar en un tiempo menor al de los otros dos casos.

Aunque ninguno de los cambios es significativo ni afecta el rendimiento del controlador, se decidió dejar los valores iniciales del experimento base como valores por defecto debido a que con ellos se obtiene un buen rendimiento del algoritmo.

Variación de los órdenes de magnitud de los parámetros a_c y b_c: Otra de las pruebas de entonación que se le realizaron al ATPI fue la de cambiar el orden de magnitud de ambos parámetros del PI para observar como afectaba esto su funcionamiento.

En la Figura 3.4 se muestra como al cambiar el orden de los parámetros a 10^{-3} la cola se estabiliza instantáneamente casi sin oscilaciones, lo que provoca que el algoritmo intente alejar los parámetros hasta que el `edp_lowlimit` se lo permite sin obtener ningún resultado efectivo. La probabilidad p es totalmente oscilatoria, lo que indica que el controlador está trabajando excesivamente.

En el caso en el que los parámetros están en el orden de 10^{-8}, la señal de control p no presenta casi oscilaciones pero la cola del enrutador se satura durante toda la simulación sin lograr estabilizarse y b_c disminuye su distancia de a_c hasta casi hacerla cero. En las gráficas centrales de la figura, se encuentran los resultados de la simulación con el orden de magnitud del experimento base (10^{-5}). Tanto el comportamiento de la cola como la probabilidad p es aceptable, al igual que el ajuste de b_c hecho por el ATPI.

3.2.2 Pruebas de robustez

Con el fin de medir la eficiencia del ATPI ante diferentes escenarios, se realizaron una serie de pruebas alterando diversos componentes de la configuración de red del experimento base. Entre las variables que se modificaron para las pruebas se encuentran la capacidad C del enlace, el número de fuentes N, entre otros. También se introdujo tráfico UDP y se probó el algoritmo con una red tipo estrella con 6 enrutadores

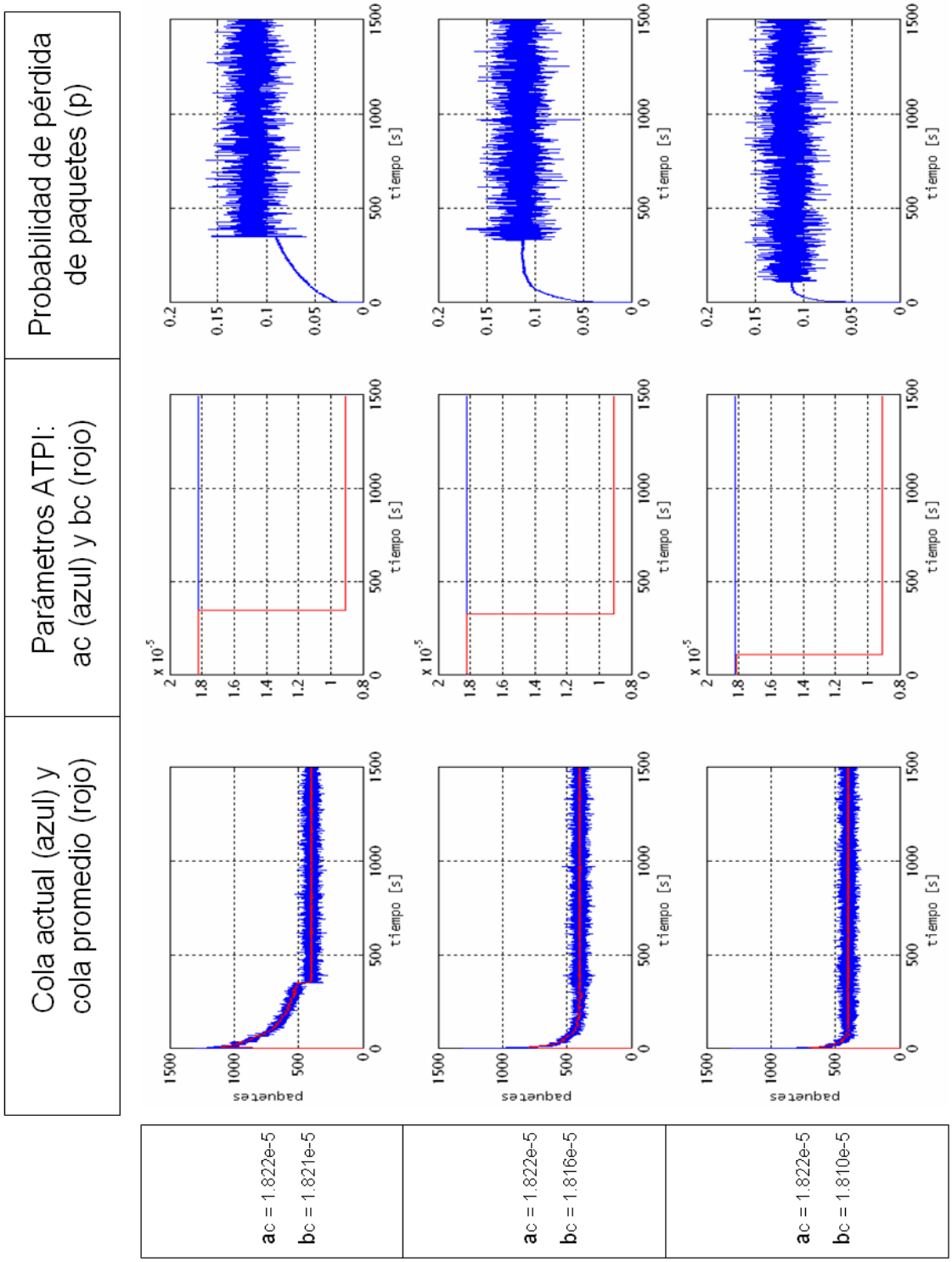

Figura 3.3: Variaciones del parámetro b_c del PI-AQM para $N = 200$

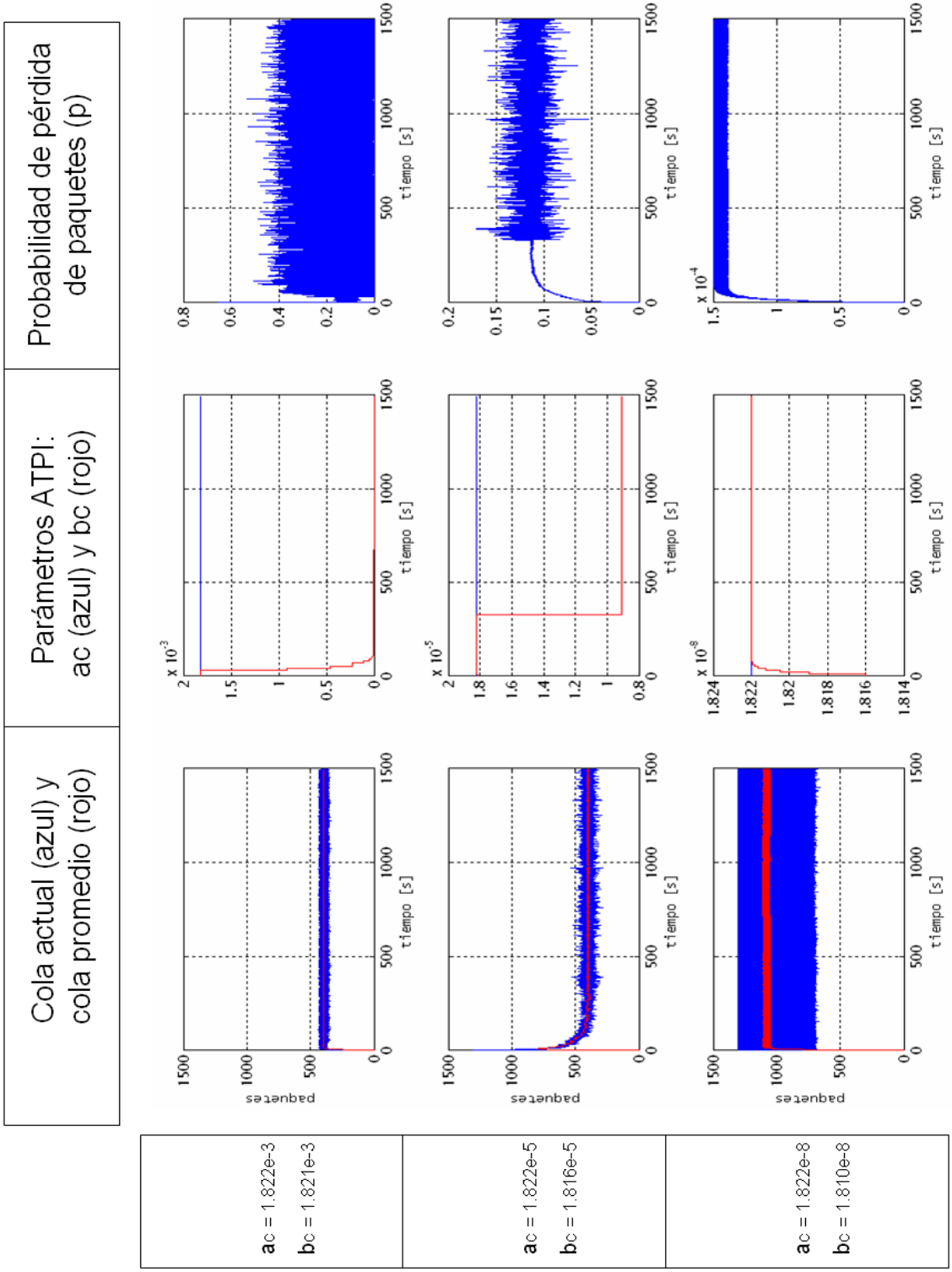

Figura 3.4: Variaciones de los órdenes de magnitud de los parámetros a_c y b_c del controlador para $N = 200$

continuos.

Respuesta en el caso de diferentes RTT: a diferencia del experimento base, en el que todas las fuentes trabajaron con el mismo valor de RTT, en la Figura 3.5 se muestran las simulaciones realizadas generando los RTT aleatoriamente, a través de una función uniforme.

Los resultados obtenidos de la cola del enrutador y ajuste de los parámetros son cualitativamente similares a los observados con RTT iguales en todas las fuentes.

Variación del número de fuentes N: en la Figura 3.6 se presentan la simulaciones resultantes al efectuar cambios en el número N de fuentes a los valores del experimento base. El comportamiento observado es el esperado en todos los casos estudiados.

Tráfico UDP adicional: a fin de observar la dinámica de la cola del enrutador, se realizaron pruebas con diferentes proporciones de tráfico UDP y tráfico TCP. Los resultados que se presentan en la Figura 3.7 son los correspondientes a 200 fuentes TCP y 50 fuentes UDP, 600 fuentes TCP y 150 fuentes UDP y 1000 fuentes TCP y 100 fuentes UDP. Se observa que aunque se ha experimentado con diferentes volúmenes de tráfico se tienen resultados similares que se encuentran dentro de lo establecido por el algoritmo y que son bastante satisfactorios en líneas generales.

Variación de N y de la capacidad C: la Figura 3.8 refleja los resultados de hacer cambios tanto en la capacidad C del enrutador como en el número N de fuentes del experimento base. Las filas 2 y 4 muestran el ajuste del parámetro b_c en los diferentes casos estudiados, mientras que las filas 1 y 3 son las contentivas de la dinámica de la cola del enrutador para cada caso específico. Se realizaron pruebas disminuyendo la capacidad del enlace enrutador-destino a $C = 10$ Mb/s para $N = 20$, $N = 200$ y $N = 1200$. Los mismo experimentos se hicieron aumentando la capacidad a $C = 50$ Mb/s. En general se puede observar, como era de esperarse, que en el caso de tener una menor capacidad en el enlace, el controlador consigue más difícil lograr el ajuste de los parámetros para llevar las oscilaciones dentro de los límites establecidos que cuando el enlace es de mayor tamaño.

