TRABAJO FINAL DE CARRERA

TÍTULO: Avaluació d'un proxy en entorns amb i sense fils

AUTOR: Álvaro Rodatos Rite

DIRECTOR: Lluis Casals Ibáñez

FECHA: 5 de septiembre de 2005

Título: Avaluació d'un proxy en entorns amb i sense fils

Autor: Álvaro Rodatos Rite

Director: Lluis Casals Ibáñez

Fecha: 5 de septiembre de 2005

Resumen

Las nuevas tecnologías han permitido el desarrollo de nuevos sistemas de telecomunicaciones en medios no guiados. Aunque que los sistemas inalámbricos cada vez disponen de más velocidad de transmisión y reducen los costes en la instalación frente a un sistema cableado, la tasa de error sigue siendo muy elevada en comparación con un entorno cableado. Los protocolos que se usan como el de transporte TCP, fueron pensados para medios guiados.

El objetivo de este proyecto es hacer un estudio detallado sobre el proxy snoop con el simulador de redes NS-2 (*Network Simulator*). El proxy es una de las propuestas para mejorar la eficiencia y el funcionamiento del protocolo de transporte TCP (*Transmission Control Protocol*) en entornos inalámbricos, pero como veremos, el proxy tiene algunas limitaciones.

Con el fin de adaptar el proxy snoop a enlaces de baja velocidad como puede ser una red GPRS (*General Packet Radio Service*), hemos modificado el módulo snoop del simulador con nuevas funcionalidades. Estas funcionalidades están basadas en unos estudios existentes [2]. Además, algunas limitaciones del simulador nos ha obligado a modificar el modelo que implementa el simulador del protocolo TCP para adaptarlo a nuestras necesidades. También hemos desarrollado una propuesta de adaptación del proxy snoop al modelo fullTCP implementado por el simulador para poder crear simulaciones con ambos módulos del NS-2.

Title: A proxy evaluation within wired and wireless simulations

Author: Álvaro Rodatos Rite

Director: Lluis Casals Ibáñez

Date: September, 5th 2005

Overview

As new technologies have allowed the development of new communications systems without wires. Nowadays, wireless systems are becoming very popular and are possible to achieve more bandwidth and therefore more transmission rate. In addition, wireless systems reduce the installation costs comparing with wired systems. The error rate continues being very high in comparison with wired architectures. The protocols used like the transmission control protocol TCP were thought especially for wired system where the most important problem is the network congestion instead of high error rate.

The goals of this project are, on one hand make a detailed study with a proxy snoop using the network simulator NS-2 (Network Simulator). The Proxy snoop is one of the approaches to improve the throughput of the transmission control protocol in wireless links, but as we'll see, the proxy has some limitations.

With the purpose of adapting the proxy snoop to low speed links as a GPRS (General Packet Radio Service) network. We've modified the snoop module of the simulator with new features. These features are based on existing studies [2]. In addition, some limitations of the simulator have forced us to modify the TCP model implemented in the simulator, just to adapt it to our necessities. Also we've developed a proposal of proxy snoop adaptation to the fullTCP model implemented by the simulator just to be able to run simulations with both modules of the network simulator.

ÍNDICE

INTRODUCCIÓN

Las redes inalámbricas cada vez son más populares y surgen muchas propuestas para mejorar la eficiencia de los protocolos de la capa de transporte como TCP (*Transmission Control Protocol*), en enlaces inalámbricos. Una de las ventajas de los sistemas inalámbricos es que tienen un coste inferior de instalación y mantenimiento ya que no es necesario un proyecto de cableado estructurado.

El protocolo TCP fue creado pensando en medios guiados dónde la tasa de error es muy baja y el principal problema que puede ocasionar perdidas de paquetes es la congestión. Por eso, los mecanismo que invoca TCP para hacer frente a cualquier perdida, están pensados para prevenir la congestión. Estos mecanismos de prevención de congestión no son adecuados cuando el principal problema de perdida de paquetes es la tasa de error de la red, que es lo que ocurre en redes inalámbricas.

Hay diversas propuestas con la finalidad de mejorar la eficiencia de TCP en enlaces inalámbricos. Los reconocimiento selectivos, notificaciones explicitas de pérdida o conexiones partidas son algunas de estas propuestas. La propuesta que nosotros vamos a estudiar esta basada en el proxy snoop. El proxy se caracteriza por trabajar a nivel de enlace pero también monitoriza las conexiones TCP a nivel de transporte. Esto permite que el proxy pueda eliminar los reconocimientos duplicados y retransmitir segmentos TCP cuando se produzcan perdidas.

No obstante el proxy no siempre mejora las prestaciones del sistema, en este caso la eficiencia de TCP. En unos estudios existentes [2], basados en una implementación real con el proxy snoop y varias operadoras GPRS, quedó demostrado la inconveniencia de usar el proxy snoop en este tipo de redes. En los documentos consultados se describe también un propuesta de adaptación del proxy snoop a una red GPRS. Algunas de las características del proxy snoop adaptado son su temporizador de retransmisiones constante y reconocimientos que abren y cierran la ventana de transmisión del emisor TCP.

Los objetivos originales del proyecto son, por un lado plasmar las nuevas funcionalidades del proxy snoop adaptado a GPRS en el simulador NS-2 y por otro lado, realizar una serie de simulaciones con el nuevo módulo en el simulador NS-2 para verificar su funcionamiento. Debido a la necesidad de modificar el modelo TCP que implementa el NS-2, parte de los objetivos originales del proyecto fueron cambiados. En particular hemos de adaptar el modelo TCP del simulador a un TCP dónde podamos abrir o cerrar la ventana de congestión. Además, presentamos una propuesta de adaptación del proxy para poder realizar pruebas con la modelo FullTCP del simulador NS-2.

En el primer y segundo capítulo, hacemos una pequeña descripción de los protocolos involucrados en la pruebas, así como el simulador de redes y las herramientas que nos proporciona el simulador para analizar los resultados de las pruebas.

En el capítulo 3 describimos las pruebas realizadas con el simulador NS-2. En ellas, hemos utilizado diferentes generadores de tráfico como FTP (*File Transfer Protocol*), CBR (*Constant Bit Rate*) ó TELNET (*Terminal Emulation*) y hemos evaluado el proxy en sistemas con enlaces de distinto ancho de banda. En este capítulo también explicamos por qué el proxy no es capaz de monitorizar una sesión FullTCP y que tendríamos que hacer para conseguir probar ambos módulos en una misma simulación. Al final del capítulo comentamos las conclusiones generales sobre las pruebas, ventajas y desventajas de la utilización del proxy.

En el capítulo 4 presentamos la propuesta de modificación del modelo proxy snoop implementado por el simulador basándonos en unos estudios ya existente [2]. Aquí describimos las funciones programadas para conseguir dicho objetivo. En el anexo A de este documento se puede encontrar los códigos completos con algunos comentarios para facilitar su entendimiento.

CAPÍTULO 1. Descripción del entorno

En este capitulo se van a describir algunos conceptos necesarios para ayudar a entender este documento. Estos conceptos son el protocolo de transporte TCP, el proxy snoop y el simulador NS-2.

1.1. El protocolo de transporte TCP

Existen diferentes versiones de éste protocolo como Tahoe, Reno, New Reno y Vegas. Tahoe es el más común e incorpora los algoritmos de arranque lento *(Slow Start),* retransmisión rápida *(Fast Retransmit)* y prevención de congestión *(Congestión Avoidance)*. TCP Reno además de los algoritmos que tiene Tahoe incluye un mecanismo de recuperación rápida *(Fast Recovery)* que explicamos a continuación. Reno es la versión de TCP que usaremos en las simulaciones.

La figura 1.1 muestra los campos de la cabecera TCP real, como veremos el simulador no utiliza muchos de estos campos en la cabecera TCP del simulador.

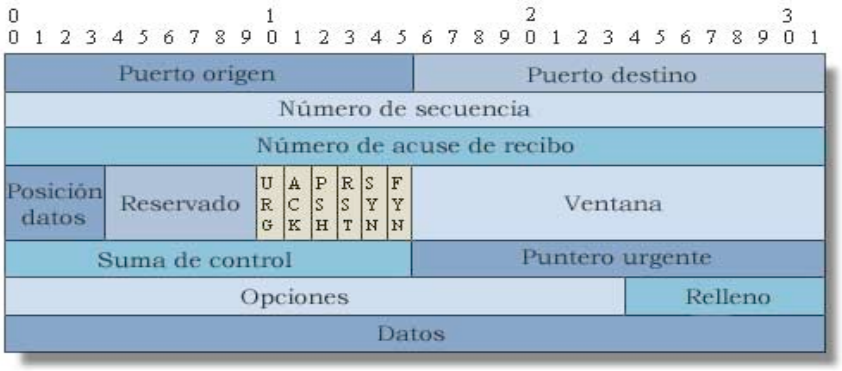

Fig. 1.1 Cabecera del protocolo TCP

Al iniciarse una conexión, el extremo emisor de la comunicación irá inyectando datos a la red de forma controlada mediante el algoritmo de arranque lento. Si no se detectan perdidas en la comunicación, la ventana de congestión del emisor TCP crecerá enviando así más segmentos.

Cuando se producen errores en la comunicación, el emisor se verá obligado a reducir la ventana de congestión a la mitad y comenzar el arranque lento (inyectar segmentos TCP poco a poco). El motivo es que TCP asume que este paquete se ha perdido debido a la congestión de la red, por lo tanto si hay congestión en la red la ventana de congestión ha de disminuir su tamaño para evitar, en la medida de lo posible, que la red siga congestionada.

Con el fin de evitar esta reducción de la ventana de congestión, el mecanismo de recuperación rápida en el emisor TCP irá inflando la ventana de congestión con cada reconocimiento por parte del receptor TCP y de esta forma el proceso de recuperación frente a errores en la comunicación es más eficiente.

Estas características nativas del protocolo TCP de reducción de ventana frente a perdidas en la transmisión de datos, hacen que disminuya la eficiencia en sistemas no cableados. A continuación describimos algunas de las propuestas para mejorar las prestaciones de TCP en enlaces inalámbricos [4].

Extremo a extremo: consiste en añadir mejoras al protocolo TCP. Las versiones Reno, New Reno, SACKS (*Selective acknowlegment*) y ELN (*Explicit Lost Notification*) son algunas de estas mejoras.

Conexiones TCP partidas o divididas: en este caso partimos la conexión TCP en dos, por un lado el nodo inalámbrico y la estación base (parte problemática) y por otro lado el nodo fijo y la estación base. El problema es que la estación base ha de procesar los paquetes dos veces, aumenta la información de estado y se rompe la semántica extremo a extremo de los acks TCP.

La propuesta que nosotros vamos a tratar consiste en conseguir un nivel de enlace fiable. Para alcanzar dicho objetivo se introduce un proxy caché, trabajando a nivel de enlace, en la estación base.

1.2. El proxy Snoop

La principal característica de cualquier dispositivo denominado proxy caché es su capacidad de almacenar información. El tipo de información que se almacenará dependerá del nivel del modelo OSI en el que trabaje el proxy. El proxy snoop monitoriza cada paquete que pasa a través de una conexión TCP (en ambos sentidos de la conexión) y guarda en una caché todos los segmentos TCP que todavía no han sido reconocidos por el receptor. En la figura 1.2 se puede ver una topología en la que podríamos usar el proxy para corregir los posibles problemas en el enlace inalámbrico.

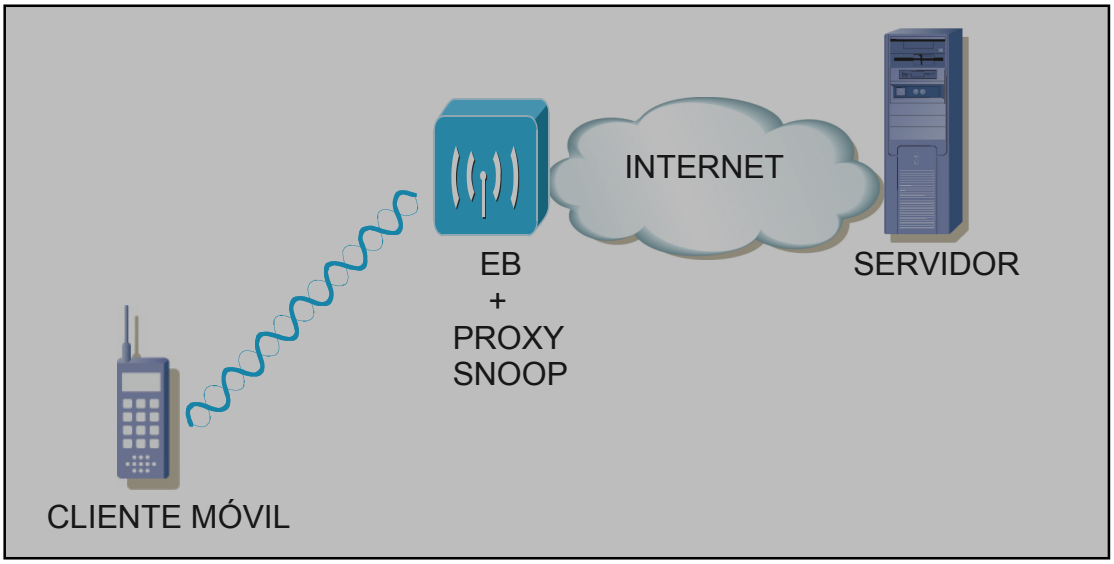

Fig. 1.2 Escenario de aplicación del proxy snoop

El proxy detecta que un paquete se ha perdido con la llegada de reconocimientos duplicados o cuando expira el temporizador local de retransmisiones regido por la llegada de reconocimientos (ACK's) desde el receptor.

Además, el proxy retransmite el paquete perdido y elimina los reconocimiento duplicados. Éste se implementa a nivel de enlace pero tiene en cuenta el funcionamiento del protocolo de transporte de la capa superior (TCP).

La principal ventaja de esta propuesta de mejora de la eficiencia de TCP en entornos inalámbricos, es que suprime los reconocimiento duplicados de los segmentos TCP y retransmite localmente los paquetes perdidos de manera que evita que el emisor TCP invoque mecanismos de control de congestión y retransmisión rápida.

1.3. El simulador de redes

El simulador de redes *Network Simulator* será nuestra herramienta principal para realizar la evaluación del proxy snoop. Este simulador nos permite caracterizar diferentes escenarios, así pondremos a prueba el funcionamiento del proxy y veremos la diferencia que se produce en el comportamiento de TCP cuando se expone a sistemas de perdidas o enlaces inalámbricos.

El simulador de redes (NS-2) combina los lenguajes de programación OTcl (Object Tool Command Language) y C++ (lenguaje C orientado a objetos). Para construir las topologías de una simulación así como el ancho de banda de los enlaces, retardo de propagación y tasa de error en paquetes para un enlace determinado, utilizaremos el lenguaje de programación TCL.

Por otro lado, C++ se puede considerar como el lenguaje de programación para implementar un modelo de cada protocolo y sus estructuras de datos. Este lenguaje es utilizado para manejar eficientemente bytes, cabeceras de paquetes e implementar algoritmos que utilizan grandes volúmenes de información. Para este tipo de tareas el tiempo de proceso es importante, y es necesario un lenguaje como C++, por lo tanto sólo programaremos en TCL para reconfiguran parámetros de la simulación en tiempo real.

La metodología que hemos utilizado, para definir, ejecutar y analizar una simulación, es la siguiente:

- Programamos las características de nuestra nueva topología a nivel TCL. Definimos generador de tráfico, ancho de banda de los enlaces y tasa de error del enlace inalámbrico.

- Utilizando el simulador NS-2 generamos un archivo con todos los eventos o trazas producidos durante el tiempo de situación.

- Generamos, aplicando varios filtros sobre las trazas, las graficas tiempo-secuencia o simplemente el throughput de la conexión.

CAPÍTULO 2. Herramientas de análisis

En este capítulo se describen las herramientas principales que nos proporciona el simulador de redes (NS-2). En el anexo A describimos el funcionamiento de un archivo programado en Perl que usamos para calcular el throughput, el archivo que usamos para filtrar las trazas y el archivo para preparar la simulación.

2.1. El escenario de pruebas

En el escenario de la figura 1.2 hemos visto el escenario de aplicación del proxy snoop. En la figura 2.1 podemos ver el modelo implementado en el simulador para la topología de la figura 1.2. Haciendo una analogía con la figura 1.2 por un lado podemos consideras L1 como Internet, aunque en este enlace no habrá ni perdidas ni congestión, y por otro lado el enlace inalámbrico constituido por la LAN, el nodo ficticio, y L2. El cliente en la topología modelada corresponde con el cliente móvil en el escenario de aplicación.

El simulador NS-2 permite crear enlaces inalámbricos pero el estándar IEEE 802.11 no es compatible con el proxy porque su implementación como módulo del NS-2 fue posterior a la del proxy. Por este motivo, simularemos un enlace inalámbrico con un enlace cableado pero con una elevada tasa de error.

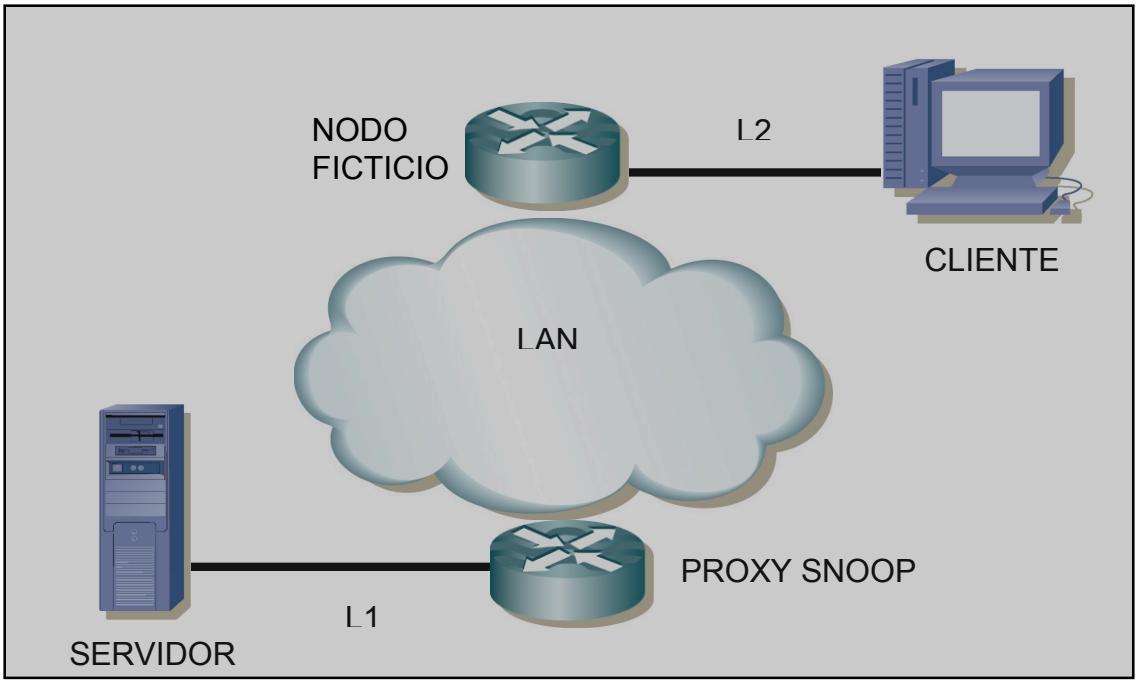

Fig. 2.1 Topología de red

En primer lugar, tenemos un servidor que utilizaremos para simular diferentes aplicaciones como FTP y distintos generadores de tráfico como una fuente CBR o TELNET. El servidor transmitirá la información al proxy a través del enlace L1. El enlace L2 que conecta el nodo ficticio y el cliente introducirá los errores en la transmisión. El nodo ficticio y el proxy están conectados a través de una LAN (*Local Area Network*). Esta LAN tiene un ancho de banda superior siempre a L1 y L2 es decir, en ningún caso limitará la conexión.

La figura 2.2 es una imagen de la herramienta NAM, con nuestro escenario de pruebas. De esta manera hemos podido modelar lo que sería una topología real como en la figura 1.2. El enlace que conecta el nodo 0 con el nodo 4 es nuestro enlace de perdidas que simula un enlace inalámbrico. En el nodo 2 esta situado el proxy snoop y el nodo 3 es el encargado de generar tráfico durante las simulaciones. Por último, el nodo 4 corresponde con el cliente móvil.