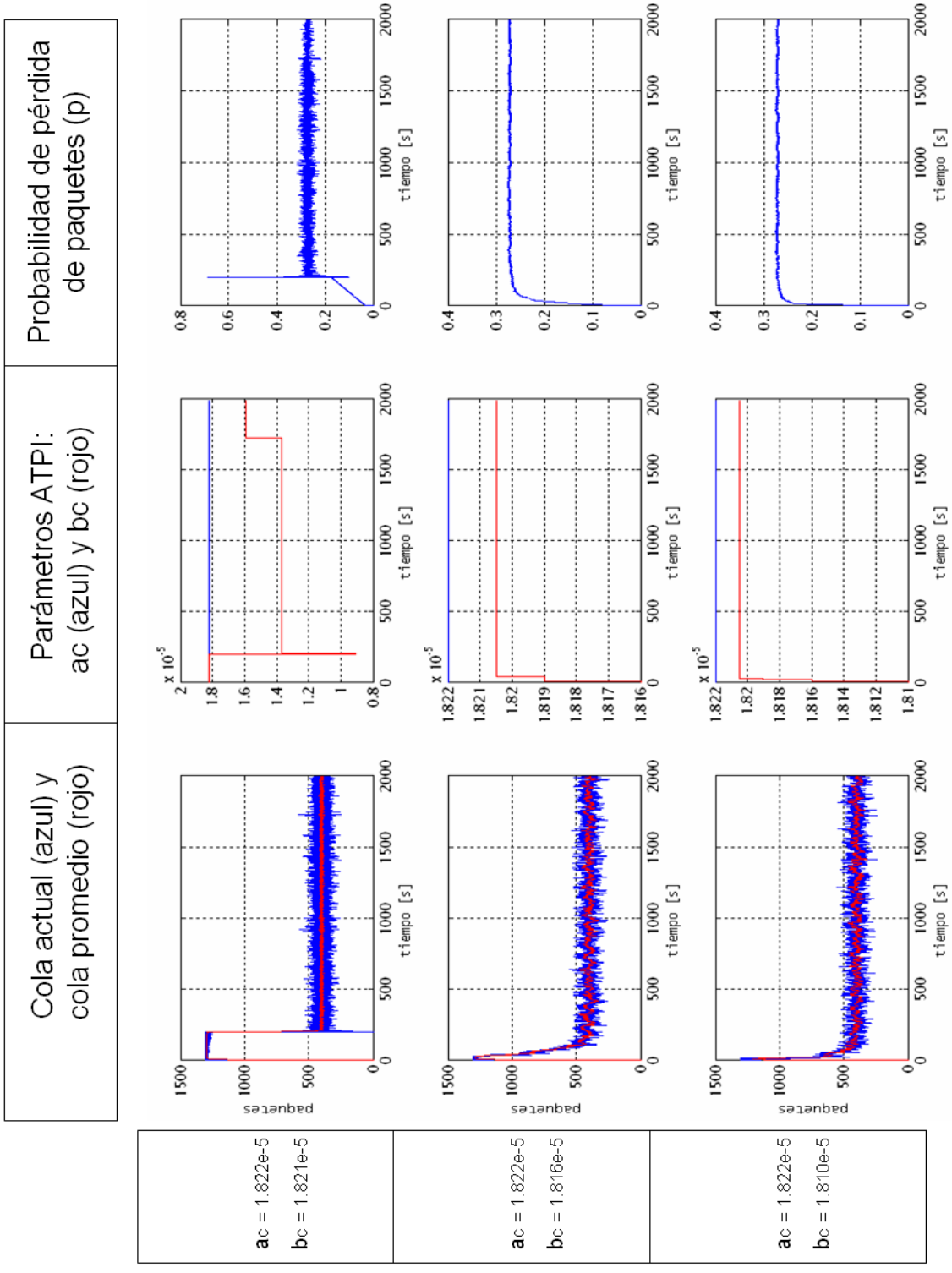

Figura 3.5: Simulación del experimento base con RTT diferentes para todas las fuentes TCP

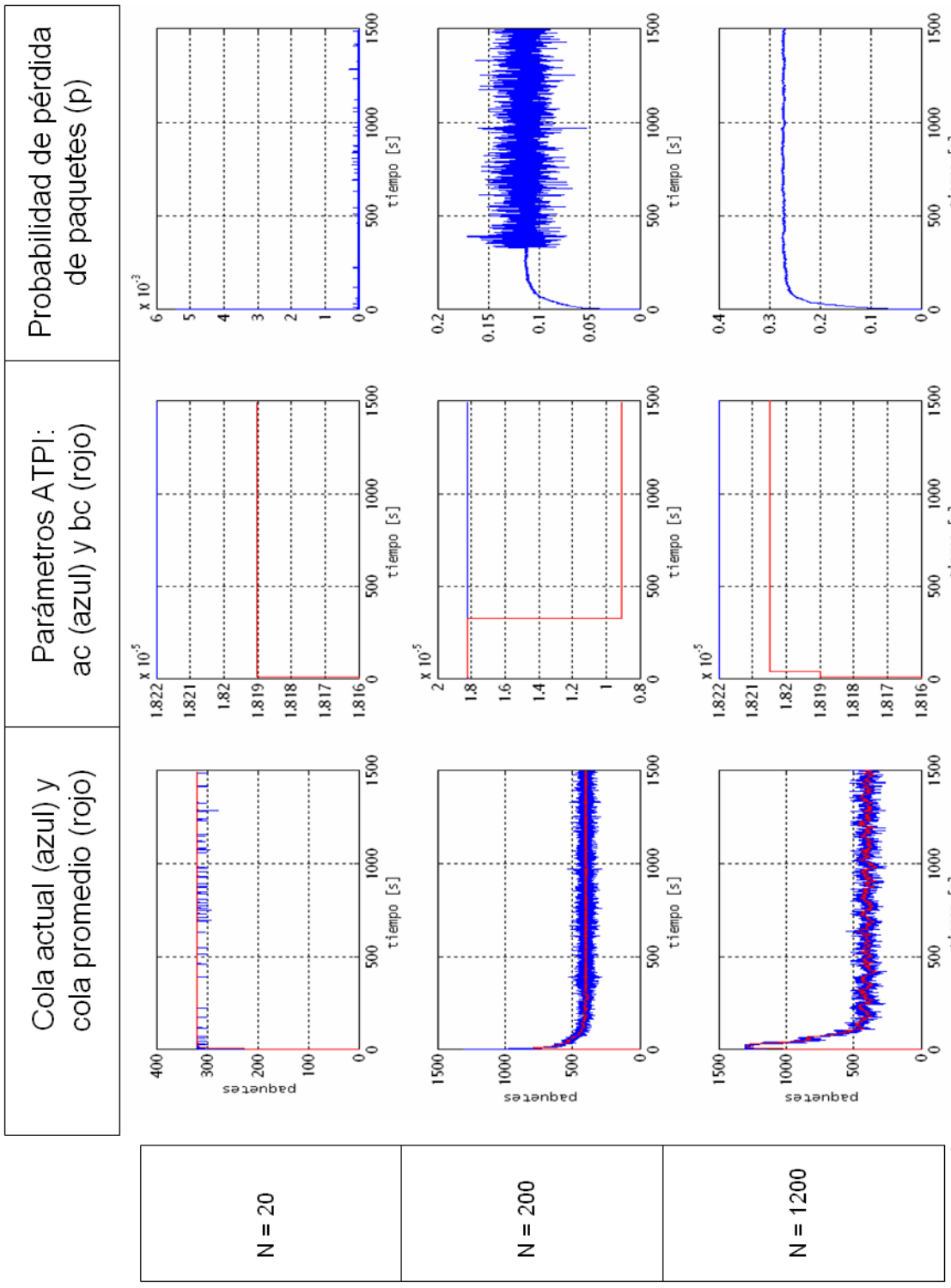

Figura 3.6: Variación de N para el experimento base

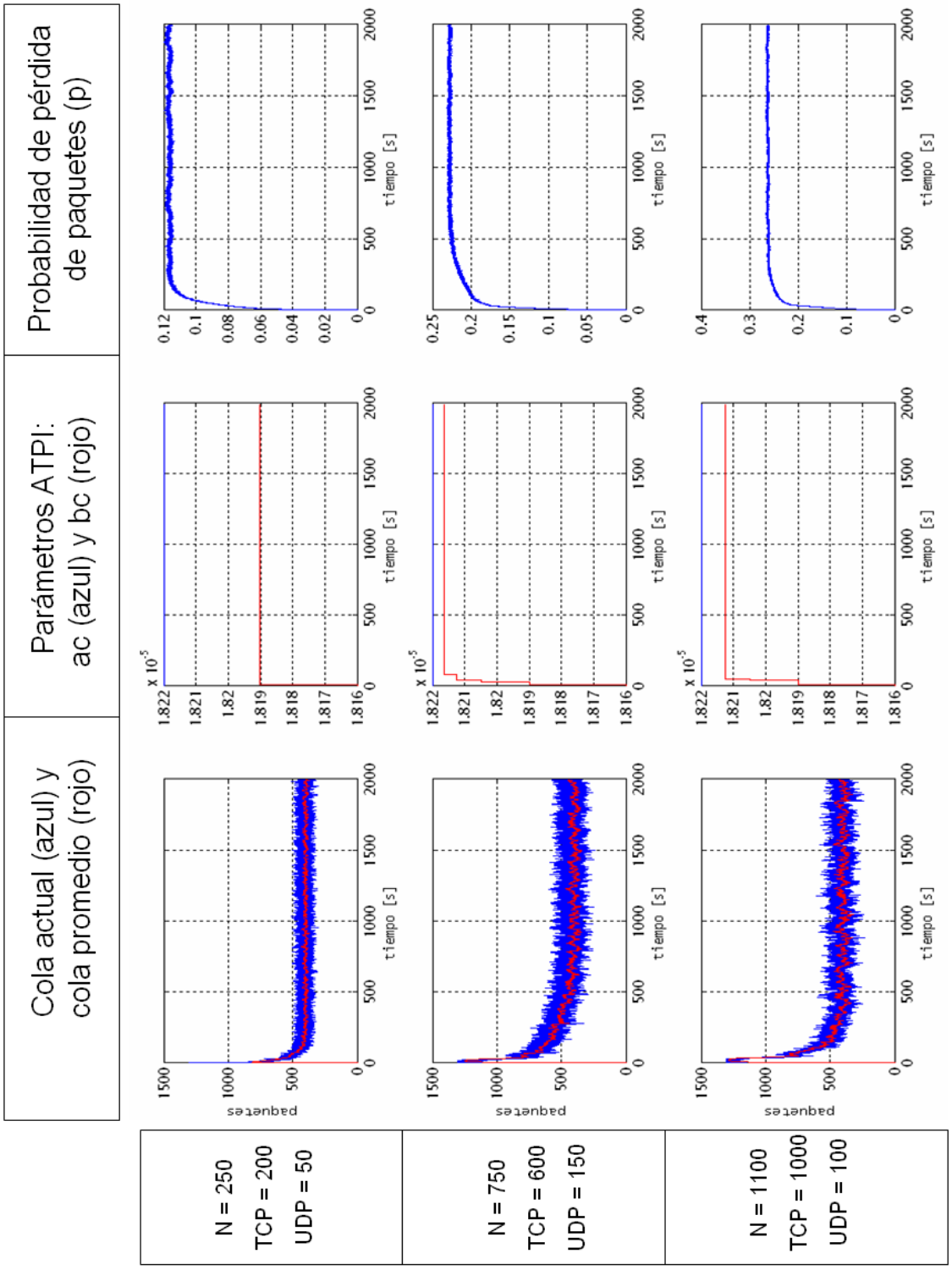

Figura 3.7: Experimento base con tráfico UDP adicional

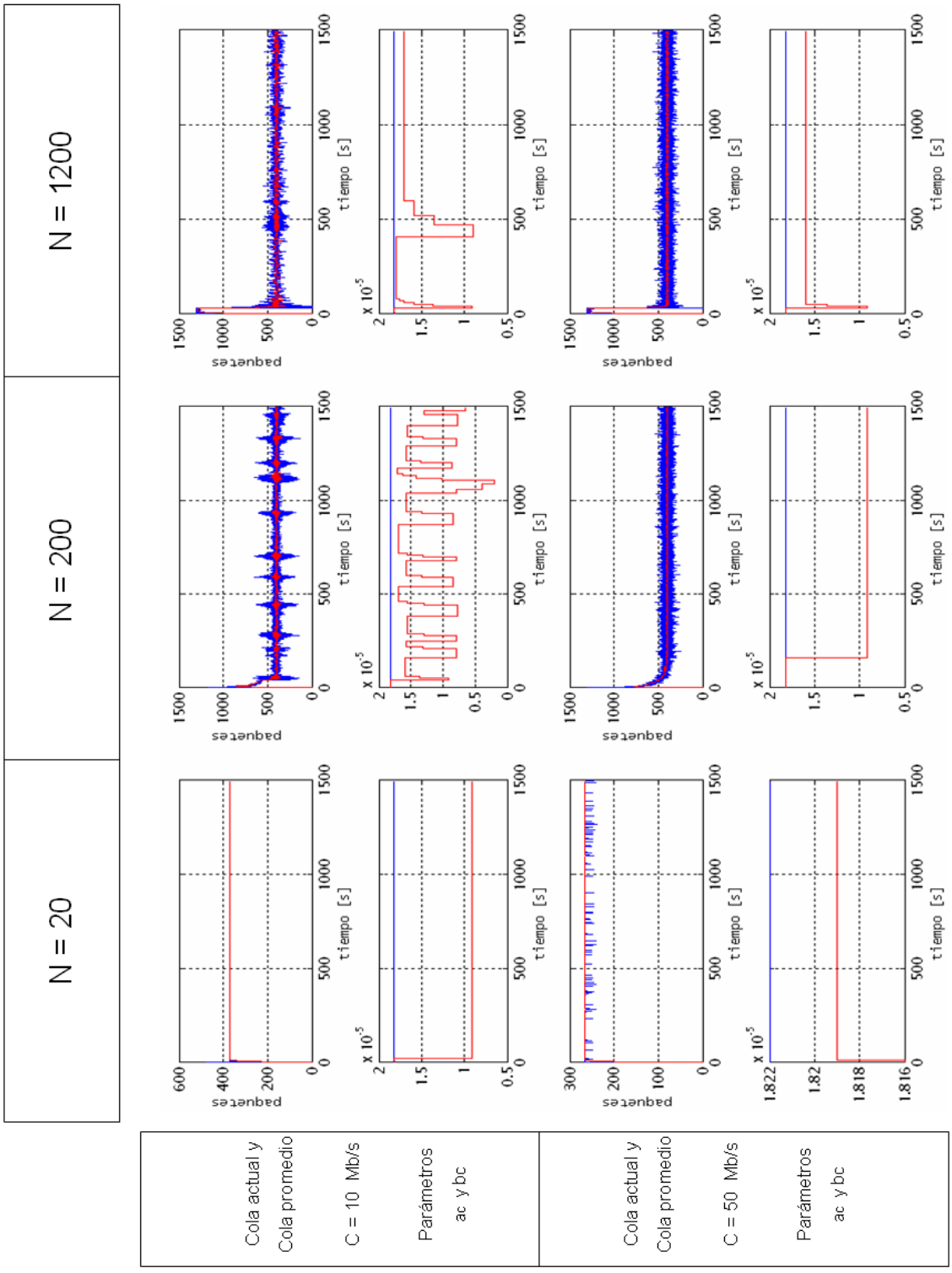

Figura 3.8: Variando N y C del experimento base

Variando N en el tiempo: Carrero (2006) realizó en su proyecto de grado una simulación de la dinámica de la cola y su respectiva señal de control p efectuando cambios en el número de fuentes N y en los valores de b_c durante el transcurso de la simulación. En el trabajo de Carrero (2006), el ajuste del parámetro b_c para la realización de esta prueba se hizo de manera manual a través del *script* de simulación.