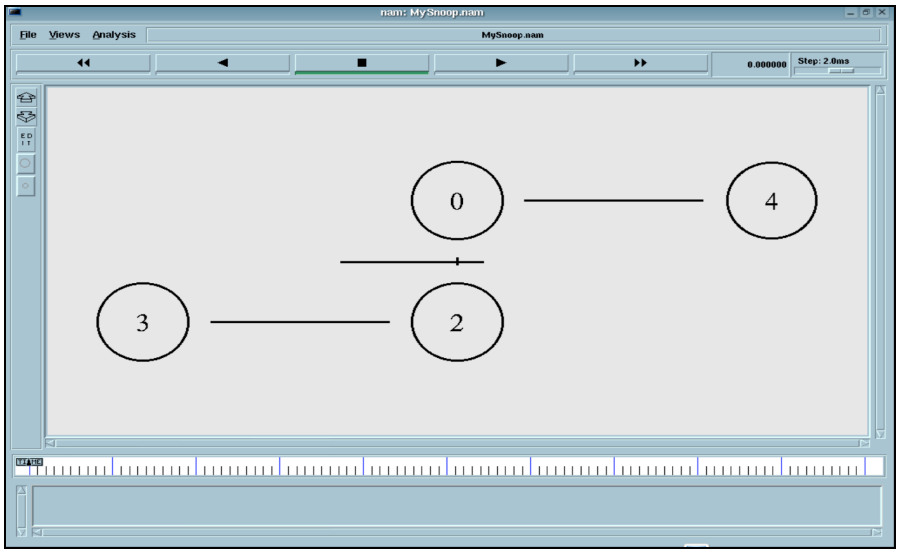

Fig. 2.2 Modelado de la topología real en el simulador

2.2. Gráficas tiempo secuencia y las trazas de la simulación

Una manera de estudiar el funcionamiento del protocolo TCP durante una conexión, es analizando las gráficas tiempo-secuencia de la simulación. Estas gráficas nos muestran la evolución del número de secuencia (número que identifica a cada segmento TCP) en función del tiempo. En este simulador el número de secuencia aumenta de uno en uno en lugar de la suma en bytes del tamaño del paquete (carga útil). Para poder obtener este tipo de graficas aplicamos un filtro, programado en TCL, sobre el archivo que contiene las trazas de la simulación.

Una vez que ponemos en marcha la simulación se crea un archivo de texto donde se detallan los eventos producidos durante todo el tiempo de observación (ó tiempo de simulación). En la figura 2.2 podemos ver la información que contiene cada una de las trazas.

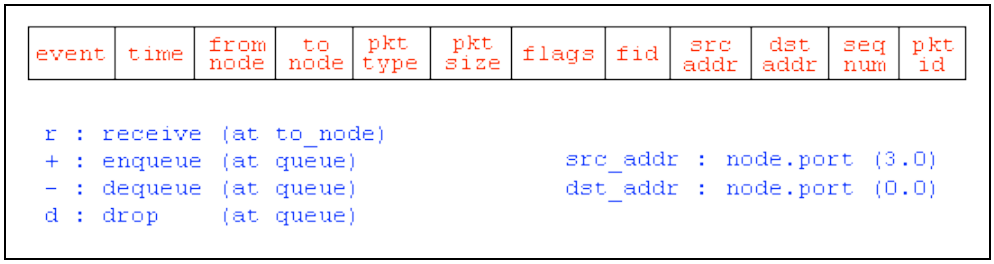

Fig. 2.2 Información que contiene las trazas de la simulación

La figura 2.3 es un fragmento de un archivo con algunos eventos producidos durante una simulación.

```
r 1.3556 3 2 ack 40 ------- 1 3.0 0.0 15 201
+ 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
- 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
r 1.35576 0 2 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
d 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
- 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
```

Fig. 2.3 Ejemplo de un archivo con eventos

En primer lugar, tenemos un paquete recibido (r) por el nodo 2 y enviado por el nodo 3, en un instante determinado (1,3556 seg.). A continuación podemos ver que se trata de un segmento TCP de 40 bytes con el flag ACK activado (es un reconocimiento) y su número de secuencia es 15 mientras que su identificador único de paquete es 201. En nuestro caso nos interesa filtrar el número de secuencia en función del tiempo (tanto segmentos TCP's como ACK's) para un enlace determinado (emisor ó receptor).

El simulador de redes nos proporciona un archivo (namfilter.tcl) que nos permite filtrar (siempre que sea una conexión TCP) los archivos con eventos y obtener una grafica tiempo secuencia para cualquier conexión TCP dentro de la simulación. En la figura 2.4 vemos una gráfica tiempo secuencia para una conexión libre de errores.

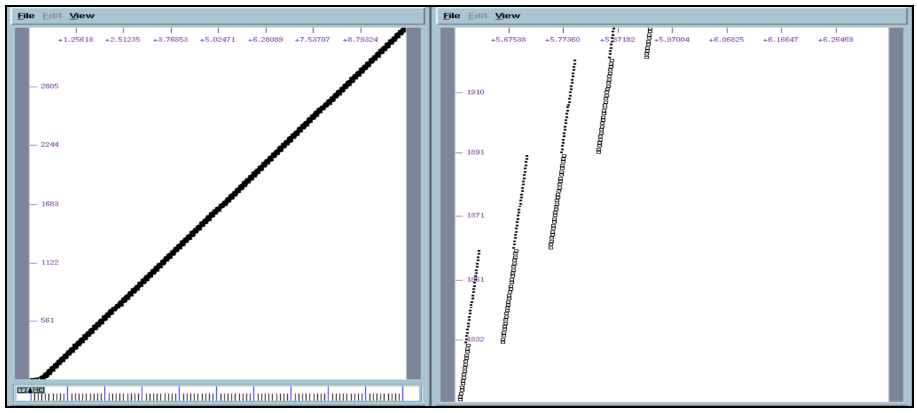

Fig. 2.4 Filtrado TCP en el emisor y ACK a la salida del proxy

La gráfica de la derecha es una ampliación de la gráfica de la izquierda, en ella podemos ver como la ventana de congestión alcanza su máximo valor (30 segmentos) durante la mayor parte de la simulación. En este caso, hemos filtrado desde el servidor, es decir todos los segmentos TCP que el propio servidor genera y todos los reconocimientos que recibe (los ACK's se filtran a la salida del proxy).

Para poder comprobar que es lo que ocurre desde el lado del cliente hemos modificado el archivo namfilter.tcl. En este caso los segmento TCP los filtramos a la salida del router IP y los ack's cuando los genera el cliente.

Debido al modo en que el simulador incrementa el número de secuencia hemos tenido que filtrar de este modo, así podemos ver por un lado los segmento TCP (cuadrados negros) y los reconocimientos (cuadrados blancos). Si no filtráramos de este forma se solaparían los segmentos TCP y los ack's.

CAPÍTULO 3. Evaluación del proxy snoop

En este capítulo se describen las pruebas realizadas con distintos generadores de tráfico. Como veremos, el proxy siempre es capaz de disminuir el número de retransmisiones del emisor TCP, en cambio no siempre es capaz de mejorar el throughput. Al final del capítulo hacemos una propuesta de adaptación del proxy a al versión fullTCP del simulador.

La metodología de las pruebas ha consistido en ir variando el ancho de banda del enlace inalámbrico y aumentando la tasa de error en %. Para cada tasa de error vamos disminuyendo el ancho de banda del enlace inalámbrico. Las pruebas se realizan con y sin la presencia del proxy snoop.

3.1. Evaluación con un servidor FTP

En nuestras primeras pruebas hemos utilizado el protocolo FTP (File Transfer Protocol). Para que el servidor este activo en todo momento, hemos considerado un archivo ilimitado que será descargado por el cliente durante el tiempo que dure la simulación (60 segundos).

3.1.1. Tasa de error en paquetes del 0%

En este caso no hemos introducido ningún error en la comunicación pero si hemos ido variando la velocidad del enlace inalámbrico para estudiar que ocurre en la eficiencia de la conexión. En la tabla 3.1 mostramos las retransmisiones que realiza el emisor TCP y el throughput.

Ancho de banda L1	Ancho de banda L2	Retransmisiones (emisor TCP)	Throughput (%)	Throughput (Mbps)
10 Mbps	10 Mbps	0	28,77	2,877
10 Mbps	2 Mbps	0	99,33	1,986
10 Mbps	0,3 Mbps	0	99,29	0,298
10 Mbps	0,024 Mbps	0	97,66	0,0234

Tabla 3.1 Resultados de las pruebas, sin errores

En principio, al ir disminuyendo la velocidad del enlace del receptor se puede observar como el comportamiento del protocolo de transporte TCP se va adaptando (no satura) y cada vez el número de paquetes que se envían es menor. El número de retransmisiones por parte del emisor es nulo ya que no hemos introducido perdidas en la simulación.

La figura 3.1 es una gráfica tiempo secuencia (de la última prueba realizada, enlace de 24 Kbps), en ella podemos ver la diferencia que hay entre la emisión

de los segmentos TCP y la recepción de los reconocimientos. Esta separación entre ambas trayectorias es debida a la diferencia de velocidades entre el enlace en el que esta el servidor y el enlace que conecta al cliente.

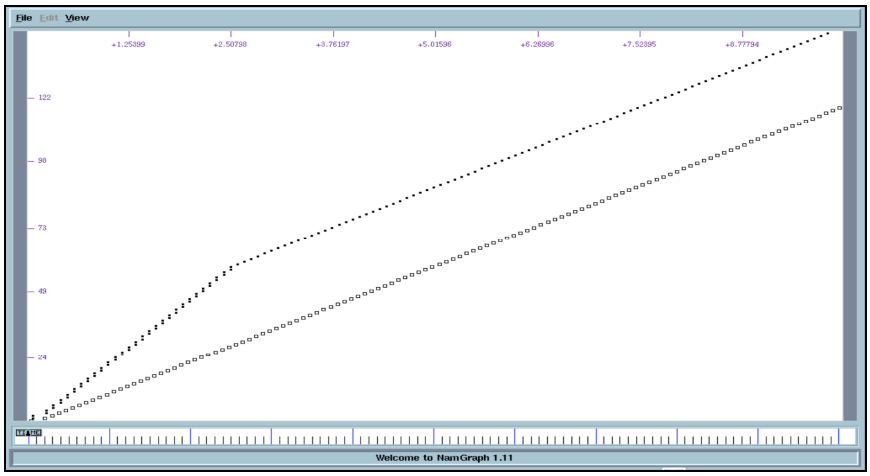

Fig. 3.1 Enlaces con distinto ancho de banda

3.1.2. Tasa de error en paquetes del 1%, sin proxy snoop

En este caso vamos a introducir perdidas. Una tasa de error en paquetes del 1%, corresponde con una tasa de error en bits de $1{,}2 \times 10^{-8}$ (BER), ya que nuestros paquetes son de 1040 bytes. En la tabla 3.2 podemos ver como cómo disminuye el throughput.

Ancho de banda L1	Ancho de banda L2	Retransmisiones (emisor TCP)	Throughput (%)	Throughput (Mbps)
10 Mbps	10 Mbps	72	8,52	0,852
10 Mbps	2 Mbps	72	40,72	0,814
10 Mbps	0,3 Mbps	24	95,82	0,287
10 Mbps	0,024 Mbps	2	99,42	0,0238

Tabla 3.2 Análisis de la comunicación FTP, PER 1%

Como podemos ver en la tabla, el emisor TCP detecta las perdidas que se producen, ya sean por reconocimientos duplicados ó por la expiración del temporizador de retransmisiones. El protocolo TCP se va adaptando a los cambios de velocidad de los enlaces ya que aunque cada vez se envía menos segmentos TCP, disminuye también, el número de retransmisiones.

En la figura 3.2 podemos ver la evolución del número de secuencia desde el servidor, es decir filtrando los segmentos TCP que genera el nodo 3 y los ack's que deja pasar el nodo 2.

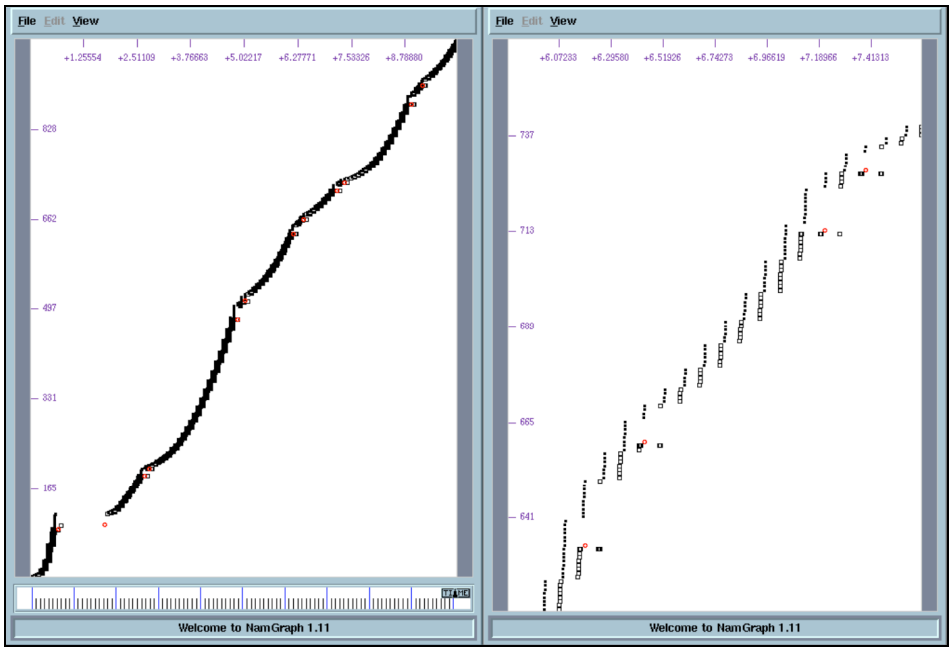

Fig. 3.2 Enlaces de igual AB, PER 1%

En la imagen de la izquierda se ve que el número de secuencia ya no aumenta linealmente, y esto es debido a las retransmisiones que vemos marcadas en rojo en ambas imágenes. Cada vez que se produce una retransmisión el emisor ha de reducir su ventana de transmisión y comenzar el arranque lento, es por esto, que en la imagen ampliada de la derecha, vemos como la ventana de transmisión nunca toma su máximo valor debido a las continuas retransmisiones.

Esto afecta directamente la eficiencia de la conexión, ya que TCP asume que estas perdidas son por congestión cuando el problema real es la tasa de error que estamos introduciendo.

3.1.3. Tasa de error en paquetes del 1%, con proxy snoop

En este caso hemos introducido el agente snoop a nivel de enlace y podemos ver como mejora en todos los casos la eficiencia de la comunicación. En la tabla 3.3 y en comparación con la tabla 3.2, el número de retransmisiones por parte del emisor TCP ha disminuido y ha aumentado la eficiencia de la conexión.

Por un lado, podemos ver que el proxy snoop cumple muy bien su misión que es evitar, en la medida de lo posible, que se produzcan retransmisiones por parte del emisor y por eso cuando detecta reconocimientos duplicados ó expira su temporizador de retransmisión, retransmite localmente el segmento sin que el emisor TCP note que se han producido perdidas.

Ancho de banda L1	Ancho de banda L2	Retransmisiones (emisor TCP)	Throughput (%)	Throughput (Mbps)
10 Mbps	10 Mbps	1	25,61	2,561
10 Mbps	2 Mbps	0	88,60	1,772
10 Mbps	0,3 Mbps	0	96,69	0,290
10 Mbps	0,024 Mbps	0	97,66	0,02344

Tabla 3.3 Análisis de la comunicación FTP, PER 1% con snoop

En la figura 3.4 podemos ver lo que ocurre en la comunicación desde el servidor. En este caso sólo hay un ack duplicados porque el proxy elimina de la red el resto.

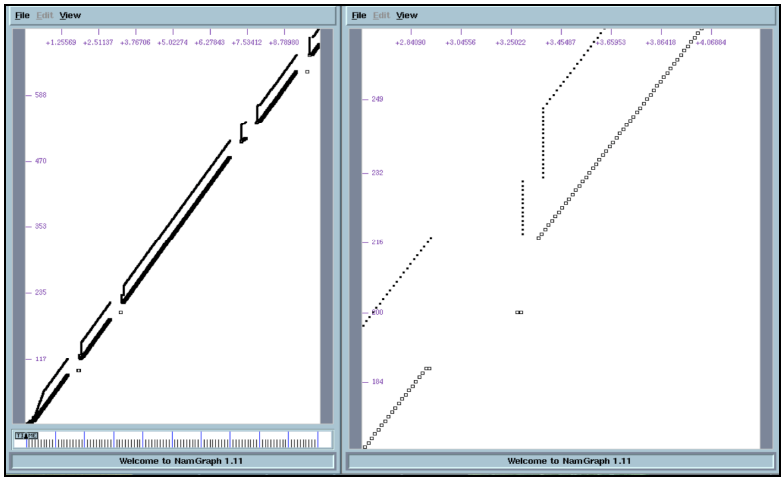

Fig. 3.4 Enlaces de distinto AB, PER 1% con snoop

En la gráfica de la derecha podemos ver una ampliación de uno de estos espacios que hay en la gráfica principal. No se produce ninguna retransmisión por parte del emisor ya que no hay puntos rojos en la gráfica y tampoco vemos los reconocimientos duplicados que genera el receptor ni el segmento retransmitido por el proxy snoop.

El motivo por el que no vemos ni las retransmisiones del proxy ni los reconocimientos duplicados es debido a que filtramos los segmentos TCP que genera el emisor TCP y los ACK que llegan al enlace L1, es decir, los reconocimientos que deja pasar el proxy.

Por otro lado, como ya previeron los creadores de este protocolo, el emisor no está exento totalmente de la percepción de que se han producido perdidas y por tanto es posible que aun con la intervención del proxy snoop se produzcan retransmisiones por parte del emisor TCP (ejemplo 1, tabla 3.3). La figura 3.5 muestra el punto de vista del cliente, es decir filtramos los TCP que salen del nodo 0 y los ack's que genera el cliente, nodo 4.

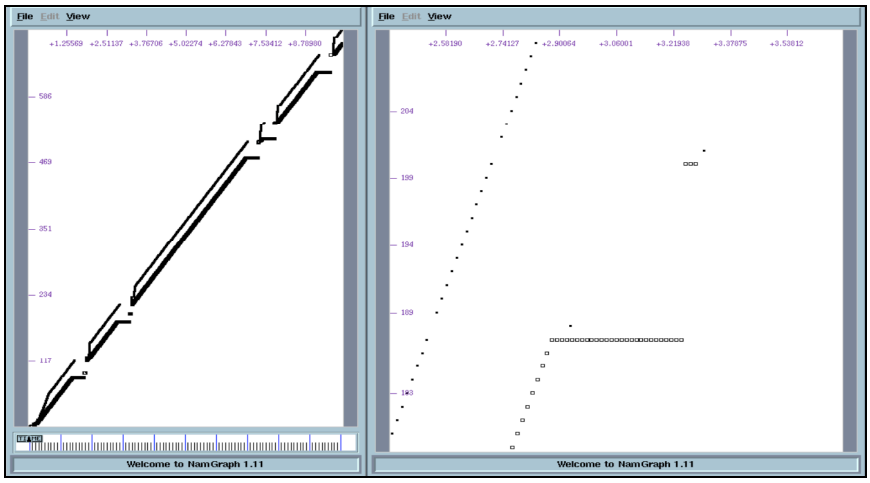

Fig. 3.5 Enlaces de distinto AB, filtrado en el cliente

Con este filtro, podemos ver todos los reconocimientos duplicados y justo encima esta el segmento TCP que retransmite localmente el proxy snoop y unos instantes antes y en la misma línea del segmento retransmitido, vemos un hueco en blanco que corresponde con el segmento perdido. El segmento perdido no lo vemos porque nunca llega al enlace que conecta al receptor y el filtro que aplicamos no lo muestra.

3.1.4. Tasa de error en paquetes del 5%, con y sin proxy snoop

Vamos a seguir aumentando la tasa de error en paquetes y de esta manera podremos evaluar la efectividad del proxy snoop. En este caso estamos probando con una tasa de error realmente alta, exactamente el doble de la tasa de error que probaron los creadores del proxy snoop [1], en unas pruebas que realizaron. En la tabla 3.3 mostramos los resultados.

Ancho de banda L1	Ancho de banda L2	Retransmisiones (emisor TCP)	Throughput (%)	Throughput (Mbps)
10 Mbps	10 Mbps	42	2,37	0,237
10 Mbps	2 Mbps	40	11,19	0,224
10 Mbps	0,3 Mbps	30	53,71	0,161
10 Mbps	0,024 Mbps	6	79,75	0,019

Tabla 3.3 Análisis de la comunicación FTP, PER 5% sin snoop

Como era de esperar, las retransmisiones por parte del emisor TCP son continuas y la eficiencia se ve muy afectada. En la tabla 3.4 vemos cómo el proxy aumenta el throughput de la comunicación.

Ancho de banda L1	Ancho de banda L2	Retransmisiones (emisor TCP)	Throughput (%)	Throughput (Mbps)
10 Mbps	10 Mbps	2	11,62	1,162
10 Mbps	2 Mbps	1	50,58	1,012
10 Mbps	0,3 Mbps	0	79,55	0,238
10 Mbps	0,024 Mbps	1	83,8	0,0201

Tabla 3.4 Análisis de la comunicación FTP, PER 5% con snoop

El throughput es mayor en todos los casos con respecto a la tabla 3.3. Como vemos, el emisor no esta exento de realizar retransmisiones (en los casos 1,2 y 4) pero la mejora con el caso anterior es notable.

Para terminar, hemos probado con una tasa de error en paquetes del 10%. El simulador nos muestra una comunicación degradada cuando probamos con el proxy, mientras que sin él, el simulador no llega a crear los archivos con las trazas y no podemos ver que ocurre en la simulación. La tasa de error es demasiado elevada.

3.2. Evaluación con una fuente CBR

En este caso hemos querido probar con una fuente que inyecta tráfico a una tasa constante (Constant Bit Rate). El simulador de redes nos permite fijar, mediante la variable *rate_* la tasa que queramos probar.

3.2.1. Tasa de error en paquetes del 0%

Vamos a ver que pasa cuando estamos en el mejor de los casos y posteriormente añadiremos perdidas. En la tabla 3.5 podemos ver el throughput, el número de retransmisiones y las diferentes tasas a las que emite la fuente CBR. La fuente CBR trabaja a nivel de aplicación mientras que el throughput lo calculamos a nivel IP.

Tasa de emisión	Ancho de banda L2	Retransmisiones (emisor TCP)	Throughput (%)	Throughput (Mbps)
1000 kbps	10 Mbps	0	10,28	1,028
	0,024 Mbps	0	97,66	0,02344
512 kbps	10 Mbps	0	5,44	0,544
	0,024 Mbps	0	97,66	0,02344
256 kbps	10 Mbps	0	2,78	0,2781
	0,024 Mbps	0	97,66	0,02344
64 kbps	10 Mbps	0	0,73	0,073
	0,024 Mbps	0	97,66	0,02344

Tabla 3.5 Análisis de la comunicación CBR, PER 0%

El objetivo principal de estudiar el comportamiento de TCP en sistemas en los que no hay perdidas es simplemente tener un valor de referencia. Con estos valores de throughput que nos servirán de referencia podremos evaluar el comportamiento de TCP en los demás casos.

Las tasas con las que emitimos al utilizar este enlace de 24 kbps no afectan al tráfico cursado por el enlace. El tráfico que es capaz de cursar el enlace es el mismo, ya que la menor de todas las tasas es más del doble que el ancho de banda del enlace.

3.2.2. Tasa de error en paquetes del 1%, con y sin proxy snoop

Vamos aumentar la tasa de error en paquetes a 1%, en este caso, la tasa tampoco se puede considerar elevada, hay que tener en cuenta que estamos usando paquetes de 1040 bytes. En la tabla 3.6 se muestran los resultados y a continuación hacemos unos comentarios sobre las pruebas.

Tasa de emisión	Ancho de banda L2	ReTX sin snoop (emisor TCP)	ReTX con snoop (emisor TCP)	Throughput sin snoop (%)	Throughput con snoop (%)	Throughput sin snoop (Mbps)	Throughput con snoop (Mbps)
1000 kbps	10 Mbps	35	2	4,65	10,28	0,465	1,028
	0,024 Mbps	1	0	97,66	97,66	0,02344	0,02344
512 kbps	10 Mbps	38	1	4,93	5,30	0,493	0,530
	0,024 Mbps	3	1	97,66	97,66	0,02344	0,02344
256 kbps	10 Mbps	23	0	2,65	2,60	0,265	0,260
	0,024 Mbps	1	0	97,66	97,66	0,02344	0,02344
64 kbps	10 Mbps	8	0	6,39	6,45	0,639	0,645
	0,024 Mbps	3	1	97,66	97,66	0,02344	0,02344

Tabla 3.6 Análisis de la comunicación CBR, PER 1% con y sin snoop

Por un lado, vamos a comentar que ocurre con los enlaces de 10 Mbps comparando los resultados con y sin la presencia del proxy. En el primer caso, vemos que usando el proxy el throughput es más del doble que sin proxy y el número de retransmisiones es 36 veces menor. En el segundo caso, con una tasa de 512 Kbps, el throughput es ligeramente superior con proxy, sin embargo, el número de retransmisiones por parte del emisor TCP sin proxy es muy superior que sin él. El tercer caso es quizás el más conflictivo ya que el throughput es similar con y sin proxy. El proxy disminuye el número de retransmisiones pero también ligeramente el throughput. Al reducir el número de retransmisiones evitamos tráfico innecesario, ayudando así a evitar un posible congestión en la red.

En el cuarto y último caso, el proxy mejora tanto el número de retransmisiones como el throughput.

Respecto a los enlaces de 24 Kbps, hemos de decir que obtenemos el mismo throughput con y sin proxy. En el apartado 3.2.3 la tasa de error es superior y veremos la conveniencia o inconveniencia de utilización del proxy en los enlaces de 24 Kbps.

3.2.3. Tasa de error en paquetes del 2%, con y sin proxy snoop

En este caso con una tasa del 2%, nos acercamos un poco más a la tasa de error en paquetes con la que probaron los creadores del proxy snoop (PER 2,2-2,3%).

Como en el caso anterior, primero comentaremos los resultados con los enlaces de 10 Mbps y después con los enlaces de 24 Kbps.

Tasa de emisión	Ancho de banda L2	ReTX sin snoop (emisor TCP)	ReTX con snoop (emisor TCP)	Throughput sin snoop (%)	Throughput con snoop (%)	Throughput sin snoop (Mbps)	Throughput con snoop (Mbps)
1000 kbps	10 Mbps	44	5	3,28	10,31	0,328	1,031
	0,024 Mbps	1	0	97,66	97,66	0,02344	0,02344
512 kbps	10 Mbps	48	3	3,02	5,32	0,3027	0,532
	0,024 Mbps	4	0	96,85	97,66	0,02324	0,02344
256 kbps	10 Mbps	34	2	2,237	2,604	0,2237	0,2604
	0,024 Mbps	5	0	97,66	98,24	0,02344	0,02358
64 kbps	10 Mbps	21	0	0,521	0,638	0,0521	0,0638
	0,024 Mbps	5	1	97,66	97,66	0,02344	0,02344

Tabla 3.7 Análisis de la comunicación CBR, PER 2% con y sin snoop

En la tabla 3.7 podemos ver que en los enlaces de 10 Mbps el proxy disminuye el número de retransmisiones en todos los casos. El throughput también es superior en todos los casos.

Por último, en los enlaces de 24 Kbps podemos aconsejar la utilización del proxy en todos los casos ya que siempre disminuye el número de retransmisiones y de esta manera evitamos tráfico innecesario.

Debido a que el proxy no aumenta considerablemente el throughput en determinadas situaciones puede ser de gran ayuda su utilización siempre y cuando el sistema en el que estemos pueda tener variaciones significativas de velocidad.

3.2.4. Tasa de error en paquetes del 5%, con y sin proxy snoop

En este caso la tasa de error es realmente alta. Hemos estudiado el comportamiento del proxy con las mismas tasas de emisión en la fuente CBR.

En la tabla 3.6 podemos ver cómo se degrada la eficiencia de la comunicación cuando introducimos esta tasa de error. A continuación comentamos los resultados.

Tasa de emisión	Ancho de banda L2	ReTX sin snoop (emisor TCP)	ReTX con snoop (emisor TCP)	Throughput sin snoop (%)	Throughput con snoop (%)	Throughput sin snoop (Mbps)	Throughput con snoop (Mbps)
1000 kbps	10 Mbps	72	8	1,218	6,42	0,1218	0,6420
	0,024 Mbps	3	0	96,59	95,19	0,02318	0,02284
512 kbps	10 Mbps	81	7	1,413	5,17	0,1413	0,517
	0,024 Mbps	9	2	91,88	97,66	0,02215	0,02344
256 kbps	10 Mbps	69	8	1,262	2,59	0,1262	0,2598
	0,024 Mbps	12	1	87,84	76,86	0,02108	0,0184
64 kbps	10 Mbps	29	1	0,613	0,645	0,06135	0,0645
	0,024 Mbps	8	1	86,11	95,93	0,02066	0,0230

Tabla 3.6 Análisis de la comunicación CBR, PER 5% con y sin snoop

En el primer caso y con los enlaces de 10 Mbps, el throughput con snoop es casi seis veces mayor mientras que el número de retransmisiones es siete veces inferior. En el segundo y tercer caso ocurre lo mismo, la presencia del proxy es más que eficiente ya que mejora el throughput del sistema.

En cambio, en el último caso dónde probamos con una tasa de 64 Kbps el throughput es ligeramente superior (con proxy) mientras que el número de retransmisiones es cincuenta veces menor. En este caso si es eficiente el uso del proxy ya que el throughput es mayor.

Por último, en la primera y tercera prueba con los enlaces a 24 Kbps podemos ver como la presencia del proxy disminuye el número de retransmisiones pero también disminuye el throughput ligeramente con respecto a la misma situación sin proxy. En cambio, en los demás casos el proxy diminuye el número de retransmisiones y aumenta el throughput en comparación con la misma situación sin proxy.

3.3. Evaluación con Emulación de Terminal, TELNET

Telnet es un protocolo que se utiliza, generalmente, para iniciar una sesión en una máquina remota. Así, desde nuestro ordenador particular podemos, por ejemplo, iniciar una sesión en el ordenador que tenemos en el trabajo.

El simulador NS-2 nos permite usar este protocolo a nivel de aplicación usando TCP a nivel de transporte. Una vez definimos el agente, tenemos la posibilidad de variar el tiempo entre generación de paquetes mediante la variable *interval_*.

Si la variable *interval_* no vale cero, el tiempo entre generación de paquetes seguirá una distribución exponencial de media igual al valor de la variable *interval_.* Si vale cero, el tiempo entre generación de paquetes sigue la distribución de la librería *tcplib-telnet.cc.*

3.3.1. Evaluación con la distribución *tcplib-telnet.cc*

La variable *interval_* irá tomando los valores, en segundos, que aparecen separados por comas en la librería *tcplib-telnet.cc.* Estos valores los podríamos modificar a conveniencia y adaptar un poco más la simulación a nuestras necesidades. Si modificamos el contenido de este archivo hemos de compilar el simulador para que se apliquen los cambios. En la figura 3.6 podemos ver el inicio y el final del archivo *tcplib-telnet.cc* .

```
static double tcplib_telnet[] = {

0.000606425,0.000617809,0.000629598,0.000643143,0.000643628,
0.00064655,0.000664476,0.000667254,0.000672394,0.000673758,
0.000682008,0.000687469,0.000691018,0.000693969,0.000724883,
0.000740577,0.000742143,0.000748767,0.000755347,0.000755526,

59.171453125,59.188242188,59.194550781,59.220753906,59.229398438,
59.2781875,59.29696875,59.360042969,59.389535156,59.444675781,
59.484121094,59.552898438,59.625878906,59.702589844,59.755855469,
59.859945312,59.935503906,59.962140625,59.981171875,64.415308594,
89.67025,115.543304688,133.35125,137.70978125,209.92121875
```

Fig. 3.6 Archivo *tcplib-telnet.cc*

En este caso el throughput es muy inferior a los casos anteriores, ya que estamos generando paquetes en instantes determinados por la variable *interval_*. La tabla 3.7 resume los resultados de las pruebas.

Ancho de banda L1	Ancho de banda L2	ReTX sin snoop (emisor TCP)	ReTX con snoop (emisor TCP)	Throughput sin snoop (%)	Throughput con snoop (%)	Throughput sin snoop (Mbps)	Throughput con snoop (Mbps)
10 Mbps	10 Mbps	21	15	0,1373	0,149	0,01373	0,01492
10 Mbps	2 Mbps	21	15	0,672	0,74	0,01345	0,01478
10 Mbps	0,3 Mbps	20	14	4,07	4,88	0,01221	0,01464
10 Mbps	0,024 Mbps	11	7	53,17	48,55	0,01276	0,01165

Tabla 3.7 Análisis de la comunicación TELNET, PER 5% con y sin snoop

La variable *interval_* empieza tomando valores muy pequeños para a continuación ir aumentando su valor. Por lo tanto el tráfico que inyectamos al enlace L1 será menor. Es por esto que el tráfico que estamos generando va a ser a ráfagas y de poca intensidad.

Como en casos anteriores, hemos ido variando el ancho de banda del enlace que conecta al cliente (L2) y vemos como el proxy mejora el throughput en las tres primeras pruebas. En cambio, en el último caso el proxy consigue disminuir el número de retransmisiones del emisor pero el throughput es inferior que sin proxy.

Por último, en la figura 3.7 podemos ver cómo los paquetes se generan poco a poco, la ventana de congestión nunca llega a crecer demasiado y el throughput se ve afectado por el intervalo que fijamos a través de la librería *tcplib-telnet.cc*.

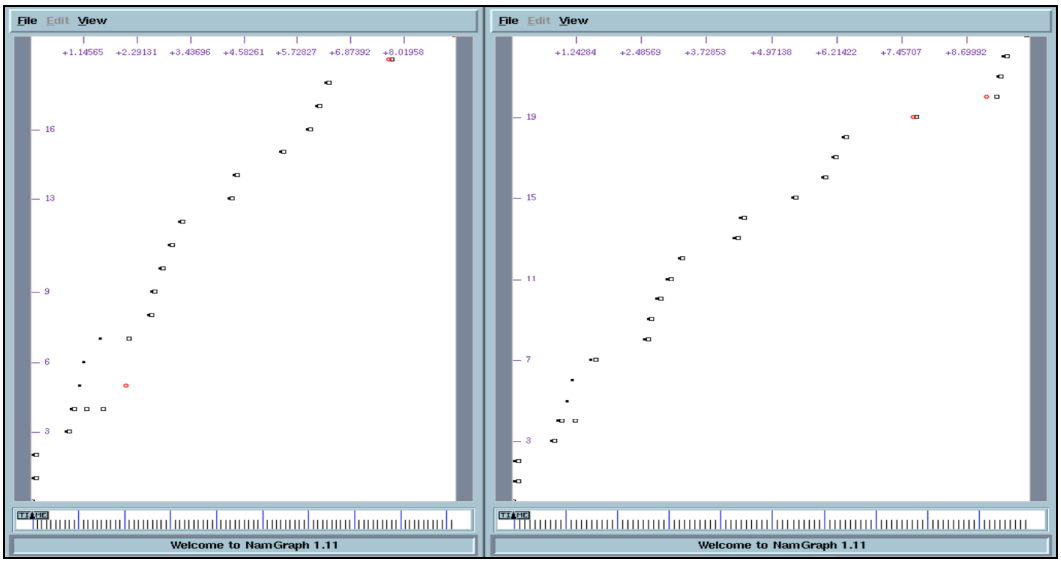

Fig. 3.7 Diagrama tiempo-secuencia para una sesión TELNET, PER 5%

La imagen de la izquierda corresponde con una sesión Telnet, con una tasa de error en paquetes del 5% (en el enlace inalámbrico) y la gráfica de la derecha corresponde con una sesión Telnet con la misma tasa de error y el proxy snoop. En este caso la librería que hemos usado no nos ha sido de gran ayuda ya que apenas hay tráfico de paquetes. En los siguientes estudios modificaremos la variable *interval_* para generar más tráfico y poder analizar mejor la comunicación.

3.3.2. Evaluación con una distribución exponencial

En este caso hemos fijado de manera constante, durante toda la simulación el valor de la variable *interval_*. La variable sigue ahora una distribución exponencial cuyo valor medio es igual al valor al que iniciemos la variable.

3.3.2.1. Tasa de error en paquetes del 1, 2 y 5%, con y sin proxy snoop

En las pruebas realizadas que mostramos a continuación se ve como el proxy no consigue aumentar el throughput en todos los casos. La tabla 3.8 resume los resultados de las pruebas realizadas y a continuación comentamos los resultados.

interval_ (seg.)	Ancho de banda L2	ReTX sin snoop (emisor TCP)	ReTX con snoop (emisor TCP)	Throughput sin snoop (%)	Throughput con snoop (%)	Throughput sin snoop (Mbps)	Throughput con snoop (Mbps)
0,0005	10 Mbps	11	0	9,70	25,3	0,971	2,53
	2 Mbps	14	0	43,45	89,58	0,869	1,792
	0,3 Mbps	2	0	94,43	95,68	0,283	0,2870
	0,024 Mbps	1	0	99,4	98,82	0,02385	0,02371
0,005	10 Mbps	12	1	10,77	16,82	1,077	1,682
	2 Mbps	10	0	49,04	83,23	0,981	1,665
	0,3 Mbps	2	0	98,91	96,33	0,297	0,289
	0,024 Mbps	0	0	98,24	98,24	0,02357	0,02357
0,05	10 Mbps	1	0	1,654	1,643	0,1654	0,1643
	2 Mbps	1	0	8,26	8,22	0,1652	0,1644
	0,3 Mbps	1	1	54,54	55,14	0,1636	0,1654
	0,024 Mbps	0	0	98,24	98,24	0,02357	0,02357

Tabla 3.8 Análisis de la comunicación TELNET, PER 1% con y sin snoop

Las retransmisiones que se producen cuando usamos el proxy normalmente son producidas por la expiración del temporizador del emisor TCP (time out), por lo tanto en algunos casos, dónde la tasa de emisión de la fuente no es elevada o dónde el enlace que conecta al cliente es de baja velocidad el proxy disminuye el número de retransmisiones pero también el throughput ya que hay más tiempo de inactividad provocado por las retransmisiones locales que realiza el proxy.

En la tabla 3.9 podemos ver algo parecido al caso anterior el proxy no siempre mejora las prestaciones del sistema, en este caso la eficiencia del protocolo TCP no siempre es superior. Si la diferencia entre el número de retransmisiones del emisor TCP con y sin proxy no es significativo veremos que el proxy disminuirá también el throughput al disminuir el número de retransmisiones del emisor.

interval_ (seg.)	Ancho de banda L2	ReTX sin snoop (emisor TCP)	ReTX con snoop (emisor TCP)	Throughput sin snoop (%)	Throughput con snoop (%)	Throughput sin snoop (Mbps)	Throughput con snoop (Mbps)
0,0005	10 Mbps	14	0	5,59	22,1	0,559	2,21
	2 Mbps	11	0	20,98	78,52	0,4196	1,57
	0,3 Mbps	7	0	86,16	91,93	0,2584	0,2758
	0,024 Mbps	1	0	99,4	98,24	0,02385	0,02357
0,005	10 Mbps	13	1	5,806	16,8	0,5806	1,68
	2 Mbps	13	0	28,41	81,09	0,568	1,62
	0,3 Mbps	10	0	78,94	89,81	0,2368	0,2694
	0,024 Mbps	1	0	97,08	97,66	0,0233	0,02344
0,05	10 Mbps	6	0	1,654	1,650	0,1654	0,1650
	2 Mbps	6	0	8,09	8,24	0,1618	0,1648
	0,3 Mbps	6	1	55,51	53,98	0,1665	0,1619
	0,024 Mbps	1	0	99,4	98,82	0,02385	0,02371

Tabla 3.9 Análisis de la comunicación TELNET, PER 2% con y sin snoop

En este último caso, la tasa de error que introducimos es muy elevada y el proxy consigue en casi todos los casos reducir el número retransmisiones y aumentar el throughput. De aquí se puede deducir que cuanto mayor sea la tasa de error en cualquier sistema de perdidas más efectivo va ser el uso del proxy.

En cambio como hemos comentado antes cuando la tasa de emisión no es elevada en comparación con el ancho de banda de los enlaces, el proxy puede no mejorar las prestaciones del sistema. En la tabla 3.10 mostramos los resultados de las pruebas realizadas con el proxy.

interval_ (seg.)	Ancho de banda L2	ReTX sin snoop (emisor TCP)	ReTX con snoop (emisor TCP)	Throughput sin snoop (%)	Throughput con snoop (%)	Throughput sin snoop (Mbps)	Throughput con snoop (Mbps)
0,0005	10 Mbps	16	3	1,97	11,31	0,1974	1,131
	2 Mbps	14	1	12,26	56,03	0,2453	1,121
	0,3 Mbps	11	0	54,08	79,59	0,1622	0,2387
	0,024 Mbps	5	0	53,17	97,08	0,01276	0,0233
0,005	10 Mbps	16	0	2,80	12,30	0,2808	1,23
	2 Mbps	15	2	9,99	55,09	0,1999	1,10
	0,3 Mbps	9	0	56,07	76,86	0,1682	0,231
	0,024 Mbps	3	0	78,02	93,62	0,0187	0,02247
0,05	10 Mbps	12	0	1,67	1,60	0,1674	0,1601
	2 Mbps	9	0	7,44	8	0,1488	0,1600
	0,3 Mbps	7	0	49,92	52,83	0,1497	0,1585
	0,024 Mbps	3	0	73,98	95,93	0,0177	0,02302

Tabla 3.10 Análisis de la comunicación TELNET, PER 5% con y sin snoop

La figura 3.8 es un diagrama tiempo secuencia que corresponde con la primera prueba realizada dónde los enlaces L1 y L2 tiene el mismo ancho de banda.