Partiendo con $a_c = 1.822 \cdot 10^{-5}$ y $b_c = 1.542 \cdot 10^{-5}$ como valores iniciales del controlador (b_c muy alejado de a_c) se tiene $N = 50$ fuentes entre 0 y 40 segundos. En $t = 40$ segundos, se aumenta el número de fuentes a $N = 600$. Posteriormente, en $t = 80$ segundos, se disminuyó de manera manual y programada (no de forma automática) la diferencia entre b_c y a_c ($b_c = 1.819 \cdot 10^{-5}$) manteniendo el número de fuentes en $N = 600$ hasta $t = 150$ segundos, donde varía nuevamente el número de fuentes a $N = 25$, valor que se mantiene hasta finalizar la simulación en $t = 200$ segundos. Los resultados obtenidos de este experimento se muestran en las figuras 3.9 y 3.10.

En la Figura 3.11 se muestra los resultados de recrear experimento de Carrero (2006) pero realizando el ajuste de b_c de manera automática a través del algoritmo ATPI presentado en este trabajo. Se puede apreciar que los cambios del valor de b_c se ven reflejados en un ajuste en la dinámica de la cola para disminuir las oscilaciones y estabilizarla en su valor de referencia.

3.2.3 Topología estrella con más de un enrutador

En la Figura 3.12 se observa una red tipo estrella con $N = 100$ fuentes, 6 enrutadores y 1 destino. Las fuentes son TCP *NewReno* y hay tráfico cruzado UDP en todos enlaces enrutador-enrutador. Los enlaces de las fuentes TCP, las fuentes UDP y el destino son *full duplex*, con RTT's generados aleatoriamente por medio de una función uniforme, capacidad $C = 10$ Mb/s y las colas están regidas por el mecanismo *DropTail* de manejo de colas. Los enlaces entre enrutadores son igualmente *full duplex* y con capacidad $C = 10$ Mb/s pero poseen retardos de 10 ms. La cola está controlada mediante el ATPI.

En $t = 0$, hay 200 fuentes TCP y 250 fuentes UDP hasta llegar a $t = 200$ segundos,

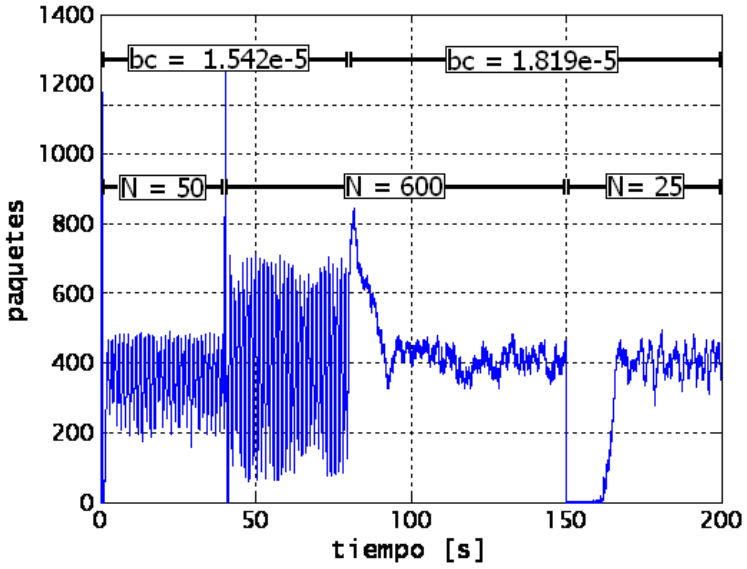

Figura 3.9: Comportamiento de la cola ante la variación de N y b_c en el tiempo. (Carrero 2006): cambio manual de b_c

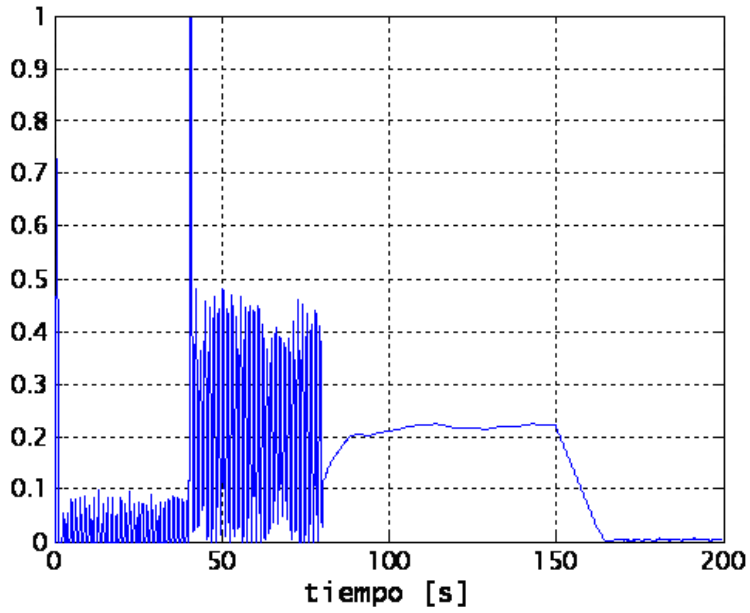

Figura 3.10: Comportamiento de p ante la variación de N y b_c en el tiempo. (Carrero 2006)

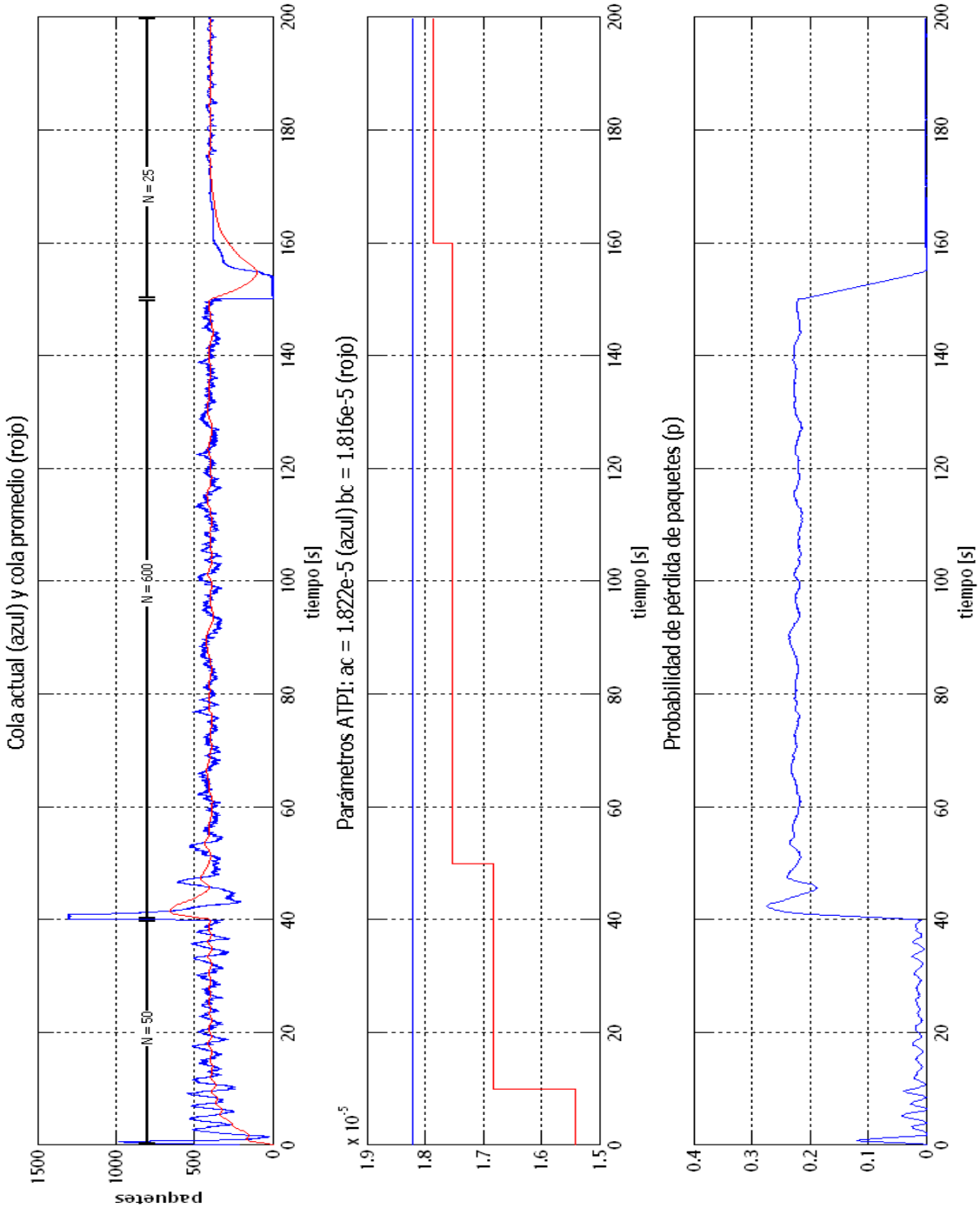

Figura 3.11: Variación de N en el tiempo: ajuste automático de b_c.

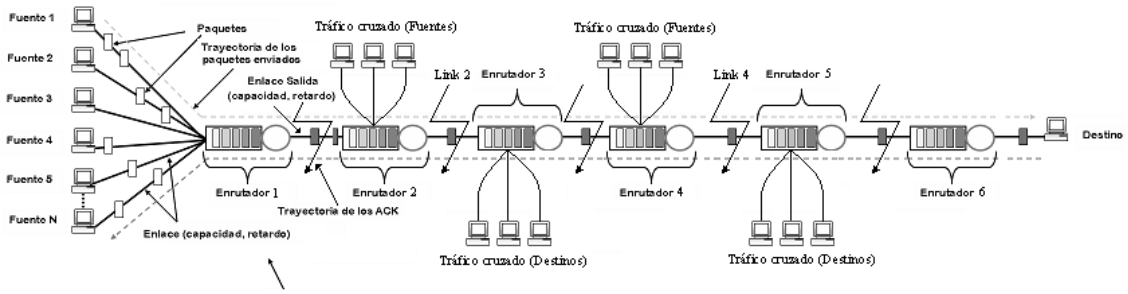

Figura 3.12: Configuración de red tipo estrella con más de un enrutador

donde se apagan 100 de las fuentes TCP que estaban en funcionamiento. Posteriormente, en $t = 400$ segundos, se agregan 200 fuentes TCP, que se mantienen encendidas junto con las primeras 100 fuentes TCP y las 250 fuentes UDP hasta la culminación de la simulación en $t = 600$ segundos. Los resultados obtenidos en este experimento pueden ser apreciados en la Figura 3.13, donde se muestra el comportamiento de las cinco colas monitoreadas, cada una junto con su respectiva gráfica de ajuste de ganancias.

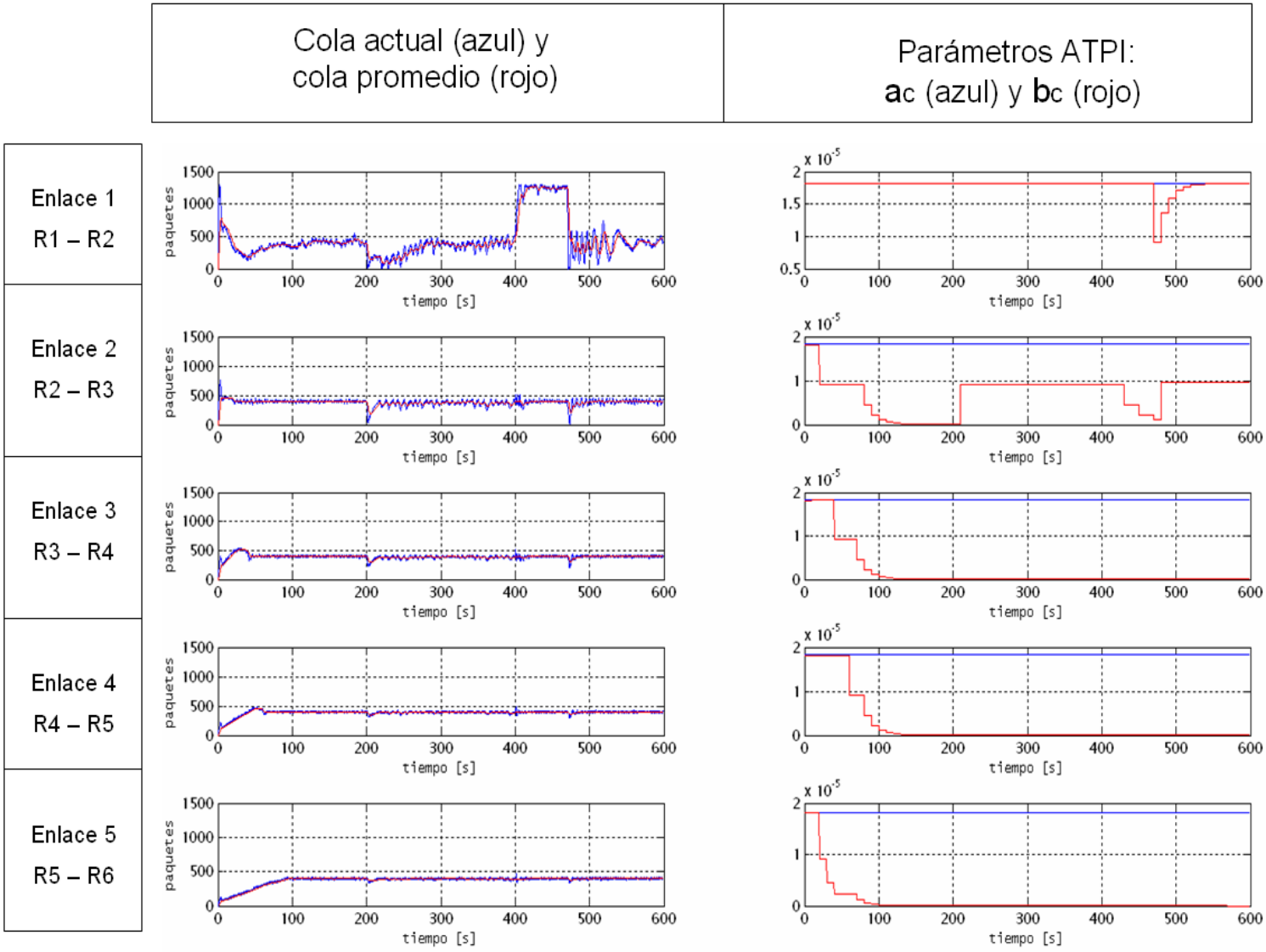

Figura 3.13: Simulación de una topología tipo estrella con 6 enrutadores y tráfico UDP