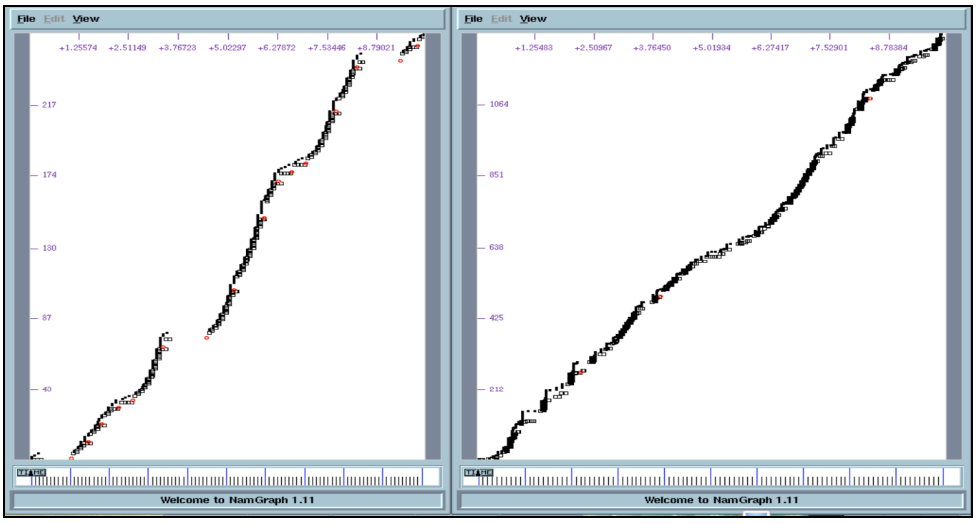

Fig. 3.8 Telnet con y sin snoop, PER 5% exponencial

En la imagen de la izquierda, las retransmisiones por parte del emisor TCP son continuas, lo que impide un aumento de la ventana de congestión y de la eficiencia de la comunicación.

En la imagen de la derecha, con la intervención del proxy tenemos el proxy snoop, vemos que el emisor no esta exento de realizar retransmisiones (pero el número de estas es inferior) y el throughput es más de cinco veces mayor que sin proxy.

3.4. Evaluación con FTP y el módulo fullTCP

En el simulador NS-2 hay disponible un modelo que implementa lo que sería una versión mas realista de TCP, en el sentido que tiene capacidad de generar segmentos TCP y ACKs en un mismo nodo. Este nuevo agente esta aún bajo desarrollo. Estas son algunas de las diferencias entre ambas versiones:

- Hay inicio y cierre de conexión, es decir se intercambian paquetes con los flags SYN/FIN, en FullTCP.

- Transmisión de datos en ambos sentidos, los extremos pueden enviar segmentos TCP y reconocimientos (ACK).

- El número de secuencia se especifica en unidades de bytes y podemos modificar a través de TCL y mediante la variable iss_ (Initial Send Sequence number), el número que iniciará los números de secuencia de la conexión TCP.

Hemos probado la versión FullTCP en un sólo sentido. Para conseguir esto, hemos adjuntado el protocolo en el nodo emisor y receptor pero sólo hemos utilizado un generador de tráfico (servidor FTP) en el emisor TCP. Al realizar las pruebas, hemos descubierto que el proxy snoop no interviene en la simulación, esto es así porque el módulo *snoop.cc* está programado para el módulo en un único sentido TCP.

3.4.1. Propuesta de adaptación de snoop a FullTCP

El módulo fullTCP aumenta el número de secuencia en unidades de bytes en función del tamaño de un paquete y el modulo snoop esta programado teniendo en cuenta el protocolo TCP implementado por el simulador.

Este protocolo TCP, en un sentido, aumenta los números de secuencia de uno en uno y el proxy snoop esta programado para eliminar segmentos duplicado sumando de uno en uno. Este es el motivo por el que el proxy no es capaz de monitorizar la conexión TCP y actuar debidamente cuando se producen perdidas.

En la figura 3.10, hay un fragmento de código dónde el proxy una vez detecta un ack duplicado, comprueba el número de secuencia del primer paquete en el buffer sin reconocer. Si este, es mayor que el número de secuencia del ack actual más uno, el proxy deja pasar el ack.

```
Int  Snoop::snoop_ack(Packet *p)
{
      Packet *pkt;

      int ack = hdr_tcp::access(p)->seqno();
      bool cong_ ;

      if (fstate_ & SNOOP_CLOSED || lastAck_ > ack)
            return SNOOP_PROPAGATE;   // dejar pasar el ack

      if (lastAck_ == ack) {
            // Es un ack duplicado
            pkt = pkts_[buftail_];
            // no tenemos el pkt, lo dejamos pasar
            if (pkt == 0)
                  return SNOOP_PROPAGATE;

            hdr_snoop *sh = hdr_snoop::access(pkt);
            if (pkt == 0 || sh->seqno() > ack + 1)
                        return SNOOP_PROPAGATE;
```

Fig. 3.10 Función para el procesado de acks

Nuestra propuesta consiste en sumar al número de secuencia del ack actual el tamaño en bytes del segmento TCP. Para conseguir esto creamos una nueva variable *segsize_*, esta variable la hacemos visible a nivel TCL para poder fijar su valor tanto en el módulo fullTCP como en el módulo proxy snoop.

Con la llegada del segundo ACK duplicado, el proxy verificará que el número de secuencia de ese ACK más el tamaño de un paquete en unidades de bytes, es superior que el número de secuencia del primer segmento sin reconocer que se encuentra en la caché. Entonces el proxy eliminará el ACK duplicado en lugar de dejarlo pasar como hacía anteriormente.

3.5. Conclusiones sobre las pruebas

A lo largo de este documento hemos ido redactando algunas conclusiones a medida que hacíamos las pruebas y analizábamos los resultados En este apartado vamos a comentar de forma general los resultados de todas las pruebas. El escenario de las pruebas básicamente se han caracterizado por el generador de tráfico usado, el ancho de banda del enlace inalámbrico y la tasa de error en este.

No hemos podido analizar el proxy con todos los generadores de tráfico del simulador NS-2. Por ejemplo, las fuentes on/off de distribución exponencial o pareto generan unas trazas tras la simulación que debido al identificador que usan como protocolo no es posible analizar. El filtro que usamos genera gráficas tiempo secuencia a partir de conexiones TCP. Además, al analizar las trazas de la simulación descubrimos que los paquetes que se envían con este tipo de fuentes no llevan número de secuencia o al menos no queda especificado en las trazas. Por lo tanto no podemos generar este tipo de gráficas con estos generadores de tráfico. En la figura 3.10 podemos ver las dos primeras y últimas trazas de una simulación con una fuente exponencial On/Off.

```
+    -t 1.23499272150262 -s 3 -d 2 -p exp -e 40 -c 0 -i 0 -a 0 -x {3.0 4.0 0 ------- null}

-    -t 1.23499272150262 -s 3 -d 2 -p exp -e 40 -c 0 -i 0 -a 0 -x {3.0 4.0 0 ------- null}

h -t 6.30721512150263 -s 2 -d 1 -p exp -e 1040 -c 0 -i 187 -a 0 -x {3.0 4.0 -1 ---null}

r  -t 6.30864072150263 -s 1 -d 0 -p exp -e 1040 -c 0 -i 187 -a 0 -x {3.0 4.0 0 ----null}
```

Fig. 3.10 Tazas con un generador de tráfico exponencial On/Off

En las pruebas realizadas con un servidor FTP el proxy snoop mejora en todos los casos el throughput de la comunicación. Hemos visto que el protocolo TCP se ha ido adaptando a los diferentes enlaces que hemos ido estudiando. Al disminuir el ancho de banda de los enlaces también ha disminuido el número de retransmisiones del emisor TCP ya que la probabilidad de que se produzcan perdidas es menor al haber menos tráfico.

El proxy ha aumentado el throughput tanto en los enlaces de 10 Mbps como en los de 24 Kbps. En la figura 3.11 se puede ver como aumenta el throughput en presencia del proxy, para una comunicación con una tasa de error en paquetes del 2%.

Por otro lado, en las pruebas realizadas con una fuente a tasa constante (CBR) y el protocolo TELNET, el proxy no mejora siempre el throughput.

Por ejemplo, en las pruebas dónde el enlace que conecta el nodo 0 y el nodo 4 es un enlace de baja velocidad (24 Kbps), se puede ver que la presencia del proxy no mejora el throughput de la comunicación TCP.

Como el proxy elimina de la red todos los reconocimientos duplicados, siempre conseguimos que el número de retransmisiones por parte del emisor TCP se reduzca. De esta manera el proxy disminuye el tráfico innecesario de la red.

Este funcionamiento permite que el tiempo de convergencia frente a perdidas sea inferior cuando trabajamos con enlaces de gran ancho de banda dónde el proxy si consigue aumentar el throughput.

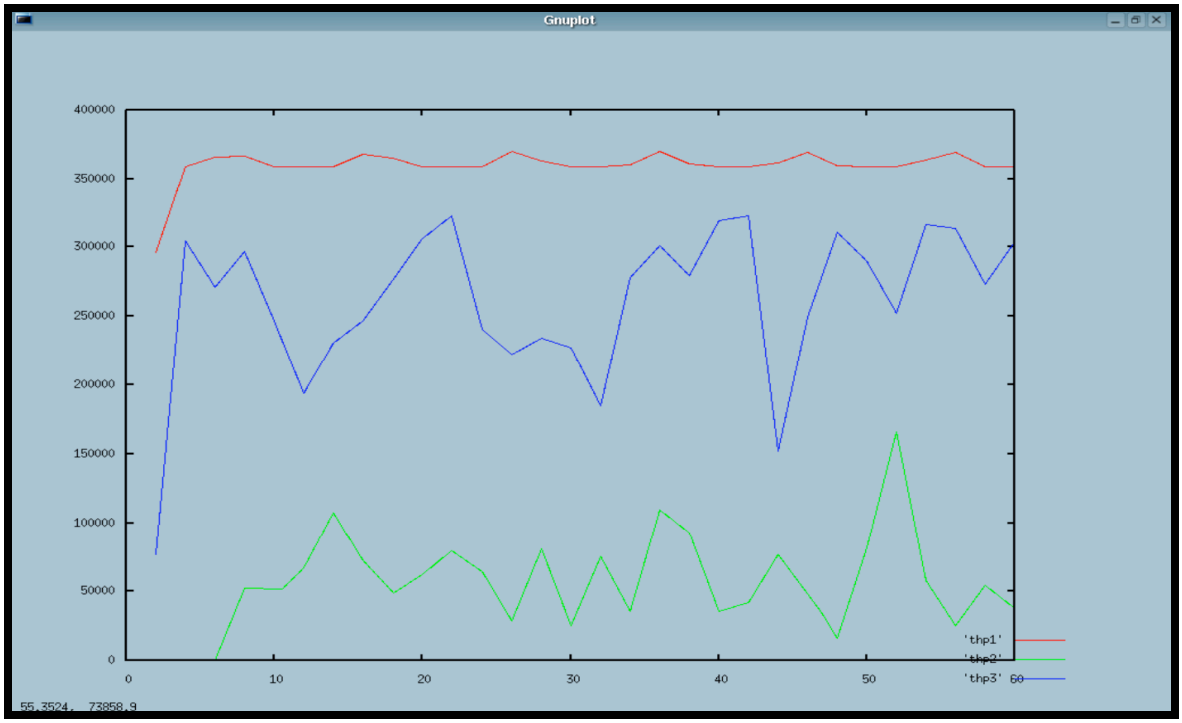

Fig. 3.11 Línea roja sin errores, azul PER 2% con snoop, verde PER 2% sin snoop

En cambio, en los enlaces de baja velocidad que hemos probado estos mecanismo reducen el número de retransmisiones del emisor pero el tiempo de convergencia del proxy es mayor que en la misma situación sin proxy.

En la figura 3.12 se puede ver que el tráfico no es continuo debido a la tasa de generación de paquetes que usamos con TELNET. La línea azul corresponde a una simulación con proxy y la verde sin proxy. En media el throughput sin proxy es superior que con proxy.

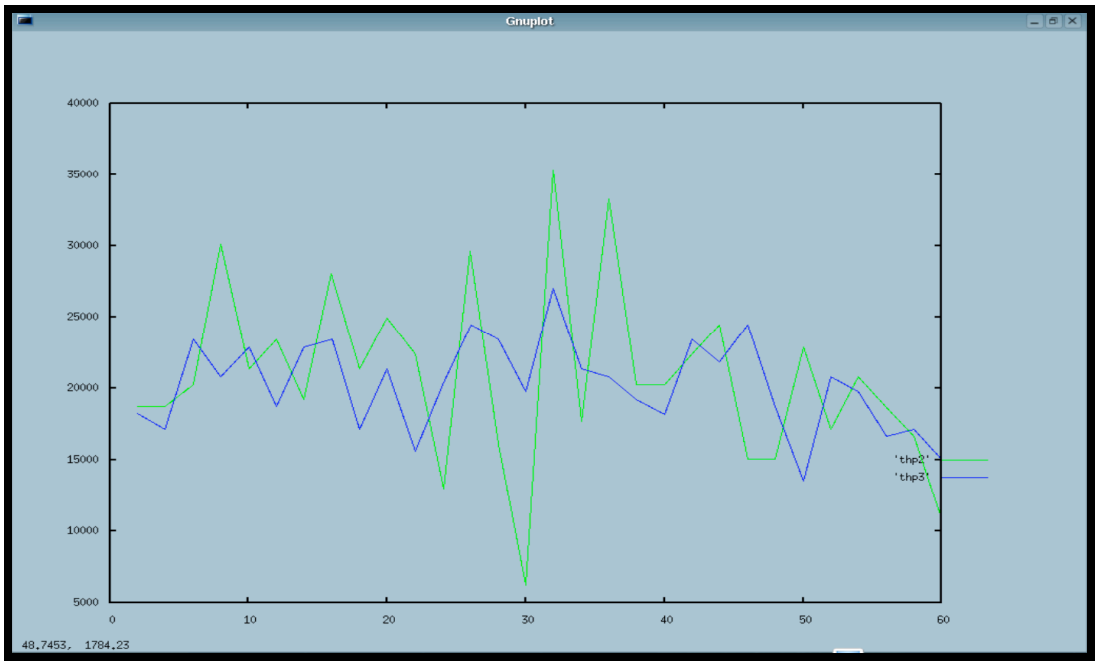

Fig. 3.12 Throughput, línea azul PER 2% con snoop, verde PER 2% sin snoop

En la figura 3.13 podemos ver como el proxy disminuye notablemente el número de retransmisiones del emisor TCP para distintas tasas de error.

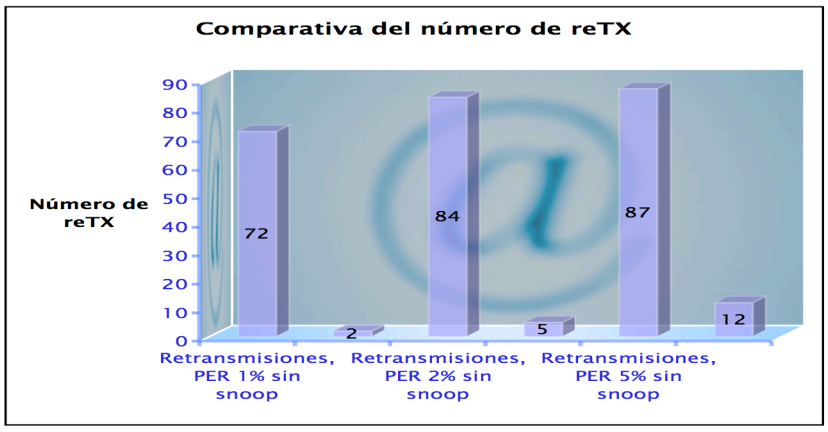

Fig. 3.13 Comparativa de las retransmisiones

El proxy en determinadas ocasiones reduce el número de retransmisiones pero no consigue aumentar la eficiencia. Este comportamiento tiene lugar cuando trabajamos con poco tráfico y a ráfagas y cuando el proxy funciona en enlaces de baja velocidad. Aunque el comportamiento del proxy se puede considerar inocuo en muchos de estos casos, en cuanto a la eficiencia que se obtiene, siempre puede ser muy útil si se producen variaciones de velocidad en el sistema en el que trabajamos o simplemente para evitar una posible congestión en la red provocada por los reconocimientos duplicados.

Si tuviéramos que analizar como se comportaría el proxy en una situación real dónde supiéramos las características y detalles de la topología, una buena aproximación seria hacer como hemos hecho en este trabajo, con simulaciones. El proxy ha conseguido mejorar el throughput en más del 87% de las pruebas que hemos realizado y en los casos que el throughput es ligeramente inferior, evita tráfico innecesario en la red.

Para terminar, el proxy introduce un tiempo de proceso extra en cada estación base donde esté situado. Aunque en el simulador el tiempo de proceso es el mismo esté o no el proxy, en una situación real hemos de tener este factor en cuenta ya que el retardo total de la comunicación es ligeramente superior.

CAPÍTULO 4. Una nueva versión del proxy snoop

El proxy snoop es eficiente en sistemas de pérdidas y enlaces inalámbricos dónde la tasa de error es elevada. En cambio, en unos estudios existentes [2] quedó en evidencia su funcionamiento y se añadieron algunas mejoras. El objetivo, es plasmar estas nuevas funcionalidades en un nuevo módulo del simulador NS-2.

4.1. Características de snoop adaptado

El nuevo módulo pretende mejorar la eficiencia de una comunicación en un entorno GPRS. Para resolver los problemas derivados de los altos y variables retardos, los temporizadores del proxy están adaptados a los RTT's convencionales de un enlace GPRS. En la figura 4.1 [2] puede observarse un esquema del funcionamiento del proxy snoop adaptado a GPRS.

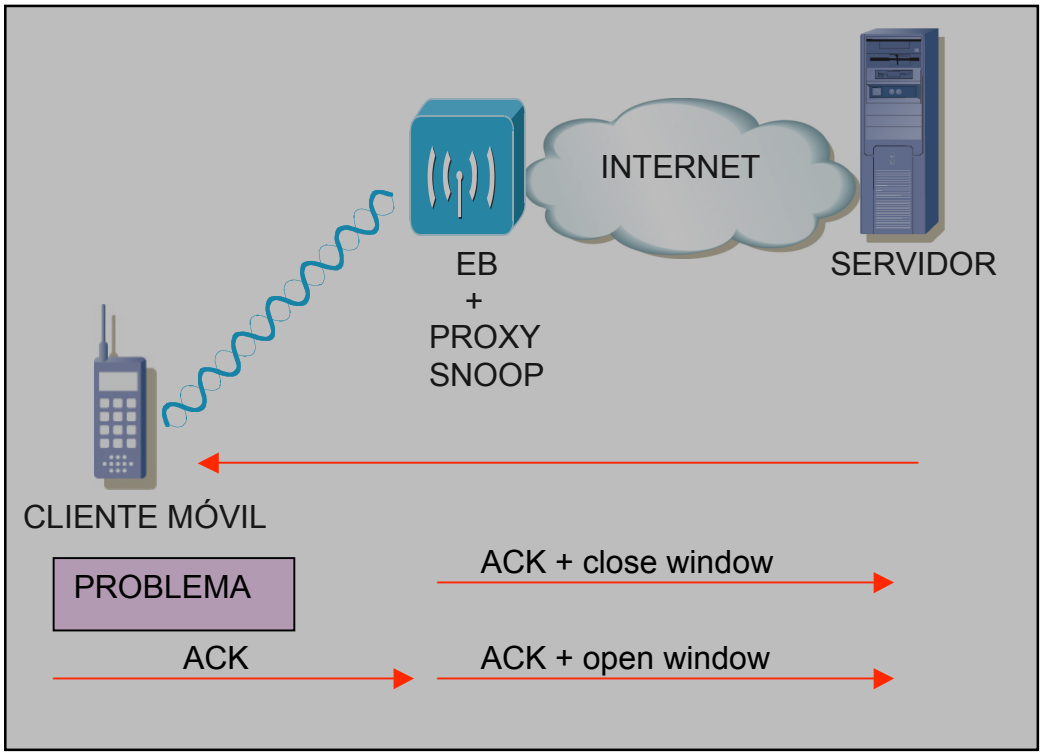

Fig 4.1 Funcionamiento de snoop adaptado a GPRS

Conjuntamente a cada retransmisión local, snoop adaptado lleva a cabo un reconocimiento local, usualmente acompañado de un cierre de la ventana de congestión, de esta manera se evita que expire el temporizador de retransmisión del servidor. Una vez que recibe el reconocimiento real del cliente móvil, se recupera la transmisión habitual del servidor como si no hubiera ocurrido ningún problema. El cierre de la ventana de congestión es para evitar que el servidor pueda saturar la caché del snoop. Además, se

guarda el sincronismo entre los paquetes que envía el emisor y las confirmaciones del cliente móvil.