Conclusiones y recomendaciones

Se diseñó un algoritmo para ajustar automáticamente las ganancias del PI-AQM implementado en *ns-2* usando como base las reglas de ajuste planteadas por Carrero (2006). Este algoritmo denominado ATPI (*auto-tuning PI*) fue diseñado de manera heurística, midiendo la dispersión de la cola del enrutador y comparando el valor obtenido con un límite externo y un límite interno, ambos establecidos especialmente para el funcionamiento de este algoritmo. Si las oscilaciones sobrepasan el límite externo, los parámetros del controlador se acercan para así disminuir las oscilaciones. Por el contrario, si el valor de dispersión de la cola se encuentra por debajo del límite interno, la diferencia entre los valores se aumenta para que el controlador no trabaje de manera excesiva.

Se realizaron todas las pruebas que se consideraron necesarias para hacer el ajuste de las variables y de los parámetros que intervienen en el algoritmo y para probar el desempeño del mismo en presencia de diferentes cambios en la configuración de la red.

Además del algoritmo de ajuste automático que se presentó, este trabajo aportó una nueva regla de ajuste a las reglas ya existentes, la regla 2b, la cual permite un ajuste más preciso a la hora de la entonación, permitiendo un mejor desempeño del controlador.

El algoritmo aquí propuesto muestra como eliminar comportamientos indeseados en la cola del enrutador de una manera muy sencilla, aplicando las reglas de Carrero (2006) para el ajuste automático de los parámetros haciendo cálculos poco complicados y sin necesitar conocer los componentes ni la configuración de la red.

Los resultados derivados de este trabajo pueden ser de gran ayuda para el estudio de la prevención de congestión en redes y para plantear futuros algoritmos que automaticen y perfeccionen el funcionamiento de los diversos mecanismos AQM existentes.

Conceptos básicos

.1 Protocolo de control de transmisión TCP

En 1974 se presentó un protocolo llamado protocolo de control de transmisión (TCP, *transmission control protocol*) como una evolución de la tecnología de conmutación de paquetes, denominada ARPAnet. Desarrollado a finales de 1969 por el departamento de defensa americano a través de su agencia de proyectos de investigación avanzada (ARPA, *advanced research projects agency*), TCP, junto con el protocolo de Internet (IP, *Internet protocol*), tenían como función encargarse de la transmisión autónoma de datos entre ordenadores y redes locales de distinto origen.

El modelo básico de la estructura de una red de computadoras es el **modelo de referencia TCP/IP**, (véase la Figura 14). En términos generales, este modelo está organizado en cuatro capas conceptuales que se construyen sobre una quinta capa de hardware. Estas cuatro capas son: de aplicación, de transporte, Internet y de acceso a red. Dentro de estas capas se incluyen otros tipos de protocolos que tienen varios propósitos o funciones, todos ellos relacionados con la transferencia de información.

La tarea del IP es llevar los datos de un sitio a otro, es decir, establecer la ruta de

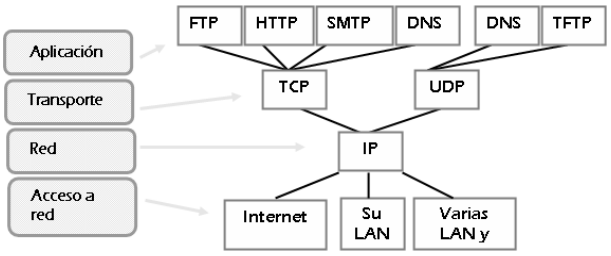

Figura 14: Modelo de referencia TCP/IP

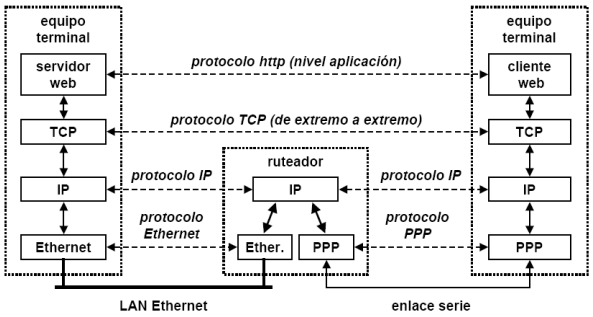

Figura 15: Conexión de "extremo a extremo" y de "punto a punto" del TCP

comunicación entre el origen y el destino y sus características fundamentales son:

- No orientado a conexión.
- Transmisión en unidades llamadas "paquetes".
- Sin corrección de errores, ni control de congestión.
- No garantiza la entrega en secuencia.

TCP es un protocolo de transferencia de datos orientado a conexión, que le ofrece a las aplicaciones un canal fiable, transparente, sin errores y en modo *full duplex*: para cada extremo de una conexión TCP, la conexión consta de dos enlaces lógicos, uno de salida y otro de entrada. Con la tecnología apropiada en el nivel de red, los datos pueden fluir simultáneamente en ambos sentidos. La cabecera TCP contiene tanto el número de secuencia de los datos de salida como el reconocimiento de los datos de entrada. TCP transporta una secuencia de bytes de "extremo a extremo"[1] (Saltzer et al. (1984)) y de "punto a punto" (véase Figura 15)[2].

TCP tiene como premisas principales hacer uso eficiente del ancho de banda (acoplando el rendimiento de la transmisión de paquetes a la disponibilidad de la misma) y hacer confiable la transmisión de paquetes enviados y la retransmisión de aquellos paquetes que se pierden en la red (for Beginners (2003)).

[1]El emisor y el receptor poseen un controlador de flujos. El controlador del emisor regula la cantidad de datos que se envían, mientras que, el del receptor, indica cuantos espacio se encuentra disponible en los *buffers*

[2]Tomada de D. Ros, "TCP: protocolo de control de transmisión en redes IP", Taller TCP ULA, Junio 2004 (lámina 7).

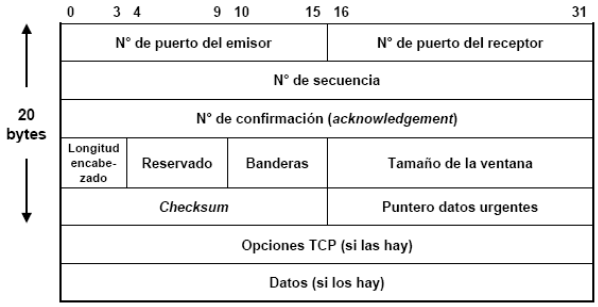

Figura 16: Cabecera del segmento TCP

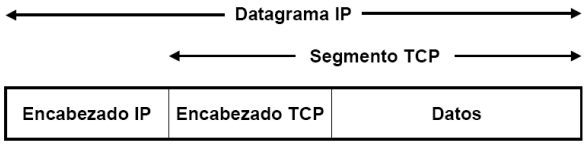

Figura 17: Datagrama IP

Para lograr su propósito, TCP divide los datos de la aplicación, antes de transmitirlos, en unidades de información llamadas "segmentos". A cada unidad de información se le asigna un número de secuencia con el fin de llevar un control de la llegada de los segmentos al destino. La confirmación de que un segmento ha salido de la red y ha llegado al destino se hace a través de los **acuses de recibo** (ACK, *acknowledgments*).

Los ACKs tienen como objetivos principales regularizar el ritmo de transmisión de TCP, asegurando que los paquetes puedan ser transmitidos solamente cuando otros paquetes han salido de la red, y fiabilizando la transmisión, dándole información a la fuente para que ella sepa si tiene que retransmitir algún paquete. (Altman (2001)). Cada segmento posee una cabecera con información útil para llevar a cabo la entrega de los datos y cada segmento TCP es encapsulado en un datagrama IP (figuras 16 y 17 respectivamente)[3].

[3]Tomado de D. Ros "TCP: protocolo de control de transmisión en redes IP" Taller TCP ULA. Junio,2004 (láminas 11 y 12)

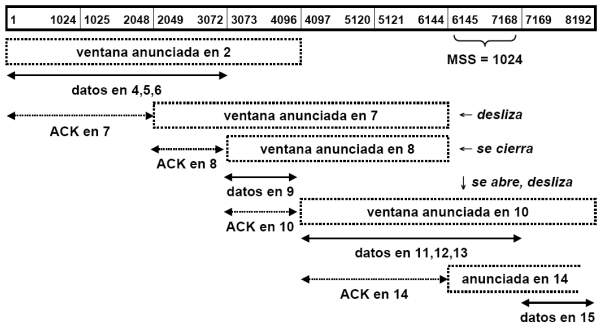

Figura 18: Ventana deslizante TCP

.2 Control de congestión

Los mecanismos de control de congestión fueron agregados en 1987, luego de haberse presentado varios casos de congestiones severas en Internet. En el comienzo de los años 80, Internet pasó de una pequeña red con servidores y enlaces similares, a una descomunal red de redes con servidores de distintas capacidades en redes con velocidades ampliamente diferentes. Inevitablemente los cuellos de botella (*bottlenecks*), creados por las disímilitudes de velocidad en los enlaces, conllevaron a la congestión y a la pérdida de paquetes (Noureddine & Tobagi (2002)).

Para controlar su velocidad de transmisión, TCP no permite a la fuente introducir en la red más que un número determinado de paquetes. Esto lo hace por medio de un mecanismo de control de flujo llamado "ventana deslizante" o "ventana de congestión" (cwnd, *congestion window*). Este mecanismo se basa en una ventana dinámica, cuyo valor limita la cantidad de datos que envía la fuente (swnd, *sending window*). El límite de swnd viene dado por el mínimo valor entre cwnd y la ventana máxima que puede aceptar el destino (rwnd, *receiver window*). El tamaño de la cwnd se negocia de forma dinámica durante la sesión TCP, es por esto que se le da el calificativo de "deslizante" (Figura 18).

El mecanismo de incremento y decremento del tamaño de la ventana se denomina **incremento aditivo, decremento multiplicativo** (AIMD, *additive increase, multiplicative decrease*).

El AIMD tiene como principio básico el crecimiento lineal de la ventana en ausencia de congestión, con el objeto de usar lo más eficazmente posible el ancho de banda, y

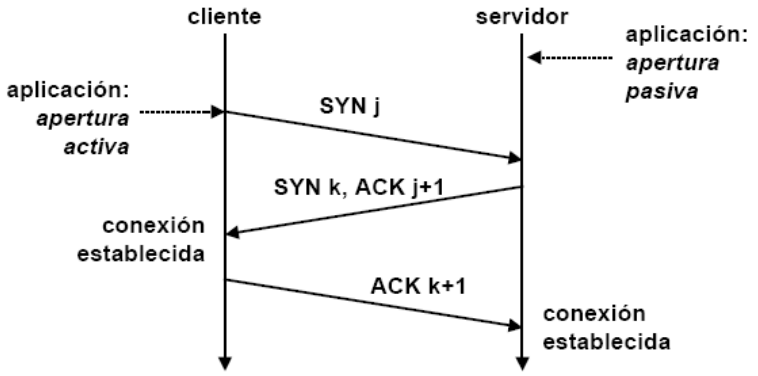

Figura 19: Tres segmentos para abrir conexión: *three-way handshake*

la disminución multiplicativa de la ventana si surge una congestión "no severa" (Floyd et al. (2000)). Se dice que éste algoritmo de control converge a un estado eficiente y justo, sin importar el valor del estado inicial (Chiu & Jain (1989)).

Las constantes que maneja este mecanismo de control de congestión son a y b, que representan los parámetros de incremento y decremento del TCP respectivamente (véase Marquez et al. (2004)). De manera que, después de un evento de pérdida, la ventana de congestión decrece de W a $(1-b)W$ paquetes, de lo contrario, la ventana de congestión se incrementará de W a $(W+a)$ paquetes cada RTT (Floyd et al. (2000)).

Para poder comenzar con todo el proceso de intercambio de datos se necesita que la fuente y el destino establezcan conexión. Para establecer dicha conexión, TCP utiliza un saludo de tres etapas (*three way handshake*) (Comer (1996)). (véase Figura 19).

Luego de haber establecido la conexión, TCP puede trabajar hasta con cuatro mecanismos de control de congestión que actúan conjuntamente:

- *slow start*: luego de establecer la conexión se comienza con esta etapa en la cual el valor de la ventana inicial es pequeño y va creciendo rápidamente, duplicando la ventana con cada ACK recibido. Si se detecta una pérdida (congestión) durante esta etapa la ventana cae drásticamente a su valor inicial (Noureddine & Tobagi (2002)).

- *congestion avoidance*: cuando la ventana alcanza un umbral "*slow-start threshold*" (que representa una estimación de la capacidad de la red), se comienza la

prevención de congestión (*congestion avoidance*). En esta fase, el crecimiento de la ventana de congestión se hace más lento (1/cwnd con cada ACK recibido), suponiendo que la capacidad de la red se ha alcanzado.(Noureddine & Tobagi (2002)).

- *fast Retransmit*[4]:
  - Hipótesis:
    - uno o dos ACKs duplicados → segmentos en desorden.
    - tres ACKs duplicados → un segmento perdido.
  - Idea: no esperar que se dispare el temporizador de retransmisión (RTO)
  - Recepción de tres ACKs duplicados = congestión no severa.
    - hay segmentos que, a pesar de todo, han llegado al receptor → no disminuir drásticamente la tasa de emisión → no pasar a *slow start*.
    - ACKs duplicados → mecanismo de detección de pérdidas.
- *fast recovery*: si se reciben tres ACKs duplicados consecutivos:
  - retransmitir inmediatamente el segmento faltante.
  - pasar a *congestion avoidance*.

[4]Tomado de D. Ros "TCP: protocolo de control de transmisión en redes IP" Taller TCP ULA. Junio, 2004 (láminas 28,29,31)

Experimentos en (*ns-2*)

A continuación se presentarán algunos de los scripts de *ns-2* usados para efectuar las simulaciones de este trabajo. Cualquier duda sobre el simulador de redes *ns-2* o acerca de los scripts de simulación véase (for Beginners 2003, Fall & Varadhan 2002) para más detalles.

.2.1 TCP/Newreno con ATPI y RTT iguales para todas las fuentes (Experimento base)

```
1   ############################
2   # ATPI14.tcl             #
3   # Mariemma Karolina Sosa #
4   ############################
5
6   # 200 fuentes comenzando al mismo tiempo (t = 0 s.)
7   # ac = 1.822e-5
8   # bc = 1.816e-5
9   # ssize = 10
10
11  set ns [new Simulator]
12
13  set dlyc 10ms
14  set dlyf 1ms
15  set C 30Mb
16  set qmax 1300
17  set tsimu 1500
18  set numf 200
19
20  # Abre los archivos de trazas
21  set pVstime [open proba14.tr w]
22  set qpVstime [open qprom14.tr w]
23  set qsize [open queuesize14.tr w]
24  set disp [open disp14.tr w]
```

```
25 set paramac [open acatpi14.tr w]
26 set parambc [open bcatpi14.tr w]
27
28 #PARAMETROS ATPI
29 #Queue/ATPI set bytes_ false
30 #Queue/ATPI set queue_in_bytes_ false
31 #Queue/ATPI set w_ 160
32 Queue/ATPI set qref_ 400
33 #Queue/ATPI set wq_ 0.002
34 #Queue/ATPI set alpha_ 0.5
35 #Queue/ATPI set minth_ 1.0
36 #Queue/ATPI set Maxth_ 5.0
37 #Queue/ATPI set lowlimit_ 0.5
38 #Queue/ATPI set ac_ 0.00001822
39 #Queue/ATPI set bc_ 0.00001816
40 #Queue/ATPI set ssize_ 10
41 #Queue/ATPI set mean_pktsize_ 500
42 #Queue/ATPI set setbit_ false
43 #Queue/ATPI set prob_ 0
44 #Queue/ATPI set qprom_ 0
45 #Queue/ATPI set disp_ 0
46 #Queue/ATPI set curq_ 0
47
48 # Crea los nodos y los enlaces cuello de botella
49 set n1 [$ns node]
50 set n2 [$ns node]
51 $ns duplex-link $n1 $n2 $C $dlyc ATPI
52
53 set cola [$ns link $n1 $n2]
54
55 # Tamaño máximo de la cola cuello de botella
56 $ns queue-limit $n1 $n2 $qmax
57
58 for {set j 1} {$j<=$numf} {incr j} {
59   set f($j) [$ns node]}
60
61 # Enlaces entre las fuentes y el cuello de botella
62 #Declaración de los links, capacidad del link y retardo
63 for {set j 1} {$j<=$numf} {incr j} {
64   $ns duplex-link $f($j) $n1 1000Mb $dlyf DropTail}
65
66 # Crea el agente de las fuentes TCP
67 for {set j 1} {$j<=$numf} {incr j} {
68   set tcpSender($j) [new Agent/TCP/Newreno]
69   $ns attach-agent $f($j) $tcpSender($j)}
70
71 # Crea el agente FTP para la fuente TCP
```

```
72  for {set j 1} {$j<=$numf} {incr j} {
73    set ftp($j) [new Application/FTP]
74    $ftp($j) attach-agent $tcpSender($j)
75    $ftp($j) set type_ FTP}
76
77  # Crea los agente destino TCP
78  for {set j 1} {$j<=$numf} {incr j} {
79    set tcpReceiver($j) [new Agent/TCPSink]
80    $ns attach-agent $n2 $tcpReceiver($j)
81    # Conecta los dos agentes TCP
82    $ns connect $tcpSender($j) $tcpReceiver($j)}
83
84  # Muestrea la cola cuello de botella cada 0.1 seg. y almacena la traza en qm.out
85  set qmon [$ns monitor-queue $n1 $n2 [open qm_pi2.out w] 0.1];
86  [$ns link $n1 $n2] queue-sample-timeout;
87
88  proc finish {} {
89      global qsize pVstime qpVstime disp paramac parambc
90      close $qsize
91      close $paramac
92      close $parambc
93      close $pVstime
94      close $qpVstime
95      close $disp
96      exit 0}
97
98  #Imprimir probabilidad de pérdida
99  proc plotProba {cola file} {
100 global ns
101 set time 0.001
102 set now [$ns now]
103 puts $file "$now [[$cola queue] set prob_]"
104 $ns at [expr $now+$time] "plotProba $cola $file" }
105
106 #Imprimir cola promedio
107 proc plotQprom {cola file} {
108 global ns
109 set time 0.001
110 set now [$ns now]
111 puts $file "$now [[$cola queue] set qprom_]"
112 $ns at [expr $now+$time] "plotQprom $cola $file" }
113
114 #Imprimir cola actual
115 proc record {} {
116 global ns qmon qsize n1 n2
117 set time 0.001
118 set now [$ns now]
```

```
119 #imprime en el archivo $qsize el tamaño actual de la cola
120 $qmon instvar pkts_
121 puts $qsize "$now [expr $pkts_]"
122 $ns at [expr $now+$time] "record"}
123
124 #Imprimir Dispersión
125 proc plotDisp {cola file} {
126 global ns
127 set time 0.001
128 set now [$ns now]
129 puts $file "$now [[$cola queue] set disp_]"
130 $ns at [expr $now+$time] "plotDisp $cola $file" }
131
132 #Imprimir parámetro ac del ATPI
133 proc plotac {cola file} {
134 global ns
135 set time 0.001
136 set now [$ns now]
137 puts $file "$now [[$cola queue] set ac_]"
138 $ns at [expr $now+$time] "plotac $cola $file" }
139
140 #Imprimir parámetro bc del ATPI
141 proc plotbc {cola file} {
142 global ns
143 set time 0.001
144 set now [$ns now]
145 puts $file "$now [[$cola queue] set bc_]"
146 $ns at [expr $now+$time] "plotbc $cola $file" }
147
148 # Correr la simulación
149 $ns at 0.1 "record"
150
151 # Programación de eventos para los agentes FTP
152 for {set j 1} {$j<=$numf} {incr j} {
153   $ns at 0.00 "$ftp($j) start"}
154
155 $ns at $tsimu "finish"
156 $ns at 0.0 "plotProba $cola $pVstime"
157 $ns at 0.0 "plotQprom $cola $qpVstime"
158 $ns at 0.0 "plotDisp $cola $disp"
159 $ns at 0.0 "plotac $cola $paramac"
160 $ns at 0.0 "plotbc $cola $parambc"
161
162 $ns run
```

.2.2 TCP/Newreno con ATPI, RTT iguales para todas las fuentes y tráfico UDP

```
1  ############################
2  # TrafUDP_ATPI2.tcl      #
3  # Mariemma Karolina Sosa #
4  ############################
5
6  # Fuentes TCP con trafico UDP
7  # 200 fuentes comenzando al mismo tiempo (t = 0.00sg)
8  # ac = 1.822e-5
9  # bc = 1.816e-5
10
11 set ns [new Simulator]
12
13 set dlyc 10ms
14 set dlyf 1ms
15 set C 30Mb
16 set qmax 1300
17 set tsimu 2000
18 set numtcpF 600
19 set numudpF 150
20
21 # Abre los archivos de trazas
22 set pVstime [open probaudp2.tr w]
23 set qpVstime [open qpromudp2.tr w]
24 set qsize [open queuesizeudp2.tr w]
25 set disp [open dispudp2.tr w]
26 set paramac [open acatpiudp2.tr w]
27 set parambc [open bcatpiudp2.tr w]
28
29 #PARAMETROS ATPI
30 #Queue/ATPI set bytes_ false
31 #Queue/ATPI set queue_in_bytes_ false
32 #Queue/ATPI set w_ 160
33 Queue/ATPI set qref_ 400
34 #Queue/ATPI set wq_ 0.002
35 #Queue/ATPI set alpha_ 0.5
36 #Queue/ATPI set minth_ 1.0
37 #Queue/ATPI set Maxth_ 5.0
38 #Queue/ATPI set lowlimit_ 0.5
39 #Queue/ATPI set ac_ 0.00001822
40 #Queue/ATPI set bc_ 0.00001816
41 #Queue/ATPI set ssize_ 10
42 #Queue/ATPI set mean_pktsize_ 500
43 #Queue/ATPI set setbit_ false
44 #Queue/ATPI set prob_ 0
```

```
45 #Queue/ATPI set qprom_ 0
46 #Queue/ATPI set disp_ 0
47 #Queue/ATPI set curq_ 0
48
49 # Crea los nodos y los enlaces cuello de botella
50 set n1 [$ns node]
51 set n2 [$ns node]
52 $ns duplex-link $n1 $n2 $C $dlyc ATPI
53
54 set cola [$ns link $n1 $n2]
55
56 # Tamaño maximo de la cola cuello de botella
57 $ns queue-limit $n1 $n2 $qmax
58
59 # Crea los nodos fuente y destino TCP
60 for {set j 1} {$j<=$numtcpF} {incr j} {
61   set ftcp($j) [$ns node]}
62
63 # Crea los nodos fuente y destino UDP
64 for {set j 1} {$j<=$numudpF} {incr j} {
65   set fudp($j) [$ns node]}
66
67 # Enlaces entre las fuentes TCP y el cuello de botella
68 # Declaración de los links, capacidad del link y retardo
69 for {set j 1} {$j<=$numtcpF} {incr j} {
70   $ns duplex-link $ftcp($j) $n1 1000Mb $dlyf DropTail}
71
72 # Enlaces entre las fuentes UDP y el cuello de botella
73 # Declaración de los links, capacidad del link y retardo
74 for {set j 1} {$j<=$numudpF} {incr j} {
75   $ns duplex-link $fudp($j) $n1 1000Mb $dlyf DropTail}
76
77 # Crea el agente de las fuentes TCP
78 for {set j 1} {$j<=$numtcpF} {incr j} {
79   set tcpSender($j) [new Agent/TCP/Newreno]
80   $ns attach-agent $ftcp($j) $tcpSender($j)}
81
82 # Crea el agente FTP para la fuente TCP
83 for {set j 1} {$j<=$numtcpF} {incr j} {
84   set ftp($j) [new Application/FTP]
85   $ftp($j) attach-agent $tcpSender($j)
86   $ftp($j) set packetSize_ 1024
87   $ftp($j) set type_ FTP}
88
89 # Crea los agente destino TCP
90 for {set j 1} {$j<=$numtcpF} {incr j} {
91   set tcpReceiver($j) [new Agent/TCPSink]
```

```
92    $ns attach-agent $n2 $tcpReceiver($j)
93    # Conecta los dos agentes TCP
94    $ns connect $tcpSender($j) $tcpReceiver($j)}
95
96  # Crea el agente de las fuentes UDP
97  for {set j 1} {$j<=$numudpF} {incr j} {
98    set udpSender($j) [new Agent/UDP]
99    $ns attach-agent $fudp($j) $udpSender($j)}
100
101 # Crea los nodos destinos UDP
102 for {set j 1} {$j<=$numudpF} {incr j} {
103   set null($j) [new Agent/Null]
104   $ns  attach-agent $n2 $null($j)
105   # Conecta los dos agentes UDP
106   $ns connect $udpSender($j) $null($j)}
107
108 # Crea el agente CBR para las fuentes UDP
109 for {set j 1} {$j<=$numudpF} {incr j} {
110   set cbr($j) [new Application/Traffic/CBR]
111   $cbr($j) attach-agent $udpSender($j)
112   $cbr($j) set rate_ 0.01Mb
113   $cbr($j) set packetSize_ 552
114   $cbr($j) set random_ false}
115
116 # muestrea la cola cuello de botella cada 0.1 seg. y almacena la traza en qm.out
117 set qmon [$ns monitor-queue $n1 $n2 [open qm_pi2.out w] 0.1];
118 [$ns link $n1 $n2] queue-sample-timeout;
119
120 proc finish {} {
121     global qsize pVstime qpVstime disp paramac parambc
122     close $qsize
123     close $paramac
124     close $parambc
125     close $pVstime
126     close $qpVstime
127     close $disp
128     exit 0}
129
130 #Imprimir probabilidad de pérdida
131 proc plotProba {cola file} {
132 global ns
133 set time 0.001
134 set now [$ns now]
135 puts $file "$now [[$cola queue] set prob_]"
136 $ns at [expr $now+$time] "plotProba $cola $file" }
137
138 #Imprimir cola promedio
```

```
139 proc plotQprom {cola file} {
140 global ns
141 set time 0.001
142 set now [$ns now]
143 puts $file "$now [[$cola queue] set qprom_]"
144 $ns at [expr $now+$time] "plotQprom $cola $file" }
145
146 #Imprimir cola actual
147 proc record {} {
148 global ns qmon qsize n1 n2
149 set time 0.001
150 set now [$ns now]
151 #imprime en el archivo $qsize el tamaño actual de la cola
152 $qmon instvar pkts_
153 puts $qsize "$now [expr $pkts_]"
154 $ns at [expr $now+$time] "record"}
155
156 #Imprimir Dispersión
157 proc plotDisp {cola file} {
158 global ns
159 set time 0.001
160 set now [$ns now]
161 puts $file "$now [[$cola queue] set disp_]"
162 $ns at [expr $now+$time] "plotDisp $cola $file" }
163
164 #Imprimir parámetro ac del ATPI
165 proc plotac {cola file} {
166 global ns
167 set time 0.001
168 set now [$ns now]
169 puts $file "$now [[$cola queue] set ac_]"
170 $ns at [expr $now+$time] "plotac $cola $file" }
171
172 #Imprimir parámetro bc del ATPI
173 proc plotbc {cola file} {
174 global ns
175 set time 0.001
176 set now [$ns now]
177 puts $file "$now [[$cola queue] set bc_]"
178 $ns at [expr $now+$time] "plotbc $cola $file" }
179
180 # Correr la simulación
181 $ns at 0.1 "record"
182
183 # Programación de eventos para los agentes FTP
184 for {set j 1} {$j<=$numtcpF} {incr j} {
185   $ns at 0.00 "$ftp($j) start"}
```

```
186
187 # Programación de eventos para los agentes CBR
188 for {set j 1} {$j<=$numudpF} {incr j} {
189   $ns at 0.01 "$cbr($j) start"}
190
191 $ns at $tsimu "finish"
192 $ns at 0.0 "plotProba $cola $pVstime"
193 $ns at 0.0 "plotQprom $cola $qpVstime"
194 $ns at 0.0 "plotDisp $cola $disp"
195 $ns at 0.0 "plotac $cola $paramac"
196 $ns at 0.0 "plotbc $cola $parambc"
197
198 $ns run
```