Por otro lado, la elección del temporizador local de snoop es un factor a tener en cuenta en la eficiencia de la transmisión. Un RTO_{snoop} local demasiado pequeño comportará retransmisiones innecesarias por parte del proxy; por el contrario, si el temporizador es mayor que el RTO del emisor TCP no podrá evitarse la aplicación de los algoritmos de control de congestión. Nosotros hemos considerado un RTO_{snoop} local fijo para cada conexión. Fijo, en el sentido que será constante durante la simulación pero configurable para distintos escenarios.

4.1.1. La función closeWin

Cuando el proxy detecte que hay un problema, ya sea por expiración del temporizador o la detección de reconocimientos duplicados, enviará un reconocimiento con el valor 0 en el campo ventana de la cabecera TCP. Así, el emisor cerrará su ventana de congestión, y evitaremos que la caché del proxy se sature.

En la figura 4.2 puede observarse esta función. En el anexo A de este documento esta la versión completa del código.

```
void
Snoop::closeWin()
{
        //Creamos el pkt que cerrará la ventana
        Packet* pkt = allocpkt();
        hdr_tcp *tcp = hdr_tcp::access(pkt);
        hdr_cmn::access(pkt)->ptype() == PT_ACK;

        //Ponemos el número de secuencia del primer pkt del buffer sin reconocer
        tcp->seqno() = hdr_snoop::access(pkts_[buftail_])->seqno());

        // especificamos la direccion, en este caso down
        hdr_cmn *ch = HDR_CMN(pkt);
        hdr_tcp::access(pkt)->wnd_ = 0;//cerramos la ventana
        ch->direction() = hdr_cmn::DOWN;
        parent_->sendDown(p);     // vector a LLSnoop's sendto()
}
```

Fig. 4.2 Método closeWin

El número de secuencia que le ponemos a este nuevo segmento TCP es el que corresponde al primer segmento de la caché sin reconocer. Así el emisor TCP comprobará que el segmento esta en secuencia y sólo cerrará la ventana de congestión si el segmento TCP así lo indica en la cabecera.

4.1.2. La función openWin

En este caso vamos abrir la ventana de congestión. Cuando el proxy retransmita el segmento solicitado por el receptor, éste enviará un reconocimiento que abrirá de nuevo la ventana de congestión y por lo tanto se enviran nuevos segmentos TCP y la comunicación continuará como si nada hubiera pasado.

En la figura 4.3 podemos ver esta función programada en C++, la versión completa del código se encuentra en el anexo A de este documento.

```
void
Snoop::openWin(int seqno_)
{
        //Creamos el pkt que abrirá la ventana
        Packet* pkt = allocpkt();
        hdr_tcp *tcp = hdr_tcp::access(pkt);
        //Añadimos el número de secuencia del ack que estamos analizando
        tcp->seqno() = seqno_;
        // especificamos la dirección, en este caso down
        hdr_cmn *ch = HDR_CMN(pkt);
        hdr_tcp::access(pkt)->wnd_ = 1; //abrimos la ventana
        ch->direction() = hdr_cmn::DOWN;
        parent_->sendDown(p);     // vector a LLSnoop's sendto()

}
```

Fig. 4.3 Método openWin

El segmento TCP que envía el cliente permite al emisor enviar nuevos segmentos. Nosotros hacemos una replica de este segmento y lo enviamos al emisor TCP

4.1.3. La función time out

La función esta regida por una variable, *rto_* que nosotros podremos modificar a conveniencia en cada simulación. Eso si, el valor rto_ permanecerá constante durante toda la simulación.

```
inline double timeout() {

            return max(rto_, snoopTick_);//devolvemos nuestro rto_ fijado desde OTcl
        }
```

Fig. 4.4 Método timeout

En la figura 4.4 vemos como la función devuelve el máximo valor entre el rto_ y snoopTick_. Esta última variable va aumentando en fracciones de 100 ms igual que en tcpTick_ y es el mínimo valor del temporizador de retransmisiones de snoop.

4.2. Limitaciones del simulador y propuesta de adaptación

En una conexión TCP real, cada vez que el emisor TCP recibe nuevos reconocimientos aumenta su ventana de transmisión en función de lo que le permite enviar el receptor (ventana ofrecida). En el simulador NS-2 la ventana ofrecida se fija en el emisor TCP y permanece constante durante toda la simulación. Además, esta variable (*window_*) la puede fijar el usuario en el archivo de simulación, programado en OTcl, después de definir el agente de transporte.

Es por esto que las estructuras de datos que definen los campos de una cabecera TCP, en el simulador, no contempla el campo window y por tanto no podremos ni cerrar ni abrir la ventana ofrecida.

Para solventar este problema hemos estudiado los códigos que corresponden con el protocolo TCP (tcp.cc) y la versión Reno de TCP (tcp-reno.cc). La versión completa de Reno TCP se encuentra en el anexo A, este código funciona conjuntamente con tcp.cc ya que desde el primer código se hacen llamadas a funciones que están en tcp.cc, es decir el archivo tcp-reno.cc sólo incluye las funcionalidades nuevas de esta versión.

Es en el archivo tcp-reno.cc donde hemos añadido nuevas funcionalidades para que el emisor pueda reaccionar frente a la llegada de un segmento con el campo window.

Por un lado, hemos añadido el campo wnd_ en las estructuras de datos que definen la cabecera TCP. Estas estructuras están en el archivo tcp.h que es usado por todas las versiones de TCP que se implementan en el simulador.

Por otro lado, en tcp-reno.cc hemos modificado la función encargada de recibir y comprobar los paquetes ACK. En esta función se comprueba también el campo ventana. En la figura 4.5 se puede ver el código de esta función implementada en el modulo tcp-reno.cc.

```
void RenoTcpAgent::recv(Packet *pkt, Handler*)
{
        hdr_tcp *tcph = hdr_tcp::access(pkt);
        int valid_ack = 0;
       if (qs_approved_ == 1 && tcph->seqno() > last_ack_)
       //Al salir de QuickStart, reducimos la ventana de congestion al tamaño que fue
realmente usado
                endQuickStart();
       if (qs_requested_ == 1)
               processQuickStart(pkt);
#ifdef notdef
        if (pkt->type_ != PT_ACK) {
                fprintf(stderr,
                        "ns: confiuration error: tcp received non-ack\n");
                exit(1);
```

```
// W.N.: comprobar si esto es de alguna encarnacion previa
        if (tcph->ts() < lastreset_) {
                // Quitar pkt y no hacemos nada
                Packet::free(pkt);
                return;
        }
         ++nackpack_;
         ts_peer_ = tcph->ts();

         if (hdr_flags::access(pkt)->ecnecho() && ecn_)
                 ecn(tcph->seqno());
         recv_helper(pkt);
         recv_frto_helper(pkt);
         if (tcph->seqno() > last_ack_) {
                 if (last_cwnd_action_ == CWND_ACTION_DUPACK || hdr_tcp::access(pkt)-
wnd_)
                         last_cwnd_action_ == CWND_ACTION_EXITED;
                 dupwnd_ = 0;
                 recv_newack_helper(pkt);
                 if (last_ack_ == 0 && delay_growth_) {
                         cwnd_ = initial_window();
                 }
         } else if (tcph->seqno() == last_ack_) {
                 if (hdr_flags::access(pkt)->eln_ && eln_) {
                         tcp_eln(pkt);
                         return;
                 }
                 if (++dupacks_ == numdupacks_) {
                         dupack_action();
                         if (!exitFastRetrans_)
                                 dupwnd_ = numdupacks_;
                 } else if (dupacks_ > numdupacks_ && (!exitFastRetrans_
                     || last_cwnd_action_ == CWND_ACTION_DUPACK )) {
                         ++dupwnd_;      // fast recovery

                 // si es el primer ack dup, y el campo ventana es nulo, cerramos la ventana de
congestion.
                 } else if (dupacks_ < numdupacks_ && singledup_ && !hdr_tcp::access(pkt)-
>wnd_) {
                         // send_one();
                         last_cwnd_action_ == CWND_ACTION_DUPACK;
                 }
         }
         if (tcph->seqno() >= last_ack_)
                 // Comprobar que es un ACK valido. Sugerido por Mark Allman.
                 valid_ack = 1;
         Packet::free(pkt);
#ifdef notyet
         if (trace_)
                 plot();
#endif
         //intentamos enviar más datos
         if (valid_ack || aggressive_maxburst_)
                 if (dupacks_ == 0 || dupacks_ > numdupacks_ - 1)
                         send_much(0, 0, maxburst_);
}
```

Fig. 4.5 Método recv() modificado

Lo primero de todo, es comprobar que el segmento que llega es un ack. A continuación comprobamos el número de secuencia y de ahí averiguamos si esta en secuencia o por el contrario es un ack duplicado.

Si esta en secuencia hemos de comprobar el estado anterior del sistema y el campo ventana de la cabecera TCP. Si uno de los dos es cierto el sistema saldrá del estado "acción congestión" y la comunicación continuará.

Si el paquetes es un ack duplicado, y el número de secuencia del ack duplicados que ha llegado al sistema es menor que el numero que invoca los algoritmos de prevención de congestión y además el campo ventana es nulo, el sistema entrará en un estado de "acción congestión" provocado por nosotros y no enviará ningún segmento TCP hasta que salgamos de este estado.

Para salir de este estado basta con que llegue un nuevo ack de datos nuevos que permita abrir la ventana y continuar el envío de segmentos TCP.

CAPÍTULO 5. Conclusiones y trabajos futuros

5.1. Conclusiones generales del proyecto

El proxy snoop es una buena solución para mejorar la eficiencia de TCP en redes inalámbricas, el hecho de eliminar de la red los acks duplicados permite que el emisor TCP no invoque los mecanismos de prevención de congestión, ya que eso supondría una reducción de la ventana de congestión. También reduce el trafico innecesario de acks duplicados.

Esta característica que diferencia el proxy de otras propuestas con la misma finalidad de mejorar el comportamiento de TCP en redes inalámbricas, hace que en determinadas situaciones, con enlaces de baja velocidad o cuando introducimos poco tráfico y a ráfagas, el proxy no consigue mejorar el throughput de la comunicación TCP.

Con el proxy snoop adaptado se pretende solventar los problemas que se producen cuando el proxy actúa en enlaces de baja velocidad. En los estudios realizados [2] se intenta dar salida comercial al proxy adaptado a un red GPRS para su posterior implantación en un sistema real.

El módulo FullTCP del simulador es una implementación mas cercana del protocolo TCP real en el sentido que puede crear tanto segmentos TCP como reconocimientos. Aunque es un módulo que esta aun bajo construcción es importante poder adaptar el proxy a este modelo para en un futuro poder realizar pruebas con ambos. En la pruebas que hemos realizado el proxy sólo monitoriza una conexión TCP. En un entorno real tendría que monitorizar cada conexión TCP que pase a través del proxy. Con el modelo FullTCP del simulador NS-2 podemos realizar este tipo de pruebas.

5.2. Trabajos futuros

A lo largo de este documento se ha descrito el trabajo realizado durante el tiempo que ha durado el proyecto. Con este documento no se pretende conseguir una versión final del proxy adaptado si no ayudar a futuros proyectistas a desarrollar nuevas versiones. Hemos planteado las ventajas e inconvenientes del proxy y creemos que seria interesante estudiar en profundidad algunas de las desventajas para crear un proxy aun más robusto y adaptable a enlaces de baja velocidad.

Un posible trabajo podría consistir en adaptar aun más el protocolo TCP implementado en el simulador tal y como hemos hecho con el modelo en un único sentido pero en este caso con el módulo FullTCP del simulador NS-2. Por otro lado, seria interesante estudiar el comportamiento del proxy en un entorno inalámbrico, si es posible en un entorno GPRS.

Por último, otra manera de continuar con este proyecto es realizando nuevas simulaciones variando más parámetros de configuración. Estos parámetros podrían ser el tamaño de los segmentos TCP que enviamos, el retardo de propagación de los enlaces así como otros generadores de tráfico.

Bibliografía

Los documentos que citamos a continuación han hecho posible el desarrollo de este proyecto. La mayoría hacen referencia a la pagina Web del simulador NS-2 y los seis primeros corresponden a estudios realizados por diferente universidades.

[1] Hari Balakrishnan, Srinivasan Seshan, Elan Amir and Randy H. Katz, Computer Science Division University of California at Berkeley, *Improving TCP/IP Performance over Wireless Networks,* 1-10 (1995)

[2] Anna Calveras Augé, María Luisa Catalán Cid, Carles Montenegro Josep Paradells Aspas, Lluís Casals Ibáñez. *Mejora del acceso a Internet a traves de GPRS mediante la utilización de un proxy snoop.* Telecom I + D 2003

[3] Sarma Vangala and Miguel A. Labrador Department of Computer Science and Engineering University of South Florida, *Performance of TCP over Wireless Networks with the Snoop Protocol*

[4] Hari Balakrishnan, V. Padmanabhan, S. Seshan, andR. Katz. *A Comparison of Mechanisms for ImprovingTCP Performance over Wireless Links*. ACM/IEEE Transactions on Networking, Vol. 5,No. 6:756–769, 1997.

[5] Hari Balakrishnan, S. Seshan, andR. Katz. *Improving ReliableTransport and Handoff Performancein Cellular Wireless Networks*. ACMWirelessNetworks, Vol. 1,1995.

[6] Christina Parsa J.J. Garcia-Luna-Aceves Computer Engineering Department Baskin School of Engineering University of California, *TULIP: A Link-Level Protocol for Improving TCP over Wireless Links.*

[7] Pagina Web del simulador, http://www.isi.edu/nsnam/ns/, 2005

[8] Manual del NS-2, http://www.isi.edu/nsnam/ns/doc/index.html, 2005

[9] Manual de Marc Greis, http://www.isi.edu/nsnam/ns/tutorial/index.html, 2005

[10] Manual para principiantes, *NS Simulator for Beginners* , 1-146 (2003) http://www-sop.inria.fr/maestro/personnel/Eitan.Altman/COURS-S/n3.pdf

[11] Jae Chung and Mark Claypool, *NS by Example*, http://nile.wpi.edu/NS/,

[12] Manual del leguaje de programación C++, http://www.cplusplus.com/doc/tutorial/, 2005

[13] Lista de distribución creada por los usuarios del simulador de NS-2, http://mailman.isi.edu/pipermail/ns-users/, todo lo relacionado con snoop.

ANEXOS

AVALUACIÓ D'UN PROXY EN ENTORNS AMB I SENSE FILS

TITULACIÓN: Ingeniería Técnica de Telecomunicaciones

AUTOR: Álvaro Rodatos Rite

DIRECTOR: Lluis Casals Ibáñez

FECHA: 5 Septiembre 2005

ANEXO A

En esta parte del documento describimos todos los códigos que hemos utilizado para realizar el trabajo final de carrera.

Los códigos que hemos utilizado para realizar las simulaciones están programaos en TCL y los módulos que describen el comportamiento de los diferentes protocolos están programados C++.

Los filtros y archivos que hemos utilizado para obtener las gráficas tiempo secuencia así como el calculo del throughput para cada simulación, están descritos al final del documento.

Archivo de simulación MySnoop.tcl

El archivo de simulación MySnoop.tcl esta programado a nivel TCL y contiene todas las funciones que hemos utilizado para realizar las simulaciones.

En él hemos ido modificando los generadores de trafico, el ancho de banda de los enlaces y hemos añadido una función para obtener gráficas que describen las variaciones que han tenido lugar en la ventana de congestión del emisor TCP. A continuación podemos ver este archivo junto con algunos comentarios para aclarar su contenido.

```
puts "sourcing ../../lan/vlan.tcl..."
source ../..//lan/vlan.tcl
source ../../lan/ns-mac.tcl


set opt(tr)   out.tr // Trazas de la simulacion
set opt(namtr)        "MySnoop.nam" // Trazas de la simulacion
set opt(seed) 0
set opt(stop) 10  // Tiempo en segundos de la simulación
set opt(node) 2   // Variable que define el número de nodos

set opt(qsize)        100
set opt(bw)   20Mb  //Ancho de banda de la LAN
set opt(delay) 1ms  // Retardo de propagacion de la LAN
set opt(ll)   LL
set opt(ifq)  Queue/DropTail
set opt(mac)  Mac/802_3
set opt(chan) Channel
set opt(tcp)  TCP/Reno  // Versión TCP que hemos usado
set opt(sink) TCPSink

set opt(app)  FTP
// con esta variable modificaremos la probabilidad de error
```

```
set loss_prob 5
set winfile [open WinFile w]


proc finish {} {
    global ns opt

    $ns flush-trace
        // Con la linea de abajo, indicamos el camino donde se encuentra el filtro
       y      a continuacion el archivo que queremos filtrar
    exec   tclsh   /home/alvaro/ns-allinone-2.27/ns-2.27/tcl/ex/snoop/namfilter.tcl
MySnoop.nam

    exec nam $opt(namtr) &

    exit 0
}

// Esta función crea las trazas de la simulación
proc create-trace {} {
    global ns opt

    if [file exists $opt(tr)] {
        catch "exec rm -f $opt(tr) $opt(tr)-bw [glob $opt(tr).*]"
    }

    set trfd [open $opt(tr) w]
    $ns trace-all $trfd
    if {$opt(namtr) != ""} {
        $ns namtrace-all [open $opt(namtr) w]
    }
    return $trfd
}

// Creamos el modelo de error
proc add-error {LossyLink} {

   global loss_prob

   # crear la variable aleatoria de la distribución uniforme
   set loss_random_variable [new RandomVariable/Uniform]
   $loss_random_variable set min_ 0    # set the range of the random variable;
   $loss_random_variable set max_ 100

   # crear el modulo de error
   set loss_module [new ErrorModel]
   $loss_module drop-target [new Agent/Null]
   $loss_module set rate_ $loss_prob  # set error rate to (0.1 = 10 / (100 - 0));
   # error unit: packets (the default);
   $loss_module unit pkt
```

```
    # attach random var. to loss module;
    $loss_module ranvar $loss_random_variable

    # guarde una manejador al módulo de pérdidas
    #set sessionhelper [$ns create-session $n0 $tcp0]
    $LossyLink errormodule $loss_module
}

// Función para crear la topologia en este caso un nodo
proc create-topology {} {
     global ns opt
     global lan node s d

     set num $opt(node)
     for {set i 1} {$i < $num} {incr i} {
          set node($i) [$ns node]
          lappend nodelist $node($i)
     }

     set lan [$ns make-lan $nodelist $opt(bw) \
                    $opt(delay) $opt(ll) $opt(ifq) $opt(mac) $opt(chan)]

     // Aqui añadimos el proxy snoop en la capa de enlace
     #set opt(ll) LL/LLSnoop
     #$opt(ll) set integrate_ 2.0

     set opt(ifq) Queue/DropTail
     $opt(ifq) set limit_ 100

     // Añadimos el nodo con el proxy a nuestra LAN
     # set up snoop agent
     set node(0) [$ns node]
     $lan addNode [list $node(0)]    $opt(bw) $opt(delay) $opt(ll) $opt(ifq)
$opt(mac)

     # iniciar fuente y conectarla al node(0)
     set s [$ns node]
     $ns duplex-link $s $node(0) 10Mb 20ms DropTail
     $ns queue-limit $s $node(0) 100000
     $ns duplex-link-op $s $node(0) orient right

     # iniciar destino y conectarla al node(1)
     set d [$ns node]
     $ns duplex-link $node(1) $d 10Mb 20ms DropTail
     $ns queue-limit $node(1) $d 100000
     $ns duplex-link-op $d $node(1) orient left
```

```
    set LossyLink [$ns link $node(1) $d]

    add-error $LossyLink
}

## MAIN ##

set ns [new Simulator]

set trfd [create-trace]

create-topology

// Con estas lineas añadimos a los nodos, la version FullTCP del simulador
set src [new Agent/TCP/FullTcp]
set sink [new Agent/TCP/FullTcp]

$ns attach-agent $s $src
$ns attach-agent $d $sink

$src set fid_ 0
$sink set fid_ 0

$ns connect $src $sink

# iniciar conexiones TCP-level
$sink listen
$src set window_ 30

#Crear un agente (FTP) de archivo infinito y conectarlo al nodo n0
#set tcp0 [new Agent/TCP/Reno]
#$tcp0 set backoff_ 2
#$tcp0 set window_ 30
#$ns attach-agent $s $tcp0

// con estas lineas adjuntamos un generador de tráfico exponencial
#set e [new Application/Traffic/Exponential]
#$e attach-agent $tcp0
#$e set packetSize_ 512
#$e set burst_time_ 500ms
#$e set idle_time_ 500ms
#$e set rate_ 200k

// con estas lineas adjuntamos un generador de tráfico pareto
#set p [new Application/Traffic/Pareto]
#$p attach-agent $tcp0
#$p set packetSize_ 512
#$p set burst_time_ 500ms
#$p set idle_time_ 500ms
#$p set rate_ 200k
```

```
#$p set shape_ 1.5

// con estas lineas adjuntamos un generador de tráfico CBR
#set c [new Application/Traffic/CBR]
#$c attach-agent $tcp0
#$c set packetSize_ 512
#$c set rate_ 1000Kb
#$c set random_ 1

// con estas lineas adjuntamos un generador de tráfico TELNET
#set Telnet [new Application/Telnet]
#$Telnet attach-agent $tcp0
#$Telnet set interval_ 0.0005

// Declaramos el receptor de segmento TCP y emisor de ACKs
#set tcp_snk0 [new Agent/TCPSink]
#$ns attach-agent $d $tcp_snk0

// Realizamos la conexión a nivel TCP
#$ns connect $tcp0 $tcp_snk0

// Declaramos el generador de trafico FTP
#set ftp0 [$tcp0 attach-app FTP]
set ftp0 [$src attach-app FTP]
#set ftp1 [$sink attach-app FTP]

// Con esta función grabamos en un archivo las variaciones de la ventana de
transmision a lo largo de la simulación.
proc plotWindow {tcpSource file} {
global ns
set time 0.1
set now [$ns now]
set cwnd [$tcpSource set cwnd_]
set wnd [$tcpSource set window_]
puts $file "$now $cwnd"
$ns at [expr $now+$time] "plotWindow $tcpSource $file" }
$ns at 0.1 "plotWindow $src $winfile"

// Iniciamos la simulacion, el generador de tráfico y finalmente la detenemos
$ns at 0.0 "$ftp0 start"
#$ns at 1.0 "$ftp1 start"
$ns at $opt(stop) "finish"
$ns run
```

Propuesta de adaptación de snoop a GPRS

En el capitulo 4 hemos realizado una propuesta de adaptación del proxy a un entorno GPRS añadiendo nuevas funcionalidades al proxy creado por la universidad de telecomunicaciones de Berckely.