.2.3 TCP/Newreno con ATPI, RTT iguales para todas las fuentes y variación de N en el tiempo

```
1   ############################
2   # VarNent.tcl            #
3   # Mariemma Karolina Sosa #
4   ############################
5
6   # De t=0 s. a t=40 s., N=50 fuentes
7   #En t=40 s. se aumenta N=600 fuentes hasta t=150 s.
8   #donde se disminuyen las fuentes a 25 hasta finalizar la simulacion en t=200 s.
9
10  # ac = 1.822e-5
11  # bc = 1.816e-5
12  # ssize = 10
13
14  set ns [new Simulator]
15
16  set dlyc 10ms
17  set dlyf 1ms
18  set C 30Mb
19  set qmax 1300
20  set tsimu 200
21  set numf 550
22
23  # Abre los archivos de trazas
24  set pVstime [open probaVarNent.tr w]
25  set qpVstime [open qpromVarNent.tr w]
26  set qsize [open queuesizeVarNent.tr w]
27  set disp [open dispVarNent.tr w]
28  set paramac [open acatpiVarNent.tr w]
29  set parambc [open bcatpiVarNent.tr w]
```

```
30
31 #PARAMETROS ATPI
32 #Queue/ATPI set bytes_ false
33 #Queue/ATPI set queue_in_bytes_ false
34 Queue/ATPI set w_ 160
35 Queue/ATPI set qref_ 400
36 #Queue/ATPI set wq_ 0.002
37 #Queue/ATPI set alpha_ 0.5
38 #Queue/ATPI set minth_ 1.0
39 #Queue/ATPI set Maxth_ 5.0
40 #Queue/ATPI set lowlimit_ 30.0
41 #Queue/ATPI set ac_ 0.00001822
42 #Queue/ATPI set bc_ 0.00001816
43 #Queue/ATPI set ssize_ 10
44 #Queue/ATPI set mean_pktsize_ 500
45 #Queue/ATPI set setbit_ false
46 #Queue/ATPI set prob_ 0
47 #Queue/ATPI set qprom_ 0
48 #Queue/ATPI set disp_ 0
49 #Queue/ATPI set curq_ 0
50
51 # Crea los nodos y los enlaces cuello de botella
52 set n1 [$ns node]
53 set n2 [$ns node]
54 $ns duplex-link $n1 $n2 $C $dlyc ATPI
55
56 set cola [$ns link $n1 $n2]
57
58 # Tamaño máximo de la cola cuello de botella
59 $ns queue-limit $n1 $n2 $qmax
60
61 for {set j 1} {$j<=$numf} {incr j} {
62   set f1($j) [$ns node]
63   set f2($j) [$ns node]
64   set f3($j) [$ns node]}
65
66 # Enlaces entre las fuentes y el cuello de botella
67 #Declaración de los links, capacidad del link y retardo
68 for {set j 1} {$j<=$numf} {incr j} {
69   $ns duplex-link $f1($j) $n1 1000Mb $dlyf DropTail
70   $ns duplex-link $f2($j) $n1 1000Mb $dlyf DropTail
71   $ns duplex-link $f3($j) $n1 1000Mb $dlyf DropTail}
72
73 # Crea el agente de las fuentes TCP
74 for {set j 1} {$j<=$numf} {incr j} {
75   set tcpSender1($j) [new Agent/TCP/Newreno]
76   set tcpSender2($j) [new Agent/TCP/Newreno]
```

```
77    set tcpSender3($j) [new Agent/TCP/Newreno]
78    $ns attach-agent $f1($j) $tcpSender1($j)
79    $ns attach-agent $f2($j) $tcpSender2($j)
80    $ns attach-agent $f3($j) $tcpSender3($j)}
81
82  # Crea el agente FTP para la fuente TCP
83  for {set j 1} {$j<=$numf} {incr j} {
84    set ftp1($j) [new Application/FTP]
85    set ftp2($j) [new Application/FTP]
86    set ftp3($j) [new Application/FTP]
87    $ftp1($j) attach-agent $tcpSender1($j)
88    $ftp2($j) attach-agent $tcpSender2($j)
89    $ftp3($j) attach-agent $tcpSender3($j)
90    $ftp1($j) set type_ FTP
91    $ftp2($j) set type_ FTP
92    $ftp3($j) set type_ FTP}
93
94  # Crea los agente destino TCP
95  for {set j 1} {$j<=$numf} {incr j} {
96    set tcpReceiver1($j) [new Agent/TCPSink]
97    set tcpReceiver2($j) [new Agent/TCPSink]
98    set tcpReceiver3($j) [new Agent/TCPSink]
99    $ns attach-agent $n2 $tcpReceiver1($j)
100   $ns attach-agent $n2 $tcpReceiver2($j)
101   $ns attach-agent $n2 $tcpReceiver3($j)
102   # Conecta los dos agentes TCP
103   $ns connect $tcpSender1($j) $tcpReceiver1($j)
104   $ns connect $tcpSender2($j) $tcpReceiver2($j)
105   $ns connect $tcpSender3($j) $tcpReceiver3($j)}
106
107 # muestrea la cola cuello de botella cada 0.1 seg. y almacena la traza en qm.out
108 set qmon [$ns monitor-queue $n1 $n2 [open qm_pi2.out w] 0.1];
109 [$ns link $n1 $n2] queue-sample-timeout;
110
111 proc finish {} {
112     global qsize pVstime qpVstime disp paramac parambc
113     close $qsize
114     close $paramac
115     close $parambc
116     close $pVstime
117     close $qpVstime
118     close $disp
119     exit 0}
120
121 #Imprimir probabilidad de pérdida
122 proc plotProba {cola file} {
123 global ns
```

```
124 set time 0.001
125 set now [$ns now]
126 puts $file "$now [[$cola queue] set prob_]"
127 $ns at [expr $now+$time] "plotProba $cola $file" }
128
129 #Imprimir cola promedio
130 proc plotQprom {cola file} {
131 global ns
132 set time 0.001
133 set now [$ns now]
134 puts $file "$now [[$cola queue] set qprom_]"
135 $ns at [expr $now+$time] "plotQprom $cola $file" }
136
137 #Imprimir cola actual
138 proc record {} {
139 global ns qmon qsize n1 n2
140 set time 0.001
141 set now [$ns now]
142
143 #imprime en el archivo $qsize el tamaño actual de la cola
144 $qmon instvar pkts_
145 puts $qsize "$now [expr $pkts_]"
146 $ns at [expr $now+$time] "record"}
147
148 #Imprimir Dispersión
149 proc plotDisp {cola file} {
150 global ns
151 set time 0.001
152 set now [$ns now]
153 puts $file "$now [[$cola queue] set disp_]"
154 $ns at [expr $now+$time] "plotDisp $cola $file" }
155
156 #Imprimir parámetro ac del ATPI
157 proc plotac {cola file} {
158 global ns
159 set time 0.001
160 set now [$ns now]
161 puts $file "$now [[$cola queue] set ac_]"
162 $ns at [expr $now+$time] "plotac $cola $file" }
163
164 #Imprimir parámetro bc del ATPI
165 proc plotbc {cola file} {
166 global ns
167 set time 0.001
168 set now [$ns now]
169 puts $file "$now [[$cola queue] set bc_]"
170 $ns at [expr $now+$time] "plotbc $cola $file" }
```

```
171
172 # Correr la simulación
173 $ns at 0.1 "record"
174
175 # Programación de eventos para los agentes FTP
176 for {set j 1} {$j<=25} {incr j} {
177   $ns at 0.00 "$ftp1($j) start"
178   $ns at 0.00 "$ftp2($j) start"
179   $ns at 150.00 "$ftp1($j) stop"}
180
181 for {set j 1} {$j<=$numf} {incr j} {
182   $ns at 40.00 "$ftp3($j) start"
183   $ns at 150.00 "$ftp3($j) stop"}
184
185 $ns at $tsimu "finish"
186 $ns at 0.0 "plotProba $cola $pVstime"
187 $ns at 0.0 "plotQprom $cola $qpVstime"
188 $ns at 0.0 "plotDisp $cola $disp"
189 $ns at 0.0 "plotac $cola $paramac"
190 $ns at 0.0 "plotbc $cola $parambc"
191
192 $ns run
```

.2.4 Topología tipo estrella con seis enrutadores, fuentes TCP/Newreno, tráfico cruzado UDP y un destino

```
1   ##########################
2   #redcompleta1.tcl       #
3   #Mariemma Karolina Sosa #
4   ##########################
5
6   #El ancho de banda de todos los enlaces es 10 Mbps, el retardo en los enlaces
7   #de los enrutadores es de 10ms, el retardo en los enlaces de las fuentes TCP,
8   #fuentes UDP y el destino estan definidos por una uniforme de 5 a 25 ms.
9   #Se genera trafico CBR como tráfico cruzado en los enlaces entre enrutadores.
10  #La dinamica del trafico TCP es la siguiente: inicialmente en t=0 s., comienzan 200 fuentes,
11  #luego, 100 fuentes caen abruptamente en t=200 s. En t=400 s., otras 200 fuentes TCP comienzan.
12  #El tiempo total de simulacion es de 600 s.
13
14  set ns [new Simulator]
15
16  #Número de fuentes y duración de la simulación
17  set NumbSrc 100
18  set Router 6
19  set Duration 600
20  set NumbUDP 50
```

```
21
22 #Archivos de trazas del tamaño de la cola, cola promedio y la probabilidad de pérdida
23 set pVstime1 [open proba1.tr w]
24 set pVstime2 [open proba2.tr w]
25 set pVstime3 [open proba3.tr w]
26 set pVstime4 [open proba4.tr w]
27 set pVstime5 [open proba5.tr w]
28 set qpVstime1 [open qprom1.tr w]
29 set qpVstime2 [open qprom2.tr w]
30 set qpVstime3 [open qprom3.tr w]
31 set qpVstime4 [open qprom4.tr w]
32 set qpVstime5 [open qprom5.tr w]
33 set qsize1 [open queuesize1.tr w]
34 set qsize2 [open queuesize2.tr w]
35 set qsize3 [open queuesize3.tr w]
36 set qsize4 [open queuesize4.tr w]
37 set qsize5 [open queuesize5.tr w]
38 set disp1 [open disp1.tr w]
39 set disp2 [open disp2.tr w]
40 set disp3 [open disp3.tr w]
41 set disp4 [open disp4.tr w]
42 set disp5 [open disp5.tr w]
43 set paramac1 [open acatpi1.tr w]
44 set paramac2 [open acatpi2.tr w]
45 set paramac3 [open acatpi3.tr w]
46 set paramac4 [open acatpi4.tr w]
47 set paramac5 [open acatpi5.tr w]
48 set parambc1 [open bcatpi1.tr w]
49 set parambc2 [open bcatpi2.tr w]
50 set parambc3 [open bcatpi3.tr w]
51 set parambc4 [open bcatpi4.tr w]
52 set parambc5 [open bcatpi5.tr w]
53
54 # Estructura de la red
55
56 #Nodos fuentes
57 for {set j 1} {$j<=$NumbSrc} {incr j} {
58     set S1($j) [$ns node]
59     set S2($j) [$ns node]
60     set S3($j) [$ns node]}
61
62 #Nodos Destinos
63 set D [$ns node]
64
65 #Cuellos de botella
66 set R1 [$ns node]
67 set R2 [$ns node]
```

```
68  set R3 [$ns node]
69  set R4 [$ns node]
70  set R5 [$ns node]
71  set R6 [$ns node]
72
73  #Crea los nodos fuentes y destinos UDP
74  for {set j 1} {$j<=$NumbUDP} {incr j} {
75      set fudp1($j) [$ns node]
76      set fudp2($j) [$ns node]
77      set fudp3($j) [$ns node]
78      set fudp4($j) [$ns node]
79      set fudp5($j) [$ns node]}
80
81  #Crea los nodos destinos UDP
82  for {set j 1} {$j<=$NumbUDP} {incr j} {
83      set null1($j) [new Agent/Null]
84      set null2($j) [new Agent/Null]
85      set null3($j) [new Agent/Null]
86      set null4($j) [new Agent/Null]
87      set null5($j) [new Agent/Null]
88      $ns  attach-agent $R2 $null1($j)
89      $ns  attach-agent $R3 $null2($j)
90      $ns  attach-agent $R4 $null3($j)
91      $ns  attach-agent $R5 $null4($j)
92      $ns  attach-agent $R6 $null5($j)}
93
94  #PARAMETROS ATPI
95  #Queue/ATPI set bytes_ false
96  #Queue/ATPI set queue_in_bytes_ false
97  Queue/ATPI set w_ 160
98  Queue/ATPI set qref_ 400
99  #Queue/ATPI set wq_ 0.002
100 #Queue/ATPI set alpha_ 0.5
101 #Queue/ATPI set minth_ 1.0
102 #Queue/ATPI set Maxth_ 5.0
103 #Queue/ATPI set ac_ 0.00001822
104 #Queue/ATPI set bc_ 0.00001816
105 #Queue/ATPI set ssize_ 10
106 Queue/ATPI set mean_pktsize_ 1000
107 #Queue/ATPI set setbit_ false
108 #Queue/ATPI set prob_ 0
109 #Queue/ATPI set qprom_ 0
110 #Queue/ATPI set disp_ 0
111 #Queue/ATPI set curq_ 0
112 $ns duplex-link $R1 $R2 10Mb 10ms ATPI
113 $ns duplex-link $R2 $R3 10Mb 10ms ATPI
114 $ns duplex-link $R3 $R4 10Mb 10ms ATPI
```

```
115 $ns duplex-link $R4 $R5 10Mb 10ms ATPI
116 $ns duplex-link $R5 $R6 10Mb 10ms ATPI
117 $ns queue-limit $R1 $R2 1300
118 $ns queue-limit $R2 $R3 1300
119 $ns queue-limit $R3 $R4 1300
120 $ns queue-limit $R4 $R5 1300
121 $ns queue-limit $R5 $R6 1300
122
123 set dly [new RNG]
124
125 #Enlaces entre las fuentes UDP y el cuello de botella
126 #Declaración de los links, capacidad del link y retardo
127 for {set j 1} {$j<=$NumbUDP} {incr j} {
128     $ns duplex-link $fudp1($j) $R1 10Mb [$dly uniform 5 25]ms DropTail
129     $ns duplex-link $fudp2($j) $R2 10Mb [$dly uniform 5 25]ms DropTail
130     $ns duplex-link $fudp3($j) $R3 10Mb [$dly uniform 5 25]ms DropTail
131     $ns duplex-link $fudp4($j) $R4 10Mb [$dly uniform 5 25]ms DropTail
132     $ns duplex-link $fudp5($j) $R5 10Mb [$dly uniform 5 25]ms DropTail}
133
134 #DRS
135 set cola1 [$ns link $R1 $R2]
136 set cola2 [$ns link $R2 $R3]
137 set cola3 [$ns link $R3 $R4]
138 set cola4 [$ns link $R4 $R5]
139 set cola5 [$ns link $R5 $R6]
140
141 #Enlaces entre las fuentes y el cuello de botella
142 for {set j 1} {$j<=$NumbSrc} {incr j} {
143     $ns duplex-link $S1($j) $R1 10Mb [$dly uniform 5 25]ms DropTail
144     $ns duplex-link $S2($j) $R1 10Mb [$dly uniform 5 25]ms DropTail
145     $ns duplex-link $S3($j) $R1 10Mb [$dly uniform 5 25]ms DropTail}
146
147 #Enlaces entre los cuellos de botella y los destinos
148 $ns duplex-link $R6 $D 10Mb [$dly uniform 5 25]ms DropTail
149
150 #Monitoreo de la cola del cuello de botella cada 0.05 seg. guardando la traza en qm.out
151 set qmon1 [$ns monitor-queue $R1 $R2 [open qm1.out w] 0.05]
152 [$ns link $R1 $R2] queue-sample-timeout;
153 set qmon2 [$ns monitor-queue $R2 $R3 [open qm2.out w] 0.05]
154 [$ns link $R2 $R3] queue-sample-timeout;
155 set qmon3 [$ns monitor-queue $R3 $R4 [open qm3.out w] 0.05]
156 [$ns link $R3 $R4] queue-sample-timeout;
157 set qmon4 [$ns monitor-queue $R4 $R5 [open qm4.out w] 0.05]
158 [$ns link $R4 $R5] queue-sample-timeout;
159 set qmon5 [$ns monitor-queue $R5 $R6 [open qm5.out w] 0.05]
160 [$ns link $R5 $R6] queue-sample-timeout;
161
```

```
162 proc finish {} {
163     global ns pVstime1 pVstime2 pVstime3 pVstime4 pVstime5 qpVstime1 qpVstime2 qpVstime3
164     qpVstime4 qpVstime5 qsize1 qsize2 qsize3 qsize4 qsize5 disp1 disp2 disp3 disp4 disp5 paramac1
165         paramac2 paramac3 paramac4 paramac5 parambc1 parambc2 parambc3 parambc4 parambc5
166     $ns flush-trace
167     close $pVstime1
168     close $pVstime2
169     close $pVstime3
170     close $pVstime4
171     close $pVstime5
172     close $qpVstime1
173     close $qpVstime2
174     close $qpVstime3
175     close $qpVstime4
176     close $qpVstime5
177     close $qsize1
178     close $qsize2
179     close $qsize3
180     close $qsize4
181     close $qsize5
182     close $disp1
183     close $disp2
184     close $disp3
185     close $disp4
186     close $disp5
187     close $paramac1
188     close $paramac2
189     close $paramac3
190     close $paramac4
191     close $paramac5
192     close $parambc1
193     close $parambc2
194     close $parambc3
195         close $parambc4
196     close $parambc5
197     exit 0}
198
199 #Fuentes y Destinos TCP
200 for {set j 1} {$j<=$NumbSrc} {incr j} {
201     set tcp_src1($j) [new Agent/TCP/Newreno]
202     set tcp_src2($j) [new Agent/TCP/Newreno]
203     set tcp_src3($j) [new Agent/TCP/Newreno]
204     set tcp_snk1($j) [new Agent/TCPSink]
205     set tcp_snk2($j) [new Agent/TCPSink]
206     set tcp_snk3($j) [new Agent/TCPSink]}
207
208 # Crea el agente de las fuentes UDP
```

```
209 for {set j 1} {$j<=$NumbUDP} {incr j} {
210     set udpSender1($j) [new Agent/UDP]
211     set udpSender2($j) [new Agent/UDP]
212     set udpSender3($j) [new Agent/UDP]
213     set udpSender4($j) [new Agent/UDP]
214     set udpSender5($j) [new Agent/UDP]
215     $ns attach-agent $fudp1($j) $udpSender1($j)
216     $ns attach-agent $fudp2($j) $udpSender2($j)
217     $ns attach-agent $fudp3($j) $udpSender3($j)
218     $ns attach-agent $fudp4($j) $udpSender4($j)
219     $ns attach-agent $fudp5($j) $udpSender5($j)}
220
221 #Conexiones
222 for {set j 1} {$j<=$NumbSrc} {incr j} {
223     $ns attach-agent $S1($j) $tcp_src1($j)
224     $ns attach-agent $S2($j) $tcp_src2($j)
225     $ns attach-agent $S3($j) $tcp_src3($j)
226     $ns attach-agent $D $tcp_snk1($j)
227     $ns attach-agent $D $tcp_snk2($j)
228     $ns attach-agent $D $tcp_snk3($j)
229     $ns connect $tcp_src1($j) $tcp_snk1($j)
230     $ns connect $tcp_src2($j) $tcp_snk2($j)
231     $ns connect $tcp_src3($j) $tcp_snk3($j)}
232
233 for {set j 1} {$j<=$NumbUDP} {incr j} {
234     $ns connect $udpSender1($j) $null1($j)
235     $ns connect $udpSender2($j) $null2($j)
236     $ns connect $udpSender3($j) $null3($j)
237     $ns connect $udpSender4($j) $null4($j)
238     $ns connect $udpSender5($j) $null5($j)}
239
240 #Fuentes FTP
241 for {set j 1} {$j<=$NumbSrc} {incr j} {
242     set ftp1($j) [$tcp_src1($j) attach-source FTP]
243     set ftp2($j) [$tcp_src2($j) attach-source FTP]
244     set ftp3($j) [$tcp_src3($j) attach-source FTP]}
245
246 #Crea el agente CBR para las fuentes UDP
247 for {set j 1} {$j<=$NumbUDP} {incr j} {
248     set cbr1($j) [new Application/Traffic/CBR]
249     $cbr1($j) attach-agent $udpSender1($j)
250     $cbr1($j) set rate_ 0.1Mb
251     $cbr1($j) set packetSize_ 552
252     $cbr1($j) set random_ false
253     set cbr2($j) [new Application/Traffic/CBR]
254     $cbr2($j) attach-agent $udpSender2($j)
255     $cbr2($j) set rate_ 0.1Mb
```

```
256     $cbr2($j) set packetSize_ 552
257     $cbr2($j) set random_ false
258     set cbr3($j) [new Application/Traffic/CBR]
259     $cbr3($j) attach-agent $udpSender3($j)
260     $cbr3($j) set rate_ 0.1Mb
261     $cbr3($j) set packetSize_ 552
262     $cbr3($j) set random_ false
263     set cbr4($j) [new Application/Traffic/CBR]
264     $cbr4($j) attach-agent $udpSender4($j)
265     $cbr4($j) set rate_ 0.1Mb
266     $cbr4($j) set packetSize_ 552
267     $cbr4($j) set random_ false
268     set cbr5($j) [new Application/Traffic/CBR]
269     $cbr5($j) attach-agent $udpSender5($j)
270     $cbr5($j) set rate_ 0.1Mb
271     $cbr5($j) set packetSize_ 552
272     $cbr5($j) set random_ false}
273
274 #PROCEDIMIENTOS
275
276 #Procedimientos "plotActualQueue"
277 proc plotActualQueue1 {} {
278     global ns qmon1 qsize1 R1 R2
279     set time 0.2
280     set now [$ns now]
281     #imprime en el archivo $qsize el tamaño actual de la cola
282     $qmon1 instvar pkts_
283     puts $qsize1 "$now [expr $pkts_]"
284     $ns at [expr $now+$time] "plotActualQueue1"}
285
286 proc plotActualQueue2 {} {
287     global ns qmon2 qsize2 R2 R3
288     set time 0.2
289     set now [$ns now]
290     #imprime en el archivo $qsize el tamaño actual de la cola
291     $qmon2 instvar pkts_
292     puts $qsize2 "$now [expr $pkts_]"
293     $ns at [expr $now+$time] "plotActualQueue2"}
294
295 proc plotActualQueue3 {} {
296     global ns qmon3 qsize3 R3 R4
297     set time 0.2
298     set now [$ns now]
299     #imprime en el archivo $qsize el tamaño actual de la cola
300     $qmon3 instvar pkts_
301     puts $qsize3 "$now [expr $pkts_]"
302     $ns at [expr $now+$time] "plotActualQueue3"}
```

```
303
304 proc plotActualQueue4 {} {
305     global ns qmon4 qsize4 R4 R5
306     set time 0.2
307     set now [$ns now]
308     #imprime en el archivo $qsize el tamaño actual de la cola
309     $qmon4 instvar pkts_
310     puts $qsize4 "$now [expr $pkts_]"
311     $ns at [expr $now+$time] "plotActualQueue4"}
312
313 proc plotActualQueue5 {} {
314     global ns qmon5 qsize5 R5 R6
315     set time 0.2
316     set now [$ns now]
317     #imprime en el archivo $qsize el tamaño actual de la cola
318     $qmon5 instvar pkts_
319     puts $qsize5 "$now [expr $pkts_]"
320     $ns at [expr $now+$time] "plotActualQueue5"}
321
322 #Imprimir la probabilidad de pérdida
323 proc plotProba {cola file} {
324     global ns
325     set time 0.2
326     set now [$ns now]
327     puts $file "$now [[$cola queue] set prob_]"
328     $ns at [expr $now+$time] "plotProba $cola $file" }
329
330 #Imprimir cola promedio
331 proc plotQprom {cola file} {
332     global ns
333     set time 0.2
334     set now [$ns now]
335     puts $file "$now [[$cola queue] set qprom_]"
336     $ns at [expr $now+$time] "plotQprom $cola $file" }
337
338 #Imprimir Dispersión
339 proc plotDisp {cola file} {
340     global ns
341     set time 0.2
342     set now [$ns now]
343     puts $file "$now [[$cola queue] set disp_]"
344     $ns at [expr $now+$time] "plotDisp $cola $file" }
345
346 #Imprimir parámetro ac del ATPI
347 proc plotac {cola file} {
348     global ns
349     set time 0.2
```

```
350     set now [$ns now]
351     puts $file "$now [[$cola queue] set ac_]"
352     $ns at [expr $now+$time] "plotac $cola $file" }
353
354 #Imprimir parámetro bc del ATPI
355 proc plotbc {cola file} {
356     global ns
357     set time 0.2
358     set now [$ns now]
359     puts $file "$now [[$cola queue] set bc_]"
360     $ns at [expr $now+$time] "plotbc $cola $file" }
361
362 #Correr simulación
363 $ns at 0.1 "plotActualQueue1"
364 $ns at 0.1 "plotActualQueue2"
365 $ns at 0.1 "plotActualQueue3"
366 $ns at 0.1 "plotActualQueue4"
367 $ns at 0.1 "plotActualQueue5"
368
369 #Programación de eventos para los agentes FTP
370 for {set j 1} {$j<=$NumbSrc} {incr j} {
371     $ns at 0.00 "$ftp1($j) start"
372     $ns at 0.00 "$ftp2($j) start"
373     $ns at 200.00 "$ftp1($j) stop"
374     $ns at 400.00 "$ftp1($j) start"
375     $ns at 400.00 "$ftp3($j) start"}
376
377 #Programación de eventos para los agentes CBR
378 for {set j 1} {$j<=$NumbUDP} {incr j} {
379     $ns at 0.1 "$cbr1($j) start"
380     $ns at 0.1 "$cbr2($j) start"
381     $ns at 0.1 "$cbr3($j) start"
382     $ns at 0.1 "$cbr4($j) start"
383     $ns at 0.1 "$cbr5($j) start"}
384
385
386 $ns at [expr $Duration] "finish"
387
388 $ns at 0.0 "plotProba $cola1 $pVstime1"
389 $ns at 0.0 "plotProba $cola2 $pVstime2"
390 $ns at 0.0 "plotProba $cola3 $pVstime3"
391 $ns at 0.0 "plotProba $cola4 $pVstime4"
392 $ns at 0.0 "plotProba $cola5 $pVstime5"
393 $ns at 0.0 "plotQprom $cola1 $qpVstime1"
394 $ns at 0.0 "plotQprom $cola2 $qpVstime2"
395 $ns at 0.0 "plotQprom $cola3 $qpVstime3"
396 $ns at 0.0 "plotQprom $cola4 $qpVstime4"
```

```
397 $ns at 0.0 "plotQprom $cola5 $qpVstime5"
398 $ns at 0.0 "plotDisp $cola1 $disp1"
399 $ns at 0.0 "plotDisp $cola2 $disp2"
400 $ns at 0.0 "plotDisp $cola3 $disp3"
401 $ns at 0.0 "plotDisp $cola4 $disp4"
402 $ns at 0.0 "plotDisp $cola5 $disp5"
403 $ns at 0.0 "plotac $cola1 $paramac1"
404 $ns at 0.0 "plotac $cola2 $paramac2"
405 $ns at 0.0 "plotac $cola3 $paramac3"
406 $ns at 0.0 "plotac $cola4 $paramac4"
407 $ns at 0.0 "plotac $cola5 $paramac5"
408 $ns at 0.0 "plotbc $cola1 $parambc1"
409 $ns at 0.0 "plotbc $cola2 $parambc2"
410 $ns at 0.0 "plotbc $cola3 $parambc3"
411 $ns at 0.0 "plotbc $cola4 $parambc4"
412 $ns at 0.0 "plotbc $cola5 $parambc5"
413
414 $ns run
```