En el siguiente codigo se puede ver el funcionamieno del proxy snoop como módulo del simulador y las funciones openWin (), closeWin() y timeout () que hemos creado.

Para llevar a cabo las funciones que abren y cierran la ventana de transmisión hemos tenido que modificar el modulo tcp-reno, ya que en el simulador los paquetes no tienen el campo ventana en la cabecera TCP por lo que el tamaño de la ventana de transmisión permanece constate.

```
#include "snoopAdapted.h"

int hdr_snoop::offset_;

class SnoopHeaderClass : public PacketHeaderClass {
public:
	SnoopHeaderClass() : PacketHeaderClass("PacketHeader/Snoop",
					sizeof(hdr_snoop)) {
		bind_offset(&hdr_snoop::offset_);
	}
 class_snoophdr;

static class LLSnoopClass : public TclClass {
public:
	LLSnoopClass() : TclClass("LL/LLSnoopAdapted") {}//nuevo id del snoop adaptado
	//LLSnoopClass() : TclClass("LL/LLSnoop") {}
	TclObject* create(int, const char*const*) {
		return (new LLSnoop());

	}
} llsnoop_class;

static class SnoopClass : public TclClass {
public:
	SnoopClass() : TclClass("Snoop") {}
	TclObject* create(int, const char*const*) {
		return (new Snoop());
	}
} snoop_class;

Snoop::Snoop() : NsObject(),
	fstate_(0), lastSeen_(-1), lastAck_(-1),
	expNextAck_(0), expDupacks_(0), bufhead_(0),
	toutPending_(0), buftail_(0),
	wl_state_(SNOOP_WLEMPTY), wl_lastSeen_(-1), wl_lastAck_(-1),
	wl_bufhead_(0), wl_buftail_(0)
{
	bind("snoopDisable_", &snoopDisable_);
	bind_time("srtt_", &srtt_);
	bind_time("rto_", &rto_);//Hacemos visible  la variable desde OTcl
	bind_time("rttvar_", &rttvar_);
	bind("maxbufs_", &maxbufs_);
```

```
	bind("snoopTick_", &snoopTick_);
	bind("g_", &g_);
	bind("tailTime_", &tailTime_);
	bind("rxmitStatus_", &rxmitStatus_);
	bind("lru_", &lru_);
	bind("pktSize_", &pktSize_);//tamaño del paquete para fullTCP

	rxmitHandler_ = new SnoopRxmitHandler(this);

	int i;
	for (i = 0; i < SNOOP_MAXWIND; i++) /* data from wired->wireless */
		pkts_[i] = 0;
	for (i = 0; i < SNOOP_WLSEQS; i++) {/* data from wireless->wired */
		wlseqs_[i] = (hdr_seq *) malloc(sizeof(hdr_seq));
		wlseqs_[i]->seq = wlseqs_[i]->num = 0;
	}
	if (maxbufs_ == 0)
		maxbufs_ = SNOOP_MAXWIND;
}

void
Snoop::reset()
{
//	printf("%x resetting\n", this);
	fstate_ = 0;
	lastSeen_ = -1;
	lastAck_ = -1;
	expNextAck_ = 0;
	expDupacks_ = 0;
	bufhead_ = buftail_ = 0;
	if (toutPending_) {
		Scheduler::instance().cancel(toutPending_);

		toutPending_ = 0;
	};
	for (int i = 0; i < SNOOP_MAXWIND; i++) {
		if (pkts_[i]) {
			Packet::free(pkts_[i]);
			pkts_[i] = 0;
		}
	}
}

void
Snoop::wlreset()
{
	wl_state_ = SNOOP_WLEMPTY;
	wl_bufhead_ = wl_buftail_ = 0;
	for (int i = 0; i < SNOOP_WLSEQS; i++) {
		wlseqs_[i]->seq = wlseqs_[i]->num = 0;
	}
}
void
Snoop::closeWin()
{
	//Creamos el pkt que cerrará la ventana
	Packet* pkt = allocpkt();
	hdr_tcp *tcp = hdr_tcp::access(pkt);
	hdr_cmn::access(pkt)->ptype() == PT_ACK;
```

```
	//Ponemos el número de secuencia del primer pkt del buffer sin reconocer
	tcp->seqno() = hdr_snoop::access(pkts_[buftail_])->seqno());

	// especificamos la direccion, en este caso down
	hdr_cmn *ch = HDR_CMN(pkt);
	hdr_tcp::access(pkt)->wnd_ = 0;//cerramos la ventana
	ch->direction() = hdr_cmn::DOWN;
	parent_->sendDown(p);	// vector a LLSnoop's sendto()
}

void
Snoop::openWin(int seqno_)
{
	//Creamos el pkt que abrirá la ventana
	Packet* pkt = allocpkt();
	hdr_tcp *tcp = hdr_tcp::access(pkt);

	//Sumamos 1? al número de secuencia del primer pkt de la caché sin reconocer
	tcp->seqno() = seqno_ //hdr_snoop::access((pkts_[buftail_])->seqno()) +1);
	// especificamos la dirección, en este caso down
	hdr_cmn *ch = HDR_CMN(pkt);
	hdr_tcp::access(pkt)->wnd_ = 1; //abrimos la ventana
	ch->direction() = hdr_cmn::DOWN;
	parent_->sendDown(p);	// vector a LLSnoop's sendto()

}

int
Snoop::command(int argc, const char*const* argv)
{
	//Tcl& tcl = Tcl::instance();

	if (argc == 3) {
		if (strcmp(argv[1], "llsnoop") == 0) {
			parent_ = (LLSnoop *) TclObject::lookup(argv[2]);
			if (parent_)
				recvtarget_ = parent_->uptarget();
			return (TCL_OK);
		}

		if (strcmp(argv[1], "check-rxmit") == 0) {
			if (empty_()) {
				rxmitStatus_ = SNOOP_PROPAGATE;
				return (TCL_OK);
			}

			Packet *p = pkts_[buftail_];
			hdr_snoop *sh = hdr_snoop::access(p);

			if (sh->sndTime()!=-1 && sh->sndTime()<atoi(argv[2]) &&
			    sh->numRxmit() == 0)
				/* candidate for retransmission */
				rxmitStatus_ = snoop_rxmit(p);
			else
				rxmitStatus_ = SNOOP_PROPAGATE;
			return (TCL_OK);
		}
	}
	return NsObject::command(argc, argv);
}
```

```
void LLSnoop::recv(Packet *p, Handler *h)
{
	Tcl &tcl = Tcl::instance();
	hdr_ip *iph = hdr_ip::access(p);

	/* get-snoop creates a snoop object if none currently exists */
	hdr_cmn *ch = HDR_CMN(p);
	if(ch->direction() == hdr_cmn::UP)
		/* get-snoop creates a snoop object if none currently exists */
		/* In ns, addresses have ports embedded in them. */
		tcl.evalf("%s get-snoop %d %d", name(), iph->daddr(),
			iph->saddr());

	else
		tcl.evalf("%s get-snoop %d %d", name(), iph->saddr(),
			iph->daddr());

	Snoop *snoop = (Snoop *) TclObject::lookup(tcl.result());

	snoop->recv(p, h);

	if (integrate_)
		tcl.evalf("%s integrate %d %d", name(), iph->saddr(),
			iph->daddr());
	if (h)			/* resume higher layer (queue) */
		Scheduler::instance().schedule(h, &intr_, 0.000001);
	return;
}

/*
 * Receive a packet from higher layer or from the network.
 * Call snoop_data() if TCP packet and forward it on if it's an ack.
 */
void
Snoop::recv(Packet* p, Handler* h )
{

	hdr_cmn *ch = HDR_CMN(p);
	if(ch->direction() == hdr_cmn::UP) {
		handle((Event *) p);
		return;
	}

	packet_t type = hdr_cmn::access(p)->ptype();
	/* Put packet (if not ack) in cache after checking, and send it on */

	if (type == PT_TCP)
		snoop_data(p);

	else if (type == PT_ACK)
		snoop_wired_ack(p);

	ch->direction() = hdr_cmn::DOWN;  // Ben added
	parent_->sendDown(p);	/* vector to LLSnoop's sendto() */
}

/*
 * Handle a packet received from peer across wireless link.  Check first
 * for packet errors, then call snoop_ack() or pass it up as necessary.
```

```
 */
oop_insert(p);
        if (toutPending_ && resetPending == SNOOP_TAIL) {
                s.cancel(toutPending_);
                // xxx: I think that toutPending_ doesn't need to be freed because snoop didn't
allocate it (but I'm not sure).
                toutPending_ = 0;
        }
        if (!toutPending_ && !empty_()) {
                toutPending_ = (Event *) (pkts_[buftail_]);
                s.schedule(rxmitHandler_, toutPending_, timeout());
        }
        return;
}

/*
 * snoop_insert() does all the hard work for snoop_data(). It traverses the
 * snoop cache and looks for the right place to insert this packet (or
 * determines if its already been cached). It then decides whether
 * this is a packet in the normal increasing sequence, whether it
 * is a sender-rexmitted-but-lost-due-to-congestion (or network
 * out-of-order) packet, or if it is a sender-rexmitted packet that
 * was buffered by us before.
 */
int
Snoop::snoop_insert(Packet *p)
{



        int i, seq = hdr_tcp::access(p)->seqno(), retval=0;

        if (seq <= lastAck_)
                return retval;

        if (fstate_ & SNOOP_FULL) {
                /* free tail and go on */
                printf("snoop full, making room\n");
                Packet::free(pkts_[buftail_]);
                pkts_[buftail_] = 0;
                buftail_ = next(buftail_);
                fstate_ |= ~SNOOP_FULL;
        }

        if (seq > lastSeen_ || pkts_[buftail_] == 0) { // en secuencia o la cache esta vacia
                i = bufhead_;
                bufhead_ = next(bufhead_);
        } else if (seq < hdr_snoop::access(pkts_[buftail_])->seqno()) {
                buftail_ = prev(buftail_);
                i = buftail_;
        } else {
                for (i = buftail_; i != bufhead_; i = next(i)) {
                        hdr_snoop *sh = hdr_snoop::access(pkts_[i]);
                        if (sh->seqno() == seq) {  // ya fue almacenado

                                sh->numRxmit() = 0;
                                sh->senderRxmit() = 1; //debe ser una retx del emisor
                                sh->sndTime() = Scheduler::instance().clock();
                                return SNOOP_TAIL;
                        } else if (sh->seqno() > seq) {
```

```
                        //no fue almacenado,
                        // encontrar la posicion, deberia ser: prev(i)

                        Packet *temp = pkts_[prev(buftail_)];
                        for (int j = buftail_; j != i; j = next(j))
                                pkts_[prev(j)] = pkts_[j];
                        i = prev(i);
                        pkts_[i] = temp;
                        buftail_ = prev(buftail_);
                        break;
                }
        }
        if (i == bufhead_)
                bufhead_ = next(bufhead_);
    }

    // lo guardamos en el buffer
    savepkt_(p, seq, i);

    if (bufhead_ == buftail_)
        fstate_ |= SNOOP_FULL;
    /*
     * If we have one of the following packets:
     * 1. a network-out-of-order packet, or
     * 2. a fast rxmit packet, or 3. a sender retransmission
     * AND it hasn't already been buffered,
     * then seq will be < lastSeen_.
     * We mark this packet as having been due to a sender rexmit
     * and use this information in snoop_ack(). We let the dupacks
     * for this packet go through according to expDupacks_.
     */
    if (seq < lastSeen_) { /* not in-order -- XXX should it be <= ? */
        if (buftail_ == i) {
            hdr_snoop *sh = hdr_snoop::access(pkts_[i]);
            sh->senderRxmit() = 1;
            sh->numRxmit() = 0;
        }
        expNextAck_ = buftail_;
        retval = SNOOP_TAIL;
    } else
        lastSeen_ = seq;

    return retval;
}

void
Snoop::savepkt_(Packet *p, int seq, int i)
{
    pkts_[i] = p->copy();
    Packet *pkt = pkts_[i];
    hdr_snoop *sh = hdr_snoop::access(pkt);
    sh->seqno() = seq;
    sh->numRxmit() = 0;
    sh->senderRxmit() = 0;
    sh->sndTime() = Scheduler::instance().clock();
}

/*
 * procesando ack's en snoop protocol.  Sabemos de antemano que es un ack.
```

```
 * Devolvemos SNOOP_SUPPRESS si queremos eliminar el ack y si no
SNOOP_PROPAGATE.
 */
int
Snoop::snoop_ack(Packet *p)
{
        Packet *pkt;

        int ack = hdr_tcp::access(p)->seqno();
        bool cong_ ;

        /*
         * Tenemos 3 casos:
         * 1. lastAck_ > ack.  En este caso lo que ha ocurrido es que el ack esta fuera de orden,
no hacemos ningun procesado
              local y reenviamos el ack.
         * 2. lastAck_ == ack.  Este es un ack duplicado. Si tenemos el pkt lo reenviamos
                y eliminamos el dupack.
         *    Si no lo tenemos hemos de dejar pasar el ack para que el emisor lo retransmita.
         *    Iniciar expDupacks_ al número de pkts ya enviados, este es el número de acks
duplicados a eliminar.
         * 3. lastAck_ < ack.  Iniciar lastAck_ = ack, y actulizar la cabecera de la cola. Tambien,
eliminar los pkts reconocidos.
         */
        if (fstate_ & SNOOP_CLOSED || lastAck_ > ack)
                return SNOOP_PROPAGATE;   // enviar ack hacia adelante

        if (lastAck_ == ack) {
                // Es un ack duplicado

                pkt = pkts_[buftail_];
                // no tenemos el pkt, lo dejamos pasar

                if (pkt == 0)
                        return SNOOP_PROPAGATE;

                hdr_snoop *sh = hdr_snoop::access(pkt);

                if (pkt == 0 || sh->seqno() > ack + 1)
                        return SNOOP_PROPAGATE;

                /*
                 * Tenemos el paquete: tenemos 3 posibilidades
                 * 1. No estamos esperando ningun dupacks (expDupacks_ == 0)
                 * 2. Estamos esperando dupacks (expDupacks_ > 0)
                 * 3. Estamos en un estado contrario (expDupacks_ == -1)
                 */


                if (expDupacks_ == 0) {      // no lo esperamos
#define RTX_THRESH 1

                        static int thresh = 0;
                        if (thresh++ < RTX_THRESH)
                                /* no hacemos nada si estamos por debajo RTX_THRESH */
                                return SNOOP_PROPAGATE;

                        thresh = 0;

                        // si el pkt es una retransmision del emisor, lo dejamos pasar
```

```
                if (sh->senderRxmit())
                        return SNOOP_PROPAGATE;

                /*
                 * De otro modo, no fue enviado por el emisor.  Si es el primero
                 * recibido, hemos de determinar cuantos
                 * dupacks llegaran para ser eliminados, y también
                 * retransmitir el pkt deseado. expDupacks_
                 *  será -1 si contamos incorrectamente por alguna razón.
                 */
                expDupacks_ = bufhead_ - expNextAck_;
                if (expDupacks_ < 0)
                        expDupacks_ += SNOOP_MAXWIND;
                expDupacks_ -= RTX_THRESH + 1;
                expNextAck_ = next(buftail_);
                closeWin();
                cong_ = true;

                if (sh->numRxmit() == 0)
                        return snoop_rxmit(pkt);

        } else if (expDupacks_ > 0) {
                expDupacks_--;
                return SNOOP_SUPPRESS;
        } else if (expDupacks_ == -1) {
                if (sh->numRxmit() < 2) {
                        return snoop_rxmit(pkt);
                }
        } else                  // dejamos que lo reciba el emisor
                return SNOOP_PROPAGATE;
    } else {                    //es un ack en secuencia

        fstate_ &= ~SNOOP_NOACK; //por lo menos se ha visto 1 ack nuevo

        //liberamos buffers
        double sndTime = snoop_cleanbufs_(ack);//ack= nª secuencia de *p

        //if (sndTime != -1)// si es -1 no hay pkts en el buffer
                //snoop_rtt(sndTime);// tiempo envio pkt en 1ª posicion del buffer
        if(cong_){

                openWin(ack);
                cong_= false;
        }

        expDupacks_ = 0;
        expNextAck_ = buftail_;
        lastAck_ = ack;

    }
    return SNOOP_PROPAGATE;
}

/*
 * Handle data packets that arrive from a wireless link, and we're not
 * the end recipient.  See if there are any holes in the transmission, and
 * if there are, mark them as candidates for wireless loss.  Then, when
 * (dup)acks troop back for this loss, set the ELN bit in their header, to
 * help the sender (or a snoop agent downstream) retransmit.
 */
```

```
void
Snoop::snoop_wless_data(Packet *p)
{
	hdr_tcp *th = hdr_tcp::access(p);
	int i, seq = th->seqno();

	if (wl_state_ & SNOOP_WLALIVE && seq == 0)
		wlreset();
	wl_state_ |= SNOOP_WLALIVE;

	if (wl_state_ & SNOOP_WLEMPTY && seq >= wl_lastAck_) {
		wlseqs_[wl_bufhead_]->seq = seq;
		wlseqs_[wl_bufhead_]->num = 1;
		wl_buftail_ = wl_bufhead_;
		wl_bufhead_ = wl_next(wl_bufhead_);
		wl_lastSeen_ = seq;
		wl_state_ &= ~SNOOP_WLEMPTY;
		return;
	}
	/* WL data list definitely not empty at this point. */
	if (seq >= wl_lastSeen_) {
		wl_lastSeen_ = seq;
		i = wl_prev(wl_bufhead_);
		if (wlseqs_[i]->seq + wlseqs_[i]->num == seq) {
			wlseqs_[i]->num++;
			return;
		}
		i = wl_bufhead_;
		wl_bufhead_ = wl_next(wl_bufhead_);
	} else if (seq == wlseqs_[i = wl_buftail_]->seq - 1) {
	} else
		return;

	wlseqs_[i]->seq = seq;
	wlseqs_[i]->num++;

	/* Ignore network out-of-ordering and retransmissions for now */
	return;
}

/*
 * Ack from wired side (for sender on "other" side of wireless link.
 */
void
Snoop::snoop_wired_ack(Packet *p)
{
	hdr_tcp *th = hdr_tcp::access(p);
	int ack = th->seqno();
	int i;

	if (ack == wl_lastAck_ && snoop_wlessloss(ack)) {
		hdr_flags::access(p)->eln_ = 1;
	} else if (ack > wl_lastAck_) {
		/* update info about unack'd data */
		for (i = wl_buftail_; i != wl_bufhead_; i = wl_next(i)) {
			hdr_seq *t = wlseqs_[i];
			if (t->seq + t->num - 1 <= ack) {
				t->seq = t->num = 0;
			} else if (ack < t->seq) {
				break;
```

```
                } else if (ack < t->seq + t->num - 1) {
                        /* ack for part of a block */
                        t->num -= ack - t->seq +1;
                        t->seq = ack + 1;
                        break;
                }
        }
        wl_buftail_ = i;
        if (wl_buftail_ == wl_bufhead_)
                wl_state_ |= SNOOP_WLEMPTY;
        wl_lastAck_ = ack;
        /* Even a new ack could cause an ELN to be set. */
        if (wl_bufhead_ != wl_buftail_ && snoop_wlessloss(ack))
                hdr_flags::access(p)->eln_ = 1;
    }
}

/*
 * Return 1 if we think this packet loss was not congestion-related, and
 * 0 otherwise.  This function simply implements the lookup into the table
 * that maintains this info; most of the hard work is done in
 * snoop_wless_data() and snoop_wired_ack().
 */
int
Snoop::snoop_wlessloss(int ack)
{
    if ((wl_bufhead_ == wl_buftail_) || wlseqs_[wl_buftail_]->seq > ack+1)
            return 1;
    return 0;
}

/*
 * clean snoop cache of packets that have been acked.
 */
double
Snoop::snoop_cleanbufs_(int ack)
{
    Scheduler &s = Scheduler::instance();
    double sndTime = -1;

    if (toutPending_) {
            s.cancel(toutPending_);
            // xxx: I think that toutPending_ doesn't need to be freed because snoop didn't
allocate it (but I'm not sure).
            toutPending_ = 0;
    };

    if (empty_())
            return sndTime;

    int i = buftail_;
    do {
            hdr_snoop *sh = hdr_snoop::access(pkts_[i]);
            int seq = hdr_tcp::access(pkts_[i])->seqno();

            if (seq <= ack) {
                    sndTime = sh->sndTime();
                    Packet::free(pkts_[i]);
                    pkts_[i] = 0;
                    fstate_ &= ~SNOOP_FULL;       /* XXX redundant? */
```

```
                } else if (seq > ack)
                        break;
                i = next(i);
        } while (i != bufhead_);

        if ((i != buftail_) || (bufhead_ != buftail_)) {
                fstate_ &= ~SNOOP_FULL;
                buftail_ = i;
        }
        if (!empty_()) {
                toutPending_ = (Event *) (pkts_[buftail_]);
                s.schedule(rxmitHandler_, toutPending_, timeout());
                hdr_snoop *sh = hdr_snoop::access(pkts_[buftail_]);
                tailTime_ = sh->sndTime();
        }

        return sndTime;
}

/*
 * Calculate smoothed rtt estimate and linear deviation.
 */
void
Snoop::snoop_rtt(double sndTime)
{
        double rtt = Scheduler::instance().clock() - sndTime;

        if (parent_->integrate()) {
                parent_->snoop_rtt(sndTime);
                return;
        }

        if (rtt > 0) {
                srtt_ = g_*srtt_ + (1-g_)*rtt;
                double delta = rtt - srtt_;
                if (delta < 0)
                        delta = -delta;
                if (rttvar_ != 0)
                        rttvar_ = g_*delta + (1-g_)*rttvar_;
                else
                        rttvar_ = delta;
        }
}


/*
 * Calculate smoothed rtt estimate and linear deviation.
 */
void
LLSnoop::snoop_rtt(double sndTime)
{
        double rtt = Scheduler::instance().clock() - sndTime;
        if (rtt > 0) {
                srtt_ = g_*srtt_ + (1-g_)*rtt;
                double delta = rtt - srtt_;
                if (delta < 0)
                        delta = -delta;
                if (rttvar_ != 0)
                        rttvar_ = g_*delta + (1-g_)*rttvar_;
                else
```

```
                rttvar_ = delta;
        }
}

/*
 * Returns 1 if recent queue length is <= half the maximum and 0 otherwise.
 */
int
Snoop::snoop_qlong()
{
        /* For now only instantaneous lengths */
        //      if (parent_->ifq()->length() <= 3*parent_->ifq()->limit()/4)

        return 1;
                //      return 0;
}

/*
 * Ideally, would like to schedule snoop retransmissions at higher priority.
 */
int
Snoop::snoop_rxmit(Packet *pkt)
{
        Scheduler& s = Scheduler::instance();
        if (pkt != 0) {
                hdr_snoop *sh = hdr_snoop::access(pkt);
                if (sh->numRxmit() < SNOOP_MAX_RXMIT && snoop_qlong()) {
                        /*                      && sh->seqno() == lastAck_+1)  */

#if 0
                        printf("%f Rxmitting packet %d\n", s.clock(),
                             hdr_tcp::access(pkt)->seqno());
#endif

                        // need to specify direction, in this case, down
                        hdr_cmn *ch = HDR_CMN(pkt);
                        ch->direction() = hdr_cmn::DOWN;  // Ben added

                        sh->sndTime() = s.clock();
                        sh->numRxmit() = sh->numRxmit() + 1;
                        Packet *p = pkt->copy();
                        parent_->sendDown(p);
                } else
                        return SNOOP_PROPAGATE;
        }
        /* Reset timeout for later time. */
        if (toutPending_) {
                s.cancel(toutPending_);
                // xxx: I think that toutPending_ doesn't need to be freed because snoop didn't
allocate it (but I'm not sure).
        };
        toutPending_ = (Event *)pkt;
        s.schedule(rxmitHandler_, toutPending_, timeout());
        return SNOOP_SUPPRESS;
}

void
Snoop::snoop_cleanup()
{
}
```

```
void
SnoopRxmitHandler::handle(Event *)
{
	Packet *p = snoop_->pkts_[snoop_->buftail_];
	snoop_->toutPending_ = 0;
	if (p == 0)
		return;
	hdr_snoop *sh = hdr_snoop::access(p);
	if (sh->seqno() != snoop_->lastAck_ + 1)
		return;
	if ((snoop_->bufhead_ != snoop_->buftail_) ||
	   (snoop_->fstate_ & SNOOP_FULL)) {
		//		printf("%f Snoop timeout\n", Scheduler::instance().clock());
		if (snoop_->snoop_rxmit(p) == SNOOP_SUPPRESS)
			snoop_->expNextAck_ = snoop_->next(snoop_->buftail_);
	}
}
```