Glosario

ACK : *acknowledgments*. Notificación enviada por un dispositivo de la red a otro para comunicar que se produjo un evento determinado.

AIMD : *additive increase, multiplicative decrease*.

Ancho de banda : se denomina ancho de banda digital a la cantidad de datos que se pueden transmitir en una unidad de tiempo. En ingeniería de redes este término se utiliza también para los métodos en donde dos o más señales comparten un medio de transmisión.

ARPA : *Advanced research projects agency*.

ARPAnet : Primera red de área amplia. considerada como la esencia de Internet.

AQM : *active queue management*. Mecanismos ejecutados en los enrutadores que gerencia el tamaño de la cola mediante la eliminación de paquetes.

Búfer : área de almacenamiento usada para manejar datos en tránsito. Los búferes se usan en las redes para compensar las diferencias en velocidad de procesamiento entre dispositivos de la red.

CBR : *constant bit rate*. Se trata de un método de compresión de datos en la codificación (generalmente de audio y vídeo) que conserva constante el bitrate en todo el fichero. El proceso opuesto es VBR. Su principal ventaja es la predicción del tamaño final del fichero en función de la duración del mismo, lo cual, permite comprimir el archivo en función de las necesidades. Su principal inconveniente es la poca eficiencia que presenta, puesto que, los ficheros de audio o vídeo presentan

fragmentos de baja complejidad (silencios, imágenes estáticas,...) para los que CBR otorga la misma capacidad de información que para los fragmentos complejos, con lo cual, se desaprovecha capacidad, obteniendo una fichero de mayor tamaño del necesario. Su uso es el más extendido y común por parte de todos los codificadores de audio y vídeo.

Cola : reserva de paquetes que esperan ser enviados por una interfaz de enrutador.

Congestión : La congestión se define como una excesiva cantidad de paquetes almacenados en los buffers de varios nodos en espera de ser transmitidos. La congestión es indeseable porque aumenta los tiempos de viaje de los paquetes y retrasa la comunicación entre usuarios.

Conmutación de paquetes : es el proceso mediante el cual un portador separa los datos en paquetes. Cada paquete contiene la dirección de origen, la dirección de su destino, e información acerca de cómo volver a unirse con otros paquetes emparentados. Este proceso permite que paquetes de distintas localizaciones se entrecrucen en las mismas líneas y que sean clasificados y dirigidos a distintas rutas.

Enrutador : es un dispositivo hardware o software de interconexión de redes de computadoras que opera en la capa tres (nivel de red) del modelo OSI. Este dispositivo interconecta segmentos de red o redes enteras. Hace pasar paquetes de datos entre redes tomando como base la información de la capa de red. El enrutador toma decisiones lógicas con respecto a la mejor ruta para el envío de datos a través de una red interconectada y luego dirige los paquetes hacia el segmento y el puerto de salida adecuados. Sus decisiones se basan en diversos parámetros. Una de las más importantes es decidir la dirección de la red hacia la que va destinado el paquete. Otras decisiones son la carga de tráfico de red en las distintas interfaces de red del router y establecer la velocidad de cada uno de ellos, dependiendo del protocolo que se utilice.

FTP : *file transfer protocol*. Es uno de los diversos protocolos de la red Internet, concretamente significa protocolo de transferencia de ficheros y es el ideal para

transferir grandes bloques de datos por la red. Su comportamiento está definido por la recomendación RFC 959.

Hardware : Se denomina *hardware* o soporte físico al conjunto de elementos materiales que componen un ordenador o una computadora.

Internet : es una red de redes a escala mundial de millones de computadoras interconectadas con un conjunto de protocolos. También se usa este nombre como sustantivo común y por tanto en minúsculas para designar a cualquier red de redes que use las mismas tecnologías que Internet, independientemente de su extensión o de que sea pública o privada.

IP : *internet protocol*. es un protocolo no orientado a conexión usado tanto por el origen como por el destino para la comunicación de datos a través de una red de paquetes conmutados. Los datos en una red basada en IP son enviados en bloques conocidos como datagramas. En particular, en IP no se necesita ninguna configuración antes de que un equipo intente enviar paquetes a otro con el que no se había comunicado antes. IP provee un servicio de datagramas no fiable y no provee ningún mecanismo para determinar si un paquete alcanza o no su destino y únicamente proporciona seguridad de sus cabeceras y no de los datos transmitidos.

Paquete : agrupación lógica de información que incluye un encabezado pedazo de información enviada a través de la red.

RED : *random early detection*.

Red : es un conjunto de computadoras y/o dispositivos conectados entre sí y que comparten información (archivos), recursos (CD-ROM, impresoras, etc.) y servicios (e-mail, chat, juegos).

Reglas CAM : reglas de ajuste del control PI-AQM para el manejo activo de colas llamadas también reglas Carrero-Márquez

Software : suma total de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de cómputo.

TCP : *transmission control protocol*.

Bibliografía

Altman, E. (2001), Modelado de sistemas informaticos y de telecomunicación, Notas de clase, Sophia-Antipolis y Universidad de los Andes.

Athuraliya, S., Li, V. H., Low, S. H. & Yin, Q. (2001), 'REM: Active queue management', Disponible en http://netlab.caltech.edu/FAST/papers/cbef.pdf.

Carrero, N. (2006), Reglas de ajuste del control PI-AQM, Proyecto de Grado EISULA, Universidad de los Andes, Escuela de Ingeniería de Sistemas.

Chiu, D. & Jain, R. (1989), 'Analysis of the increase and decrease algorithms for congestion avoidance in computer networks', Disponible en http://www.cse.wustl.edu/ jain/papers/ftp/cong_av.pdf.

Comer, D. (1996), *Redes globales de información con Internet y TCP/IP*, 3 edn, Prentice Hall.

Fall, K. & Varadhan, K. (2002), *The NS Manual*. Disponible en http://www.isi.edu/nsnam/ns/doc/index.html.

Feng, W., Kandlur, D. & an K. Shin, D. S. (2002), 'The blue active queue management alghorithms', Disponible en http://delivery.acm.org.

Feng, W., Kandlur, D., Saha, D. & Shin, K. (1999), 'A self-configuring RED gateway', Disponible en http://debanjan.sahafamily.com/papers/00752150.pdf.

Firoiu, V. & Borden, M. (2000), 'A study of active queue management for congestion control', Disponible en http://victor.firoiu.org/papers/red-dynamics-conf.pdf.

Floyd, S., Hadley, M. & Padhye, J. (2000), 'A comparison of equation-based and AIMD congestion control', Disponible en http://www.icir.org/tfrc/aimd.pdf.

Floyd, S. & Jacobson, V. (1993), 'Random early detection gateways for congestion avoidance', Disponible en http://www.icir.org/floyd/papers/early.twocolumn.pdf.

for Beginners, N. S. (2003), 'E. altman and t. jiménez', Disponible en http://www-sop.inria.fr/maestro/personnel/Eitan.Altman/COURS-NS/n3.pdf.

Hassan, M. & Jain, R. (2004), *High Performance TCP/IP Networking: concepts, Issues and solutions*, Prentice Hall.

Hollot, C., Misra, V., Towsley, D. & Gong, W. (2002), 'Analysis and design of controllers for AQM routers supporting TCP flows', Disponible en http://dna-wsl.cs.columbia.edu/pubsdb/citation/paperfile/30/TAC_special.pdf.

Jacobson, V. (1988), 'Congestion avoidance ans control', Disponible en http://citeseer.ist.psu.edu/article/jacobson88congestion.html.

Kunniyur, S. & Srikant, R. (2001), 'Analysis and design of an adaptive virtual queue algorithm for active queue management', Disponible en http://www.sigcomm.org/sigcomm2001/p10-kunniyur.pdf.

Long, C., Zhao, B., Guan, X. & Yang, J. (2005), 'The yellow active queue management algorithm', Disponible en http:www.sciencedirect.com.

Marquez, R., Altman, E. & Solé-Álvarez, S. (2004), Modeling TCP and HighSpeed TCP: A nonlinear extension to aimd mechanisms, *in* 'Proceedings of 7th IEEE International Conference on High Speed Networks and Multimedia Communications (HSNMC'04)', Toulouse, Francia, pp. 685–702.

Miguel, J. (2003), 'Simulador de redes telemáticas', Disponible en http://www.redeweb.com/microbit/articulos/660603.pdf.

Misra, V., Gong, W. & Towsley, D. (2000), Fluid-based analysis of a network of AQM routers, suporting TCP flows with an application to RED, *in* 'Proceedings of ACM/SIGCOMM', Estocolmo, Suecia.

Noureddine, W. & Tobagi, F. (2002), 'The transmission control protocol'.

Ott, T., Lakshman, T. & Wong, L. (1999), 'Sred: stabilized red', Disponible en http://web.njit.edu/ ott/Papers/Lakshman/Infocom1999.pdf.

Park, E. C., Lim, H., Park, K. J. & Choi, C. H. (2004), 'Analysis and design of the virtual rate control algorithm for stabilizing queues in tcp networks', *Computer Networks* **44**(1), 17–41.

Saltzer, J., Reed, D. & Clark, D. (1984), 'End to end arguments in systems design', Disponible en http://www.web.mit.edu/Saltzer/www/publications/endtoend/endtoend.pdf.