Propuesta de adaptación de adaptación de TCP a snoop

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>

#include "ip.h"
#include "tcp.h"
#include "flags.h"


static class RenoTcpClass : public TclClass {
public:
	RenoTcpClass() : TclClass("Agent/TCP/Reno") {}
	TclObject* create(int, const char*const*) {
		return (new RenoTcpAgent());
	}
} class_reno;

int RenoTcpAgent::window()
{

	// reno: infla la ventana de congestion con dupwnd_
	//	dupwnd_ no será cero durante fast recovery,
	//	en ese momento contedrá el num de ack's duplicados

	int win = int(cwnd_) + dupwnd_;
	if (frto_ == 2) {

		// primer ack que llega depues de un RTO.
		// abrir ventana para permitir dos nuevos segmentos "out with F-RTO."

		win = force_wnd(2);
	}
	if (win > int(wnd_))
		win = int(wnd_);
	return (win);
}

double RenoTcpAgent::windowd()
{
```

```
        //
        // reno: infla la ventana de congestion con dupwnd_
        //       dupwnd_ no será cero durante fast recovery,
        //       en ese momento contedrá el num de ack's duplicados
        //
        double win = cwnd_ + dupwnd_;
        if (win > wnd_)
                win = wnd_;
        return (win);
}

RenoTcpAgent::RenoTcpAgent() : TcpAgent(), dupwnd_(0)
{
}

void RenoTcpAgent::recv(Packet *pkt, Handler*)
{
        hdr_tcp *tcph = hdr_tcp::access(pkt);
        int valid_ack = 0;
       if (qs_approved_ == 1 && tcph->seqno() > last_ack_)
       //Al salir de QuickStart, reducimos la ventana de congestion al tamaño que fue realmente
usado
                endQuickStart();
       if (qs_requested_ == 1)
                processQuickStart(pkt);
#ifdef notdef
        if (pkt->type_ != PT_ACK) {
                fprintf(stderr,
                        "ns: confiuration error: tcp received non-ack\n");
                exit(1);
        }
#endif
       // W.N.: comprobar si esto es de alguna encarnacion previa
       if (tcph->ts() < lastreset_) {
                // Quitar pkt y no hacemos nada
                Packet::free(pkt);
                return;
       }
        ++nackpack_;
        ts_peer_ = tcph->ts();

        if (hdr_flags::access(pkt)->ecnecho() && ecn_)
                ecn(tcph->seqno());
        recv_helper(pkt);
        recv_frto_helper(pkt);
        if (tcph->seqno() > last_ack_) {
                if (last_cwnd_action_ == CWND_ACTION_DUPACK || hdr_tcp::access(pkt)->wnd_)
                        last_cwnd_action_ == CWND_ACTION_EXITED;
                dupwnd_ = 0;
                recv_newack_helper(pkt);
                if (last_ack_ == 0 && delay_growth_) {
                        cwnd_ = initial_window();
                }
        } else if (tcph->seqno() == last_ack_) {
                if (hdr_flags::access(pkt)->eln_ && eln_) {
                        tcp_eln(pkt);
                        return;
                }
                if (++dupacks_ == numdupacks_) {
                        dupack_action();
```

```
                if (!exitFastRetrans_)
                        dupwnd_ = numdupacks_;
            } else if (dupacks_ > numdupacks_ && (!exitFastRetrans_
                || last_cwnd_action_ == CWND_ACTION_DUPACK )) {
                ++dupwnd_;      // fast recovery

            // si es el primer ack dup, y el campo ventana es nulo, cerramos la ventana de
congestion.
            } else if (dupacks_ < numdupacks_ && singledup_ && !hdr_tcp::access(pkt)->wnd_) {

                // send_one();
                last_cwnd_action_ == CWND_ACTION_DUPACK;
            }
        }
        if (tcph->seqno() >= last_ack_)
            // Comprobar que es un ACK valido. Sugerido por Mark Allman.
            valid_ack = 1;
        Packet::free(pkt);
#ifdef notyet
        if (trace_)
            plot();
#endif

        //intentamos enviar más datos

        if (valid_ack || aggressive_maxburst_)
            if (dupacks_ == 0 || dupacks_ > numdupacks_ - 1)
                send_much(0, 0, maxburst_);
}

int
RenoTcpAgent::allow_fast_retransmit(int last_cwnd_action_)
{
        return (last_cwnd_action_ == CWND_ACTION_DUPACK);
}

/*
 * Dupack-action: que hacer con DUP ACK.  Despues de comprobar inicialmente
 * 'recover' debajo, esta función implementa la siguiente tabla de la verdad:
 *
 *      bugfix  ecn     last-cwnd == ecn        action
 *
 *      0       0       0                       reno_action
 *      0       0       1                       reno_action     [imposible]
 *      0       1       0                       reno_action
 *      0       1       1                       retransmit, return
 *      1       0       0                       nothing
 *      1       0       1                       nothing         [imposible]
 *      1       1       0                       nothing
 *      1       1       1                       retransmit, return
 */

void
RenoTcpAgent::dupack_action()
{
        int recovered = (highest_ack_ > recover_);
        int allowFastRetransmit = allow_fast_retransmit(last_cwnd_action_);
        if (recovered || (!bug_fix_ && !ecn_) || allowFastRetransmit) {
                goto reno_action;
        }
```

```
        if (ecn_ && last_cwnd_action_ == CWND_ACTION_ECN) {
                last_cwnd_action_ = CWND_ACTION_DUPACK;

                // Si hay una accion DUPACK seguida de cerca por ECN
                // seguida de cerca por una accion DUPACK?
                // Lo mas optimo seria recordar todas las acciones de congestion
                // de la ventana de datos mas reciente.  De otro modo, "bugfix" no podrá prevenir
todos los Fast Retransmits
                // innecesarios.

                reset_rtx_timer(1,0);
                output(last_ack_ + 1, TCP_REASON_DUPACK);
                dupwnd_ = numdupacks_;
                return;
        }

        if (bug_fix_) {

                // La linea de abajo, evita problemas con
                //multiples fast retransmits en una misma ventana de datos.

                return;
        }

reno_action:
        //Ahora vamos a entrar en fast-retransmit y rastrearemos ese evento
        trace_event("RENO_FAST_RETX");
        recover_ = maxseq_;
        last_cwnd_action_ = CWND_ACTION_DUPACK;
        slowdown(CLOSE_SSTHRESH_HALF|CLOSE_CWND_HALF);
        reset_rtx_timer(1,0);
        output(last_ack_ + 1, TCP_REASON_DUPACK);       // Desde arriba
       dupwnd_ = numdupacks_;
        return;
}

void RenoTcpAgent::timeout(int tno)
{
        if (tno == TCP_TIMER_RTX) {
                dupwnd_ = 0;
                dupacks_ = 0;
                if (bug_fix_) recover_ = maxseq_;
                TcpAgent::timeout(tno);
        } else {
                timeout_nonrtx(tno);
        }
}
```

Filtro namfilter.tcl para las gráficas tiempo secuencia

El archivo namfilter.tcl permite filtrar el contenido de las trazas creadas por el progama NAM (Network Animator Monitor) de tal manera que podemos ver la evolución del número de secuencia en función del tiempo, siempre que tengamos una simulacion con una sesión TCP. A continuacion describimos el contenido de este archivo y como modificarlo.

Este archivo, se puede modifcar para filtrar tanto en el cliente como en el servidor de la transmisión, de hecho podemos filtrar el tráfico en cada nodo de la simulación. A continuacion describimos el archivo.

```
# Nam filter: Processing ns nam tracefile to produce info for namgraph

set analysis_OK 0
set analysis_ready 0

# color database
set colorname(0) black
set colorname(1) dodgerblue
set colorname(2) cornflowerblue
set colorname(3) blue
set colorname(4) deepskyblue
set colorname(5) steelblue
set colorname(6) navy
set colorname(7) darkolivegreen

set fcolorname(0) red
set fcolorname(1) brown
set fcolorname(2) purple
set fcolorname(3) orange
set fcolorname(4) chocolate
set fcolorname(5) salmon
set fcolorname(6) greenyellow
set fcolorname(7) gold
set fcolorname(8) red
set fcolorname(9) brown
set fcolorname(10) purple
set fcolorname(11) orange
set fcolorname(12) chocolate
set fcolorname(13) salmon
set fcolorname(14) greenyellow
set fcolorname(15) gold

# get trace item
proc get_trace_item { tag traceevent} {
   set next 0
   foreach item $traceevent {
      if { $next == 1 } {
```

```
            return $item
        }
        if { $item == $tag } {
            set next 1
        }
    }
    return ""
}

# process version info
proc handle_version { line } {

    global analysis_OK nam_version analysis_ready

    set nam_version [get_trace_item "-v" $line]
    if { $nam_version >= "1.0a5" } {
        set analysis_OK 1
    }
    set analysis_ready [get_trace_item "-a" $line]
}

# preanalysis
# return
#       0:      no filter can be applied
#       1:      filters have been applied
#       2:      Applying filters

proc nam_analysis { tracefile } {
    global analysis_OK nam_version analysis_ready

    set file [open $tracefile "r"]
    set line [gets $file]
    set time [get_trace_item "-t" $line]

    # skip all beginning non "*" events
    while {([eof $file]==0)&&([string compare $time "*"]!=0)} {
        set line [gets $file]
        set time [get_trace_item "-t" $line]
    }

    while {([eof $file]==0)&&([string compare $time "*"]==0) } {
        set cmd [lindex $line 0]
        switch "$cmd" {
            "V" {
                handle_version $line
                break
            }
        }
        set line [gets $file]
        set time [get_trace_item "-t" $line]
```

```
    }
    close $file

    # old nam, skip it
    if { $analysis_OK == 0 } {
        puts "You are using the tracefile format older than 1.0b5"
        puts "which will not allow you to run namgraph"
        return 0
    }

    # check if analysis ready
    if { $analysis_ready == 1 } {
         puts "Filters have been applied to the tracefile before"
         return 1

    } else {
         return 2
    }
}

proc nam_prefilter { tracefile } {

    # it only supports tcp & srm so far

    global colorname colorindex session_id highest_seq colorset groupmember
    global proto_id


    set file [open $tracefile "r"]
    set file2 [open $tracefile.tmp "w"]
    set colorindex 0

    # set color value
    while {[eof $file]==0} {
       set line [gets $file]
       set time [get_trace_item "-t" $line]
         set prot [get_trace_item "-p" $line]

       if { [string compare $time "*"]==0 } {
            puts $file2 $line
            continue
       }

       # get extended info after -x
       set extinfo [get_trace_item "-x" $line]

       if {$extinfo==""} {
            puts $file2 $line
            continue
         }
```

```
        # find the biggest tcp sequence number for the specific session
        #set tmp_seq [lindex $extinfo 2]
        #if { ($highest_seq < $tmp_seq) && ([string compare $prot "tcp"] == 0 \
           #   || [string compare $prot "ack"] == 0) } {
          #  set highest_seq $tmp_seq
        #}

        set src  [lindex $extinfo 0]
        set dst  [lindex $extinfo 1]

          set subtype [lindex $extinfo 4]
          set subtype_s [split $subtype "_"]
          set srm_flag [lindex $subtype_s 0]

          # tcp packet

        if {(![info exists colorset($src.$dst)]) && (![info exists colorset($dst.$src)]) \
             && ([string compare $prot "tcp"] == 0 || [string compare $prot "ack"] ==
0) } {

           set colorset($src.$dst) $colorindex
           set colorset($dst.$src) $colorindex

           set protocol [get_trace_item "-p" $line]

           set session_id($colorindex) "TCP session \
              between node $src and node $dst"
             set proto_id($colorindex) "TCP"

           incr colorindex
          }

          # SRM packet

          if { [string compare $srm_flag "SRM"] == 0 } {

             if { ![info exists colorset($dst)] } {
                set colorset($dst) $colorindex
                  set groupmember($dst) ""
                  set session_id($colorindex) SRM-$dst
                incr colorindex
             }

             set matchflag 0

             foreach member $groupmember($dst) {
                   if { [string compare $member $src] == 0 } {
                      set matchflag 1
                   }
```

```
            }

            if { $matchflag == 0 } {

                lappend groupmember($dst) $src
            }

        }

        #set specific color

       set attrindex [lsearch $line -a]
       incr attrindex

        if { [string compare $srm_flag "SRM"] == 0 } {
          set line2 [lreplace $line $attrindex $attrindex $colorset($dst)]
          set line "$line2 -S $colorset($dst)"
        } else {
            if { [string compare $prot "tcp"] == 0 || \
                 [string compare $prot "ack"] == 0 } {
              set line2 [lreplace $line $attrindex $attrindex $colorset($src.$dst)]
              set line "$line2 -S $colorset($src.$dst)"
            }
        }

        puts $file2 $line
    }

    close $file
    close $file2
    exec mv $tracefile.tmp $tracefile

#    set fcnt $colorindex

}

proc nam_filter { tracefile } {
    global filters filter_num colorindex colorname mcnt

    # set color value in the tracefile
    set tracefilebackup $tracefile.tmp
    set file [open $tracefile "r"]
    set file2 [open $tracefilebackup "w"]

    # define color value in the tracefile

    #for {set i 0} {$i < $colorindex} {incr i} {
    #    puts $file2 "c -t * -i $i -n $colorname($i)"
    #}
```

```
    while {[eof $file]==0} {

        set line [gets $file]
        set cmd [lindex $line 0]

        if {$line==""} {continue}

          # Apply filters here

          for {set i 0} {$i < $filter_num} {incr i} {
              set filtercmd [list $filters($i) $line]
              set line [eval $filtercmd]
          # puts $file2 $result
        }

          puts $file2 $line

    }

    close $file
    close $file2

    exec mv $tracefile.tmp $tracefile
}

# Filter srm packet

proc srm_filter {event} {
    global colorset fname fcnt mcnt colorindex srmdata_s srmsess_s srmrqst_s \
          srmrepr_s  srmdata_r  srmsess_r  srmrqst_r  srmrepr_r  srmrqst_rf
srmrepr_rf \
          srmrepr_sf srmrqst_sf groupmember

    set extinfo [get_trace_item "-x" $event]
    set type [lindex $event 0]

    if {$extinfo!=""} {

        #get srm flags
        set flagitem [lindex $extinfo 4]

        #filter tcp packet

        set protocol [get_trace_item "-p" $event]
        set cmd  [lindex $event 0]

        #1. SRM_DATA
        #2. SRM_SESS
        #3. SRM_RQST
        #4. SRM_REPR
```

```
if {[string compare $flagitem "SRM_DATA"] == 0 } {

    set src [get_trace_item "-s" $event]
    set ssrc [lindex $extinfo 0]
    set ssrcs [split $ssrc "."]
    set ssrc0 [lindex $ssrcs 0]

    # sending SRM_DATA from the source

    if { [string compare $src $ssrc0 ] == 0  && \
         [string compare $type "h" ] == 0} {

     if { ![info exists srmdata_s] } {
        set srmdata_s $mcnt
        set fname($mcnt) "SRM_DATA_SEND"
        incr mcnt
     }

     set fbit [get_trace_item "-f" $event]
         set sid [get_trace_item "-S" $event]

     if { $fbit == "" } {
        set event "$event -f $sid -m $srmdata_s"
     }
  }

    set ddst [lindex $extinfo 1]
    set dst [get_trace_item "-d" $event]
    set cnt 0

  foreach value $groupmember($ddst) {
         set ddsts [split $value "."]
         set ddst0 [lindex $ddsts 0]

         if { [string compare $ddst0 $dst] == 0 } {
            set cnt 1
            break
         }
  }

    if { $cnt == 1 && [string compare $type "r" ] == 0 } {
         if { ![info exists srmdata_r] } {
            set srmdata_r $mcnt
        set fname($mcnt) "SRM_DATA_RECV"
        incr mcnt
     }

     set fbit [get_trace_item "-f" $event]
     set sid [get_trace_item "-S" $event]
```

```
        if { $fbit == "" } {
            set event "$event -f $sid -m $srmdata_r"
        }
    }

}

  # SRM_SESSION

if {[string compare $flagitem "SRM_SESS"] == 0 } {

      set ddst [lindex $extinfo 1]
    set src [get_trace_item "-s" $event]
    set cnt 0

    foreach value $groupmember($ddst) {
        set ddsts [split $value "."]
        set ddst0 [lindex $ddsts 0]

        if { [string compare $ddst0 $src] == 0 } {
            set cnt 1
            break
        }
    }


    # sending SESSION from the source

    if { $cnt == 1 && [string compare $type "h" ] == 0 } {

        if { ![info exists srmsess_s] } {
            set srmsess_s $mcnt
            set fname($mcnt) "SRM_SESS_SEND"
            incr mcnt
        }

        set fbit [get_trace_item "-f" $event]
        set sid [get_trace_item "-S" $event]

        if { $fbit == "" } {
            set event "$event -f $sid -m $srmsess_s"
        }
    }

    set ddst [lindex $extinfo 1]
    set dst [get_trace_item "-d" $event]
    set cnt 0

    foreach value $groupmember($ddst) {
```

```
        set ddsts [split $value "."]
        set ddst0 [lindex $ddsts 0]

        if { [string compare $ddst0 $dst] == 0 } {
            set cnt 1
            break
        }
    }

    if { $cnt == 1 && [string compare $type "r" ] == 0 } {
        if { ![info exists srmsess_r] } {
            set srmsess_r $mcnt
            set fname($mcnt) "SRM_SESS_RECV"
            incr mcnt
        }

        set fbit [get_trace_item "-f" $event]
        set sid [get_trace_item "-S" $event]

        if { $fbit == "" } {
            set event "$event -f $sid -m $srmsess_r"
        }
    }


}


# RQST

if {[string compare $flagitem "SRM_RQST"] == 0 } {

    set ddst [lindex $extinfo 1]
    set src [get_trace_item "-s" $event]
    set cnt 0

    foreach value $groupmember($ddst) {
        set ddsts [split $value "."]
        set ddst0 [lindex $ddsts 0]

        if { [string compare $ddst0 $src] == 0 } {
            set cnt 1
            break
        }
    }


    if { $cnt == 1 && [string compare $type "h" ] == 0 } {

        if { ![info exists srmrqst_s] } {
```

```
            set srmrqst_s $mcnt
                set srmrqst_sf $fcnt
            set fname($mcnt.$fcnt) "SRM_RQST_SEND"
            incr mcnt
                incr fcnt
        }

        set fbit [get_trace_item "-f" $event]
        set sid [get_trace_item "-S" $event]

        if { $fbit == "" } {
            set event "$event -f $srmrqst_sf -m $srmrqst_s"
        }
    }

    set ddst [lindex $extinfo 1]
    set dst [get_trace_item "-d" $event]
    set cnt 0

    foreach value $groupmember($ddst) {
        set ddsts [split $value "."]
        set ddst0 [lindex $ddsts 0]

        if { [string compare $ddst0 $dst] == 0 } {
            set cnt 1
            break
        }
    }

    if { $cnt == 1 && [string compare $type "r" ] == 0 } {
        if { ![info exists srmrqst_r] } {
                set srmrqst_rf $fcnt
            set srmrqst_r $mcnt
            set fname($mcnt.$fcnt) "SRM_RQST_RECV"
            incr mcnt
                incr fcnt
        }

        set fbit [get_trace_item "-f" $event]
        set sid [get_trace_item "-S" $event]

        if { $fbit == "" } {
            set event "$event -f $srmrqst_rf -m $srmrqst_r"
        }
    }

}

  # REPR
```

```
if {[string compare $flagitem "SRM_REPR"] == 0 } {

    set ddst [lindex $extinfo 1]
    set src [get_trace_item "-s" $event]
    set cnt 0

    foreach value $groupmember($ddst) {
        set ddsts [split $value "."]
        set ddst0 [lindex $ddsts 0]

        if { [string compare $ddst0 $src] == 0 } {
            set cnt 1
            break
        }
    }


    if { $cnt == 1 && [string compare $type "h" ] == 0 } {

        if { ![info exists srmrepr_s] } {
            set srmrepr_s $mcnt
            set srmrepr_sf $fcnt
            set fname($mcnt.$fcnt) "SRM_REPR_SEND"
            incr mcnt
            incr fcnt
        }

        set fbit [get_trace_item "-f" $event]
        set sid [get_trace_item "-S" $event]

        if { $fbit == "" } {
            set event "$event -f $srmrepr_sf -m $srmrepr_s"
        }
    }

    set ddst [lindex $extinfo 1]
    set dst [get_trace_item "-d" $event]
    set cnt 0

    foreach value $groupmember($ddst) {
        set ddsts [split $value "."]
        set ddst0 [lindex $ddsts 0]

        if { [string compare $ddst0 $dst] == 0 } {
            set cnt 1
            break
        }
    }

    if { $cnt == 1 && [string compare $type "r" ] == 0 } {
```

```
            if { ![info exists srmrepr_r] } {
                set srmrepr_rf $fcnt
                set srmrepr_r $mcnt
                set fname($mcnt.$fcnt) "SRM_REPR_RECV"
                incr mcnt
                incr fcnt
            }

            set fbit [get_trace_item "-f" $event]
            set sid [get_trace_item "-S" $event]

            if { $fbit == "" } {
                set event "$event -f $srmrepr_rf -m $srmrepr_r"
            }
        }

    }

    return $event

}

# Filter tcp packet
proc tcp_filter { event } {

    global colorset fname fcnt mcnt highest_seq tcpmark

    set extinfo [get_trace_item "-x" $event]

    if {$extinfo!=""} {

        #get tcp flags
        set flagitem [lindex $extinfo 3]
        set tcpflags [split $flagitem "-"]

        #filter tcp packet

        set protocol [get_trace_item "-p" $event]
        set cmd  [lindex $event 0]

      set src  [lindex $extinfo 0]
      set dst  [lindex $extinfo 1]

      set lsrc [get_trace_item "-s" $event]
      set ldst [get_trace_item "-d" $event]
      set rsrc [lindex [split $src .] 0]
      set rdst [lindex [split $dst .] 0]
        set sid [get_trace_item "-S" $event]
```

```
        if {([string compare $protocol "tcp"] == 0) && \
            ([string compare $cmd "+"] == 0) \
              && ([string compare $ldst $rdst] == 0) } {

                if {![info exists tcpmark]} {
                    set tcpmark $mcnt
                    set fname($mcnt) "tcp"
                    incr mcnt
                }

                set event "$event -f $sid -m $tcpmark"
                set sid [get_trace_item "-S" $event]
                set seqno [lindex $extinfo 2]
                if {![info exists highest_seq($sid)]} {
                        set highest_seq($sid) $seqno
                }

                if { $seqno > $highest_seq($sid) } {
                     set highest_seq($sid) $seqno
                }

          }

    }

    return $event

}

#Filter ack packet

proc ack_filter { event } {

    global colorset fname fcnt mcnt ackmark

    set extinfo [get_trace_item "-x" $event]

    if {$extinfo!=""} {

        #get tcp flags
        set flagitem [lindex $extinfo 3]
        set tcpflags [split $flagitem "-"]

        set is_ecn [lsearch $tcpflags "E"]
        set is_echo [lsearch $tcpflags "C"]

        #filter ack packet
```

```
        set protocol [get_trace_item "-p" $event]
        set cmd  [lindex $event 0]

        set src  [lindex $extinfo 0]
        set dst  [lindex $extinfo 1]

        set lsrc [get_trace_item "-s" $event]
        set ldst [get_trace_item "-d" $event]
        set rsrc [lindex [split $src .] 0]
        set rdst [lindex [split $dst .] 0]

          set sid [get_trace_item "-S" $event]

        if {([string compare $protocol "ack"] == 0) && \
            ([string compare $cmd "-"] == 0) && \
              ([string compare $lsrc $rsrc] == 0)} {
            if {![info exists ackmark]} {
                set ackmark $mcnt
                  set fname($mcnt) "ack"
                incr mcnt
            }
            set event "$event -f $sid -m $ackmark"
        }
    }
    return $event

}

#ecn filter

proc ecn_filter { event } {

    global colorset fname fcnt tcpecn tcpcong ackecho ackecn colorindex mcnt \
            tcpecn_m tcpcong_m ackecho_m ackecn_m mcnt

    set extinfo [get_trace_item "-x" $event]

    if {$extinfo!=""} {

        #get tcp flags
        set flagitem [lindex $extinfo 3]
        set tcpflags [split $flagitem "-"]

        set is_ecn [lsearch $tcpflags "E"]
        set is_cong [lsearch $tcpflags "A"]
        set is_echo [lsearch $tcpflags "C"]

        #filter tcp packet

        set protocol [get_trace_item "-p" $event]
```

```
set cmd  [lindex $event 0]

  #1. tcp ecn
  #2. tcp cong
  #3. ack echo
  #4. ack ecn

if {([string compare $protocol "tcp"] == 0) && \
      ($is_ecn != -1)} {

   if { ![info exists tcpecn] } {
      set tcpecn $fcnt
          set tcpecn_m $mcnt
      set fname($mcnt.$fcnt) "tcp_ecn"
      incr fcnt
          incr mcnt
   }

   set fbit [get_trace_item "-f" $event]

     if { $fbit == "" } {
        set event "$event -f $tcpecn -m $tcpecn_m"
     } else {

          set findex [lsearch $event -f]
          incr findex
          set line2 [lreplace $event $findex $findex $tcpecn]
          set event $line2

          set mindex [lsearch $event -m]
          incr mindex
          set line2 [lreplace $event $mindex $mindex $tcpecn_m]
          set event $line2

     }
  }

  if {([string compare $protocol "tcp"] == 0) && \
   ($is_cong != -1)} {

   if { ![info exists tcpcong] } {
      set tcpcong $fcnt
          set tcpcong_m $mcnt
      set fname($mcnt.$fcnt) "tcp_cong"
      incr fcnt
          incr mcnt
   }

   set fbit [get_trace_item "-f" $event]
```

```
    if { $fbit == "" } {
        set event "$event -f $tcpcong"
    } else {

        set findex [lsearch $event -f]
        incr findex
        set line2 [lreplace $event $findex $findex $tcpcong]
        set event $line2

        set mindex [lsearch $event -m]
        incr mindex
        set line2 [lreplace $event $mindex $mindex $tcpcong_m]
        set event $line2

    }
}

  if {([string compare $protocol "ack"] == 0) && \
   ($is_echo != -1)} {

    if { ![info exists ackecho] } {
        set ackecho $fcnt
            set ackecho_m $mcnt
        set fname($mcnt.$fcnt) "ack_echo"
        incr fcnt
            incr mcnt
    }

    set fbit [get_trace_item "-f" $event]

    if { $fbit == "" } {
        set event "$event -f $ackecho"
    } else {

        set findex [lsearch $event -f]
        incr findex
        set line2 [lreplace $event $findex $findex $ackecho]
        set event $line2

        set mindex [lsearch $event -m]
        incr mindex
        set line2 [lreplace $event $mindex $mindex $ackecho_m]
        set event $line2


    }
}

  if {([string compare $protocol "ack"] == 0) && \
   ($is_ecn != -1)} {
```

```
            if { ![info exists ackecn] } {
                set ackecn $fcnt
                    set ackecn_m $mcnt
                set fname($mcnt.$fcnt) "ack_ecn"
                incr fcnt
                    incr mcnt
            }

            set fbit [get_trace_item "-f" $event]

            if { $fbit == "" } {
                set event "$event -f $tcpecn"
            } else {

                set findex [lsearch $event -f]
                incr findex
                set line2 [lreplace $event $findex $findex $ackecn]
                set event $line2

                set mindex [lsearch $event -m]
                incr mindex
                set line2 [lreplace $event $mindex $mindex $ackecn_m]
                set event $line2


            }
        }
    }
    return $event
}

proc nam_afterfilter { tracefile } {

    global colorindex colorname nam_version session_id highest_seq
fcolorname \
            fcnt fname groupmember mcnt proto_id

    set tracefilebackup $tracefile.tmp
    set file [open $tracefile "r"]
    set file2 [open $tracefilebackup "w"]

    #set SRM session info
    for {set i 0} {$i < $colorindex} {incr i} {

        set items [split $session_id($i) "-"]
          set item1 [lindex $items 0]
#         set item1 [lindex $session_id($i) 0 ]
          if { [string compare $item1 "SRM" ] == 0 } {
             set item2 [lindex $items 1]
```

```
            set session_id($i) "SRM session with \
            members: $groupmember($item2)"
            set proto_id($i) "SRM"
            set memberlist($i) $groupmember($item2)

            set cnt 1
            foreach value $groupmember($item2) {
                set groupindex($i.$value) $cnt
                lappend groupm($i) $value
                incr cnt
            }
            set highest_seq($i) $cnt

        }
    }

    #Write nam version info first

    #set realf [expr $fcnt-$colorindex]

    puts $file2 "V -t * -v $nam_version -a 1 -c $colorindex -F $fcnt -M $mcnt"

    # Write color info

    for {set i 0} {$i < $colorindex} {incr i} {
        puts $file2 "c -t * -i $i -n $colorname($i)"
    }
    # write color info for filters

    for {set i $colorindex } {$i < $fcnt} {incr i} {
        set fid [expr $i-$colorindex]
          puts $file2 "c -t * -i $i -n $fcolorname($fid)"
    }

    # Write session info

    for {set i 0} {$i < $colorindex} {incr i} {
#       puts $file2 "N -t * -S $i -n \{$session_id($i)\}"
        if ![ info exists memberlist($i) ] {

            set memberlist($i) ""

        }
        puts $file2 "N -t * -S $i -n \{$session_id($i)\} -p $proto_id($i) \
             -m \{$memberlist($i)\}"

    }

    # Write y mark if necessary
```

```
for {set i 0} {$i < $colorindex} {incr i} {
      if { [info exists groupm($i)] } {
         for {set j 0} { $j < [llength $groupm($i)]} {incr j} {
            puts $file2 "N -t * -S $i -m [lindex $groupm($i) $j]"
         }
      }
}

# Write highest y value for each session

for {set i 0} {$i < $colorindex} {incr i} {
   puts $file2 "N -t * -S $i -h $highest_seq($i)"
}

# Write Filter info
foreach index [array names fname] {
      set fids [split $index "."]
      set fid [lindex $fids 1]
      set mid [lindex $fids 0]

      if { $fid == ""} {
         puts $file2 "N -t * -F 0 -M $mid -n $fname($index)"
      } else {
         puts $file2 "N -t * -F $fid -M $mid -n $fname($index)"
      }
}

while {[eof $file]==0} {

   set line [gets $file]
   set cmd [lindex $line 0]

   #ignore the original color value & version info
   if { [string compare $cmd "c"]==0 } {continue}
     if { [string compare $cmd "V"]==0 } {continue}
   if {$line==""} {continue}

      set timev [get_trace_item "-t" $line]
      if {[string compare $timev "*"]==0 } {
         puts $file2 $line
         continue
      }

      # color match

      set fv [get_trace_item "-f" $line]
      if { $fv > 1 } {

       set attrindex [lsearch $line -a]
       incr attrindex
```

```
  set line2 [lreplace $line $attrindex $attrindex $fv]

  set line $line2
}

# group setting
set extinfo [get_trace_item "-x" $line]
set srmflag [lindex $extinfo 4]
set srmvalues [split $srmflag "_"]
set srmvalue [lindex $srmvalues 0]
if { [string compare $srmvalue "SRM"] == 0 } {

    set sid [get_trace_item "-S" $line]
    set src [lindex $extinfo 0]
    set newseq $groupindex($sid.$src)
    set newindex [lreplace $extinfo 2 2 $newseq]
    set extindex [lsearch $line -x]
    incr extindex
    set line2 [lreplace $line $extindex $extindex $newindex]
    set line $line2
}

#  define y val in nam graph


set sess_id [get_trace_item "-S" $line]

if [info exists proto_id($sess_id)] {

    set extinfo [get_trace_item "-x" $line]
    set yval [lindex $extinfo 2]

    switch -exact -- $proto_id($sess_id) {
       TCP {

            set ymark $yval
       }
       SRM {
            set ymark [lindex $extinfo 0]
       }
       default {
            puts "Unknown protocol event found!"
            set ymark $yval
      }
    }

    set line "$line -y {$yval $ymark}"

}
```

```
        #filter out -x

        set extinfo [get_trace_item "-x" $line]
        if { $extinfo != "" } {
            set xindex [lsearch $line -x]
            incr xindex
            set line2 [lreplace $line $xindex $xindex]
            set line $line2

            set xindex [expr $xindex-1]
            set line2 [lreplace $line $xindex $xindex]
            set line $line2
        }

        puts $file2 $line
    }
    close $file
    close $file2

    exec mv $tracefile.tmp $tracefile
}

#---------------------------------------------
#main starts here

if { $argc < 1 } {
    puts "Usage: namfilter tracefile-name"
    exit
}

set tracefile [lindex $argv 0]

# save filter name
# CUSTOMIZED PLACE START

set filters(0) tcp_filter
set filters(1) ack_filter
set filters(2) ecn_filter
set filters(3) srm_filter

set filter_num 4

# END OF CUSTOMIZED

set fcnt 0
set mcnt 0

global fname
```

```
if [catch { open $tracefile r } file] {
    puts stderr "Cannot open $tracefile: $fileId"
    exit
}

close $file

set filtering_flag [nam_analysis $tracefile]
if { $filtering_flag != 2 } {exit}

# Decide how many flows (TCP only so far)
nam_prefilter $tracefile

set fcnt [expr $fcnt+$colorindex]

# Applying filters
nam_filter $tracefile

#Add control info to the head of the tracefile
nam_afterfilter $tracefile

puts "Filters have been applied to the tracefile: $tracefile"

exit
```

Archivo para el calculo del Throughput

El archivo thoughput.pl esta programado en perl y calculamos el throughput, en intervalos definidos por el usuario, al aplicarlo sobre las trazas de la simulación.

Si queremos calcular el throughput en el nodo 4 de una simulación con intervalos de un segundo y grabar los resultado en un archivo (thp) para su posterior analisis, haremos lo siguiente en la linea de comandos; **perl throughput.pl out.tr 4 1 > thp.**

Este es el código en perl del archivo.

```
# type: perl throughput.pl <archivo trazas> <nodo deseado> <intervalos de tiempo>  >  fichero de salida

$infile=$ARGV[0];
$tonode=$ARGV[1];
$granularity=$ARGV[2];

#Calculamos como algunos bytes fueron transmitidos  durante un intervalo de tiempo especificado en segundos
$sum=0;
```

```
$clock=0;

     open (DATA,"<$infile")
       || die "Can't open $infile $!";

   while (<DATA>) {
         @x = split(' ');

#la columna 1 corresponde con el tiempo
if ($x[1]-$clock <= $granularity)
{
#comprobamos que el evento sea una recepción (r)
if ($x[0] eq 'r')
{
#comprobamos si nodo destino corresponde con argv 1
if ($x[3] eq $tonode)
{
#comprobamos que sea un paquete TCP
if ($x[4] eq 'tcp')
{
   $sum=$sum+$x[5];
}
}
}
}
else
{   $throughput=$sum/$granularity;
    print STDOUT "$x[1] $throughput\n";
    $clock=$clock+$granularity;
    $sum=0;
}
}
  $throughput=$sum/$granularity;
    print STDOUT "$x[1] $throughput\n";
    $clock=$clock+$granularity;
    $sum=0;

   close DATA;
exit(0);


}
```